Neurocomputing for Unmanned Aerial Vehicles: A Study

IS NEURAL NETWORK BACKPROPAGATION COMPARABLE IN PERFORMANCE TO LOGISTIC REGRESSION FOR RECOGNISING SHAPES FROM A UAV AND CAN AN EFFICIENT PATH BE CALCULATED TO FLY THROUGH THEM?

Daniel Jon Chalmers

SUPERVISOR: DR MARK ELSHAW

COVENTRY UNIVERSITY | SCHOOL OF COMPUTING, MATHEMATICS AND DATA SCIENCE | 2023

6001CEM Declaration of originality

I declare that this project is all my own work and has not been copied in part or in whole from any other source except where duly acknowledged. As such, all use of previously published work (from books, journals, magazines, internet etc.) has been acknowledged by citation within the main report to an item in the References or Bibliography lists. I also agree that an electronic copy of this project may be stored and used for the purposes of plagiarism prevention and detection.

Statement of copyright

Statement of ethical engagement

I declare that a proposal for this project has been submitted to the Coventry University ethics monitoring website (https://ethics.coventry.ac.uk/) and that the application number is listed below:

Signed: [illegible] Date: 11/04/23

First Name	Daniel
Last Name	Chalmers
Student ID number	9127359
Ethics Application Number	P146097
Supervisor	Mark Elshaw

Abstract

This project randomly generates an environment, populates it with augmented shapes, recognises which images are of which shapes, plans a path through each image of a configurable shape, and flies the calculated path.

The report includes a comparison of the performance of Logistic Regression (LR) against that of a Convolutional Neural Network (CNN) for identifying which shape (circle, square, star, pentagon, or triangle) is present in a highly augmented image. The models' performances were measured by calculating the F-scores, sensitivity and specificity as well as analysing their respective Confusion Matrices. Both models were found to be highly accurate post-tuning, but the CNN performed better overall and was quicker once trained. The LR model had an accuracy of 97.15%, and an F-score of 0.9427, whereas the CNN had an accuracy of 98.75% and an F-score of 0.98732.

The algorithms were built from the ground up, including the calculation of derivatives for gradient descent and tuned by modifying their hyperparameters. The CNN was chosen as the Machine Vision model to run in the simulation due to its accuracy, speed, and ability to load a pre-tuned weights file.

To find an efficient route through all the relevant shapes, a Minimum Spanning Tree heuristic algorithm (Kruskal's method) was determined to be the most suitable because of its relatively quick time complexity and local-optimal solutions.

A software-in-the-loop, open-source simulation (Mission Planner) was used to visualise the UAV's route, making use of the MAVLink connection protocol for communication between the UAV and the host machine.

Keywords: Machine Vison, Logistic Regression, Convolutional Neural Network, UAV, UAS, Heuristics, Simulation, Gradient Descent, Compression, Backpropagation, Augmentation, Pixel, Plasticity.

Development Public Repository -> https://github.com/Dan-Chalmers/ICARUS

Demonstration Public Repository -> https://github.coventry.ac.uk/chalme10/ICARUS

Acknowledgments

The author is thankful for the support and guidance provided by the following persons:

- Mark Elshaw (Supervisor)
- Daniel Perks
- Amanda Chalmers
- Jon Chalmers

Contents

List of Tables

List of Figures

1.0 – Introduction

1.1 – Project Aim

The goal of this project is to design, develop, train, and tune Logistic Regression and Convolutional Neural Network algorithms to recognise basic shapes and create a shortest-path heuristic algorithm to allow a simulated UAV to fly through all shapes of a configurable type.

1.2 – Objectives

- Create primary image shape data.
- Randomly augment the image data to create training and validation sets to train and test the Machine Vision models.
- Design, develop and tune a Logistic Regression model.
- Design, develop and tune a Backpropagating Convolutional Neural Network
- Train both models using supervised learning with the data sets.
- Analyse the performance of each model using multiple metrics.
- Design and develop a shortest-path heuristic algorithm to allow the UAV to fly through all configured shapes in the quickest time possible.
- Visualise the UAV's path with *Mission Planner* simulation software and the *dronekit-sitl* Python library.

1.3 – Background

In recent years, Artificial Intelligence has been at the forefront of technological evolution and public scrutiny. Innovations such as *DeepMind's AlphaFold* and *OpenAI's GPT-3* have pushed humanity's understanding of what is possible to new levels and spawned an entirely new approach to problem solving. Artificial General Intelligence (AGI) is a rapidly developing field that is likely to continue changing the way we live our lives. As such, our reliance on such technology will only grow. The idea of this project is to explore how practical AGI is in the context of Unmanned Aerial Systems (UAS).

The results and analysis of this research will help decide which statistical modelling methods are most helpful in integrating Machine Vision into aerospace and UAS applications. This investigation will produce a high-level demonstration of AGI-enhanced UAV heuristics along with an examination of what this technology will mean for the commercial aerospace, agriculture, transport and disaster management sectors and future possibilities. It is my hope that this project (ICARUS) will demonstrate the next steps in aerial object recognition.

A large focus of this project will be the design and development of a demonstration. I hope that my findings, whether in full or in part, can be utilised by the aerospace sector and any other business area where the findings can be applied. The use of agile, sprint-based, development allows for a people-centric outlook of the design's progress (Abrahamsson et al., 2017, p. 106). Such an iterative development lifecycle gives flexibility for changes to be made and stories to be added at any point.

1.4 – Motivation

The motivation for this research project derives itself from my passion for AI and my time spent working in the aerospace industry during my industrial placement at Lockheed Martin. My research can optimise existing solutions and products as well as creating opportunities that could be integrated into emerging technologies.

Furthermore, I wanted to not only create an object recognition system, but also incorporate it into a use case (UAVs) with other features such as pathfinding and a simulation to make a complete product and to explicitly highlight the benefits of such technology in a contextual manner.

1.5 – Report structure

The project is organised into the following structure:

- Literature Review. Explores past research into various relevant fields as well as looking at what the future holds for Machine Vision and UAV/UAS applications.
- Project Management. Discusses the methodologies, technologies and metrics used to plan, organise, and manage the project.
- Data Collection and Pre-processing. Outlines the methods used to create the primary data as well as the algorithms used to transform it into the appropriate forms.
- Logistic Regression. Illustrates the design, implementation, training, tuning and analysis of the Logistic Regression Machine Vision model.
- Convolutional Neural Network. Outlines all the same stages as the Logistic Regression section but for the implementation of a Convolutional Neural Network.
- Constructing the System. Integrating the pre-processing and Machine Vision aspects together as well as demonstrating the pathfinding and simulation capabilities.
- Conclusion. Summarises the report and outlines the ethical considerations for such a technology and ideas for extending the project.

2.0 – Literature Review

The purpose of this literature review is to understand past research into similar topics and analyse whether I could use or expand upon other people's findings in my own design and development of a Machine Vision system for UAVs.

The four focuses of this review are: Machine Vision; Logistic Regression; Neural Networks and current UAV research. The latter three all relate to the research question and Machine Vision is included as it's the overarching concept.

2.1 – Machine Vision

Machine Vision is a focused branch of an otherwise expansive artificial intelligence tree. (Alves et al., 2018) discuss the evolution of Machine Vision methods from early knowledge-based systems to more recent cognitive learning approaches. They define cognitive methods as an amalgamation of knowledge-based models and data-driven learning. Logistic Regression and Neural Networks are large parts of such cognitive methods. The authors list object recognition, scene understanding and activity recognition as examples of cognitive learning approaches. This project focuses on object recognition.

A good example of Machine Vision being used by UAVs is the research project outlined in the paper: "-*A Forest Fire Recognition Method Using UAV Images Based on Transfer Learning"* (Zhang et al., 2022). This project focused on identifying forest fires from an aerial perspective by utilising a pre-trained model. The authors found that their network was 77.47% accurate. This seems high, considering the chaotic nature of a flame. My project is slightly different to the one outlined in this paper because of the method of training. Instead of using transfer learning, I will use a more traditional supervised machine learning approach, training my models from scratch with labelled data. In theory, I should get a higher accuracy.

Machine Vision is quickly becoming an expanding field and, according to (Labudzki et al., 2014), it plays an increasingly important role in the automation, quality control, and surveillance sectors. They clearly outline the main stages of machine vision as: image acquisition; pre-processing; feature extraction and decision-making. These are the steps that I'll follow when designing and implementing my machine vision models.

An interesting approach to the pre-processing stage is to utilise image signal processor (ISP) enhancement (Park et al., 2021). The authors describe a method to enhance the input image quality by using augmentations such as colour correction, denoising and gamma correction. Although the application slightly differs from the context of my project, I will use the features described in their method in reverse, e.g., inducing gamma to make the images less recognisable. Therefore, making it more challenging for the machine vision models to recognise the shapes.

A fascinating example of a research project that uses Machine Vision is the paper by (Çelik et al., 2013), entitled: *"Development of a Machine Vision system: real-time fabric defect detection and classification with neural networks."* The authors use a neural network to classify defects in certain fabrics. Importantly, they discuss the pre-processing steps that they underwent to prepare the data – including transforming all the image data into greyscale pixel values. By processing the images as greyscale, it allowed the network to focus in on variations in the brightness and intensity of the fabric. I will use a similar technique in my models to focus them to look for shapes instead of colours.

2.2 – Logistic Regression

A research paper written by (Waliyansyah & Hasbullah, 2021) is useful for my project as it focuses on different statistical methods and their effectiveness for classifying images. The authors discuss the implementation and analysis of Logistic Regression in the context of classifying different coffee beans. Although the application is similar to my project, the paper in question only utilises two classifiers: *Arabica* and *Robusta* (types of coffee bean). In my case, the project will have five classifiers (shapes). The conclusion of this paper states that *"Logistic Regression Method AUC 0.998, CA = 0.966, F1 = 0.965, Precision = 0.968 and Recall = 0.966"*. These values are extremely high, meaning that LR (in this context and application) is very accurate. This is a good reason to use Logistic Regression as a benchmark to test my neural network against.

(Wang et al., 2019) delve into the assumptions and limitations of logistic regression. They conclude that logistic regression assumes that: the relationship between the independent variables and the log of the dependant variable is linear; the observations are independent of each other; there is no multicollinearity among the independent variables and that the error terms are normally distributed. Furthermore, the authors outline the method's limitations as: poor performance for non-linear relationships; sensitivity to outliers and the inability to work with time-series data due to the variables' independence. I'll keep these limitations in mind when designing my regression model and deal with any outliers when or if they appear.

The paper *"Logistic Regression: From Art to Science"* by (Bertsimas & King, 2017) discusses the development of logistic regression from a widely used statistical technique based on intuition and subjective judgement, to the foundation of computational algorithms such as 'maximum likelihood estimation' and 'Newton's method'. Although not directly related to my specific application, this research provided a good opportunity to understand the basis of LR and how it can be used in a wider context. What's more, the authors debate potential extensions to LR such as Ridge and Lasso regularisation; a method of reducing the model's variance, therefore avoiding overfitting (Miller, 2019). If I find that my LR algorithm has coefficients that become too great, I will consider incorporating regularisation into the code.

2.3 – Neural Networks

The paper *"Bag of tricks for Image Classification with Convolutional Neural Networks"* by (He et al., 2019) explores various ways of modifying Neural Networks without changing their foundation and analyses the results. The model architecture is refined without affecting computational complexity. The authors try a few methods of tweaking the CNNs' output. The greatest improvement comes from utilising 'transfer learning'. This is the process of taking an existing deep learning model that has been trained to an acceptable standard with unseen data and apply it to the relevant dataset. This proves to be quick, as the model is already trained, and accurate.

One of the primary objectives of my research project is to compare the performance of Neural Networks against that of Logistic Regression. This is not considered in this paper and it will be interesting to see how logistic regression copes with similar tuning methods. My project also furthers the research done in this paper by applying context to the topic (UAVs).

A similar piece of literature is a book on neural networks and how their makeup is reminiscent of biology (Bhoi et al., 2021). Of interest is their investigation into activation functions – particularly the part they play in the CNN's structure and the affect they have on the output. On pages 212 to 240, the authors implement, test and analyse a series of different activation functions in a Neural Network. This investigation is incredibly useful for my project as it allows me to narrow down my search for the best activation function. I've chosen the ReLU, Softmax and Sigmoid functions to try out in my network based on their findings.

There are many types of neural network architectures – each with their own advantages and ideal applications. (Pecheninia et al., 2021) look at a few of these in detail to try and prove which is best for machine vision. They experimented with CNNs, Recurrent Neural Networks (RNN), and Deep Belief Networks (DBN). The authors determine that CNNs outperform RNNs and DBNs using a dataset of industrial images and measuring performance with metrics such as accuracy and processing time. They conclude that CNNs are the most suitable neural network architecture. According to (Laskowski, 2021), RNNs are more popular for applications such as Speech Recognition and Natural Language Processing.

Although slightly out of the scope of this project, DBNs make for interesting reading and present potential possibilities to expand the code base and test suite. According to (Canuma, 2020), DBNs are generative models that utilise a deep architecture. The foundations for these networks are Boltzmann Machines. These are graphs that are used to aid the understanding of parameters such as entropy on the quantum states in the field of thermodynamics. They differ from classical neural network structures in that they don't have output nodes and don't perform gradient descent calculations.

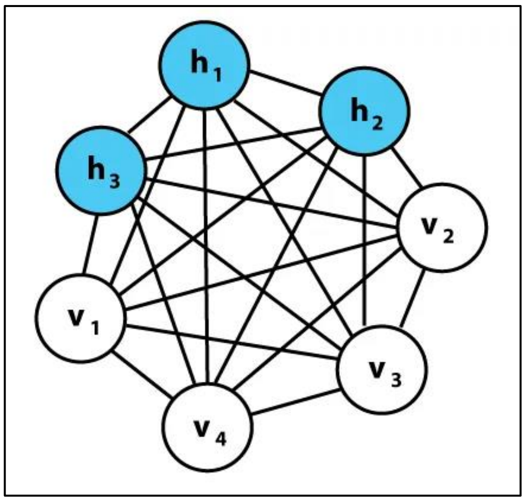


Figure 1 - A traditional Boltzmann Machine (Canuma, 2020)

Boltzmann Machines work by sharing information between nodes, generating subsequent data, and undergoing 'Contrastive Divergence'. Along with image generation and classification, Boltzmann Machines are used in video recognition and motion capture. A Deep Belief Network is simply a directional Boltzmann Machine.

Concerning Convolutional Neural Networks, (Ajit et al., 2020) discuss the overall architecture as well as optimisation techniques. An interesting point outlined in their paper is the use of stochastic gradient descent being superior to that of batch gradient descent when it comes to calculating the derivatives of weights in a network. They also illustrate the danger of the 'vanishing gradient problem'.

The 'vanishing gradient problem' occurs when the gradients of the cost function with respect to the network's parameters become very small. This makes it very hard, and in some cases, impossible for the network to learn (Hu et al., 2021). The authors propose a fix for this issue in the form of 'Artificial Derivatives (AD)'. This AD technique involves adding artificial noise to the weights' gradients during the process of backpropagation to prevent the gradients becoming too small and 'vanishing'. Depending on how well my network learns, I may have to incorporate this method into the implementation.

2.4 – UAV/UAS Current Research

My project is contextualised by applying the technology to UAVs and UASs. The following analysis represents background reading and research into the systems themselves.

According to (Alzahrani et al., 2020), UAVs have become increasingly popular all over the world, especially in sectors such as agriculture and disaster management. The authors describe the advantages of UAVs as being able to cover large areas quickly and reducing human intervention in potentially dangerous areas. The paper concludes by outlining the next steps in UAV/UAS development, namely increased autonomy and improved safety.

Figure 2 - A UAV aiding in disaster management during the Indonesia earthquake (Team Rubicon Australia, 2018)

Figure 3 - A UAV being used for farming (DJI Agriculture, 2021)

Hildmann & Kovacs (2019) describe an application for UAVs where they act as Mobile Sensing Platforms (MSPs) for disaster response, public safety and civil security. An MSP is a system that collects data by using a multitude of on-board sensors and cameras. Using a UAV as an MSP means that it can easily relocate rather than being confined to a specific location. The authors discuss the different types of UAV, namely: fixed-wing; rotary-wing and hybrid, and how they can be used for different disaster response and civil security scenarios. Such technology also has its limitations, such as regulatory issues, safety concerns and technical restrictions. The advantages of using UAVs as MSPs are the ability to provide situational awareness, cover large spaces quickly and access hard to reach areas.

3.0 – Project Management

3.1 – Agile Development

Management and task tracking played a vital part in keeping a coherent structure to reviewing literature, project research and code implementation. I used an agile-based approach known as 'scrum' to structure the project and manage user stories (scrum.org, 1993). At a high level, the concept of scrum is to break up a project into smaller, more manageable tasks (user stories) that are completed over the course of a much shorter period. These iterations are known as sprints. Each sprint had a goal that guided my focus and supported frequent deliverables (Rover et al., 2015, p.2). I configured a Jira board and wrote up the user stories for my first two sprints before I designed or wrote any software. Jira is a proprietary issue tracking and work management tool, built by *Atlassian* that specialises in Kanban board organisation and automated project metrics (Atlassian, 2002).

Using Jira's Kanban board feature allowed me to quickly see what needed doing, what's currently in progress and what I had already completed during the corresponding sprint.

Below is a snapshot of my Kanban board from sprint two:

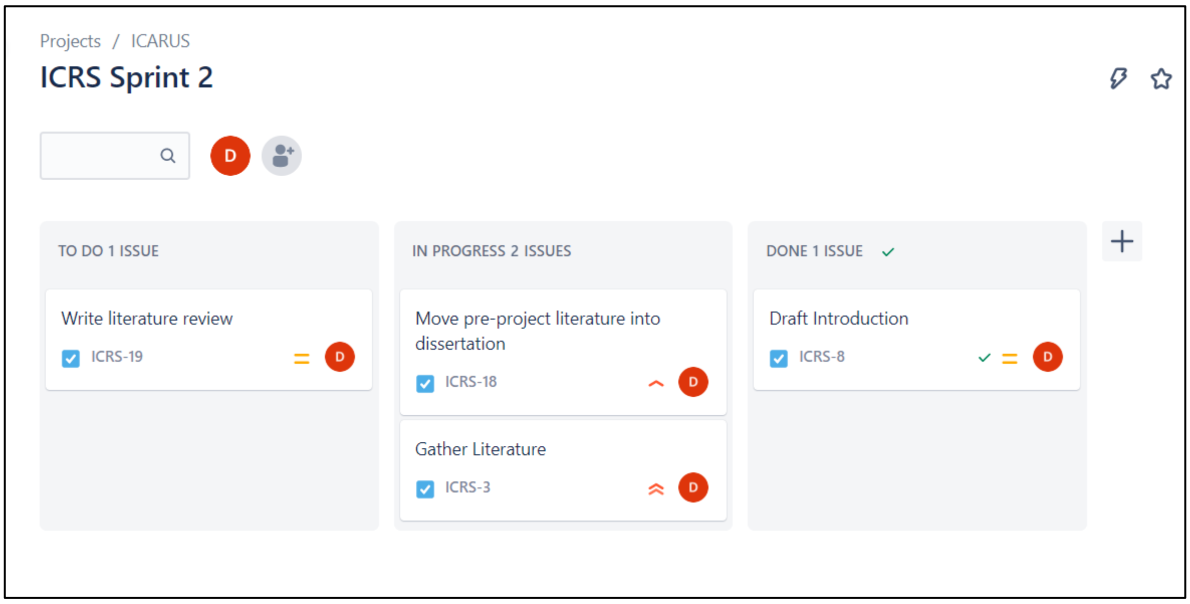


Figure 4 - ICRS Kanban board for Sprint 2

There are three columns: 'To Do'; 'In Progress' and 'Done'. Each task is linked to *Icarus* with the project code ICRS. Each task is assigned to me – indicated by the 'D'. An important and helpful feature is the small symbol to the left of the user profile on each task. This represents its priority. In the context of this Kanban board, the priority is a scale of how important the corresponding task is in comparison to the others on the board. For example, a task with the 'Highest priority' requires immediate attention or could have a different task with a 'Low' priority depending on its completion. A list of all possible priorities and their associated symbols is shown below:

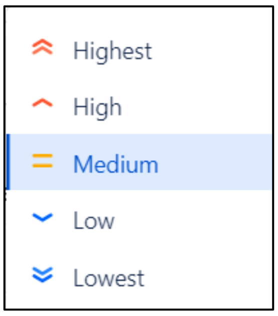
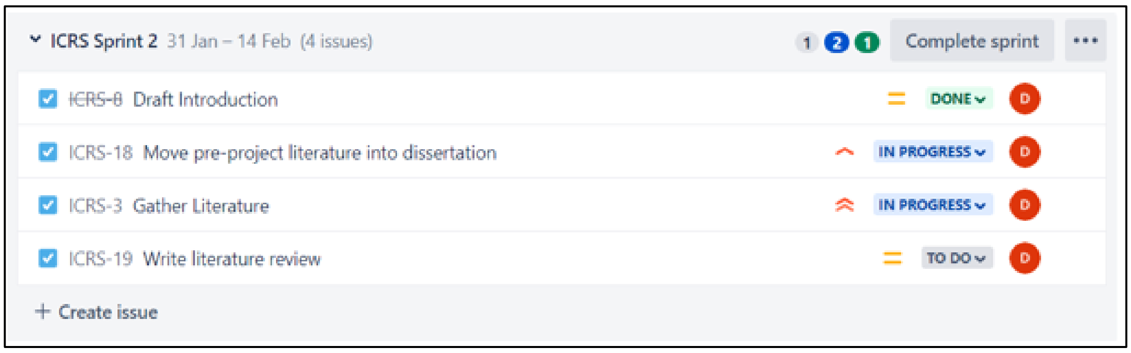


Figure 5 - Jira's priority levels

My Jira project is configured in such a way that the priority is automatically set to 'Medium' until I adjust it manually on a task-by-task basis. Assigning a priority level to each task allows me to quickly identify the most pressing jobs.

From the tool's 'Projects' menu, a high-level overview of the active sprint can be viewed:



ICRS Sprint 2 31 Jan – 14 Feb (4 issues) 1 2 1 Complete sprint

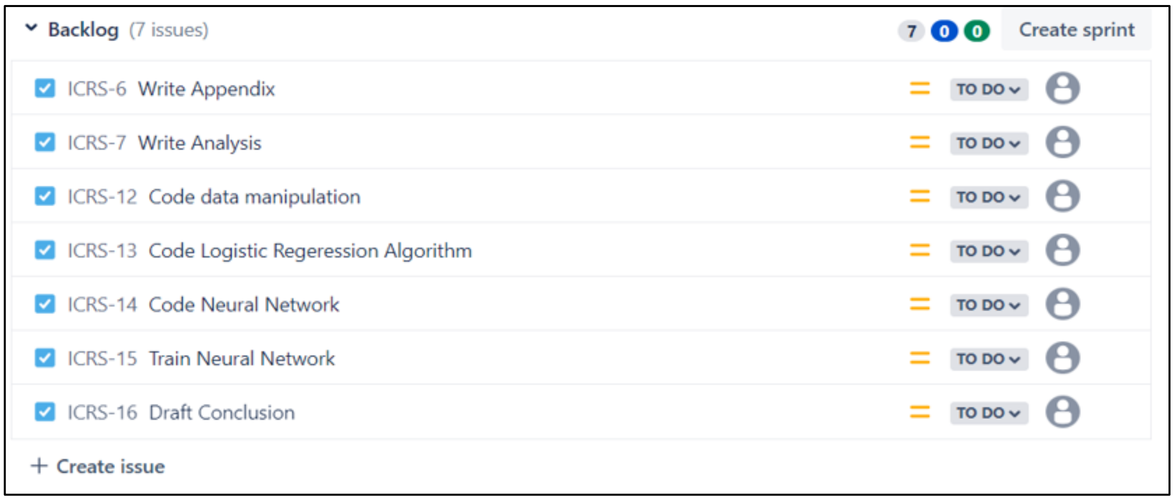
✓ ~~ICRS-8~~ Draft Introduction	DONE	D
✓ ICRS-18 Move pre-project literature into dissertation	IN PROGRESS	D
✓ ICRS-3 Gather Literature	IN PROGRESS	D
✓ ICRS-19 Write literature review	TO DO	D

+ Create issue

Figure 6 - List of tasks in the current Sprint

When all the tasks have been completed, I can close the sprint off with the 'Complete Sprint' button in the top right.

Below the sprint overview is the project's backlog. This is where all future tasks are held. Throughout the project, these are moved into sprints, a few at a time, until there is nothing left in the backlog:


Backlog (7 issues) 7 0 0 Create sprint

ICRS-6 Write Appendix TO DO

ICRS-7 Write Analysis TO DO

ICRS-12 Code data manipulation TO DO

ICRS-13 Code Logistic Regeression Algorithm TO DO

ICRS-14 Code Neural Network TO DO

ICRS-15 Train Neural Network TO DO

ICRS-16 Draft Conclusion TO DO

+ Create issue

Figure 7 - List of tasks currently in the backlog

One of the primary practises of agile methodologies is test-driven development (TDD). This is a method of testing code that requires the unit tests to be written before the source code itself. I used this strategy during implementation so that I could write the unit tests without having preconceived knowledge of how the code works. This was important as it meant I wasn't subliminally writing tests to pass rather than tests that check the requirements. Using TDD also had the advantage of being able to run the tests immediately rather than having to spend time writing them once the source code is ready to be scrutinised (Janzen & Saiedian, 2005, p.2).

Agile is an effective method for delivering many smaller parts of the project in a relatively short time frame. However, this can cause a fragmented output, meaning that the individual deliverables may not be completely compatible with each other (Sharma et al, 2012, p.4). To prevent this fragmentation from occurring, I included an integration testing step on completion of each coding task. Performing integration testing at every possible step allowed me to identify and remove any bugs as soon as they appeared and meant the system could be built iteratively rather attempting a 'big bang' integration at the end of the project. Continuous integration is an example of the *DevOps* framework which solely focuses on the fast and flexible development of software (Ebert et al, 2016, p. 1).

There are many metrics that can be used to measure the project's state of completion and how far ahead or behind schedule I am. Jira automatically generates a range of analytics, graphs, and metrics. One example is a burndown chart that shows jobs outstanding (y), against time (x) as well as a line that represents constant velocity (the rate at which tasks are completed), known as the 'guideline'. The burndown chart for sprint two is shown below:

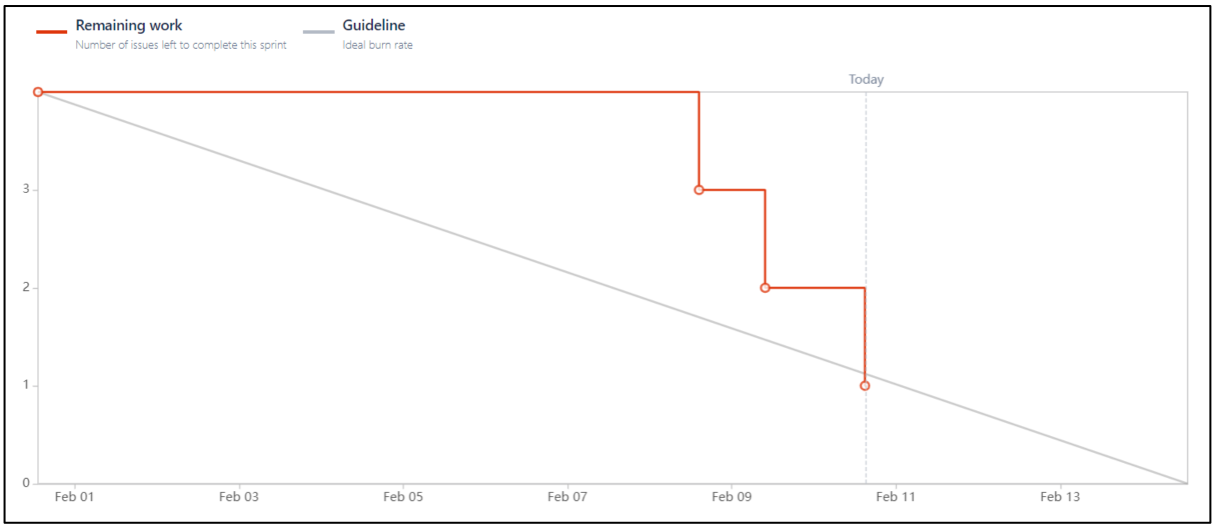


Figure 8 - Burndown chart for Sprint 2

The sprint began with four tasks to be completed and ended with one outstanding (rolled over into the next sprint). Most tasks were completed in the last quarter of the sprint as I tended to work on them simultaneously and ticked them off at similar times, typically towards the end of the two weeks. A burnup chart is also generated. This shows the same data but on a reversed y axis.

This report and the code implementation and testing took place over the course of five sprints (Jan 16th – Mar 31st) with a further two weeks for finalising any outstanding tasks. This sixth sprint is known as the 'hardening sprint'.

3.2 – Version Control

To keep my code base organised, I utilised a Version Control System (VCS), namely: *git* and *GitHub*. After any change or additions to the code base, I committed everything to my GitHub repository. This allowed me to organise the code and, if needed, revert to a previous iteration.

Version control meant that, if anything were to happen to the code on my local machine, I'd still have access to everything through GitHub's cloud storage. Furthermore, if this project were to be continued with a larger development team, all members would be able to see the project's history. Shared repositories also open up the ability to host code reviews and merge requests allowing for greater collaboration and communication.

Post-completion of the project, the commit history looks like the following:

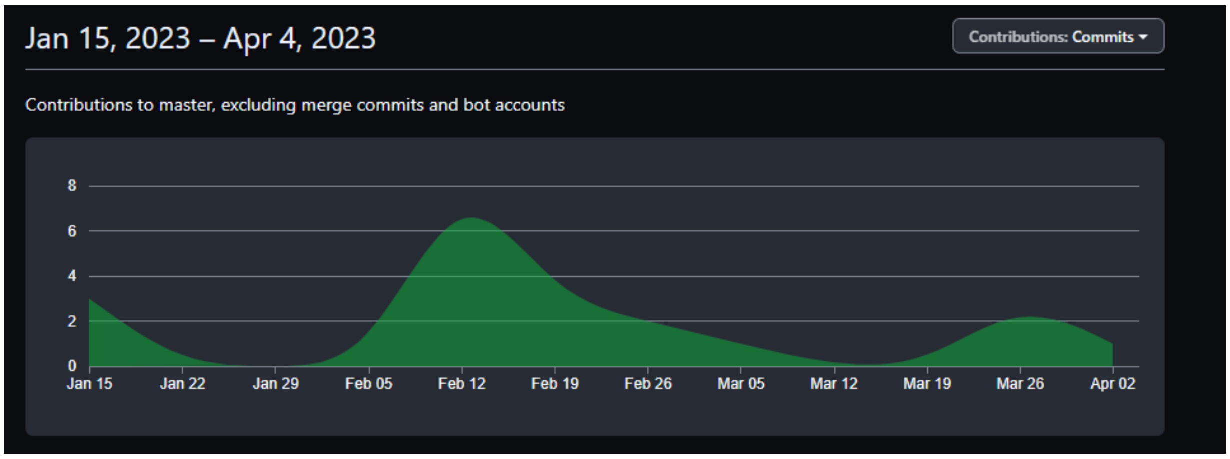


Figure 9 - Graph showing the frequency of commits over the course of the project (generated from GitHub)

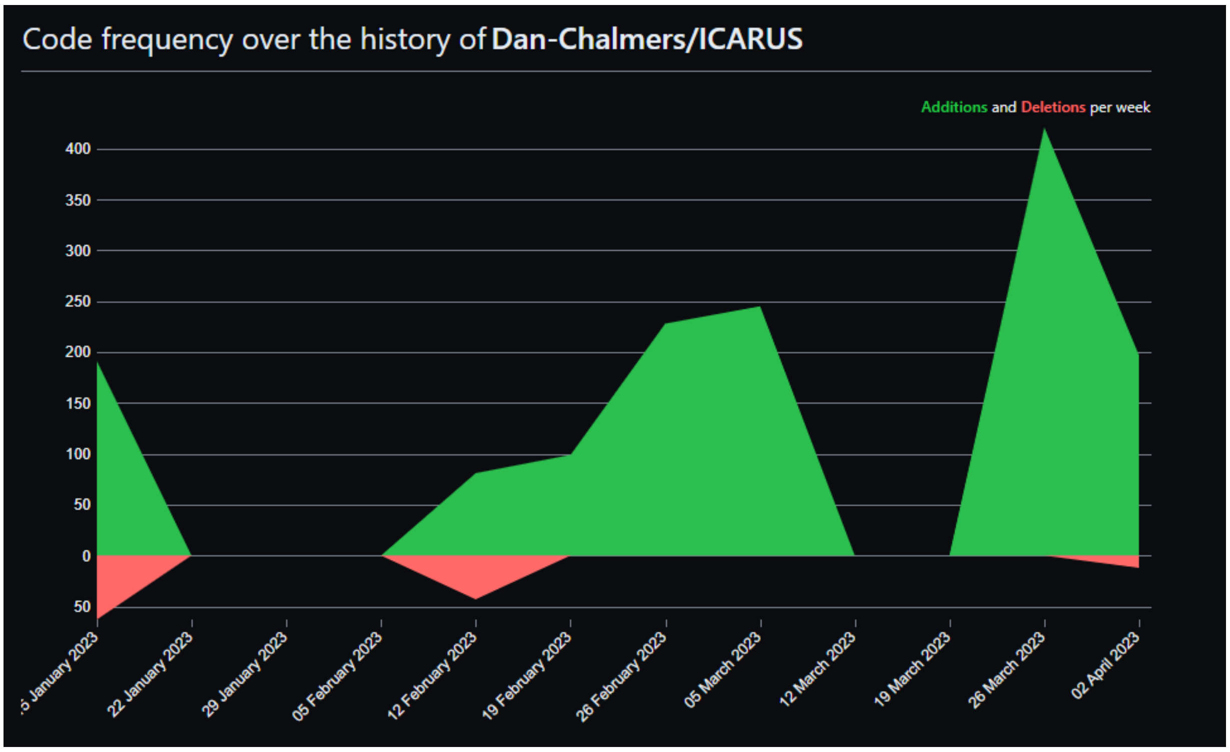


Figure 10 - Graph showing the frequency of code add/deleted over the course of the project (generated from GitHub)

4.0 – Data Collection and Pre-processing

4.1 – Initial data creation

Before being able to implement any statistical models, I needed a dataset. I knew I wanted a large set to optimise the algorithms' predictions as much as possible. Using *Microsoft Visio*, I drew five shapes with the same image dimensions. These are circle, pentagon, square, star and triangle:

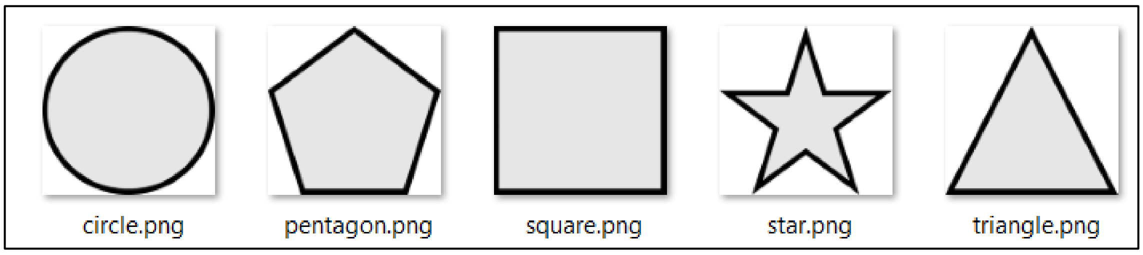


Figure 11 - The shapes that the training/validation data will be based on (primary data)

In context, these shapes will act as 'targets' for the UAV. The 'vehicle' will identify all the shapes present in the environment and fly through all the shapes of the configured parameter type. For example, if the user sets the target to 'circle', under ideal conditions, the UAV will fly through all the circles, following the most efficient route and avoiding all other shapes.

I chose these five shapes as they represent common geometry in nature. I added the star and pentagon to test the capabilities of the models and to explore whether they struggle with more complex shapes than simplistic ones.

4.2 – Retaining Eigenvalue Consistency for Image Augmentation

If the data was kept exactly as shown above, then the algorithms do not need to work very hard to differentiate between shapes! This is not 'recognition'. To force the system into recognising shapes within the images that are not identical to the source image, I needed to augment the shapes.

After some research, I came across the python library: *Albumentations* (Busalaev et al, 2019). This library can modify an image with plethora number of possible augmentations. I composed an 'augmentation pipeline' that pseudo-randomly chooses two out of a predefined list of augmentations. The augmentation methods I chose to implement are listed here (Albumentations/API-Reference, 2019):

- *RandomBrightnessContrast* (Randomly changes the brightness and contrast of image)
- *ShiftScaleRotate* (Randomly scales and rotates the image)
- *HueSaturationValue* (Randomly changes the hue and saturation of the image)
- *RGBShift* (Randomly changes the red, green, and blue pixel values of the image)
- *Blur* (Blurs the image by a random amount)
- *Defocus* (Defocuses the image by a random amount)
- *GaussianBlur* (Blurs the image using a Gaussian filter with a random kernel size)
- *ChannelShuffle* (Randomly rearranges the red, green and blue channels in the image)
- *PixelDropout* (Sets pixel values to zero with random probability)
- *RandomFog* (Overlays fog onto the image)
- *RandomGamma* (Overlays gamma onto the image)
- *RandomSunFlare* (Overlays sun flare onto the image)

It was important to not modify any image's resolution. This could break the machine vision models due to the parameters (pixel data) requiring a consistent size. To verify that none of my chosen augmentations influenced the images' resolutions, I took inspiration from eigendecomposition methods (Abdi, 2007, p. 2).

I analysed the eigenvalues before and after augmentation, this highlighted the corresponding effect on the image's resolution. If the eigenvalues were not consistent, then the number of pixels in each dimension would have been changed. Augmentations that do not apply an eigenvector transformation do not affect resolution and are not of concern. The augmentations that do apply an eigenvector transformation are *ShiftScaleRotate* and *Defocus*. Instead of applying a new layer over the top of the image, these two actions perform a matrix operation.

To validate the eigenvalue consistency, I ensured that each eigenvector transformation satisfied the equation:

$$Av = \lambda v$$

Where A is the transformation matrix (*ShiftScaleRotate*|*Defocus*), v is the eigenvector that is being changed, and λ is the eigenvalue (scalar for eigenvector).

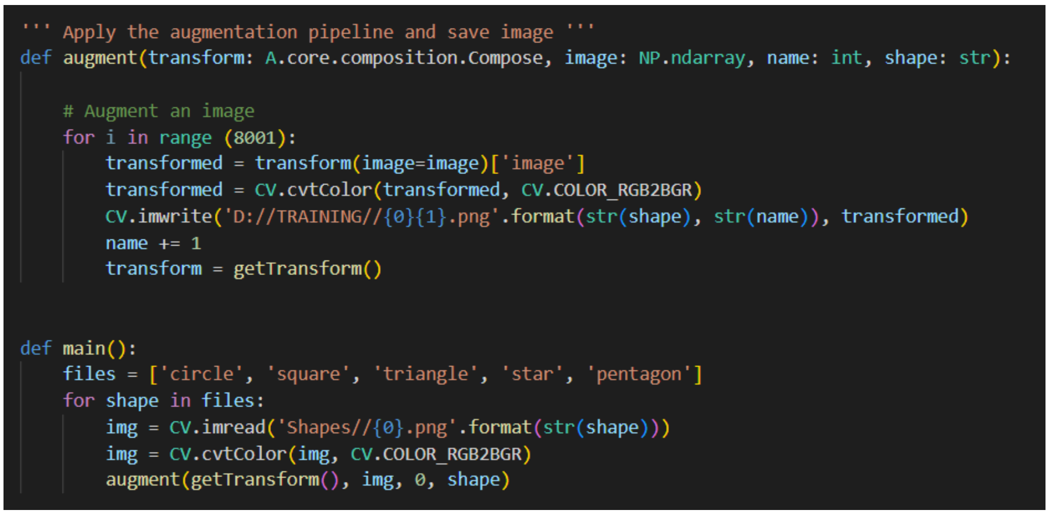
Because the transformations shift the image by an angle θ, around a constant centre point, the length of the affected eigenvector does not change, and the eigenvalue is 1.

The only conditions where this logic fails is if θ is equal to 0. To account for this, any pipeline that utilises either of these augmentations requires the value of θ to be greater than 0.

4.3 – Applying the Augmentation Pipelines

Any singular pipeline contains two randomly selected augmentations from the list. There are 144 pipeline combinations and near infinite available image augmentations due to the random nature of the methods' parameter generation.

The code below takes each shape in turn (read in with the OpenCV library) and generates and applies an augmentation pipeline (function 'transform'). It then saves the new image to a folder on an external HDD which becomes the 'training set'. It does this 8,000 times for each shape.



```python
''' Apply the augmentation pipeline and save image '''
def augment(transform: A.core.composition.Compose, image: NP.ndarray, name: int, shape: str):

    # Augment an image
    for i in range (8001):
        transformed = transform(image=image)['image']
        transformed = CV.cvtColor(transformed, CV.COLOR_RGB2BGR)
        CV.imwrite('D://TRAINING//{0}{1}.png'.format(str(shape), str(name)), transformed)
        name += 1
        transform = getTransform()


def main():
    files = ['circle', 'square', 'triangle', 'star', 'pentagon']
    for shape in files:
        img = CV.imread('Shapes//{0}.png'.format(str(shape)))
        img = CV.cvtColor(img, CV.COLOR_RGB2BGR)
        augment(getTransform(), img, 0, shape)
```

Figure 12 - Code runner script to create the training set.

After running the Python script, I have a training set that consists of 40,000 unique images. Snapshots of the training set are shown below:

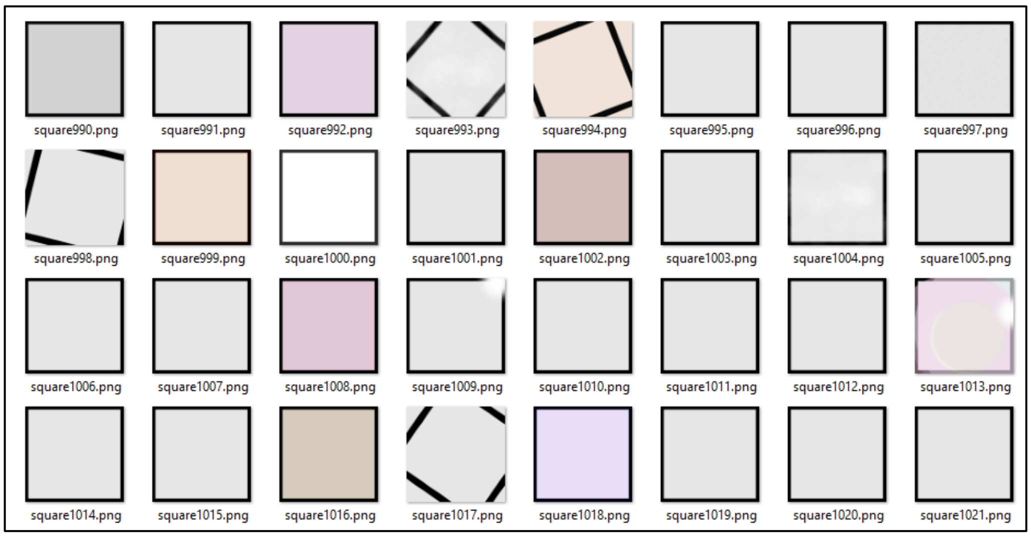


Figure 13 - Some augmented shapes based on the square.

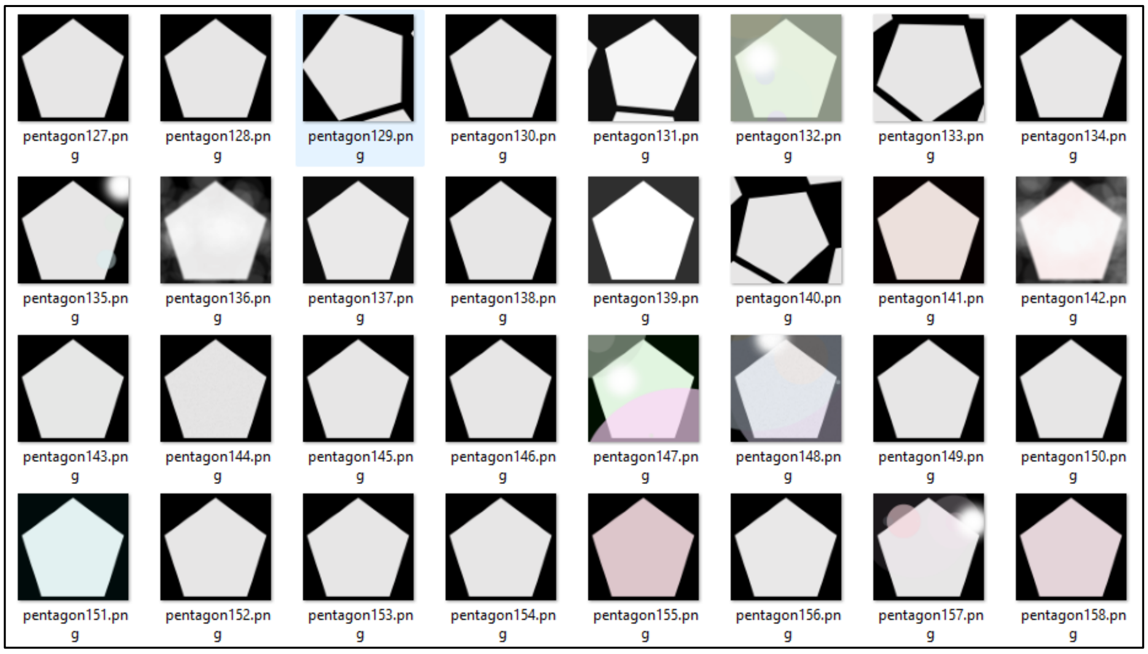


Figure 14 - Some augmented shapes based on the pentagon.

The samples clearly show the application of the rotation, overlay and colour shift augmentations. The only constant in all the images is the respective shape, this will 'force' the algorithms into classifying the images by shape rather than any other attribute e.g., colour or orientation.

During training the models will be fed all the images with their corresponding labels, so they know what they are meant to be classifying the shapes as.

The Training dataset is used to 'teach' the algorithms the difference between the five shapes. On its own, the training set is not useful because there is no way to check the algorithms work post-training. The training set cannot be used to test the models as there would be no way to know whether they were performing well because they recognise the shapes or simply because they are looking at the images' labels (meta-data).

To check the models' performance, a 'validation set' is needed. Traditionally, the validation set for a Machine Learning application would be created by performing an image extraction method such as k-fold cross validation or random grid-search on the training set, prior to training (Miller et al, 2016). However, because my data is primary and generated from fundamentals, I can generate the validation set in isolation.

Using the same code with a couple of modifications to edit the target directory and the number of image augmentation iterations, the validation set came to 10% the size of the training data (800 images per shape). Post-training, the models are run on the validation set without knowledge of the labels, a classification is made for each of the 4,000 images and then checked against the labels. The models' performance is calculated by dividing the number of images through the number of correct classifications.

The training set occupies 19.8 GB and the validation set 2.0 GB.

4.4 – Manipulating Raw Pixel Data

The Machine Learning models compute numerical data and therefore I had to process the images into a format that can be piped into the models' arithmetic. To do this, I wrote image signal processing code that takes each image in turn and transforms the image into a CSV file with each cell containing an array of size three, representing the red, green, and blue colour data for each pixel. Each CSV is 30 MB in size and takes 10 seconds of processing time to read, translate and save the file. A second algorithm was created that took the RGB data and transformed each pixel array into a single 'luminance' floating point number to create a scaled black and white version of each image matrix.

To switch between colour scales, I used the Luminance Formula (ITU-R, 2011, p. 3):

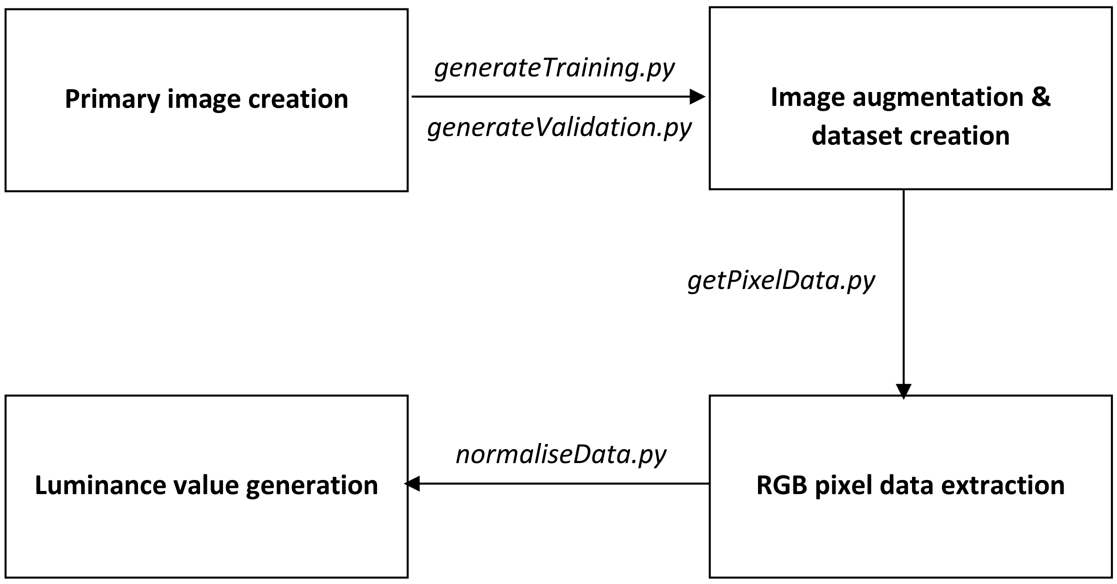
$$E'_{\gamma} = 0.299E'_{R} + 0.587E'_{G} + 0.114E'_{B}$$

Where E'_R , E'_G & E'_B are the values stored in each array relating to the amount of each colour (red, green & blue) is in the pixel on a scale of 0 – 255. The luminance float, E'_{γ} , has an identical range and represents the colour array as a weighted ratio of its elements.

The luminance values for each image are added to a data frame and then pushed to a new combined-CSV (1.3 TB). Each separate row of the CSV contains the luminance values of a single image after execution. The rows can then be decoded by the Machine Vision models to compute the raw pixel data.

The transformation code can be found in the GitHub repository (Pixel Transformation/getPixelValues.py & normaliseData.py).

A high-level overview of the data collection process is shown below:



4.5 – Image Compression

Loading the 1.3TB pixel data for use in the Logistic Regression algorithm, caused challenges for my computing platform. I tried several methods of processing the luminance values stored in the CSV file, for example: reading it into a *pandas* data frame; reading it in as partitions; and using *numpy* arrays. All these attempts were hindered by a combination of processing speed, RAM capacity and USB read/write speeds.

The solution to this bottleneck was to use "lossy compression". I designed and implemented a compression algorithm that takes the size of each row (pixels per image) and shrinks it. It works by taking a parameter representing the number of pixels that each image should have, calculating a ratio between said parameter and the current resolution and scaling the width and height accordingly (lowerResolution.py).

The algorithm reduced the size of each by a factor of 4 whilst retaining the image's core shape and features.

At this point, the data was small enough to compute, but was still taking an impractical amount of time to access. To solve this, I moved the dataset from the external HDD onto an SD card. Furthermore, when I come to implement the Machine Vision models, I'll read the data in parallel, distributing the load over all the CPU's cores as opposed to computing sequentially.

5.0 – Logistic Regression

This chapter details how I implemented a Logistic Regression algorithm from fundamentals and how it recognises shapes as well as a high-level dive into the mathematical foundations on which the model lies. The source code is found at '*Logistic Regression/logisticRegression.py*' in the ICARUS repository.

5.1 – Algorithm Design

The algorithm can be broken down into smaller components – each component representing a function with a specific purpose. An abstracted flow diagram of the system is shown below:

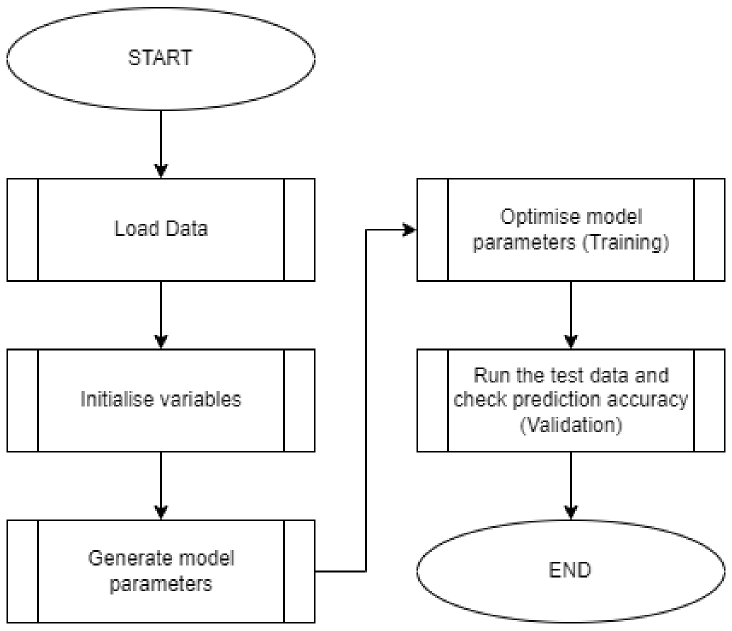

START
Load Data
Initialise variables
Generate model parameters
Optimise model parameters (Training)
Run the test data and check prediction accuracy (Validation)
END

Figure 15 - Flow Chart representing an abstraction of the Logistic Regression algorithm

The first step is loading in the data. After applying my compression algorithm, the data is small enough to load into a *pandas* data frame. Even though it is stored on an SD, the size of the data is too large to feasibly read it in serially. To overcome this, I have imported the *dask* python library. Using *dask*, the CSV data is split into partitions. Each partition is then read into a *dask* data frame simultaneously with a processor thread designated to each partition. Once all the threads have completed their partition computations, each populated *dask* data frame is appended to a single *numpy* array with the line daskDataFrame.compute().

The pixel data's corresponding labels are held in separate xlsx files in the same directory. There is one file for the training data's labels and one for the validation data. Each file has five sheets – one for each shape. Each shape's label data is stored in the format: 1 if the corresponding image is the shape in question, and 0 if it is not. An xlsx sheet is read in as a *pandas* data frame depending on which shape is being trained/validated at the time.

The program initialises the variables that will be used in the training steps once all the external data has been loaded into python data structures. These are *weights* and *bias*. *Weights* is a *numpy* array with the same number of elements as there are images in the data set and are all set to zero initially. These values will change throughout the training process and hopefully make the predictions more accurate. The other initialised variable, *bias*, is an int set to zero. This will keep track of the error in each iteration of the *weights* optimisation and apply a *bias* to each calculation to account for it.

After configuration, the next step is to train the model.

5.2 – Algorithm Implementation

The training is done primarily over two functions: *regress* and *optimiseParams*. The first function forward-propagates through the training data, calculating a predicted output for each image. This is done using the sigmoid logistic function in the equations:

$$h_{\theta}(x) = \frac{1}{1 + e^{-z}}$$

Given that:

$$z = w^T x + b$$

Where w^T is the transposition of the weights array, x is the training set array, and b is the current bias value.

The weights must be transposed so that the shape is compatible with the training data array. This means the matrices can be multiplied together.

This prediction value is then used to calculate the cost function. The cost function represents how much an effect on the final prediction an incorrect weight could have. The equation for calculating this is:

$$J(\theta) = \frac{-1}{m} \sum_{i=1}^{m} -y \log\big(h_{\theta}(x)\big) - (1 - y) \log(1 - h_{\theta}(x))$$

Where m is the number of images in the dataset, and y is the array of labels (Pant, 2019).

The model's aim is to undergo 'gradient descent' and lower the cost function as much as possible. According to (Ruder, 2017), "Gradient descent is a way to minimise an objective function $J(\Theta)$ parameterised by a model's parameters $\theta \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the gradient of the objective function $\nabla_{\theta} J(\theta)$ to the parameters." This method is known as 'batch gradient descent' as the gradient of the cost function is computed for every image in the dataset.

To find the gradients of the weights and the bias, the formula can be derived through the partial differentiation of the cost function:

(1) Re-write logistic cost function in terms of w and b

$$J(w,b) = \frac{-1}{m}\sum_{i=1}^{m} -y\log\big(h_\theta(w^Tx+b)\big) - (1-y)\log(1-h_\theta(w^Tx+b))$$

(2) Solve for the partial derivative with respect to w

$$\begin{aligned}\frac{\partial J(w,b)}{\partial w} &= \frac{\partial J(w,b)}{\partial h_\theta(w^Tx+b)} * \frac{\partial h_\theta(w^Tx+b)}{\partial w}\\ &= \frac{1}{m} * \frac{\partial J(w,b)}{\partial h_\theta(w^Tx+b)} * \frac{\partial h_\theta(w^Tx+b)}{\partial(w^Tx+b)} * \frac{\partial(w^Tx+b)}{\partial w}\end{aligned}$$

$$\frac{\partial J(w,b)}{\partial h_\theta(w^Tx+b)} = \frac{-y}{h_\theta(w^Tx+b)} + \frac{1-y}{1-h_\theta(w^Tx+b)}$$

$$\frac{\partial h_\theta(w^Tx+b)}{\partial(w^Tx+b)} = (1+e^{-(w^Tx+b)})^{-1} = h_\theta(w^Tx+b) * (1-h_\theta(w^Tx+b))$$

$$\frac{\partial(w^Tx+b)}{\partial w} = x + 0 = x$$

$$\therefore$$

$$\begin{aligned}\frac{\partial J(w,b)}{\partial w} &= \left(\frac{-y}{h_\theta(w^Tx+b)} + \frac{1-y}{1-h_\theta(w^Tx+b)}\right) * \Big(h_\theta(w^Tx+b) * \big(1-h_\theta(w^Tx+b)\big)\Big) * x\\ &= \frac{1}{m}x((w^Tx+b)-y)\end{aligned}$$

(3) Solve for the partial derivative with respect to b

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m} * \frac{\partial J(w,b)}{\partial h_\theta(w^Tx+b)} * \frac{\partial h_\theta(w^Tx+b)}{\partial(w^Tx+b)} * \frac{\partial(w^Tx+b)}{\partial b}$$

$$\frac{\partial(w^Tx+b)}{\partial b} = 1$$

$$\therefore$$

$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}((w^Tx+b)-y)$$

The full calculation for $\frac{\partial h_\theta(w^Tx+b)}{\partial(w^Tx+b)}$ can be found in the appendix (A.1).

Using these partial derivatives, the logistic regression model calculates the gradients for weights and bias on every iteration (epoch). *optimiseParams* runs *regress* several times equal to a user's input *numIterations*. After each iteration of *regress*, the weights and bias are updated using the returned gradients and the hyperparameter 'learningRate'. By the end of the loop, the weights and bias will be as optimised as possible. These final values will be used when making a prediction on previously unseen data (validation set).

The structure of the logic and the code flow for using the trained regression model is identical but instead of feeding in the training set to calculate the optimal parameters, the validation set, and its corresponding label data is passed into a new function *makePrediction*.

This function works by multiplying the new optimised weights matrix with the validation pixel data matrix, adding the bias value, and returning an integer 0 or 1, depending on the output of the prediction calculation:

$$\rho = \begin{cases} 1 & if\ h_\theta(w^T\varphi + b) > 0.5 \\ 0 & if\ h_\theta(w^T\varphi + b) \leq 0.5 \end{cases}$$

Where ρ is the numerical prediction and φ is the validation data matrix.

For each validation image, the numerical prediction is a value between 0 and 1, with 0 meaning the algorithm is 100% certain that the image being checked is not the shape entered as a parameter, and 1 indicating 100% certainty that the image is of the entered shape. The function will return a prediction of 1 (is shape) if ρ is greater than 0.5, and 0 (is not shape) if ρ is less than or equal to 0.5. Once a prediction is made, it is checked against the shape's actual label – if the prediction is correct, a counter is incremented. Once the validation set has been exhausted, the total number of correct predictions is divided through the total number of images in the set and multiplied by 100 to give the model's final 'accuracy' as a percentage.

5.3 – Tuning Hyperparameters

To train the Logistic Regression model as efficiently as possible and attain the highest possible accuracy, two hyperparameters are used to adjust or accelerate parts of the computation. *numIterations* represents the amount of training epochs the model performs and therefore, the number of times the weights and bias are changed. The *learningRate* is a measure of how fast the algorithm should 'learn'. The value assigned to *learningRate* is multiplied by the weights and bias gradients on every epoch.

To begin with I set *numIterations* and *learningRate* to 100 and 0.5 respectively. From here, I ran the training and validation datasets through the model, recorded the accuracy and modified the hyperparameters accordingly. Below are graphs plotting the change in accuracy against the change in hyperparameter values. The values tested for *numIterations* are 10, 100, 500, 1000 and 2000. The values tested for *learningRate* are 0.01, 0.05, 0.1 and 0.5. The *numIterations* graphs are shown on the left, and the *learningRate* graphs on the right.

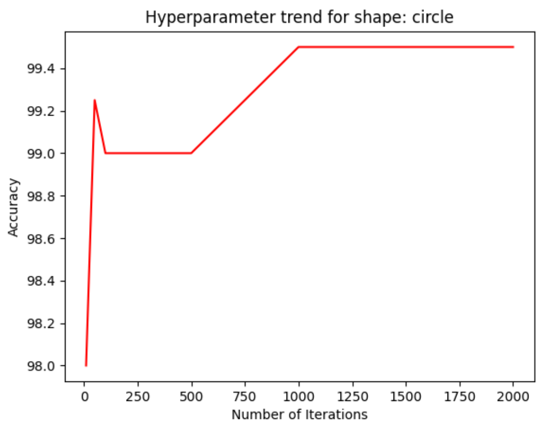


Figure 16 - Relationship between Accuracy and numIterations (Circle)

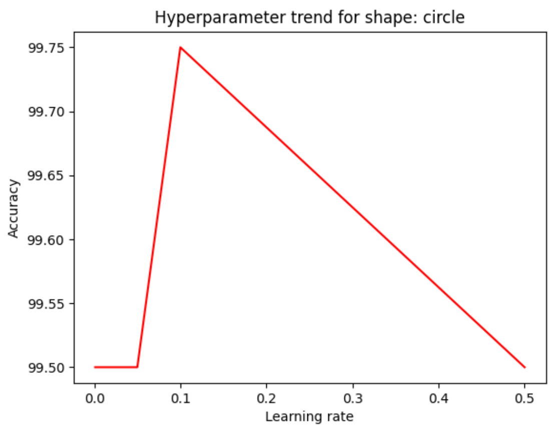


Figure 17 - Relationship between Accuracy and learningRate (Circle)

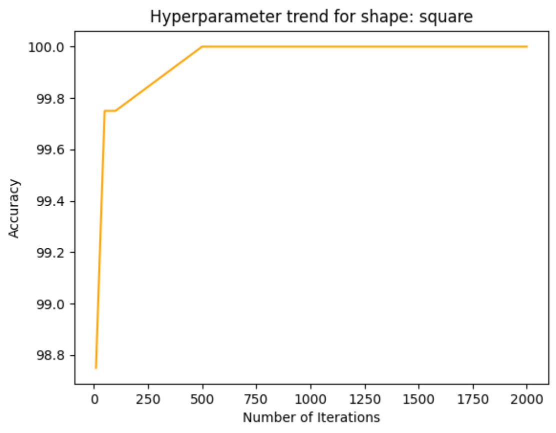


Figure 18 - Relationship between Accuracy and numIterations (Square)

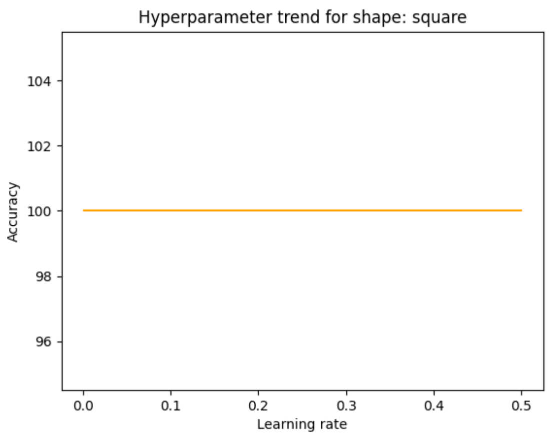


Figure 19 - Relationship between Accuracy and learningRate (Square)

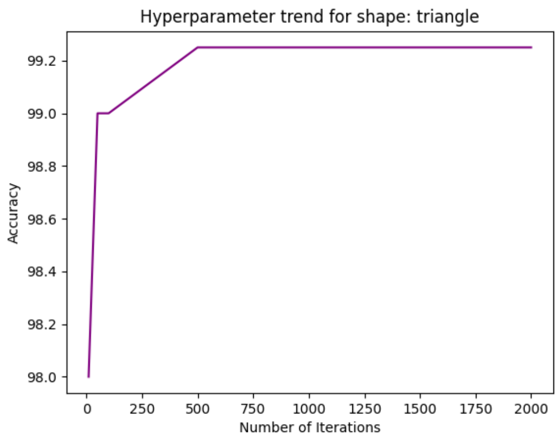


Figure 20 - Relationship between Accuracy and numIterations (Triangle)

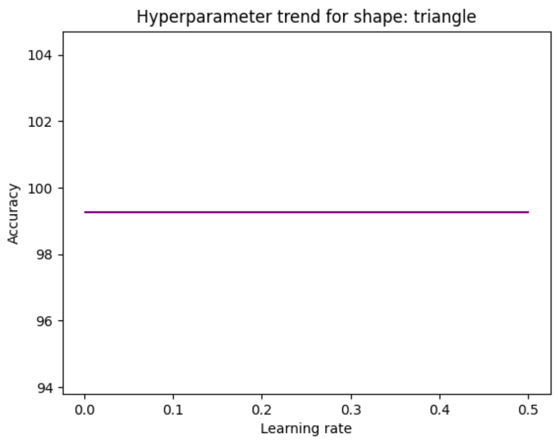


Figure 21 - Relationship between Accuracy and learningRate (Triangle)

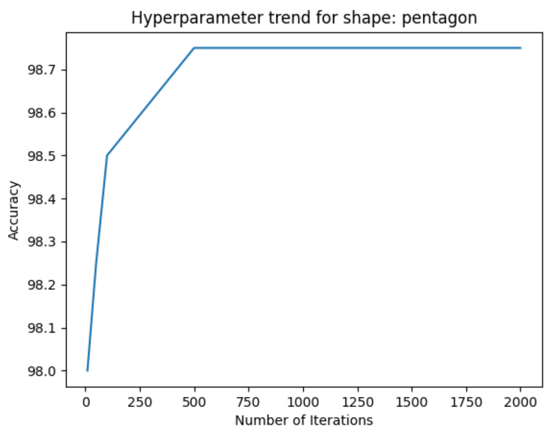


Figure 22 - Relationship between Accuracy and numIterations (Pentagon)

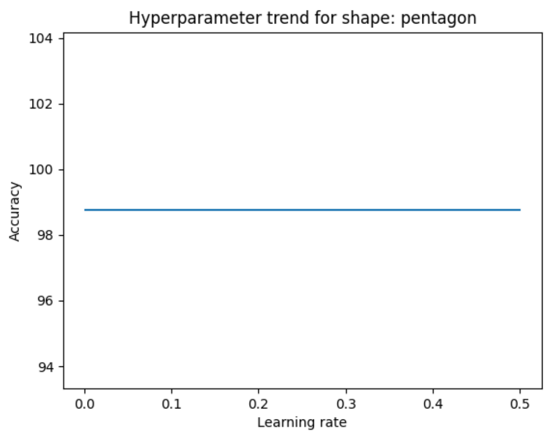


Figure 23 - Relationship between Accuracy and learningRate (Pentagon)

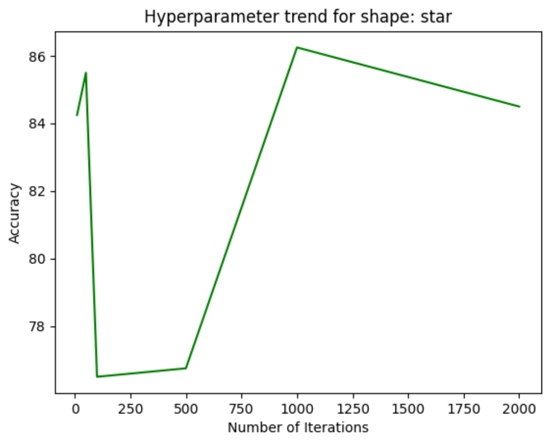


Figure 24 - Relationship between Accuracy and numIterations (Star)

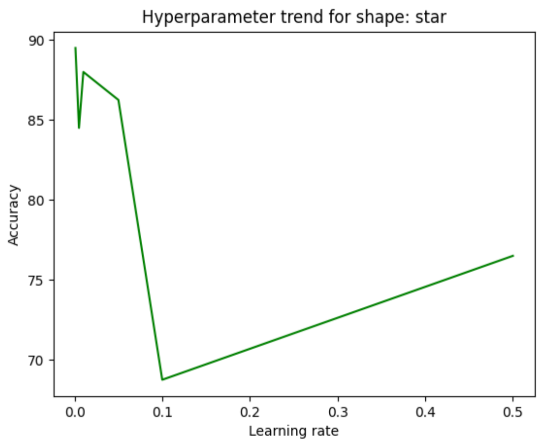


Figure 25 - Relationship between Accuracy and learningRate (Star)

In some instances the change in hyperparameter value had no effect on the final accuracy but, in others the difference was substantial. I took the hyperparameters for each shape that yielded the highest accuracy and re-ran the training. The table below illustrates the optimised hyperparameter values and the final accuracy for each shape.

Shape	**Number of Epochs**	**Learning Rate**	**Accuracy (%)**
Circle	1000	0.1	99.75
Square	500	0.01	100
Triangle	500	0.01	99.25
Pentagon	500	0.01	98.75
Star	1000	0.01	88

Table 1 - The optimum hyperparameter values and maximum accuracy for each shape

After optimisation of the weights and bias values and tuning of the hyperparameters, the resulting accuracies are very impressive – with most of the shapes hovering around the 99% value. The only exception to this is star, with an accuracy of 88%. Although this is still a respectable level of accuracy, it's significantly lower than the other shapes – this could be down to the training set being too small, the complexity of the shape, or a combination of both.

5.4 – Performance and Analysis

Examining the accuracy is a good method of seeing how many predictions were correct but isn't necessarily beneficial in finding out how well the Machine Learning model performs. At first, this may seem like a contradictory statement however, a large number of correct predictions may not translate to a high-performing algorithm. An example is if the model returns a prediction of 0 for every single image. In this case, the accuracy would be 80% because for any one shape, the validation labels file is 80% zeros – this clearly isn't a 'high-performing' model. To truly measure the performance of each model, I have analysed the sensitivity and specificity. Sensitivity is the measure of how many images of shape x were correctly identified as x, whereas specificity is the measure of how many images that are $!x$ were correctly identified (Kumar, 2022).

To calculate sensitivity and specificity, the results need to be portrayed as confusion matrices. The Matrices for each shape, post-validation, are shown below:

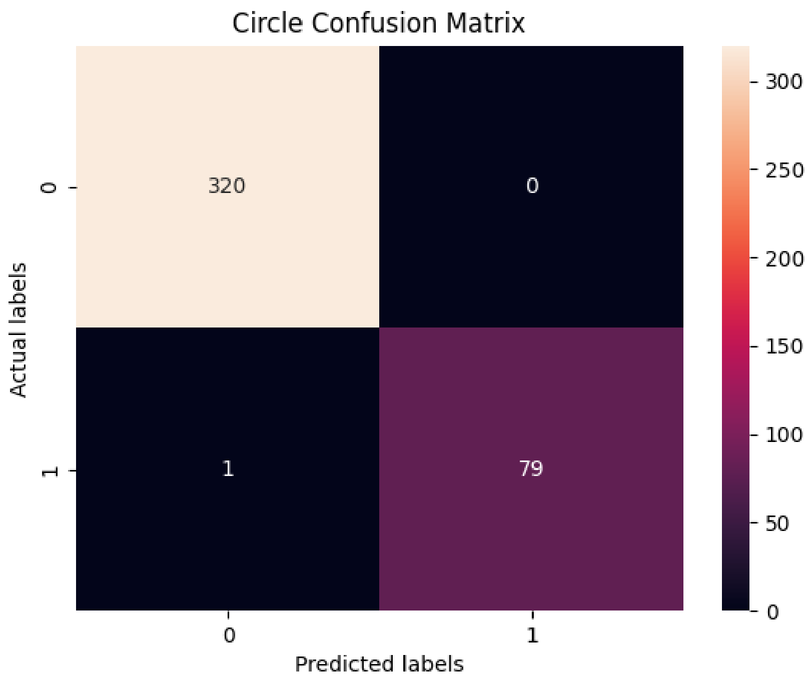


Figure 26 - Confusion Matrix for shape: Circle

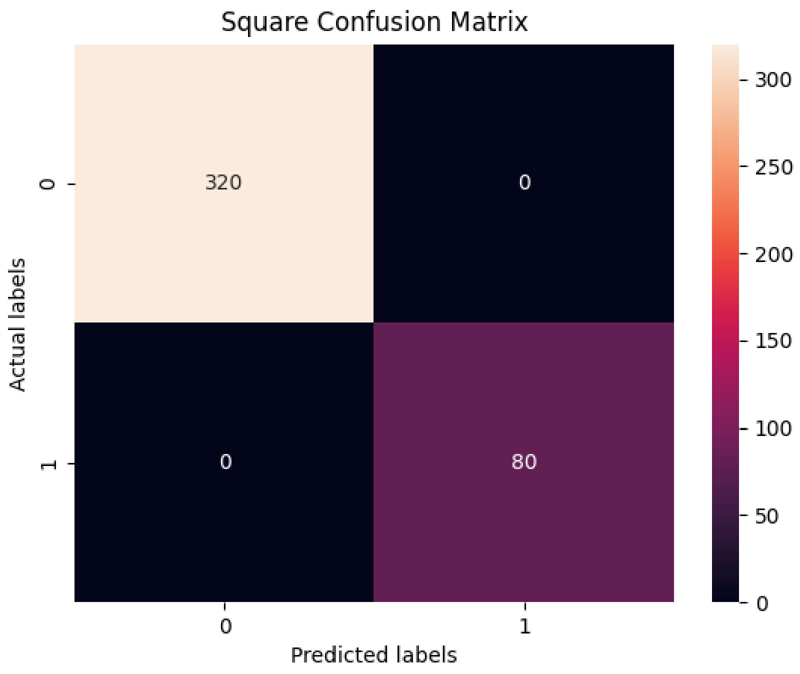


Figure 27 - Confusion Matrix for shape: Square

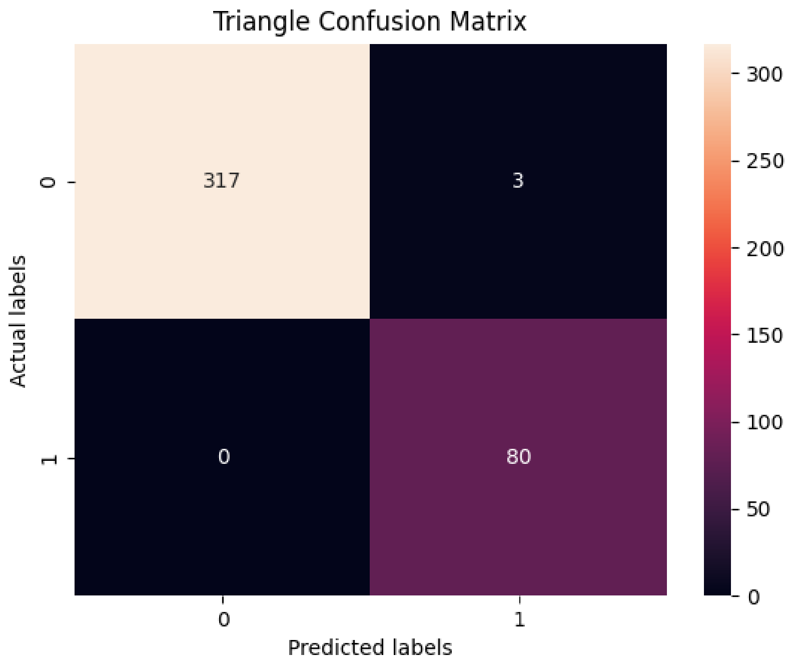


Figure 28 - Confusion Matrix for shape: Triangle

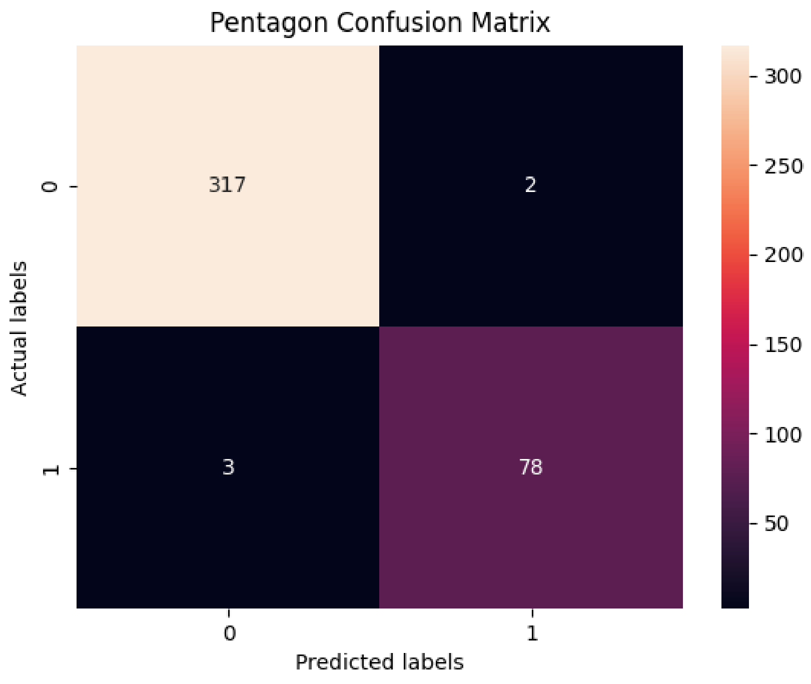


Figure 29 - Confusion Matrix for shape: Pentagon

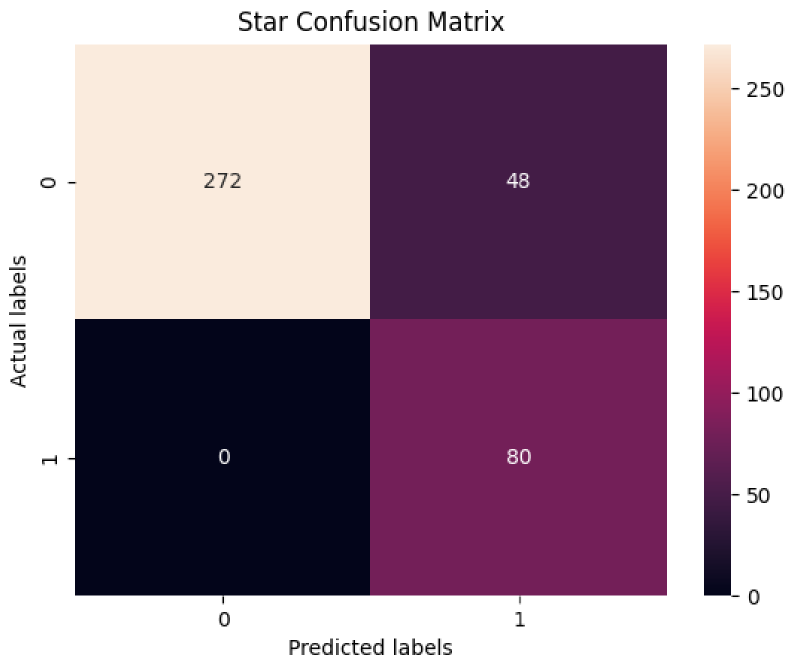


Figure 30 - Confusion Matrix for shape: Star

These Confusion Matrices work by decomposing the accuracy into four categories. The top left represents the number of images that have the label 0 and were predicted as such. This category is referred to as the *True Negatives*. The opposing, bottom right, category represents the *True Positives* and holds a value equal to the number of images that had a label 1 and were predicted as such. The top right and bottom left are the *False Positives* and the *False Negatives* respectively. These are the number of images that were predicted incorrectly as 1 and incorrectly as 0.

The sensitivity and specificity are calculated by:

$$sens = \frac{TP}{TP + FN}$$

$$spec = \frac{TN}{TN + FP}$$

Where, TP is the True Positive value, FN is the False Negative value, *TN* is the True Negative value, and FP is the False Negative value.

The sensitivity and specificity values for each shapes' confusion matrix is tabulated below:

Shape	Sensitivity (4 d.p.)	Specificity (4 d.p.)
Circle	0.9875	1.0000
Square	1.0000	1.0000
Triangle	1.0000	0.9906
Pentagon	0.9630	0.9937
Star	1.0000	0.8500

Table 2 - The Sensitivity and Specificity of each shape

These values prove that the Logistic Regression model is highly accurate, with triangles being the easiest to identify. The sensitivity is extremely high across the board, however the specificity for stars is 0.85. This means that 15% of the time, the model will classify a star as 'not star'. This is a low percentage but still sub-optimal.

To bring the ratios of correct to incorrect predictions together, a calculation can be carried out to generate an *F-Score* for each shape. According to (wood, T. 2019), "the F-Score is delineated as the harmonic mean of the model's precision and recall", and "[the F-Score] is a measure of a model's accuracy on a dataset".

The F-Score in term of confusion matrix categories is as follows:

$$F_1 = 2\frac{\left(\frac{TP}{TP + FP}\right)\left(\frac{TP}{TP + FN}\right)}{\left(\frac{TP}{TP + FP}\right) + \left(\frac{TP}{TP + FN}\right)}$$

The F-Scores for each shape are below:

Shape	**F_1-Score**
Circle	0.9937
Square	1.0000
Triangle	0.9816
Pentagon	0.9690
Star	0.7692

Table 3 - The F-Scores of each shape

These values will be used as the final 'accuracies' for Logistic Regression and the primary method for comparison against the Artificial Neural Network model.

6.0 – Convolutional Neural Network

6.1 – Algorithm Design

The purpose of this chapter is to explain the implementation of my Convolutional Neural Network (CNN), analyse its predictions and compare them to those of the Logistic Regression model. The algorithm's decomposition is shown below:

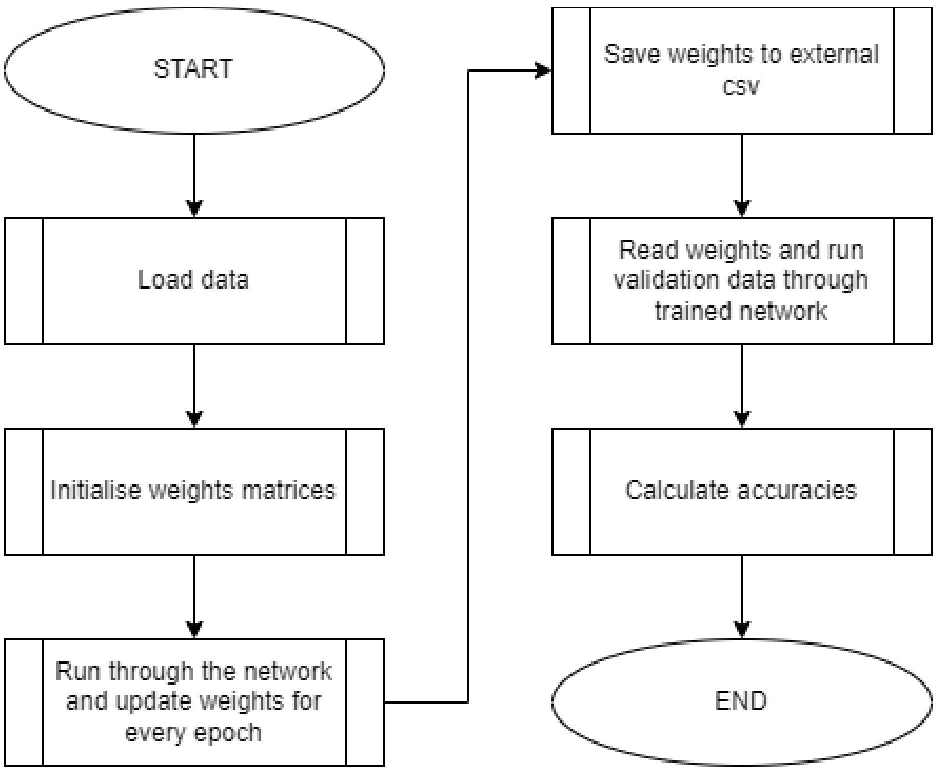

START
Load data
Initialise weights matrices
Run through the network and update weights for every epoch
Save weights to external csv
Read weights and run validation data through trained network
Calculate accuracies
END

Figure 31 - A Flow Chart showing an abstracted view of the CNN algorithm

6.2 – Algorithm Implementation

Akin to the Logistic Regression algorithm, the CNN starts by reading in the training and validation data CSV files as well as their respective label xlsx files. The algorithm then initialises the weights matrices. The neural network has a much greater number of weights than the regression model. This is down to the number of nodes that make up the network. The basic mesh structure of the CNN consists of 4899 input nodes (one node per luminance pixel), 1000 hidden nodes, and 5 output nodes. As a result, there are 4.9E+06 connections (weights) between the input layer and the hidden layer and 5000 connections between the hidden layer and the output layer. Overall, the weights mesh is roughly 1200 times the size of the regression models. However, the CNN's weights mesh is designed for predicting all shapes, whereas the logistic regression mesh is designed for a single shape at a time. Accounting for this, each shape in the CNN has 240 times more weights in its mesh than its regression equivalent.

There are five output nodes to represent the five possible shapes. Each output will be a probability, between zero and one, that the shape with the respective index in the array (circle, pentagon, square, star, triangle) is the correct shape. The highest value will be taken as the network's prediction.

Once the network's architecture and the weight matrices have been configured, it is trained on the training set with a configurable learning rate and number of epochs. The model trains itself by calculating an error and utilising backpropagation techniques to optimise the weights. The weights are saved to external CSVs once they are optimised so that the network doesn't have to be re-trained every time a hyperparameter is modified.

Finally, for the validation step, the weights are read back into the model as numpy arrays, and the test data is run through the network using a target array generated by the validation labels to check the model's accuracy.

6.3 – Optimising weights with backpropagation

In neuroscience, there exists a term: neuroplasticity. According to (Mateos-Aparicio & Rodriguez-Moreno, 2019), 'neuroplasticity' can be defined as "the ability of the nervous system to change its activity in response to intrinsic or extrinsic stimuli by reorganising its structure, functions or connections."

To mould the weights into their optimal values, backpropagation is used to continually reduce the difference in the desired output and the actual output. This gradient is known as the 'error'. In context of neuroplasticity, the error on each output node acts as the system's intrinsic stimuli. As a response to this stimulus, the network reorganises its connections by modifying each weight – starting at the connection between the first output node and the first hidden node and ending with the connection between the last hidden node and the last input node. The error's effect on each individual node ripples, or 'back-propagates', through the entire network until every weight has been modified.

In Logistic Regression, a batch gradient descent technique was used for calculating gradients, for the neural network's learning process, I opted for a stochastic gradient descent approach instead as it is more suited for learning one example at a time (Bottou, 2012). The explanation of gradient descent in chapter 5.1 holds true in this case as well, with the caveat of a slightly more complex expression:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta, x^{i}, y^{i})$$

Essentially, this formula performs a parameter θ update for each training example x^{i} and label y^{i} (Ruder, 2017).

The derivation for the weight modification equation is shown below:

(1) Use the chain rule to form an expression for $\frac{\partial J}{\partial w_2}$:

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial h_2(x)} * \frac{\partial h_2(x)}{\partial z_2} * \frac{\partial z_2}{\partial w_2}$$

(2) Solve partial derivatives:

$$\frac{\partial J}{\partial h_2(x)} = \frac{-y}{\partial h_2(x)} + \frac{1-y}{1-h_2(x)} = \frac{h_2(x)-y}{h_2(x)\big(1-h_2(x)\big)}$$

$$\frac{\partial h_2(x)}{\partial z_2} = \frac{\partial}{\partial z_2}[(1+e^{-z_2})^{-1}] = \frac{e^{-z_2}}{(1+e^{-z_2})^2} = h_2(x)\big(1-h_2(x)\big)$$

$$\frac{\partial z_2}{\partial w_2} = h_1(x)$$

(3) Combination of parts:

$$\frac{\partial J}{\partial w_2} = \frac{h_2(x)-y}{h_2(x)\big(1-h_2(x)\big)} + h_2(x)\big(1-h_2(x)\big) + h_1(x)$$

$$\therefore$$

$$\frac{\partial J}{\partial w_2} = h_1(x)(h_2(x)-y)$$

(4) Rearrange for $h_1(x)$:

$$h_1(x) = \frac{\partial J}{\partial w_2} * \frac{1}{(h_2(x)-y)}$$

It's important to note that this equation only holds true when calculating the weights in the second connection mesh (hidden node to output node). To calculate the weights in the first mesh the equation is done twice in order to account for the change in error and the changes in the second weights matrix.

6.4 – Tuning Hyperparameters

It was clear that this algorithm was performing well from the very first validation results, with accuracies in the mid-90s for most shapes. However, with tuned hyperparameters, the performance could only increase.

The algorithm was tested with a constant number of hidden nodes and epochs and a variable learning rate. The results for the different learning rates are shown in the graph below:

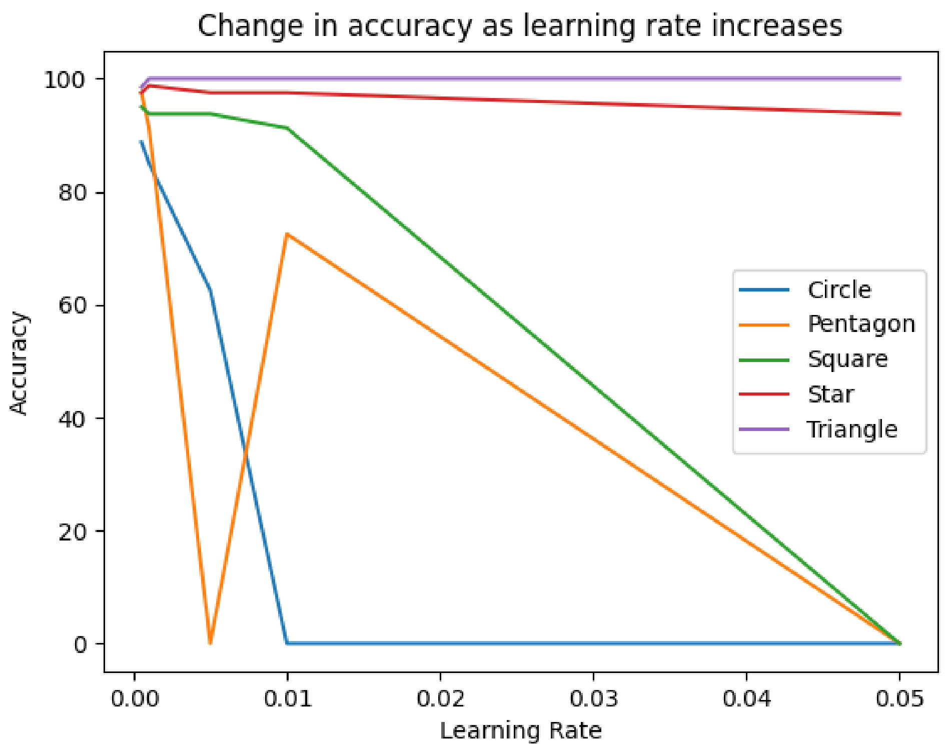


Figure 32 - Relationship between Accuracy and learningRate for each shape

Except for a single point on the pentagon line, the graph shows a strong negative correlation between the learning rate and accuracy for the shapes of circle, pentagon and square, as well as weak negative correlation for star. A confident conclusion can be made from this graph, that increasing the learning rate has a drastic, negative effect on the model's prediction accuracy. A possible reason for this trend could be the size of the training set. The model is processing a great amount of data and therefore can afford to 'learn' at a slower rate. The larger learning rates may be forcing the model to learn too fast, resulting a sub-optimal set of weights and an unstable training process (Brownlee, 2019).

To find the optmum number of epochs for training, I took inspiration from transfer learning. After running the model for 25 epochs, the weights were saved to external CSV files. For the next training cycle the data was processed another 25 times but with the saved weights as the initial matrices. This meant that by the end the network had been trained for 50 epochs even though it only took up 25 epochs worth of time. This was repeated multiple times until the highest performing hyperparamter became clear: 100 epochs.

6.5 – Performance and Analysis

The previous hyperparameter graph and the following confusion matrix represent the data cumulatively, as opposed to the LR’s separate graphics, because the neural network returns a continuous prediction rather than a discrete one. This means the CNN can be tested with all shapes at once.

The Confusion Matrix representing the highest performance achieved with tuned hyperparameters is shown below:

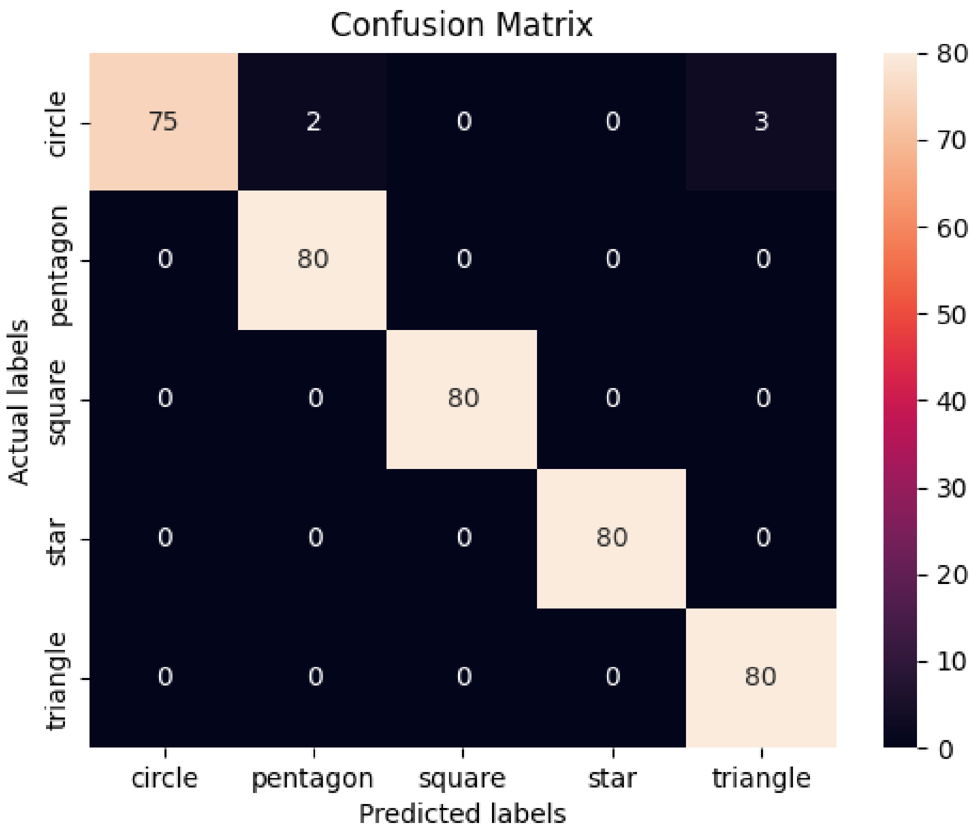


Figure 33 - Confusion Matrix for the CNN Model

The model performed extremely well with only circle not being predicted 100% accurately. The sensitivity, specificity and the respective F-scores of each shape are tabulated below:

Shape	**Sensitivity**	**Specificity**	**F_1-Score**
Circle	0.9375	1.000	0.9677
Pentagon	1.0000	0.9756	0.9877
Square	1.0000	1.0000	1.0000
Star	1.0000	1.0000	1.0000
Triangle	1.0000	0.9639	0.9812

Table 4 - Sensitivity, Specificity and F-Score for each shape

In every case, the Neural Network outperforms the LR model. On top of this, the CNN is substantially quicker to recognise shapes due to the weights being saved externally and loaded in at runtime rather than having to be regenerated every time.

There is little doubt that the CNN is the better of the two Machine Vison algorithms in this context. This will be the model used in the simulation.

These results are better than those produced by (Çelik., et al. 2013) and (Zhang., et al. 2022), which were 98% and 79.5% respectively. However, the forest fire application of (Zhang., et al. 2022) network has a substantially more difficult pattern to recognise. On the other hand they also had a lot more data at their disposal.

7.0 – Constructing the System

The below sections outline how the data creation, compression and conversion scripts coalesce with the Neural network and a random coordinate allocation function to form a complete system.

7.1 – Generating an Environment

I needed data to represent before loading my system into a graphical simulation. The combined system begins by calling the data creation and pre-processing code to generate 25 instances of shapes. These shapes are then converted into their equivalent luminance arrays and fed into the CNN. The network loads the weights file and makes a prediction for each of the 25 shapes. These shapes are then assigned a random coordinate tuple within a 40m x 40m square (see A.2), with centre point at the middle of London Heathrow's northern most runway (51.47767, -0.45971). I chose this location as the test area because the runway markings make it easy to see the UAV's movements. Also, it's quite apt for an aerial vehicle to be tested (virtually) at an airport.

An example 2D environment abstraction is shown on the next page. Each shape has been 'recognised' as such from its corresponding generated image, using the CNN. The environment is an abstraction because it does not display the satellite imagery that will be present in the simulation, nor the altitude of each shape.

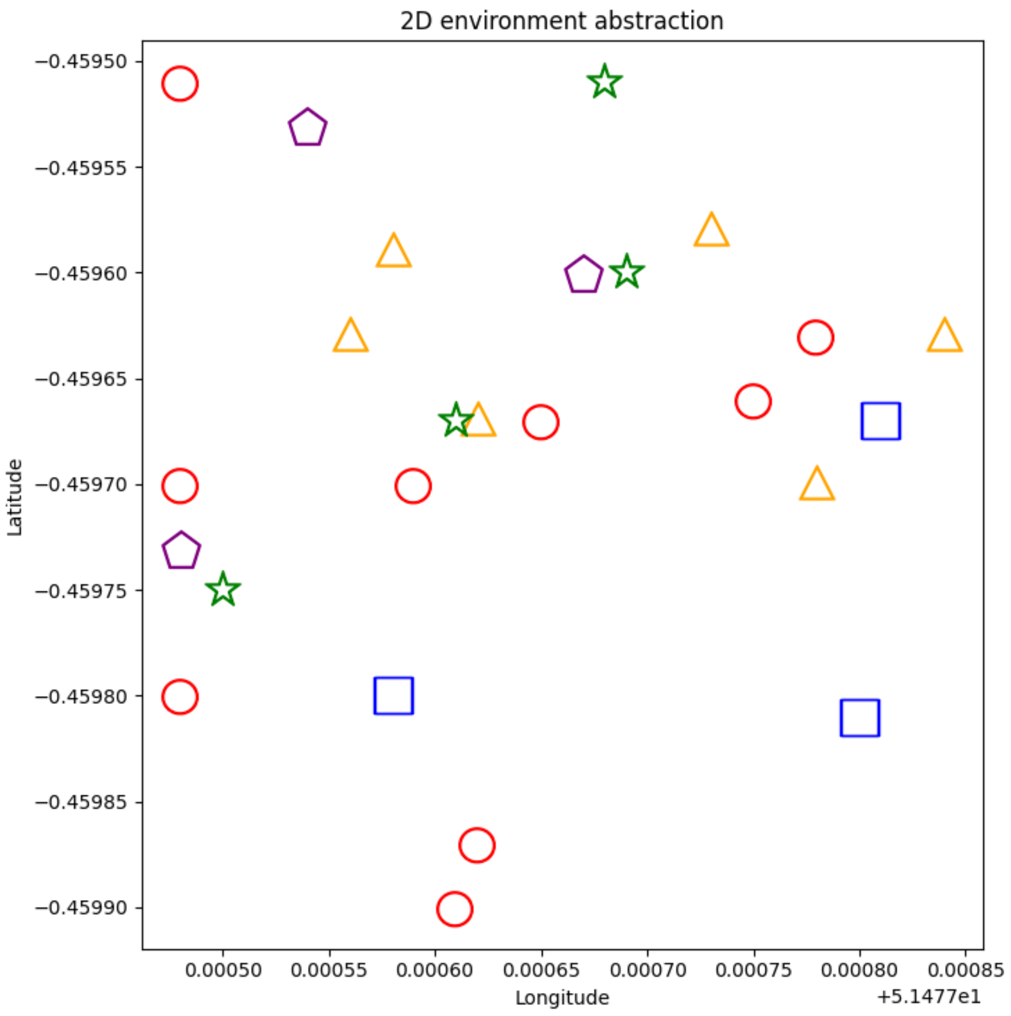


Figure 34 - A randomly generated environment using matplotlib

7.2 – Heuristics

Now that the system generates a 2D environment and populates it with shapes, the UAV needs a path to follow so that it can fly through all the instances of a stated shape. I had two criteria when it came to selecting a heuristic algorithm. Firstly, it had to successfully find an efficient path through all the desired shapes, and secondly, it had to calculate the solution quickly because the simulation runs in real-time.

I investigated two well established heuristic algorithms: Dijkstra and Kruskal's Minimum Spanning Tree (MST). Both do a good job at finding an efficient route, but MST is potentially a lot quicker at doing so due to it utilising a priority queue (Ravikiran, 2023). It has a complexity of $O(ElogV)$ where E is the number of edges; and v, the vertices. Dijkstra has a complexity of $O(V^2)$.

I decided to implement an MST algorithm. While this doesn't always produce the absolute optimal path, it always produces the local optimal route from any given node. This is a minor trade-off for the gained benefits of its speed. An example of MST being used on a generated environment is shown on the next page.

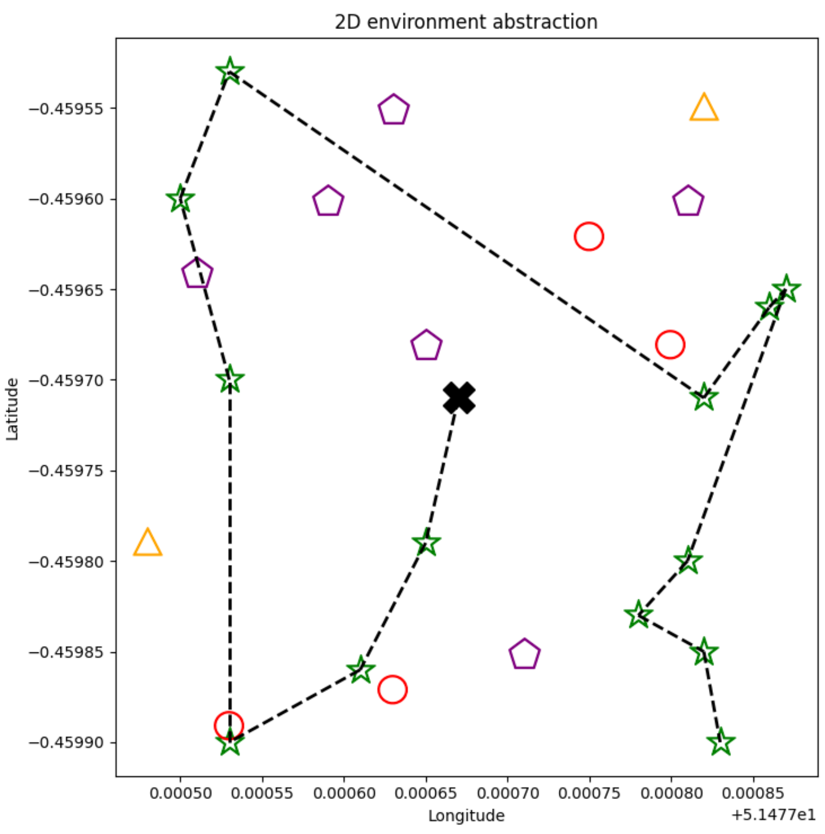


Figure 35 - A generated environment with a path connecting all instances of 'star'

7.3 – Software-in-the-loop path visualisation

The final stage of this project was to transfer the calculated path into the simulation. The simulation I used was an open-source, community-maintained piece of software called *Mission Planner* (ArduPilot, 2010). This, along with the python API *dronekit-sitl*, allowed me to form a TCP connection between my device and the virtual UAV (known as 'copter').

The simulation compatibility code is in '*Path Finding/pathfinding.py*' in the git repository. The code's construction is made up of three core blocks. The first of these connects to the copter using a specialist protocol known as Micro Air Vehicle Link (MAVLink). This protocol has the benefit of being extremely efficient with just eight bytes overhead per packet (MAVLink, 2009). Once a connection is made, the program 'arms' the copter by initialising all the sensors and motors. It also sets the copter's mode to 'guided' to prepare it to receive movement commands.

The second software-in-the-loop (SITL) part of the program sends a 'take off' command to the copter. A while loop is used to consistently verify the UAV's altitude until it reaches a pre-defined value. Once the copter is stable at the configured altitude, the script breaks out the loop, leaving the copter to hover (the simulation has self-stabilisation built into it).

Finally, the program creates several 'goto' commands with waypoints equal to the coordinates of each specified shape (e.g., circle), in the order in which the MST heuristic identified. Each 'goto' command is then processed sequentially allowing the copter to travel the same route as defined in the 2D matplotlib environment. Once the last waypoint is reached, the copter is ordered to land and the MAVLink connection is terminated.

Below are the matplotlib and Mission Planner environments generated from a single execution of *pathfinding.py*. The black 'X' and the green marker represent the 'home' (starting) location for the environment and simulation respectively:

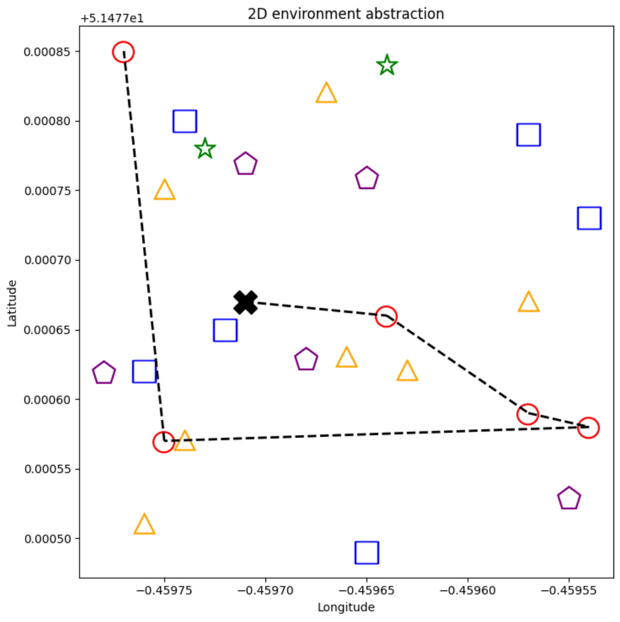


Figure 36 - Matplotlib representation of the desired path

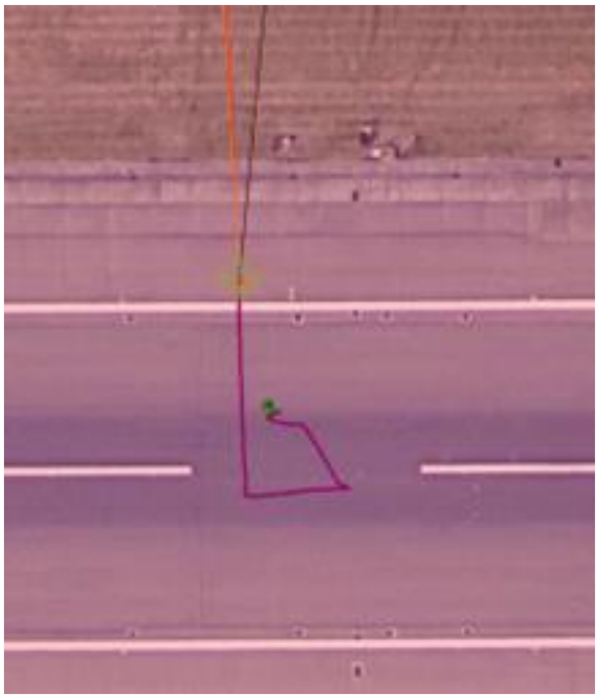


Figure 37 - A zoomed-in screenshot of the path followed by the copter.

The UAV in the simulation follows the path calculated in the matplotlib environment.

The complete simulator UI is shown below:

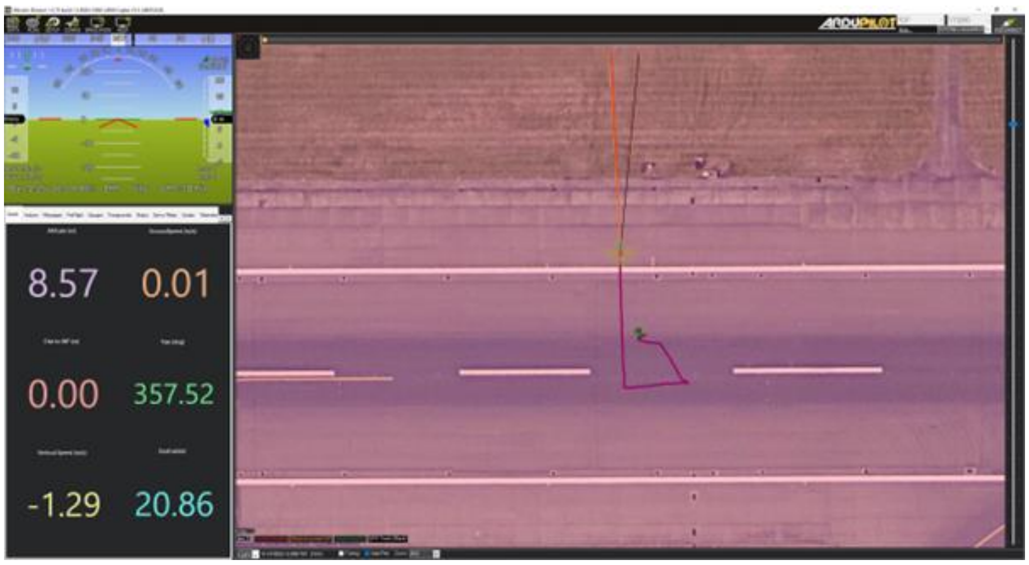


Figure 38 - A complete view of the Mission Planner UI

The left panel shows various information such as current altitude, the distance to the next waypoint and the pitch/roll/yaw angles in real-time.

To show that the drone can be placed anywhere, here is a screenshot of the copter over Coventry University's Engineering, Environment and Computing building:

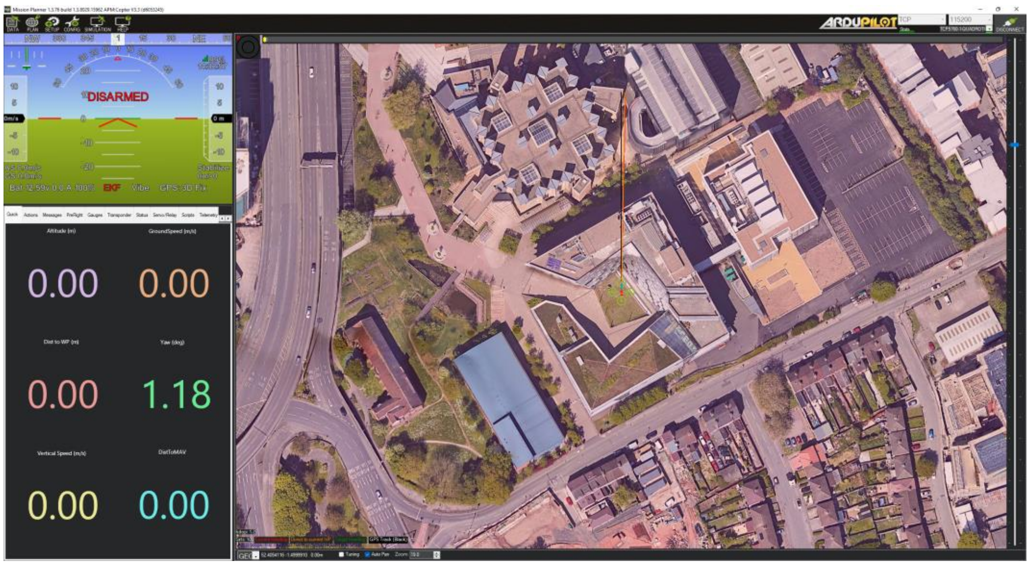


Figure 39 - A simulated copter over EEC

To see the environment generation, heuristic calculation, and simulated path following in real-time, see the following 2:39 video:

https://www.youtube.com/watch?v=wzYtEAfxoeE&ab_chCNNel=DanChalmers

8.0 – Conclusion

8.1 – Summary of Findings

To summarise this report: Not only is a backpropagation approach for Convolutional Neural Networks comparable in performance to that of Logistic Regression, but it out-performs it on every analysed metric. The CNN was faster and more accurate, although it took longer to train and tune. This high-performing Machine Vision model, integrated with the fast Minimum Spanning Tree heuristic algorithm and the efficient MAVLink connection protocol, made for a UAS that can:

- generate, augment, compress and convert images
- predict each image with nearly 100% accuracy
- generate an environment around these predictions
- plan an efficient route through them
- run a simulation to visualise said route

All in less than three minutes.

All the objectives stated in the introduction have been met in full over the cause of this project.

8.2 – Project Limitations

While this project succeeded in answering its question, some aspects could have been improved or made more efficient.

When training the models, the amount of data that could be used was limited by the storage I had available and the amount of training time was limited by the constraints of the project's sprints. Had these factors not been an issue, I'm confident that both LR and CNN models would have been even better performing than they currently are.

The host machine's hardware also played a part in slowing down the training and visualisation. The project was developed on a laptop fitted with an 8th generation i7 CPU, GTX 1080 GPU, and 8GB RAM. Although these specifications make for a device that is far from slow, any improvement on one or more of the components could have shaved hours off the image creation and model training times.

8.3 – Legal and Ethical Considerations

There are many ethical considerations when it comes to developing software for UAVs. Although this project serves as a proof of concept for using Machine Vision and automation to fly a drone, many people may see the application for this technology in the defence sector. While this is true, a single, arguably unethical, use case should not be a reason to hinder the development of such a technology. Automated 'intelligent' UASs have been, and will continue to be, vital assets in situations such as disaster management and agriculture.

It's also important to note the seemingly negative public perception of Artificial Intelligence due, in part, to modern media representations. Development and research into AI should not be restrained as I firmly believe we have more to gain than lose.

8.4 – Potential Project Extensions

There are a few ways in which this project can be continued and expanded upon. One of these is to extend the simulator UI to include more details, more readings and visual waypoints. Potentially the code base could be altered to connect to an actual UAV and for it to follow a route in real-life. In theory, changing the application to an actual quadcopter rather than a simulated one shouldn't require many changes – the MAVLink protocol works with both simulated and physical vehicles.

Another potential expansion could be to train the network on real-world objects such as a particular animal, rather than generic shapes. This would open the door to further optimisation and tuning on the CNN and more specific applications such as searching for endangered species from the air. This can be done by using the data creation and Machine Vision code files to generate new training sets and a new neural network.

Finally, a feature that I wanted to include in this project but would have been too resource-heavy to implement given the time constraints: swarming. The project could be expanded to consider multiple drones pathing simultaneously and they could work together and communicate to create a distributed aerial network. The Mission Planner simulator can have multiple nodes (copters) connected to it at any one time, each running its own instance of the navigation script in real-time.

8.5 – Reflection

Overall, I'm really pleased with how well this project has gone. Seeing the little lime-green quadcopter moving across a virtual runway whilst knowing how much computation was going on in the background for it to achieve this, is a very good feeling.

The performance of both Machine Vision algorithms exceeded my expectations, especially as everything from the gradient descent calculations to the hyperparameter tuning was done by hand, from fundamentals.

The other aspects of this project that I thought went very well were the primary data collection and the project management. The use of Jira and GitHub were crucial to organising my time and resources. I'm certain that I wouldn't have been able to get everything completed to a high standard in the allotted time without being able to visualise my current tasks and where I was in development.

If I could change anything about this work, it would be to calculate the actual size of the post-processing data sets before generating them. As a result of not doing this, I was left with the unforeseen task of having to compress all the data in a way that was equally legible but of a size that wouldn't cause RAM capacity issues. This also meant that I had to reorganise my tasks.

References

Abdi, H. (2007). *The Eigen-decomposition: Eigenvalues and Eigenvectors.* Encyclopaedia of Measurement and Statistics
https://personal.utdallas.edu/~herve/Abdi-EVD2007-pretty.pdf

Alves, T., Oliveria, C., Sanin, C., & Szczerbicki, E. (2018). *From Knowledge based Vision Systems to Cognitive Vision Systems: A Review*
https://www.sciencedirect.com/science/article/pii/S1877050918313553?via%3Dihub

Ajit, A., Acharya., & Samanta, A. (2020). *A Review of Convolutional Neural Networks.* 2020 International Conference on Emerging Trends in Information Technology and Engineering (ic-ETITE)
https://ieeexplore.ieee.org/document/9077735

Alzahrani, B., Oubbati, O., Barnawi, A., Atiquzzaman, M., & Alghazzawi, D. (2020). *UAV Assistance paradigm: State-of-the-art in applications and challenges.* Journal of Network and Computer Applications (Vol166)
https://www.sciencedirect.com/science/article/pii/S1084804520301806?via%3Dihub

ArduPilot. (2010). *Mission Planner*
https://ardupilot.org/plCNNer/

Atlassian.com (2002). *What is Jira used for?*
https://www.atlassian.com/software/jira/guides/use-cases/what-is-jira-used-for

Bertsimas, D., & King, A. (2017). *Logistic Regression: From Art to Science*
https://projecteuclid.org/journals/statistical-science/volume-32/issue-3/Logistic-Regression-From-Art-to-Science/10.1214/16-STS602.full

Bottou, L. (2012). *Stochastic Gradient Descent Tricks.* Neural Network: Tricks of the Trade
https://link.springer.com/chapter/10.1007/978-3-642-35289-8_25

Bhoi, A., Mallick, P., Liu, CM., & Balas, V. (2021). *Studies in Computational Intelligence 903: Bio-inspired neurocomputing*
https://link.springer.com/content/pdf/10.1007/978-981-15-5495-7.pdf

Brownlee, J. (2019) *Understand the Impact of Learning Rate on Neural Network Performance*
https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

Busalaev, A., Iglovikov, V., Khvedchenya, E., Parinov, A., Druzhinin, M., & Kalinin, A. (2019). *Albumentations: Fast and Flexible Augmentations*
https://www.mdpi.com/2078-2489/11/2/125

Canuma, P. (2020). *What Are RBMs, Deep Belief Networks and Why Are They Important to Deep Learning?* The Startup
https://medium.com/swlh/what-are-rbms-deep-belief-networks-and-why-are-they-important-to-deep-learning-491c7de8937a

Çelik, H., Dülger, L., & Topalbekiroglu. (2013). *Development of a Machine Vision system: real-time fabric defect detection and classification with neural networks*
https://www.tandfonline.com/doi/full/10.1080/00405000.2013.827393

Ebert, C., Gallardo, G., Hernantes, J., & Serrano, N. (2016). *DevOps* IEEE Software (Vol.33, Issue.3)
https://ieeexplore.ieee.org/abstract/document/7458761

He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., & Li, M. (2019). *Bag of Tricks for Image Classification with Convolutional Neural Networks*
Bag of Tricks for Image Classification with Convolutional Neural Networks | IEEE Conference Publication | IEEE Xplore

Hu, Z., Zhang, J., & Ge, Y. (2021). *Handling Vanishing Gradient Problem Using Artificial Derivative* IEEE Access (Vol.9)
https://ieeexplore.ieee.org/document/9336631

ITU-R. (2011). *Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios (ITU-R 601 7th Edition)*
https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.601-7-201103-I!!PDF-E.pdf

Janzen, D., & Saiedian, H. (2005). *Test-driven development concepts, taxonomy, and future direction* Computer (Vol.38, Issue.9)
https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1510569

Kumar, A. (2022). *Machine Learning – Sensitivity vs Specificity Difference.* Vitalfux.com
https://vitalflux.com/ml-metrics-sensitivity-vs-specificity-difference.

Labudzki, R., Legutko, S., & Raos, P. (2014). *The essence and applications of machine vision*
https://www.semanticscholar.org/paper/The-essence-and-applications-of-machine-vision-Labudzki-Legutko/34e86f153d00f2cf25119eec24011edf083ebd2a

Mateos-Aparicio, P., & Rodriguez-Moreno, A. (2019). *The impact of studying brain plasticity.* Frontiers in cellular neuroscience (Vol.13)
https://www.frontiersin.org/articles/10.3389/fncel.2019.00066/full

MAVLink. (2009). *MAVLink Developer Guide*
https://mavlink.io/en/

Miller, M., Langefeld, C., Tierney, W., Hui, S., & McDonald, C. (2016). *Validation of Probabilistic Predictions*
https://journals.sagepub.com/doi/abs/10.1177/0272989X9301300107?journalCode=mdma

Miller, M. (2019). *The Basics: Logistic Regression and Regularization.* Towards Data Science
https://towardsdatascience.com/the-basics-logistic-regression-and-regularization-828b0d2d206c

Park, K., Chae, M., & Cho, JH. (2021). *Image Pre-Processing Method of Machine Learning for Edge Detection with Image Signal Processor Enhancement*
https://www.mdpi.com/2072-666X/12/1/73

Pant, A. (2019). *Introduction to Logistic Regression*
https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

Pechenina, E., Pechenin, V., & Bolotov, M. (2021). *Comparative analysis of neural network architectures for industrial machine vision.* 2021 International Scientific and Technical Engine Conference (EC)
https://ieeexplore.ieee.org/document/10016880

Ravikiran, A. (2023) *Your One-Stop Solution to Learn Kruskal Algorithm From Scratch*
https://www.simplilearn.com/tutorials/data-structure-tutorial/kruskal-algorithm

Rover, D., Ullerich, C., Scheel, R., Wegter, J., & Whipple, C. (2015). *Advantages of Agile methodologies for software and product development in a capstone design project*
https://ieeexplore.ieee.org/document/7044380

Ruder, S. (2017). *An overview of gradient descent optimization algorithms*
[1609.04747] An overview of gradient descent optimization algorithms (arxiv.org)

Scrum.org (1993). *What is Scrum?*
https://www.scrum.org/resources/what-is-scrum

Sharma, S., Sarkar, D., & Gupta, D. (2012). *Agile Processes and Methodologies: A Conceptual Study* International Journal on Computer Science and Engineering (IJCSE) (Vol.4, No.5)
https://www.yashada.org/yash/egovcii/static_pgs/TC/IJCSE12-04-05-186.pdf

Waliyansyah, R., & Hasbullah, U. (2021). *Comparison of Tree Method, Support Vector Machine, Naïve Bayes, and Logistic Regression on coffee Bean Image.* EMITTER International Journal of Engineering Technology (Vol.9, No.1)
https://emitter.pens.ac.id/index.php/emitter/article/view/536

Wang, Q., Yu, S., Qi, X., Hu, Y., Zheng, W., Shi, J., & Yao, H. (2019) *[Overview of logistic regression model analysis and application]*
https://www.semanticscholar.org/paper/%5BOverview-of-logistic-regression-model-analysis-and-Wang-Yu/ffcbe21fa55b02a4f0716fe4e613169632d9190f

Wood, T. (2019). *F-Score*. DeepAI.org
https://deepai.org/machine-learning-glossary-and-terms

Zhang, L., Wang, M., Fu, Y., & Ding, Y. (2022). *A Forest Fire Recognition Method Using UAV Images Based on Transfer Learning*
https://www.mdpi.com/1999-4907/13/7/975

Appendix

A.1 – Derivation calculation

Solving $\frac{\partial h_\theta(w^T x+b)}{\partial(w^T x+b)}$:

$$\frac{\partial h_\theta(w^T x+b)}{\partial(w^T x+b)} = \left(\frac{1}{\left(1+e^{(w^T x+b)}\right)^2}\right) * e^{-(w^T x+b)}$$
$$= \frac{e^{-(w^T x+b)}}{\left(1+e^{-(w^T x+b)}\right)^2}$$

Given that:

$$e^{-(w^T x+b)} = \frac{1-h_\theta(w^T x+b)}{h_\theta(w^T x+b)}$$

Solution equals:

$$\frac{1-h_\theta(w^T x+b)}{\left(1+\left(1-h_\theta(w^T x+b)\right)\right)^2} = \frac{\left(\frac{\left(1-h_\theta(w^T x+b)\right)}{h_\theta(w^T x+b)}\right)}{\left(1+\frac{\left(1-h_\theta(w^T x+b)\right)}{h_\theta(w^T x+b)}\right)^2}$$

$$= h_\theta(w^T x+b) * (1-h_\theta(w^T x+b))$$

A.2 – Calculating coordinate ranges

To create a 40m x 40m environment, I needed to assign a coordinate range for the shapes to be placed within. The below method details how these ranges were calculated.

Home (Heathrow) coordinates = 51.47767 (latitude), -0.45971 (longitude)

These are the angles (degrees) that the position is away from the Equator (0° latitude) and the Prime Meridian (0° longitude). The latitudinal curvature of the Earth forms an arc with the planet's centre, and the longitudinal curvature forms another.

Calculate how far away the home point is from the Equator in metres (Earth's radius = 6371000):

$$Arc\ Length = \left(\frac{\pi}{180}\theta\right)r$$
$$= \left(\frac{\pi}{180}51.47767\right)6371000 = 5724055.739$$

Calculate Upper and Lower bounds for Latitude and Longitude:

<u>Latitude Upper Bound</u>

$$5724055.739 + 20 = 5724075.739$$
$$5724075.739 = \left(\frac{\pi}{180}\theta_{upper}\right)6371000$$
$$\therefore \theta_{upper} = 51.47784986°$$

<u>Latitude Lower Bound</u>

$$5724055.739 - 20 = 572403\ 5.739$$
$$5724035.739 = \left(\frac{\pi}{180}\theta_{lower}\right)6371000$$
$$\therefore \theta_{lower} = 51.47749014°$$

<u>Longitude Upper Bound</u>

$$Arc\ Length = \left(\frac{\pi}{180}(-)0.45971\right)6371000 = 51117.41973$$
$$51117.41973 + 20 = 51137.41973$$
$$51137.41973 = \left(\frac{\pi}{180}\theta_{upper}\right)6371000$$
$$\therefore \theta_{upper} = -0.4598898643°$$

<u>Longitude Lower Bound</u>

$$51117.41973 - 20 = 51097.41973$$
$$51097.41973 = \left(\frac{\pi}{180}\theta_{lower}\right)6371000$$
$$\therefore \theta_{lower} = -0.4595301357°$$

A.3 – Supervisor meeting log

16/01/23 – Briefly discussed an overview of the entire project and talked through how I was planning to create my training and validation datasets and what augmentations I was going to apply to the data.

23/01/23 – Discussed my created datasets and my configured Kanban board. Talked about writing the literature review and its estimated word count. Discussed what to write about when documenting project management, research methodologies and implementation methodologies.

30/01/23 – Talked about adding sprints, priorities, and a backlog to the Jira project. Showcased first draft of the introduction.

06/02/23 – Wrote code to generate images and apply the augmentations. Showed a few augmented images to supervisor who said he was happy.

13/02/23 – Discussed the limitations of my data creation method, namely: the final images being far too large. Spoke about potential compression methods and how they could be applied.

27/02/23 – Showcased my working compression algorithm and the results from the Logistic Regression model to my supervisor. We also discussed how I could go about designing the Neural Network and how to write up the LR analysis.

06/03/23 – Showcased my working Neural Network. The results were poor, but it hasn't been tuned yet.

13/03/23 – Demonstrated the tuned Neural Network – this time with a much superior accuracy. We also spoke about writing up the comparison and improving my literature review.

03/04/23 – Demonstrated the path-finding algorithm working on the 2D matplotlib environment. Discussed how it would translate to the similar and went through a few formatting inaccuracies.

END