DEPARTMENT OF MECHANICAL & AEROSPACE ENGINEERING

University of Strathclyde Engineering

Image Processing Using Deep Learning for Optical Navigation of an Asteroid Deep in Space

Aidan Kelly

201831699

Supervisor: Dr. Jinglang Feng

A thesis submitted in partial fulfilment of the requirements of the MEng Mechanical Engineering degree.

April 19, 2022

Word count: 9682

Abstract

In order to improve the autonomy of the navigation strategy for an asteroid deep in space, this project introduces artificial intelligence (AI). To resolve the issue of large delays in communication between earth and spacecraft. Deep Learning (DL), which is a subcategory of Machine Learning (ML), must be implemented to achieve this autonomy. The DL solution used is optical navigation, a strategy that requires an on-board camera to take pictures of the asteroid and an Image Processing (IP) algorithm capable of extracting visual data necessary for the navigation. When employing DL, the performance of an IP algorithm grows exponentially as the dataset grows. With consideration of the number of images being processed for data, this project focuses on DL. The main body of this report will give a detailed account of constructing and optimising a convolutional neural network (CNN) for the use of centroid detection of Didymain, which is within the binary asteroid system (65803) Didymos. The centroid of Didymain can be considered the centre of mass of the binary asteroid system, as one of the asteroids is significantly larger than the other. This project in particular looks at optimising the hyperparameters of a neural network to gain accurate results, with little underfitting. The addition of more convolutional layers, solved through mathematical manipulation, meant the model could characterise the asteroid precisely when used at the simulated 30km distance. The validation and training losses, as well as graphical representations, reveal that satisfactory results may be produced with a relatively simple deep neural network. With further research and optimisation, the IP algorithm achieved excellent results.

Contents

Table of Figures

Table of Tables

Glossary of Terms

Backpropagation	Backpropagation, short for "backward propagation of errors," is an algorithm for supervised learning of artificial neural networks using gradient descent. Given an artificial neural network and an error function, the method calculates the gradient of the error function with respect to the neural network's weights [1].
Binary Asteroid System	A system of two asteroids orbiting their combined centre of mass. Typically a small asteroid and a large asteroid.
CIFAR-10	The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images [2]. It is used for image classification.
CubeSat	A CubeSat is a miniature satellite (10cm x 10cm x 10cm), weighing roughly 1kg.
Downsampling	Downsampling of 2D images is a technique employed in order to reduce the resolution of an input image [3]**.**
Epochs	The number of complete passes through the training and validation datasets.
Hyperparameters	The parameters used to control the learning process of a machine learning model.
Image Classification	An IP algorithm used to classify different objects in an image, used to i.e. distinguish pictures with dogs from pictures with cats

Keypoint Detection	An IP algorithm used for detecting specific coordinates on an image.
Stride	The number of pixels the kernel moves along horizontally, or vertically during the convolutional operation.

Note

All figures, and section numbers referred to, have been hyperlinked, to take the reader to the desired section or figure. Unfortunately the figures hyperlink takes the reader to the figure description, instead of the actual figure, so readers may need to scroll up a small amount, to see the figure they have been taken to. Issues with the format updating in bold, meant that the glossary terms could not be hyperlinked in the main text

Nomenclature

<u>**Abbreviation**</u>	<u>**Meaning**</u>
AIDA	**A**steroid **I**mpact and **D**eflection **A**ssessment mission
CNN	**C**onvolutional **N**eural **N**etwork
CPU	**C**entral **P**rocessing **U**nit
DART	**D**ouble **A**steroid **R**edirection **T**est
DL	**D**eep **L**earning
DRACO	**D**idymos **R**econnaissance and **A**steroid **C**amera for **O**ptical navigation
ECP	**E**arly **C**haracterisation **P**hase
ESA	**E**uropean **S**pace **A**gency
GPU	**G**raphics **P**rocessing **U**nit
IP	**I**mage **P**rocessing
ML	**M**achine **L**earning
MSE	**M**ean **S**quared **E**rror
NASA	**N**ational **A**eronautics and **S**pace **A**dministration
PANGU	**P**lanet **A**steroid **G**eneration **U**tility
ReLU	**R**ectified **L**inear **U**nit
TRL	**T**echnology **R**eadiness **L**evel

Symbol	Description
d_i	The desired output of the model
F	The Filter size (size of the kernel matrix)
N	Number of images within the specified sample (batch or epoch)
P	Number of zero padding layers
p_i	The predicted output of the model
s	The size of the stride taken after each process of the kernel
W	The input size to the layer

Acknowledgements

First and foremost, I want to express my gratitude to my supervisor, Dr. Jinglang Feng, and supporting Ph.D. student, Aurelio Kaluthantrige, for their tremendous guidance and encouragement during this complex and unique project. Aurelio Kaluthantrige has been particularly helpful with the Deep Learning aspects of this project. I would also like to thank my family and friends for their support.

1.0 Introduction

Deep Learning is an evolved subset of traditional ML, capable of exponentially improving in performance as the size of the dataset being processed increases [4], whereas an ML model reaches a saturation point. This is the point whereby an increase in the size of the dataset to the program, causes no change in the performance of the algorithm. On the contrary, DL algorithm performance increases as the size of the dataset is increased.

The conceptual understanding of machine learning was first published in 1943 by neurophysiologist Warren McCulloch and cognitive psychologist Walter Pitts, under the title “A Logical Calculus of the ideas Imminent in Nervous Activity”, in which they described the “McCulloch – Pitts neuron”, which was considered the first mathematical model of a neural network [5]. Their explanations of how previous layers can give permanent changes in the network, was an understanding of what is now known as model fitting, which can be detrimental to models, as explained in section 2.5.8. Another interesting aspect of this history is that neurons require connections from neighbouring neurons to reach an impulse that can subsequently activate said neuron. Neurons are either active or not active, it is known as an “all-or-none” process. The fitting of models is now understood to be significantly affected by the learning rate and/or the dropout percentage, along with other hyperparameters, which can randomly affect one another when changed in value. This paper was constructed by building on the ideas of Alan Turing’s “On Computable Numbers”, published in 1937, which conveyed the idea of a computer being able to solve any mathematical problem in a symbolic form using the thought pattern of a human [6]. Alan’s paper was describing what came to be known as the “Turing Machine”, which was ultimately the foundation of ML. One of the first papers written explaining a convolutional neural network model used for pattern recognition was by Kunihiko Fukushima in 1979, later published in 1980. His ideas were extensions of the work done by Hubel and Wiesel, Nobel prize-winning neuroscientists who had the understanding that neural circuits were used to process information in a way that allowed it to make a decision or response. Figure 1.1 shows the interconnections between layers in the Neocognitron, this was the 1980 representation of a convolutional neural network (CNN). Fukushima’s Neocognitron model as shown could detect the similarities in patterns even when they had slightly different geometries or changes in spatial position. Previous researchers failed to produce models that could recognise pattern similarities after changes in position and geometry of the original image [7].

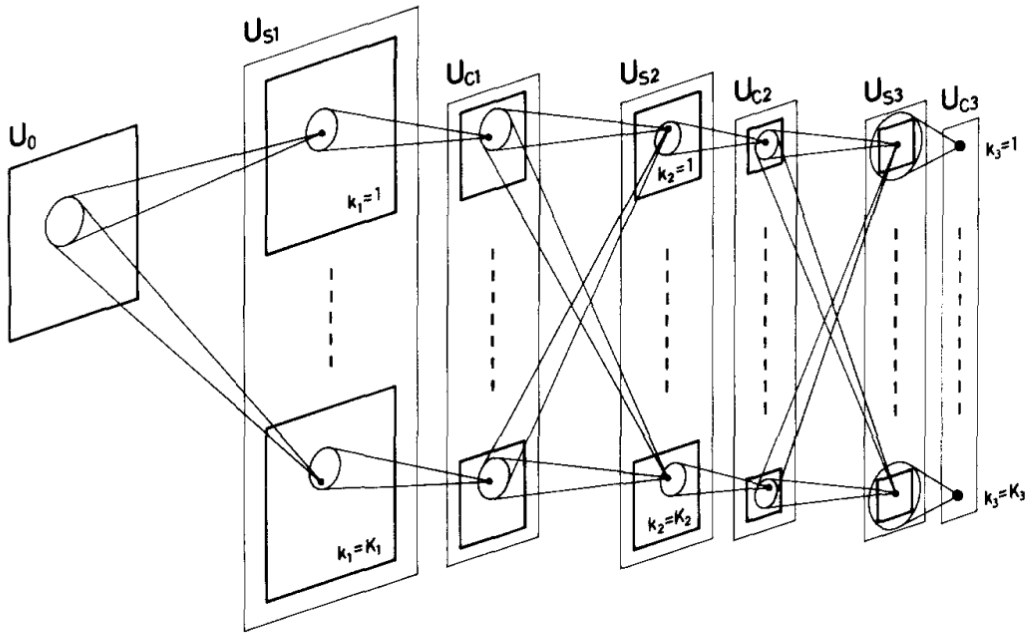


Figure 1.1: Fukushima's Neocognitron, a depiction of what is now known as a CNN [7].

Over the years these ideas have been elaborated into what is known as Deep Learning, which has led to being used for problems such as centroid detection of asteroids in space, as detailed in this paper.

Didymos was first discovered on the 11th of April 1996, by Joseph Montani, a researcher of Spacewatch at Kitt Peak National Observatory in Tucson, Arizona. It was later discovered in 2003 that Didymos was two asteroids i.e. it had a moon orbiting the main body, so was then confirmed as a binary asteroid system [8]. The approximate diameters of the asteroids are 780m for the largest (Didymain), and 160m for the moonlet asteroid (Dimorphos).

Space is vast and observing what can be seen of it from Earth is no mean feat, asteroids close to the planet evade detection all the time. There is always a real threat of an asteroid hurtling toward earth, which is why there is a need for a planetary defence system. The National Aeronautics and Space Administration (NASA) and the European Space Agency (ESA) have joined forces to explore the complicated nature of asteroid deflection. One of the first tests that will be carried out is the Asteroid Impact Deflection and Assessment mission (AIDA) that has a sub mission "HERA", which is responsible for monitoring changes in the orbit of the Dimorphos asteroid. The Direct Asteroid Redirection Test (DART) mission, is the parent mission of HERA, where a space probe will crash into the moonlet asteroid of the binary system. Four years after the crash has taken place, HERA will measure the changes in orbit of the moonlet. During the mission one of the most important tasks to accomplish is orbiting Didymos, which will be attempted at multiple orbital distances. As the spacecraft will experience a huge delay in response time from earth, there cannot be entirely manual piloting of the craft. At the time of writing the spacecrafts communication with earth would be delayed by 13 minutes and 24 seconds [9].

The solution to navigate this binary asteroid system, is through a sub category of machine learning, which is deep learning. This paper will cover tracking of the centroid position of the Didymain asteroid, using any given image of the asteroid system (65803) Didymos. As explained in this report, the Early Characterisation Phase (ECP) of this mission, where the orbital distance from the asteroids is 30 kilometres, will utilise an IP algorithm for optical navigation. Autonomous navigation allows the spacecraft to update its position relative to the asteroid without ground-based information. Having this on board knowledge is a huge benefit to the mission as the spacecraft is able to change its position quickly without having to send signals back and forth between earth and spacecraft [10]. As there is always a risk of asteroids colliding with earth, there should be commissioning of more important missions to mitigate the chances of earth being hit, as it could lead onto total extinction of the human race. This paper details the training and validation phases of a CNN used for keypoint detection of Didymain's centroid. Further on in the paper will explain the convolutional layering structure implemented to produce accurate results.

2.0 Literature Review

To better comprehend the context and intricacy of the areas of interest specific to this report, the following literatures were examined in conjunction with this project. To acquire a better grasp of IP using DL and orbital navigation, multiple published research papers, publications, and other sources have been thoroughly analysed.

2.1 Asteroid Deflection Missions

As recent as the 11th of March 2022, NASA reported an asteroid hitting earth just 2 hours after it was first detected by astronomers [11]. An article published that recent, goes to show the threat is continuous. This proves that the human race must continue to research the nature of asteroids and take the matter seriously. Orbiting asteroids, and analysing the changes in orbit of asteroids, is the first steps of many that NASA, ESA and other space agencies will partake in to reduce the threat of Earth being the final destination of local asteroids in the solar system [12].

The asteroid deflection mission detailed in this project will be carried out by NASA and ESA. The mission known as DART, carried out by NASA was launched heading for Didymos on the 24th of November 2021, by Space X's Falcon 9 rocket, as shown in Figure 2.1. After stage 2 of the Falcon 9, DART separated from the rocket, and will continue its journey towards (65803) Didymos [13].

Figure 2.1: SpaceX Falcon 9 lifts off with NASA's DART spacecraft on board [14].

The probe will collide with Dimorphos at a speed of 6.6 kilometres per second, between the 26th of September and the 1st of October, 2022. At which point in time the asteroid system will be roughly 11 million kilometres from Earth [15]. This will be the world’s first large-scale mission of asteroid deflection technology for planetary defence. Just before the crash of the spacecraft, DART’s only onboard hardware, the Didymos Reconnaissance and Asteroid Camera for Optical navigation (DRACO), will get up close detailed images of the asteroid, which can be relayed onto the spacecrafts being used for the later missions, such as HERA. After the DART probe has carried out its purpose, a secondary spacecraft will be deployed from DART a few days before its annihilation. This spacecraft is known as LICIACUBE CubeSat, a contribution from the Italian Space Agency. This Italian CubeSat will be used to get detailed photos of the impact crater if there isn’t an abundant amount of debris in the way of the optical device after the collision. It will also go on to take pictures of the rear side of the asteroid that DRACO wasn’t able to get. Once the initial impact of DART occurs, there is a follow up mission four years later, launching in October 2024, which will carry out an investigation of Dimorphos in late 2026. This is the ESA’s HERA mission, ultimately responsible for conducting analysis of the effects the collision has had on the orbit of Dimorphos around Didymain over the last four years [16]. LICIACUBE is important to the HERA mission as it will benefit tremendously from all the images taken prior to its arrival. HERA will also have an onboard laser which will measure the size, shape and mass of the moonlet asteroid after collision.

2.2 HERA’s Autonomous Vision-Based Navigation

Jointly the asteroid deflection missions are known as AIDA, where the collaborators are NASA and ESA, carrying out the two missions respectively. Knowledge gained from ESA sources, helped to make up the building blocks of this project, and gave a good level of understanding on the topics being explored for this report. Figure 2.2 shows a depiction of the main asteroid Didymain, with Dimorphos beside it.

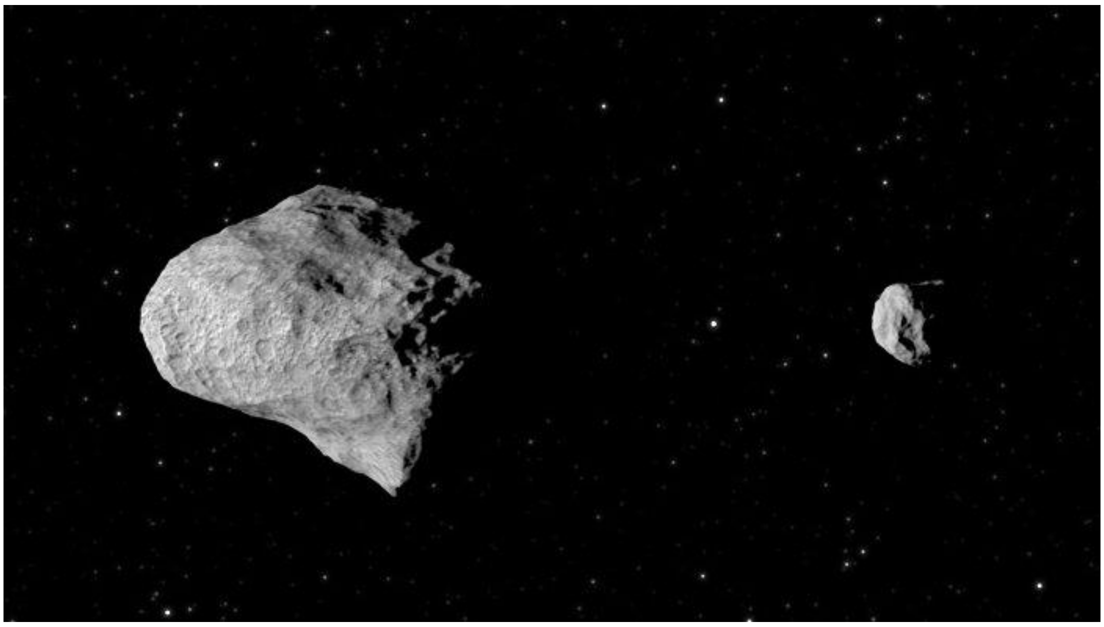


Figure 2.2: An idea of how the binary asteroid system (65803) Didymos will look [17].

As the ECP is the aim of this project, it was key to analyse the papers provided from the engineers and scientists involved, to understand this specific phase better. One of the papers explains how the ECP will take place beyond the distance of the gravitational attraction of Didymoon (roughly a 30km orbital distance) [18]. This is important to note, as this is a distance where ground-based navigation and autonomous navigation will be working simultaneous, in order to achieve stable orbiting of the binary system. This literature covers the analysis of position errors and shows how they aren’t relevant for a distance of 30km, as it is such a large orbiting distance.

In order to begin simulations of the asteroid, ESA used what is known as the Planet Asteroid Generation Utility (PANGU) to generate the images used in the datasets of this project [19]. These images can give an accurate representation of how Didymos would look in space, and can use a range of different positions relative to the sun to simulate shadows being cast over the asteroid by using programmable Graphics Processing Unit (GPU) shaders. The PANGU images used in this project used a variety of different positions of the asteroid, which adds huge complication to the IP algorithm accurately predicting the centroid. This was important to use, as the IP algorithm had to be predicting the centroid positions of Didymain, as if it were a real life mission. Without the use of different asteroid positions, the IP algorithm wouldn’t be considered robust or accurate. The centroid of Didymain was considered as the centre of mass of the binary asteroid system, as it is significantly larger than the moonlet Dimorphos, so it is safe to make this assumption.

Another report analysed, covered the vision based navigation details and how CNN’s can be used for IP of asteroids, in order to work out the centroid of the asteroid. During later stages of the mission, it is important to cover areas such as close proximity orbits, these will be carried out using full autonomy. Although a constant stream of information can go back and forth, the spacecraft is still operating 13 minutes ahead of what the ground navigation team know. This would be too dangerous a mission to attempt during close contact operations, as it could result in collision. Full autonomy will allow the spacecraft to update in real time, correcting its course in small time intervals, instead of getting delays if it were ground based navigation [10]. The IP algorithm for centroid detection, alone, would not work for close proximity operations, as there may be other factors to account for at closer distances, such as perturbations of the spacecraft trajectory, and resonance. This mission will pave the way for future asteroid deflection missions, and help space agencies to understand the threat of asteroids and effective means to resolve the threat posed by near earth objects.

2.3 Technology Readiness Level

Nasa proposed a technology readiness level (TRL) to decide whether or not developed technologies can be used in real life missions [20]. This is very important when using such advanced and complexed software for important missions. This tier list is visually explained in Figure 2.3 showing that the scale is from 1 to 9, where 1 is the lowest and 9 is the highest. At TRL 1 the research has begun on the technology and results of the study are being used to further developed the software or hardware. TRL 2 is when the basic principles of the technology have been rigorously studied, and now applications of the technology have been established. Further up the scale at TRL 6, the technology has a working prototype or representation model. TRL 7 is when the TRL 6 working model has been tested in a space environment. TRL 8 has been tested and “flight qualified” and is ready to be implemented into an already existing technology or technology system. After the technology has been successful in a mission it can be considered “flight proven” and can be given TRL 9 status.

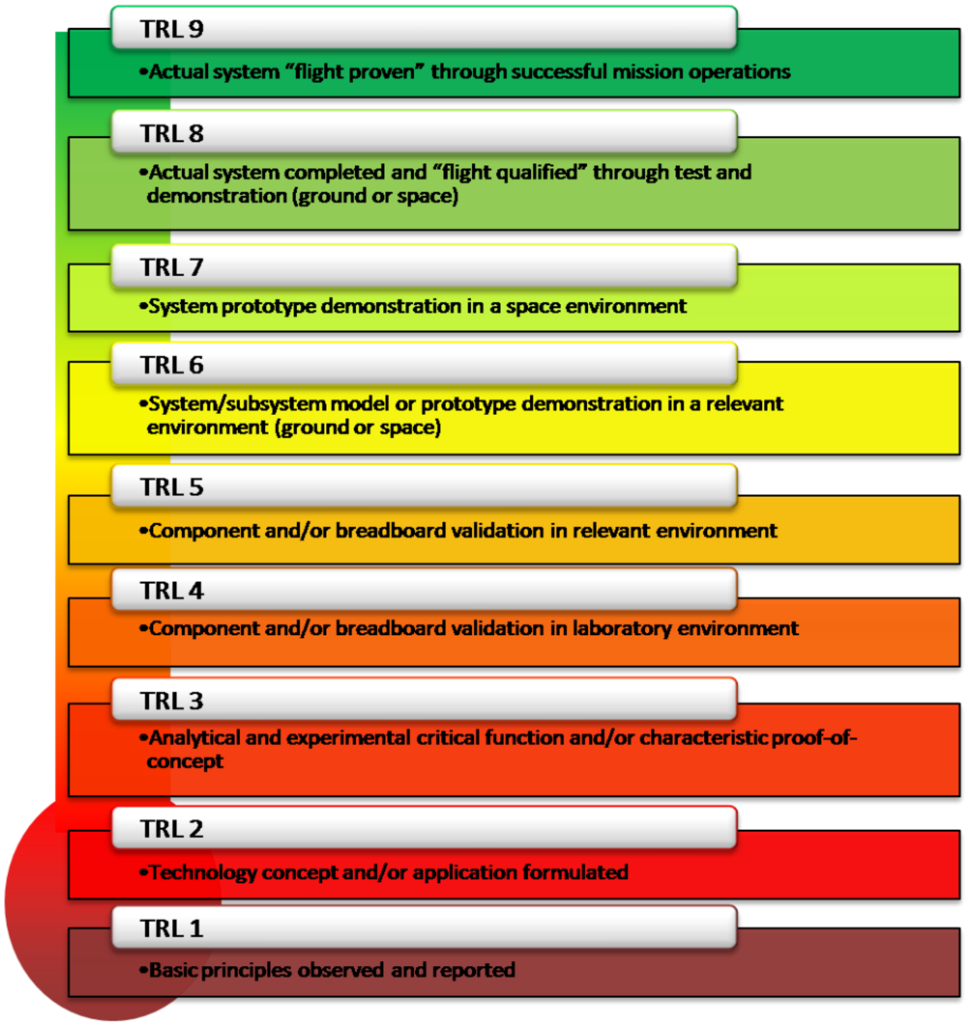

TRL 9
•Actual system “flight proven” through successful mission operations
TRL 8
•Actual system completed and “flight qualified” through test and demonstration (ground or space)
TRL 7
•System prototype demonstration in a space environment
TRL 6
•System/subsystem model or prototype demonstration in a relevant environment (ground or space)
TRL 5
•Component and/or breadboard validation in relevant environment
TRL 4
•Component and/or breadboard validation in laboratory environment
TRL 3
•Analytical and experimental critical function and/or characteristic proof-of-concept
TRL 2
•Technology concept and/or application formulated
TRL 1
•Basic principles observed and reported

Figure 2.3: The Technology Readiness Level (TRL) used by NASA and ESA [20].

The TRL has been developed so that early-stage technology or conceptual technology, has to prove itself before being used for expensive missions. The TRL used by the ESA is very similar to the one used by NASA, if not the same. This TRL scale has aided in the prevention of errors occurring during missions, by eliminating risky technology in the design stages. If the technology designed for this project were to be used for such a mission, it would require an extensive amount of testing before being at the TRL suitable enough for real life missions. Often the older technology is utilised, as this testing comes at a cost, which might not be in the budget of these already very expensive missions [21].

For the HERA mission, the technologies mentioned in this report have been tested via simulations of the asteroid, as shown in Figure 2.4.

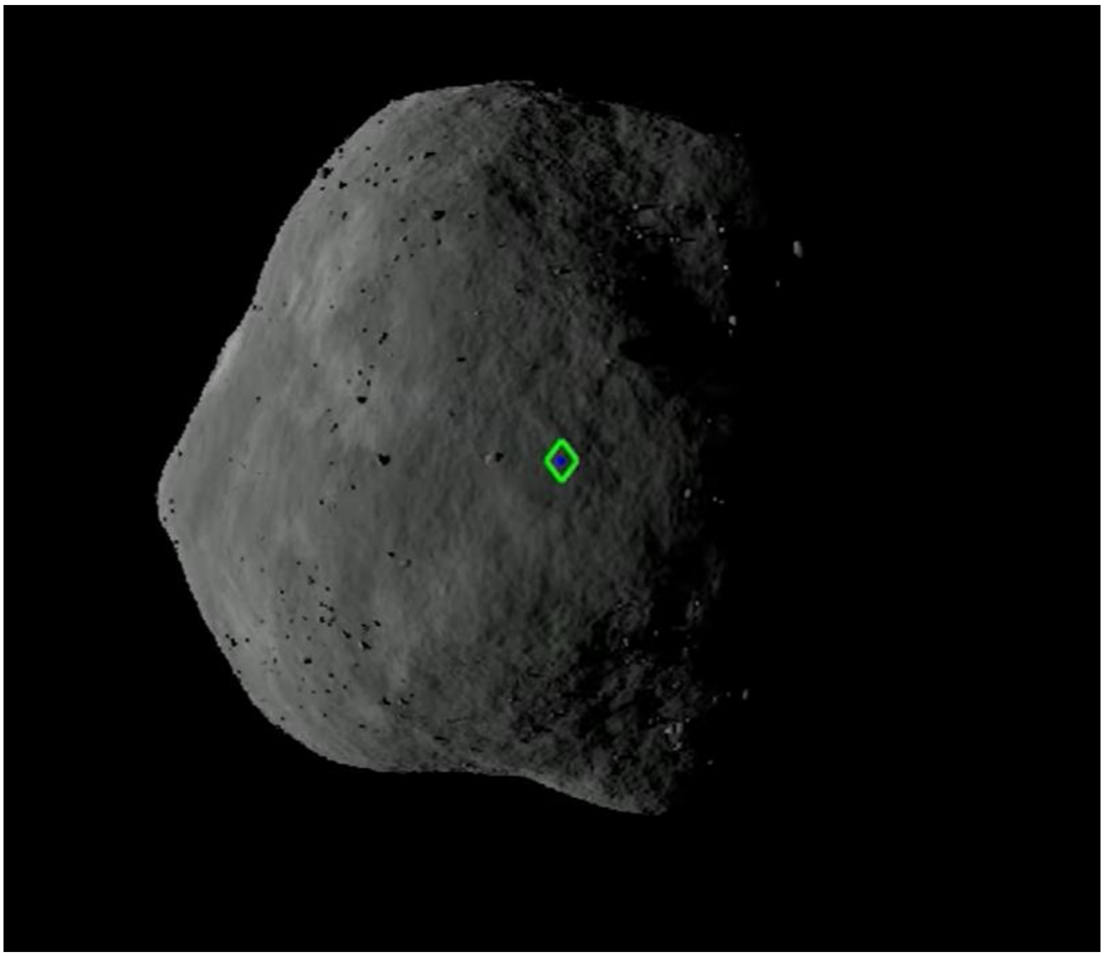


Figure 2.4: Centroid Detection of a simulated version of the Didymain asteroid [22].

This shows that HERA DL technologies have been tested in a laboratory environment. This would give HERA a TRL of 4, it may potentially be a TRL of 5 if the technology has been tested more extensively via real life asteroid scaled down prototypes. There is currently no record of this, and the HERA technologies might have been given further validation from its similarities with flight proven technology. The mission design of HERA is low in cost and highly effective, which will help usher in a new era of deep space missions [22].

2.4 Deep Learning

Deep Learning (DL), is a better tool for large data sets, in comparison to traditional machine learning (ML). Applications of DL only occurred recently i.e., in the early 2000s. It concludes that ML may work better for small data sets, but not large ones, as ML has a saturation point, which means performance eventually converges at a certain size of data set. DL on the other hand excels exponentially with the more data it is provided, as long as the algorithm is equipped with enough processing power. This processing power is normally provided by GPUs, although the Central Processing Unit (CPU) was enough for the IP algorithm in this project. When deriving sigmoid activation functions, to return to previous layers an issue was encountered with the sigmoid function saturating (going to 0), this meant that the signal required to correct the loss error was lost. This literature gave good analysis of the vanishing gradient issue of the activation function in deep learning, and how computer scientists worked tirelessly to find a solution, which was the Rectified Linear Unit (ReLU) activation function, which will stay at a gradient of 1 at any value above 0, allowing the use of very deep neural networks [23]. This project used the ReLU activation function for all simulations carried out, as explained in section 2.5.3.

2.5 Convolutional Neural Networks

CNN's are a class of artificial neural networks, which are used for IP algorithms. The use of them is effective for data with a grid pattern i.e. an image. They aren't like other neural networks where the input is always mapped to the output, and all neurons are connected [24]. This type of network uses dropout effect to help the learning rate of the network, it can't rely on specific neurons to be active, as it may randomly lose reliant neurons from the dropout effect. Different layers make up a CNN, the three most common sorts of layers are: convolutional layers, pooling layers and fully connected layers. Figure 2.5 shows a simple CNN architecture.

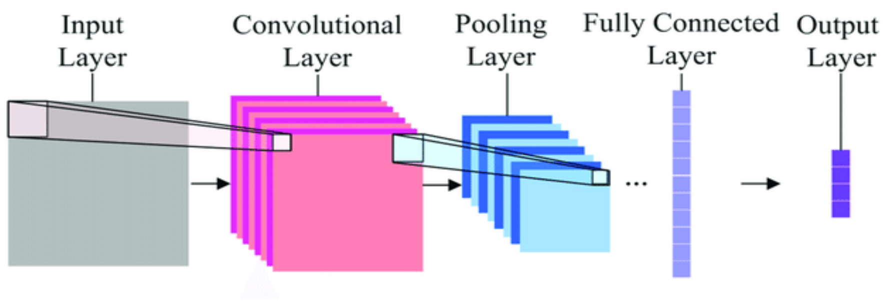


Figure 2.5: A simple CNN architecture [25].

This architecture can be better represented by showing the layering of a CNN, as shown in Figure 2.6, which is the layering of a keypoint detection model, using 5 convolutional layers.

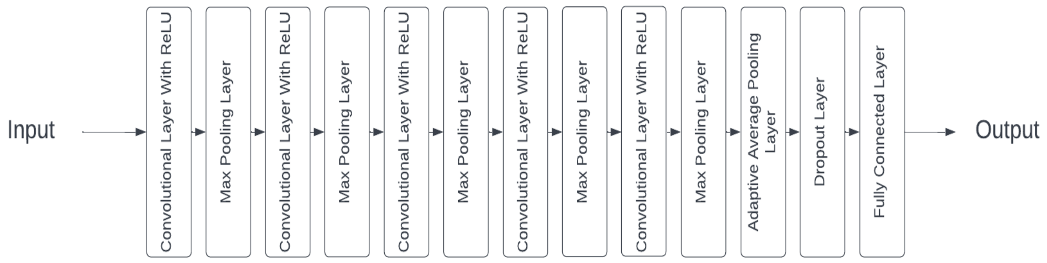

Input
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Max Pooling Layer
Adaptive Average Pooling Layer
Dropout Layer
Fully Connected Layer
Output

Figure 2.6: A typical CNN used for keypoint detection.

The reason this model in Figure 2.6 is considered to be used for a keypoint detection model, is that there is no activation function on the fully connected layer at the very end of the architecture, this allows the program to produce the predicted keypoints as the output.

2.5.1 Convolutional Layer

This layer has a 2D input matrix, with one colour channel if being used for grayscale images. It uses a convolutional kernel to look for the key attributes of the image, in this case it will scan using kernels of varying sizes to find the asteroid. After scanning the image, the activation function can be used, as explained in section 2.5.3. Shown in Figure 2.7 is the kernel operation, where the kernel (K) in the image is being used to weight different areas of the image. It uses a kernel weighting randomly generated by the program originally, and then after each image batch the IP algorithm runs through, the weights update [26], via use of an optimiser.

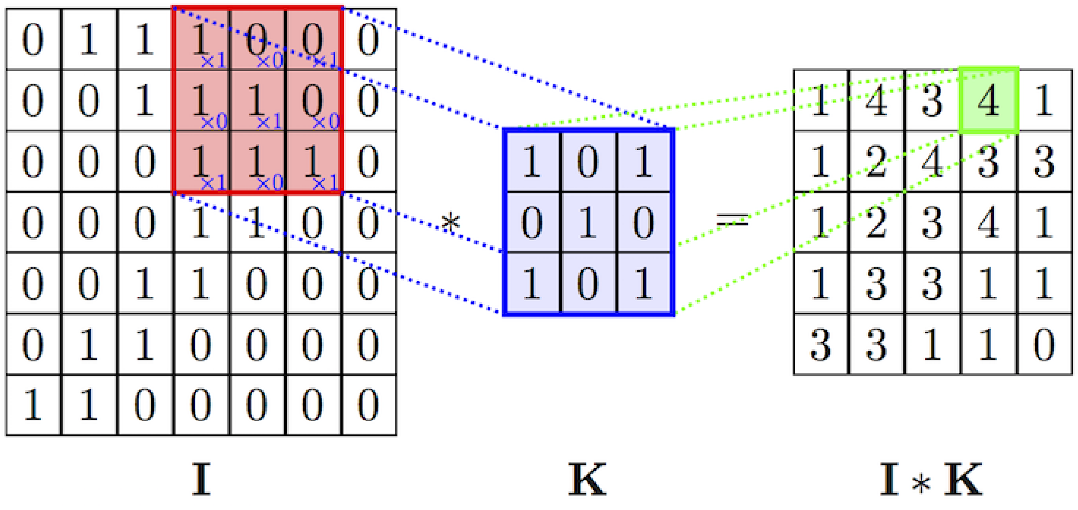


*Figure 2.7: How convolutional kernels work, "I" represents the input to the layer, "K" represents the 3x3 kernel and "I*K" represents the convolutional matrix being formed [27].*

Padding can be added around an input matrix to the next layer to mitigate downsampling of the inputs, this is an important aspect of CNN's as there is a certain input size that is allowed for the last max pool layer of the keypoint detection network. Once the output reaches e.g. a 2x2 matrix, then no further layers can be added. To get a bigger output matrix, padding can be implemented to add an extra border of zero's around any matrix at a given layer. This use of padding is used in section 3.6.

2.5.2 Learning Rate

This is the hyper parameter used to learn the loss graph, it employs step changes after each image is processed, in order to try and reach the lowest point of the loss graph (zero loss). The lowest point may never be reached but the learning rate can be optimised to produce the smallest possible losses for the specific IP algorithm. A learning rate that is too small will never reach close to the lowest point because the step size will be too small to get to the point of lowest loss [28]. A large learning rate will fluctuate between a high and low loss, as the step change is too large, so it skips past the lowest point and then over compensates in the other direction by negating the gradient, which again makes it skip over the loss function, but now in the opposite direction. An illustration of this is shown in Figure 2.8.

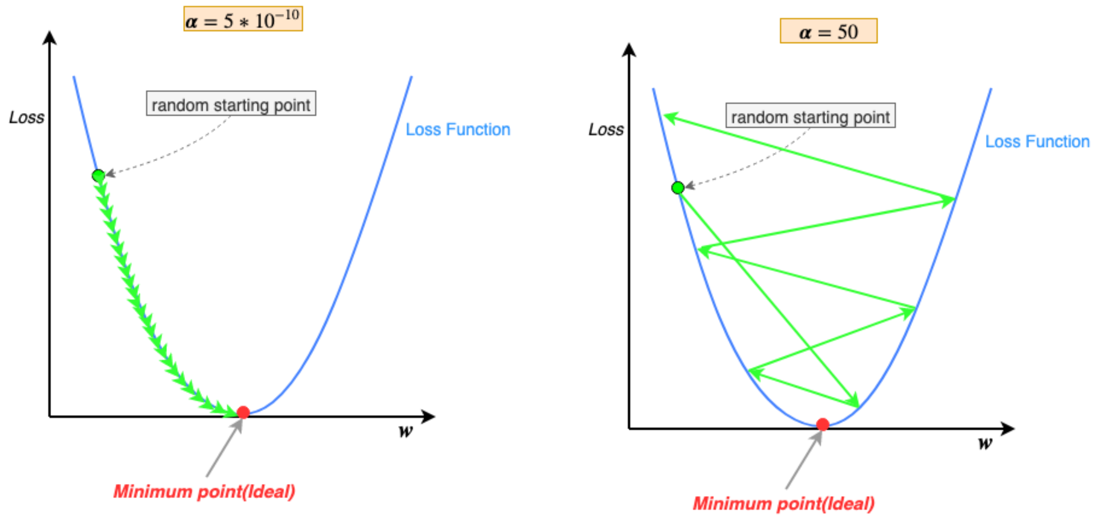


Figure 2.8: A Visualisation of the Loss against Weight graph of a neural network, where α represents the value of the learning rate being used [29].

These graphs show how step sizes change as the learning rates change. The left-hand side shows a small learning rate, where the green arrow heads are close together, taking a long time to reach a low point of loss. On the graph to the right, it can be seen how the large learning rate will cause the prediction to change from a negative gradient to a positive gradient in one step change, this fluctuation is not suitable for a CNN, and too small a learning rate will take a very long time to converge. The initial generation of weights can be seen in the graph as well, denoted as "random starting point", which is describing that the program assigns a random value for the weights at the start of every centroid detection simulation.

2.5.3 ReLU Activation Function

Rectified Linear Unit (ReLU) is the common function to use for CNN's, it is a piecewise linear function. The gradient of the ReLU function is either zero or one, based on whether its input is negative or not, i.e. the neuron is either negative or zero in the convolutional kernel if it is irrelevant to the model [24]. If the input is x to the activation function, it will output x, where as if the input is 0 or negative it will output 0. In pattern recognition, this is explained as if there is a certain pattern detected, the output will be non-zero, but if the pattern isn't recognised then the output will be zero. For the images presented in this project, the zero values would be the black background of the images, as the pixels in black are represented as the number zero by the computer for this IP algorithm. Therefore, it is computationally inexpensive to calculate this gradient, unlike the sigmoid activation function where the computational cost scales exponentially as the number of activation layers increases. Equation 2.1 shows the ReLU activation function.

$$f(x) = \max(0, x) \tag{2.1}$$

This function is used in the layer after each convolutional layer of a network, to activate the relevant areas of the model. there is no parameter inside the ReLU function, hence no need for parameter learning in this layer [26].

2.5.4 Pooling Layers

The pooling layers do the opposite of padding, they downsample the images [30], by halving each dimension of the input resolution given to the pooling layer. The input to the pooling layers is divided into kernels and either the max number in the kernel is taken, or the average of the numbers in the kernel is taken. This is known as max pooling and average pooling, respectively. Figure 2.9 shows the 2x2 kernels created from the pooling layers. It also shows that the average values are not rounded, and can contain negative numbers or decimal places.

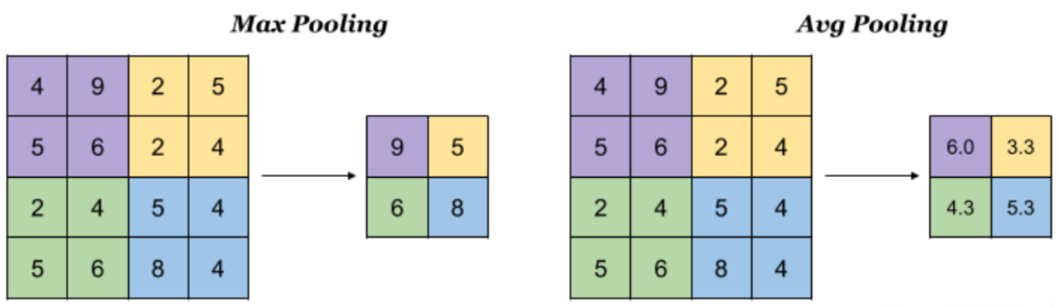


Figure 2.9: The workings of max and average pooling layer of a CNN [31].

2.5.5 Fully Connected Layer

The final stage of any CNN is the fully connected layer, this is responsible for flattening the image to a 1 column by many rows matrix. This one dimensional making of the input to the fully connected layer, is how the computer can read the features of the image in the best way [32].

2.5.6 Designing the CNN

To design the CNN correctly, there must be maths applied in order to work out the output of each convolutional layer, which helps maximise the layers of the network, which is especially useful when using a lower resolution. Equation 2.2 is used to calculate the output of any convolutional layer (input of the next layer).

$$output\ of\ convolutional\ layer = \left\lceil \frac{W - F + 2P}{s} \right\rceil + 1 \qquad (2.2)$$

Where W is the input image size, F is the filter (size of the kernel), P is the number of zero padding layers used and s is the **Stride**.

Using an input of 128x128 resolution, a kernel = 9x9, no zero padding and a **Stride** of 1, would give the output of this convolutional layer (input of the next layer) as:

$$output\ of\ convolutional\ layer = \left\lceil \frac{128 - 9 + 2(0)}{1} \right\rceil + 1$$

$$output\ of\ convolutional\ layer = 120$$

Then the next layer after this convolutional layer, conventionally would be a pooling layer, which is worked out by halving the input to its layer.

$$output\ of\ pooling\ layer = \frac{120}{2}$$

$$output\ of\ pooling\ layer = 60$$

The size of the output of the last max pooling cannot be any smaller than the desired output given to the fully connected layer, if the smallest size is reached, the model can be described as maximising its layers.

2.5.7 Dropout

Dropout layers are used to remove some of the previous layer's neurons. This effect is achieved by the computer randomly selecting neurons and then setting those neurons to a value of zero (effectively dropping them out of the network). The dropout effect is intended to prevent the model from becoming overly reliant on specific aspects in an image [33], so that the model path produced is robust enough to understand features of an image that it isn’t used to, during the testing phase. After the model has been trained, the dropout is no longer used.

2.5.8 Training and Validation Loss

These losses are used to determine if a model is underfitting or overfitting to the intended response. The training loss is measured after each batch, and the validation loss is measured after each epoch. Before the two losses are read, the validation loss is given extra gradient updates. As a result, if the validation loss is larger than the training loss, it is said to be overfitting, and if it is less than the training loss, it is said to be underfitting [9]. Overfitting is a bad thing, and if it happens, the entire model may need to be re-evaluated.

2.6 Problems of Hyperparameters

The main hyperparameters of the model are: batch size, number of epochs, optimisers, learning rate, momentum, activation functions, dropout, number of convolutional layers. There are more hyperparameters to look at as well, but adding more increases the complexity of the code [34]. When deciding on values for hyper parameters a combination of factors must be taken into account. The consensus from researchers around the globe is that there are currently no exact methods of optimising the hyper parameters. The selection of certain values often comes down to an onslaught of trial and error, which even the experts of the field frequently use as a method of solving to discover the optimal quantities. Many of the papers out there don’t agree with one another, and for this project it was hard to decipher what was correct information for the task at hand. This is due to there being many parameters which are all interchangeable, and the correction of one hyper parameter may not work for the others being considered [28]. Another factor when considering the hyper parameters of the models, is the application it is being used for. Certain hyper parameters may work for a specific application such as classification of the CIFAR-10 categories [35], although they may be obsolete for something such as keypoint detection of an asteroids centroid, which is ultimately the objective of this paper. To counteract the parameter issue, optimisers can be implemented, this allows the program to update its parameters as the training runs.

3.0 Methodology

To estimate the centroid position through deep learning, the following methods were used in Python, mainly via a sub category called PyTorch, which was in charge of all the modules used to design the CNN part of the code. The model used was a deep learning IP algorithm. The specific processing was centroid detection of the asteroid in the images. The training phase used an image dataset of 3,000 images and the validation phase used an image dataset of 700 images. This was found to be the dataset which could handle the task, in an optimal amount of time at the resize of 128x128 resolution. There were different python code files used to layout the code in a way that it was easy to alter parameters.

3.1 Codes

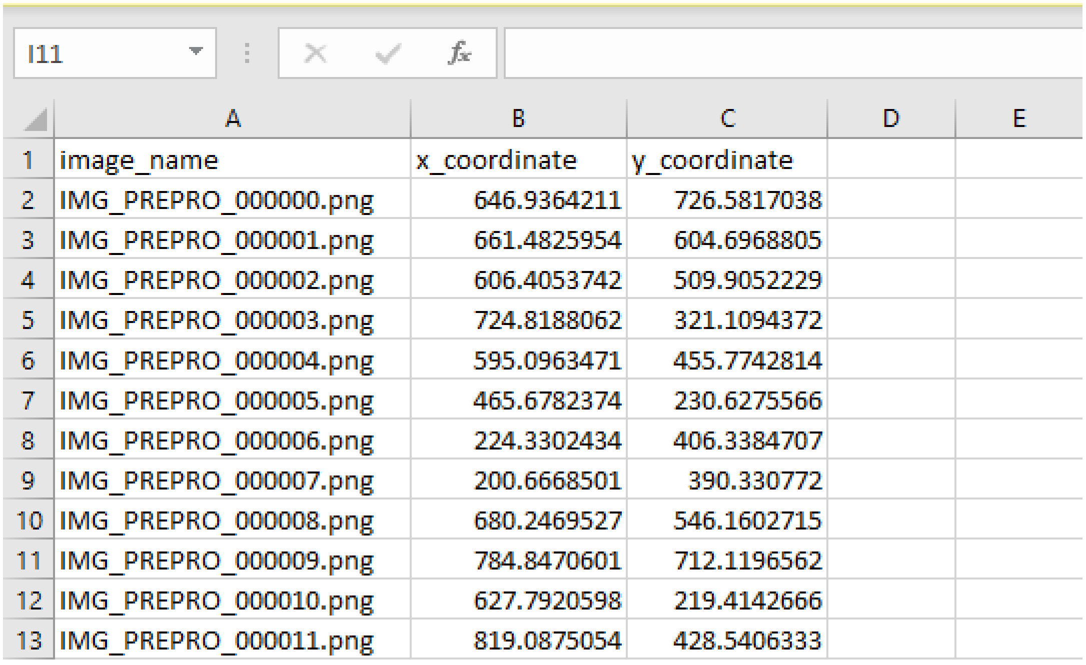
The CNN used in this project, was built using five different codes, similar to the structure of an IP algorithm used for facial keypoint detection [36]. NumPy arrays were used to construct certain parts of the CNN, and to turn images into an array of numbers [24]. A full account of all the codes mentioned in sections 3.1.1 – 3.1.5, can be found in the Appendices.

3.1.1 Config.py

This file was used to define some of the hyper parameters, which were batch size, learning rate and the number of epochs. It was also where the root path and output path of the model were defined, the root path being the location of all the important dataset files of the program, and the output path being the folder where the validation images, model path and loss graph are outputted to.

3.1.2 Dataset.py

Dataset files are used to load the images with their respective keypoint, in this case it was from the csv spreadsheets and the image files, which were located in the root_path. A class called “AsteroidDataset” was created, which has multiple steps within it. The first section of the class is the initialisation of the images, a module called Pandas was used to read the image names and centroid positions (x and y coordinates) in the csv file as shown in Figure 3.1 and convert them to a Pandas data frame, which can be read by Python.



I11

	A	B	C	D	E
1	image_name	x_coordinate	y_coordinate		
2	IMG_PREPRO_000000.png	646.9364211	726.5817038		
3	IMG_PREPRO_000001.png	661.4825954	604.6968805		
4	IMG_PREPRO_000002.png	606.4053742	509.9052229		
5	IMG_PREPRO_000003.png	724.8188062	321.1094372		
6	IMG_PREPRO_000004.png	595.0963471	455.7742814		
7	IMG_PREPRO_000005.png	465.6782374	230.6275566		
8	IMG_PREPRO_000006.png	224.3302434	406.3384707		
9	IMG_PREPRO_000007.png	200.6668501	390.330772		
10	IMG_PREPRO_000008.png	680.2469527	546.1602715		
11	IMG_PREPRO_000009.png	784.8470601	712.1196562		
12	IMG_PREPRO_000010.png	627.7920598	219.4142666		
13	IMG_PREPRO_000011.png	819.0875054	428.5406333		

Figure 3.1: The structure of the CSV files used for implementing the datasets to Python.

From this data frame, the root directory of the image file can be joined to the first column of the file i.e. the image_name column as shown above. This is done using another python module called OS, which is capable of joining the pandas data frame to the image file. Using “i” in the codes file locator means the image_names can be joined to the actual images, for the entire dataset, as “i” represents a column with an endless number of rows, so the code can run through each row, whilst simultaneously adding the image to the respective image_name row. The next module used is skimage, in specific the import is io, io.imread is used to take the located image and pass it to python, it is then appended to the self.images which is the set used to resize the image, change it to a grayscale format, and transpose the image [24]. The final stage of the program is to load the training and validation datasets, this data loader uses shuffle so that each time the training or validation is carried out, the order in which images are processed is different.

3.1.3 Utils.py

This code file was used to plot the predicted and actual keypoints onto the validation images, so that the user can see improvement in the losses after each set of epochs. The number of epochs before each validation image was pre-defined, using the mod operator. Utils.py was also used for showing a verification batch of 30 images, before the program begins running, which plotted a sample of the images so that the user can check the ground truth keypoints (in light blue) are plotting correctly on the images. Figure 3.2 shows the verification batch generated before each code ran.

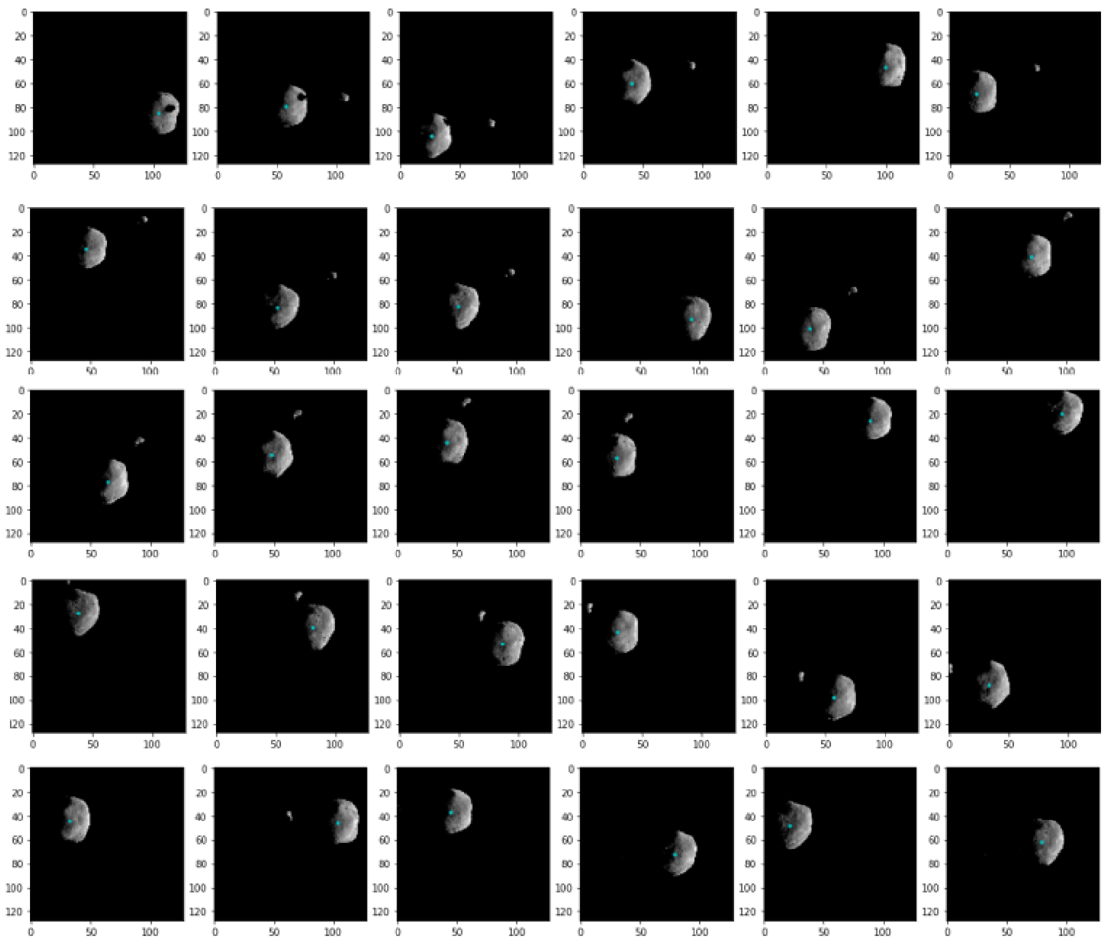


Figure 3.2: The verification batch which was plotted at the start of every training, used to check the ground truth of the images were plotted correctly (dots were plotted in light blue, as green was really difficult to see when plotted on 30 smaller images).

3.1.4 Model.py

The most important part of the CNN is the model, it is responsible for the actual processing of the images, using a combination of convolutional layers and max pooling layers. Padding could also be added to help the issue of downsampling. This program was used to experiment with different layering to find the best outcome. The model used was constructed by: having repetition of a convolutional layer to detect the main visual properties, then removing any irrelevant neurons with a ReLU activation function, the last repetition was a max pooling layer to amplify all the key attributes of the image. This layering structure was done several times over, to achieve accurate backpropagation. The final stage was reshaping the IP matrix using a fully connected layer i.e. many rows with a single column. Lastly two layers were used, a dropout layer to lose a percentage of the final neurons, and then a fully connected layer, which is used so that the program can read the output. No activation is used after the fully connected layer because without the activation, the program outputs the regressed coordinates of the keypoints, which is the desired outcome [36].

3.1.5 Train.py

This code also included some parameters of the model, the first thing inputted is the actual model itself from model.py, then from there the optimiser and loss criterion are initialised as well. The next thing is the training begins, firstly the number of batches are calculated and then the program optimises and calculates the losses after the first batch has ran, it then continues to calculate the losses after every batch. After a certain number of pre-defined epochs, the validation image is outputted via a function in the Train.py code. The final part of the code is responsible for plotting the loss graph after the last epoch and it also outputs the model path, which can be used for giving the optimised weights to the testing phase. The testing phase was not implemented in this project, it entails testing the pre-trained weights on a new dataset of asteroid images, in order to see how well the weights work on new images that have not been trained or validated with.

3.1.6 Flowchart of the Python code

To understand the way in which the code processes images, and how its inputs and outputs work, Figure 3.3 was created.

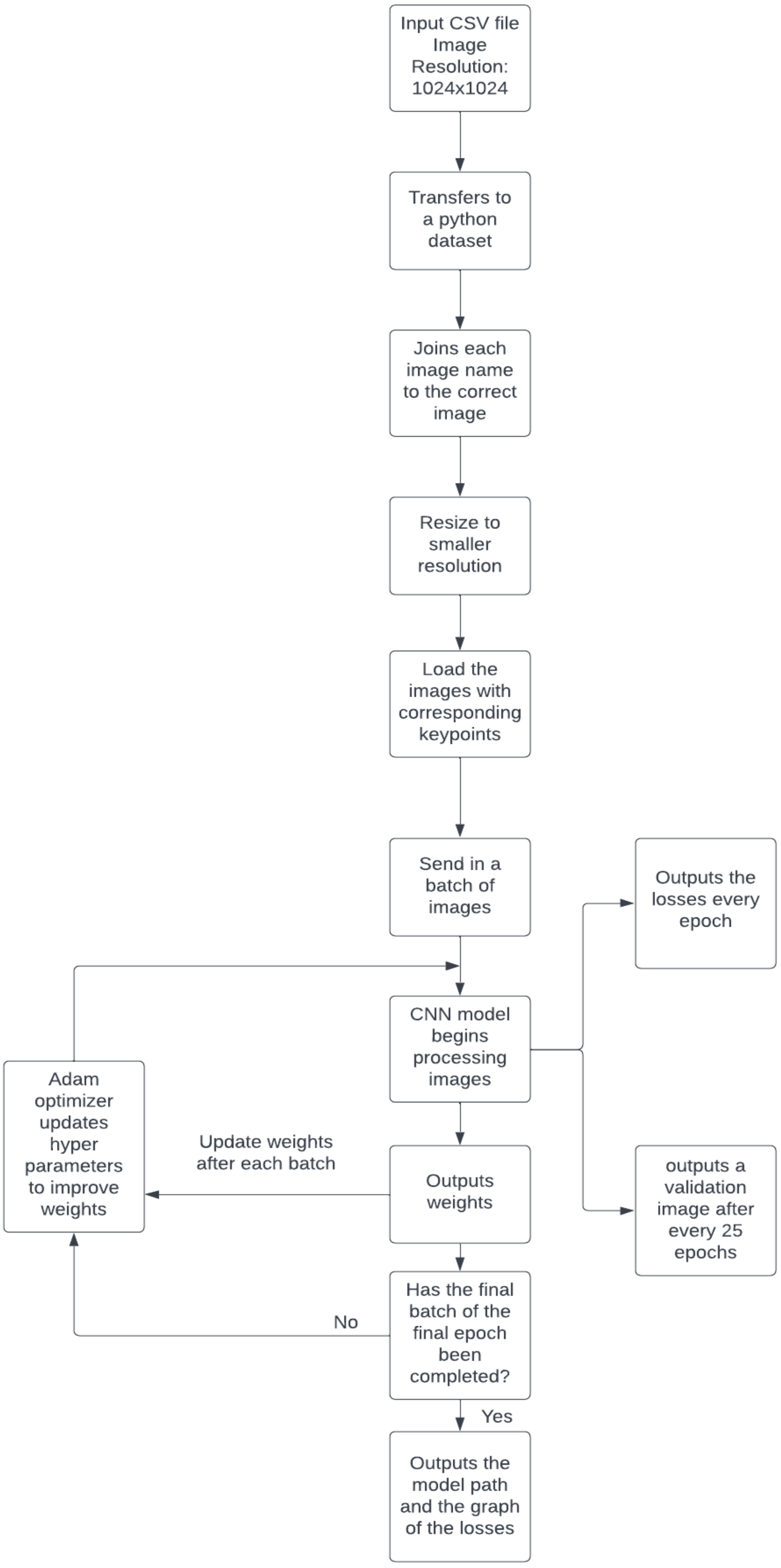

Input CSV file Image Resolution: 1024x1024
Transfers to a python dataset
Joins each image name to the correct image
Resize to smaller resolution
Load the images with corresponding keypoints
Send in a batch of images
CNN model begins processing images
Outputs the losses every epoch
outputs a validation image after every 25 epochs
Outputs weights
Update weights after each batch
Adam optimizer updates hyper parameters to improve weights
Has the final batch of the final epoch been completed?
No
Yes
Outputs the model path and the graph of the losses

Figure 3.3: Flowchart of the Python code performing the training and validation of the IP algorithm.

3.2 Batch Size

The batch size refers to how many images the algorithm reads at once. For many applications, there is an optimal batch size; the parameters impacting batch size include image resolution and the sheer volume of the image dataset. Using a dataset of 3,000 training images and 700 validation images, meant that an appropriate batch size was 32, considering the images were resized from 1024x1024 pixels, to 64x64 pixels or 128x128 pixels. Using an optimal batch size of 32 [37], and achieving the losses presented in this report, showed that the model was relatively stable, as the batch size is small in comparison to the dataset, yet it still yielded good training and validation losses.

3.3 Datasets

There are three types of datasets used in an IP algorithm, each type of dataset includes its own unique images. This project only considered the training and validation datasets, as the testing phase was not initiated. The use of the training data is to make the algorithm learn the different types of images it is given, these samples essentially create the model. This is the largest of the three datasets and it is important to have a large variation in images, so that the computer can understand any image given to it in the testing phase (not carried out in this project), and can plot estimates on it to an accurate degree.

The validation dataset is used during the training phase of the program, in this project it was used through a separate csv file. The validation dataset is used to show an estimate of what the test dataset will produce in terms of error. It is good practice to use validation, as it can show areas of improvement for the model during the training phase, which can be updated before moving on to the testing phase [38].

3.4 Mean Squared Error Loss

This error is in charge of showing how well the model has trained. Mean squared error loss (MSE) is the loss found by taking the difference between the models estimation and the ground truth, it then squares this loss and finds the average error across the whole dataset or one batch, depending on whether it is training loss or validation loss being calculated. For MSE the lower the loss the better. Advantages of using MSE is that any images with significant error will disrupt the overall average of the epoch or batch, because this method squares the errors [39]. This is a great loss criterion to use for centroid detection, as the application of the model is orbital navigation, so there cannot be large errors at any point, or the spacecraft may crash, or lose the field of view of the asteroid. This error can be given by the simple equation 3.1.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (d_i - p_i)^2 \tag{3.1}$$

Where N represents the number of images within the specified dataset, d_i is for the desired output and p_i is the predicted output.

3.5 Optimiser

Optimisers are used to accurately update the weights of the program after each batch has ran, in order to produce lower errors, which are calculated at the end of each epoch. The use of learning rate schedules, was proposed for this paper. They are used to decrease the learning rate during training in accordance with a pre-defined schedule. Although as the other hyper parameters are fixed from the start, the learning rates being reduced during training, can end up being detrimental to the loss of the model [40]. Setting the learning rate too high can cause fluctuation in the loss graph, and using a low learning rate, means the program takes a long duration to reach the point of convergence [41].

Considering all the points mentioned in section 2.6, the method used in the end was an Adam optimiser. The use of this optimiser is to update the learning rate to improve the prediction, which is through a learning rate schedule as described above. The other use of an Adam optimiser is to update the parameters, which are directly estimated using the averages of the first and second moment of the loss curves gradient [42].

3.6 6 layer CNN Model

The structure used was a 6 convolutional layer set up the following way, as shown in Figure 3.4

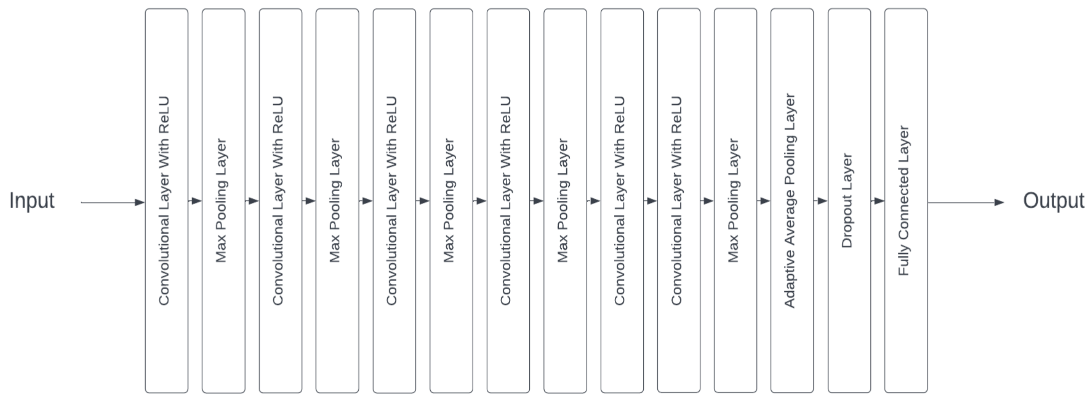

Input
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Max Pooling Layer
Convolutional Layer With ReLU
Convolutional Layer With ReLU
Max Pooling Layer
Adaptive Average Pooling Layer
Dropout Layer
Fully Connected Layer
Output

Figure 3.4: The final CNN used for the IP algorithm.

The layers of this network were maximised, which meant the last max pooling layer gave the smallest possible output the program allows. This output was found by using the equations described in section 2.5.6. Having two convolutional layers one after the other as shown in Figure 3.4, allowed the program to build up a better understanding of the image. Using two convolutional layers consecutively, allowed for more layering, as using a max pooling after each convolutional layer, meant that the input was reduced by a factor of two, eventually resulting in a maximum allowance of 5 layers when using a max pooling after each convolutional layer. Zero padding was also implemented on convolutional layers 2 and 5, to pass 1 layer of zeros around the input to help keep the inputs and outputs of each layer divisible by two, which is important for the final output. The convolutional layering used for the final CNN as demonstrated in Figure 3.4 is shown in depth using Figure 3.5.

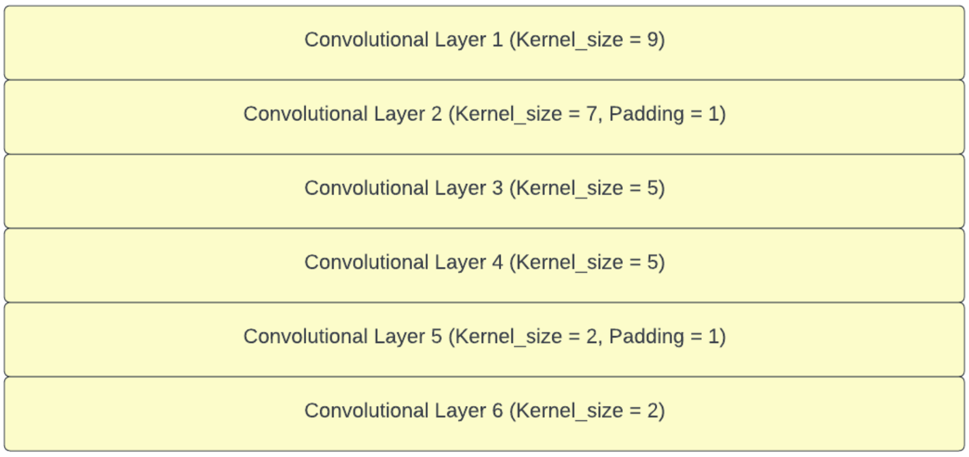

Convolutional Layer 1 (Kernel_size = 9)
Convolutional Layer 2 (Kernel_size = 7, Padding = 1)
Convolutional Layer 3 (Kernel_size = 5)
Convolutional Layer 4 (Kernel_size = 5)
Convolutional Layer 5 (Kernel_size = 2, Padding = 1)
Convolutional Layer 6 (Kernel_size = 2)

Figure 3.5: The convolutional layering used for the final CNN.

The dropout used for the network was a probability of dropout equal to 17.5%. Originally, 20% was used, but it was discovered that a slight change down to 17.5% yielded slightly better changes in the loss after each epoch. All of the slight changes added up over time, resulting in a significantly smaller loss overall.

3.7 CUDA

Processing images through the GPU was considered for this phase of the project, using Compute Unified Device Architecture (CUDA), which is an installable module for IP algorithms, which runs the processing of images through the GPU [43]. It proved difficult to understand which version of CUDA was to be used for this specific area of image-processing, and access to it is not simple. A secondary computer was used with CUDA running on it, but it was discovered that the laptop was executing tasks quicker, even without CUDA installed. It is possible to install CUDA for the GPU on the laptop carrying out the simulations, but it involves editing the build of PyTorch, which is difficult and time consuming. Entire simulations were taking 6-12 hours, and in smaller resizes such as 64x64, less than 2 hours, so CUDA was discarded in the end.

4.0 Results

4.1 Resizing

This section looks at the effects of resizing the images in the datasets. It specifically looks at how resizing changes the spatial position of the predicted points shown in the validation images. All of the images start with the reference coordinates at the top left corner of the image, shown in Figure 4.1, which represents the validation image produced after the program ran for 100 epochs, using a resolution of 1024x1024 pixels.

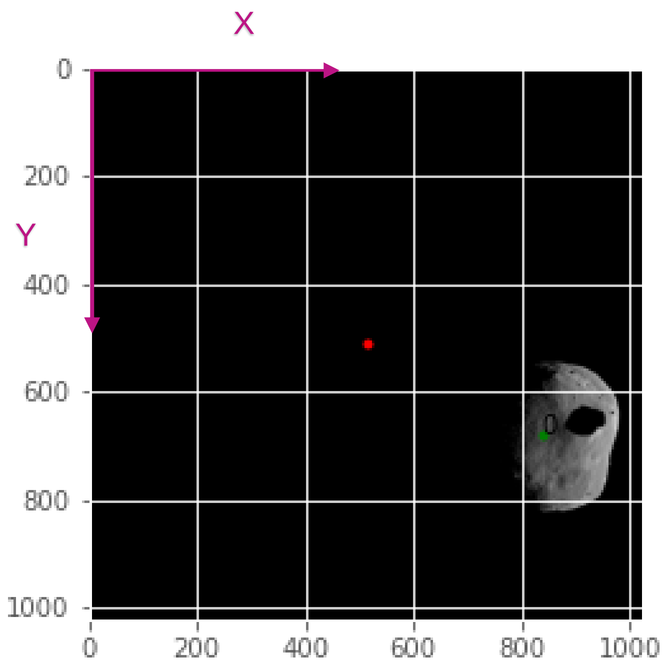


Figure 4.1: The validation image of Didymain at image resolution 1024x1024, after 100 epochs.

As can be observed, the IP algorithm's predicted keypoint is shown as the red dot, and the ground truth is shown in green. The “0” just above the green dot in the illustration, indicates that the given point is the first, and further might be added. If the program were to be used for feature tracking, additional points that were being detected, would be labelled 1,2,3, and so on. The number label is given in black so that it doesn’t make the ground truth difficult to see.

Figure 4.2 shows how the resizing of the image helps the IP algorithm to gain better results. The 1st quadrant shows the resize at 400x400 pixels after 100 epochs, the 2nd shows the resize at 128x128 after 100 epochs, the 3rd shows the resize at 96x96 after 100 epochs and the 4th shows the resize at 64x64 after 100 epochs.

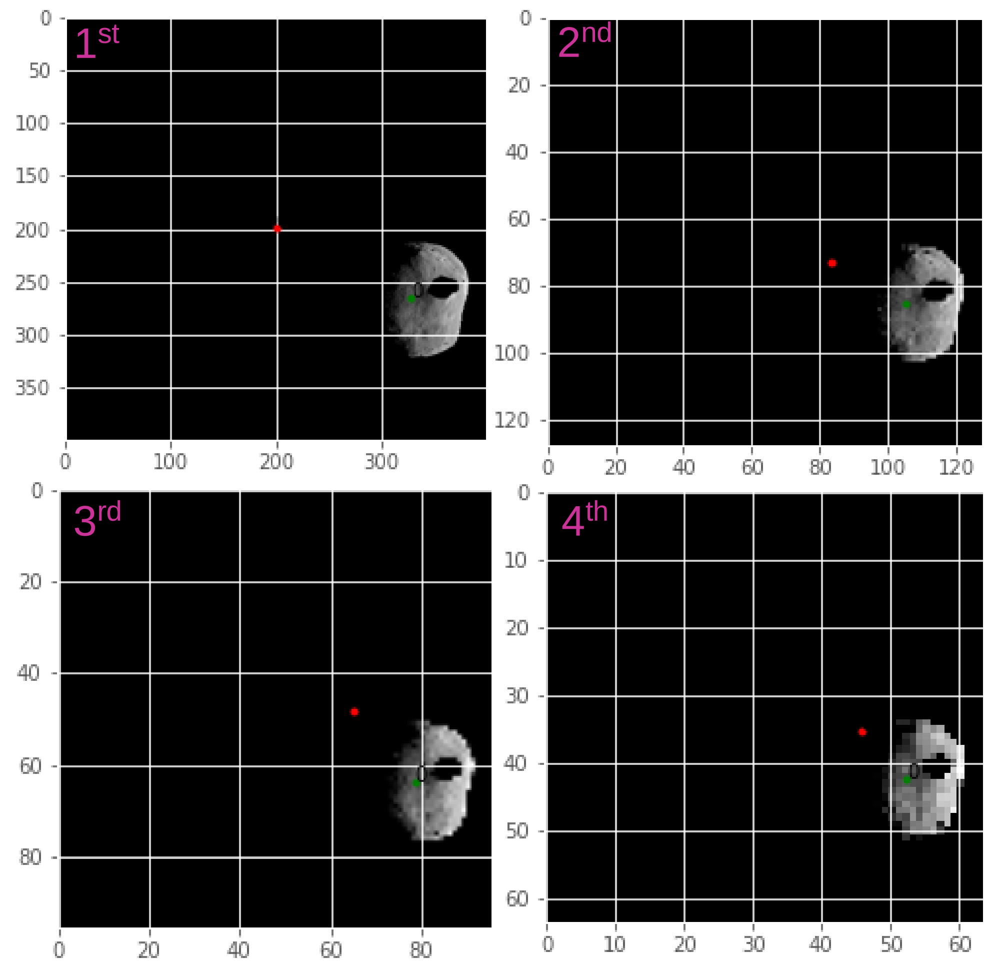


Figure 4.2: The validation images of Didymain at image resolution 400x400, 128x128, 96x96 and 64x64, respectively. The Images are showing the validations produced after 100 epochs.

4.2 Optimisation

The next area looked at was optimising the hyper parameters to gain the best results, a mixture of trial and error, and methodical approaches were used. Figure 4.3 shows the effect on loss when giving the program 400 epochs (iterations) to run.

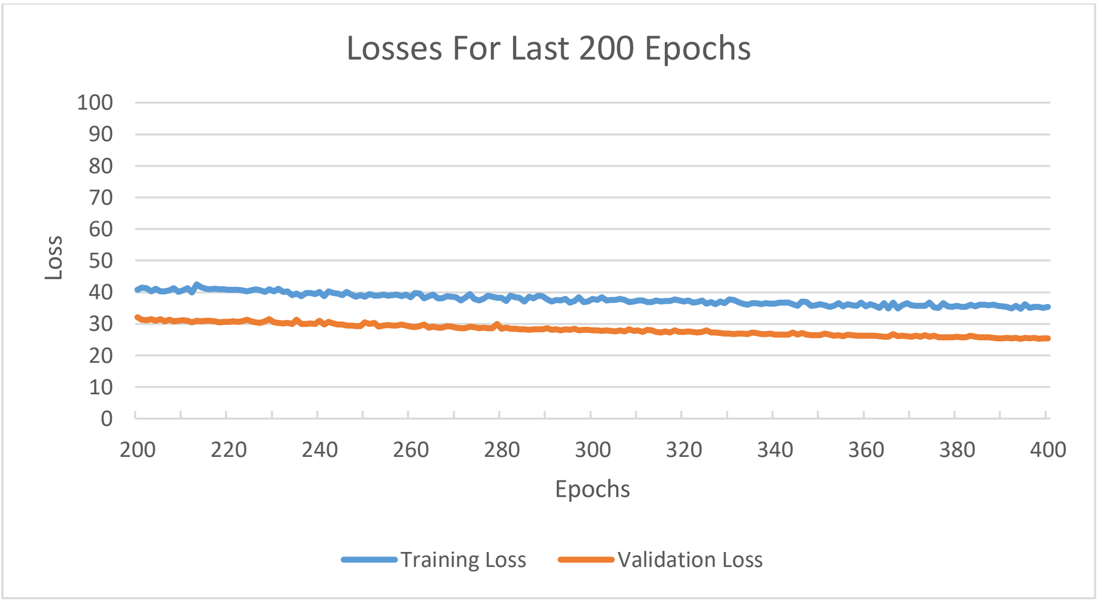


Figure 4.3: The loss graph for the last 200 epochs of a 400 epoch simulation. For a 64x64 resize, using the standard parameters.

Figure 4.3 shows the graph looks like the continuing trend will be to drop in loss, but it doesn't change significantly in loss past 400 epochs.

The graph shown in Figure 4.4 shows the 1200 epochs for the 6 CNN model. Figure 4.5 shows the graph produced for a learning rate of 0.001, and Figure 4.6 shows the validation image after 200 epochs.

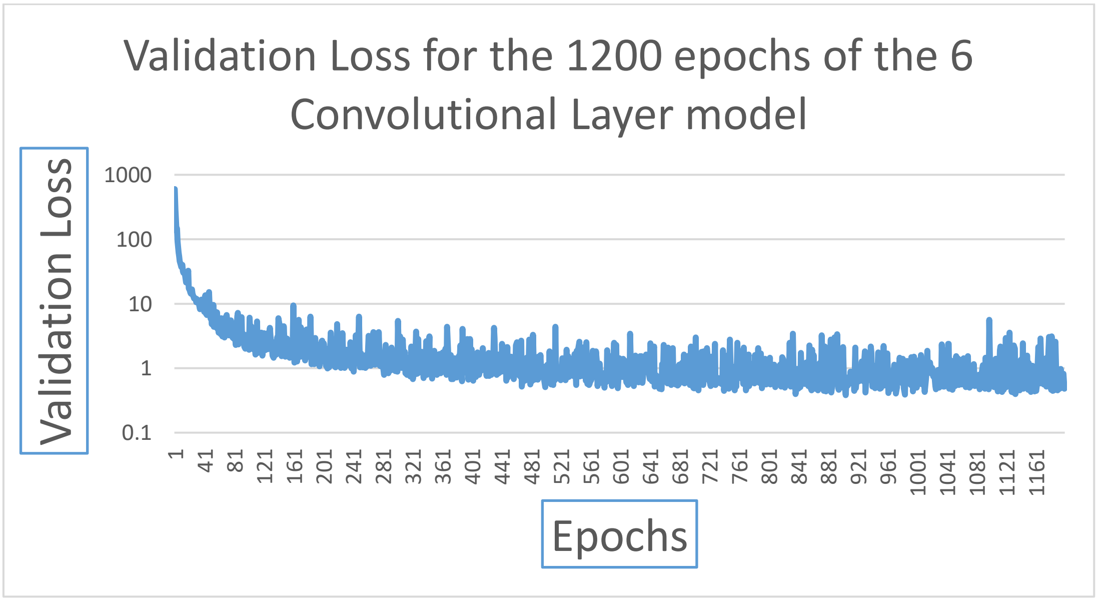


Figure 4.4: A Graph showing the change in validation loss, for an image resize of 128x128, ran for a 1200 epoch simulation of the 6 convolutional layer model.

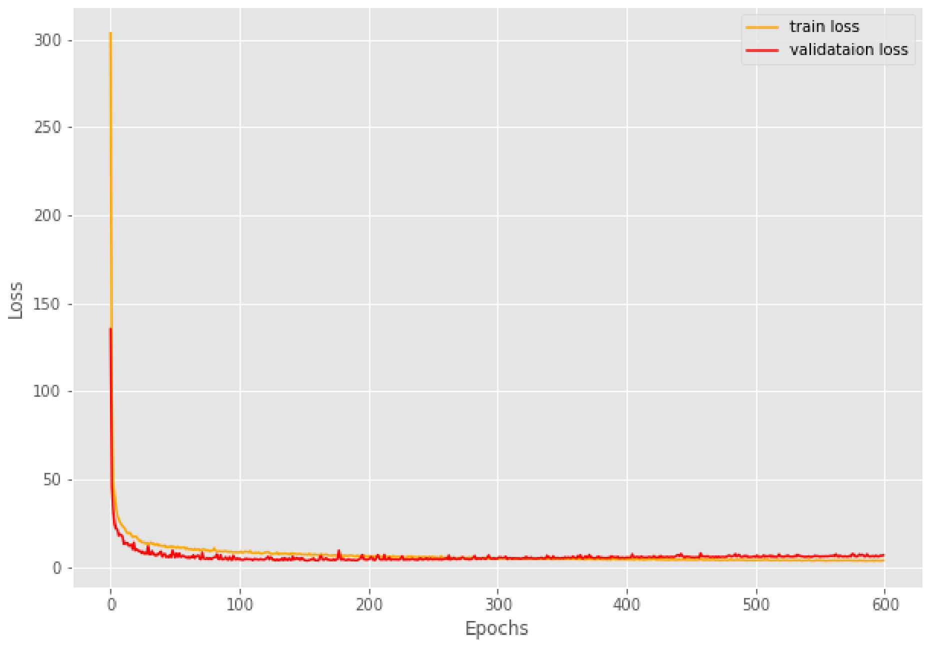


Figure 4.5: The Loss graph produced by the program, for 600 epochs, of a 64x64 image, using a learning rate of 0.001.

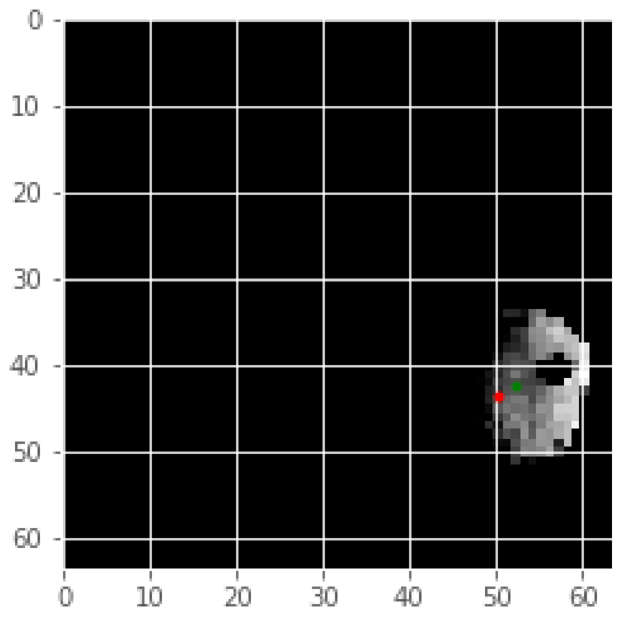


Figure 4.6: The validation image of Didymain at image resolution 64x64, after 200 epochs, using a learning rate of 0.001 and 4 convolutional layers.

Figure 4.7 is showing the image at resolution 64x64, which achieved loss convergence and only a small amount of underfitting.

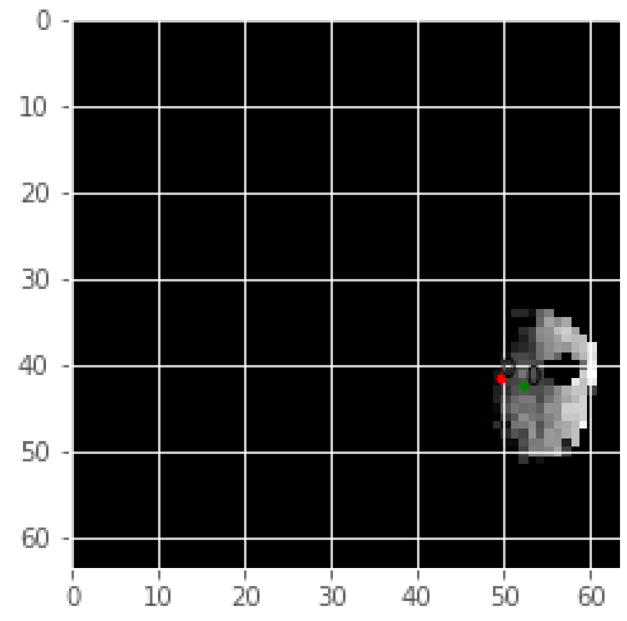


Figure 4.7: The Validation image of Didymain at 64x64 resolution, with dropout probability 17.5% and 4 convolutional layers, after 400 epochs.

Figure 4.8 shows the result of a 5 convolutional layer model, using a resolution of 128x128 pixels.

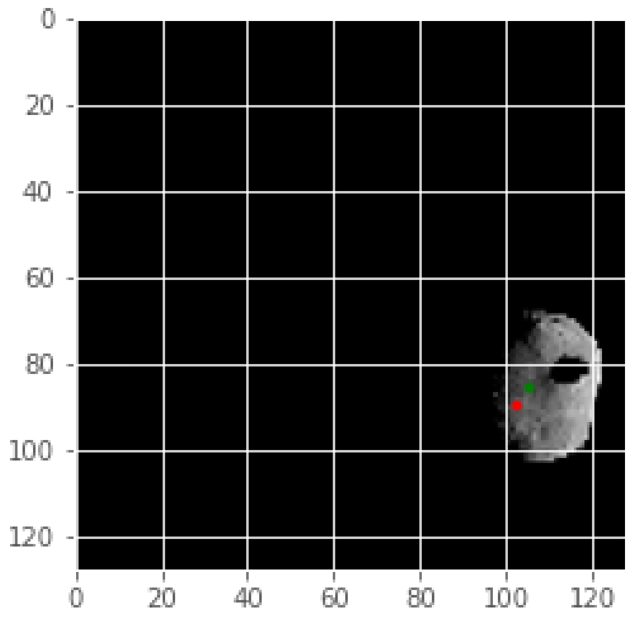


Figure 4.8: The validation image of Didymain at 128x128 resolution, with dropout probability 17.5% and 5 convolutional layers, after 400 epochs.

Figure 4.9 shows epoch 725 for a CNN with 6 convolutional layers, this yielded the best result, and would be more than suitable to use for testing. The red dot is barely visible in this image, because the prediction is very accurate.

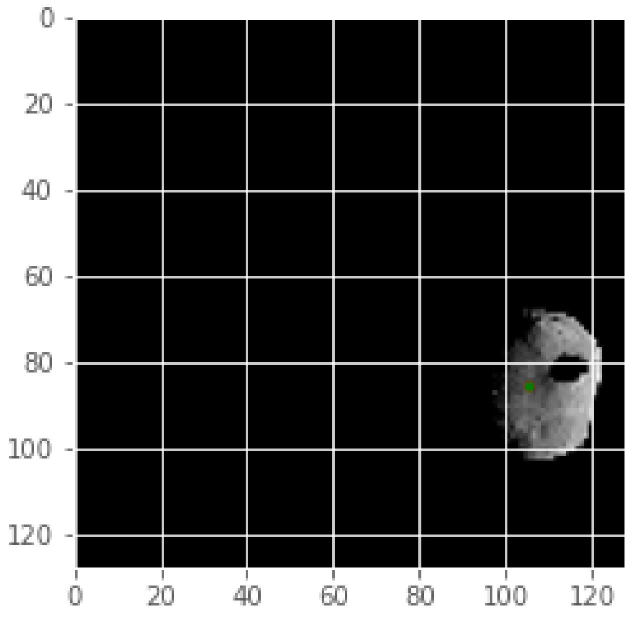


Figure 4.9: The validation image of Didymain produced using a 6 layer CNN mode, which was the best model, in terms of performance, used in this project.

Similar validation images were produced using the same model, these shown in Figure 4.10. Again some of the red dots are barely visible, as the prediction is very good.

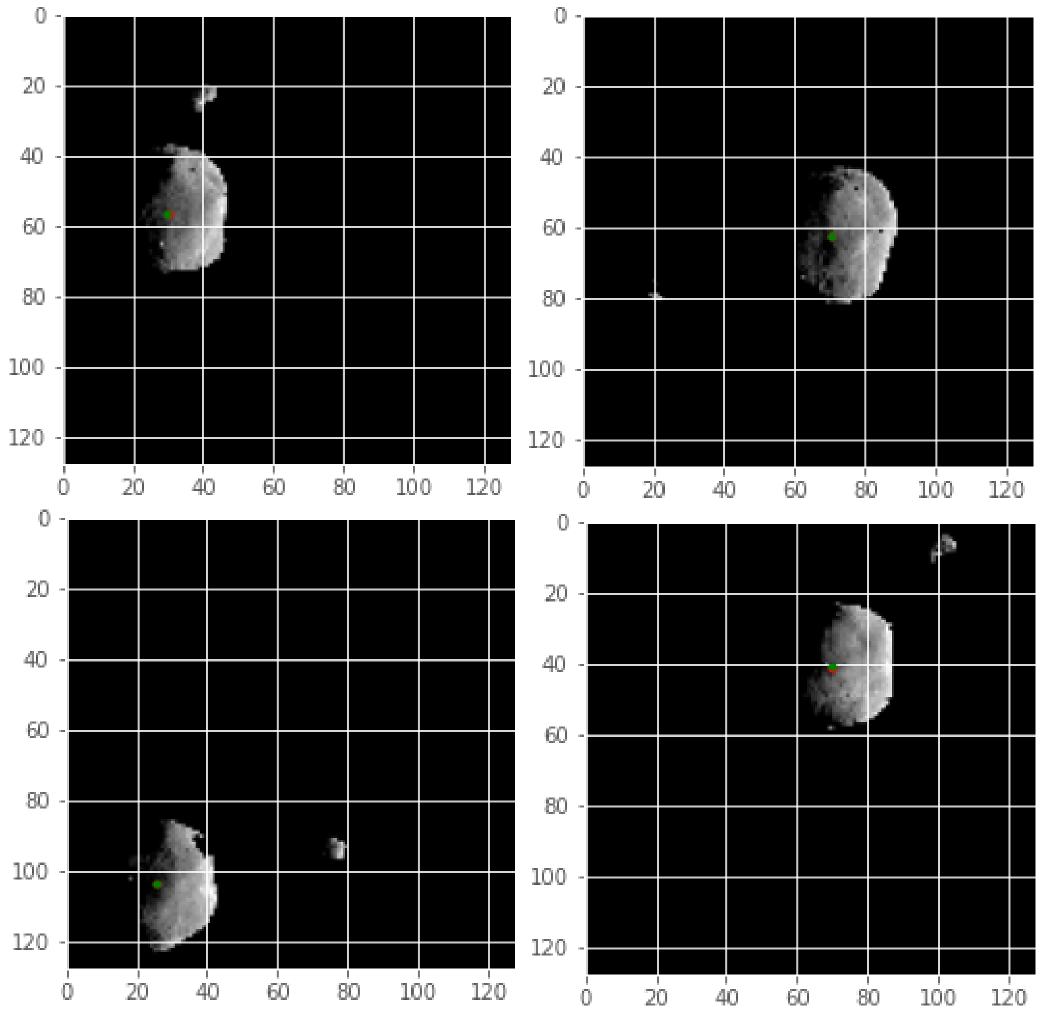


Figure 4.10: More validation images produced by the 6 layer CNN model, for various positions of the asteroid.

5.0 Discussion

5.1 Approaches Used to Improve Parameters

To better outline the approaches used to improve the programs results, Table 5.1 shows how certain areas were analysed.

Table 5.1: This table outlines clearly where different research approaches were used.

Parameter	**Approach Used**	**Best value**
Resizing	Methodical Approach	128x128
Batch size	Trial and Error	32 images
Learning Rate	Trial and Error	0.0001
Dropout	Methodical Approach	0.175 (17.5%) dropout probability

As can be seen above, the approach used for the resizing, was to slowly decrease the image resolution, to see at what point the characteristic features of the asteroid were lost. It was also important to see what resizing produced 100 epochs in a run time of approximately one hour. Initially the batch size was changed through trial and error, and then the batch size of 32 was found to give the best result. After researching online, the research papers also agreed with a batch size of 32 for certain IP applications. The Learning rate used in [36] was found to give the optimal result, so after changing the learning rate multiple times randomly, none of the results produced anything better than the initial 0.0001, this could be down to the Adam optimiser being used. Finally, the dropout used in the network was seen to be too high for the IP application being used, so it was slowly dropped down by half a percent at a time, originally starting at 20%. Taking the dropout further down to 17% showed a higher loss, and other smaller dropouts were an even higher loss. Therefore, it was best to use 17.5%. The parameters used were great in terms of results produced, but change of one parameter can affect the other, so it is inconclusive which parameter values work the best, and scientists have been trying to work out a purely methodical approach for years.

5.2 Resizing and Convolutional Layering

When using the size of the original image i.e. 1024x1024, the programs loss was very large, as shown in Figure 4.1. After researching why, it became clear that to use such a large resolution, requires a very complexed model. This large resolution would require possibly 12-15 convolutional layers, and the addition of more hyperparameters. It would be of a high complexity, beyond the scope of this work and would require lots of research to gain accurate results. Even if the convolutional layering was correct, the batch sizing may need an extensive amount of research to correct, or the learning rate may need more research, so this size of resolution was disregarded. Using the “0” beside the keypoint was useful for this specific IP algorithm, if it were to be used for feature tracking points of the asteroid, instead of centroid detection, it could be easy to label certain features of the asteroid using the numbering technique.

The number of layers that are allowed to be used depends on the size of the resolution. For the 128x128 resize, the final CNN’s layers used were maxed out at 6 convolutional layers. This is due to each time max pooling is used, the input is reduced by a factor of 2, so there cannot be an infinite number of layers, as explained in section 2.5.4. Furthermore, padding was used to increase the number of allowable convolutions, but excessive use of padding was avoided, as it can negatively affect the IP algorithm.

At an image size of 1024x1024 as shown in Figure 4.1, the program struggled to compute 100 epochs of the dataset. To run the IP algorithm at this resolution, it required a duration of 50 hours. This is due to such a large density of pixels and limiting software capabilities. Eventual resizing down to 64x64 as shown in the 4th quadrant of Figure 4.2, gave a result significantly more accurate than the 1024x1024 image. This can be due to the AI being able to recognise the asteroid better, as there are less pixels for it to analyse at such a low resolution. The downside of this is that the asteroid loses some of its characteristics, such as its shape and textures. Although the loss scale is shown to be changing in Figure 4.2, a loss of 10 at 100x100 and a loss of 100 at 1000x1000 doesn’t mean that the loss is 10 times smaller at 100x100, it doesn’t linearly scale. Having said that, the overall trend showed that the smaller resizes gave a better loss. The image resize at 128x128 shown in the 2nd quadrant of Figure 4.2, shows that at 128x128, the resize resolution is still good enough to characterise the asteroid. The other great attribute of this quadrant, is the fact that the program could run the IP algorithm for 100 epochs in 58 minutes, it was decided that 128x128 was the best resize to use for the project, 96x96 was too low a resolution to be considered.

5.3 The Selection of Validation Images

For the preliminary results of this project, the validation image used in the simulations, were carefully selected. The characteristics of the validation image used were, a full representation of Didymain and a shadow cast over it of the moonlet Dimorphos. This was used as it gave a good idea of how the program ran on images with slight distortion, such as shadows. Using an image that was clear for the program to read would not have given any indication of how well the program runs on the less asteroid characterised images. Factors that made it difficult to read were: it was to the side of the image, it had a shadow on it, it was also lower down on the image. In the final CNN used, there were multiple validation images produced, to see how robust the capabilities of the program was.

5.4 Optimising The Hyperparameters

The next step in optimisation was defining the correct number of epochs, where one epoch in this case equates to all the images in the validation and training datasets being iterated through. There was minimal change in loss after 400 epochs for the 3 convolutional layer network, at 64x64, as explained in Figure 4.3. There was minimal change after 600 epochs for the 4 convolutional layer network at 96x96. It is still hard to decide whether the convolutional layering is affecting the number of epochs, so the number of epochs to use was determined through setting the IP algorithm to run for a large number of epochs. For the chosen resize of 128x128, 600 epochs was best for 5 convolutional layers and 800 epochs was optimal for 6 convolutional layers. Displayed in Figure 4.4 is the 1200 epochs of the 6 convolutional layer model for the image resize of 128x128, the curve seems to converge after 500 epochs, but the best validation images and losses were produced after 700 epochs, with some losses reaching as low as 0.5. Anything after 800 epochs showed very little change overall. The large losses sometimes calculated after certain epochs, were between 2-5 in magnitude. This phenomenon can be down to the loss criterion used, as explained in section 3.4.

After this, the next step was altering the learning rate, the general way of changing it is by one decimal point position. Shown in Figure 4.5, a learning rate of 0.001 never converges close to the point of zero loss, it always fluctuates to a high loss and then a low loss. This is because the learning rate is too large, so when it calculates the loss gradient, it will go above the lowest point and then try to reduce down smaller, essentially over compensating making the loss go very low. Another explanation for this fluctuation is that using the Adam optimiser, which updates the learning rate after each batch, needs a smaller learning rate to initialise with, for this specific IP algorithm to not fluctuate in loss. The fluctuation continues and the loss values begin to increase as well, this IP algorithm may take years to reach convergence using this initial learning rate, or it may never converge. So although this produced great results after 200 epochs, as shown in Figure 4.6, the overall trend was that it fluctuated high and low in error, never reaching convergence. Showing that it only will train and validate well on certain image inputs of the CNN. Lack of convergence, and changing trends of the graph, show it is not a suitable model path.

The miniscule change in dropout gained better results than before, using a probability of 17.5% showed a slight decrease in validation loss, and the resulting validation images proved this to be correct, as shown in Figure 4.7. This shows that the network was losing too many important characteristics of the image at 20% dropout probability with a learning rate of 0.0001, as learning rate and dropout are correlated. The model ran with 5 convolutional layers as shown in Figure 4.8, where the validation image of epoch 400 can be seen, showed similar results to the 64x64 images. As the image is of higher resolution, it is expected that there is a higher loss, although relative to the validation image, the results are similar to the ones seen in Figure 4.7. The benefit of the upscaling of the image resolution is being able to keep more of the characteristics of the asteroids shape, and at this resolution there could be an edition of another convolutional layer as the resolution was large enough to handle it. The downside is the computational time at 128x128 resizing. It is still inconclusive whether the loss is dramatically changed by the resolution going from 64x64 to 128x128. The hyperparameters were optimised for the 64x64 image, and the optimal numbers found for learning rate and dropout, didn’t change for the larger resolution of 128x128. This potentially means that for this specific increase in resolution (64x64 to 128x128), does not affect the dropout and learning rate of the network.

5.5 The Final CNN

For the original validation image of the 6 convolutional layer CNN, as shown in Figure 4.9, the losses were very small (<0.6). This can be due to the changing of kernel size. For the final model, there was a large 9x9 kernel in the beginning and eventually two 2x2 kernels at the end, as explained in section 3.6. This meant that the program could read the asteroid very well. This could be down to the program needing to understand a large portion of the asteroid in order to be able to accurately predict the centroid for each image given to it. As a large portion of the image is black, using the 9x9 kernel was important to make sure that the areas of the asteroid could be characterised properly by the program. Before this layering structure, the program was reading too small a section each time, when characterising the asteroid with the first two convolutional layers. Enlarging the area being read at each **Stride** of the kernel, meant that the program could accurately predict the centroid. The final images, shown in Figure 4.10, are proof that the model is working on different asteroid positions. This gives credit to the robustness of the program, and shows it could be used to test on images of all different asteroid orientations and spatial positions.

6.0 Conclusions and Future Work

6.1 Conclusions

For the time frames given to learn and produce such complexed code, the results are of a high standard. That is considering that researchers who spend years creating machine learning models, still use an extensive amount of trial and error to gain optimal results.

The final model used in this project, was the 6 convolutional layer model, results produced were of a very high standard. The validation images show that the model was learning well on all the images given to it. This is true, because the validation was done for a range of different images, and the validation losses and training losses were relatively consistent. It is also important to note that the validation images for the final CNN used, were images that had the asteroid at very different areas of the image, showing that the program was working for all the different spatial positions and orientations. Slight variations in the loss were only due to the program finding it difficult to detect the centroid on a few images, out of the hundreds it was given, which can be explained from the loss criterion used, as detailed in section 3.4.

During this project it was discovered that the hyperparameters values that were optimised at lower resolutions, could be used with the same values for the higher resolutions. Not only did this reduce the simulation time for optimisation, but it also meant that only slight refinements had to be made at the larger resizes, which reduced the complexity of the model design at these larger resolutions. The use of an optimiser aided in reducing the complexity of improving hyperparameters, as they were changed automatically during training, and using online sources helped to understand how to use the optimiser in the most efficient way.

Creating a basic deep neural network from scratch, pales in comparison to some of the well-known CNN's, such as the High Resolution Network (HR Net) [44]. Optimisation of an IP algorithm for use on an asteroid dataset has been a difficult task, considering the software limitations and time constraints. Another important thing to note is that the simulations were ran from home on a laptop, so the larger resolution images (1024x1024 and 400x400) took durations of up to 50 hours to complete their training. On resizing to 128x128, with 6 convolutional layers and various other parameter changes, it can be seen that the model works well when being upscaled in the number of convolutional layers used in comparison to the smaller resizes. It was vital to find the correct resize that matched an optimal number of convolutional layers, after extensive research, this was achieved. The results show that for a well-constructed prediction model with 6 convolutional layers, centroid estimation of Didymain is achievable.

6.2 Future Work

As a result of computational limitations, the simulations ran for this project were of a reduced resolution. In the future, an improvement made would be using better resolutions. Such as a resize at 1024x1024 pixels, although this would induce a need for larger and significantly more complexed CNN models. The second consideration would be to initiate the testing phase, as the time span given for this project limited the amount of work that could be achieved, hence why the testing phase never came to fruition. Undertaking the testing phase would give an understanding of how well the program acts on new data, and gives another insight on how well the original training and validation datasets fitted to the model. Thirdly, after determining the cause of slight underfitting of the model, the hyper parameters could be fine-tuned to a larger degree to provide extra optimisation of the current results. One of the changes which could be made to improve the program, is to plot more validation images during each training phase, this would allow multiple comparisons to be drawn between different images, to better understand which asteroid positions and orientations the model is struggling to characterise.

Fine tuning of the model would gain better convergence as it still fluctuates within a tolerance of 1.5-0.5 for most epochs, this is considered reasonable, but by no means optimal. Improving the model can also entail reducing the time for simulations, as this factor hasn't been dealt with in the framework of this project. As explained for other processes detailed in this project, the basic time reduction implementations are done by substantial trial and error, with only a smattering of methodical approaches. For example, as stated in section 3.7 adding CUDA software into the Python scripts through the custom installation of the PyTorch module, or by optimising the learning rates via a different optimiser, could allow for better results. Many other parameters can be adjusted, such as the pooling layers, or the optimiser used in the training of the program. The different hyperparameters that can be manipulated by the optimiser, could be looked into, to further improve the learning process of the model.

This dataset of asteroid images could be used on an open source CNN model, such as the High Resolution Network (HR Net), which would provide multiple methods of further optimisation [44]. Another important consideration in future work would be real life testing, this model could be used by setting up a camera with the software, and taking thousands of pictures of a prototype asteroid, to see how well it works when being implemented onto an actual camera. In order to do this, a method of implementing the code onto a camera would have to be achieved, this could be done by alteration of a real time facial recognition system [45]. This project acts as a good foundation for further research and refinement of the IP algorithm presented.

7.0 References

[1] J. McGonagle, G. Shalkouski and C. Williams, “Backpropagation,” Brilliant, [Online]. Available: https://brilliant.org/wiki/backpropagation/. [Accessed 16 April 2022].

[2] A. Krizhevsky, “The CIFAR-10 dataset,” cs.Toronto, September 2017. [Online]. Available: https://www.cs.toronto.edu/~kriz/index.html. [Accessed 14 April 2022].

[3] D. Dumitrescu and C.-A. Boiangiu, “A Study of Image Upsampling and Downsampling Filters,” *computers,* vol. 8, no. 2, 2019.

[4] K. Foote, “A Brief History of Deep Learning,” *Data Topics,* 4 February 2022.

[5] W. McCulloch and W. Pitts, “A Logical Calculus of the Ideas Immanent in Nervous Activity,” vol. 5, pp. 115-133, 1943.

[6] A. Turing, “On Computable Numbers,” *Proceedings of the London Mathematical Society,* Vols. s2-42, no. 1, pp. 230-265, 1937.

[7] K. Fukushima, “Neocognitron: A Self-organizing Neural Network Model,” *Biological Cybernetics,* vol. 36, no. 4, pp. 193-202, 1980.

[8] R. Lea, “What to Know About Asteroids Dimorphos, Didymos That NASA Will Smash Into,” *Tech & Science,* 24 November 2021.

[9] “Asteroid 65803 Didymos (1996 GT),” The Sky Live, [Online]. Available: https://theskylive.com/didymos-info. [Accessed 12 04 2022].

[10] J. Gil-Fernandez and G. Ortega-Hernando, “Autonomous Vision-based Navigation for Proximity Operations around Binary Asteroids,” *CEAS Space Journal,* vol. 10, no. 4, 2018.

[11] J. Mendoza, “Fridge-sized asteroid hit Earth 2 hours after it was first spotted, NASA says,” *USA Today,* 19 March 2022.

[12] “Technology Demonstration,” European Space Agency, [Online]. Available: https://www.esa.int/Safety_Security/Hera/Technology_demonstration2. [Accessed 10 04 2022].

[13] L. Herridge, “NASA’s DART Spacecraft Traveling on its Own,” *Double Asteroid Redirection Test (DART) Mission,* 24 November 2021.

[14] L. Herridge, “NASA, SpaceX Launch DART: First Planetary Defense Test Mission,” *Double Asteroid Redirection Test (DART) Mission,* 24 November 2021.

[15] A. Barnett, “Didymos,” *Solar System Exploration,* 29 November 2021.

[16] “Name given to asteroid target of ESA’s planetary defence mission,” *HERA,* 23 June 2020.

[17] “CubeSats joining Hera mission to asteroid system,” ESA, 7 January 2019. [Online]. Available: https://www.esa.int/Safety_Security/Hera/CubeSats_joining_Hera_mission_to_asteroid_system. [Accessed 22 March 2022].

[18] A. Pellacani, M. Graziano, M. Fittock, J. Gil-Fernandez and I. Carnelli, “HERA vision based GNC and autonomy,” in *8th EUCASS association Conference*, Madrid, 2019.

[19] A. Kaluthantrige, J. Feng, J. Gil-Fernandez and A. Pellacani, “CENTROIDING TECHNIQUE USING MACHINE LEARNING ALGORITHM FOR SPACE OPTICAL NAVIGATION,” in *3rd IAA Conference on Space Situational Awareness (ICSSA)*, Madrid, 2022.

[20] I. Tzinis, “Technology Readiness Level,” *NASA: Technology,* 1 April 2021.

[21] B. Mckinney, “How Spacecraft Are Designed: Balancing Requirements With Reality,” *aerospace|engineering,* 23 September 2020.

[22] “Hera spacecraft self-driving navigation test,” 5 April 2019. [Online]. Available: https://www.esa.int/ESA_Multimedia/Videos/2019/04/Hera_spacecraft_self-driving_navigation_test/(lang)/en. [Accessed 10 April 2022].

[23] J. Heras, “2018-MachineLearning-Lectures-ESA/ Deep Learning,” 26 June 2018. [Online]. Available: https://github.com/jmartinezheras/2018-MachineLearning-Lectures-ESA/blob/master/4_NN-DeepLearning/4_NN-DeepLearning.pdf. [Accessed 14 November 2021].

[24] “Coursera,” [Online]. Available: https://www.coursera.org/learn/deep-neural-networks-with-pytorch#syllabus. [Accessed 12 Nov 2021].

[25] C. Wang, M. Peng, T. Chen, G. Liu and X. Fu, “Dual Temporal Scale Convolutional Neural Network for Micro-Expression Recognition,” *Frontiers in Psychology,* vol. 8, 2017.

[26] J. Wu, “Introduction to Convolutional Neural Networks,” Wenzhong Li at Nanjing University, Nanjing, 2017.

[27] “Drawing a Convolution,” StackExchange, February 2021. [Online]. Available: https://tex.stackexchange.com/questions/437007/drawing-a-convolution-with-tikz. [Accessed 24 March 2022].

[28] L. N. Smith, “A Disciplined Approach To Neural Network Hyper-parameters,” US Naval Research Laboratory, Washington, 2018.

[29] R. Khan, “Understanding & Creating Neural Networks with Computational Graphs from Scratch,” kdnuggets, August 2019. [Online]. Available: https://www.kdnuggets.com/2019/08/numpy-neural-networks-computational-graphs.html. [Accessed 15 April 2022].

[30] M. Berry, B. Yap, A. Mohamed and M. Koppen, Book: Soft Computing in Data Science, Lizuka: Springer, 2019.

[31] “Student Notes: Convolutional Neural Networks (CNN) Introduction,” IndoML, 7 March 2018. [Online]. Available: https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/. [Accessed 4 April 2022].

[32] Y. Y. Lee, A. Halim and M. Ab Wahab, “License Plate Detection Using Convolutional Neural Networks,” *IEEE Access,* vol. 10, pp. 22577-22585, 2022.

[33] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *Journal of Machine Learning Research,* vol. 56, no. 15, pp. 1929-1958, 2014.

[34] P. Radhakrishnan, “What are Hyperparameters ? and How to tune the Hyperparameters in a Deep Neural Network?,” *Towards Data Science,* 9 August 2017.

[35] L. N. Smith, “Cyclical Learning Rates for Training Neural Networks,” in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Santa Rosa, CA, 2017.

[36] S. R. Rath, “Debugger Cafe,” 26 Oct 2020. [Online]. Available: https://debuggercafe.com/getting-started-with-facial-keypoint-detection-using-pytorch/. [Accessed 16 Feb 2022].

[37] I. Kandel and M. Castelli, “The Effect of Batch Size on the Generalizability of the Convolutional Neural Networks,” *ICT Express,* vol. 6, pp. 312-315, 5 May 2020.

[38] M. Kuhn and K. Johnson, Book: Applied Predictive Modeling, New York: Springer, 2013, pp. 66-67.

[39] R. Castellon, “Towards Data Science,” 01 Apr 2020. [Online]. Available: https://towardsdatascience.com/where-does-mean-squared-error-mse-come-from-2002bbbd7806. [Accessed 17 Mar 2022].

[40] S. Lau, “Learning Rate Schedules and Adaptive Learning Rate Methods for Deep Learning,” *Towards Data Science,* 29 July 2017.

[41] R. Bhat, “Adaptive Learning Rate: AdaGrad and RMSprop,” *Towards Data Science,* 10 October 2020.

[42] D. Kingma and J. Ba, “ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION,” in *3rd International Conference on Learning Representations, ICLR 2015*, San Diego, CA, 2015.

[43] A. Verma, “Introduction to CUDA Programming,” *GeeksforGeeks,* 16 October 2021.

[44] “High-Resolution Network: A universal neural architecture for visual recognition,” *Computer Vision,* 17 June 2020.

[45] J. Gan, X. Liang, Y. Zhai, L. Zhou and B. Wang, “A Real-Time Face Recognition System Based on IP Camera and SRC Algorithm.,” *Biometric Recognition,* vol. 8833, pp. 120-127, 2014.

8.0 Appendices

8.1 Python Codes

8.1.1 Config.py

```python
# file paths, where the root path goes to the datasets file, and the
#output path goes to the folder where the validation images, loss graph
#and model path are outputted.
ROOT_PATH = 'C:\\Users\\ak234\\OneDrive\\Documents\\Year 4
Uni\\Dissertation\\1513 images\\input\\asteroid com detection'
OUTPUT_PATH = 'C:\\Users\\ak234\\OneDrive\\Documents\\Year 4
Uni\\Dissertation\\1513 images\\outputs'

# learning parameters
BATCH_SIZE = 32
# Learning Rate
LR = 0.0001
EPOCHS = 800

# show dataset keypoint plot
SHOW_DATASET_PLOT = True
```

8.1.2 Dataset.py

```python
import torch
import os
import cv2
import pandas as pd
import numpy as np
from skimage import io
import config
import utils
from torch.utils.data import Dataset, DataLoader
from tqdm import tqdm

resize = 128
# creation of the dataset, which is transferred to python via the
#pandas module. OS is used to connect the images to the dataset image
#names.
class AsteroidDataset(Dataset):

    def __init__(self, csv_file, root_dir):

        self.data = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.images = []
        for i in tqdm(range(len(self.data))):
            img_name = os.path.join(self.root_dir,
                                    self.data.iloc[i, 0])
            image = io.imread(img_name)
            self.images.append(image)
    def __len__(self):

        return len(self.images)
    def __getitem__(self, index):
        image = self.images[index].reshape(1024, 1024)
        orig_w, orig_h = image.shape
        # resize the image into `resize` defined above
        image = cv2.resize(image, (resize, resize))
        # again reshape to add grayscale channel format
        image = image.reshape(resize, resize, 1)
        image = image / 255.0
        image = np.transpose(image, (2, 0, 1))
        # transpose for getting the channel size to index 0
        # get the keypoints
        keypoints = self.data.iloc[index][1:]
        keypoints = np.array(keypoints, dtype='float32')
        # reshape the keypoints
        keypoints = keypoints.reshape(-1, 2)
        # rescale keypoints in accordance with the new image resize
        keypoints = keypoints * [resize / orig_w, resize / orig_h]
        return {
            'image': torch.tensor(image, dtype=torch.float),
            'keypoints': torch.tensor(keypoints, dtype=torch.float),}
```

```
# initialize the dataset - `AsteroidDataset()`
print('\n-------------- PREPARING DATA --------------\n')
# as the validation and training datasets were separate, they must be
#loaded seperately
train_data =
AsteroidDataset(csv_file='C:\\Users\\ak234\\OneDrive\\Documents\\Year 4
Uni\\Dissertation\\1513 images\\input\\asteroid com
detection\\training\\training.csv' ,
root_dir='C:\\Users\\ak234\\OneDrive\\Documents\\Year 4
Uni\\Dissertation\\1513 images\\input\\asteroid com
detection\\training\\training and validation images' )
valid_data =
AsteroidDataset(csv_file='C:\\Users\\ak234\\OneDrive\\Documents\\Year 4
Uni\\Dissertation\\1513 images\\input\\asteroid com
detection\\validation\\validation.csv' ,
root_dir='C:\\Users\\ak234\\OneDrive\\Documents\\Year 4
Uni\\Dissertation\\1513 images\\input\\asteroid com
detection\\validation\\validation images' )
#print('\n-------------- DATA PREPRATION DONE --------------\n')
# preparing data loaders into the program. Batch_size is inherited from
#the config.py folder. Shuffle is set to true, meaning that each batch
#being trained is made up of randomly selected images from within the
#datasets.

train_loader = DataLoader(train_data,
                          batch_size=config.BATCH_SIZE,
                          shuffle=True)

valid_loader = DataLoader(valid_data,
                          batch_size=config.BATCH_SIZE,
                          shuffle=False)

# used to decide whether to show dataset keypoint plots or not
if config.SHOW_DATASET_PLOT:
    utils.dataset_keypoints_plot(valid_data)
```

8.1.3 Utils.py

```
import matplotlib.pyplot as plt
import numpy as np
import config

def valid_keypoints_plot(image, outputs, orig_keypoints, epoch):
    """
    This function plots the regressed (predicted) keypoints and the
actual
    keypoints after each validation epoch for one image in the batch.
    """
    # just get a single datapoint from each batch
    img = image[11]
    output_keypoint = outputs[11]
    orig_keypoint = orig_keypoints[11]
# convert image to numpy array, transpose and reshape to the resize
#given in Dataset.py
    img = np.array(img, dtype='float32')
    img = np.transpose(img, (1, 2, 0))
    img = img.reshape(128, 128)
    plt.imshow(img, cmap='gray')

    output_keypoint = output_keypoint.reshape(-1, 2)
    orig_keypoint = orig_keypoint.reshape(-1, 2)
    for p in range(output_keypoint.shape[0]):
# plot the predicted keypoint in red and the ground truth in green

        plt.plot(output_keypoint[p, 0], output_keypoint[p, 1], 'r.')

        plt.plot(orig_keypoint[p, 0], orig_keypoint[p, 1], 'g.')
# save the validation image in the output folder
    plt.savefig(f"{config.OUTPUT_PATH}/val_epoch_{epoch}.png")
    plt.close()


def test_keypoints_plot(images_list, outputs_list):
    """
    This function plots the keypoints for the outputs and images
    in the `test.py` script which used the `test.csv` file.
    """
    plt.figure(figsize=(10, 10))
    for i in range(len(images_list)):
        outputs = outputs_list[i]
        image = images_list[i]
        outputs = outputs.cpu().detach().numpy()
        outputs = outputs.reshape(-1, 2)
        p.subplot(3, 3, i+1)
        plt.imshow(image, cmap='gray')
        for p in range(outputs.shape[0]):
                plt.plot(outputs[p, 0], outputs[p, 1], 'r.')
        plt.axis('off')
    plt.savefig(f"{config.OUTPUT_PATH}/test_output.png")
    plt.show()
    plt.close()

def dataset_keypoints_plot(data):
    """
    This function shows the keypoint plots that the model
    will actually see. This is a good way to validate that our dataset
is in
    fact correct and the asteroid aligns with the keypoint features.
The plot
    will be show just before training starts.
    """
    plt.figure(figsize=(20, 40))
    for i in range(30):
        sample = data[i]
        img = sample['image']
        img = np.array(img, dtype='float32')
        img = np.transpose(img, (1, 2, 0))
        img = img.reshape(128, 128)
        plt.subplot(5, 6, i+1)
        plt.imshow(img, cmap='gray')
        keypoints = sample['keypoints']
        for j in range(len(keypoints)):
            # plots the 30 verification images using a cyan keypoint
            plt.plot(keypoints[j, 0], keypoints[j, 1], 'c.')
    plt.show()
    plt.close()
```

8.1.4 Model.py

```
import torch.nn as nn
import torch.nn.functional as F
# creates the CNN class, and defines all the inputs and outputs of the
#convolutional layers, as well as defining max pool kernel size and the
#dropout probability (0.175 = 17.5%)
class AsteroidModel(nn.Module):
    def __init__(self):
        super(AsteroidModel, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=9)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=7, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=5)
        self.conv4 = nn.Conv2d(128, 256, kernel_size=5)
        self.conv5 = nn.Conv2d(256, 512, kernel_size=2, padding = 1)
        self.conv6 = nn.Conv2d(512, 1024, kernel_size=2)
        self.fc1 = nn.Linear(1024, 2)
        self.pool1 = nn.MaxPool2d(2, 2)
        self.dropout = nn.Dropout2d(p=0.175)
      # begins the forward steps of the neural network, in order to
#train the network using the convolutional layers, max pooling layers,
#and ReLU activation functions.
    def forward(self, x):
         x = F.relu(self.conv1(x))
         x = self.pool1(x)
         x = F.relu(self.conv2(x))
         x = self.pool1(x)
         x = F.relu(self.conv3(x))
         x = self.pool1(x)
         x = F.relu(self.conv4(x))
         x = self.pool1(x)
         x = F.relu(self.conv5(x))
         x = F.relu(self.conv6(x))
         x = self.pool1(x)
         bs, _, _, _ = x.shape
         x = F.adaptive_avg_pool2d(x, 1).reshape(bs, -1)
         x = self.dropout(x)
# connects the network into a 1 dimensional matrix, to output the
#predicted keypoints
         out = self.fc1(x)
         return out
```

8.1.5 Train.py

```python
import torch
import torch.optim as optim
import matplotlib.pyplot as plt
import torch.nn as nn
import matplotlib
import config
import utils
import os
import io
from model import AsteroidModel
from dataset import train_data, train_loader, valid_data, valid_loader
from tqdm import tqdm

matplotlib.style.use('ggplot')
# model
model = AsteroidModel()
# optimizer
optimizer = optim.Adam(model.parameters(), lr=config.LR)
# we need a loss function which is good for regression like MSELoss
criterion = nn.MSELoss()


# training function
def fit(model, dataloader, data):
    print('Training')
    model.train()
    train_running_loss = 0.0
    counter = 0
    # calculate the number of batches
    num_batches = int(len(data)/dataloader.batch_size)
    for i, data in tqdm(enumerate(dataloader), total=num_batches):
        counter += 1
        image, keypoints = data['image'], data['keypoints']
        # flatten the keypoints
        keypoints = keypoints.view(keypoints.size(0), -1)
        optimizer.zero_grad()
        outputs = model(image)
        loss = criterion(outputs, keypoints)
        train_running_loss += loss.item()
        loss.backward()
        optimizer.step()
    train_loss = train_running_loss/counter
    return train_loss

# validation function
def validate(model, dataloader, data, epoch):
    print('Validating')
    model.eval()
    valid_running_loss = 0.0
    counter = 0
    # calculate the number of batches
    num_batches = int(len(data)/dataloader.batch_size)
    with torch.no_grad():
        for i, data in tqdm(enumerate(dataloader), total=num_batches):
            counter += 1
            image, keypoints = data['image'], data['keypoints']
            # flatten the keypoints
            keypoints = keypoints.view(keypoints.size(0), -1)
            outputs = model(image)
            loss = criterion(outputs, keypoints)
            valid_running_loss += loss.item()
            # plot the predicted validation keypoints after every...
            # 25 epochs, epochs+1 because the indexing starts at index
            # 0, therefore epoch 25 is epoch 24 to the program.
            if (epoch+1) % 25 == 0 and i == 0:
                utils.valid_keypoints_plot(image, outputs, keypoints,
epoch)

    valid_loss = valid_running_loss/counter
    return valid_loss

train_loss = []
val_loss = []
for epoch in range(config.EPOCHS):
    print(f"Epoch {epoch+1} of {config.EPOCHS}")
    train_epoch_loss = fit(model, train_loader, train_data)
    val_epoch_loss = validate(model, valid_loader, valid_data, epoch)
    train_loss.append(train_epoch_loss)
    val_loss.append(val_epoch_loss)
    print(f"Train Loss: {train_epoch_loss:.4f}")
    print(f'Val Loss: {val_epoch_loss:.4f}')


# loss graphs are generated here and then sent to the output path.
plt.figure(figsize=(10, 7))
plt.plot(train_loss, color='orange', label='train loss')
plt.plot(val_loss, color='red', label='validataion loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.savefig(f"{config.OUTPUT_PATH}\\loss.png")
plt.show()
# saving the weights for testing.
torch.save({
            'epoch': config.EPOCHS,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': criterion,
            }, f"{config.OUTPUT_PATH}\\model.pth")
print('DONE TRAINING')
```