GALAXY IMAGE CLASSIFICATION: USING MACHINE LEARNING IN PYTHON

by

Jack T Donaldson

A Dissertation Supervised by Dr. Robert Lyon

Statement:

‘This Report is submitted in partial fulfilment of the requirements for the BSc Honours Computing (or Web Systems Development, Computer Science) Degree at Edge Hill University.

Submission Date: 13th May 2020

Contents

[Abstract 4](#_Toc40282146)

[acknowledgements 4](#_Toc40282147)

[Chapter 1: Introduction 5](#_Toc40282148)

[Chapter 1.1 - Project Motivation 5](#_Toc40282149)

[Chapter 1.2 – Aim and Objectives 6](#_Toc40282150)

[Chapter 1.3 – Project Scope 7](#_Toc40282151)

[Chapter 1.4 – Project Methods 7](#_Toc40282152)

[Chapter 1.5 – Report Structure 8](#_Toc40282153)

[Chapter 2: Literature Review & Related Work 9](#_Toc40282154)

[Chapter 2.1 – Classifying Morphological Galaxies 9](#_Toc40282155)

[Chapter 2.2 – Useful Galaxy Datasets 10](#_Toc40282156)

[Chapter 2.3 – Image Recognition Techniques 11](#_Toc40282157)

[Chapter 3: Research Design 18](#_Toc40282158)

[Chapter 3.1 – Projects Life Cycle 18](#_Toc40282159)

[Chapter 3.2 – Methodology 18](#_Toc40282160)

[Chapter 3.3 – Choosing the right dataset 19](#_Toc40282161)

[Chapter 3.4 – Dataset Chosen 20](#_Toc40282162)

[Chapter 3.5 – Developing a Decision Tree 21](#_Toc40282163)

[Chapter 3.6–Use of Surveys 24](#_Toc40282164)

[Chapter 3.7– Description of the type of Requirement Gathering Process 24](#_Toc40282165)

[Chapter 3.8– Description of where the requirements Derived from 24](#_Toc40282166)

[Chapter 3.9 – Functional & Non-Functional Requirements 25](#_Toc40282167)

[Chapter 4: Development 26](#_Toc40282168)

[Chapter 4.1 – Setting up Model 26](#_Toc40282169)

[Chapter 4.2 1st – Design 26](#_Toc40282170)

[Chapter 4.3 1st - Convolutional Network Layers and Parameters 28](#_Toc40282171)

[Chapter 4.4 1st - Designing a Dataflow Diagram for testing Validation and Accuracy 30](#_Toc40282172)

[Chapter 4.5 1st – Implementation 30](#_Toc40282173)

[Chapter 4.6 1st – testing 33](#_Toc40282174)

[Chapter 4.1 2nd Prototype 35](#_Toc40282175)

[Chapter 4.1 2nd – Design 35](#_Toc40282176)

[Chapter 4.2 2nd – Development 37](#_Toc40282177)

[4.3 2nd – Development 41](#_Toc40282178)

[Chapter 4.4 2nd – Testing 42](#_Toc40282179)

[chapter 4.5 2nd - EVALUATING TEST PREDICTIONS 44](#_Toc40282180)

[Chapter 5- Conclusion 47](#_Toc40282181)

[Chapter 5.1 – conclusion 47](#_Toc40282182)

[Chapter 5.2 - – Summary of the Project with Critical Evaluations on each Stage 47](#_Toc40282183)

[Chapter 5.2– Future Work 48](#_Toc40282184)

[References 49](#_Toc40282185)

[Appendices 52](#_Toc40282186)

[Appendix A: setting up Model 57](#_Toc40282187)

## Figures

[Figure 1 - Edwin Hubble’s Classification Scheme 10](#_Toc40282188)

[Figure 2 - Overview of the pupil detection framework 13](#_Toc40282189)

[Figure 3 – comparison between biological neuron and its mathematical model 13](#_Toc40282190)

[Figure 4 - A Mostly complete chart of neural networks 15](#_Toc40282191)

[Figure 5 - A basic architecture of a Recurrent Neural Network (RNN) 16](#_Toc40282192)

[Figure 6 - A basic architecture of a Deconvolutional Network (DN) 16](#_Toc40282193)

[Figure 7 - Text to photo realistic image synthesis using Deconvolutional Network 17](#_Toc40282194)

[Figure 8 – A basic architecture of a Deep Convolutional Inverse Graphics Network (DCIGN) 17](#_Toc40282195)

[Figure 9 - A photograph translated to famous painter styles using Cycle-Consistent Adversarial Networks 18](#_Toc40282196)

[Figure 10 - Evolutionary Prototyping Plan 19](#_Toc40282197)

[Figure 11 - pedestrians counted as pedestrians 20](#_Toc40282198)

[Figure 12 - 1st iteration of an earlier proposed decision tree 22](#_Toc40282199)

[Figure 13 - Questions leading to the developing of the Decision Tree 23](#_Toc40282200)

[Figure 14 - 2nd iteration of a Decision Tree based off figure 13’s questions 24](#_Toc40282201)

[Figure 15 - Dataflow diagram for accessing, retrieving data and training the model 27](#_Toc40282202)

[Figure 16 - First iteration of the CNN architecture design 28](#_Toc40282203)

[Figure 17 - Kernel Size Example 29](#_Toc40282204)

[Figure 18 - types of data for each 37 classes 30](#_Toc40282205)

[Figure 19 - Dataflow Diagram for the testing stage 31](#_Toc40282206)

[Figure 20 - Library’s used for 1st prototype 31](#_Toc40282207)

[Figure 21 - initializing some variables and arrays for later use 32](#_Toc40282208)

[Figure 22 - Creating the data augmentation, training and validation classes 32](#_Toc40282209)

[Figure 23 - Creating the CNN model based on the Design chapter 4.2 33](#_Toc40282210)

[Figure 24 - developing the graphs with use of classifier history functions 34](#_Toc40282211)

[Figure 25 - MODEL ACCURACY AND LOSS PLOTTED AGAINST EPOCHS 34](#_Toc40282212)

[Figure 26 - model accuracy and loss plotted against epochs (2nd iteration) 35](#_Toc40282213)

[Figure 27 - Model Runtime 35](#_Toc40282214)

[Figure 28 - 2nd iteration of the CNN architecture design of the 2nd prototype model 36](#_Toc40282215)

[Figure 29 - dataflow diagram for writing new probabilities to Empty CSV file 37](#_Toc40282216)

[Figure 30 – amended 2nd iteration of the CNN architecture design of the 2nd prototype model 38](#_Toc40282217)

[Figure 31 - 2nd iteration of model developed following amended architecture design 38](#_Toc40282218)

[Figure 32 - last amended code to the first python script 39](#_Toc40282219)

[Figure 33 - Added library for additional functionality 39](#_Toc40282220)

[Figure 34 - test image generator 40](#_Toc40282221)

[Figure 35 - writing predictions to empty CSV file and reading values based on image ID 41](#_Toc40282222)

[Figure 36 - training and validation loss and accuracy for epochs ranging from 3 to 10 43](#_Toc40282223)

[Figure 37 - training and validation loss and accuracy for 30 epoch iterations 44](#_Toc40282224)

[Figure 38 - First set of test data 45](#_Toc40282225)

[Figure 39 - second set of test data 45](#_Toc40282226)

[Figure 40 - third set of test data 45](#_Toc40282227)

[Figure 41 - Root mean Squared Error Function 46](#_Toc40282228)

[Figure 42 - First set of Survey Test Data 46](#_Toc40282229)

[Figure 43 - Second set of Survey Test Data 46](#_Toc40282230)

[Figure 44 - Image of galaxy ID 100062 47](#_Toc40282231)

[Figure 45 - Checking tensor flow is working by looking for GPU 49](#_Toc40282232)

[Figure 46 - Pedestrian Detection Code 53](#_Toc40282233)

[Figure 47 - classifier history from the first iteration of testing 53](#_Toc40282234)

[Figure 48 - classifier history from the 2nd iteration of testing with the first prototype 54](#_Toc40282235)

[Figure 49 - added library 54](#_Toc40282236)

[Figure 50 - weights saved 55](#_Toc40282237)

[Figure 51 - the images produced with figure 30 code 55](#_Toc40282238)

[Figure 52 - Use of new functionality 55](#_Toc40282239)

[Figure 53 - Hardware Performance Limit 56](#_Toc40282240)

[Figure 54 - Survey Example 57](#_Toc40282241)

## Abstract

The problem of classifying galaxies in images can become a very challenging and time-consuming task when considering the number of galaxies expected to be found ranging from 100 billion to 200 billion (Howell,2018). Astronomers have the enormous task of classifying each galaxy however time constraints demand constant attention to accurately depict the morphological representation of each galaxy state. Not being able to accurately classify new waves of galaxy data slows down the discovery of the evolutionary nature of the universe (Greshko, 2019). Current research in machine learning poses new solutions to alleviate this problem with many diverse, complex and accurate techniques already being applied to image data. such techniques known as Convolutional Neural Network at the forefront of image recognition models due the highly applicable and adaptable nature of designing models. Using CNN with galaxy image datasets available and conducted by a previous Galaxy Zoo challenge (GAMA | Galaxy and- Mass Assembly, 2020), weighted probabilistic distributions where applied to each image to gather the results representing the likelihood of the type of class a galaxy belonged to. This success of this project reached the low 60% quartile is measuring accuracy and with the sample of test data conducted with the same evaluation metric (root mean squared error), each sample of data was compared with machine learning and the human eye.

## acknowledgements

I would like to thank my supervisor Dr Robert Lyon for the guidance, encouragement and advice he provided throughout the Year of developing various works. I was extremely lucky to have a supervisor who cared a lot about my work and the machine learning field offering helpful feedback along the way. I would also like to thank all the members in the computer science department responsible for setting up an engaging and interesting curriculum around machine learning through later modules as this also helped me decide the type of project to undertake.

## Chapter 1: Introduction

### Chapter 1.1 - Project Motivation

Studying morphological galaxies can be an extensive and tedious task for many different areas of study, especially with manually assigning and classifying galaxies with astronomers handling this area. Most galaxies will exist in several forms either “elliptical”, “lenticular”, “spiral” or “irregulars”. These are what are referred to as morphological names. Understanding the evolution of galaxies is one of the preferred ways to determine the root of the history of the universe, as experts can determine how the “milky way and other types of galaxy came to be” (Herts.ac.uk, 2019). This in turn allows researchers to gather ideas and construct many different theory’s from how from atoms up to the expansion of the universe (in great depth of course). However, as exciting and serious as this area of study is, the approach followed by many astronomers when classifying millions of galaxies can become incredibly time consuming as it is manual. This manual process is error prone. Especially because experts can have varying opinions on the correct morphology of a galaxy. There is also disagreement on the total number of galaxies we expect to find in the night sky, e.g. “an acceptable range is between 100 billion and 200 billion galaxies” (Howell, 2018). The vast number of potential galaxies to be classified make this problem difficult, and classifications are often the subject of debate as “expert-devised classification schemes may be subject to cognitive biases” (Nolte et al., 2019). This large and daunting task can be solved by using other automated techniques. One of these involves using machine learning algorithms to sort through millions of images of galaxies and classify them on their own, with accuracy rivalling astronomers and without the cognitive bias that stirs debate amongst the astrophysics community.

Using machine learning to save large amounts of time and at the same time being able to achieve highly accurate results, does not come easily. There are many problems and ideas to consider while aiming to develop an algorithm which can produce these sought-after results. Some problems rise with the datasets we use to teach our algorithms, as astronomical images can be of low quality due to the distance between Earth and a galaxy being imaged or even faults/flaws/inconsistencies in the telescopes used when gathering suitable images for training, validating and testing. The complicated nature of galaxy formations, coupled with the variation in the quality of the images in modern astronomy datasets, has made “the classification of galaxies challenging and not accurate”. (Donaldson 2020)Many different machine learning approaches have been proposed for this problem, for example, a model consisting of oblique decision trees preforming the classification of galaxy images. It achieved an accuracy level of 64.6 (Owens et. al., 1996) demonstrated using 5-fold cross validation. Another model capable of classifying galaxies using Artificial Neural Networks (ANNs) could predict the morphology of galaxies via their spectra with approximately 70% accuracy (Madgwick, 2003) . The accuracy levels achieved by these classification models is incredibility important in justifying the credibility of a system, however, a model that results in lower accuracy may not be worse. It may be that less accurate methods are simply more sensitive to the inherent variation between galaxies. Such approaches may be better at determining the different stages of a galaxy, instead of just generalizing them into 3 subcategories as is typically done when during classification. This suggests that a model should be created to account for as many forms a dataset that could be collected in practice to fully capture the variation between sources. If datasets are enough quality and variability, it may be possible to build a classifier that yields an accuracy level deemed acceptable by professional standards. Researching and understanding the different ways to achieve this is crucial in designing, modelling and implementing a respectable system that can classify well enough to pass these standards.

### Chapter 1.2 – Aim and Objectives

Aim:

* The primary aim of this project is to develop a Galaxy classifier that accurately separates Galaxies according to their “morphology” (shape, size, colour).

Objectives:

1. Gathering user Requirements

* The use of a questionnaire or other use of resources to help document and gain functional & non-functional requirements.
* Using the requirements to shape the artefact.
* With new iterations of the prototype, implement and document any new requirements into the project.

2. Designing the first Prototype

* Research and experiment with different sorting algorithms finding best preforming algorithm with small samples of data.
* Research image processing techniques that can be applied to this project.
* Experiment and analyse the functioning algorithm against accuracy and determine any changes need to be made.
* Find datasets and make sure they are partitioned.
* Document any findings.

3. Developing & Refining the Prototype

* Develop a small-scale prototype that utilizes datasets by training upon data and making predictions on unseen data not in the training set.
* Experiment on finished prototype to find the accuracy and efficiency of the artefact.
* Implement new features based off new requirements and experiments.
* Update the requirement section and design section.
* Repeat step 3 until final prototype is sufficiently developed.

4. Conclude the final Prototype

* After the final prototype has evolved to a required standard, preform final testing of the artefact.
* Report any limitations of advantages of the final algorithm on the final test and training dataset.

### Chapter 1.3 – Project Scope

The focus of this project will be limited to using machine learning techniques that can be dynamically applied to developing and altering a Convolutional neural network through various iterations of prototypes. Supervised learning was the initial direction of this project but due the implementation utilizing of similarities in images with no definitive classes, unsupervised learning techniques will be focused on instead. The data for training and testing has already been recorded but will be partition in the development of each iteration to produce various training and validation splits. The test data near the end was previously meant to be predicted for the use of recoding more data for the challenge but this will be analysed in later chapters to discuss the accuracy and validity of the latest design prototype model. Some external techniques will be looked at like adding more functionality to the programme, any new design choices through a Literature Review/Related Work can be amended into the programme but definitive choices will be made to push for early prototype versions.

### Chapter 1.4 – Project Methods

The methods being used in this will allow for a number of iterations before meeting the aims, objectives, requirements and overall satisfaction of the artefact’s performance with the evolutionary prototyping model as the backbone of the project’s life cycle pushing for new improvements. This evolutionary prototyping will cycle through requirement gathering by means of a feasibility study of other related work in this field with the aid of related work contributing to whether the focus towards the end artefact can be practically materialized in terms of design, implementation and testing. These requirements will be split between the functional which focus on the on the “behaviour or function” (Eriksson, 2012) of the system being developed and the non-functional, focusing on the “quality attributes” (Eriksson, 2012). The method of experimenting with each stage will the evolutionary process progress will be quantitively monitored. An in-depth analysis of a past project’s (case studies) will create the foundation of the first design iteration that will produce a prototype, this will be most evident in chapter 3. Once this first design is implemented, testing will occur to refine the sampling and batch processing methods based on hardware limitations and accuracy of the model. The project will be research focused with conclusions of each iteration of the project used to gain insight and lead to changes that will provide incremental improvement. This approach will require no surveys or interviews, but observation and measurement applied during algorithm testing stages.

### Chapter 1.5 – Report Structure

Chapter 2 – This chapter covers a literature review with related work supporting the findings of various sub chapters. These sub chapters discuss how to classify a galaxy based on morphological state, useful galaxy datasets that already exist, some important image recognition techniques currently being applied in the industry and similar research projects some of which include cascade, neural networks and convolutional neural networks.

Chapter 3 – In chapter 3 the methodology is chosen and explained through testing conducted from the proposal of this project. This included the projects life cycle choice, techniques being design for the convolutional neural network, dataset chosen with reason why, the iterations of decisions trees developed with the use of a questionnaire, and the end discusses the gathering process for functional and non-functional requirements.

Chapter 4 1st – Shows the design stage for developing the first prototype with dataflow diagrams, CNN architectural design with reason behind each layer and parameter chosen. Some extra dataflow diagrams will be designed to show various data being accessed and used with later stages in the development. The implementation follows the design into each section of code describing any problems encountered and explanation of certain sections. Lastly in the testing stage use of accuracy and loss will be plotted to depict the rate of success with certain hypothesis answered and improvements assumed for the 2nd iteration.

Chapter 4 2nd – This chapter follows the same layout but with changes made in the design stage with any added or found improvements from previous testing. This will include new CNN architectural designs and dataflow diagrams. Development will undergo the same construction with any added coding blocks discussed with and new functionality to the system. New scripts will be discussed with explanations of the code developed for the 2nd prototype will be discussed as well as the testing stage. Here the new model should improve on the old one with enough constancy being able to accurately test on new test image data.

Chapter 5 – This chapter will hold information on critical evaluating the report with listed limitations and improvements. There will also be a future work sections for ways to improve the system in possible later work.

Appendices and references – These last two chapters will hold information on the references used throughout the report with figures which would disrupt the flow of the report. There will be one added appendix section stating how to run the finished python software.

## Chapter 2: Literature Review & Related Work

### Chapter 2.1 – Classifying Morphological Galaxies

Understanding the difference between galaxies is important to separate them into different categories or “morphologies”. Understanding the data helps us to focus on what type of algorithms will need to be developed. The scheme originally proposed by Edwin Hubble as far back as 1926 (Abraham and van den Bergh, 2001) uses a “tuning fork” classification diagram that separates galaxies into five basic classification categories.

The Hubble Classification scheme is shown to have 5 broad categories,

* Elliptical Galaxies – These are ordered towards the start of the diagram with “E” representing the elliptical class with “n” being the concentration of stars within the galactic core of the galaxy. For example, “E4” would have a much brighter and compact middle section (bulge) than “E1” so “elliptical galaxies are label E0, E1, and so on” (Martel, Premadi and Matnzer, 1997).
* Lenticular Galaxies – These types of galaxies are very similar to elliptical galaxies due to the similarity of the highly condensed galactic core but contains a disk with “no appreciable amount of cold gas or dust and no spiral arms”. Martel, Premadi and Matnzer, 1997).
* Spiral Galaxies – These are flattened disk-like galaxies with stars forming in the middle very similar to the ellipticals but contain spiral-like arms sprouting out from the budge. These are labelled S for spirals and nth letter of the alphabet to distinguish the “disk-to-bulge luminosity ratio (D/B)” Martel, Premadi and Matnzer, 1997)
* Barred Spiral Galaxies – just like spiral galaxies in the way the bulge is connected by arms of stars, gas and dust, however, they consist of two parallel bars. These are labelled “SB” for spiral barred along with the nth number of the alphabet representing the disk to bulge ratio. Later nth characters represent further superaged arms.
* Irregular Galaxies – galaxies that do not belong to any previous classes previously mentioned, most irregular galaxies fail to be categized properly (especially with the Hubble classification) due to most being “small, gas-rich galaxies” Martel, Premadi and Matnzer, 1997).

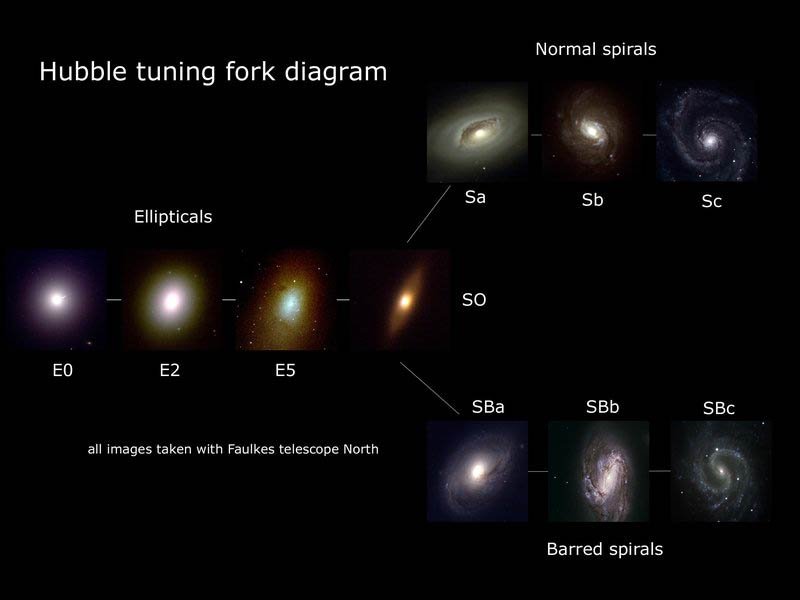


Figure 1 - Edwin Hubble’s Classification Scheme

Image Credit: Galaxy Classification | Las Cumbres Observatory, 2020

What Edwin Hubble developed was incredibly useful in the early 1900s, but the “hooker telescope” (Nasa.gov, 2002) he used, became outdated quickly and is especially primitive compared to modern telescopes like the Sloan Foundation 2.5m Telescope (Sdss.org, 2019) being used to scan 304,122 (Royal Astronomical Society. 2013) galaxies. Machine learning algorithms (if developed properly) can make use of more complex galaxy classification schemes which in turn increase the accuracy of specific galaxies with a higher variance.

One scheme that can make use make use of collected data that has higher levels of variability with respect to Galaxy morphology (that can be used to better classify), was developed early in the history of galaxy classification field but has remained prominent for classifying galaxies with ideas built on the Hubble sequence. The most notable being a scheme developed by Gerard Henri de Vaucouleurs (W. Hodge, 2019). His scheme contains all the previously mentioned classes, but further splits them to obtain purer classification categories (less cross-over between them). For example, using the “cD” notation to describe “galaxies with abnormally large, distended shapes”, whilst also providing scope for combining different classes. However, even though the de Vaucouleurs system proposed interesting new ways to classify galaxies, it has many categories. This is relevant because here, the goal is to develop a machine-learning algorithm to correctly predict the type of galaxy. The numerous class labels provided by the Vaucouleurs system, may, when used to train classification algorithms, cause less accurate results. This can happen as there are classes for which there are few example images (i.e. few training samples). Indeed, more than “90% of luminous nearby galaxies fit within” (Abraham and van den Bergh, 2001) the Hubble system. Any irregularities or faint dwarf galaxies that are too “difficult to detect at large distances” (Abraham and van den Bergh, 2001) are generally part of the available datasets and could cause less accurate results. Focusing on prototypes instead of one iterative method-based project will allow use of the newer schemes and proposed classification strategies. This may be the case during the design stage if the earlier proposed classification schemes prove to be not useful, this can be drawn from the design stage for choosing certain datasets or whether implementing one scheme proves to ineffective during the testing stage.

### Chapter 2.2 – Useful Galaxy Datasets

In the paper Galaxy Classification: A Machine Learning analysis of GAMA catalogue data, Aleke Nolte discusses how making use of “SOM-clustering results” with typical galaxy classification techniques resulted in around “73%” accuracy. They concluded that his machine learning algorithm did not fully support galaxy classification schemes which he stated was due to a low accuracy However, he states this low accuracy could be caused by absence “of essential parameters in the data set”. Making use of the proper algorithm can be said with any problem that is insufficiently solved however the dataset he had used came from the GAMA catalogue which can be used in multiple ways to represent galaxies and how it can be utilized by different classification and regression methods. The Galaxy Zoo dataset has 37 weight probabilities determining the likelihood that a corresponding image is a type of galaxy. Using these parameters may not be fully supported by Aleke Noltes proposed classification techniques however, instead of working through multiple surveys recorded in the GAMA catalogue, the data may prove to be more useful offering specific parameters on one entire dataset. This could possibly fix the absence of essential parameters. A project developed by Kyle W. Willet utilised these datasets to train classification algorithms and achieved 90% accuracy when comparing to professional astronomers’ predictions (Kyle W. 2013). Making use of this dataset may have allowed for Aleke Nolte to produce a SOM Clustering algorithm with a higher accuracy. Galaxy classification is further discussed by a group of researchers with the leading member “Aleke Nolte” (Nolte et al., 2019) who focus on multiple surveys recorded by the GAMA catalogue analysed galaxies based on a few notable features. Some noticeable features including using morphological features like spirals, elliptical bulge and “spectroscopic” (Nolte et al., 2019) which is the study of brightness metrics based on certain galaxy formations. However, using said surveys for datasets had no benefit in using “individual catalogues nor a combined dataset based on all 5 catalogues” can fully support image-based galaxy classification schemes. This suggests the current catalogues available from GAMA cannot support the focus of using image classification schemes but rather used for other areas of study not directly focused with using machine learning to classify galaxies based on their morphological state. A solution to this is finding a dataset which can support machine learning techniques being applied in this area of study. One possible dataset which fits this criterion was developed and gathered by the incorporation of using the “Hubble space” telescopes (recently funded by “NASA” (Hubble Space Telescope – Advanced Camera for Surveys, 2014)) with a Kaggle competition called “Galaxy Zoo – The Galaxy Challenge” (Galaxy Zoo - The Galaxy Challenge | Kaggle, 2014). This dataset presents over “61,597” trainable galaxy images and “79,975” testable images with “37” (Galaxy Zoo - The Galaxy Challenge | Kaggle, 2014) weighted possibilities. This dataset benefits from the competition being hosted supporting the idea behind using machine learning to classify large datasets with already partitioned data ready to be implemented saves time and does not needs to extra surveys to be conducted or combined with an already public, available dataset.

### Chapter 2.3 – Image Recognition Techniques

Image classification is a useful tool for “categorizing images into one of several predefined classes” (Rawat and Wang, 2017) but problems arise based on the set of tasks such as “localization, detection and segmentation” (Rawat and Wang, 2017). Many computer vision techniques approach was developed and nurtured to best support the vast amount of image data that in turn could extract certain conclusions to benefit the area of study.

One prominent image classifier in machine learning is called cascade, which is a cascade of boosted classifiers working with Haar like features. This is formally known as Haar cascade and is a “machine learning object detection algorithm” (Berger, 2014) used to identify a large range of objects if that’s in images or videos. This is an incredibly useful technique as well, shown in Will Berger’s feature extraction method obtaining “160000+ features”. However, using cascade independently like many machine learning algorithms, preform worse but with the same 160000+ features Will Berger uses a “concept called Adaboost” which has been predominately paired up with cascade classifier in the past on multiple projects. This is also evident in a paper “*Cascade Classifier For Face Detection”* (Wang and Yang, 2016) where Huachun Yang and Xu An Xang also use AdaBoost for another facial detection classifier as stated to be “an efficient learning algorithm for constructing a strong classifier” (Wang and Yang, 2016) as an additive when weak classifier are only slightly better than random. This is due to AdaBoost training on certain patterns after each iteration has passed. Cascades with

Adaboost has been further into specific features of a face taking facial detection to another level of accuracy. Chao Gou and his automation team at the academy of sciences in Beijing where they have “achieved the state-of-the-art pupil detection” through a model support by a regression model. This is done by starting with an “initial semantic key point” (Gou et al., 2019), followed iteratively updating the positions through “sequential regression models until convergence”.

A screenshot of a cell phone

Description automatically generated

Figure - Overview of the pupil detection framework

Image Credit: Gou et al., 2019

While Facial Detection and object detection still being thoroughly researched in cascade image detection, neural networks have consistently used in many areas of image detection. Neural networks have originally been inspired of by modelling a biological neural system. These types of models have achieved “good results in machine learning tasks” (CS231n Convolutional Neural Networks for Visual Recognition, 2020) as well as in engineering tasks.

Chapter 2.4 – Neural Network Architectures

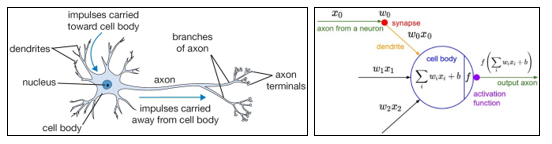


Figure – comparison between biological neuron and its mathematical model

Image Credit: CS231n Convolutional Neural Networks for Visual Recognition, 2020

While the neural network is loosely modelled after the human brain, there are rules it still follows as a result of this. With approximately 86 billion neurons in the human nervous system and which relate to approximately 10^14 – 10^15 synapses (CS231n Convolutional Neural Networks for Visual Recognition, 2020) each neuron receives input signals with output signals being produced along an axon (depicted in the biological neuron above). The mathematical model follows this path very closely as well with the signal (inputs like an image or x0) interacting multiplicatively (with a weight and input or as w0x0). The weights (w0) become learnable and control the strength throughout the process of the neural network. While a neural network is active a sum of the weights and input are summed up which can be looked at as with 3 inputs, y = x0w0 + x1w1 + x2w2. If all the products are summed up and the resulting number is above a threshold, the node fires at a rate which can be modelled by with an activation function. Activation functions represent the frequency along a biological neuron with the choice of a function being dictated by the model, data or results gathered through testing. Also, if the summed value is below a certain threshold, then no data is passed to the next layer in the neural network. (CS231n Convolutional Neural Networks for Visual Recognition, 2020)

A neural network or more precisely known as Artificial neural networks (ANN) belong to a class of machine learning algorithms which have gained a large amount of attention due to the amount of availability of Bid data and fasting computing facilities. Many classes of neural networks can be picked out for the foundation of the architecture being built with many supporting image recognition and classification schemes however some are better suited for certain datasets. One highly useful network called convolutional neural networks (CNNs / convNets) make the explicit assumption that inputs are images (CS231n Convolutional Neural Networks for Visual Recognition, 2020). This suggests the implementation can be encoded with certain properties within the architecture which make certain functions and properties more efficient to implement and reduce the amount of unneeded tailing to the network reduced. However, CNNs are still very similar to ANNs with inputs being passed through a neuron with learnable weights and biases with the network still expressing a single differentiable score function and loss functions. This infers that many of the ideas for designing and developing a neural network still apply to CNNs.

Although it is still important to note a CNN is a subclass of neural networks which put very simply have a least one convolutional layer, Andrew Tch discuses the differences between a chart of neural networks developed by Fjodr van Veen from the Asimov institute (Tch, 2017).

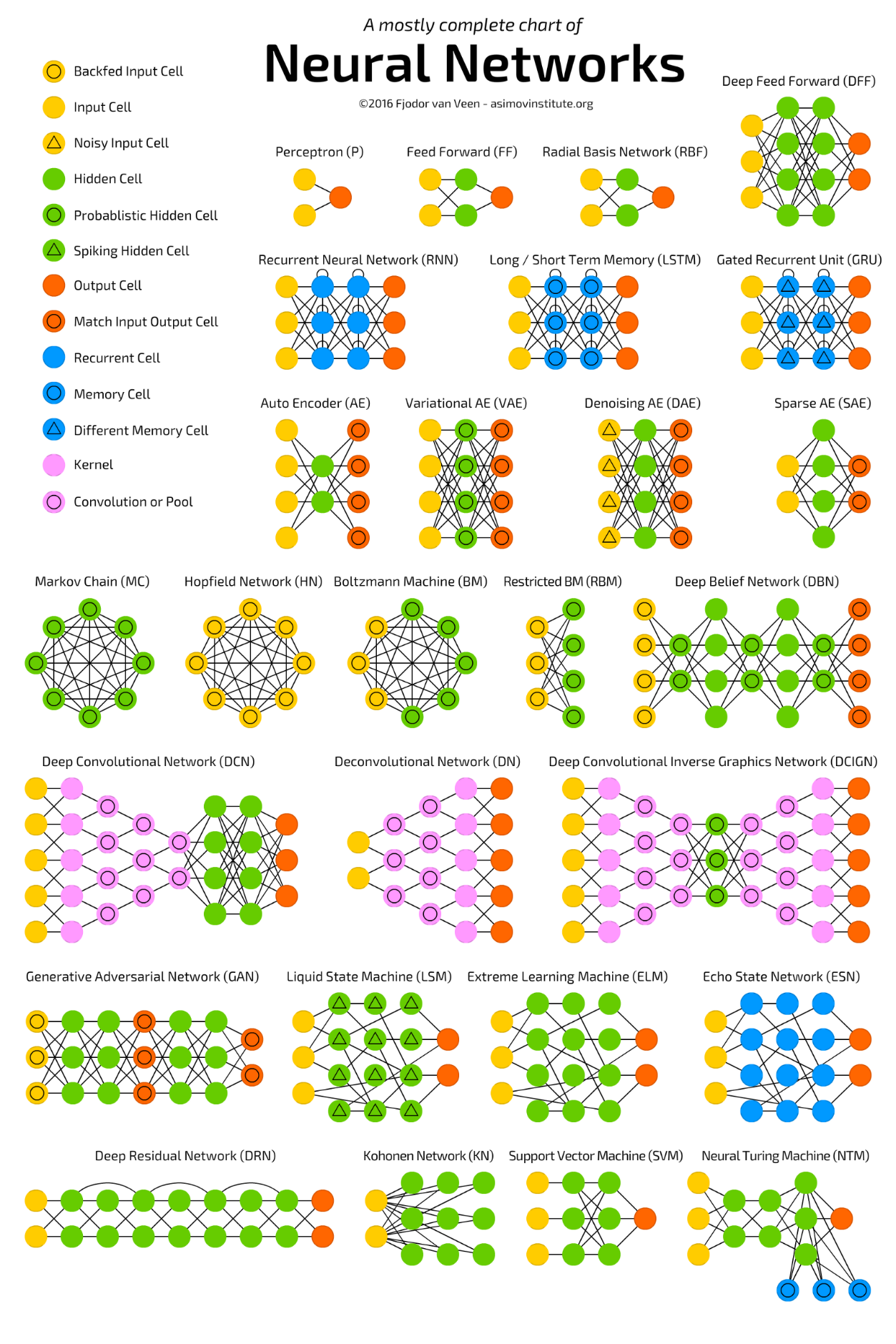


Figure - A Mostly complete chart of neural networks

Image Credit: Andrew Tch, 2017

The Andrew Tch’s explanation of each neural network is incredibly important when understanding the different uses and applications each proposes as the foundation of the architecture being designed in later chapters can follow general rules. This is evident through the explanation on just a few of the neural networks for example the Recurrent neural network uses recurrent cells with many variations using nodes, variable delays. The idea of this network is based on the idea that context is more important with decisions from earlier iterations of samples can influence current ones (Andre Tch, 2017). The general use of this network is used for datasets containing text like independent words, sentences or paragraphs. Each text will only be analysed in previous texts which influence the current samples or iterations.

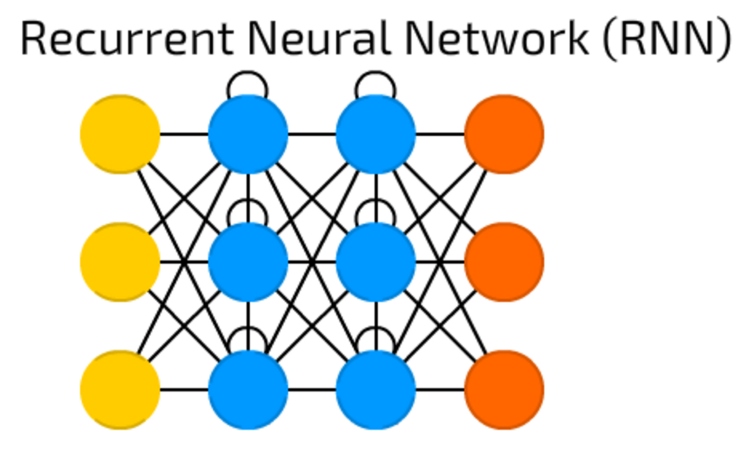


Figure - A basic architecture of a Recurrent Neural Network (RNN)

Image Credit: Andrew Tch, 2017

Andrew Tch also discusses networks which work in the inverse direction of Convolutional networks called Deconvolutional Network (DN). This type of network takes an image and produces a vector corresponding to the image (single dog image produces a vector {dog: 1, cat:0} ). This then allows for a DN to produce an image based on the vector.

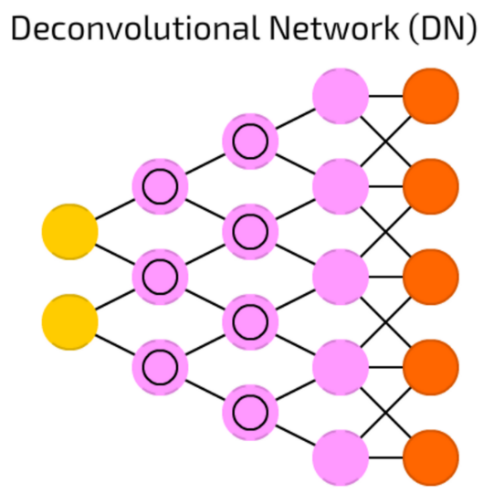


Figure - A basic architecture of a Deconvolutional Network (DN)

Image Credit: Andrew Tch, 2017

A deconvolutional network is taken a step further by a group of researchers at Cornell University who propose a model utilizing a deconvolutional network in two stages. The first stage takes the text generated image and adds primitive shape and colours based on the object given (bird, flower etc…). stage 2 takes the first stage results and text descriptions and generates high resolution images with photo realistic details. There is a 3rd layer which adds more unsampling layers but is unable to produce any plausible images. (Zhang et al., 2017).

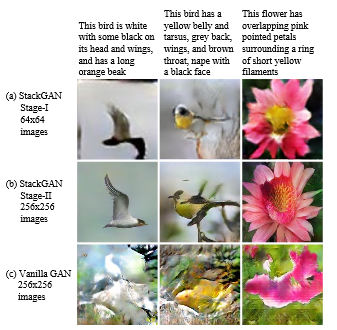


Figure - Text to photo realistic image synthesis using Deconvolutional Network

Image Credit: Han Zhang, Tao Xu, Hongsheng Li,Shaoting Zhang, Xiaogang Wang, Xiaolei Huang2, Dimitris Metaxas, 2017

The compelling results gathered from this team of researchers show the depth of accuracy (when creating recognisable images) and direction a deconvolutional network can be developed to when using text to create realistic images. However, this network developed is impressive there are networks that can make use of certain design features of earlier architectures that almost create a network that looks like it has been combined with two. One great network that fits this description is called a Deep Convolutional Inverse Graphics Network although it’s not exactly a combination of DCN or DN, the network makes use of these network by acting as “spacers for input and output of the network” (Andre Tch, 2017). See figure 8 for a basic description of this network below.

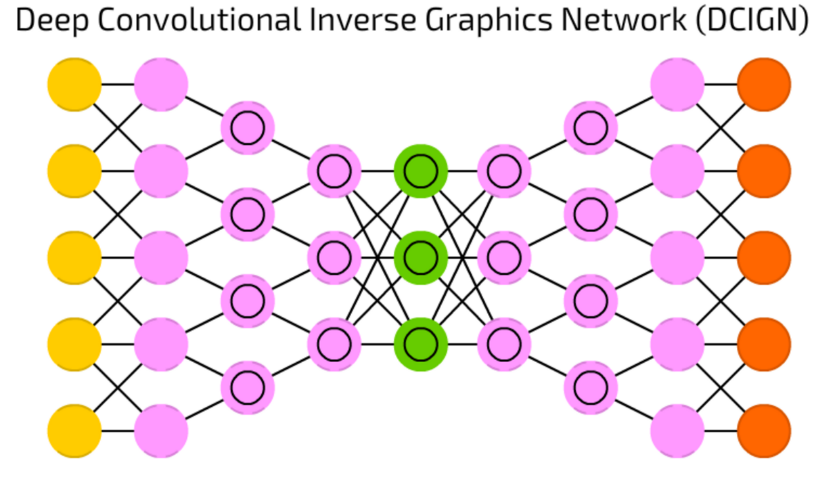


Figure – A basic architecture of a Deep Convolutional Inverse Graphics Network (DCIGN)

Image Credit: Andrew Tch, 2017

Using this type of network (DCIGN) is generally used for image processing that can process images that have not been trained at the start of the network. This is due to the abstraction levels which can pick and remove certain objects from an initial image and in turn can be remade (Andre Tch, 2017). A team of researchers at Berkeley AI research laboratory developed an unpaired image to image translation using cycle-Consistent adversarial networks with the idea of using a Deep Convolutional Inverse Graphics Network as the architecture(Zhu, Park, Isola and A. Efros, 2018). Through their efforts they were able to produce new images based on earlier inputs of normal photos. Each example use image to image translation meaning when a photo of a Zebra is translated to a Horse the same can occur for a Horse to Zebra. The most noticeable depth of their developed network was able to take a photograph and translate it to the style of a painter like Van Gogh.



Figure - A photograph translated to famous painter styles using Cycle-Consistent Adversarial Networks

Image Credit: Zhu, Park, Isola and A. Efros, 2018

With many of the impressive models developed by various teams of researchers across the globe many if not all of them developing neural networks follow an architecture as the backbone of funnelling various algorithms and implementation techniques into areas of development. However, finding the most suitable architecture is up to the direction of designing a model around datasets and what you want to do with the data can be with a rough guess because sadly there is no generic way to determine the best layers for a network. A solution to this is to start with gathering a dataset and understanding the direction of how you want to manipulate and monitor success of testing data. For example, classifying images based on a set of parameters may work best with CNNs (convolutional neural networks) as there is no limit to adding layers to find the best preforming model. Even fine tuning the parameters for these layers proves to beneficial based on the dataset being used, this is especially true if the dataset is large ReLU activation may work most efficiently.

## Chapter 3: Research Design

### Chapter 3.1 – Projects Life Cycle

The projects life cycle will go through evolutionary prototyping and with this development method the first prototype will be constructed but this approach differs slightly from rapid prototyping. This scheme uses “the best understood requirements; whereas in rapid prototyping, the developer implements the least understood requirements” (Sherrell, 2013). This method is best suited to how gathering requirements through previous case studies and related work gathering techniques will build greatly understood functional and non-functional requirements of the earlier prototype iteration. This is like incremental development as specific parts will be looked at during the software’s life cycle, with each subsequent prototype “additional functionality or improvements” (Sherrell, 2013) will be added until the final artefact is finished.

A screenshot of a cell phone

Description automatically generated

Figure - Evolutionary Prototyping Plan

### Chapter 3.2 – Methodology

The first method for this project was to iteratively test data on redefined/developed algorithms which met the criteria of gathered requirements and sought-after results. However this approach was naïve to start with, as it left the project to be taken in many directions which could be said adaptability would allow for any needed changes made but this is what led to taking more time to test and develop starting algorithms having less focused direction. This started when developing some image processing techniques in “Spyder” to introduce machine learning to image processing. The algorithms developed where very basic and did not achieve very much but allowed for direction to be taken into cascade classification. This algorithm was developed with only one image and was the reason why it preformed lower than initially expected (see figure 1 below for pedestrians counted as pedestrians).

A group of people in a store

Description automatically generated

Figure - pedestrians counted as pedestrians

As the image depicts the algorithm did not achieve results close to what can be produced (see figure 11 Code in the Appendix under figure 46). After researching multiple image processing techniques, using cascade image classification did seem to be a suitable algorithm for detecting galaxies as its “trained with a few hundred sample views of a particular object” for example a spiral type galaxy against an irregular type galaxy but when comparing spiral against barred spiral galaxies the accuracy of this algorithm would quickly diminish when considering the basic nature of the algorithm. This is where extra time was put into researching other possible alternatives.

The cascade algorithm being used to train and test data independently this would limit the accuracy on the test data due to the low number of parameters that can be used. Finding a new algorithm became the next focus. Using convolutional neural networks best suits the project and scope when considering the adaptive nature of redefining parameters and multiple layers to best outperform earlier iterations of each algorithm developed. Using CNN will allow for techniques like “Average Pooling”, “Max Pooling” (Brownlee, 2019) and making use of multiple “convolutional” , “dense” (Brownlee, 2019) and other layers to best suit the dataset provided (Galaxy images containing all classification in Edwin’s Hubble Classification Scheme.” (Abraham and van den Bergh, 2001)). As the cascade could only perform well with basic images that contain the same object, convolutional layers allow for precision on specific portions of pixels and which with the use of max pooling techniques can make use of data augmentation as the images presented in the dataset are rotation, scale and translation invariant. However the use of these invariants as of now can only be theorized to be put into use, overfitting could be reduced by displaying images in different ways, splitting up the same image to represent the same classifier could reduce overfitting as no image would be utilized more than once, this would increase the computing power needed (and time needed) but should produce higher results than using the same image through each layer without being utilized fully by data augmentation.

### Chapter 3.3 – Choosing the right dataset

Choosing a dataset that can be best utilised by machine learning includes many factors and approaches that need to be discussed and chosen with reason. There are many public space catalogues like the GAMA catalogue which has recorded over 10 surveys of galaxies, stars and many other extraordinary wonders of the universe. However many of these can be partitioned into useful datasets the amount of useless imagery and parameters for unique sets of data like the “SDSS-IV” (Sloan Digital Sky Surveys | SDSS, 2014) survey, with a decade of construction and further gathered results focuses “extending precision cosmological measurements” (Sloan Digital Sky Surveys | SDSS, 2014) to an early cosmic phase in history. This does support what was earlier said in the introduction with respect to the importance that using machine learning for galaxies too large to be efficiently supported by physical labour classification, with even further information of each image being taken further with each survey offering different type of data corresponding to many cosmological measurements like heat signatures, life span, size, colour and certain conclusion drawn from these measurements themselves. However, the current software being developed cannot fully support such a large spectrum of data and the focus is on galaxies morphological current state classification. This means the data regarding anything not suited to the galaxies morphological state will be disregarded and unused.

Some of these datasets available through the GAMA catalogue (GAMA | Galaxy and- Mass Assembly, 2020) can be partitioned to only use specific imagery however this action still requires further testing and implementation of the already classified. This type of testing is required due to the large amount of variance each galaxy will hold due to surveys using different parameters and variables to store each image. This will result in too much data partitioning which could be reported independently with the amount of unique functions, library’s, methods and time needed to just get a dataset through this route.

The limitation previously discussed with pre partitioned data pushed the route to be taken with a previous competition hosted by Kaggle called the “Galaxy Zoo – The Galaxy Challenge” (Galaxy Zoo - The Galaxy Challenge | Kaggle, 2014). The amount of work and effort put into this challenge made it incredible useful to utilize the data already present in the data section of the challenge. However, the one disadvantage of using a pre partitioned dataset means the machine learning algorithm being designed is limited due to the “37 weighted probabilities” restricting the use of other galaxy catalogues.

### Chapter 3.4 – Dataset Chosen

From all considered in the previous chapter, the “Galaxy Zoo – The Galaxy Challenge” (Galaxy Zoo - The Galaxy Challenge | Kaggle, 2014) offers the most benefits. However, some changes in the direction of this project need to be addressed before any further advancements can be made. With the initial focus of this report (especially in the introduction) the problem at hand is not completely a classification problem but with the focus the weighted probability distribution developed by the decision tree in figure 14 means testing the data will not be to classify out of basic categories. Although the weighted probabilities can still be looked at as a form of categorical distribution as each weighted probability dictates the likelihood the image is a certain type of class. Instead of definitively stating the category of a galaxy but the chance it may be of that class which in turn will allow for results to depict specific outputs instead of a few generalizations of what the galaxy type is.

Table - Datasets chosen Descriptions

|  |  |  |
| --- | --- | --- |
| File Name | File Size | Description |
| Images\_training | 800 MB (839,270,083 bytes) | JPG images of 61,578 galaxies. Files are named according to their galaxy ID |
| Solutions\_training | 14.3 MB (15,049,808 bytes) | Probability Distributions for the classifications of each of the training images. |
| Images\_test | 1.01 GB (1,091,990,290 bytes) | JPG images of 79,975 galaxies. Files are named according to their galaxy ID |
| All\_Zeros\_benchmark | Empty CSV file | This CSV will be used to record the prediction values from the test data. |

### Chapter 3.5 – Developing a Decision Tree

### 

The earlier iteration of the decision tree designed was on the foundation of a dataset focused on definitive categorical predictions with further decision trees being developed based on advancements throughout a prototype modelling design.

A picture containing text, map

Description automatically generated

Figure - 1st iteration of an earlier proposed decision tree

The basic decision tree shown in figure 12 shows the earlier design proposed. However, with the use of 37 new probability classifications a newer decision tree classification scheme can be modelled to suit this new advancement.

Galaxy Zoo’s datasets come with already defined decision trees but each one is suited based on the dataset created. The galaxy zoo website https://data.galaxyzoo.org/ consists of all the catalogues released with each one having a decision tree representing the different weighted distributions used on each one. For galaxy zoo 2’s catalogue a decision tree is already present to represent each decision made by registered participants. Below are the questions asked to participants for classifying each image,

A screenshot of a social media post

Description automatically generated

Figure - Questions leading to the developing of the Decision Tree

Image Credit: (Galaxy Zoo - The Galaxy Challenge | Kaggle, 2014)

These questions will add to a probability that a corresponding image ID has the likelihood to falling under a certain class. Each question asked (node) has an initial probability of a classification that will sum to 1.0. For each previous node answered, the subsequent node will have the “probabilities computed and then multiplied by the value” (Galaxy Zoo - The Galaxy Challenge | Kaggle, 2014)which led to the last node. From a design aspect the emphasis on a final iteration of the prototype of this model is dictated through each class down a node needing a better solution to achieve accurate predictions.

Below is the decision tree developed by galaxy zoo with the use of the questioner.

A close up of a map

Description automatically generated

Figure - 2nd iteration of a Decision Tree based off figure 13’s questions

Image Credit: (Galaxy Zoo - The Galaxy Challenge | Kaggle, 2014)

While using a prototype would allow for new iterations in the design state to be developed, newer decision trees do not need to be further developed or changed due to each weighted probability being represented. The focus on new iterations of each learning algorithm is emphasized on the idea that each node further down the tree represents a harder achievable accuracy based on how the weighted distribution is calculated with fairness and only a good solution can accurately predict an images classification probability.

### Chapter 3.6–Use of Surveys

After the development of the last prototype is developed surveys will be conducted to compare again the test data to compare how both models compare with a machine and human eye. This will allow for certain conclusions to be drawn like how the accuracy competes with A CNN and someone who understands the basic morphological state of a galaxy. The methods for quantifying human predictions will be with the same evaluation metric used for the training and validation recorded by galaxy zoo (Root mean Square Error). An example will be included in the appendix under figure 54 with all the surveys recorded present in the folder of this report.

### Chapter 3.7– Description of the type of Requirement Gathering Process

The requirements will be gathered from an extensive and thorough related work review of similar projects containing the same type of datasets and architectures. These projects will be gathered from online case studies, publications of research, papers and many other trustworthy and validated pieces of work. Through this type of gathering process and understanding on the type of requirements other projects have, this technique will create requirements best suited to the project being developed with certain benchmarks deemed acceptable in the field of image recognition in machine learning with the primary focus on galaxy datasets being utilized by Convolutional neural networks. Other works that take in different types of data sets and utilise with different architectures will be taken into considering when designing some of the functional and non-functional requirements however will on them with less influence. Due to the prototype life cycle of this project, new iterations of function and non-functional requirements will be added in new implementations of the model. However, to follow the structure of the report, any new requirements will be discussed in later chapters in the design stage.

### Chapter 3.8– Description of where the requirements Derived from

These requirements derived from previous similar projects developed for other galaxy classification models. Previously discussed in chapter 2 – literature review, related work and models developed where contrived from the idea that galaxy data sets recorded by the GAMA catalogue or the galaxy zoo data set allowed for various sizes of the datasets to be tested. Making use of data augmentation and varied datasets allow for adaptable scripts to be written with useful library’s like Keras. These are mostly support by python 3.0+ with most IDE offering quick access and useful built in functions which can support the development stage of the prototypes. Some of the library’s to be used make use of counting time a model has run/taken to finish which shows if the model can run on normal time constrains. Classifying a highest number of galaxies depicts the success of a model but with many other related projects aiming for and exceeding 80%, this creates a stable requirement to aim for.

### Chapter 3.9 – Functional & Non-Functional Requirements

Functional Requirements

* CNN will be used with multiple ML algorithms between updated layers
* Algorithm Must be able to correctly classify 80% of images
* All images in the dataset must be used on end algorithm
* The programme must be executed in python 3.0+
* CNN Layers must make use of image invariance for data augmentation
* Algorithm is to be developed in a supported IDE

Non-functional requirements

* Algorithm can make use of varied sizes of the datasets
* Datasets other than galaxy’s zoo should be used to demonstrate tailored algorithm’s (layers)
* The software should be able to execute on multiple OS
* Time to test/train data should be within normal time constraints

Chapter 3.10 – Hardware and Software Requirements

Software being used for developing the prototypes is in Anaconda with Spyder IDE on windows 10. Necessary imported libraries can be seen implemented in chapters 4 and 5 but go as follows,

* import Keras
* import h5py
* from keras.models import Sequential
* from keras.layers import Activation, Dense, Dropout, Flatten, Conv2D, MaxPooling2D
* import os
* import warnings
* import matplotlib.pyplot as plt
* import numpy as np
* from tqdm import tqdm
* import pandas as pd
* from keras.preprocessing.image import ImageDataGenerator
* from keras import backend as K

Hardware Requirements do dictate the performance on weather a machine can run later developed prototypes, the current Computer specifications these prototypes will be run on go as follows,

* Processor: Intel® Core™ i7-4790 CPU @ 3.60Ghz
* Installed Memory (RAM) 16.0 GB
* Free Space up to 3GB

Graphics card will not be used with the Python script.

## Chapter 4: Development

### Chapter 4.1 – Setting up Model

Setting up the model section will be developed after each prototype is finished however for a finished instructions on how to up a new environment with the developed prototypes please see Appendix A – Setting up Model.

### Chapter 4.2 1st – Design

Following the prototype methodology chapter 4 will be conducted through iterations of the implementation stage from design to development and finally testing. However due to the nature of this project the first iteration will be focusing on developing a model around the decision tree and dataset previously discussed in chapter 3. By going taking this approach the focus on later iterations will be to implement better proposed models based on a series of testing conducted at the end of each iteration. Also as previously discussed in chapter 3.7, a lot of the research gathered can act as a starting point for the first model as fully designing a finished convolutional neural network from scratch is too farfetched when considering the amount of fine tuning and gathered conclusions from testing is only gathered by the end of a model. That said firstly designing a python script which can take the 37 classes from the CSV solutions file and assigning and ID to each image will be first implemented in the development stage. Through this a basic CNN model will be developed with the idea more layers can be added, changed or deleted. When the testing stage occurs measurements from the training and validation set will be taken into consideration before moving on into the testing stage. This is to make sure that when testing the new data on a trained set, accurate conclusions can be drawn after a model proposes results instead of looking at results that come from a poorly developed model.

The first step is designing a model that can access the galaxy Images, assign a galaxy ID based on the column of data in the solutions file corresponding to each image with an ID for 37 classes. Then to send this to a training/validation training model to gather weights and train the models as well.

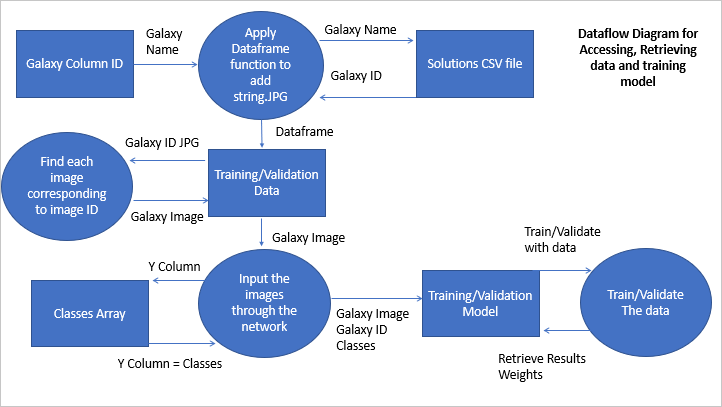


Figure - Dataflow diagram for accessing, retrieving data and training the model

Figure 15 shows the design for accessing the data from a solutions file and applying a function to amend a CSV reading array to gather ID’s of each image which inherently are the start of the image JPG names. So, for galaxy “100018” in Column 1 (index position 0) on Row 2 (index position 1) this can be simply amended to add “.JPG” to find the corresponding image.

This can be achieved by using a function proposed by Vijayabhaskar J who uses this to return name+”JPG” with his own Dataframe data, “traindf[“id”]=trainf[“id”].apply(append\_ext)” (J, 2018). Here he uses two separate arrays for the model he is designing however this code will be amended to suit the one array of data needed for the dataflow diagram presented in figure 15.

Developing a Convolutional neural network model will require trial and error, but the first state is getting an idea of the type of model being design. However, there are some common practices and popular techniques used in this field which has been looked at thoroughly by Stanford university which will aid in developing the Convolutional neural network architecture.

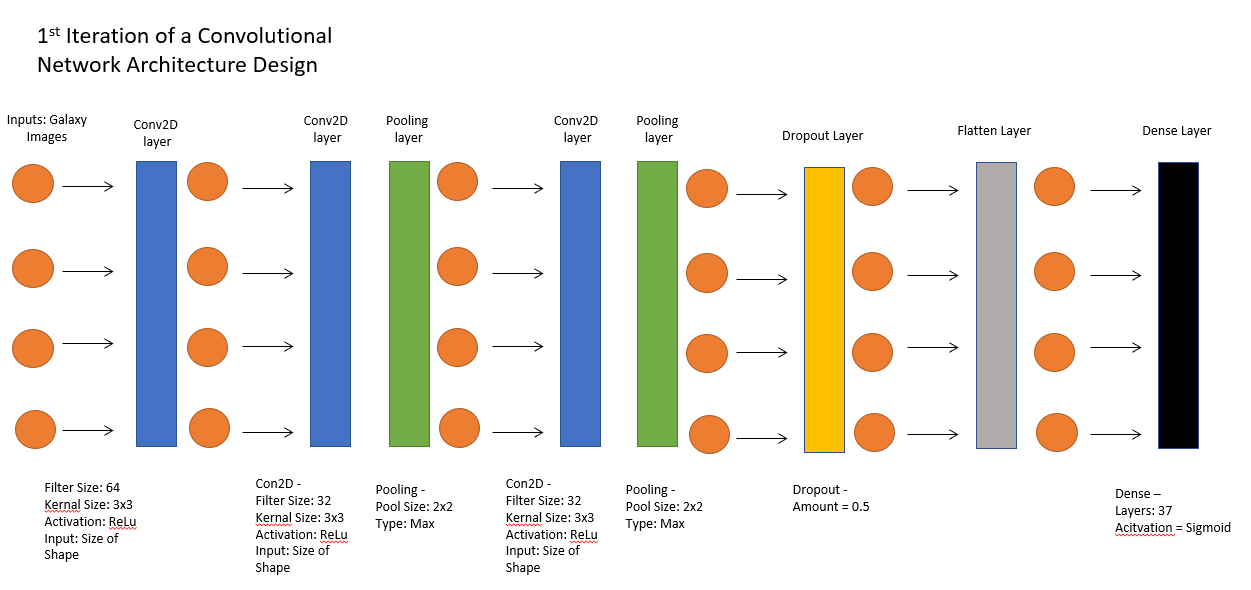


Figure - First iteration of the CNN architecture design

### Chapter 4.3 1st - Convolutional Network Layers and Parameters

### 

Choosing the starting parameters for the convolutional layers will most likely be changed later through testing due to the nature of this model learning on the parameters given. However, previously stated chapter 4.1 there are common techniques used. The filters are chosen at a lower value and moved up to 128 based on the common practices in other similar works(Rosebrock, 2018). These filters will act by applying various distribution techniques like normal gaussian but with different initialization criteria, each filter will be trained slightly different (Saama, 2017). Also doubling the filter size will help the network learn hierarchical features, however this is beneficial when comparing entirely different objects, the sizes of galaxies can be dramatically different. For example, when looking at an eclipse galaxy, this may be technically a larger galaxy when compared to a spiral type, the shape normally depicts an illusion that a spiral is bigger. This creates and illusion a galaxy is bigger or smaller but can still aid in hierarchical features of distinguishing probabilities. When choosing the kernel size, it is rate to exceed 7x7 and with many images exceeding 128x128 (height, width) commonly use 3x3, this helps in reducing volume size and allows for learning larger spatial filters (Rosebrock, 2018). The use of strides may be better to reduce spatial dimensions for the output volume which dramatically reduces the amount of computing power needed for the network however this is commonly effective for dealing with large image datasets. This also paired with a higher kernel size for example 7x7, this suggests ways to optimizing the performance based on computational power may need the use of strides in the convolutional layer.

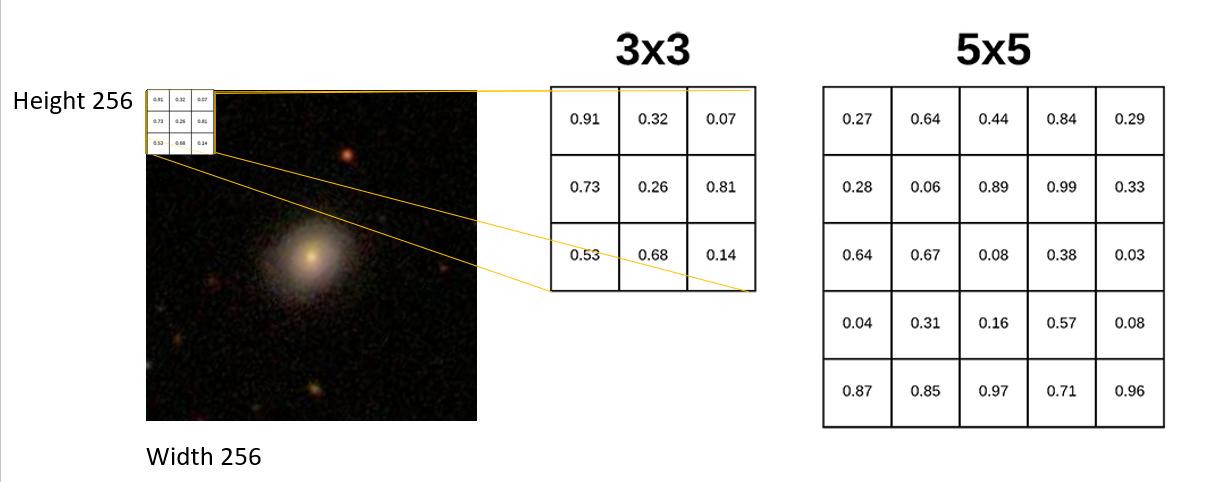


Figure - Kernel Size Example

Figure 17 depicts the use of a 3x3 kernel size on the galaxy image dataset being used in the implantation. Due to the size no use of a higher kernel size than 3x3 is needed and in turn using strides is not needed either.

Another common practice for convolutional neural networks involves using a pooling later in between each convolutional layer which has been applied in figure 16’s design. This will progressively reduce the spatial size of the image’s representation; this helps to reduce the number of parameters and computation in the network which suggests this will help with any overfitting the network may produce.

The fully connected layers near the end of the model (shown in figure 16 above) are great to reducing overfitting which is especially true for the dropout layer. Very frequently a dropout layer is used to reduce overfitting, this can be done by using ensemble methods to make predictions on each model from different neural networks (Brownlee, 2018). Though this will create a problem as this approach requires “multiple models to be fit and stored” (Brownlee, 2018) and by doing this a large amount of time will be wasted when compared to using a dropout layer. By using this dropout layer near the end, forces the nodes to take responsibility for the inputs which suggests that mistakes can be corrected from prior layers creating a more robust model better at generalization.

The use of another fully connected layer is needed to pass through the 2-dimensional data into a 1-dimensional array to allow for the final layer to interpret the data. by using the flatten layer the 1-dimentsional array becomes a single long vector, this allows for the predication with the lastly connect layer to occur. There is no reason to suggest this produces a loss of accuracy as even with 3D models the use of flatten layer is used without any loss (Jin, Dundar and Culurciello, 2014). This suggests a 1d array can act as an effective substantive for the 2d data being passed through.

After the model has been fully trained all the learning features are applied to the last fully connected layer containing the 37 possible outputs when applied with the training and validation generator the model to the 37 weighted probabilities an image is a type of class. The activation for the finale dense layer was chosen to be sigmoid and not SoftMax. The reason behind this was to develop outputs of probabilities between 0 and 1 which correspond well to the data given in the training solutions file. Figure 18 below shows the type of data being used to predict a probability, applying SoftMax would cause the dense layer to output either 0 or 1’s not in-between. This would potentially kill off any reliably probability distribution.

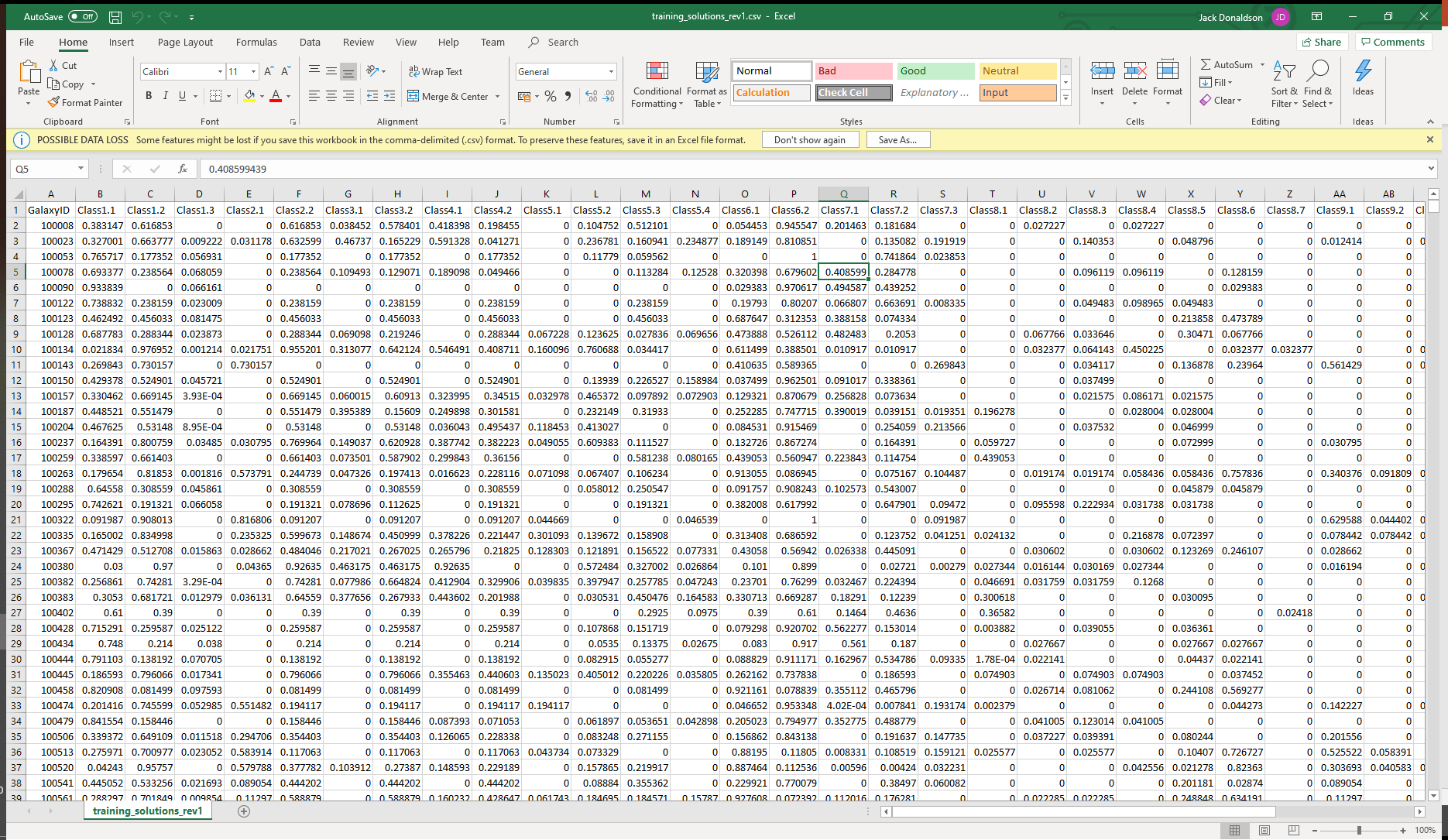


Figure - types of data for each 37 classes

Through each convolutional layer an activation is set before feeding any data to the next layer. This is to ensure the network does not collapse into a single linear function but instead the use of non-linear activations like ReLU at the output of data before feeding into the next layer. The choice for a ReLU activation is important to note as the parameters in the convolutional and fully connected layers will be trained with a gradient descent so that the network is consistence’s with computing the labels in the training set for each image. Other non-linearity activation functions where considered like the sigmoid and tanh functions; however, these fell short when comparing the acceleration of the convergence of stochastic gradient decent when being compared to Reu. This is not to say the design choice for this activation function is definite, due to common practice and early an early prototype being developed this can be subject to change. There is a chance if too many units die in the training stage, ReLU may be too fragile to use and the use of leaky ReLU may need to be implemented in the 2nd iteration(CS231n Convolutional Neural Networks for Visual Recognition, 2020).

### Chapter 4.4 1st - Designing a Dataflow Diagram for testing Validation and Accuracy

After the convolutional layers are developed testing needs to occur to check the accuracy and validation of the training\validation data.

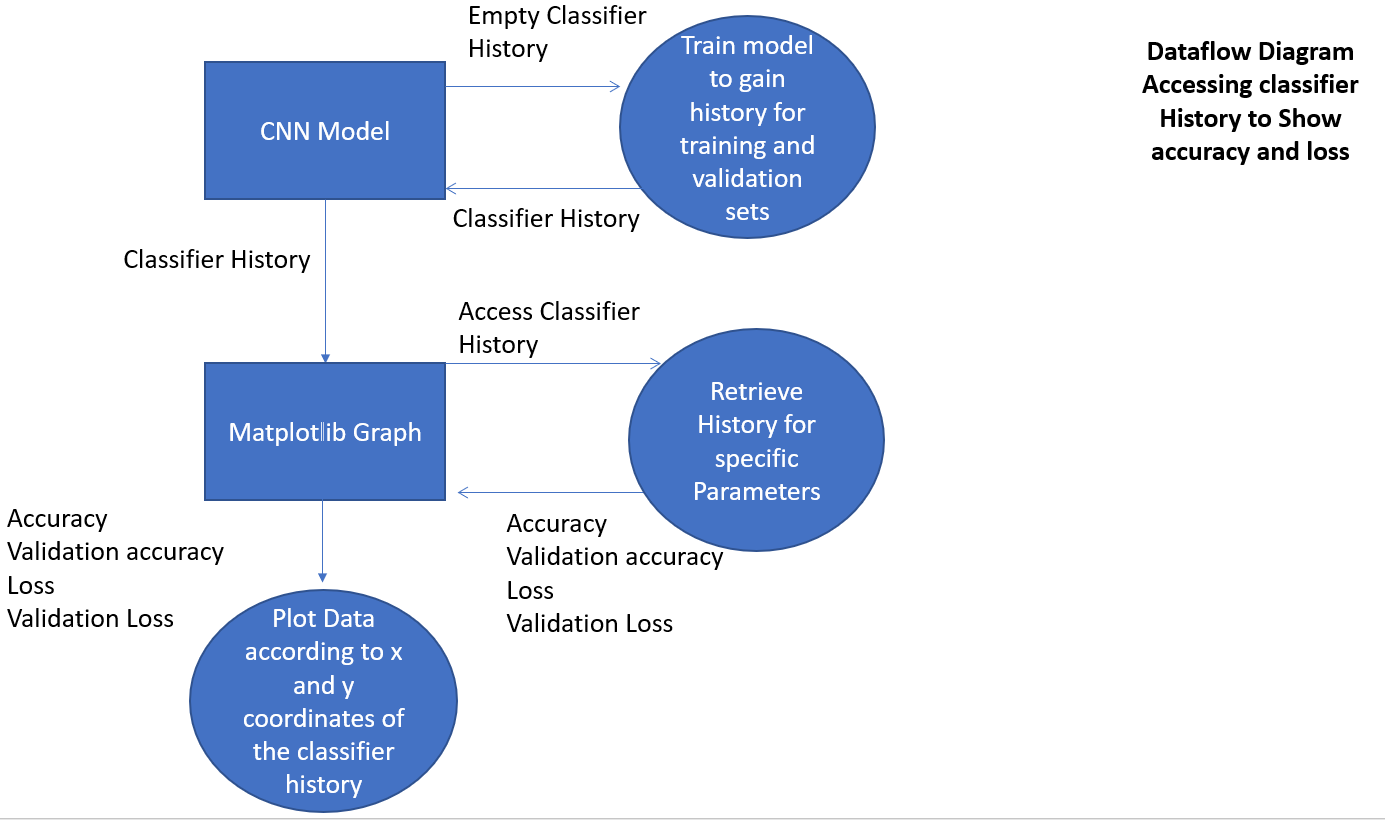


Figure - Dataflow Diagram for the testing stage

This dataflow diagram shows the stages that will be developed in the implementation chapter which will then be used to gather results in the testing chapter. As the convolutional network has some built in uses, the classifier will develop history which can be gathered at the end of training/validation to see the results of the model. This will be used as insight for developing a model which can surpass the accuracy and loss of the previous iteration.

### Chapter 4.5 1st – Implementation

The implementation requires some library’s to be imported or installed using either PIP command lines or conda command lines based on what software is being used.

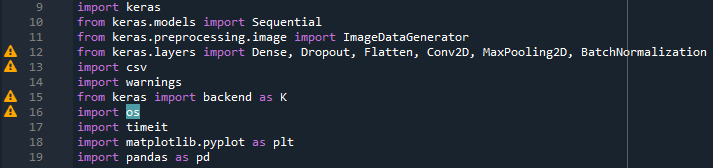


Figure - Library’s used for 1st prototype

The Keras library offers all the convolutional, pooling, flatten and dense layers needed to be added to a model. CSV and pandas offer some built in functions to read CSV files for headers, rows and columns. Other less noticeable library’s offer some unique customization to the code to help better depict what is going on throughout the training and validation process.

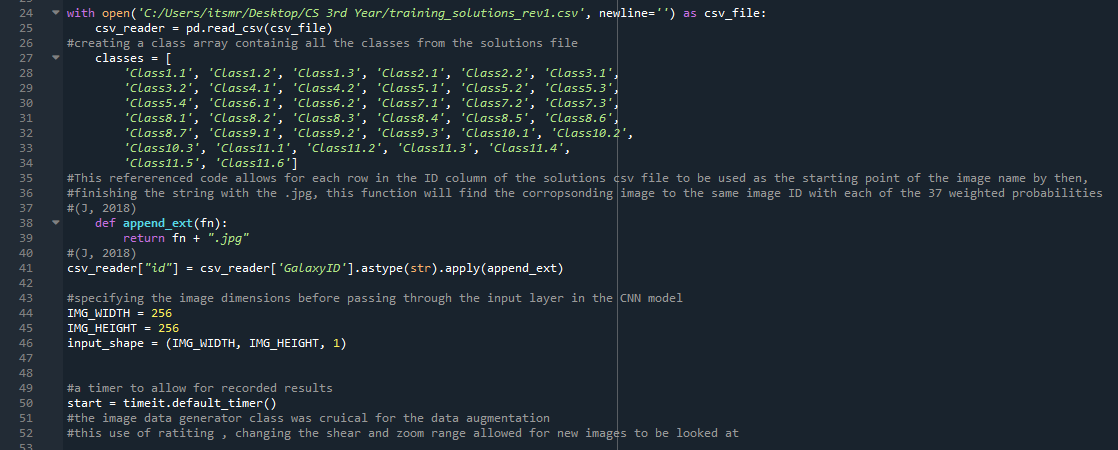


Figure - initializing some variables and arrays for later use

Here is the start of the development with the use of the classes array being developed to use later in the training and validation classes. The use of the earlier explained function on “def append\_ext” on line 38 has been amended with the “csv\_reader” code on line 41 to suit this model instead. Without this use of this function a custom generator would have needed to be developed in order to gather the images and ID’s. Other variables have been declared to use later as well with image height and width being the size of each image. The “start = timeit” will also be used later onto to conclude final time on the process run time.

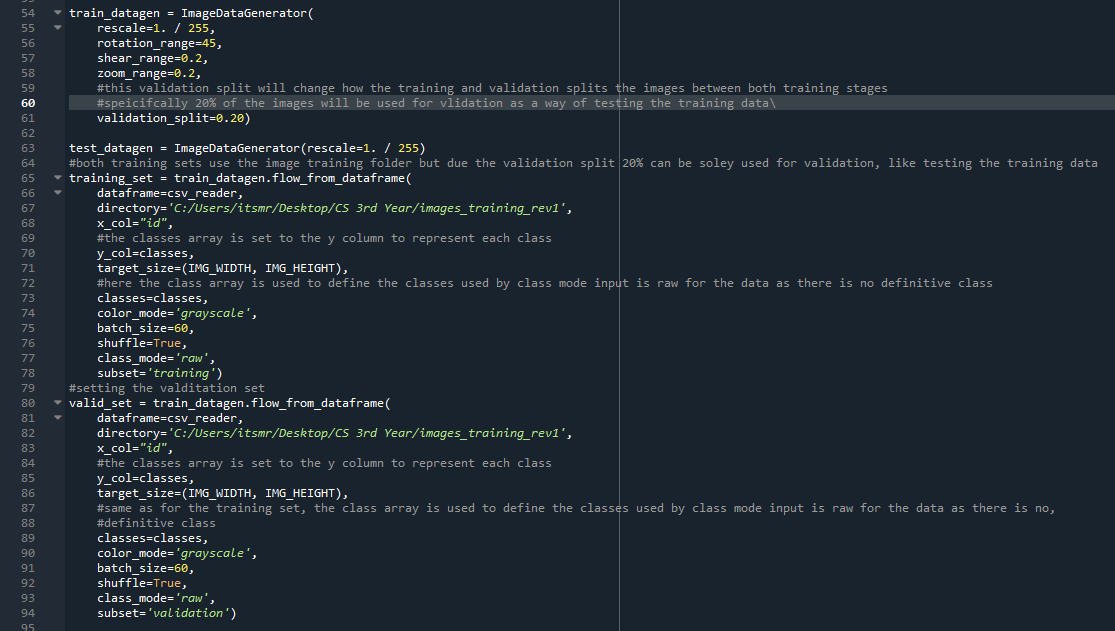


Figure - Creating the data augmentation, training and validation classes

The first class in figure 22 allowed for multiple data augmentation techniques to be applied without external python scripts needed to manipulate data in specific ways. Keras offered many useful classes and functions with this being the most noticeable in saving time. One parameter to note is “validation\_split” on line 61 which allowed for the training and validation classes to make use of subset training, allowing for training data to be tested at the same time changing weights.

Each of the training and validation classes follow a similar design in accessing the Dataframe instead of a dataset directory. This was recently added in 2018 and stops the need of writing a custom generator. The classes array is also used here to set the y column to the 37 distribution classes.



Figure - Creating the CNN model based on the Design chapter 4.2

Creating the model design in chapter 4.2 was very simple with the use of the keras library offering all the components and built in parameters for each layer. One problem encountered here was at two different points of the code. Firstly, trying to create the length of classes as the output for the dense layer could not be passed through the built-in parameter instead a definitive integer was used (37 on line 112). No examples exist where the number of classes changes at the output for this type of classification but to store an integer value here from this development would require for a separate integer variable holding the value between the model. The 2nd problem encountered occurred when fitting the classifier with the “model.fit\_generator” on line 127. Here a few runs where tested (later discussed in chapter 4.5 testing) where the current CPU of the system would crash. This was due to the parameters demanding to much computing power.

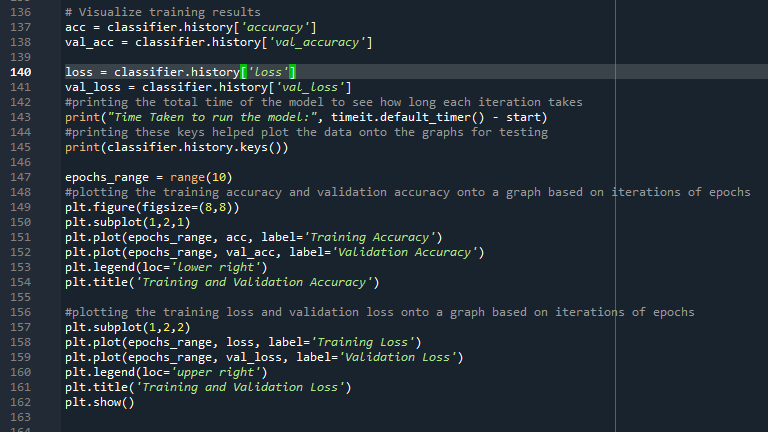


Figure - developing the graphs with use of classifier history functions

Figure 24 shows the use matplotlib library to plot the training accuracy, validation accuracy, training loss and validation loss to allow for visual testing and final conclusions to be drawn in the testing stage. Line 145 of figure 24’s code shows the use of function to get the keys required for plotting onto the graph, this helped receive the parameters as well as the data for each iteration of the training and validation classes with the CNN model.

### Chapter 4.6 1st – testing

Testing occurred at the end of fine tuning some of the parameters with one example in figure 23 line 117 of the code, learning rate variable was declared to see if the weights trained more effectively but resulted in major loss when deferring from the base learning rate of the Adam model. This will be staying the same default value 0.001 which same value for the learning rate still left, this is due to further testing in later prototypes in case with a later model is more effective with a different learning rate.

Figure 24 in the last chapter discusses the implementation of the graphs which proved to be highly useful when determining the effectiveness of the model. After few iterations the classifier history stayed almost the same with very few different results. below is the first iteration of testing,

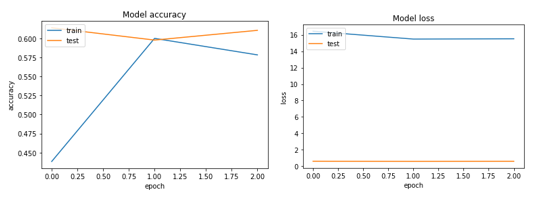


Figure - MODEL ACCURACY AND LOSS PLOTTED AGAINST EPOCHS

Figure 25 is only through 3 epochs but from this certain conclusion could made. First the model works and still at 60% accuracy for the first model was a great 1st prototype. Secondly the hardware constraints where put near the maximum of the CPU and RAM operating speed meaning that the number of epochs can be increased to gather an overview on the model’s performance. Therefor below in figure 26 contains the overall performance of this model over 10 epochs.



Figure - model accuracy and loss plotted against epochs (2nd iteration)

The hypothesis on setting a higher value for the number of epoch iterations suggested from figure 25 the accuracy should keep climbing to higher results with a certain threshold reached. However, this was not the case in figure 26 showing this series of testing conducted. The results clearly show a decline in validation and training loss which shows the model is retraining the weights in the right direction. The training and validation accuracy on the other hand seems to each a certain range of accuracy between 60% and 58%. This suggests the model may not be training the weights with enough parameters as the loss with a few other parameters before the final choice on the current chosen parameters shown in earlier implementation figures pushed for a much higher validation and training loss. With SoftMax and a higher number of filters preforming dramatically worse. From this a higher number of convolutional layers will be added in the 2nd prototype with added functionality to better test the training and validation data.

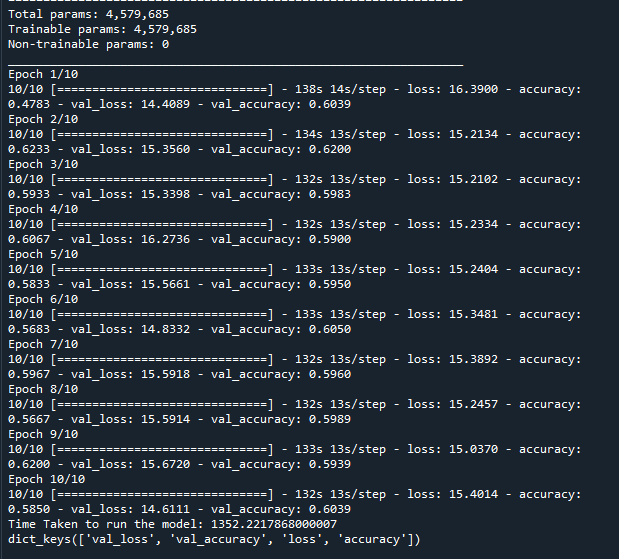


Figure - Model Runtime

## Chapter 4.1 2nd Prototype

### Chapter 4.1 2nd – Design

A few new design choices have been looked at for the second prototype iteration. Firstly, for testing new data a new dataflow diagram needed to be developed and with some further research a method of using a HDF5 with an added imported library called H5PY was discovered. Utilizing this library means that the weights trained at the end of training the model with training and validation sets can be saved. These files can then be uploaded into the 2nd python script responsible for executing the test data with the previously trained model weights, saving huge amounts of time (MachineCurve, 2020).

A few architectural design changes were made with the addition of another Convolutional layer pushing each layer up by its previous amount \* 2 to follow the common practice of developing a convolutional neural network previously discussed in chapter 4.1 1st iteration of the prototype. However, a few dropout and dense layers where added with an activation layer in-between to keep the flow of data consistent throughout.

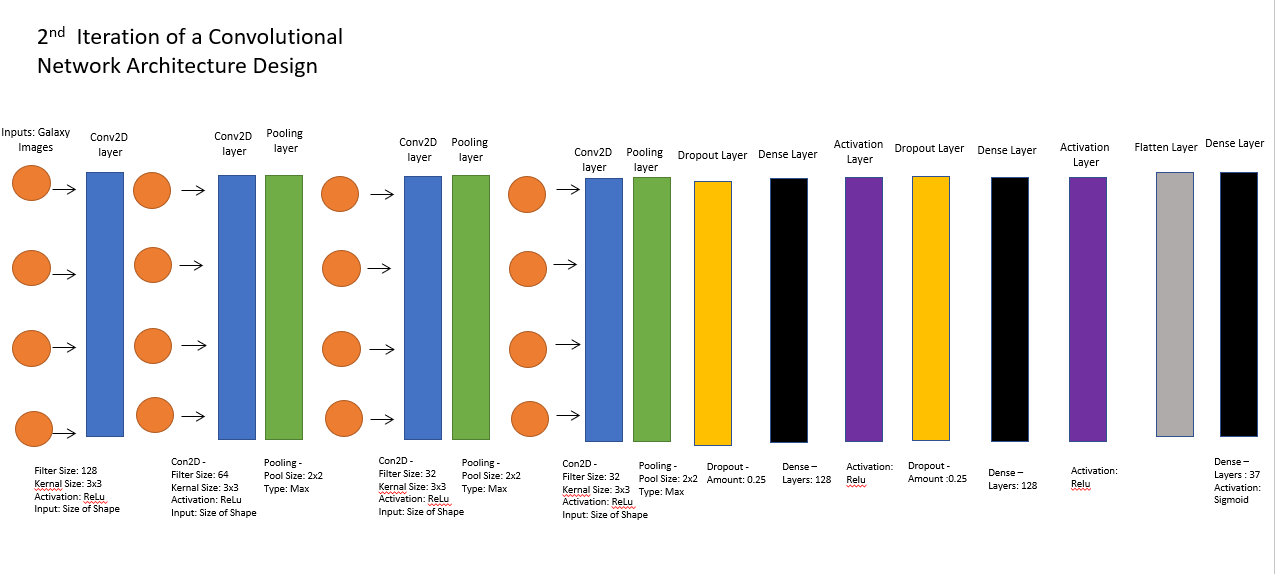


Figure - 2nd iteration of the CNN architecture design of the 2nd prototype model

The two split drops out layers have been added to try and split the amount of dropped weights down by half through two iterations. The hypothesis behind this is the weights being trained can use a bigger portion of data but through two iterations there’s two opportunities for learnable factors to be found. The activation layers where needed as some fully connected layers cannot use the activation function as a parameter. The two newly added dense layers should reduce overfitting even further with the hypothesis for these layers to add better weights for the testing model.

The 2nd python script being designed for the 2nd prototype will be using the previously design model to test on new images previously not used by any of the training set to write new probabilities to an empty CSV file. A dataflow diagram below in figure 29 shows the flow of data being written to a new CSV file.

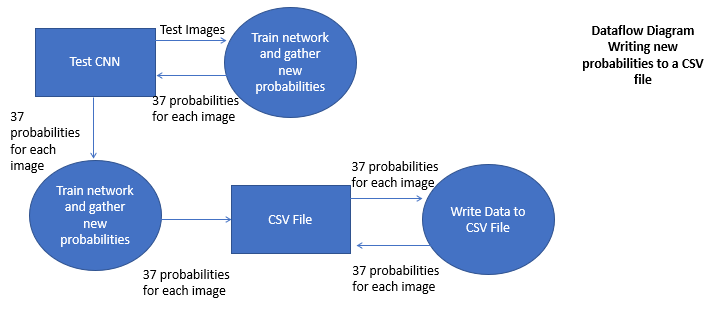


Figure - dataflow diagram for writing new probabilities to Empty CSV file

### Chapter 4.2 2nd – Development

Through development an unfortunate flaw in the model was discovered. The proposed model in chapter 4.1 2nd was unable to execute and run on the current computers hardware, refining this to develop a model just at the limit of the systems maximum computing power was developed instead. Therefor an amended CNN architectural design was designed to suit this new model. See below in figure 30 the newly designed Model from which the development process will follow.



Figure – amended 2nd iteration of the CNN architecture design of the 2nd prototype model

Important note to make is that the current convolutional layers where removed by one to just run on the current hardware constraints with the filter size reduced to the previous 1st iteration of 64, 32, 32.

As this 2nd prototype works of the foundation of the previously developed model most of the code for the first python script remains unchanged except for 4 parts. Firstly, to make use of the H5PY library it was first imported (see figure 49 in the appendix for code) and then later used after the model had been trained (see figure 50 in the appendix for saving weights). The change to the model was changed to follow the amended model designed in the previous figure 30.

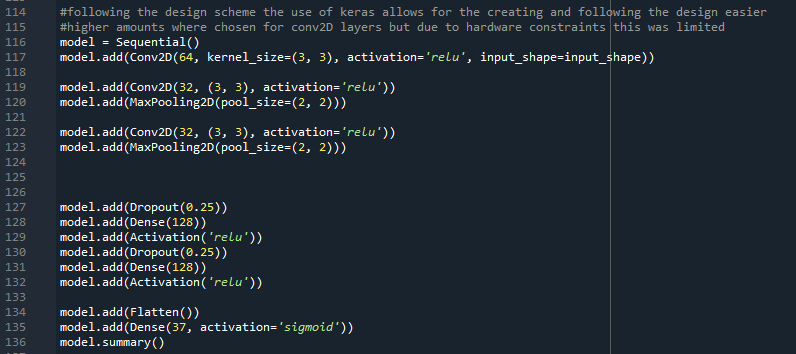


Figure - 2nd iteration of model developed following amended architecture design

The last amendment to this prototype 1 code to finish the first script for the 2nd prototype was adding more functionality to the programme.

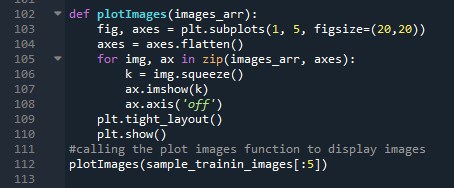


Figure - last amended code to the first python script

The code shown in figure 32 (see figure 51 for produced images appendix) was to gain insight to how the images where being shown before being trained into the CNN.

The 2nd python script responsible for testing the design model no new test images starts off with all the same library’s and CNN model with one exception.

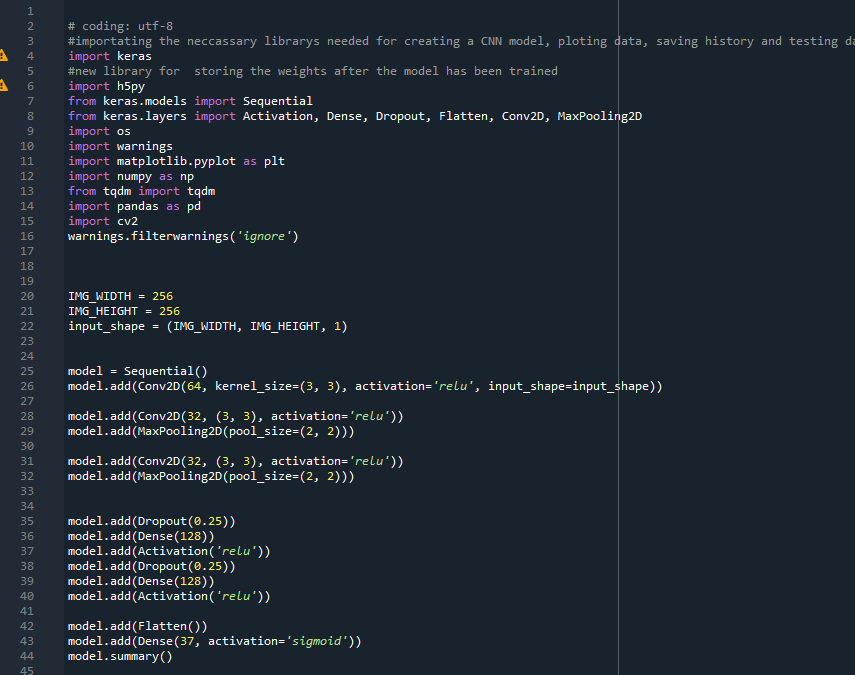


Figure - Added library for additional functionality

Figure 33 shows the same model developed for the testing data with the additional a tdqm library which shows the runtime of the model with clear functionality.

The sciprt is developed differently compared to the training and validation scprit, this is to add a way to wrtite the collected prababilties to a csv.

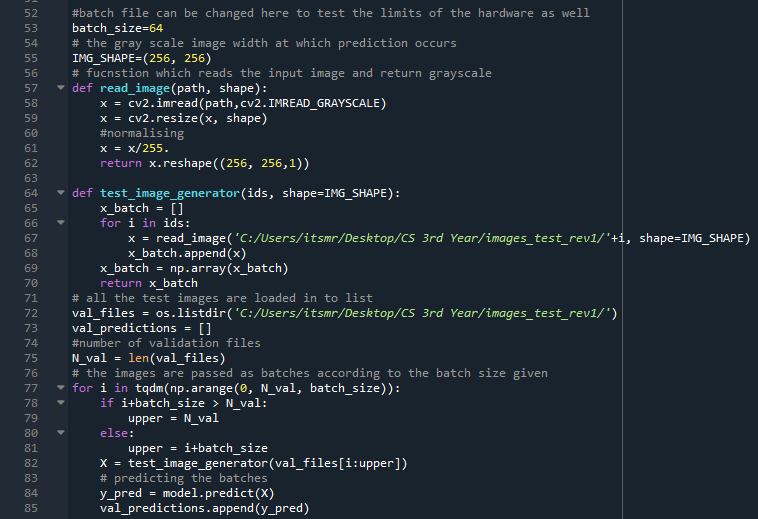


Figure - test image generator

Here the use of a custom image generator is developed to read a number of images based of the batch size, this was developed in the hypothesis by accesing the images in batches, sintead of independtantly, time constraints would be cut down. A y predicitions array was then created in line 73 of the code as an empty array which can be then be stored. To make use of the new functionality provided by tqdm in line 77 of code in figure 34. Through this function gets rid of the messy console output of each epoch (shown in figure 52 in the appendix ) and simply shows a bar clearing up the console. This function also trains the model to predict probabiltiies for a given image (line84 of the code in figure 34). Then by predicting the model these proababilities can then be saved to the earlier created empty array.

The last set of code produced for this python script makes use of the predictions going to be stored into the “Val\_predcitions” array in line 91 of figure 35.

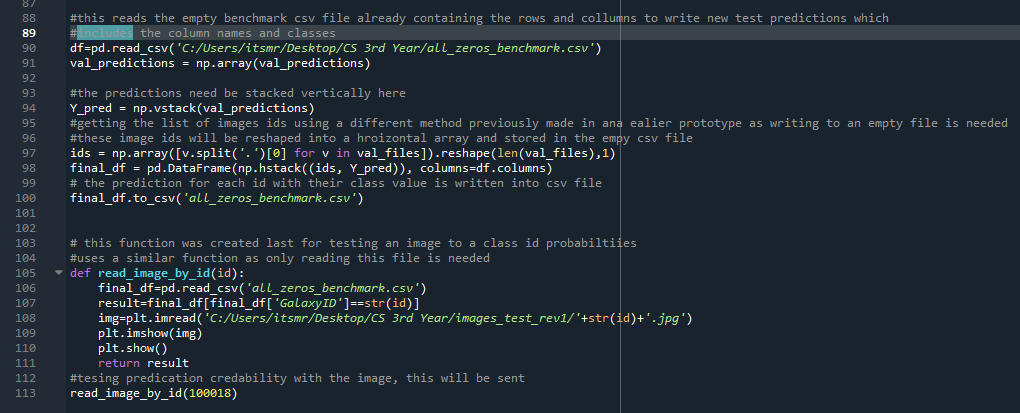


Figure - writing predictions to empty CSV file and reading values based on image ID

After the model and tested the new images the predictions will be stored and sorted into a horizonal array to iterate through each galaxy image ID. This will then write those predictions into each row which can then be accessed by the last function. The idea behind this function is by looking at some sample images and seing if the probabiltiies match to the image based on the series of questions asekd in the decision tree shown previously in chapter 3.5 (figure 14 ).

### 4.3 2nd – Development

As previously in chapter 3.7 gathering new requirements are discovered with new iterations of each prototype. Here the finished functional and non-functional requirements are stated below.

2nd Functional Requirements

* CNN will be used with multiple ML algorithms between updated layers
* Algorithm Must be able to correctly classify 60%+ of images
* All images in the dataset must be used on end algorithm
* The programme must be executed in python 3.0+
* CNN Layers must make use of image invariance for data augmentation
* Algorithm is to be developed in a supported IDE
* Model must be able to read and partition data between CSV files
* Model Must be able to Write new Test data to CSV files
* The Model should read a binary stored data file in the test between the training and test script

2nd Non-functional requirements

* Algorithm can make use of varied sizes of the datasets
* Datasets other than galaxy’s zoo should be used to demonstrate tailored algorithm’s (layers)
* The software should be able to execute on multiple OS
* Time to test/train data should be within normal time constraints
* The Model should be able to use binary store features like H5PY to quickly retrieve trained model weights
* The model should be able to change its parameters based on new development in areas
* The model should allow for graphs/System reports to be seen during rune time of each python script.

### Chapter 4.4 2nd – Testing

The series of testing occurred after the model trained on the same training and validation data with the use of data augmentation in increase the amount of image variance. This is achieved in the image data generator for the first python script for the 2nd prototype making use of horizontal and vertical flip. Below are the results for using epochs ranging from 3 to test if the model worked with success show in figure 36 to the right. to the right shows the use of 15 epochs and stability was not reached but an accuracy stayed above 58%. Validation loss had an abnormal spike with 15 epochs occurred possibly due to a random occurrence of poor weighted dense layers.

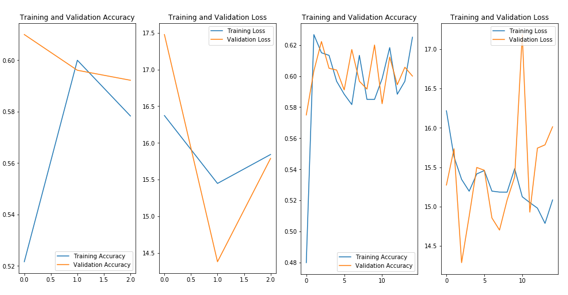


Figure - training and validation loss and accuracy for epochs ranging from 3 to 10

To find the conclusive state at which this algorithm hits a preofmrance limit further testing was conducted before the test data could be applied confidently. To test the limit of the model an epoch value was chosen at 30 to see if the accracy and loss stabilized around a certain percentage. This suggests with a model running with enough epochs the accraucy should keep rising only if by a few percent through each iteration.

Below shows in figure 37 the training, validation accuracy is shown to staiblize around 60% throughout each iteration. These results are below to what was expected with the idea the accracy would contine to climb to a ceratin point but these results depict a very unprediactable model. With an average of 58% accuracy through 30 epochs it’s close to guessing around 50% of the time, however some of the results achieving 60% do contradict this assumption. Based on this, its safe to assume some of the test data will be up to 60% accurate. The dense layers reducing overfitting should still allow for similar results to be reached.

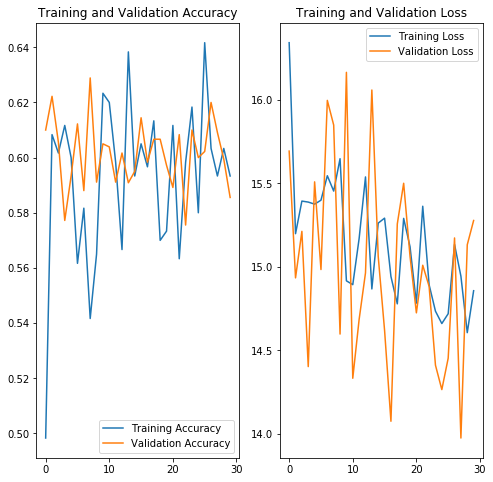


Figure - training and validation loss and accuracy for 30 epoch iterations

One successful note to take form the training and validation loss is that this consistantly lowers through each epoch meaning the weights are being trained to be better suited to the datasets. With a model consistent enough with an accuracy around 60% the model can now be applied to the test images. These images should produce 37 reliable weighted probabilities which will be compared to the galaxy test image responsible for producing each row of probabilities. As no prior test probability solutions are available through the galaxy zoo dataset, this enforces the the rout taken to gather some new deicison tree data following the

### chapter 4.5 2nd - EVALUATING TEST PREDICTIONS

The test images were successfully written to the empty CSV file however looking below at figures 38, 39 and 40 of a sample of 11 galaxy image probabilities were found analysing the model.

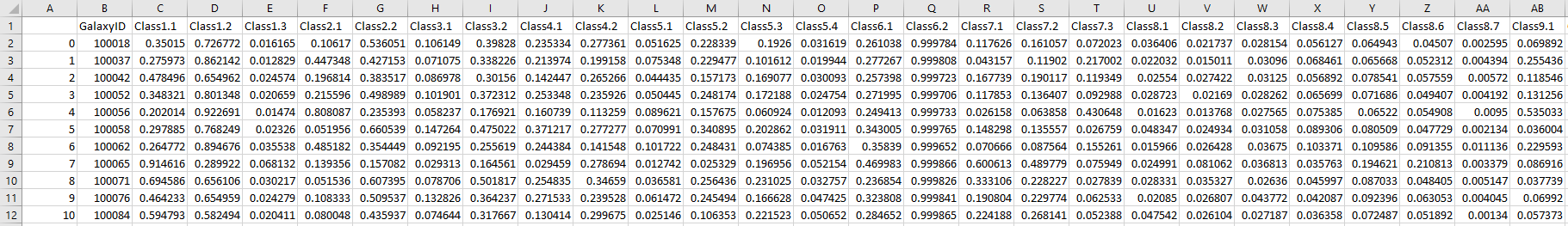


Figure - First set of test data

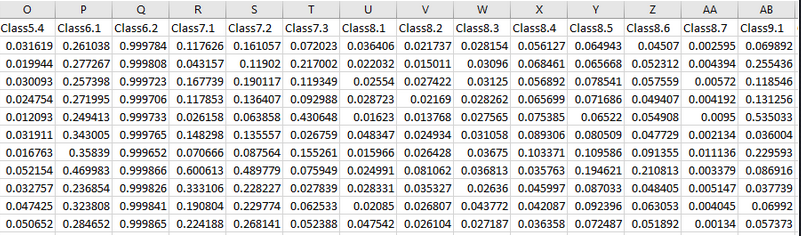


Figure - second set of test data

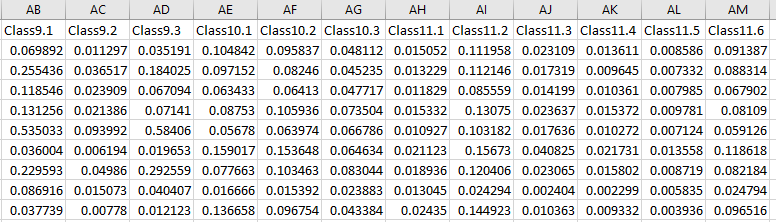


Figure - third set of test data

One clear limitation is some probabilities require hard 0’s based on the decision tree discussed in chapter 3.4 (figure 14). If a question is answered stating it has an elliptical nature, then question one on the decision tree takes the user to question 7(class 7. Onward) with only a couple other classes able to be quantified adding to the sum of one. This problem occurs based on how the data is being distributed among all 37 classes with an overall possibility that a question is answered adding to the pool of data for each row of galaxy images. This is partially the reason behind the accuracy of the model not exceeding certain limits as the distribution is not directly linked the schematic classification model proposed in earlier decision tree diagrams. However, this sample data can still be analysed in a general sense to see if certain shapes and sizes dictate the probabilities based on the broad classification Hubble scheme. The hypothesis is that classes corresponding to certain galactical morphological states should have higher values than the classes which don’t fit into the Hubble classification scheme. For example, for classes 1 to 7, and to 6 should have higher distributions if the image falls under the elliptical class. To avoid bias a sample of participants will be selected to represent the new test data solutions file to monitor the accuracy and validity of probability distribution. The same evaluation metric for gathering the results previously used in the solutions (Root Mean Squared Error) file will also be applied to the sample of 11 galaxy images.

The 5 questioners gathered and then evaluated to add the root squared mean error ( in figure 41 below) to acquire and plot new data into another CSV file to compare.



Figure - Root mean Squared Error Function

See below in figures 41 and 42 for the test data gathered from 5 surveys conducted with 11 images analysed by participants (see figure 54 in the appendix for an example of the survey conducted ).



Figure - First set of Survey Test Data

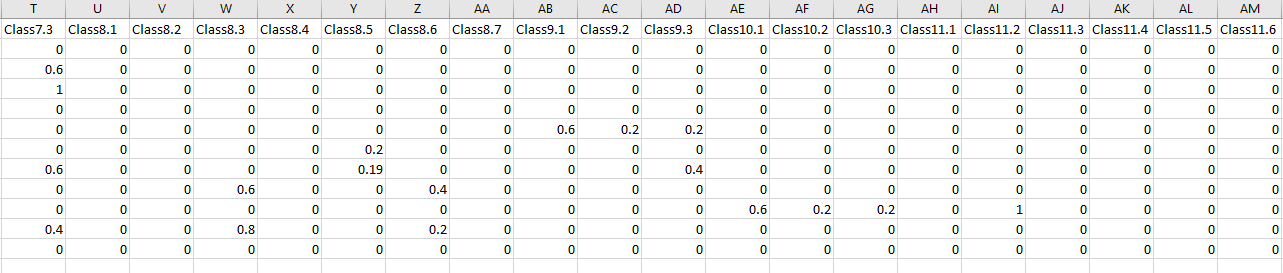


Figure - Second set of Survey Test Data

The test file contains a limited range of data available to properly deduct the complete accuracy of the test predictions written in figures 38, 39 and 40. However some high occurring values are consistent with some of the images, for example in galaxy ID 100062 (see figure 44 for the related galaxy Image) had high values for elliptical related classes (class 1.1), and in class 6.2 (anything odd about the galaxy = No) obtaining a high value of 0.999866. Some classes do follow this scheme.



Figure - Image of galaxy ID 100062

Some classes do contradict this accuracy of the prediction values with some of the uncertainty coming from a couple of images maybe harder to depict from human predictions. Galaxy’s 100065 and 100076 both contain odd morphological states being commonly detected in all 5 surveys but having a low prediction value from the test data with values 0.469983 and 0.323808 for class 6.1 stating any irregularities with these two images. This suggests the number of regular galaxies in the dataset heavily outweigh the number of irregular galaxies contained in this dataset. This is further supported by most of the classes in 6.2 (class for no irregularities) containing higher probabilities values

A solution for this would be in the next iteration of a prototype to allow for stricter distribution techniques that follow the 37 questions in the decision tree more closely. Instead of allowing for all 37 weights to be distributed without an effect to stop data being entered in certain classes when a certain series of questions met the end of the decision tree.

## Chapter 5- Conclusion

### Chapter 5.1 – conclusion

The project aim started with intention of developing a galaxy classifier that accurately separates galaxies according to their morphological state. This was achieved with a slight change of course by using probabilistic weights distribution methods. The galaxy classifier was developed by using convolutional neural networks in two different prototype iterations making use of different layers. These separated each galaxy into a range of 37 classes depicting the likelihood the galaxy is of that state. However as previously discussed in chapter 4.4 2nd – Evaluating Test Predictions, the accuracy started to underperform with further classes down the decision tree proposed in chapter 3. For certain general classes like irregular galaxies, the algorithm underperformed but exceeded in classifying elliptical and spiral galaxies. The lower 60% quartile being reached in the 2nd prototype shows the accuracy of the model however this can be improved quite a lot, 60% accuracy is enough to disregard the idea the model acted on random generalization. However, this model is far from perfect, learning how to use activations functions with CNN proved to be useful with ReLU being very popular. Although even being able to fine tune some activation layers like ReLU(Rectified Linear Unit) with a leaky variant allows for datasets that have many reoccurring positives (1’s) and negatives (0’s) allow for features to continue contributing to “the model’s decision power” (Using Leaky ReLU with Keras – MachineCurve, 2019). There are still many variants of architectures to CNN models with a large number of parameters able to fine tuned to suit various datasets means that there is still many things to learn in this field of machine learning.

### Chapter 5.2 - – Summary of the Project with Critical Evaluations on each Stage

Stage 1 of the objectives presented earlier in chapter 1.2 of the report starts with gathering user requirements by either the use of a questionnaire or other use of resources. This was achieved by a thorough and concise literature review of related work gaining and understanding of current and similar projects constraints and archivable requirements. After these were gathered the first time a new iteration in chapter 4.3 depict the new functional and non-functional requirements based on the updated prototype.

Stage 2 of designing the first prototype was aided in chapter 3 methodology for finding the best suited models for using the type of dataset previously used. first was the Cascade HAAR classifier was looked at but disregarded as CNN had much more extensive research on the matter with many networks discussed in chapter 3 highlighting some of the more noticeable projects developed. The datasets were found from galaxy zoos dataset and where documented throughout this report, both in chapter 3 and references. The use of Class diagrams where considered but due to the flow of prototyping UML diagrams where not developed however for the 2nd prototype using this would have proved to be beneficial in developing the interactions between two the python scripts.

Stage 3 of the objectives focused on developing the nest iteration of a prototype with refined parameters and implementations. This was achieved by designing and developing a better CNN model which achieved similar accuracy but was able to push above 60% more frequently between epochs. This 2nd model became the finished prototype, so the testing script was developed for unseen testing data for the model find new probabilities. The use of one of the functional requirements stating the model must use a form of binary storing allowed for the weights to be passed over to this newer script cutting down run time. The focus on using CNN also allowed for focus on the type of architecture to be used but with the amount of freedom expressed in earlier chapters, the requirements where not tightly linked to a design methodology. This same architectural design allowed for versatile designs to be developed alongside testing and developing prototypes. The use of dataflow diagrams where designed to show the flow of data between reading and writing CSV file data to arrays and funnelling into training, validation and testing models.

Stage 4 of the objectives focus on testing the finished prototype with limitations of the model. This was also done for the first prototype in the 1st iteration of testing which allowed for improved designing and development to occur in the latter prototype. Limitations of the model derived from chapter 4.5 evaluation test predictions where improvements on following the decision tree more strictly was the solution for more accurate distribution of weighted probabilities for irregular galaxy types. The sections of code responsibly for reading and writing CSV files where handy with the dataset chosen however any new training data was restricted heavily by the recorded probabilities. Gathering any new data would have to undergo this process to be able to fit into the training/validation classes with the same with the same classes linked to new images. However, this is not the same for the testing script as any new images of galaxies could be fit into the data to which the model only then predicts estimated probabilities. The large amount of training and validation data does allow for newer models to developed meaning a lot of other surveys recorded by the GAMA catalogue can be accessed. These images can then be retrieved and with the use of reshaping and data augmentation in the 2nd prototype model, only a small script is needed if the images have no ID.

### Chapter 5.2– Future Work

In future work the focus on developing a model which achieves a higher accuracy can be based on a few sections. With the Hardware constraints on training and testing the model reaching the systems maximum performance other routes where considered. The most applicable to this model would be using tensor flow with keras to run the operations with this systems GPU instead of a CPU. Putting fewer computing operations on the CPU and using a GPU will read the images at a faster rate (Shaikh, 2017). An attempt was made to try and develop this for one of the prototypes due to time constraints and the process of installing specific library’s this alternative could not be chosen. Below in figure 45 shows the code for trying to detect a GPU on the system used to develop the python scripts.

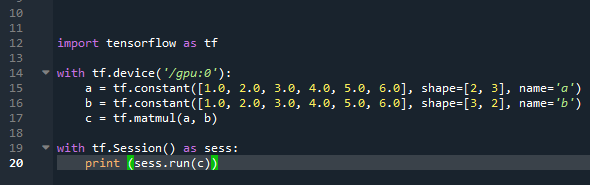


Figure - Checking tensor flow is working by looking for GPU

This would solve the hardware constraints and shorten the waiting time between training models. Another alternative found much later after the development stages was using a cloud supercomputer hosted by google to virtual operate the models through a superior system (Deep Learning VM documentation  |  Google Cloud, 2020). This direction would be the primary focus for developing any future work with the idea of using machine learning with CNNs.

## References

Nolte, A., Wang, L., Bilicki, M., Holwerda, B. and Biehl, M. (2019). Galaxy classification: A machine learning analysis of GAMA catalogue data. *Neurocomputing*, 342, pp.172-190.

MARTEL, H., PREMADI, P. and MATZNER, R. (1997). *ShieldSquare Captcha*. [online] Iopscience.iop.org. Available at: https://iopscience.iop.org/article/10.1086/305472/fulltext/35795.text.html

Herts.ac.uk. (2019). *The structure, formation and evolution of galaxies*. [online] Available at: https://www.herts.ac.uk/research/centres/car/extragalactic/the-formation-and-evolution-of-galaxies

Abraham, R. and van den Bergh, S. (2001). [online] Go-gale-com.edgehill.idm.oclc.org. Available at: https://go-gale-com.edgehill.idm.oclc.org/ps/i.do?p=AONE&u=edge&id=GALE|A77748767&v=2.1&it=r&sid=summon.

W. Hodge, P. (2019). *Galaxy - Irregular galaxies*. [online] Encyclopedia Britannica. Available at: https://www.britannica.com/science/galaxy/Irregular-galaxies.

Huertas-Company, M. (2018). *Deep Learning and Galaxy Classification*. [online] American Scientist. Available at: https://www.americanscientist.org/article/deep-learning-and-galaxy-classification.

Nasa.gov. (2002). *NASA - The Hubble Story*. [online] Available at: https://www.nasa.gov/mission\_pages/hubble/story/the\_story.html.

Sdss.org. (2019). *Telescopes and Instruments | SDSS*. [online] Available at: https://www.sdss.org/instruments/.

Royal Astronomical Society. (2013) Monthly Notices of the Royal Astronomical Society, Volume 435. [online] Available at: https://academic.oup.com/mnras/issue/435/4/.

Kyle W. Willett, Chris J. Lintott, Steven P. Bamford, Karen L. Masters, Brooke D. Simmons, Kevin R. V. Casteels, Edward M. Edmondson, Lucy F. Fortson, Sugata Kaviraj, William C. Keel, Thomas Melvin, Robert C. Nichol, M. Jordan Raddick, Kevin Schawinski, Robert J. Simpson, Ramin A. Skibba, Arfon M. Smith, Daniel Thomas, Galaxy Zoo 2: detailed morphological classifications for 304 122 galaxies from the Sloan Digital Sky Survey, Monthly Notices of the Royal Astronomical Society, Volume 435, Issue 4, 11 November 2013, Pages 2835–2860.

Sdss.org. 2014. *Sloan Digital Sky Surveys | SDSS*. [online] Available at: <https://www.sdss.org/surveys/>.

Gama-survey.org. 2020. *GAMA | Galaxy and Mass Assembly*. [online] Available at: <http://www.gama-survey.org/.

Nolte, A., Wang, L., Bilicki, M., Holwerda, B. and Biehl, M., 2020. *Galaxy Classification: A Machine Learning Analysis of GAMA Catalogue Data*. 1st ed. Poland: Leiden University, pp.1-22.

Kaggle.com. 2014. *Galaxy Zoo - The Galaxy Challenge | Kaggle*. [online] Available at: https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge.

NASA. 2014. *Hubble Space Telescope – Advanced Camera for Surveys*. [online] Available at: <https://www.nasa.gov/content/hubble-space-telescope-advanced-camera-for-surveys> [Accessed 4 May 2020].

2018. *Deep Galaxy: Classification of Galaxies Based on Deep Convolutional Neural Networks*. 1st ed. Cariro: Computer science Department Integrated Thebes Institutes Cairo, Egypt, pp.1-4.

E. A. Owens, R. E. Griffiths, K. U. Ratnatunga, Using oblique decision trees for the morphological classification of galaxies, Monthly Notices of the Royal Astronomical Society, Volume 281, Issue 1, July 1996, Pages 153–157

Darren S. Madgwick, Correlating galaxy morphologies and spectra in the 2dF Galaxy Redshift Survey, Monthly Notices of the Royal Astronomical Society, Volume 338, Issue 1, January 2003, Pages 197–207, https://doi.org/10.1046/j.1365-8711.2003.06033.x

Seif, G., 2018. *A Guide for Building Convolutional Neural Networks*. [online] Medium. Available at: <https://towardsdatascience.com/a-guide-for-building-convolutional-neural-networks-e4eefd17f4fd.

Eriksson, U., 2012. *Functional Requirements Vs Non-Functional Requirements*. [online] ReQtest. Available at: <https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/ .

Sherrell, L., 2013. *1-9*. [online] Evolutionary Prototyping. Available at: <https://link.springer.com/referenceworkentry/10.1007%2F978-1-4020-8265-8\_201039#howtocite.

Rawat, W. and Wang, Z., 2017. *Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review*. 1st ed. Massachusetts: Massachusetts Institute of Technology, pp.1-288.

Wang, X. and Yang, H., 2016. *Cascade Classifier for Face Detection*. 2nd ed. Xidian: uachun Yang, School of Life Science and Technology, Xidian University, Xi’an 710086, China. Email: 64858034@qq.com, pp.187-197.

Berger, W., 2014. *Deep Learning Haar Cascade Explained - Will Berger*. [online] Will Berger. Available at: <http://www.willberger.org/cascade-haar-explained/> [Accessed 6 May 2020].

Gou, C., Zhang, H., Wang, K., Wang, F. and Ji, Q., 2019. *Cascade Learning from Adversarial Synthetic Images for Accurate Pupil Detection*. 1st ed. Beijing: Institute of Automation, Chinese Academy of Sciences, Beijing 100190, China, pp.584-594.

C. Useche Murillo, P., Jiménez Moreno, R. and O. Pinzón Arenas, J., 2017. *Comparison Between CNN And Haar Classifiers for Surgical Instrumentation Classification*. 1st ed. Bogotá: Nueva Granada Military University Bogotá, Colombia, pp.1-5.

MachineCurve. 2019. *Using Leaky ReLU With Kera’s – Machinecurve*. [online] Available at: <https://www.machinecurve.com/index.php/2019/11/12/using-leaky-relu-with-keras/#.

Cs231n.github.io. 2020. *Cs231n Convolutional Neural Networks for Visual Recognition*. [online] Available at: <https://cs231n.github.io/.

Tch, A., 2017. *The Mostly Complete Chart of Neural Networks, Explained*. [online] Medium. Available at: <https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464> [Accessed 11 May 2020].

Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X. and Metaxas, D., 2017. *Stackgan: Text to Photo-Realistic Image Synthesis with Stacked Generative Adversarial Networks*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1612.03242.

Zhu, J., Park, T., Isola, P. and A. Efros, A., 2018. *Unpaired Image-To-Image Translation using Cycle-Consistent Adversarial Networks*. [online] Arxiv.org. Available at: <https://arxiv.org/pdf/1703.10593.pdf.

J, V., 2018. *Tutorial on Kera’s Flow\_From\_Dataframe*. [online] Medium. Available at: <https://medium.com/@vijayabhaskar96/tutorial-on-keras-flow-from-dataframe-1fd4493d237c.

Rosebrock, A., 2018. *Keras Conv2d And Convolutional Layers - Pyimagesearch*. [online] PyImageSearch. Available at: <https://www.pyimagesearch.com/2018/12/31/keras-conv2d-and-convolutional-layers/.

Saama.com. 2017. *Different Kinds of Convolutional Filters*. [online] Available at: <https://www.saama.com/different-kinds-convolutional-filters/.

Brownlee, J., 2018. *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/.

Lco.global.2020. Galaxy Classification | las Cumbres Observatory. [online] Available at: http: https://lco.global/spacebook/galaxies/galaxy-classification/.

MachineCurve. 2020. *How To Use H5py And Keras To Train With Data From HDF5 Files? – Machinecurve*. [online] Available at: <https://www.machinecurve.com/index.php/2020/04/13/how-to-use-h5py-and-keras-to-train-with-data-from-hdf5-files/#.

Greshko, M., 2019. *Galaxy Information And Facts*. [online] Nationalgeographic.com. Available at: <https://www.nationalgeographic.com/science/space/universe/galaxies/.

Shaikh, F., 2017. *Why Are Gpus Necessary For Training Deep Learning Models?*. [online] Analytics Vidhya. Available at: <https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/.

Google Cloud. 2020. *Deep Learning VM Documentation  |  Google Cloud*. [online] Available at: <https://cloud.google.com/ai-platform/deep-learning-vm/docs.

## Appendices

A screenshot of a social media post

Description automatically generated

Figure - Pedestrian Detection Code

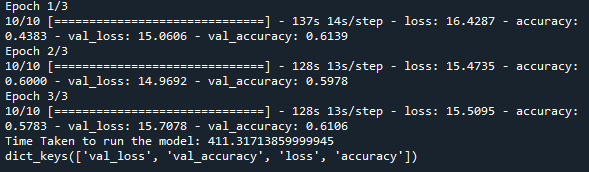


Figure - classifier history from the first iteration of testing

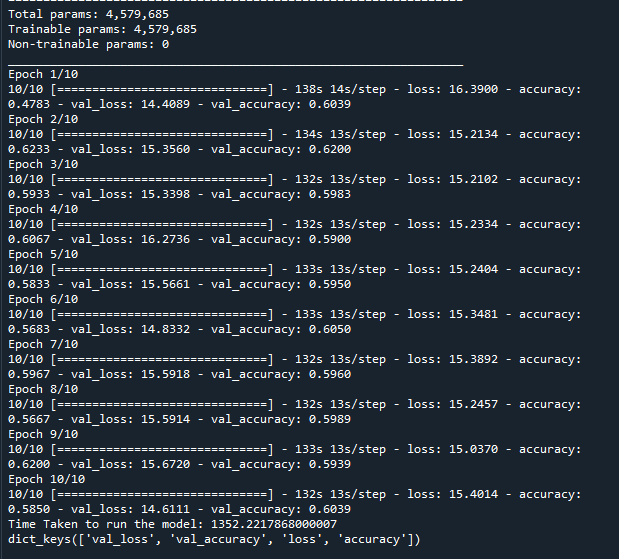


Figure - classifier history from the 2nd iteration of testing with the first prototype

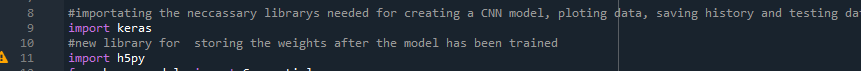


Figure - added library

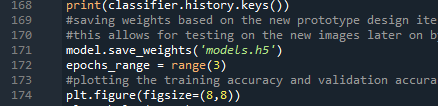


Figure - weights saved

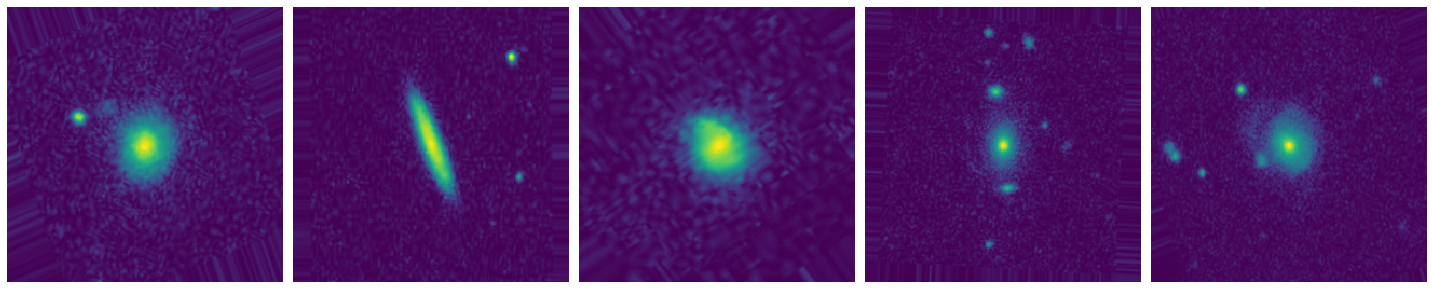


Figure - the images produced with figure 30 code

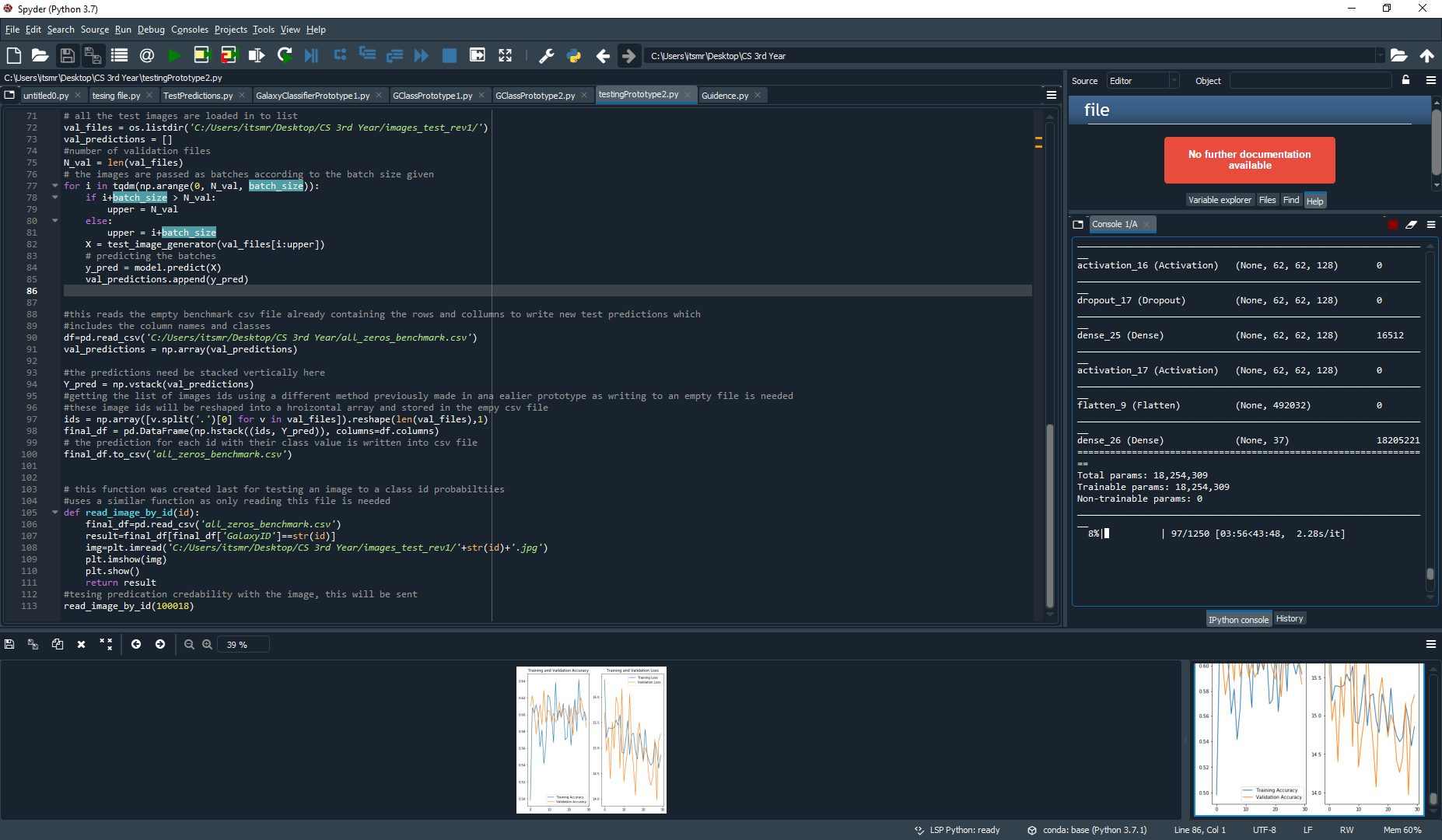


Figure - Use of new functionality

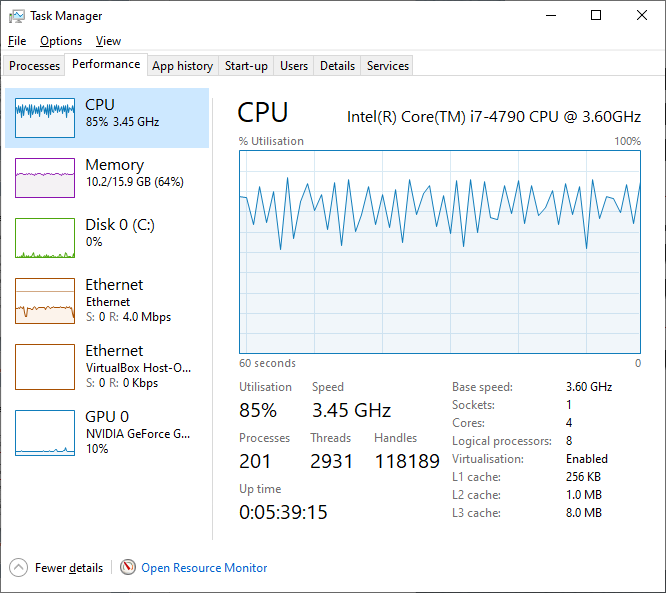


Figure - Hardware Performance Limit

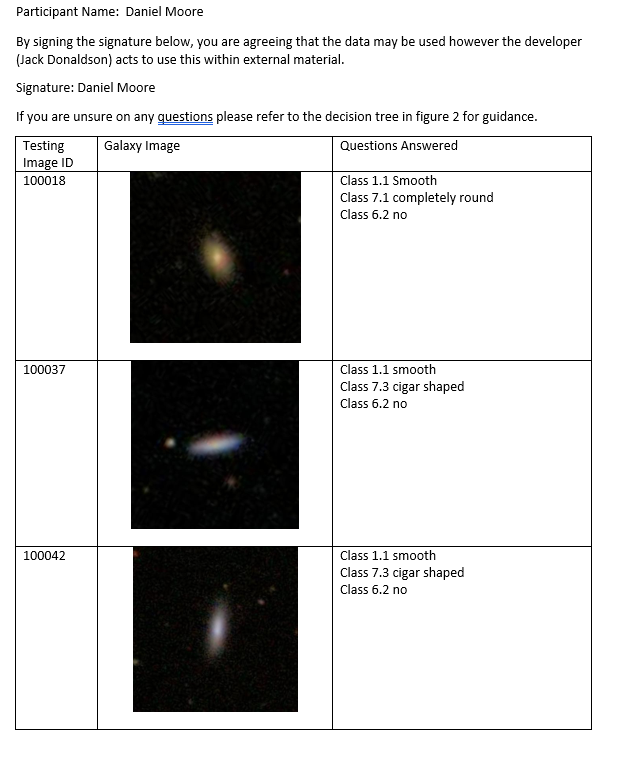


Figure - Survey Example

### Appendix A: setting up Model

Setting up a model requires a few steps to first run the python scripts. First the datasets used where downloaded from

<https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/data>

Required datafiles:

* Images\_training
* Solutions\_training
* Images\_test
* All\_zeros\_benchmark

After downloading the 4 datafiles simply place in a directory that can be accessed and retrieved from inside the python scripts.

After Downloading the ZIP file containing the artefact 4 files should be present,

* GClassPrototype1.py
* GClassPrototype2.py
* testingPrototype2.py
* models.h5

This is run on python 3.7 and windows 10 operating system. Other python versions may contain errors. At the top of each file the necessary library’s need to be imported if not already done so. The three python scripts operate as follows:

* GClassPrototype.py simply runs the model by training the Images\_training and Solutions\_training. Replace the directory of the downloaded data files into the same directory’s present in the python script.
* GClassPrototype2.py must be run to train the models training and validation weights the H5PY file.
* After these have been trained the testingPrototype2.py script can be run on testing data. Make sure to change the test directory to Images\_test directory.
* Make sure to All\_zeros\_benchmark directory is correct.

If any further testing is needed for the testingPrototype.py make sure to download another empty version to have separate test runs between data files.