Improving Asteroid Photometry Using Artificial Neural Networks

By LEO HOSHI

MSc, Advanced Computing Technologies

Project Report

Sep 2022

Supervisors:

Dr. Tingting Han - Birkbeck, University of London

Dr. Juan L. Rizos - University of Maryland

Department of Computer Science and Information Systems

Birkbeck, University of London

Abstract

The enormous amount of data acquired by spacecrafts, space telescopes, and ground-based telescopes is making data processing more complex in astronomy. As a result, in recent years, Data Science and Machine Learning have become useful in the field of astronomy and space explorations. This project will be the continuation of Dr.Rizos's work from the University of Maryland on asteroid photometry, which if solved, can be applied to other airless bodies in the Solar System.

There are multiple classical approaches to performing asteroid photometry on images taken by spacecrafts. This project is aiming to automize processes and use Machine Learning algorithms to achieve faster and more precise results in asteroid photometry. By using Artificial Neural Networks (ANNs) we will try to create a fast modern photometry model with higher accuracy than before. Dr.Rizos and his team have worked on this topic in the past and what I will do in this project is to use new methods and algorithms and try to make the ANN network even better [1].

Contents

Acknowledgments

I would like to thank Dr. Juan Rizos from the University of Maryland, Department of Astronomy for allowing me to continue his work, granting me access to his code and data sets, and helping me with the astronomy part of the project.
I would also like to thank my university supervisor, Dr. Tingting Han, for her support along the way.
And last but not least, I would like to thank the entire NASA OSIRIS-REx team for making this project possible.

1. Introduction

1.1 Background and Motivation

1.1.1 What is Asteroid Photometry?

There are multiple definitions of photometry, but we can say that photometry is the science of the measurement of brightness [2].

Photometry is one of the most powerful investigation techniques in physics, optics, and many other fields as well as astronomy. Astronomers doing research on asteroids and other bodies in the Solar System gain a large amount of data on these objects' shapes, sizes, optical properties, structures, and orbits by using photometry methods. In the past few decades, with multiple spcacecrafts flying by asteroids and imaging them, the science of photometry has proven to be essential in our understanding of their properties.

Asteroid photometry is about the relationship between the brightness of the surface of an object and the viewing geometry. When a spacecraft takes an image of an asteroid's surface with the sun as the light source, there are multiple angles to be calculated such as the angle between the camera and the sun. To better understand surface geometry such as roughness, particle size, and porosity, astronomers perform photometry on asteroid images. Figure 1.1 shows the observing geometry used in this research.

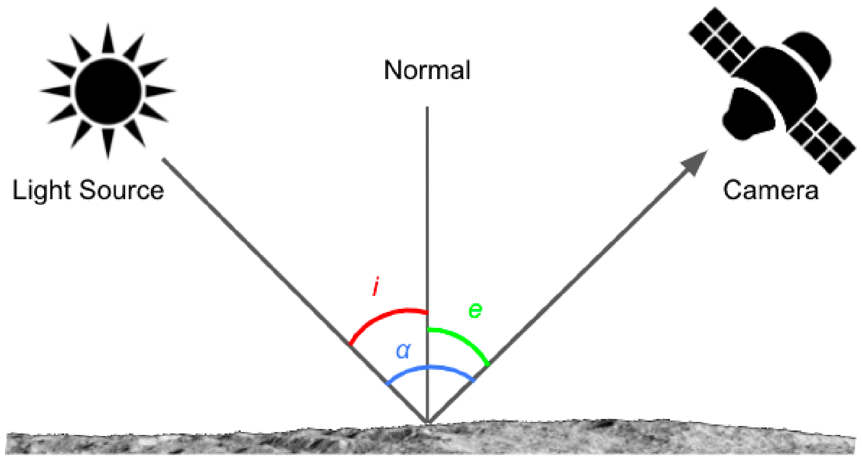

Figure 1.1 Diagram of incidence (i), emission (e), and phase (α) angles in asteroid imaging

The angle between the sun and normal is the incidence angle, i; the angle between the spacecraft camera and normal is the emission angle, e; and the angle between the sun and the spacecraft camera, as seen from the asteroid, is phase angle, α. Physical properties of the surface, such as porosity or particle size, along with the viewing geometry, will affect the brightness variation. For this reason, we need to make photometric corrections.

By using photometric models, we can relate the measured brightness under any geometric values (α_m, e_m, i_m) to reference ones (α_0, e_0, i_0) (Equation 1.1). The most used reference angles in photometric modeling are (α_0, e_0, i_0) = (0°, 0°, 0°) and (30°, 0°, 30°) [26][27].

$$\mathrm{RADF}\ (\alpha_0, e_0, i_0) = \mathrm{RADF}\ (\alpha_m, e_m, i_m)\frac{RADF_{model}(\alpha_0, e_0, i_0)}{RADF_{model}(\alpha_m, e_m, i_m)} \qquad \text{Equation 1.1}$$

The Radiance Factor (RADF) is the ratio of the reflected light from the target surface to that from a reference Lambertian surface. Lambertian reflectance defines the ideal "matte", in other words diffusely reflecting surface. A surface is Lambertian when the brightness of the surface appears the same to the observer from any angle. This surface has a bidirectional reflectance rL(0, e, e) = $\frac{1}{\pi}$ in any direction, with a unit of [$\frac{1}{sr}$]. Therefore, RADF is dimensionless. RADF is equivalent to I/F, another common notation for this magnitude in related literature [3][4].

To have a better idea of asteroid imaging, a schematic of the OSIRIS-REx Equatorial Stations observation is shown in Figure 1.2 below. The spacecraft performs hyperbolic flybys at seven local solar times, each for a full Bennu rotation, ensuring near-global coverage of Bennu's surface. The phase angle (α) and a representative MapCam image (taken with the v-filter) are shown for each station [5].

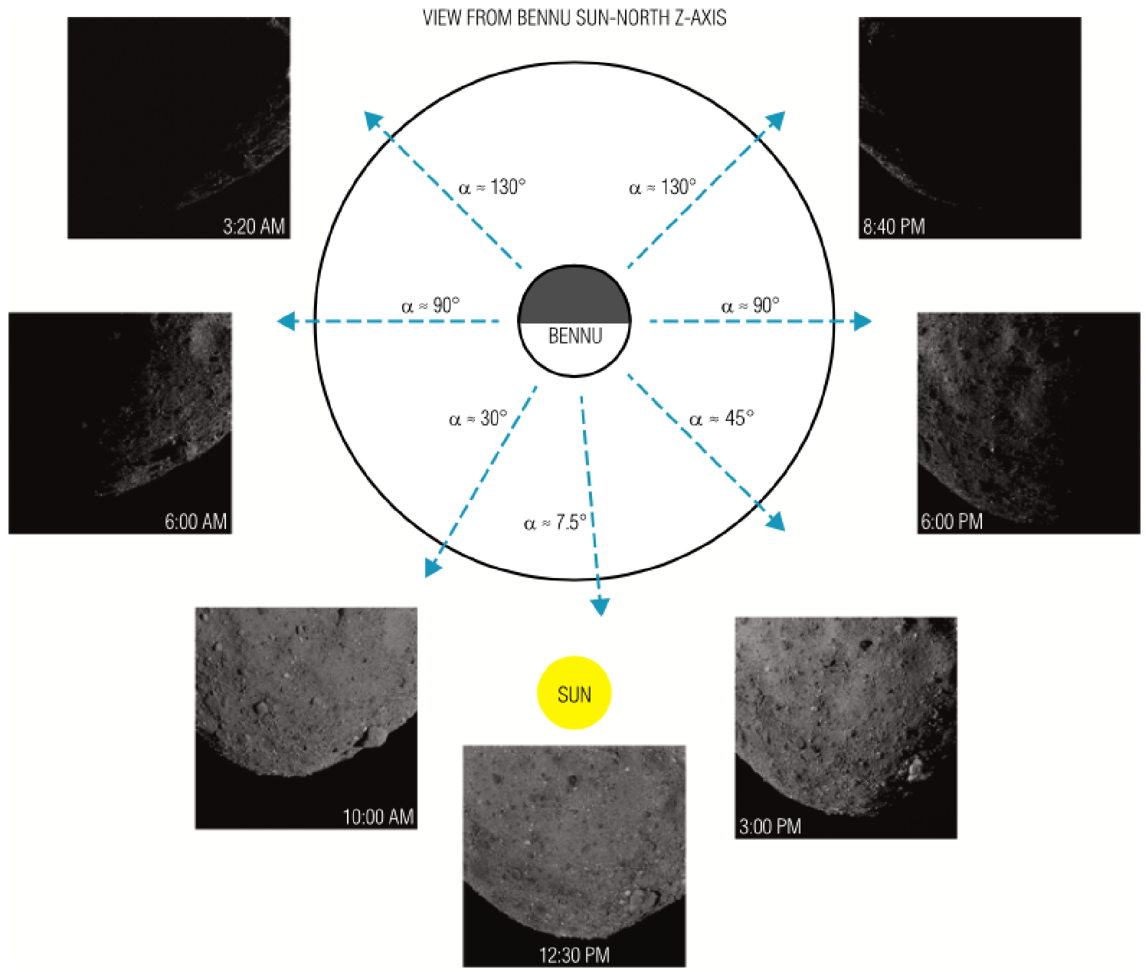

Figure 1.2 Schematic of the OSIRIS-REx equatorial stations observation [5]

1.1.2 Importance of Asteroid Photometry

As Li et al. (2015) explain in their paper, there are multiple reasons why asteroid photometry is important in astronomy. In this work, we focus on the two most important reasons.

First, we can understand the physical properties and compositions of asteroid surfaces. The surface properties such as roughness, porosity, and particle size determine how the light is reflected from the surface of an asteroid. By measuring this reflected light from different positions with different illumination angles, we can understand the surface properties.

Second, the brightness of the reflected light varies depending on the geometry. Therefore, we need to correct or standardize it to a reference geometry to compare regions with different viewing geometry [3].

1.1.3 Motivation for This Work

As mentioned above, asteroid photometry is important in astronomy to deepen our knowledge of the Solar System we live in. In the past decade, machine learning has proven to be very useful in the field of astronomy. However, so far there have not been many scientific reports to use machine learning in asteroid photometry. One notable paper about this matter is by Rizos et al. (2021). In this paper, Dr.Rizos and his team developed a new method for asteroid photometry using Artificial Neural Networks. Dr.Rizos from the University of Maryland, and an official member of the Image Processing Working Group of the OSIRIS-REx mission agreed that we work together on this project. The objective of this project is to improve their work and achieve better results.

1.2 Literature Review

1.2.1 Mathematical Photometric Models

So far, there have been mathematical and empirical photometric models that relate reflectance with phase, incidence, and emission angles. In this study, we call these models “classical models”.

Mathematical photometric models such as Hapke models (Hapke et al. 2012) are based on radiative transfer theory. The radiative transfer theory, also called transport theory, is the theory explaining how electromagnetic waves travel through a medium object and its characteristics. This theory was developed to describe radiation absorption [6].

Hapke models are the most popular photometric models that are used widely. The Hapke parameters used in these models are parameters that describe the directional reflectance properties of the airless regolith surfaces of objects in the Solar System. Regolith is the soft layer on top of the bedrock of a celestial object made up of dust and unconsolidated rock. The model has been developed by astronomer Bruce Hapke at the University of Pittsburgh. Because of the complexity of the model and its parameters, the details will not be included in this project report [7].

However, there are multiple studies where correlations between Hapke parameters — which are supposed to be independent — have been found (Shkuratov et al. 2012) [8].

Shkuratov et al. (2012) explain in this paper that the Hapke model has shortcomings that weaken its applications. One shortcoming is that some of the model parameters are empirical and mutually dependent. Furthermore, the photometric model is diverse and approximate. One example is that the model simultaneously considers the single-scattering phase function as backscattering and isotropic when describing, respectively, incoherent and coherent multiple scattering, which is physically impossible.

They show multiple examples in this work showing how the Hapke model is not as complete as it should be. In this report, we do not explain these examples due to their irrelevancy to computer science but just keep in mind that a new method would be very useful.

1.2.2 Empirical Photometric Models

Empirical models such as Lambert or Minnaert, that have been developed ad hoc (Li et al. 2015) are also used in some studies. Empirical models are used in (Zou et al. 2020) for photometric modeling of images taken by OSIRIS-REx Camera Suite (OCAMS). In the research paper of Zou et al. (2020), they first select the spectra with the highest quality to be used in the photometric model. In order to prepare the data, they perform the following steps:

(1) Selecting only filled spots, because the unfilled spots are not calibrated
(2) Removing spots with incidence or emission angles greater than 70◦ due to poor data quality and restrictions in the disk-function modeling ability
(3) Excluding all spectra with noise spikes or negative values due to saturation at the longest wavelength
(4) Excluding all spectra with jumps greater than 10% at filter segment boundaries

Later in this study, we will show that using a new method using machine learning algorithms have better results in comparison to classical models [3][9].

1.2.3 ANN Model

A search on google scholar about using machine learning in asteroid photometry had 1900 results but when looking at the topics and contents, I could not find any paper conducting research on asteroid photometry using machine learning.
There are many types of neural networks that could be used to build models for this project and one might think that we are dealing with images, so why not use Convolutional Neural Networks instead of Artificial Neural Networks? The answer is that our input and output data is not of image type, but values of pixels of those images. Therefore, our input and output are numeric values which makes more sense to use ANNs.
Rizos et al. (2021) used artificial neural networks to improve the photometric modeling of airless bodies, a genius idea that can revolutionize the way astronomers perform airless body photometry.
In their work, they used asteroid Bennu's image data taken by the MapCam camera installed on NASA's OSIRIS-REx spacecraft which was launched in 2016.
MapCam is an imager with four narrowband filters based on the Eight-Color Asteroid Survey (ECAS, Tedesco et al., 1982): b′, v, w, and x, with wavelengths at 473, 550, 698, and 847 nm, respectively. We have access to thousands of MapCam images from different viewing geometries of Bennu for all color filters [10][22].

In their work, they used PyTorch to build the ANN model and achieved 14.3% lower MSE than classical methods for the x-filter image data.

1.2.4 The Improved ANN Model

This project is an improvement of (Rizos et al. 2021) work, trying to achieve even lower MSE than classical models. Multiple improvements were made, and results are available in section 7.
First of all, I noticed that the code is not reproducible. Therefore, I added "seed" lines to the code so that no matter how many times we run the code, we get exact same results. In Torch we set seed using torch.manual_seed(**).
Making use of Feature Engineering is another method I used to improve the accuracy of the model. Autoencoder is used to generate new features and Principal Components Analysis (PCA) is used to select the 3 most informative features to be used in the ANN.
Furthermore, I used Keras to build models, and surprisingly compared to PyTorch models they had better performance and faster speed.
Finally, there are multiple hyperparameters that could be tuned in each model, and I did try many combinations to find the best set of hyperparameter values.
These are all explained in detail in later sections of this report.

2. Problem Presentation and Objectives

Although the classical methods described in the previous section work well in most cases, there are limitations to their use and there is a need to develop new approaches. The objective of this work is to find and develop a new way that could either replace or complement the classic approach to photometric modeling.

The number of robotic space explorations is growing rapidly and there is a need for better photometric modeling methods more than ever before. By developing new modern photometric modeling tools, data from missions to airless bodies such as OSIRIS-REx, Hayabusa2, and Martian Moons eXploration – MMX could be processed faster and with higher accuracy.

Using Artificial Neural Networks (ANNs) is an efficient way to develop automatic photometric modeling. ANNs have a powerful application to analogue problems in astronomy.

This project aims to develop even better models with lower MSE using other ANN methods and by adjusting hyperparameters in the previous studies.

We will reproduce classical modeling results to compare with the ANN results at the end of the project for conclusions.

3. Data Collection and Pre-processing

3.1 Data Collection and Processing

The data used in this research is from MapCam images of NASA's OSIRIS-REx mission available on the Planetary Data System (PDS) [11]. The necessary data was collected and processed by Dr.Rizos and shared with me for this project.
The image processing procedure performed by Dr.Rizos to prepare data for this project is as follows:

(1) Convert PDS images to cubes in order to be manipulated with a customized Integrated Software for Imagers and Spectrometers (ISIS) version developed by the OSIRIS-REx team [28].

(2) Using the ISIS, retrieve phase, incidence, and emission angles generating backplanes via SPICE kernels [29]. For getting these backplanes, version 28 of the Bennu shape model is used. Developed by the Altimetry Working Group of the OSIRIS-REx Team, this shape has a resolution of up to 80 cm/pixel.

(3) Some images have better resolution than the shape model. To avoid degeneracy, i.e., several pixels with different brightness inside the same facet (having the same phase, emission, and incidence angles), the image resolutions were reduced to 80 cm/pixel. Therefore, this is the limit of our photometric modeling. In other words, the limit of our work is that we are not seeing anything smaller than 80cm.

(4) Create a fits file for each image consisting of 7 layers of data.
1st layer is the RADF value
2nd layer is the phase angle
3rd layer is the emission angle
4th layer is the incidence angle
5th layer is the latitude
6th layer is the longitude
7th layer is the pixel resolution
Note that in this project we only use the 1st~4th layer data.

MapCam data are stored in fits format, widely used in the field of astronomy. In each file, we find a float 1024x1024 matrix representing the image itself, along with a header, containing information about the observing time, the color filter used, or the units after calibration. In addition, we need to get, for each pixel, the incident, phase, and emission angles. They were generated by Dr. Rizos using SPICE kernels of the mission and shared with me for this work [12].

About 800~900 images will be used in each color filter. To shorten computing time, we will use a specific number of pixels from each image, which are randomly selected from the data set. Therefore, the dataset is already shuffled and we do not need to shuffle it before the train test split. The dataset will be split into training and validation sets to avoid overfitting.
If the image is larger than 1000 pixels, we choose 1000 pixels from it randomly; if it is smaller than 1000 pixels, we use all the pixels of that image.
The four data sets are as follows:

B-filter data
There are 951 image files in the b-filter data set creating 463,505 records of data.
V-filter data
There are 878 image files in the b-filter data set creating 391,299 records of data.
W-filter data
There are 873 image files in the b-filter data set creating 385,638 records of data.
X-filter data
There are 868 image files in the b-filter data set creating 384,536 records of data.

To create, test, and optimize PyTorch and Keras models b-filter data was used. Later, I chose the best optimizations and hyperparameter values and ran the models on the other 3 data sets to calculate their performance and compare them with classical method results.

3.2 Data Cleaning

After loading the fits image data and getting necessary parameters such as emission, incidence, and phase angles we perform data cleaning. Neural networks can have numeric data as their input and our data must be checked to be numeric. As our inputs are geometrical angles, they are numeric data. We also have to check for missing values as having many missing values will lead to low performance of the network.
In this study, we limit the input and output to specific values only. For emission and incident angles, we only choose values between 0 and 82 degrees and for phase angles between 0 and 90 degrees. For reflectance, we choose only values between 0.001 and 1.
The reason for choosing these specific limits is as follows:

- A negative angle below 0 is by definition impossible. All angles are positive according to our definition.
- Limiting emission and incidence angles at 82º is because, for higher values, photometric models do not perform correctly. By referring to page 2 of Rizos et al. (2021) we can see the following sentence: “Photometric models have limited accuracy at an incidence or emission angles >80º ” [27].
- The same reason is applied for limiting the phase angle to a maximum of 90º.
- RADF can not be larger than 1 as of its definition which is a ratio of solar incidence. And below 0.001 are shadows with a very low signal-to-noise ratio. This was checked manually by Dr.Rizos.

3.3 Data Scaling

As neural networks sum inputs and weights values between neurons, it is necessary to scale the data. If the input values or weights are very large, the network will not converge and have very poor performance. To have a network with good performance it is important to rescale the data beforehand. In this step, we rescale the data between 0 and 1. In python, there are functions such as Minmaxscaler() for this purpose but here we rescale them manually by diving them by the limits set above in section 3.2. For example, the incidence angle value will be divided by 82 to have a value between 0 and 1.

4. Artificial Neural Networks

4.1 Network Structure

An artificial neural network (ANN), is a computing system based on the biological neural networks that constitute the human brain [13]. ANNs are composed of neurons arranged in layers linked to the previous and next layers of neurons. The first layer is the input layer, the last one is the output layer and the layers in between are called hidden layers. Each hidden layer accepts the data from the previous layer, processes it, and passes it to the next layer. In each neuron, the inputs (x_1, x_2, … ,x_m) from the previous layer's neurons are converted into an output by means of weights (w_1, w_2, … ,w_m), a bias (b), and a non-linear function called the activation function (f) as in Equation 4.1.

$$output = f\left(\sum_{j=0}^{m} w_j x_j + b\right) \qquad \text{Equation 4.1}$$

Weights and biases are randomly created with a normal distribution and specified standard deviation. To avoid converging difficulties in deep neural networks we will use the Kaiming method (Kaiming et al. 2015) which has proven to be working well in practice. Kaiming is an initialization method for ANNs that uses non-linear activation functions such as ReLU to avoid reducing or magnifying input signals exponentially [1][14].

A simple ANN with one hidden layer is shown in Figure 4.1.

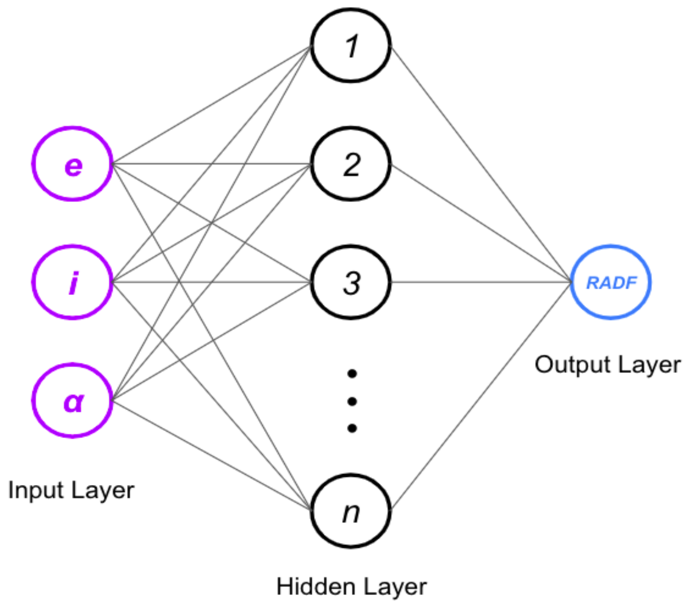

Figure 4.1 A simple one hidden layer ANN showing the flow of data from the input to the output layer

4.2 Supervised Learning

Supervised learning method is when both the predictor X and the response Y are known such as Logistic regression or SVM. Unsupervised learning method is when we only have X such as clustering models.

In our network, emission, incidence, and phase angles are the 3 inputs, and reflectance or RADF is the output. Here, we are trying to predict the reflectance by using these 3 input values. In other words, there are X1, X2, X3 inputs, and Y output.

This is a supervised learning method because the output is known. The network knows the real output and tries to predict an output as close to the real output value as possible by going back and forth between hidden layers. This process will be explained in more detail in the backpropagation section.

4.3 Training and Validation Data

It is very common in machine learning to split the data into two parts of training and validation sets. The network will train itself using the training data set and then test its accuracy on the validation data set. This way it will be used for unseen data and later if we give it new data we can expect good performance. If we train the model on the whole data set, the network most probably will not perform well on unseen data.

There are multiple methods in python to split the data set into train and validation data such as train_test_split, K-Fold validation, and LeaveOneOut.

In this project, we split the data set randomly to have 90% training and 10% validation data.

4.4 Overfitting and Underfitting

While training a neural network, sometimes the network achieves very high accuracy for the training data set and does not perform well on the validation or test data sets. In this case, we say that the network is overfitting and this is a big problem. There are multiple ways that we can fix the overfitting problem such as adjusting hyperparameters of our network, number of epochs, and batch size.

If the number of epochs is too small or the batch size is too large, the network does not have sufficient time to perform well and learn the relationship between input and output, leading to underfitting problem. On the other hand, if the number of epochs is too large or the batch size is too small, the network will learn the relationship between input and output of the training data set too well, and when it is faced with unseen data it can not perform well, leading to the overfitting problem.

In our models, for the PyTorch model, the best number of epochs is 200, and for the Keras model 20.

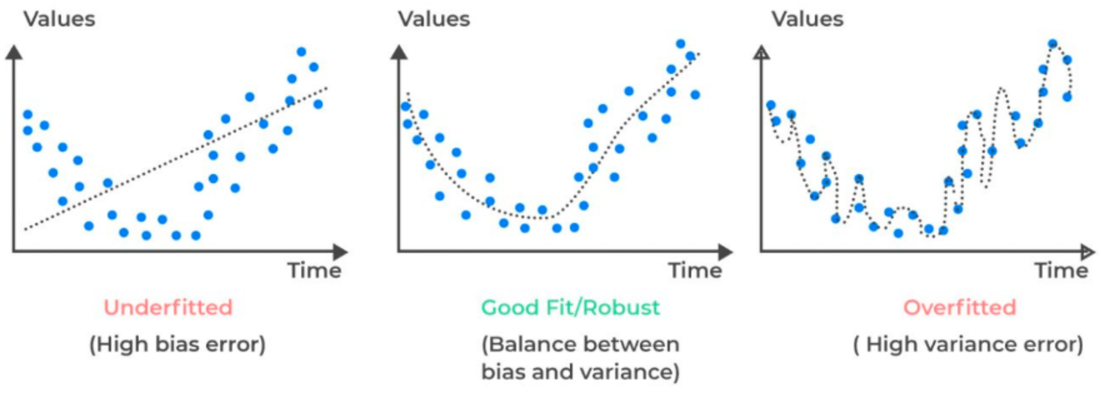

Figure 4.2 Overfitting and Underfitting example [23]

As we can see in Figure 4.2 above, when a model is underfitted, the bias is high and many data points are very far from the model line. When a model is overfitted, the model line is crossing almost all data points and we can say that the variance is high. In this case, if we add a new data point (unseen data) it will most probably not be on the model line. The tradeoff between bias and variance is an important topic in machine learning, as shown in Figure 4.3 below. We can see that the optimum model complexity where the overfitting and underfitting are not happening is in the middle of the graph where both bias and variance are low.

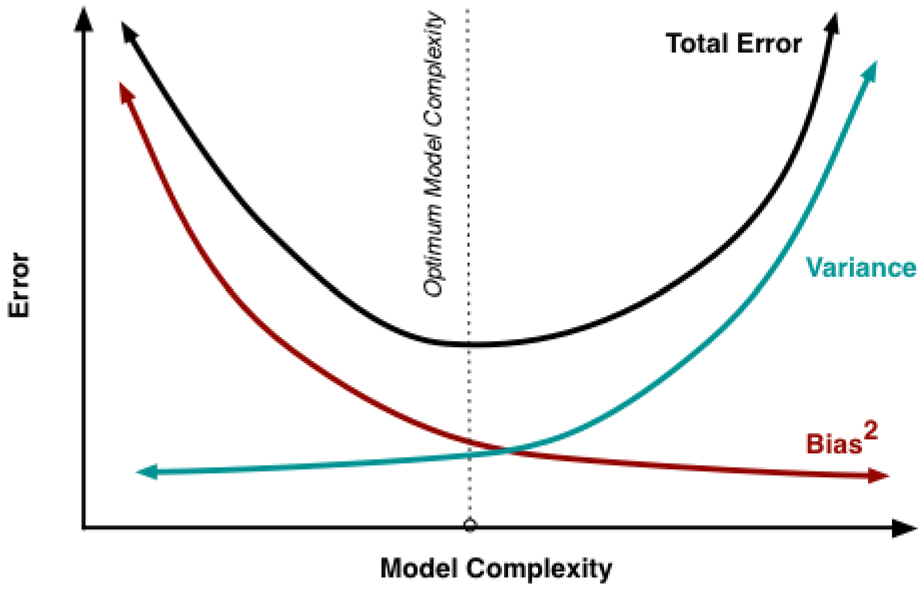

Figure 4.3 Bias Variance tradeoff [24]

4.5 Activation Functions

Activation functions are functions that are used between layers receiving inputs, weights, and bias from the previous layer and converting them into a new form, and passing them to the next layer. This way, depending on the activation function used, some neurons will fire and some will not.

One of the most popular activation functions in neural networks is the Sigmoid function which converts any input into an output between 0 and 1. In our network, we use the Sigmoid function for the output layer. An image of a sigmoid function is shown in Figure 4.4 below [15].

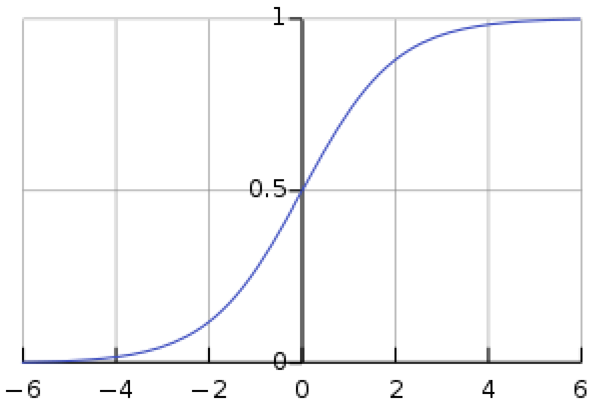

Figure 4.4 Sigmoid Function [15]

Another popular activation function is the Rectified Linear Unit function (ReLU). It is mostly used for hidden layers in neural networks and in our network, we use ReLU for the hidden layers. If we use sigmoid for hidden layers, the values will be kept between 0 and 1, and multiplying them will make the values smaller and smaller converging to zero. What ReLU does is turn off the negative values setting them to zero and keep the positive values the same as they are. It can be thought to be a linear function for positive values but as a whole, it is a non-linear function. A simple ReLU function is shown in Figure 4.5 [16].

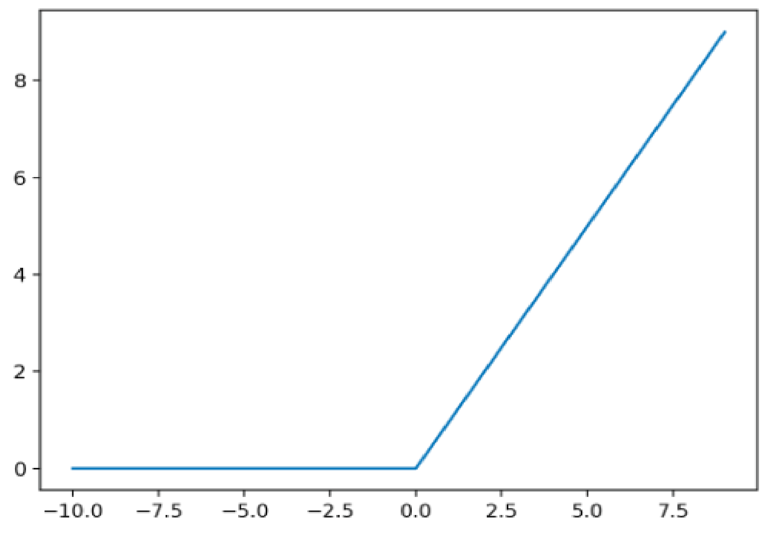

Figure 4.5 ReLU Function [16]

4.6 Backpropagation

In neural networks, each layer receives input from the previous layer and based on weights, biases, and activation function of the layer, it calculates a value that will be passed to the next layer. In other words, information pass from the left (input side) to the right (output side). In supervised learning, when the output is calculated, it will be compared with the real output data we have in our data set and the difference between these two will be calculated as an error. The smaller this error, the closer our predicted output will be to the expected output. In backpropagation, what the network does is send back this error backward through the network from the output layer to the input layer so that the weights can be adjusted to minimize the error after each epoch. As this process is repeated in each epoch, having too many epochs will minimize the error but as explained in section 4.4, it will overfit the training data and does not work well on unseen data. Choosing the right number of epochs is important in getting the best results from our network.

4.7 Learning Rate

The learning rate in a neural network is a hyperparameter that determines how much to update the weights after each epoch in order to minimize the error explained in section 4.6. If the learning rate is too large, the network might not achieve the optimum result and becomes unstable. On the other hand, if the learning rate is too small, the training can take a very long time.

I used learning rates of 1e-2, 1e-3, 1e-4, 1e-5 in the PyTorch network. The learning rate of 1e-5 gave optimum results and the calculated MSE kept on decreasing after each epoch.

Larger learning rates resulted in an unstable network where MSE was increasing and decreasing after each epoch. Therefore, the learning rate for all experiments is set to 1e-5.

4.8 Gradient Descent and Cost Function

Let us consider a terrain with a mountain and a sea. Now imagine that we are standing on the top of the mountain and without seeing anything want to move step by step down to the sea. We take the first step downwards toward the steepest direction and continue this step by step until we arrive at the lowest part of the terrain, which is the sea. In neural networks, in order to achieve the lowest error possible, the algorithm moves towards the steepest descent by changing weights at each epoch. This is called gradient descent. It is an optimization algorithm to find the local minimum cost function. The cost function tells us how well our model is predicting the output. The slope of the cost function's curve tells us how to update the weights to have more accurate predictions.

One big problem with gradient descent is that when there are one or more local minimums in the curve before the global minimum - that we are trying to reach - the algorithm might get stuck in those local minimums and think that this is the best it can do and never reaches the real minimum error.

4.9 Optimization Algorithms

There are already multiple gradient descent optimization algorithms used in neural networks that work well and do not fall for the local minimum explained above. In this research, we use Adaptive Moment Estimation (Adam) as an optimizer. Adam is simple and efficient, does not need a lot of memory, and can work well with big data, making it the most commonly used optimization algorithm in deep learning [17].

4.10 Hyperparameter Adjustment

When it comes to deep learning, some people might think that the more hidden layers they have or the more neurons each layer has is better. We have to consider that with a lot of hidden layers and neurons the computing cost and time increase remarkably and network training takes longer. In this work, we created neural networks with 3,4, and 5 hidden layers and represented the results. The number of neurons in each layer was set from 5 to 40 neurons. I was curious to see if having the same number of neurons in each layer works better or having different ones, therefore, I set the number of neurons to different numbers which can be seen in the results section.

5. Feature Engineering

5.1 Feature Generation

Feature generation, also known as feature construction or feature extraction is a method in machine learning to generate new features from the original features of data. The main goal of feature generation is accuracy improvement. In this case, the resulting feature space will most likely contain more features than the original feature space. This can be achieved by generating new features and combining them with the original features.

There are many methods of feature generation such as Partial Least Squares (PLS), Linear Discriminant Analysis (LDA), Principal Component Analysis (PCA), and Autoencoders.

In this work, we use an autoencoder to generate new features. Later on, we combine the new features with the original ones.

There are 3 features in our data set that are incidence, emission, and phase angles.

Using Autoencoder, we generated 3,6, and 9 new features.

An Autoencoder is a type of unsupervised neural network that uses backpropagation to generate an output value that is as close to the input value as possible. An autoencoder takes the input image or vector and learns a code dictionary that changes the raw input from one representation to another. There are three main blocks in an autoencoder: encoder, decoder, and latent space as shown in Figure 5.1. When we feed the autoencoder with data, the encoder extracts useful features of the input data and keeps them in the latent space. The decoder does the same process but in the opposite order, which is reconstructing the input data using the latent space.

Therefore, when we use autoencoder as a feature generation method, we set the number of new features we want as the latent space size, and the most useful feature information from the input will be extracted and set there [18].

An autoencoder class with one hidden layer containing 64 neurons using the ReLU activation function is shown below. The number of latent dimensions that is the desired new features is set to 9 and the number of inputs is 3. The output layer's activation function is sigmoid.

```
class Autoencoder(Model):
  def __init__(self,latent_dim=9):
    super(Autoencoder,self).__init__()
    self.latent_dim=9
    self.encoder=tf.keras.Sequential([
      layers.Dense(64,activation="relu"),
      layers.Dense(latent_dim,activation="relu")])
    self.decoder=tf.keras.Sequential([
      layers.Dense(64,activation="relu"),
      layers.Dense(3,activation="sigmoid")])

  def call(self,x):
    encoded=self.encoder(x)
    decoded=self.decoder(encoded)
    return decoded
```

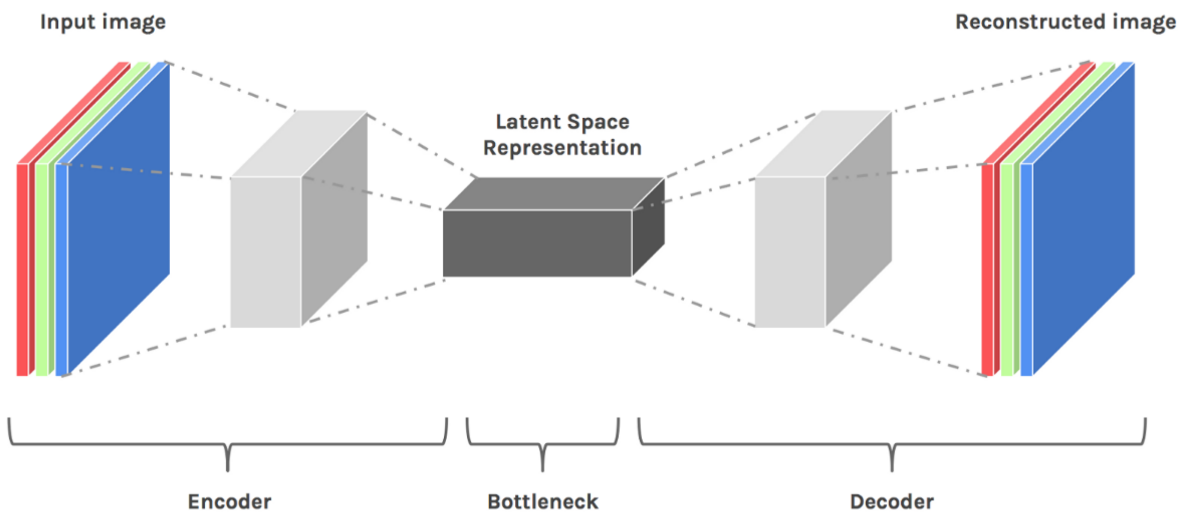

Figure 5.1 Autoencoder building blocks [18]

5.2 Feature Selection

Feature selection is the process of selecting a subset of relevant features for use in model construction. It selects and excludes given features without changing them. It helps the model to become less overfit and while it reduces dimensionality, the complexity of the model decreases. Popular feature selection methods include: Pairwise correlation, Principal Component Analysis (PCA), Cluster analysis, Forward selection, Backward elimination, and Decision Tree.

In this work, we use PCA as the feature selection method because of its simplicity and popularity.

PCA is a dimensionality-reduction method that is used to transform a large number of variables into a smaller set that still contains most of the information. When we reduce the number of variables or features, usually the accuracy will decrease but the simplicity of the model improves. As we saw in Figure 5.1, complex models tend to have overfitting problems. Therefore, we trade off a little bit of accuracy for simplicity. In addition, a smaller number of features helps us to explore and visualize the data easier and machine learning algorithms work better with a smaller number of features [25].

As explained above, we have 3 original features, and using an autoencoder we generate 3,6, or 9 new features. After this step, we combine the new features with the original ones.

One problem that we have to deal with at this step is the correlation between features. Feeding a network of highly correlated features will decrease its accuracy. Therefore, we check if any of the features are correlated. We can set the correlation percentage manually. In this project we set it to 90%, meaning any two features that are 90% or more correlated with each other will be removed from the feature list. In the next step, we select the 3 most informative features from the remaining features using PCA.

Because our model originally has 3 inputs, we set the PCA number of components - that is the number of features to be selected - to 3.

When deciding how many features are useful, one interesting method is plotting the “Explained Variance per Principal Components” graph.

Let us see how this method works. Let us generate 9 new features with the autoencoder and combine them with the original 3 features. We will have 12 features of which 2 were correlated with other ones and were removed from our feature list. When we plot the explained variance as shown in Figure 5.2, we can see that after principal component 2 (which is the 3rd feature) the variance does not change much. This means that by using only 3 features out of 10 we can access the most information in our data set and do not lose much information.

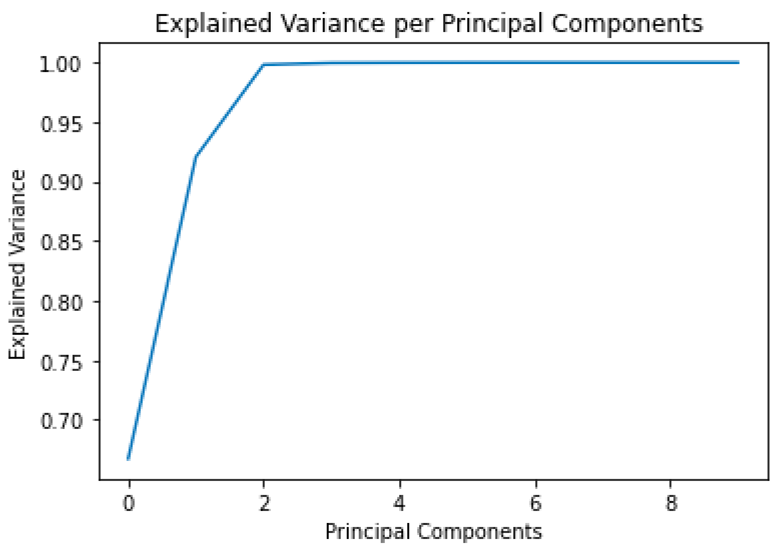

Figure 5.2 Explained variance per principal components graph

6. Model Design and Implementation

6.1 Model Design and Structure

There are multiple packages to create artificial neural networks in python. In this study, we use PyTorch and Keras and compare the results.

For each method, we create two systems, one with only the original features and one using feature engineering. Therefore, we will have 4 models in total.

A simple diagram of the model without feature engineering is shown in Figure 6.1 below. This diagram shows the steps taken to build a complete machine learning model.

At first, the data is retrieved and passed to the data preprocessing part. In this part, the data is prepared, cleaned, and rescaled to be used in a machine learning algorithm.

Then, the data is fed into the PyTorch or Keras networks and the MSE is calculated. In another part of the code, Another MSE is calculated using the classical method. These two results are compared to see if the ANN is more accurate than the classical approach.

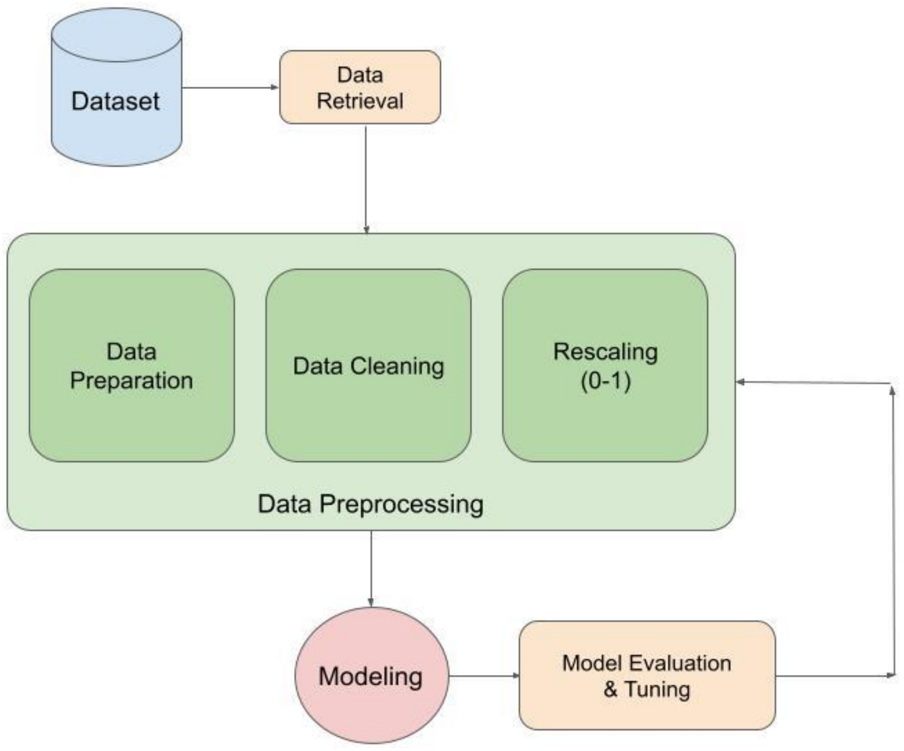

Figure 6.1 Artificial Neural Network model (both PyTorch and Keras) diagram without feature engineering

Figure 6.2 shows the model diagram with feature engineering. The steps are the same as explained above, but after preprocessing, feature engineering is performed on the data. 3,6 or 9 new features will be generated with an autoencoder, then combined with the original features. In the next step, we check feature correlations and remove the correlated ones from the feature list, and at the end use the PCA feature selection method to choose the 3 most informative features and pass them to the network as inputs.

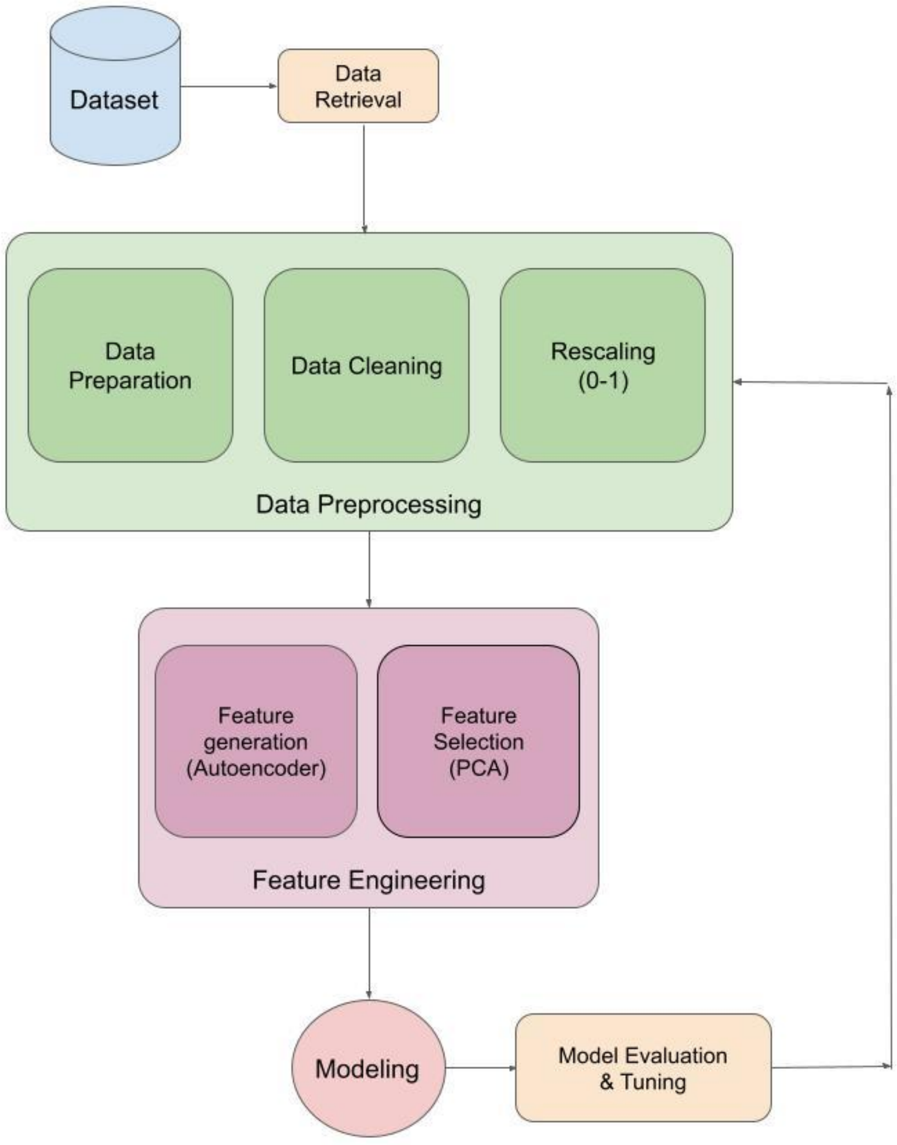

Figure 6.2 Artificial Neural Network model (both PyTorch and Keras) diagram with feature engineering

After model evaluation, we change the hyperparameters of the network and run the model again until we achieve the best result.

6.2 PyTorch

PyTorch is an open-source machine learning framework that accelerates the path from research prototyping to production deployment. It is based on the Python programming language and the Torch library is popular for deep learning research. PyTorch is mainly used for applications of research, data science, and artificial intelligence (AI).

PyTorch is similar to NumPy in terms of using tensors for computations. Tensors are a type of data structure similar to arrays that we can do more than hundreds of mathematical operations on them.

Some benefits of PyTorch are:

- It is based on Python, making it easy to learn and code.
- Debugging is easy with python tools.
- Major platforms support it.
- The interface is user-friendly

In comparison to Tensorflow, PyTorch is easier to work with, is lighter, and is a better option for creating prototypes quickly and conducting research.

On the contrary, Tensorflow is a better option for projects that require production models and scalability, as it is ready for production [19][20].

6.3 Keras

Keras is one of the most common methods to create a neural network that works with TensorFlow. Its structure is sequential, meaning we are able to add new hidden layers and specify the number of neurons and activation functions easily.

Keras's APIs are simple, the number of user actions for use cases is less than other methods, and it has clear and actionable error messages. It can be deployed on iOS, Android, Javascript, and web API. It is used by NASA, CERN, and many other scientific institutes [21].

After using both PyTorch and Keras, I would say Keras is simpler and easier to implement and surprisingly gave better results in this project.

The Keras ANN framework code looks something like the code below.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()
model.add(Dense(**, input_shape=(input_size,), activation='**'))
model.add(Dense(**, activation='**'))
model.add(Dense(**, activation='**'))
model.compile(loss='**', optimizer='**')
modelfit = model.fit(X_train, Y_train, validation_data=(X_val, Y_val), epochs=**, batch_size=**)
```

6.4 Classical Method Reproduction

To compare our ANN results with the classical methods, we add a code block in our program that calculates MSE using classical methods. Here we pass the three inputs, radiance, emission, and phase angles to the function and after converting degrees to radians, calculate phase and disk functions. Multiplying these two functions will give us the predicted reflectance which is the output and compare it with the expected output we have in our dataset will give us the error in calculations or MSE.

6.5 Improving Runtime Using GPU

When running python codes, depending on the computer's GPU model, we are able to use the GPU instead of the CPU. Unfortunately, my computer's GPU is intel and does not support this feature but here is some useful information about using GPU rather than CPU in machine learning.

Python is one of the most popular programming languages for many purposes such as engineering, data science, and deep learning. However, it is considered to be too slow for high-performance computing. Running python on GPU improves the speed and performance remarkably.

Two main advantages of GPUs include the following:

1- High Data Throughput: GPUs consist of many cores performing the same operation on multiple data in parallel. Therefore, a GPU can push large volumes of processed data through a workload, speeding up tasks, which a CPU can not handle.

2- Massive Parallel Computing: CPUs work better for more complex computations. On the other hand, GPUs can handle extensive calculations with multiple similar operations, such as matrix calculations.

Therefore, for machine learning models, it is a good idea to run the code on GPU [30].

7. Test and Evaluation

7.1 Loss Function

The predicted output value will be compared to the real value from the dataset using a loss function. One of the most common ways to check the accuracy of a model is using the mean square error (MSE) (Equation 7.1), which is widely used in machine learning.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(real\ value_i - predicted\ value_i)^2 \qquad \text{Equation 7.1}$$

The improvement factor is the comparison of MSE in both methods as shown in Equation 7.2.
ROLO here means the classical model.

$$Improvement\ (\%) = \frac{MSE_{ROLO} - MSE_{ANN}}{MSE_{ROLO}} \cdot 100 \qquad \text{Equation 7.2}$$

7.2 PyTorch Model Results

PyTorch model results for 3, 4 and 5 hidden layer network are shown in Tables 7.1~3 respectively.
Network specifications are as follows:
Dataset used: b-filter
Number of epochs: 200
Batch size: 1000
Learning rate: 1e-5
Optimizer: Adam

The left column in the tables with "Torch" title is the model without feature engineering and the 3 columns on the right are results with feature engineering. (Torch-AE-3 indicating autoencoder generated 3 new features).
The autoencoder has 2 hidden layers, each with 64 neurons. The optimizer is "adam" and the loss function is "MSE".

Number of Neurons	Torch		Torch-AE-3		Torch-AE-6		Torch-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)
5,5,5	0.004426	-0.58	0.00449	-2	0.004584	-4.17	0.004513	-2.34
10,10,10	0.004621	-5	0.003915	11.04	0.003983	9.5	0.004035	8.3
15,15,15	0.004069	7.5	0.003982	9.5	0.004009	8.9	0.003906	11.23
10,15,10	0.004289	2.5	0.003914	11.05	0.003916	11.02	0.003837	12.81
20,10,20	0.004064	7.6	0.003919	10.93	0.004054	7.87	0.003887	11.67
30,30,30	**0.003882**	**11.9**	**0.003821**	**13.18**	**0.003888**	**11.66**	**0.003798**	**13.7**
15,35,15	0.003969	9.8	0.003892	11.56	0.00391	11.14	0.00385	12.57
40,20,40	0.003952	10.19	0.003844	12.65	0.003933	10.63	0.0038	13.65
40,40,40	0.003945	10.35	0.003833	12.91	0.004003	7.2	0.003831	12.95

Table 7.1 3 hidden layer PyTorch model results. The bold color in each column shows the best result of that model and the blue color shows the best result in all models.

In Table 7.1 we can see that using feature engineering and generating 3 and 9 new features gives 1.8% lower MSE than the simple Torch model. Furthermore, we can see that in all models, having 30 neurons in each layer gives the best result. When having a small number of neurons like 5 or 10, we can see that the model performance is very weak, showing that the model was not able to learn from the training data. High number of neurons did not improve the results either as networks with 40 neurons had poor performance too.

Another observation is that when the number of neurons is like a mountain (10,15,10) or valley (20,10,20) the performance is not as good as flat numbers (30,30,30).

Number of Neurons	Torch		Torch-AE-3		Torch-AE-6		Torch-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)
15,15,15,15	0.003978	9.59	**0.003855**	**12.39**	0.00395	10.23	**0.003801**	**13.61**
10,20,20,10	0.003964	9.92	0.003877	11.9	**0.003917**	**10.99**	0.003822	13.14
20,10,10,20	0.003931	10.66	0.00386	12.27	0.004004	9.02	0.003907	11.21
30,30,30,30	0.003924	10.83	0.003963	9.94	0.004068	7.56	0.003884	11.75
40,30,20,10	0.00393	10.69	0.003857	12.35	0.003977	9.63	0.003871	12.03
10,20,30,40	0.003947	10.3	0.003878	11.87	0.003964	9.92	0.003855	12.39
40,40,40,40	**0.003843**	**12.66**	0.003911	11.11	0.004057	7.8	0.003887	11.67

Table 7.2 4 hidden layer PyTorch model results. The bold color in each column shows the best result of that model and the blue color shows the best result in all models.

In Table 7.2 we can see that having 4 hidden layers change the results in a way that 30 neurons in each layer is not the best case anymore. For the simple Torch model having 40 neurons gave the best result, while in autoencoder models 15 neurons were the best case.

Number of Neurons	Torch		Torch-AE-3		Torch-AE-6		Torch-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)
15,15,15,15,15	**0.003874**	**11.95**	**0.003853**	**12.37**	**0.004**	**8.97**	**0.003853**	**12.43**
30,30,30,30,30	0.003941	10.43	0.003969	9.73	0.004128	6.19	0.003877	11.87

Table 7.3 5 hidden layer PyTorch model results. The bold color in each column shows the best result of that model and the blue color shows the best result in all models.

In Table 7.3 we can see that having 5 hidden layers does not improve the MSE at all and having 15 neurons is the best case.

Comparing all three tables, we can see that AE-6 has the worst result among the 4 models built. The reason why generating 6 new features gives lower accuracy than generating 3 or 9 new features is unclear. In addition, I would like to note that generating 12 features resulted in a very unstable network, therefore we can conclude that 9 new features is the best case for this project.
The best result in PyTorch models was achieved by the 3 hidden layer networks each with 30 neurons and the use of an autoencoder to generate 9 new features.
This best result is a 13.7% improvement of MSE in comparison to the classical model, which is a little less than Dr.Rizos' results for the b-filter which was 14.15%.

7.3 Keras Model Results

Keras model results for 3, 4, and 5 hidden layer networks are shown in Tables 7.4~6 respectively.
Network specifications are as follows:
Dataset used: b-filter
Number of epochs: 20
Batch size: 200
Optimizer: Adam

The left column in the tables with "Keras" title is the model without feature engineering and the 3 columns on the right are results with feature engineering. (Keras-AE-3 indicating autoencoder generated 3 new features).
The autoencoder specifications are exactly the same as the Torch model.

Number of Neurons	Keras		Keras-AE-3		Keras-AE-6		Keras-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)
5,5,5	0.003749	14.72	0.003853	12.44	0.003822	13.15	0.00375	14.77
10,10,10	0.003725	15.29	0.003621	17.72	0.003734	15.15	0.003742	14.96
15,15,15	0.003615	17.78	0.003612	17.91	0.003679	16.38	0.003601	18.17
10,15,10	0.003711	15.6	0.003635	17.39	0.003721	15.44	0.003631	17.48
20,10,20	0.003648	17.03	0.003615	17.86	0.003723	15.39	0.003578	18.69
30,30,30	0.003589	18.37	**0.003598**	**18.23**	0.003696	16	**0.003564**	**19.02**
15,35,15	**0.003577**	**18.65**	0.003617	17.8	0.003667	16.66	0.003608	18
40,20,40	0.00362	17.67	0.003601	18.17	**0.003659**	**16.85**	0.003578	18.69
40,40,40	0.003603	18.05	0.003608	18.01	0.003661	16.08	0.003577	18.71

Table7.4 3 hidden layer Keras model results. The bold color in each column shows the best result of that model and the blue color shows the best result in all models.

In Table 7.4 we can see that similar to Torch models, using feature engineering and generating 3 and 9 new features give the best results. The best MSE was obtained from the AE-9 model with 30 neurons in each hidden layer.
In Keras models, we can see that having a small number of neurons like 5 or 10, does not affect the model performance that much when compared to Torch models.

Number of Neurons	Keras		Keras-AE-3		Keras-AE-6		Keras-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)
15,15,15,15	0.003636	17.38	0.00363	17.5	0.003697	15.98	0.003581	18.61
10,20,20,10	0.003593	18.34	0.003625	17.62	0.003674	16.51	0.003607	18.02
20,10,10,20	0.00361	17.96	0.003615	17.84	0.003705	15.81	0.003596	18.29
30,30,30,30	0.003571	18.84	**0.003573**	**18.79**	0.003674	16.51	**0.003553**	**19.25**
40,30,20,10	0.003574	18.77	0.003613	17.9	0.00372	15.46	0.0036	18.18
10,20,30,40	0.003592	18.38	0.0036	18.18	**0.003677**	**16.45**	0.003593	18.34
40,40,40,40	**0.003555**	**19.22**	0.003623	17.67	0.003722	15.4	0.003614	17.88

Table 7.5 4 hidden layer Keras model results. The bold color in each column shows the best result of that model and the blue color shows the best result in all models.

In Table 7.5 we can see that networks with 30 and 40 neurons have the best performance and again AE-9 model has the best performance.

Number of Neurons	Keras		Keras-AE-3		Keras-AE-6		Keras-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)	MSE	Improvement (%)
15,15,15,15,15	0.003621	17.7	0.003609	17.99	**0.003682**	**16.31**	0.003588	18.45
30,30,30,30,30	**0.003555**	**19.22**	**0.003592**	**18.37**	0.003698	15.96	**0.003578**	**18.69**

Table 7.6 5 hidden layer Keras model results. The bold color in each column shows the best result of that model and the blue color shows the best result in all models.

Looking at the results from all 3 tables, we have achieved an outstanding result of 19.25% lower MSE than the classical model. With PyTorch, we could only get 13.7% improvement.
We can conclude that the Keras models we developed in this project are capable of achieving lower MSE than PyTorch models. To become sure that these models work well, we are going to test them on 3 other datasets and compare the results with classical models and Dr.Rizos's previous work.

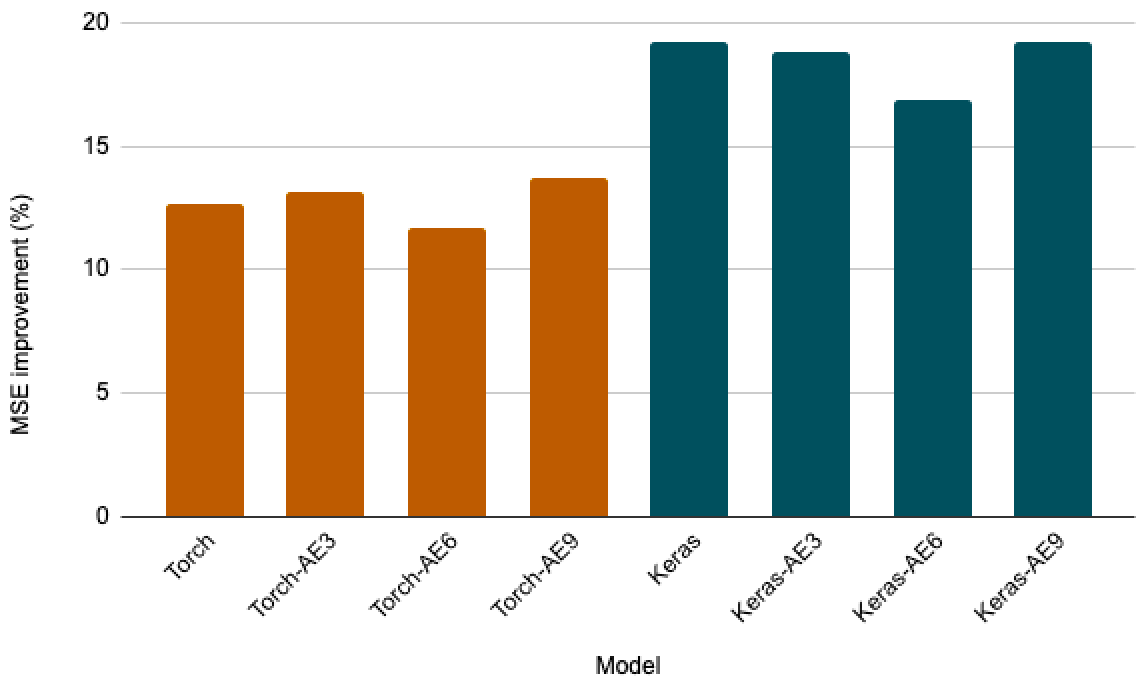

Figure 7.1 Comparison of best results of each model

In Figure 7.1 we have plotted the best result of each model to compare. We can see that Keras models in green have a higher improvement percentage in comparison to Torch models. Furthermore, AE-6 models have the lowest performance of all.

7.4 Other Filter Datasets

All the above results were using the b-filter dataset. The Mapcam data contains 4 datasets in total.
We selected the best network configuration from the results of the b-filter and ran the code on the other 3 data sets. The results are presented in this section. Notice that the number of neurons selected for Torch and Keras models are different. From the autoencoder models, only AE-9 was chosen as it gave the best results in the b-filter.

v-filter

Table 7.7 shows the PyTorch result for the v-filter data and Table 7.8 shows the Keras results. Similar to the b-filter, Keras models have lower MSE and in comparison to the classical model have an improvement of 16.17% in a model with 4 hidden layers, 30 neurons in each layer.

Number of Neurons	Torch		Torch-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)
30,30,30	0.004246	9.74	0.004238	9.9
15,15,15,15	0.004327	8.01	0.004286	8.88
40,40,40,40	**0.004205**	**10.61**	0.004424	5.96
15,15,15,15,15	0.004228	10.12	0.004265	9.32

Table 7.7 PyTorch model results using v-filter data

Number of Neurons	Keras		Keras-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)
30,30,30	0.003956	15.9	0.003976	15.48
30,30,30,30	**0.003943**	**16.17**	0.003991	15.16
40,40,40,40	0.003944	16.15	0.004018	14.58
30,30,30,30,30	0.004003	14.91	0.004018	14.58

Table 7.8 Keras model results using v-filter data

w-filter

Table 7.9 shows the PyTorch result for the w-filter data and Table 7.10 shows the Keras results. In comparison to the classical model, the Keras model has an improvement of 14.24% in a model with 4 hidden layers, 40 neurons in each.

Number of Neurons	Torch		Torch-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)
30,30,30	0.004115	9.59	0.004192	7.9
15,15,15,15	0.004178	8.21	0.004134	9.18
40,40,40,40	**0.00407**	**10.59**	0.00445	2.23
15,15,15,15,15	0.004098	9.97	0.004216	7.37

Table 7.9 PyTorch model results using w-filter data

Number of Neurons	Keras		Keras-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)
30,30,30	0.003947	13.27	0.003946	13.31
30,30,30,30	0.00393	13.66	0.003956	13.09
40,40,40,40	**0.003904**	**14.24**	0.003973	12.7
30,30,30,30,30	0.003948	13.26	0.003978	12.61

Table 7.10 Keras model results using w-filter data

x-filter

Table 7.11 shows the PyTorch result for the x-filter data and Table 7.12 shows the Keras results. In comparison to the classical model, the Keras model has an improvement of 17.13% in a model with 4 hidden layers and 30 neurons in each layer.

Number of Neurons	Torch		Torch-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)
30,30,30	0.003643	10.43	0.00386	5.08
15,15,15,15	0.003694	9.16	0.003812	6.27
40,40,40,40	**0.003597**	**11.56**	0.004012	1.35
15,15,15,15,15	0.003627	10.82	0.003788	6.86

Table 7.11 PyTorch model results using x-filter data

Number of Neurons	Keras		Keras-AE-9	
	MSE	Improvement (%)	MSE	Improvement (%)
30,30,30	0.003401	16.37	0.003458	14.93
30,30,30,30	**0.00337**	**17.13**	0.003473	14.56
40,40,40,40	0.003399	16.42	0.003481	14.36
30,30,30,30,30	0.003456	15.02	0.003495	14.01

Table 7.12 Keras model results using x-filter data

We can conclude that Keras models work well on all 4 datasets.

Figure 7.2 shows the MSE obtained in this work for classic, Torch, and Keras models using 4 different filter datasets. We can see that for all data sets PyTorch model has lower MSE than the classic model and Keras has the lowest MSE of all models. This graph shows that the models developed in this project work well on different data sets and can be used for data from other asteroids or Solar System objects too.

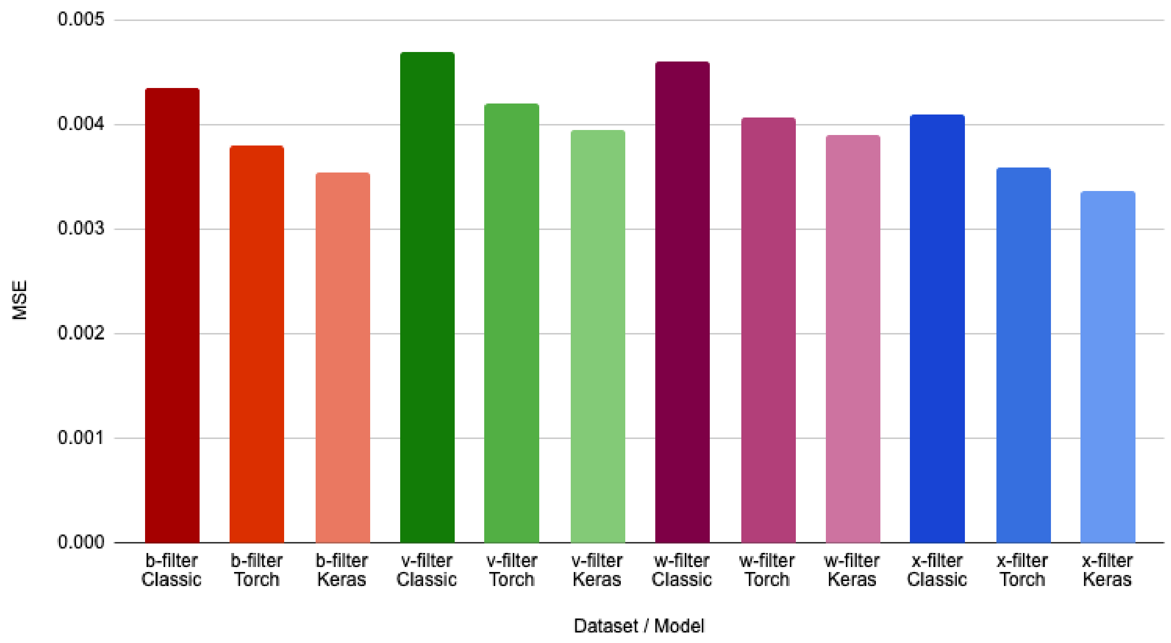

Figure 7.2 Comparing Torch and Keras model results on all datasets

7.5 Comparing Results to Previous Works

Table 7.13 is a comparison between improvement achieved by Dr.Rizos team and this project's best results.

We can see that Keras gave better results in all datasets and we could improve the results by about 3~5%.

The same comparison can be seen in Figure 7.3.

Data set	Dr.Rizos's work Improvement (%) in comparison to classical models	This project Improvement (%) in comparison to classical models
b-filter	14.15	19.25
v-filter	12.54	16.17
w-filter	11.13	14.24
x-filter	14.30	17.13

Table 7.13 Comparing model performance improvement with previous work of Dr.Rizos

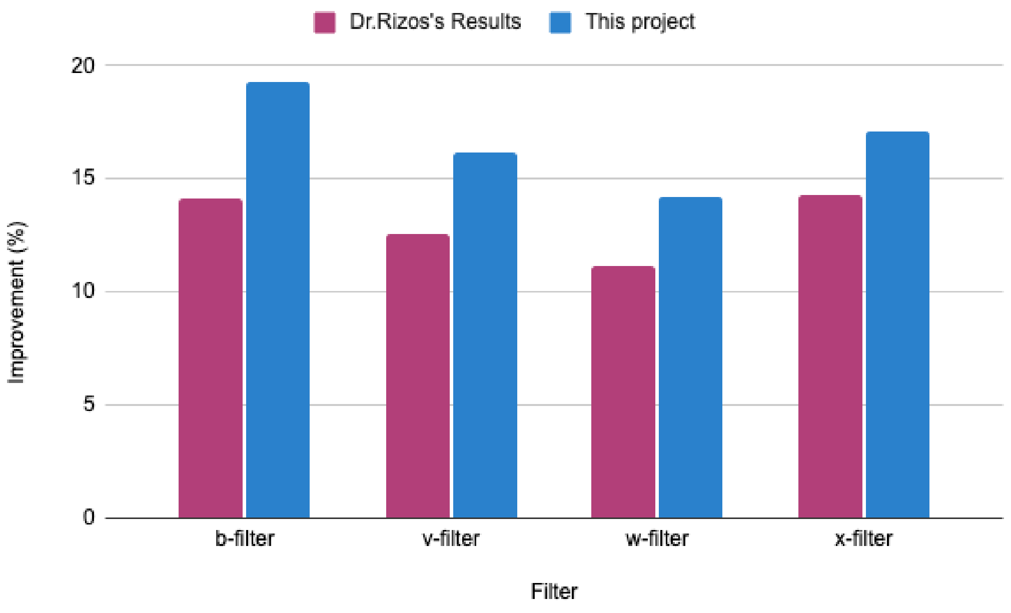

Figure 7.3 Comparing model accuracy improvement with previous work of Dr.Rizos

8. Conclusion and Future Work

When applying corrections before analyzing spectral data from resolved surfaces, photometric modeling is necessary to remove or decrease the effect of varying observation and illumination conditions.

Dr.Rizos's work has shown that developing an Artificial Neural Network can improve the accuracy of photometric modeling in comparison to classical models. In this work, we replicated the classical model and developed multiple ANN models to compare with them. We could achieve 19.25% lower MSE than the classical approach, which is 5.1% better results than Dr.Rizos's previous work.

There are still many configurations that could be done and tried to achieve better results in the 2 models used in this work. Additionally, using a new type of network model would be beneficial to see if it might be more accurate than Torch and Keras.

We can expect that another way to improve accuracy in our models is by adding latitude and longitude along with the phase, emission, and incidence angles as input. This is called regional photometry and having more information on each pixel might improve the accuracy. Although we might need other neural network models rather than an ANN.

Finally, photometric corrections are only one application of photometric modeling. There are many characteristics of the surface such as particle size, porosity, and roughness than can be inferred from studying reflected light from an asteroid surface. This is usually done with the Hapke model, which presents a set of parameters with physical meaning. Developing a machine learning algorithm like the one we developed in this work can overcome some of the difficulties of the classical models such as convergence issues and correlations between supposedly independent parameters as well as improving accuracy.

References

[1] J. L. Rizos et al, 2021b, Using artificial neural networks to improve photometric modeling in airless bodies, https://arxiv.org/abs/2106.01363, last accessed 15/8/2022

[2] Photometry, Wikipedia, available at https://en.wikipedia.org/wiki/Photometry_(optics), last accessed: 20/8/2022

[3] Li et al. 2015. Asteroid photometry. Asteroids IV (P. Michel et al., eds.), pp. 129–150. Univ. of Arizona, Tucson. https://doi.org/10.2458/azu_uapress_9780816532131-ch007, last accessed 28/8/2022

[4] Lambertian reflectance, available at https://en.wikipedia.org/wiki/Lambertian_reflectance, last accessed 17/8/2022

[5] Golish, D.R. et al., 2021a. Disk-resolved photometric modeling and properties of asteroid (101955) Bennu. Icarus 113724. DOI: 10.1016/j.icarus.2020.113724

[6] Hapke, B. 2012. Theory of Reflectance and Emittance Spectroscopy, 2nd ed. Cambridge University Press, Cambridge. https://doi.org/10.1017/CBO9781139025683, last accessed 28/8/2022

[7] Hapke Parameters, Wikipedia, available at https://en.wikipedia.org/wiki/Hapke_parameters, last accessed: 20/8/2022

[8] Shkuratov, Y., et al., 2012. A critical assessment of the Hapke photometric model. J. Quant. Spectr. Rad. Transf., 113, 2431-2456. https://doi.org/10.1016/j.jqsrt.2012.04.010, last accessed 28/8/2022

[9] Xiao-Duan Zou et al. 2021. Photometry of asteroid (101955) Bennu with OVIRS on OSIRIS-REx, https://www.sciencedirect.com/science/article/pii/S0019103520305194, last accessed 28/8/2022

[10] OSIRIS-REx mission, NASA, available at https://solarsystem.nasa.gov/missions/osiris-rex/in-depth/ last accessed: 20/8/2022

[11] Planetary Data System, available at https://pds.nasa.gov, last accessed 20/8/2022

[12] OSIRIS-REx SPICE Kernel collection, available at https://arcnav.psi.edu/urn:nasa:pds:orex.spice:spice_kernels, last accessed 15/8/2022

[13] Artificial neural network, available online at https://en.wikipedia.org/wiki/Artificial_neural_network, last accessed 12/8/2022

[14] Kaiming, H., et al. 2015. Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. 2015 IEEE International Conference on Computer Vision (ICCV). https://doi.org/10.1109/ICCV.2015.123, last accessed 15/8/2022

[15] Sigmoid Function, available at https://en.wikipedia.org/wiki/Sigmoid_function, last accessed: 22/8/2022

[16] Rectified Linear Unit (ReLU) available at https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/, last accessed: 21/8/2022

[17] Adaptive Moment Estimation (ADAM), available at https://blog.marketmuse.com/glossary/adaptive-moment-estimation-adam-definition/, last accessed: 22/8/2022

[18] Autoencoders, available at https://medium.com/ai%C2%B3-theory-practice-business/understanding-autoencoders-part-i-116ed2272d35, last accessed: 20/8/2022

[19] PyTorch, available at https://pytorch.org/, last accessed 12/8/2022
[20] PyTorch, available at https://www.techtarget.com/searchenterpriseai/definition/PyTorch, last accessed 12/8/2022
[21] Keras, available at https://keras.io/, last accessed: 20/8/2022
[22] Tedesco, E.F., et al., 1982. The eight-color asteroid survey – Standard stars. Astron. J. 1585–1592
[23] Overfitting and methods of addressing it, Image data from https://analystprep.com/study-notes/cfa-level-2/quantitative-method/overfitting-methods-addressing/, last accessed 28/8/2022
[24] Bias-Variance tradeoff, Image data from https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html, last accessed 28/8/2022
[25] A Step-by-Step Explanation of Principal Component Analysis (PCA), available at https://builtin.com/data-science/step-step-explanation-principal-component-analysis, last accessed: 22/8/2022
[26] Rizos et al. 2019. Spectral clustering tools applied to Ceres in preparation for OSIRIS-Rex color imaging of asteroid (101955) Bennu. Icarus. Available at https://www.sciencedirect.com/science/article/abs/pii/S0019103518307942?via%3Dihub, last accessed 28/8/2022
[27] Rizos et al. 2021a. Bennu's global surface and two candidate sample sites characterized by spectral clustering of OSIRIS-REx multispectral images. Icarus. Available at https://www.sciencedirect.com/science/article/abs/pii/S0019103521001482, last accessed 28/8/2022
[28] Integrated Software for Imagers and Spectrometers (ISIS), available at https://www.usgs.gov/software/integrated-software-imagers-and-spectrometers-isis, last accessed 28/8/2022
[29] Introduction to SPICE, available at https://naif.jpl.nasa.gov/pub/naif/toolkit_docs/IDL/info/intrdctn.html, last accessed 28/8/2022
[30] CPU vs. GPU, available at https://www.weka.io/blog/cpu-vs-gpu/#:~:text=High%20Data%20Throughput%3A%20a%20GPU,what%20a%20CPU%20can%20handle, last accessed 28/8/2022