Classification of Deep Space Objects using Deep Learning Techniques

Configuration Manual
MSc in Data Analytics

Cillín Ó Foghlú
Student ID: 18186751

School of Computing

National College of Ireland

Supervisor: Dr Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Cillín Ó Foghlú		
Student ID:	18186751		
Programme:	MSc in Data Analytics	**Year:**	2020
Module:	Research Project		
Supervisor:	Dr Catherine Mulwa		
Submission Due Date:	August 2020		
Project Title:	Classification of Deep Space Objects using Deep Learning Techniques		
Word Count:	……9063…… **Page Count**……56……..……..		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ……Cillín Ó Foghlú……
Date: ……16/08/2020……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Cillín Ó Foghlú
Student ID: 18186751

1 Introduction

This report forms part of the submission by student Cillín Ó Foghlú as the configuration manual which describes how to implement that Classification of Deep Space Objects using Deep Learning Techniques submission for M. Sc in Data Analytics.
Section 2 covers additional details on the research project which complement the Research Report paper submitted. Section 3 details the hardware used while section 4 describes how to install the relevant software. Section 5 details the process to select and download the images used. Section 6 describes the python code used to acquire and pro-process the SDSS images. Section 7 covers the extraction of images form STScI. Section 8 describes the models training and section 9 covers the results from the training.
.

2 Supplementary Details from Technical Report

The following are additional details which were identified or investigated as part of the research project, however due to constraints in the documentation, were not inserted into the final report, however are detailed below for information and supplementary details in support of the research.

2.1 Research Methodology

TensorFlow is a machine learning system that operates at large scale and in heterogeneous environments. TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. It maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, general purpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). This architecture gives flexibility to the application developer: whereas in previous "parameter server" designs the management of shared state is built into the system. TensorFlow enables developers to experiment with novel optimizations and training algorithms. TensorFlow supports a variety of applications, with a focus on training and inference on deep neural networks.
Several of Googles' services use TensorFlow in production, and it has been released as an open-source project and it has become widely used for machine learning research. In this paper, the TensorFlow dataflow model is described and demonstrated. The compelling performance that TensorFlow achieves for several real-world applications. (Abadi et al., n.d.) (Chapman, et al., 2000) was identified as the best suited to this research problem

2.2 Learning Approach

As all the outcomes for the classifications were known in advance, the data mining fell into the class of supervised training methodology (Post Grad Programme in Data Analytics, 2019). The images were presented to the ANN using TensorFlow objects and the Keras method. This is a computer vision and image classification method which is widely regarded as industry standard currently. A CNN is a class of deep learning neural networks which are commonly used for image analysis as they do not require complex methods such as "*momentum, weight decay, structure- dependent learning rates, averaging layers, tangent prop, or even finely-tuning the architecture*" (Simard, et al., 2013)

CNN's layers are randomly seeded with values initially and these values get modified by the model as it "leans" through the process of back-propagation as it computes the loss function

and gradient (Goodfellow, et al., 2016). Features are not defined initially, they are "learnt", and the network used adjusts these values based on the outcomes of each image used and the known outcome.

2.3 Convolutional Neural Networks

An artificial neural network is a computer system whose design is based on the structure of biological neural networks and is designed to perform "human like" tasks, such as image or handwriting recognition. The concept dates back to early 1960's when (Hubel & Wiesel, 1962) introduced what many considered as the first convolutional neural network. While is lacked the advancements od recent years, such as back-propagation which was proposed (LeCun, et al., 1989) in applying recognition to handwritten zip codes. Back propagation is used to compute the gradient of the loss function so that the model can adjust weights and biases.

2.4 Design for Data Acquisition

Once the URL's were known, then a locally executed "wget" command was called using an input text file with just the URL's as a parameter. This triggered the download of the associated fits files to the local hard drive. It is important to restate that the goal of this project was to validate the ability of local processing to complete the work as much as possible to allow home users support the professional community.

FITS files were downloaded from https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/
SDSS JPEG files were available at: http://skyserver.sdss.org/dr16/SkyServerWS/ImgCutout/
And STScI jpeg files at: http://skyserver.sdss.org/dr16/SkyServerWS/ImgCutout/getjpeg? followed by the ra, dec, and size of the required images.

2.5 Design for Processing and Modelling of Data

Once the data was retrieved in fits format, it was then processed to reduce both size and to convert to an image format for later processing. This processing along with the remainder of the model design, training and testing was completed locally. FITS files were downloaded and saved into sub-directories, one per class. Then the images were extracted into a plot from each FITS image and saving out as a PNG before closing the image and moving on to the next image in the directory. This process was coded in Python and full details on the hardware and software used are in the Configuration Manual submitted with this report.
For SDSS JPEG images this was again using a 'wget' command and the files were also saved into subdirectories, one per class. Python was used to rename all images to *.jpg to allow for future processing as images input to the models. For STScI jpeg images, the same CAS results were used and 'wget' commands used to download these images to subdirectories, one per class.

2.6 Data Selection and extraction

The data source in this case is the Slone Deep Space Survey archive servers. The data used is held on the dr12.sdss.org servers. In this case dr12 related to the 12th data release of data from the survey's results and is described, by the SDSS press release as : " *Data Release 12 (DR12) is the final data release of the SDSS-III, containing all SDSS observations through July 2014. It includes the complete dataset of the BOSS and APOGEE surveys, and also newly includes stellar radial velocity measurements from MARVELS"*. All data has already been pre-processed and digital noise or "bad" images removed. Images impacted by poor

weather conditions or unfavourable atmospherically conditions were also removed prior to the data release.

2.7 Data Transformation – possible cut and move to config

Generally, data transformation refers, with the computing sphere, process of converting data from one format to another. In this case there are multiple steps in the process, and these are detailed below.

2.7.1 Extracting Images from FITS format to PNG

Using Python, the fits files were opened, and the images plotted before being saved out as "png" files. Astronomical data is generally held in fits files which contain more data than was necessary for this CNN – the fits file sizes made utilisation of them too cumbersome and memory / CPU intensive. This reduces the size of the files from between 2 and 4 MB to approximately 120k. It also reduced the image size from 1409x2048 to 576x432. The file names were used minus the ".fits.bz2" as the new names for the images. A leading character was added, "s" for Stat files, "g" for Galaxy files and "q" for quasar files. he images files were also written to individual folders, based on their classification, for future processing. *Figure 1 – Sample images from SDSS across all filters post stage 2 processing* shows a sample of the images extracted from the "fits" files. Local processing allowed for approx. 40 fits files to be converted to smaller ".png" files at the rate of 40 per minute.
Further transformation was carried out by using Python to combine the visible light filters into a 2 channel RGB format and saving the files as colour files, see Figure 1. This was also used as training input to the CNN and the results are discussed in Section 8.4

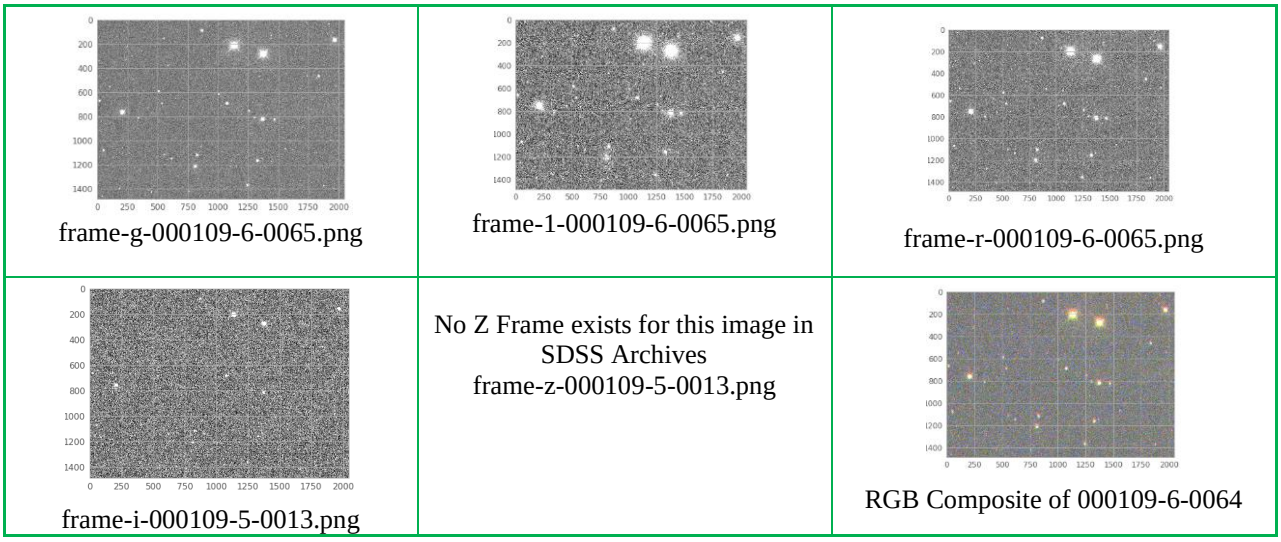
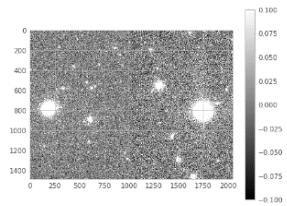
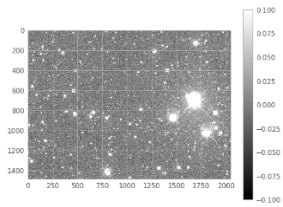


Figure 1 – Sample images from SDSS across all filters post stage 2 processing

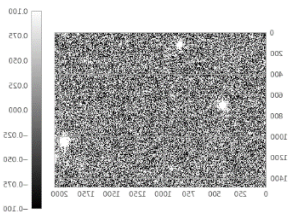
This process had the double advantage of both reducing the size of files, in the example of start fits from 150GB per filter to under 10GB and also only having images to be processed, all superfluous data was not brough forward into the next stage of the process – the neural network. It was noted that images, examples in Figure 2, had more than just one object in them and that this was a potential issue for later processing.



qframe-i-002126-1-0263



sframe-g-000307-3-0162



gframe-z-002964-4-0243

Figure 2 - Samples of all categories with more miltiple objects in a single image

For images extracted from STScI another Python script was used, a modification to the script published in the STScI's GitHub pages. The modification took an input text file in 3 columns and generated a call to the STScI's servers, and the returned jpeg was saved to a working directory locally. A sample of the imagery from STSci is in Figure 3

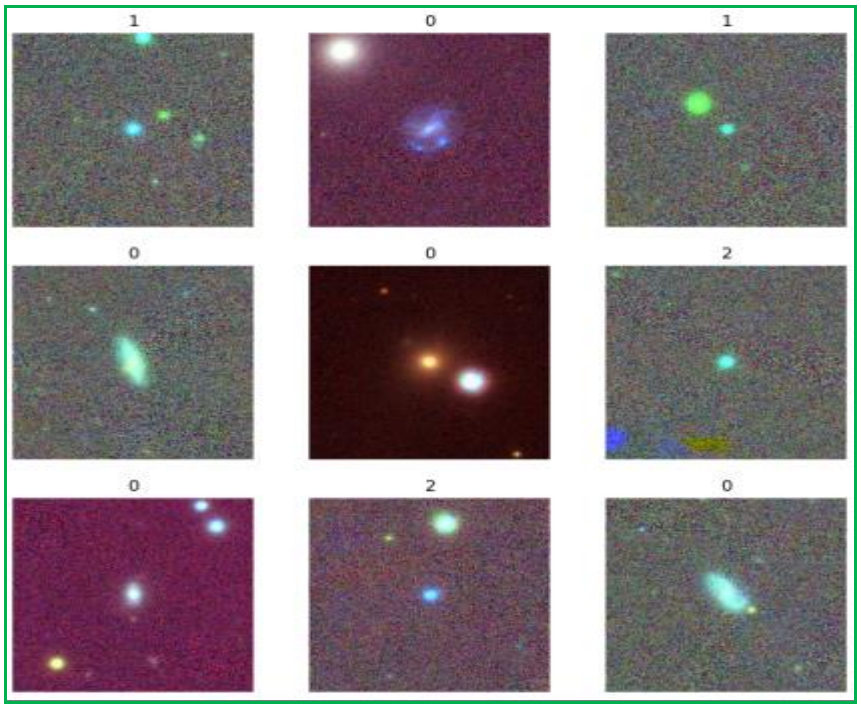


Figure 3 : Sample of 9 jpg Images from STScI with category indices

2.8 CNN Model

CNN models are made up of input layers, a number of hidden layers and an output layer. The number of hidden layers used is not an exact science, research conducted by (Ma, et al., 2014) on using hidden layers in language processing tended for a trial and error approach, that there was no fixed rules in selecting the correct number of layers at each stage within a deep learning model. Other reasons to limit the depth of the ANN was degradation and vanishing/exploding gradient problems. These factors were factored into the design of the modifications to the models used.

2.8.1 Activation Functions within TensorFlow

Deep Learning models make use of several different types of activation functions, optimisation functions as well as metrics to evaluate the output. Tensor has many options for each, and the ones used in this project are listed below along with the rational for their use.

2.8.2 Adam Optimiser

The Adaptive moment estimation or Adam optimisation is a stochastic gradient descent method that is based the Root Mean Square Propagation (RMSprop) optimiser and momentum. Momentum takes past gradients as input to smooth the gradient steps in the model. According to (Diederik P. Kingma, 2015) the method is *"computationally efficient, has little memory requirement, invariant to diagonal rescaling of gradients, and is well suited for problems that are large in terms of data/parameters"* It offers a straightforward implementation which is computationally efficient and has little memory requirements. It is well suited for problems with large data or parameters. In choosing which optimizer to select out of the various options, research by (Ruder, 2016) and (Kingma & Lei Ba, 2015) guided the selection. Figure 4 : Comparison of Adam to other Optimization Algorithms taken from (Ruder, 2016) report titles "An overview of gradient descent optimization algorithms"

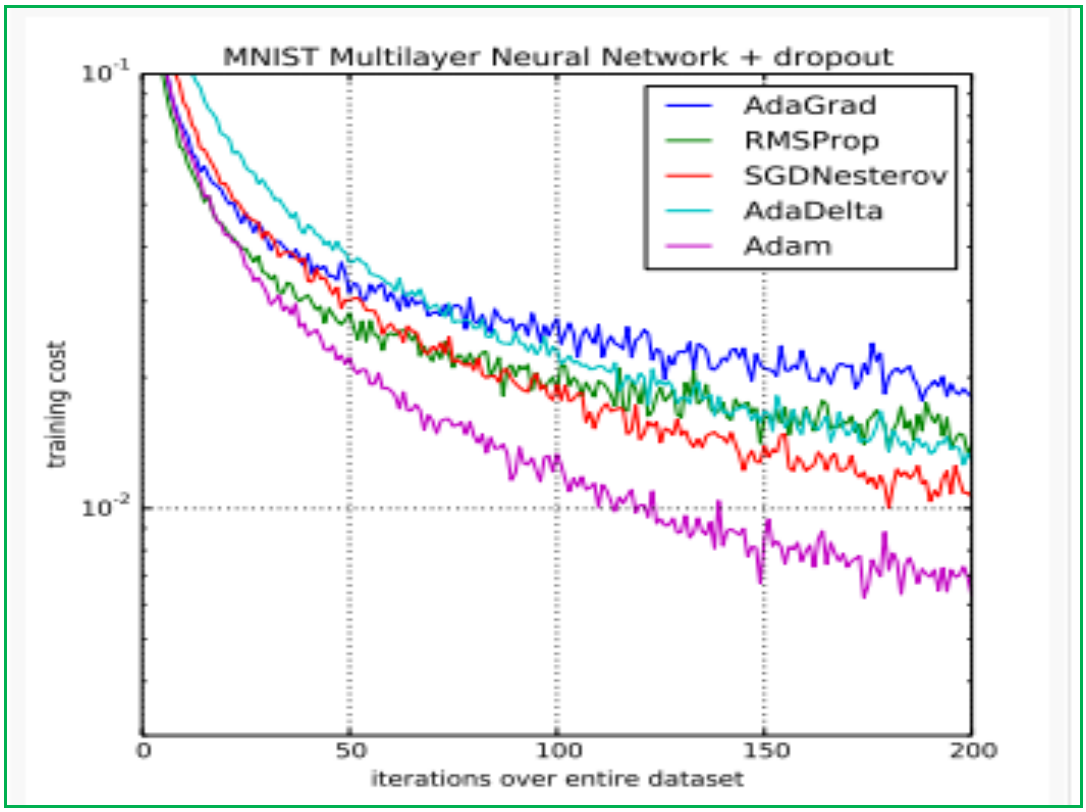


Figure 4 : Comparison of Adam to other Optimization Algorithms

2.8.2.1Relu Activation

The Rectified Linear Unit (ReLU) function, Figure **5**, within CNN's, is used to increase the non-linearity in the images presented to the CNN. Its purpose is to increase the non-linearity of the images. It is used to transform the summed weighted input from a node into the activation of the nodes output for that input (Brownlee, 2019) Based on researching a number of different CNN's it was noted that this was the de facto standard used in many

NN's as Brownlee said "*because a model that uses it is easier to train and often achieves better performance".*

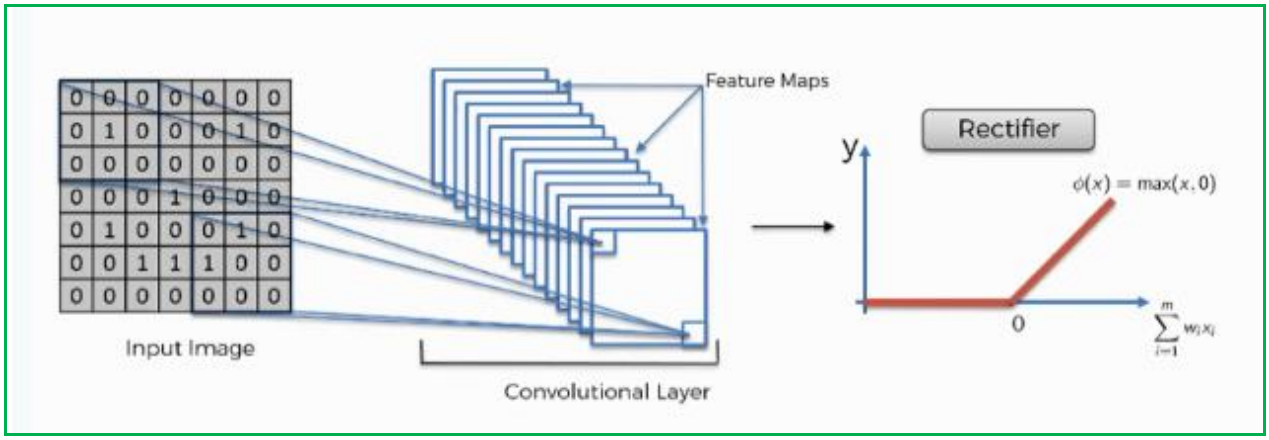


Figure 5 : ReLU Activation Function

2.8.2.2Softmax () Activation

Softmax() is a function, Equation 1, used to set the outcome into a set of probabilities. In this case the outcomes were a probability of an image being of a star, galaxy or a quasar. Using softmax() the outcome with the highest probability became the category of the image. (Wikepedia, n.d.) describes s*oftmax() as a "generalization of the logistic function that "squashes" a K-dimensional vector of arbitrary real values to a K-dimensional vector of real values in the range [0, 1] that add up to 1.*

$$\sigma(x_j) = e^{x_j} / (\sum (i=1 \text{ to } n)\, e^{x_i}) \text{ (for } j=1 \text{ to } n)$$

Equation 1 - Softmax Function

2.9 The ResNet50

The degradation problem was overcome by the introduction of residual networks. ResNet50 stacks residual blocks stacked into 50 layers. A sample of this design, Figure 6 is in below.

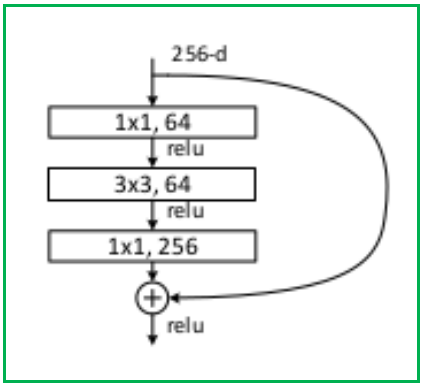


Figure 6 : Basic block in ResNet 50

3 Hardware Configuration

PC:	Custom Build PC
Processor	Intel i7-3770 @3.40 GHz 8 cores
RAM	24Gb DDR3
OS	Windows 10 Pro 64 bit edition
Graphics Card	NVIDIA GForce GT640 2Gb DDR3 128Bit Bus This was replaced with NVIDIA GForce GTX1660 Ti ,6GB DDR6

4 Software Used and Installation Process

This section lists all installed applications required to run the project along with the detailed installation process and screen shots.

4.1 Microsoft Applications

No installation instructions are provided for the standard Microsoft Office and Operating system used in this research. Any additional components are listed which were used.

Microsoft Word
    Add-In for Zotero
Microsoft Excel
Microsoft PowerPoint
Microsoft Snipping Tool

4.2 Anaconda 3 IDE & Spyder Python

Both applications are contained in a single install package which is detailed below

1) Download from https://www.anaconda.com/distribution/

2) Chose 64-bit Graphical Installer and save file as per Figure **7**

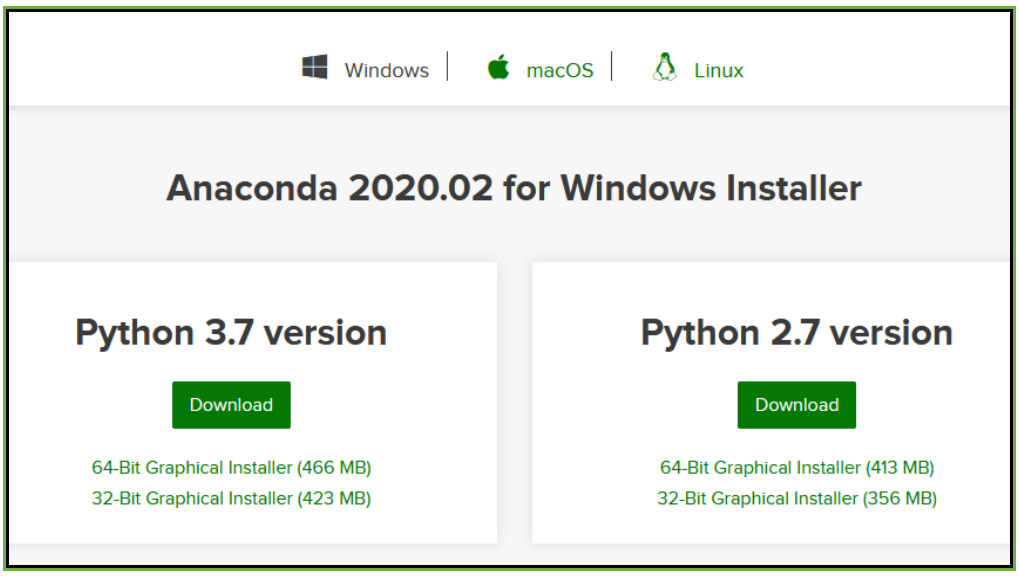


Figure 7 : Chose version of Python

3) Open Installation package from downloaded directory as per Figure **8**

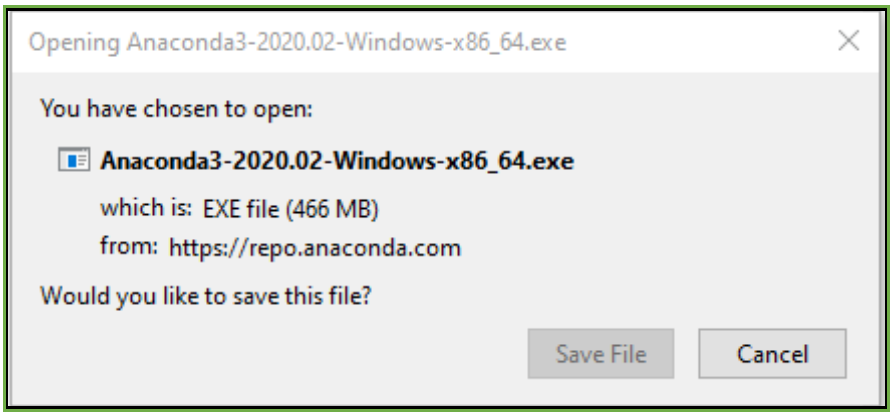


Figure 8 : Select where to install Python

4) Select Next to start the installation on

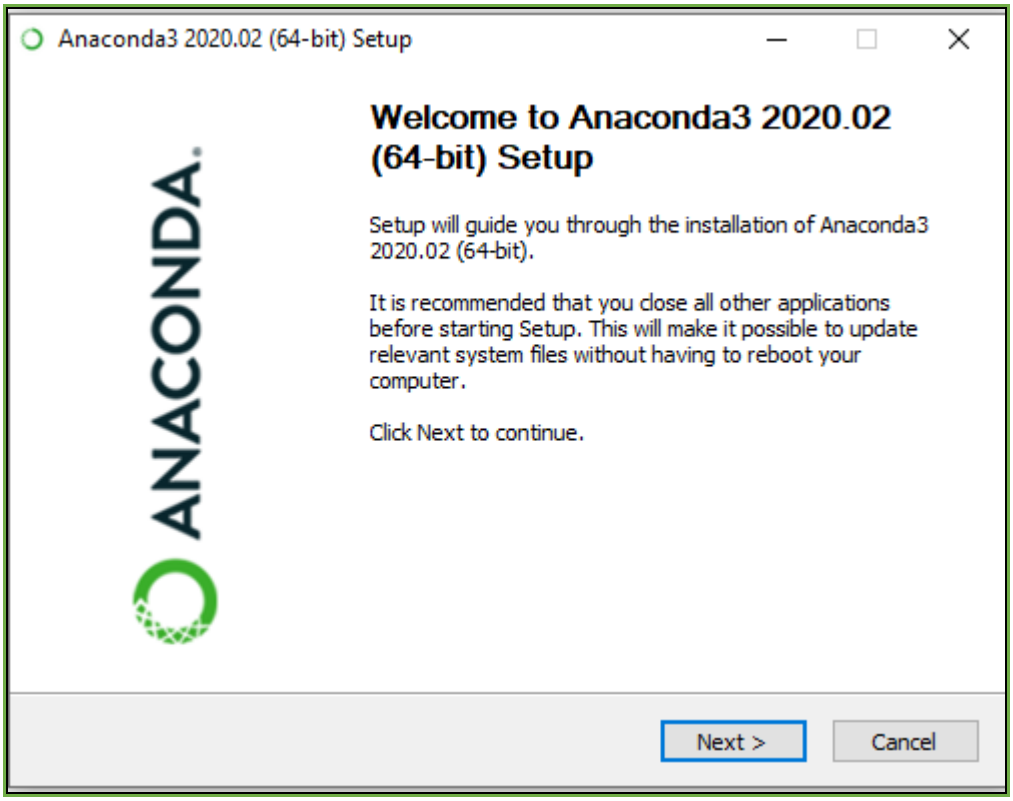


Figure 9 : Anaconda Installation Welcome Screen

5) Accept the Licencing Agreement and select **Next** as in Figure 10

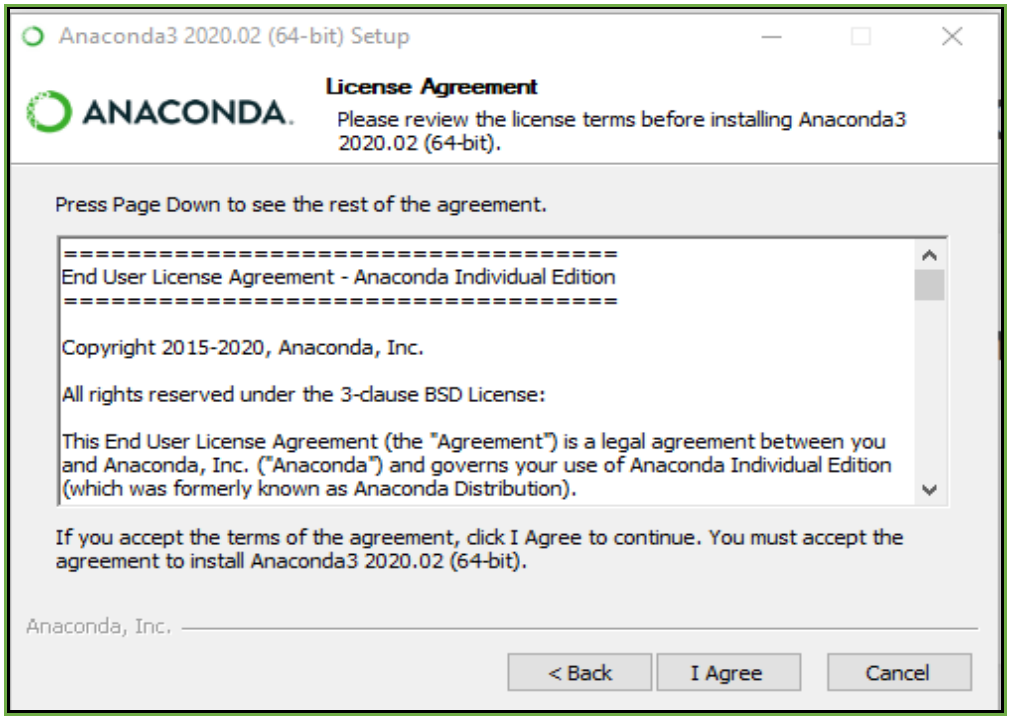


Figure 10 : Anaconda Licence Screen

6) Chose Installation Type - in this case the recommended option was chosen and select **Next** as in Figure 11

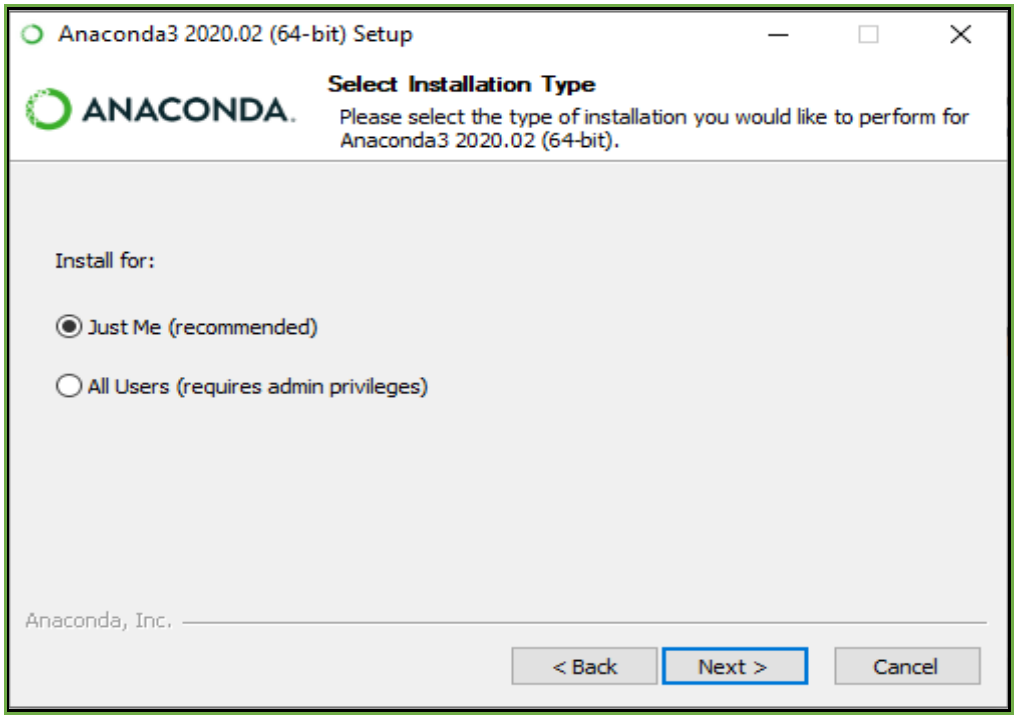


Figure 11 : Anaconda Select Installation Type

7) Chose Installation Directory and select **Next** as described in Figure 12

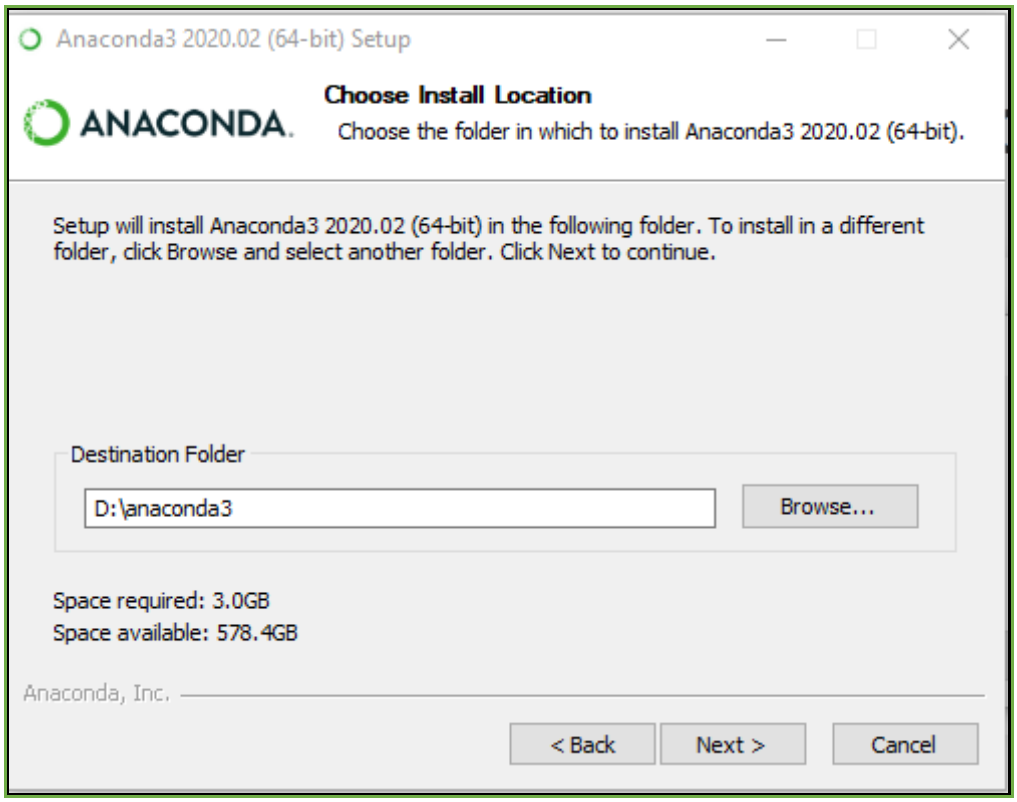


Figure 12 : Anaconda Install Folder

8) Register Anaconda Python as default Python installation and select Next to include Python in the Windows Path as shown in Figure 13

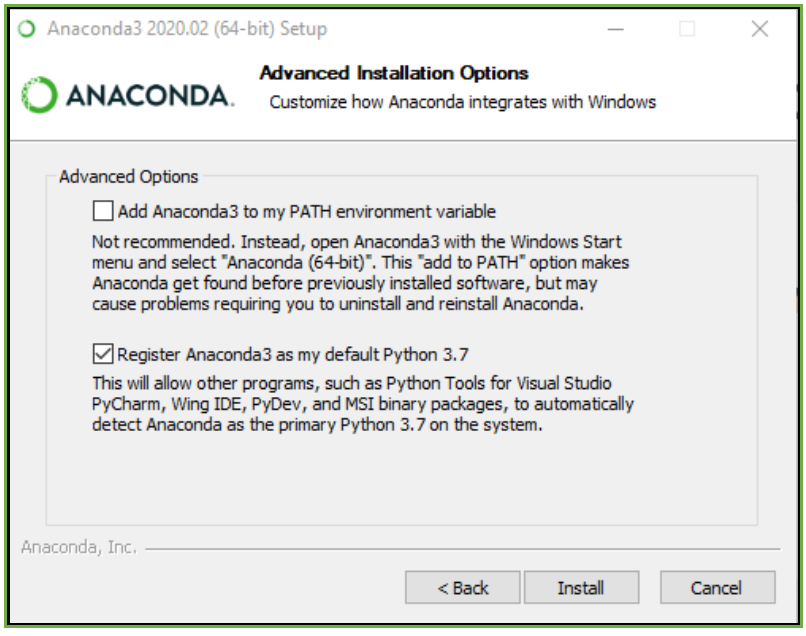


Figure 13 - Add Python to PATH in Windows

9) The installation process will commence as per Figure 14

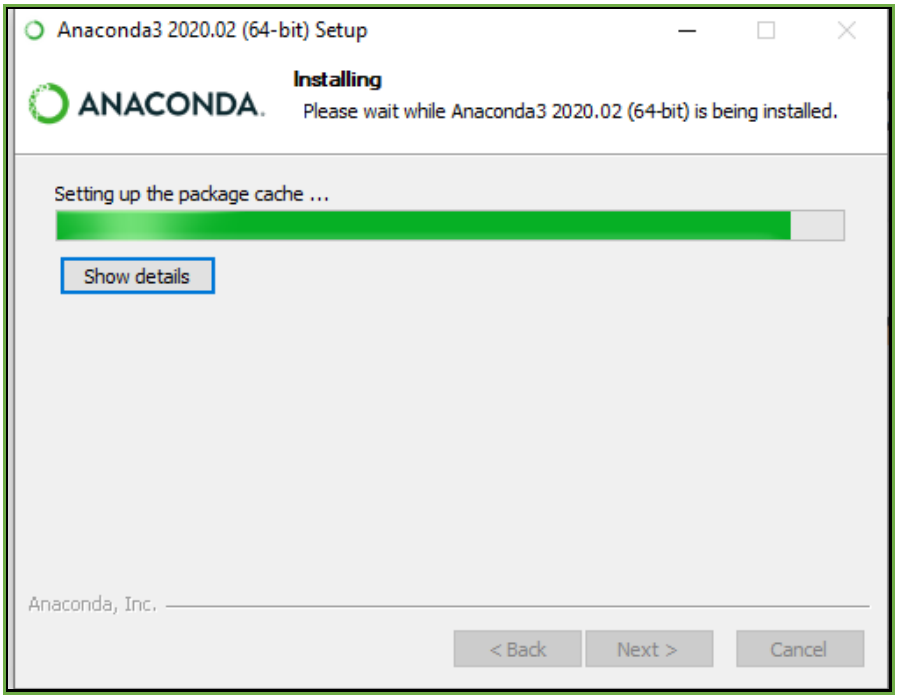


Figure 14 - Python starting installation process

10) Installation Completed. Select **Next** to complete as per **Figure 15**

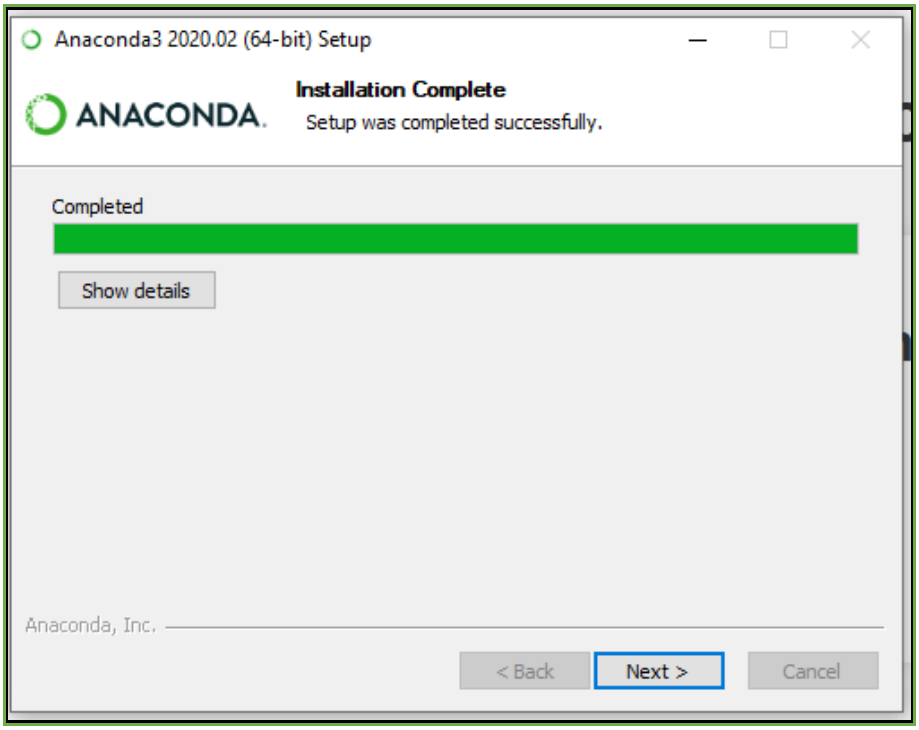


Figure 15 - Completion of Installation Notification

11) Select Next to complete the installation process and see available support site as per Figure 16

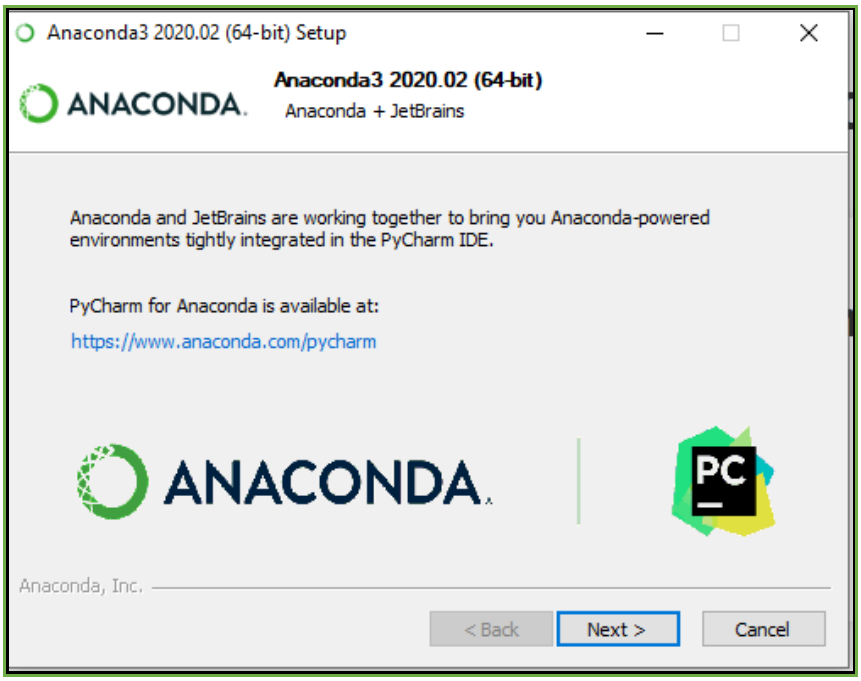


Figure 16 - Link to PyCharm site

12) And finally click the Finish button as per Figure 17

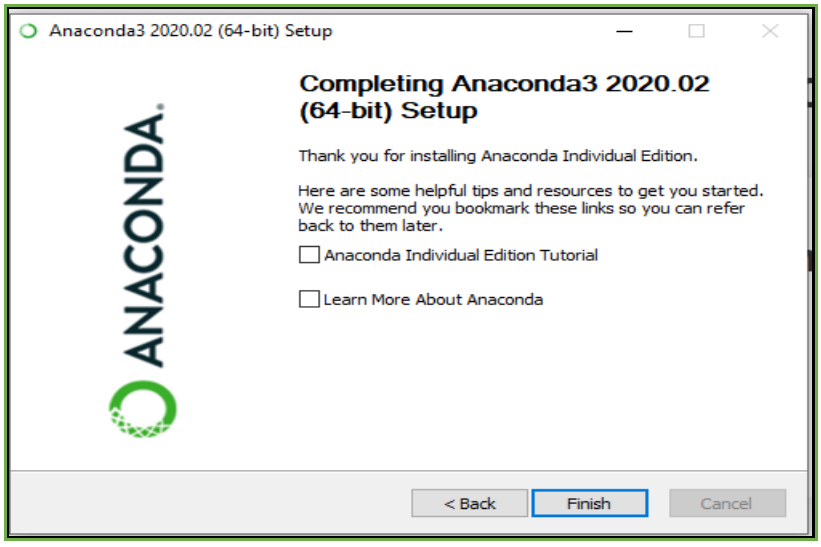


Figure 17 - Final Installation Notification

13) Optional installation of Kite was chosen. Kite is an autocompletion tool within the Anaconda framework which helps by suggesting remaining test to be written – Figure 18

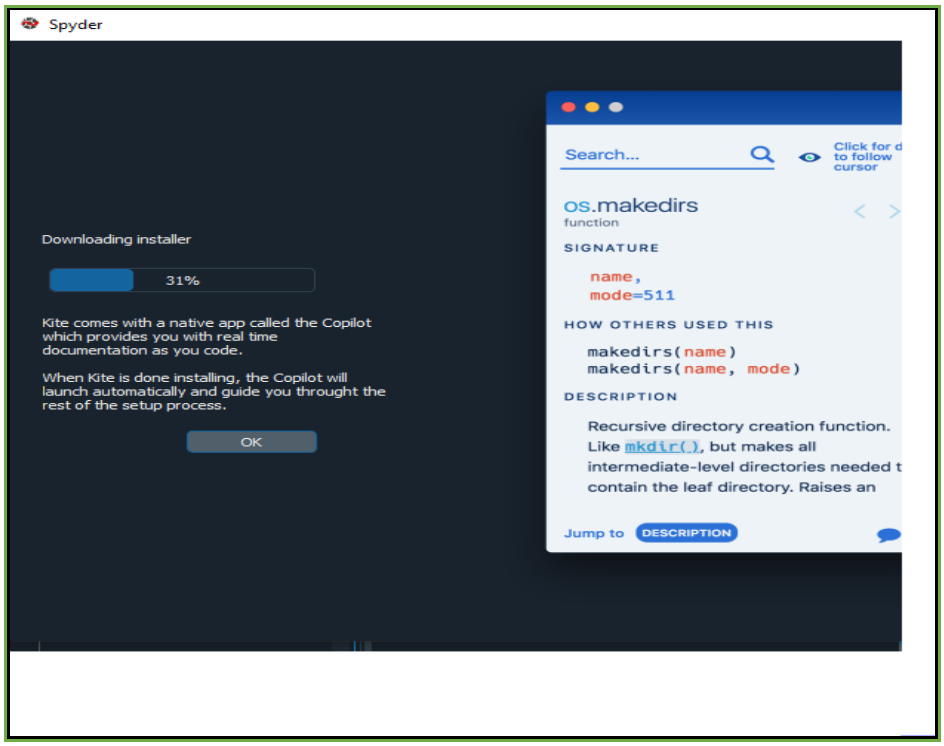


Figure 18 - Add Kite to installation

14) Spyder Python comes as part of this installation and the version being used is 4.0.1 see Figure 19

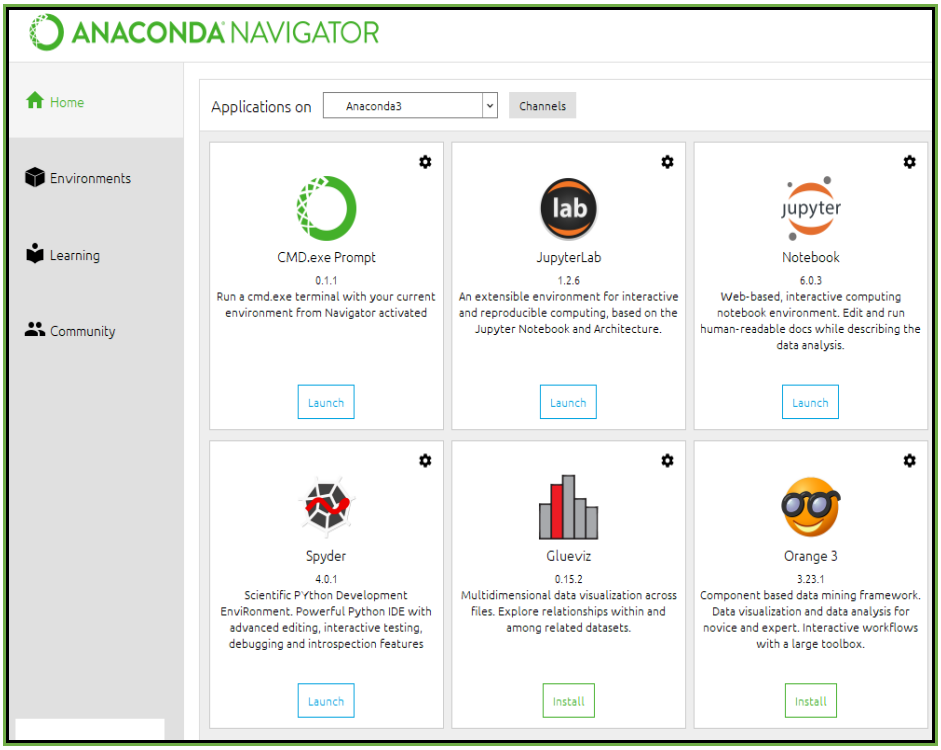


Figure 19 - Anaconda Package Installer

4.3 Install “wget” for windows

The required file can be downloaded from:

https://medium.com/@medasuryatej/install-tensorflow-gpu-2-0-f4e215438199

Installation only required a copy of the downloaded file to the required directory. This file was used to download the image files from the SDSS archive servers.

4.4 CUDA 9.0

In order to utilise the GPU, Nividai CUAD was required and is available at https://developer.nvidia.com/cuda-90-downloadarchive?target_os=Windows&target_arch=x86_64

Select the appropriate version for your computer as be Figure 20

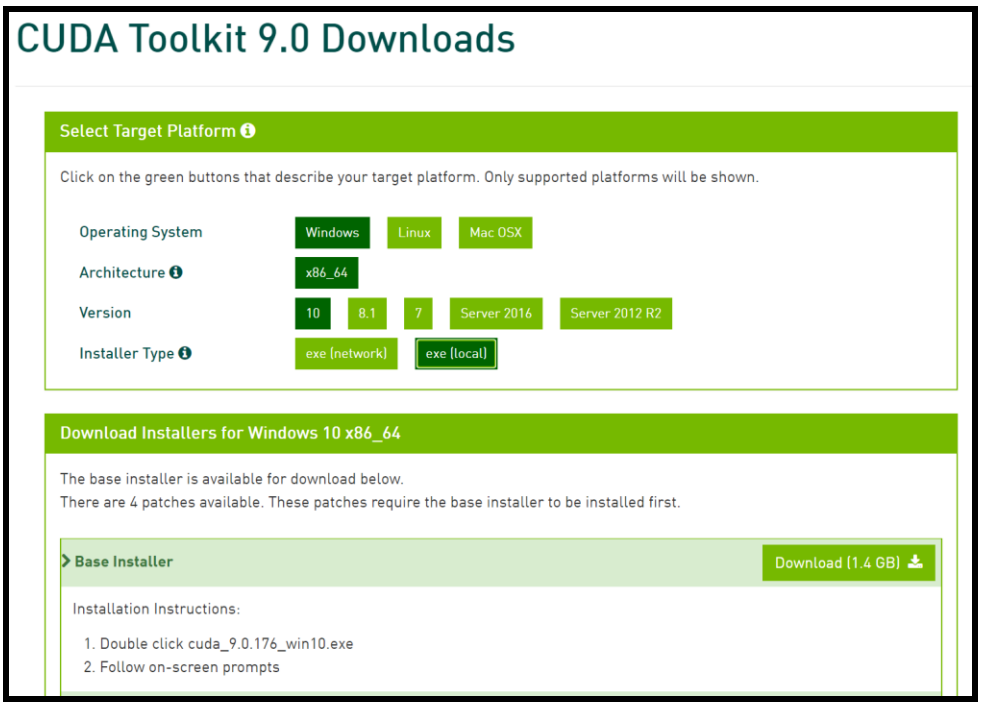


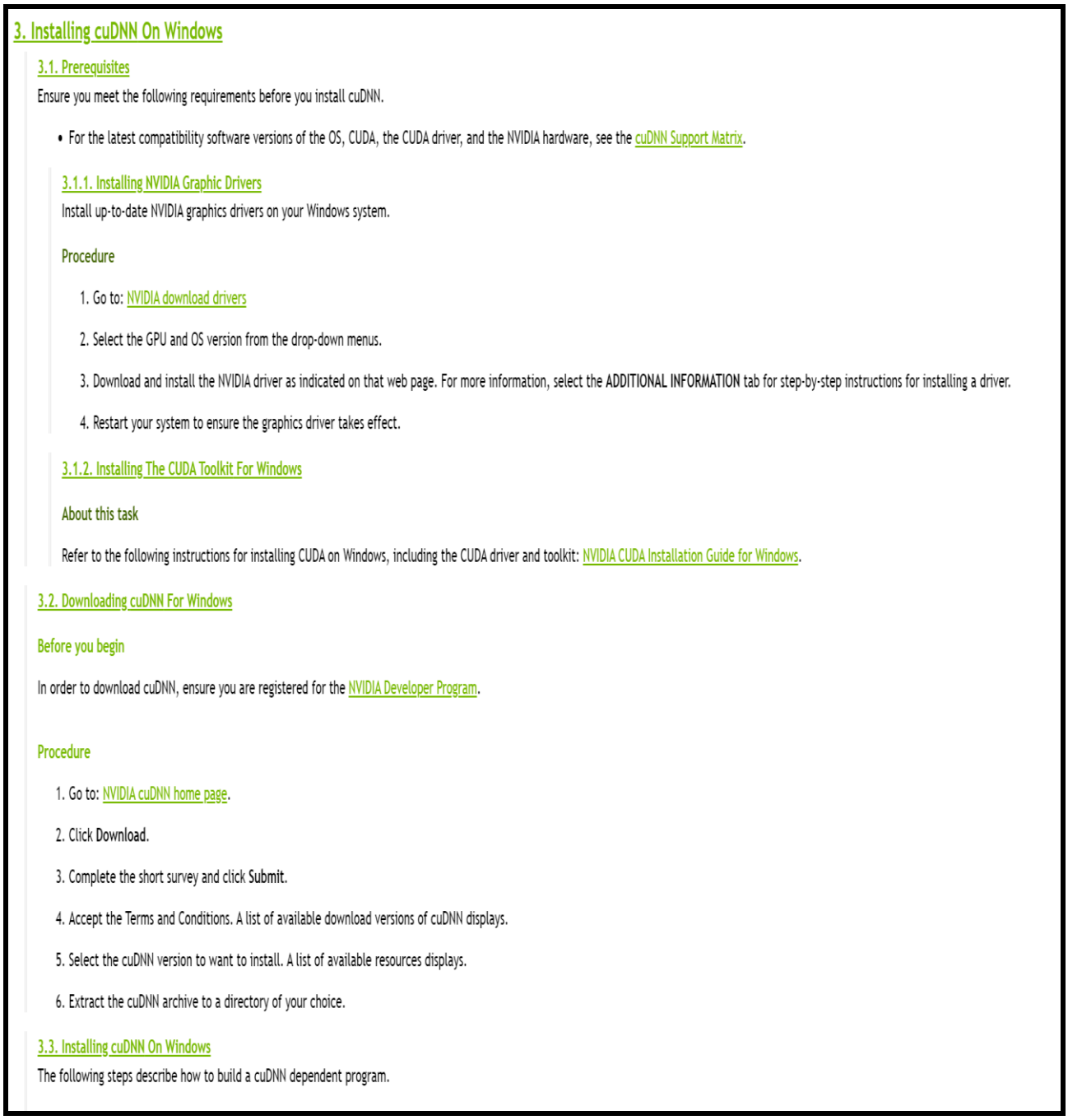
Figure 20 : Download CUDA Toolkit

Once downloaded this package was installed. By clicking the package as downloaded. No configuration is required.

4.5 CUDNN v7.6.5

The Nividia support toolkit for Neural Networks is available at: https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html

there is an ethics agreement required when logging in first time. This is to ensure that the toolkit is not used for rational or genttic profiling and in accordance with Nidivia’s ethical statement. You must agree to this before downloading the package. Once downloaded the files are extreacted from a zip folder and the pocess to install is detailed in Figure 21 and Figure 22



3. Installing cuDNN On Windows

3.1. Prerequisites

Ensure you meet the following requirements before you install cuDNN.

- For the latest compatibility software versions of the OS, CUDA, the CUDA driver, and the NVIDIA hardware, see the cuDNN Support Matrix.

3.1.1. Installing NVIDIA Graphic Drivers

Install up-to-date NVIDIA graphics drivers on your Windows system.

Procedure

1. Go to: NVIDIA download drivers
2. Select the GPU and OS version from the drop-down menus.
3. Download and install the NVIDIA driver as indicated on that web page. For more information, select the **ADDITIONAL INFORMATION** tab for step-by-step instructions for installing a driver.
4. Restart your system to ensure the graphics driver takes effect.

3.1.2. Installing The CUDA Toolkit For Windows

About this task

Refer to the following instructions for installing CUDA on Windows, including the CUDA driver and toolkit: NVIDIA CUDA Installation Guide for Windows.

3.2. Downloading cuDNN For Windows

Before you begin

In order to download cuDNN, ensure you are registered for the NVIDIA Developer Program.

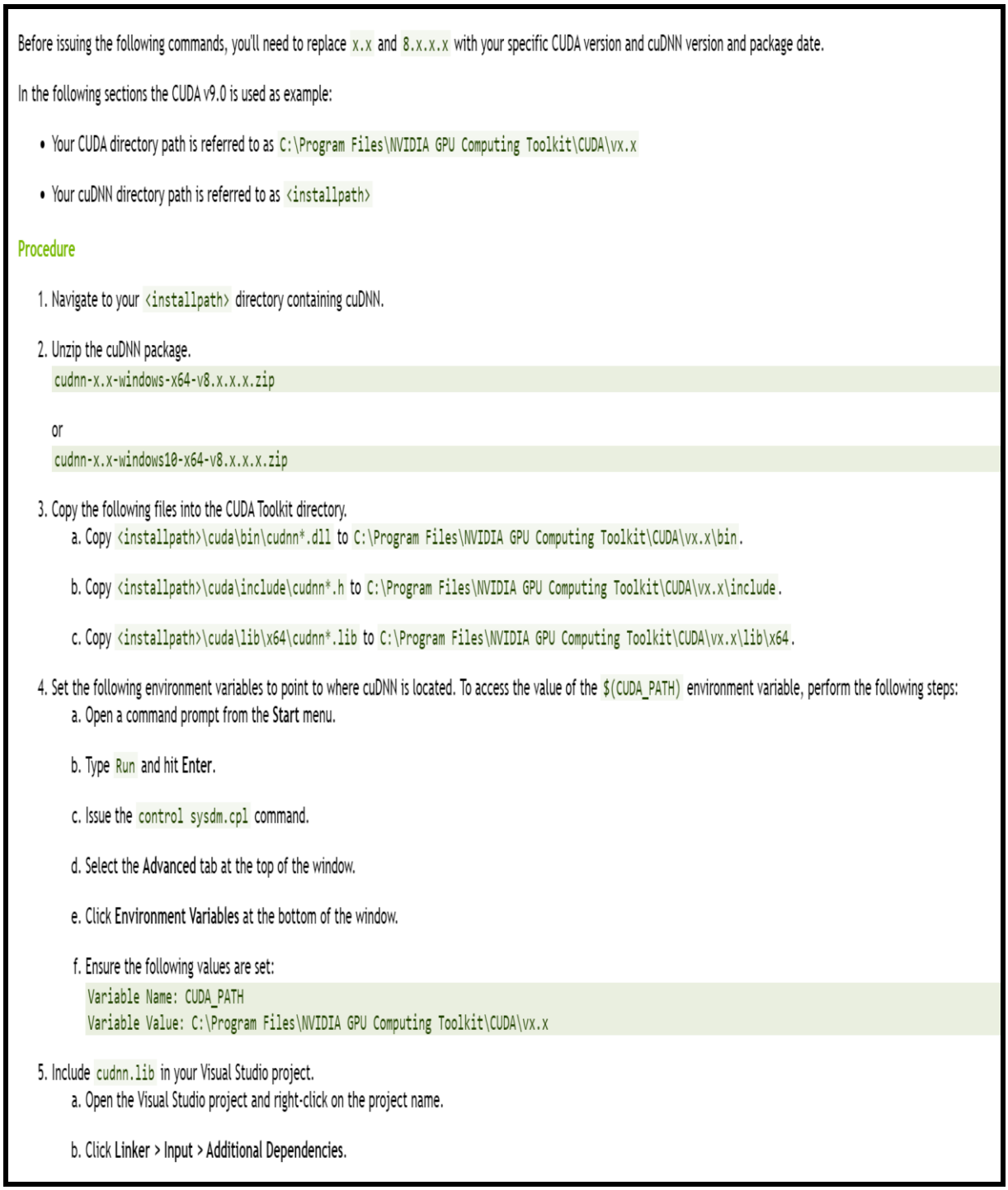
Procedure

1. Go to: NVIDIA cuDNN home page.
2. Click **Download**.
3. Complete the short survey and click **Submit**.
4. Accept the Terms and Conditions. A list of available download versions of cuDNN displays.
5. Select the cuDNN version to want to install. A list of available resources displays.
6. Extract the cuDNN archive to a directory of your choice.

3.3. Installing cuDNN On Windows

The following steps describe how to build a cuDNN dependent program.

Figure 21 : CUDNN Installation Process Stage 1



Before issuing the following commands, you'll need to replace `x.x` and `8.x.x.x` with your specific CUDA version and cuDNN version and package date.

In the following sections the CUDA v9.0 is used as example:

- Your CUDA directory path is referred to as `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x`
- Your cuDNN directory path is referred to as `<installpath>`

Procedure

1. Navigate to your `<installpath>` directory containing cuDNN.
2. Unzip the cuDNN package.
   `cudnn-x.x-windows-x64-v8.x.x.x.zip`

   or

   `cudnn-x.x-windows10-x64-v8.x.x.x.zip`
3. Copy the following files into the CUDA Toolkit directory.
   a. Copy `<installpath>\cuda\bin\cudnn*.dll` to `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\bin`.

   b. Copy `<installpath>\cuda\include\cudnn*.h` to `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\include`.

   c. Copy `<installpath>\cuda\lib\x64\cudnn*.lib` to `C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x\lib\x64`.
4. Set the following environment variables to point to where cuDNN is located. To access the value of the `$(CUDA_PATH)` environment variable, perform the following steps:
   a. Open a command prompt from the **Start** menu.

   b. Type `Run` and hit **Enter**.

   c. Issue the `control sysdm.cpl` command.

   d. Select the **Advanced** tab at the top of the window.

   e. Click **Environment Variables** at the bottom of the window.

   f. Ensure the following values are set:
      `Variable Name: CUDA_PATH`
      `Variable Value: C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\vx.x`
5. Include `cudnn.lib` in your Visual Studio project.
   a. Open the Visual Studio project and right-click on the project name.

   b. Click **Linker** > **Input** > **Additional Dependencies**.

Figure 22 : CUDNN Installation Process Stage 2

Both packages are required to utilize the advances TensorFlow has made in GUP support.

4.6 Using Anaconda Navigator

Anaconda Navigator was used to install some of the required packages. Others were directly installed using the PIP command. Installation of opencv for python is detailed in Figure 23

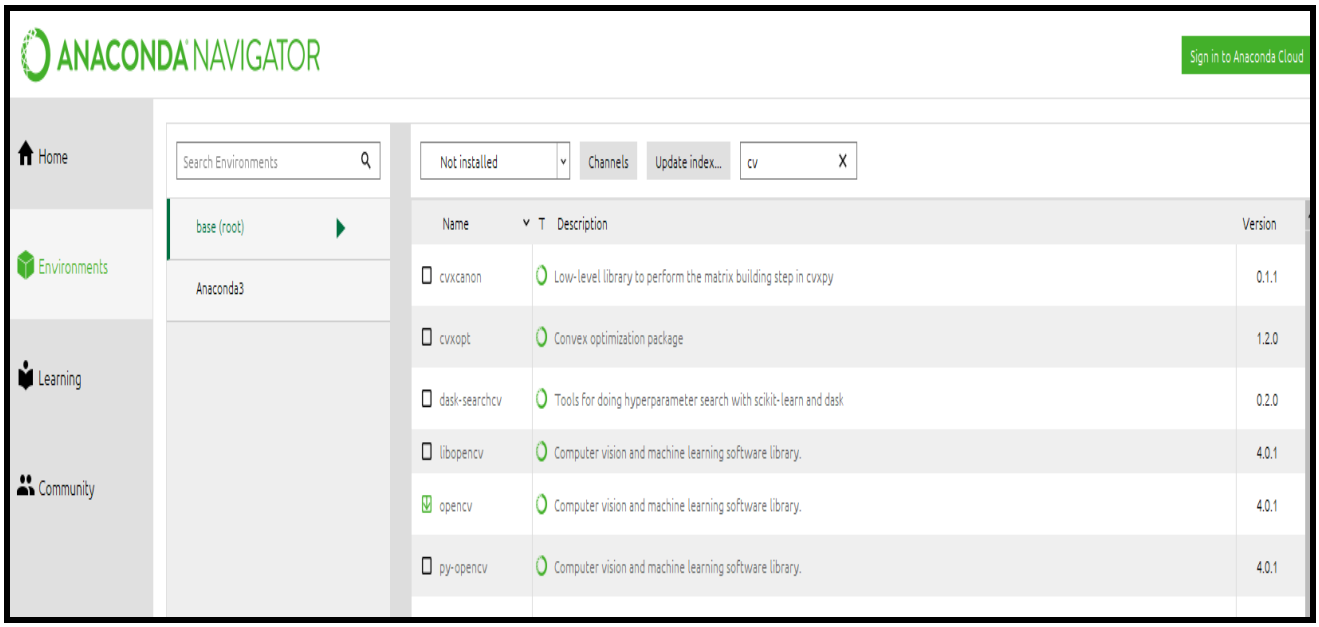


Figure 23 : Using Anaconda Navigator to install packages

This was to support the cv2 library required to work with images in python. And resulted in additional dependencies, see Figure 24, being updated.

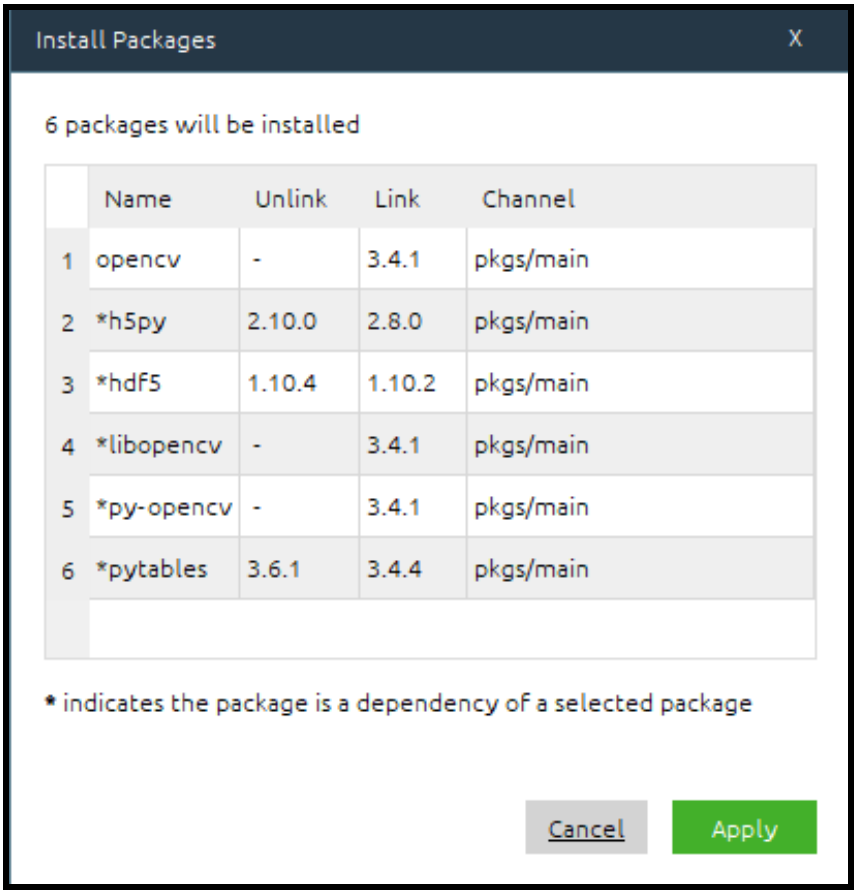


Figure 24 : Packages installed via Anaconda Navigator

4.7 Packages installed using PIP

Httplib was installed using the Python PIP command as detailed in Figure 25:

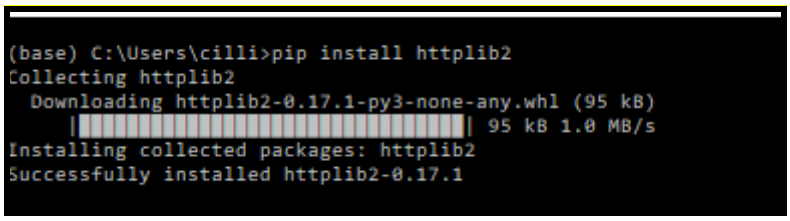


Figure 25 : Screen Shot of HTTPLIB installation

Tensorflow Nightly Update to get latest support for the libraries being used

Pip install tf-nightly

The version of Tensorflow installed for this project was 2.3 and a nightly dev version which had stable support for GPU and other packages used in this project is detailed in Figure 26

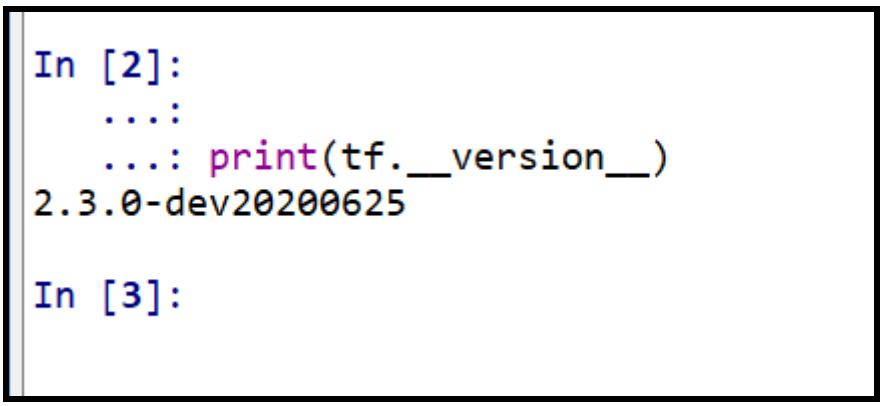


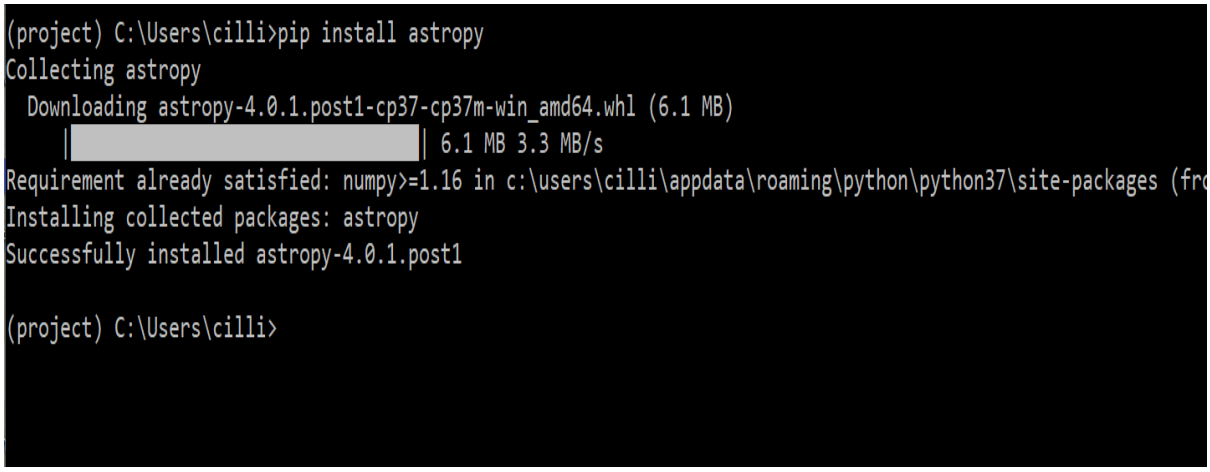
Figure 26 : Print version of TensorFlow

PIP install tf-nighly-gpu

To get the latest GPU support – this allowed for faster processing by moving some of the processing from the CPU to the GPU's in the graphics card.

The following Python libraries were installed using "pip install" followed by the package name:

- matplotlib
- PLI
- Pillow
- Install aspropy in order to be able to manage FITS files as in Figure 27.



- Figure 27 : Installation of astropy

5 How to download project images

Images were downloaded from SDSS and Kepler archive servers. In both cases the format of the queries was dictated by the site in question. For SDSS it was a simple wget command as outlines below in Section 5.1 to below.

The query from Kepler was designed by the STSCI.EDU and a modification to their script allowed for it to take an input file and work through lines in the input file to process more than a single image at a time.

5.1 SQL to SDSS Catalogue Archive Servers

The command below, as per Figure 28 was run on the SDSS CAS servers, which are found at: http://skyserver.sdss.org/dr9/en/tools/search/sql.asp

. The command to extract the details of the required images is per Figure 28, Figure 29

```
Line#
1.    SELECT TOP 100000
2.     p.objid,p.ra,p.dec,p.u,p.g,p.r,p.i,p.z,
3.     p.run, p.rerun, p.camcol, p.field,
4.     s.specobjid, s.class, s.z as redshift,
5.     s.plate, s.mjd, s.fiberid
6.    FROM PhotoObj AS p
7.     JOIN SpecObj AS s ON s.bestobjid = p.objid
8.    WHERE
9.     p.u BETWEEN 0 AND 19.6
10.    AND g BETWEEN 0 AND 20
```

Figure 28 - SQL Command to extract catalogue from SDSS Catalogue Servers

The extracted excel file is included in the accompanying pack and it called Skyserver_SQO4_6_2020 1_30_47 PM (version 4).xls

This returned 10000 records which listed the fields which later were used to create the input files for "wget" to extract the images from the sites. The pattern for SDSS and STScI were different, however the data returned by the SDSS CAS servers allowed both request formats to be generated.

The SQL returned a CSV file, Figure 29, which was taken into Excel and the two extraction formats for images were generated using a combination of concatenation of returned fields and text.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1																		
2	objid	ra	dec	u	g	r	i	z	run	rerun	camcol	field	specob	class	redshif	plate	mjd	fiberid
6	1.24E+18	49.6275	-1.04177	17.6561	16.1713	15.5894	15.3785	15.2674	109	301	1	100	1.71E+18	STAR	-9.77E-05	1515	52932	208
7	1.24E+18	40.2721	-0.64251	19.234	17.5333	16.8743	16.6316	16.4976	109	301	2	37	1.20E+18	STAR	-5.22E-05	1070	52591	114
9	1.24E+18	40.582	0.13477	18.6456	16.4434	15.5245	15.1819	14.9886	109	301	4	39	8.25E+18	STAR	0.00017	7330	56684	997
12	1.24E+18	57.2816	0.01877	16.4848	14.9299	14.5605	14.5305	14.1939	109	301	4	151	1.40E+18	STAR	-0.00014	1242	52901	476
13	1.24E+18	57.5121	0.08489	18.839	17.6309	17.0908	16.8463	16.7146	109	301	4	153	1.84E+18	STAR	8.89E-05	1633	52998	491
14	1.24E+18	57.6054	0.02728	18.218	15.9543	14.9567	14.5948	14.3627	109	301	4	153	1.40E+18	STAR	-2.62E-05	1242	52901	495
16	1.24E+18	57.9435	0.05968	16.934	15.3849	14.6991	14.4432	14.3309	109	301	4	155	1.40E+18	STAR	0.00012	1242	52901	544
18	1.24E+18	58.304	0.01381	18.5322	17.2466	16.7749	16.5976	16.5032	109	301	4	158	1.72E+18	STAR	4.66E-05	1529	52930	533
19	1.24E+18	58.3957	0.20977	17.0049	15.3609	14.4984	14.3981	13.7894	109	301	4	158	1.40E+18	STAR	0.00061	1242	52901	633
20	1.24E+18	36.6537	0.6311	19.4573	18.126	17.6266	17.453	17.3283	109	301	5	13	8.26E+18	STAR	9.13E-05	7333	56686	921
21	1.24E+18	37.6901	0.63037	19.25	18.3297	17.9823	17.8607	17.7824	109	301	5	20	8.81E+18	STAR	0.0001	7828	57039	838

Figure 29 : CSV output from SDSS CAS Servers opened in Excel

5.2 SDSS Image Download – FITS files

Complete documentation on how to download the images from SDSS was described at https://dr12.sdss.org/documentation as is in Figure 30

The **fields** page, which displays a given imaging field and information about the data, can be accessed directly with the RUN, CAMCOL, and FIELD of the field:

`http://dr12.sdss.org/fields/runCamcolField?run=RUN&camcol=CAMCOL&field=FIELD`

e.g., http://dr12.sdss.org/fields/runCamcolField?field=187&camcol=3&run=3712

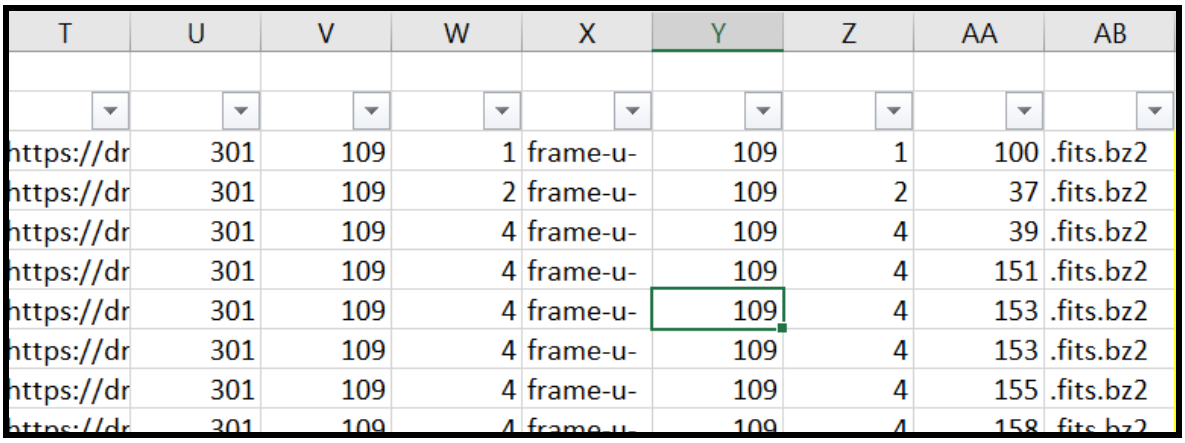
This page opens onto a view of the field, with some of its associated information, with some links to download the full FITS format spectrum. To download the images directly, the most efficient way is to access the flat file at its URL:

`http://dr12.sdss.org/sas/dr12/boss/photoObj/frames/RERUN/RUN/CAMCOL/frame-FILTER-RUN6-CAMCOL-FIELD.fits.bz2`

where 'RERUN' should be replaced by the appropriate imaging reduction number (currently '301'), 'RUN' should be replaced by the run number, 'CAMCOL' should be replaced by the single-digit camcol number (1-6), 'FILTER' should be replaced by the filter name ('u', 'g', 'r', 'i', 'z'), 'RUN6' should be replaced by the zero-padded, 6-digit run number and 'FIELD' should be replaced by the zero-padded, 4-digit field number.

Figure 30 : Instructions from SDSS on how to generate URL to download images

By following th eformat as perscribed by SDSS and adding in new fields to make up the desired enteries to the output from the CAS Excel file,Figure 31, a subsequent concatanation function was run to complete the formatting and allow for extraction of the data into text files for further processing



T	U	V	W	X	Y	Z	AA	AB
https://dr	301	109	1	frame-u-	109	1	100	.fits.bz2
https://dr	301	109	2	frame-u-	109	2	37	.fits.bz2
https://dr	301	109	4	frame-u-	109	4	39	.fits.bz2
https://dr	301	109	4	frame-u-	109	4	151	.fits.bz2
https://dr	301	109	4	frame-u-	109	4	153	.fits.bz2
https://dr	301	109	4	frame-u-	109	4	153	.fits.bz2
https://dr	301	109	4	frame-u-	109	4	155	.fits.bz2
https://dr	301	109	4	frame-u-	109	4	158	.fits.bz2

Figure 31 : Excel fields used to generate URL

The contatanation to build the final URL required is in Figure 32:

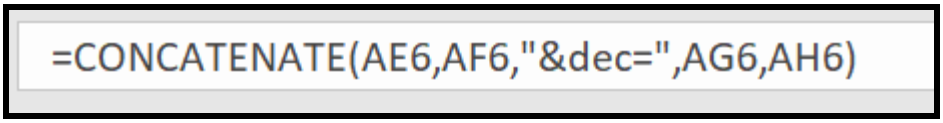
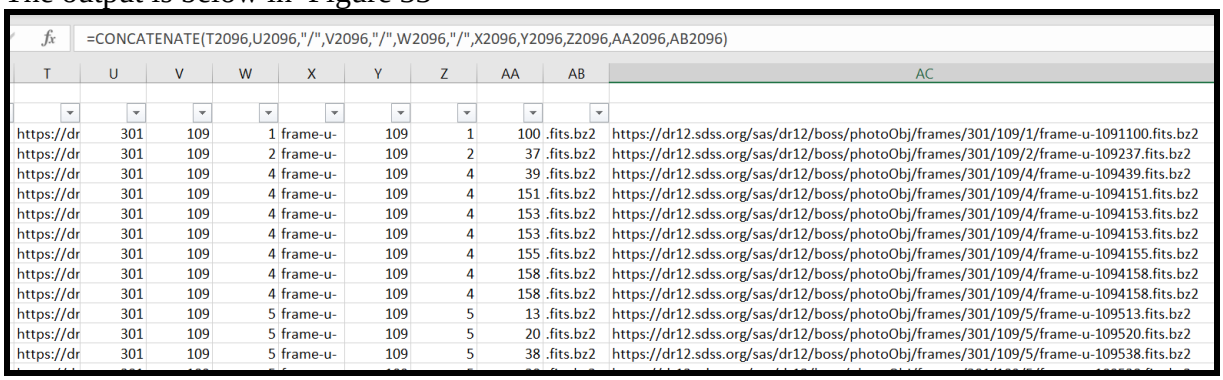


Figure 32 : concatenation function to generate URL

The output is below in Figure 33



T	U	V	W	X	Y	Z	AA	AB	AC
https://dr	301	109	1	frame-u-	109	1	100	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/1/frame-u-1091100.fits.bz2
https://dr	301	109	2	frame-u-	109	2	37	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/2/frame-u-109237.fits.bz2
https://dr	301	109	4	frame-u-	109	4	39	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/4/frame-u-109439.fits.bz2
https://dr	301	109	4	frame-u-	109	4	151	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/4/frame-u-1094151.fits.bz2
https://dr	301	109	4	frame-u-	109	4	153	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/4/frame-u-1094153.fits.bz2
https://dr	301	109	4	frame-u-	109	4	153	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/4/frame-u-1094153.fits.bz2
https://dr	301	109	4	frame-u-	109	4	155	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/4/frame-u-1094155.fits.bz2
https://dr	301	109	4	frame-u-	109	4	158	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/4/frame-u-1094158.fits.bz2
https://dr	301	109	4	frame-u-	109	4	158	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/4/frame-u-1094158.fits.bz2
https://dr	301	109	5	frame-u-	109	5	13	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/5/frame-u-109513.fits.bz2
https://dr	301	109	5	frame-u-	109	5	20	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/5/frame-u-109520.fits.bz2
https://dr	301	109	5	frame-u-	109	5	38	.fits.bz2	https://dr12.sdss.org/sas/dr12/boss/photoObj/frames/301/109/5/frame-u-109538.fits.bz2

Figure 33 : Example of format required for SDSS files to be downloaded

Using wget and the concatenation from above as an input file the following commands were used to download the required images from the SDSS servers

Wget -i input.txt where input.txt is a text file listing the fits files in the format as per Figure 34

This proceeded to call the SDSS servers and download the fits files as per the input.txt file. |n example of the flow is shown in Figure 34 which shows the “starts” files being downloaded.

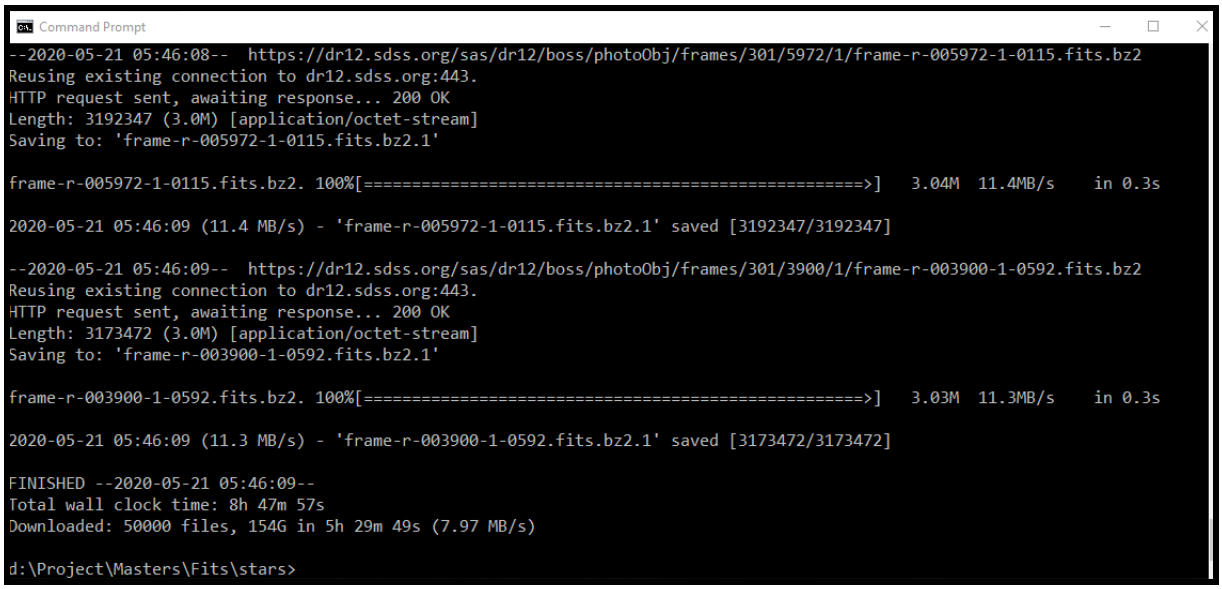


Figure 34 : Fits files downloading

5.3 Structure of FITS files

The following section shows the structure of the FITS files extracted from the SDSS CAS servers. Is detailed in Figure 35, Figure 36, Figure 37,.

```
In [202]: hdu.info()
Filename: d:\project\masters\frame-u-005194-2-0645.fits
No.    Name      Ver    Type      Cards   Dimensions   Format
  0  PRIMARY       1 PrimaryHDU      96   (2048, 1489)   float32
  1                1 ImageHDU         6   (2048,)   float32
  2                1 BinTableHDU     27   1R x 3C   [49152E, 2048E, 1489E]
  3                1 BinTableHDU     79   1R x 31C   [J, 3A, J, A, D, D, 2J, J, D, D, D, D, D, D, D, D,
D, D, D, D, D, D, D, D, D, D, D, D, E, E]
```

Figure 35 : General structure of SDSS FITS file

```
In [204]: hdu[0].header
Out[204]:
SIMPLE  =                    T /
BITPIX  =                  -32 / 32 bit floating point
NAXIS   =                    2
NAXIS1  =                 2048
NAXIS2  =                 1489
EXTEND  =                    T /Extensions may be present
BZERO   =              0.00000 /Set by MRD_SCALE
BSCALE  =              1.00000 /Set by MRD_SCALE
TAI     =        4617336121.16 / 1st row - Number of seconds since Nov 17 1858
RA      =            214.07860 / 1st row - Right ascension of telescope boresigh
DEC     =            19.900436 / 1st row - Declination of telescope boresight (d
SPA     =             100.719  / 1st row - Camera col position angle wrt north (
IPA     =             131.780  / 1st row - Instrument rotator position angle (de
IPARATE =              0.0005  / 1st row - Instrument rotator angular velocity (
AZ      =            35.304994 / 1st row - Azimuth  (encoder) of tele (0=N?) (de
ALT     =            74.462173 / 1st row - Altitude (encoder) of tele        (de
FOCUS   =           -158.00000 / 1st row - Focus piston (microns?)
DATE-OBS= '2005-03-12'         / 1st row - TAI date
TAIHMS  = '09:22:01.15'        / 1st row - TAI time (HH:MM:SS.SS) (TAI-UT = appr
COMMENT  TAI,RA,DEC,SPA,IPA,IPARATE,AZ,ALT,FOCUS at reading of col 0, row 0
ORIGIN  = 'SDSS    '
TELESCOP= '2.5m    '
TIMESYS = 'TAI     '
RUN     =                 5194 / Run number
FRAME   =                  649 / Frame sequence number within the run
CCDLOC  =                   32 / Survey location of CCD (e.g., rowCol)
STRIPE  =                   19 / Stripe index number (23 <--> eta=0)
STRIP   = 'S       '           / Strip in the stripe being tracked.
FLAVOR  = 'science '           / Flavor of this run
OBSERVER= 'sjnk    '           / Observer
SYS_SCN = 'mean    '           / System of the scan great circle (e.g., mean)
EQNX_SCN=           2000.00    / Equinox of the scan great circle. (years)
NODE    =           95.00000   / RA of the great circle's ascending node (deg)
INCL    =           22.50000   / Great circle's inclination wrt cel. eq. (deg)
XBORE   =           22.74      / Boresight x offset from the array center (mm)
YBORE   =           0.00       / Boresight x offset from the array center (mm)
OBJECT  = '19 S    '           / e.g., 'stripe 50.6 degrees, north strip'
EXPTIME = '53.907456'          / Exposure time (seconds)
SYSTEM  = 'FK5     '           / System of the TCC coordinates (e.g., mean)
CCDMODE = 'DRIFT   '           / 'STARING' or 'DRIFT'
C_OBS   =                26322 / CCD row clock rate (usec/row)
COLBIN  =                    1 / Binning factor perpendicular to the columns
ROWBIN  =                    1 / Binning factor perpendicular to the rows
DAVERS  = 'v14_47  '           / Version of DA software
SCDMETHD= 'sqrtDynamic'        / scdMethod
SCDWIDTH=                 1280 / scdDisplayWidth
```

Figure 36 :Details of FITS Header - 1

```
SCDDECMF=                   1 / scdDecimateFactor
SCDOFSET=                 410 / scdDisplayOffset
SCDDYNTH=                 -19 / scdDynamicThresh
SCDSTTHL=                  30 / scdStaticThreshL
SCDSTTHR=                  30 / scdStaticThreshR
SCDREDSZ=                 478 / scdReduceSize
SCDSKYL =                   0 / scdSkyLeft
SCDSKYR =                   0 / scdSkyRight
COMMENT  CCD-specific parameters
CAMROW  =                   3 / Row in the imaging camera
BADLINES=                   0 / Number of bad lines in frame
EQUINOX =             2000.00 /
SOFTBIAS=                1000 / software "bias" added to all DN
BUNIT   = 'nanomaggy'         / 1 nanomaggy = 3.631e-6 Jy
FILTER  = 'u         '        / filter used
CAMCOL  =                   2 / column in the imaging camera
VERSION = 'v5_6_3  '
DERV_VER= 'NOCVS:v8_23'
ASTR_VER= 'NOCVS:v5_24'
ASTRO_ID= '2009-06-05T19:43:21 08156'
BIAS_ID = 'PS      '
FRAME_ID= '2009-10-04T08:42:48 27573'
KO_VER  = 'devel   '
PS_ID   = '2009-06-05T03:58:20 02241 camCol 2'
ATVSN   = 'NOCVS:v5_24'       / ASTROTOOLS version tag
RADECSYS= 'ICRS    '          / International Celestial Ref. System
CTYPE1  = 'RA---TAN'          /Coordinate type
CTYPE2  = 'DEC--TAN'          /Coordinate type
CUNIT1  = 'deg     '          /Units
CUNIT2  = 'deg     '          /Units
CRPIX1  =       1025.00000000 /X of reference pixel
CRPIX2  =       745.000000000 /Y of reference pixel
CRVAL1  =       213.904063951 /RA of reference pixel (deg)
CRVAL2  =       19.1827457096 /Dec of reference pixel (deg)
CD1_1   =    2.03108409294E-05 /RA deg per column pixel
CD1_2   =    0.000108085004949 /RA deg per row pixel
CD2_1   =    0.000108089982035 /Dec deg per column pixel
CD2_2   =   -2.03493686233E-05 /Dec deg per row pixel
HISTORY GSSSPUTAST: Oct  4 08:43:04 2009
COMMENT  Calibration parameters
COMMENT  Floats truncated at 10 binary digits with FLOATCOMPRESS
NMGY    =            0.0100702 / Calibration factor [nMgy per count]
NMGYIVAR=             0.151854 / Calibration factor inverse variance
VERSIDL = '7.0     '          / Version of IDL
VERSUTIL= 'v5_5_5  '          / Version of idlutils
VERSPOP = 'v1_11_1 '          / Version of photoop product
PCALIB  = '/clusterfs/riemann/raid006/bosswork/groups/boss/calib/2009-06-14/cal'
PSKY    = '/clusterfs/riemann/raid006/bosswork/groups/boss/photo/sky' / Value of
RERUN   = '301     '          / rerun
HISTORY SDSS_FRAME_ASTROM: Astrometry fixed for dr9 Sat Jun 23 07:46:30 2012
```

Figure 37 : Details of FITS Header - 2

6 SDSS Images

The images from SDSS used fel into 2 categories, either filter images across the 5 bands or RGB which were a combination of the R, G, B bands of the filters from category 1.

6.1 Extracting Images from FITS files

This file sets the working directory to whichever of the folder the images were downloaded. In this case it was d:\project\masters\fits and then either the galaxy, starts or quasar folders. It then recursively goes through the images and extracts a plot for each image and saves to the same folder. The request is all plots are extracted for each of the image types in one of three folders corresponding to their classification. See Figure 38 for details.

The plot is shown on the console as well as the image name to show the progress through the images, as well as a count to ensure that progress is monitored.

```
"""
Created on Tue Mar 31 20:30:38 2020

@author: - Cillin O Foghlu
Student ID 18186751
"""

#import numpy as np
#import pandas as pd
#from array import *
from astropy.io import fits
import matplotlib.pyplot as plt
from astropy.visualization import astropy_mpl_style
plt.style.use(astropy_mpl_style)
import os
import glob, os
os.chdir('d:\\project\\masters\\fits\\galaxy')
#import cv2

# open input file to process it line by line.
# each line is the name of a fits compressed file
# after taking in the file name the end of line ( "\n" ) character must be removed

i = 0

for file in glob.glob("**/*.fits.bz2", recursive = True):
    fits_file = fits.open(file, memmap=True)
    image = fits_file[0].data

#   plot the image and then save it with the new naming format
#   close plots once saved to conserve memory

    plt.figure()
    plt.imshow(image, cmap='gray', vmin=0, vmax=0)
    plt.colorbar()
    # remore the "'fits.bz2 " from file names
    file = file[:-9]
    # add an "g" to the file to signify that the image file is for a galaxy
    plt.savefig("g" + file + ".png")
    plt.close()
    i= i+1
    # tracker to show progress
    print(i)
    print(file)
```

Figure 38 : Code to extract plot from FITS file

The images were extracted, and a sample is in Figure 39

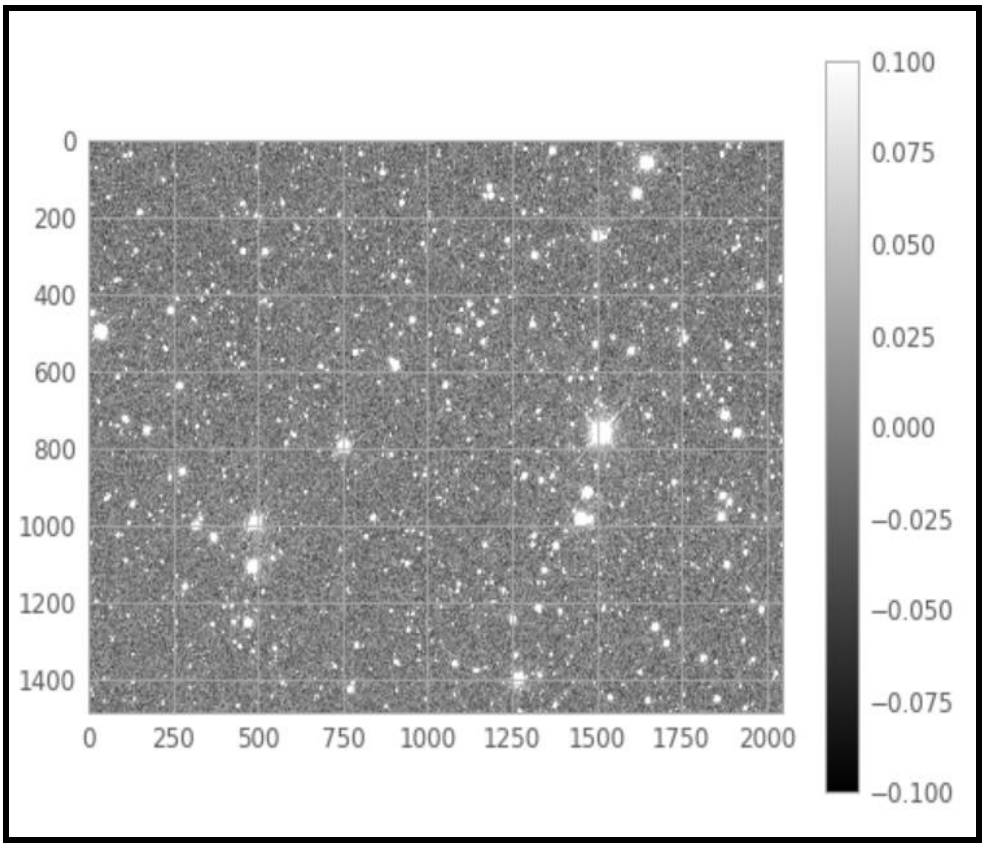


Figure 39 - This is a galaxy plot extracted from a FITS file.

The images were saved to separate folders for each category, this later became the input folders for the different datasets and working folders parameters.

6.2 Creating RGB from FITS filter images

This Python file opens the filter images from SDSS based on their "G" band and then combines the R and B bands along with the G band to make a single image which is then saved as a PNG to the working directory. This provided a second image format for training and testing by the ANN and the process is in Figure 40 with a sample output in Figure 41

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 31 20:30:38 2020

@author: - Cillin O Foghlu
Student ID 18186751
"""

#import numpy as np
#import pandas as pd
#from array import *
#import numpy as np
import cv2
import glob, os

#from matplotlib import pyplot as plt
#from PIL import Image

os.chdir('d:\\project\\v2\\galaxy')

# open input file to process it line by line.
# for each "u" band image open the correcponding
# other filter images to make an RBG image

i = 0

for file in glob.glob("gframe-u-*.png", recursive = True):
    # remove the .png
    file = file[:-4]
    # remove the first 9 charecters from name
    #print(file)
    file_start = file[:7]
    file_end = file[8:]
    r_channel = cv2.imread(file_start + "r" + file_end + ".png",0)
    g_channel = cv2.imread(file_start + "g" + file_end + ".png",0)
    b_channel = cv2.imread(file_start + "u" + file_end + ".png",0)

    # print(r_channel, g_channel, b_channel)

    file_name = "RGB"+ file_end + ".png"
    print(file_name)
    img = cv2.merge((b_channel, g_channel, r_channel))
    cv2.imwrite(file_name, img)
    i = i+1
    print(i)
```

Figure 40 : Code to Combine RBG filters to a single file

The directory used here is for “v2” which refers to the RGB Images from SDSS. Other directories were used referring to the filters and jpg images from the data sources – see section 8.3 later in this document for more details.

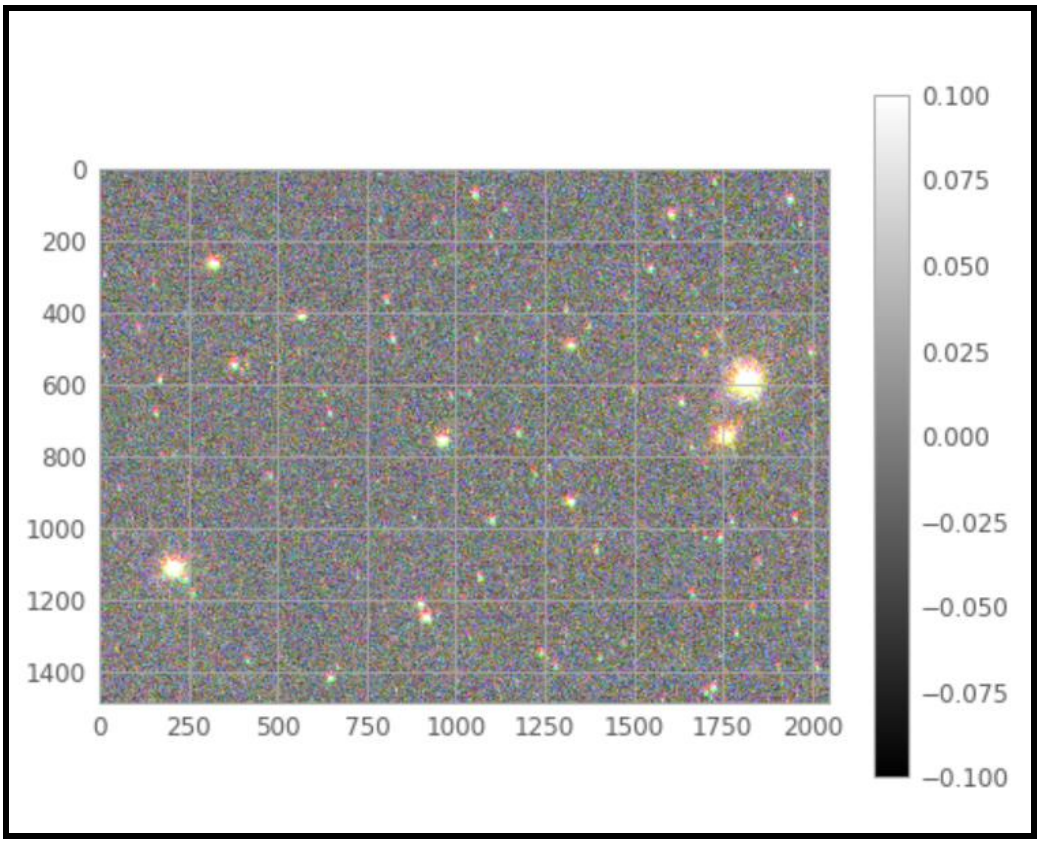


Figure 41 : Sample Star image below is a combination of RGB B&W filters as be the script above.

6.3 Extracting RGB images directly form SDSS

Details of how to extract cut-out jpg images form the FITS images produced is documented at http://skyserver.sdss.org/dr16/en/help/docs/api.aspx#imgcutout

This was built using the same output from SQL CAS request excel file and a concatenation function to generate the URL's which were extracted using a *wget* command. As per Figure 42 with a sample of the output in Figure 43 - Sample of a Quasar JPG from SDSS Archive Servers

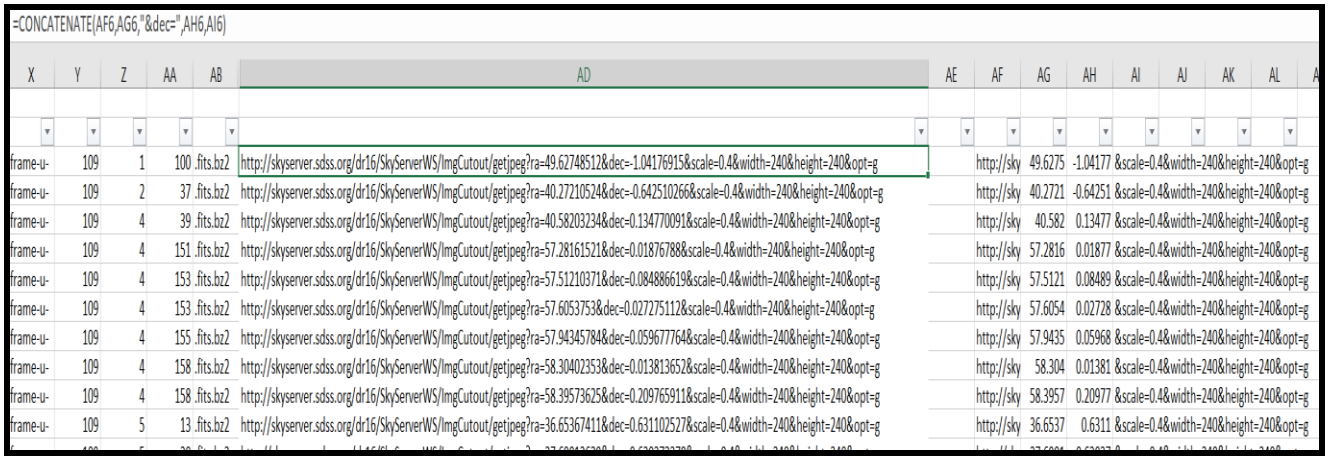


Figure 42 : Concatenation function to generate SDSS RGB Image

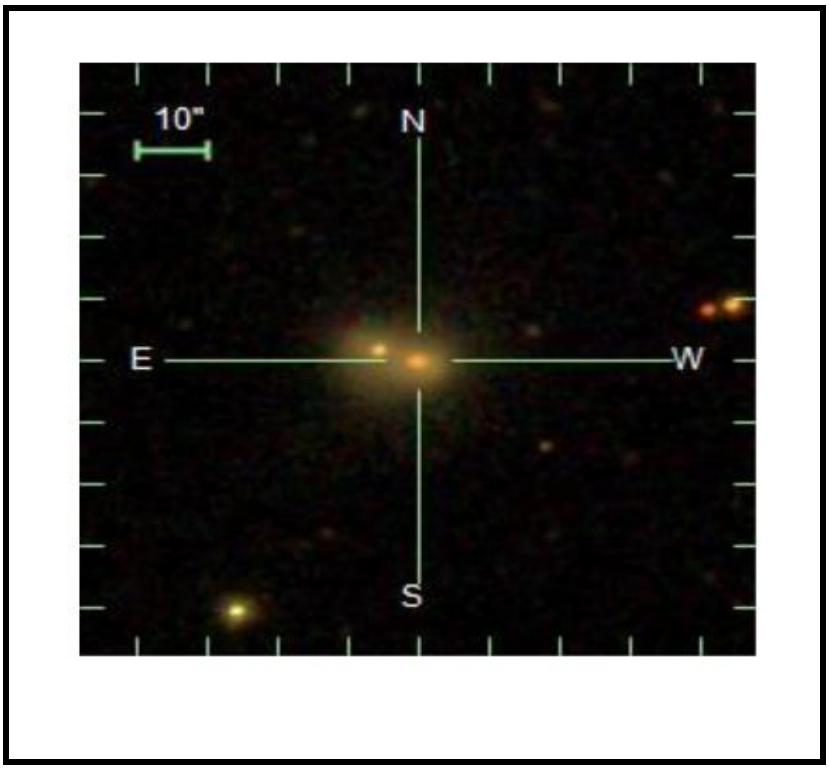


Figure 43 - Sample of a Quasar JPG from SDSS Archive Servers

7 STScI Images

Most of the code used below is provided by the STScI support team to allow for the extraction of jpg images from the site is outlined in Figure **44** : Code to extract JPG from STScI site - part 1Figure 44 and Figure 45

```
@author: cilli
"""
"""
Script thanks to STSCI.EDU

"""
#from __future__ import print_function
import matplotlib
import numpy
from astropy.table import Table
import requests
from PIL import Image
from io import BytesIO
import pylab
import cv2
import csv
import numpy as np
import os

os.chdir('d:\\project\\V2\\Images')
# wk_dir = 'd:\\project\\masters\\RGB'


def getimages(ra,dec,size=240,filters="grizy"):

    """Query ps1filenames.py service to get a list of images

    ra, dec = position in degrees
    size = image size in pixels (0.25 arcsec/pixel)
    filters = string with filters to include
    Returns a table with the results
    """

    service = "https://ps1images.stsci.edu/cgi-bin/ps1filenames.py"
    url = ("{service}?ra={ra}&dec={dec}&size={size}&format=fits"
           "&filters={filters}").format(**locals())
    table = Table.read(url, format='ascii')
    return table


def geturl(ra, dec, size=240, output_size=None, filters="grizy", format="jpg", color=False):

    """Get URL for images in the table

    ra, dec = position in degrees
    size = extracted image size in pixels (0.25 arcsec/pixel)
    output_size = output (display) image size in pixels (default = size).
                  output_size has no effect for fits format images.
    filters = string with filters to include
    format = data format (options are "jpg", "png" or "fits")
    color = if True, creates a color image (only for jpg or png format).
            Default is return a list of URLs for single-filter grayscale images.
    Returns a string with the URL
    """

    if color and format == "fits":
        raise ValueError("color images are available only for jpg or png formats")
    if format not in ("jpg","png","fits"):
        raise ValueError("format must be one of jpg, png, fits")
    table = getimages(ra,dec,size=size,filters=filters)
    url = ("https://ps1images.stsci.edu/cgi-bin/fitscut.cgi?"
           "ra={ra}&dec={dec}&size={size}&format={format}").format(**locals())
    if output_size:
        url = url + "&output_size={}".format(output_size)
    # sort filters from red to blue
    flist = ["yzirg".find(x) for x in table['filter']]
    table = table[numpy.argsort(flist)]
    if color:
        if len(table) > 3:
            # pick 3 filters
            table = table[[0,len(table)//2,len(table)-1]]
        for i, param in enumerate(["red","green","blue"]):
            url = url + "&{}={}".format(param,table['filename'][i])
    else:
        urlbase = url + "&red="
        url = []
        for filename in table['filename']:
            url.append(urlbase+filename)
    return url
```

Figure 44 : Code to extract JPG from STScI site - part 1

```
def getcolorim(ra, dec, size=240, output_size=None, filters="grizy", format="jpg"):

    """Get color image at a sky position
        ra, dec = position in degrees
    size = extracted image size in pixels (0.25 arcsec/pixel)
    output_size = output (display) image size in pixels (default = size).
                  output_size has no effect for fits format images.
    filters = string with filters to include
    format = data format (options are "jpg", "png")
    Returns the image
    """

    if format not in ("jpg","png"):
        raise ValueError("format must be jpg or png")
    url = geturl(ra,dec,size=size,filters=filters,output_size=output_size,format=format,color=True)
    r = requests.get(url)
    im = Image.open(BytesIO(r.content))
    im.save(Class + str(i)+".jpg")
    return im

input = 'd:\\project\\V2\quasar.txt'
i = 0

file = open(input,'r')
reader = csv.reader(file, delimiter='\t')
for row in reader:
    ra = row[0]
    dec = row[1]
    Class = row[-1]
    getcolorim(ra, dec)
#    img = np.asarray(im)
#    cv2.imwrite(Class + str(i)+".jpg", im)
    i = i+1
    print(i)
print("EoF Encountered")
file.close()
```

Figure 45 : Code to extract JPG from STScI site - part 2

By changing the code to recursivally go through an input file the code was modified to take an input and extract, download all required images in one process. The input fiole tool the format as per Figure 46 : Input to GetImagesfromKepler.py. The data for these images was also taken form the excel file “Skysaver_SQL4_6_2020 1_3-+47 PM (version4).xls which is submitted as part if the ICT solution pack. These fields refer to the RA and DEC and classifiation of the object in question.

132.1085388	53.20049113	QSO
132.1900337	53.17556786	GALAXY
132.3151955	53.07444632	STAR
132.354367	53.09479769	GALAXY
132.2900078	53.14695407	GALAXY
132.322154	53.29881358	QSO
132.3247705	53.2467918	GALAXY
132.3321314	53.29167705	GALAXY
132.3423819	53.23397582	GALAXY
132.455415	53.2577198	GALAXY
132.5234916	53.21036377	QSO
132.4113391	53.31478984	GALAXY
132.6239735	53.27014764	GALAXY
132.719505	53.33968201	GALAXY
132.732436	53.31753694	GALAXY
132.8019542	53.37078788	STAR
132.8496165	53.58139549	GALAXY
133.0020506	53.64085797	GALAXY
133.0206175	53.61730754	GALAXY
133.0355275	53.70006787	GALAXY
133.1291602	53.6505037	STAR
133.0895598	53.73079153	GALAXY
133.285901	53.90305126	GALAXY
133.408163	53.89774132	QSO

Figure 46 : Input to GetImagesfromKepler.py

The images downloaded with no extension so a simple Python code was developed to rename the files. All images started with “getjpeg” and this was used as key to find image files and append a jpg to the filename. See Figure 47 for the code and Figure 48 for a sample of the putput.

```
# -*- coding: utf-8 -*-
"""
Created on Tue Mar 31 20:30:38 2020

@author: - Cillin O Foghlu
Student ID 18186751
"""

import glob, os


os.chdir('d:\\project\\v4\\quasar')
i = 0

for file in glob.glob("getjpeg*.*", recursive = True):

    print(file)
    os.rename(file, file + '.jpg')
    i = i+1
    print(i)
```

Figure 47 : code to rename STScI download images as JPG files

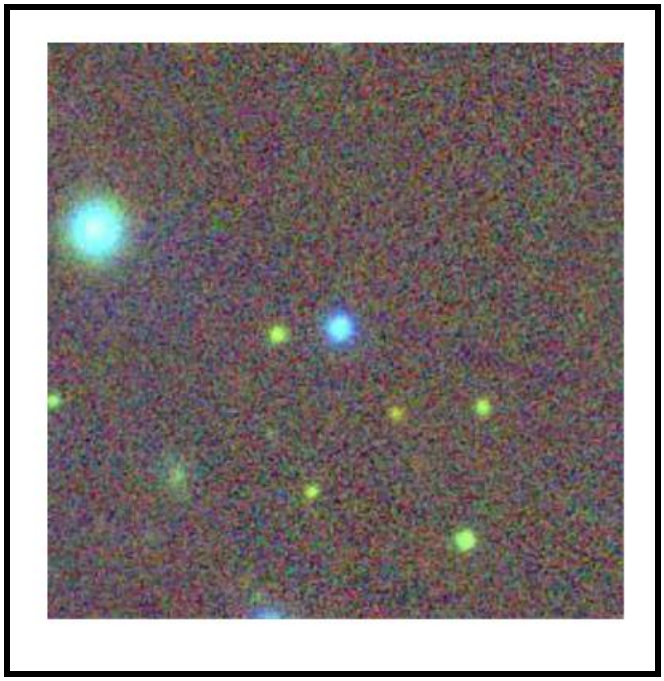


Figure 48 - Sample quasar from STScI Kepler servers

8 Models and Training Code

All models used for training the various datasets are included in the accompanying pack. The models covered in this project utilised MobileNet, ResNet50, VGG16, and Xception. Each

model followed the same format, however the trained weights from the model as it was trained using Image Net images was downloaded on the initial run for each model – see Figure 49.

```
gg16_weights_tf_dim_ordering_tf_kernels_notop.h5
5408256/58889256 [=====================>.......] - ETA: 1s
```

Figure 49 : example of weights being downloaded

The code used comes in 4 parts – the initial addition of required python libraries, the inclusion of the ImageNet trained model, the importing of the dataset for training and testing, followed by the training, testing and output of results. The results for all models will be covered in Section 8.4

8.1 Inclusion of required Python Libraries and environmental variables

The same libraries were included in all models and the code is as per Figure 50:

```
import tensorflow as tf
import numpy as np
import os
import matplotlib.pyplot as plt
import datetime
import pathlib

#from tensorflow import keras
#from tensorflow.keras import preprocessing
#from tensorflow.keras.applications.resnet50 import ResNet50
#from tensorflow.keras.preprocessing import image
#from tensorflow.keras.applications.resnet50 import preprocess_input, decode_predictions
from tensorflow.keras.callbacks import TensorBoard
from tensorflow.keras.layers import Dense,GlobalAveragePooling2D
from tensorflow.keras.applications import MobileNet
#from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet import preprocess_input
#from tensorflow.keras.models import Model
#from tensorflow.keras.optimizers import Adam
#from tensorflow.keras.models import Sequential
#from tensorflow.keras.layers import Dense, Conv2D, Flatten, Dropout, MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# GPU initialization/configuration options when using CUDA
from tensorflow.compat.v1 import ConfigProto, InteractiveSession
config = ConfigProto()
config.gpu_options.allow_growth = True
config.gpu_options.per_process_gpu_memory_fraction = 0.6
sess = InteractiveSession(config=config)

# set session variables
BATCH_SIZE = 32
IMG_HEIGHT = 224
IMG_WIDTH = 224
EPOCHS = 10

# Print the version of Tensorflow installed
print(tf.__version__)
```

Figure 50 : standard Python libraries included in all CNN programs

This also set the parameters to utilise the GPU to perform the training. Also, the version of Tensorflow was printed to confirm that all models were run using the same version.

The number of Epochs was set to 10 which was increased to 50 for the best performing models.
The working directory and the input folders were also set at this stage. The folders path was changed to correspond to each dataset., see Figure 51 With the following order:

1 – SDSS Fits Plots
2 – RGB from FITS Filters
3 – STScI JPG Images
4 – SDSS Jpg Images

```
# Set parameters for training later
os.chdir('d:\\project\\v2')
train_path='d:\\project\\v2\\images\\train' #path
test_path='d:\\project\\v2\\images\\test' #path
```

Figure 51 : set working directory and path to test / train images

By changing the os.chdir, train_path and test_path variables, it is easy to repoint the models at new datasets and ensure that the model processed the required images.

8.2 Base Model and modification to it

Each model was imported using the following code – the name of each model was changes and by changing the variable "model_name" to the relevant model name. This was used to output the models name as part of the final trained model with weights.
The base model was imported, the final layer was not. The models were then set to untraonable to locl the already trained layers – these could already extract features from images. 5 additional layers were added to the models and set to be trainable. As per Figure 51

```
model_name = "MobileNet"
model_source = "SDSS-RGB"
# select base model without the output layer
# include the IKmageNet weights

base_model=MobileNet(weights='imagenet',include_top=False)
#imports the mobilenet model and discards the last 1000 neuron layer.
for i,layer in enumerate(base_model.layers):
  print(i,layer.name)

# lock the model to leverage the tained filters and weights of base model
base_model.trainable = False

# List our the new layers to be added
ADD1=GlobalAveragePooling2D()
ADD2=Dense(1024,activation='relu')
#we add dense layers so that the model can learn more complex functions and classify for better results.
ADD3=Dense(1024,activation='relu') #dense layer 2
ADD4=Dense(512,activation='relu') #dense layer 3
prediction_layer=Dense(3,activation='softmax') #final layer with softmax activation

# create a new model with the base model and the addition of new layers
model = tf.keras.Sequential([
  base_model,
  ADD1,
  ADD2,
  ADD3,
  ADD4,
  prediction_layer
])

# confirm the new model structure
for i,layer in enumerate(model.layers):
  print(i,layer.name)

# Print the structure of the new model
print(" Model structure  ")
model.summary()
```

Figure 52 : Import base models and add new layers.

The last layer of the pre-trained model was not imported and this version of the model was set to be untrainable. This then allowed for the addition of 5 more layers to be added to the model and these layers were allowed to be adjusted in line with the models loss function and training later. The layers were displayed and the model summery was also displayed.

8.3 Identification of Data for training and testing

The images were imported and split into training and testing data. The functions used allowed the import based on the directory structure for each image set. An example of the folder structure for one set of test images is below in .

Figure 53 : Sample folder structure for all datasets

Using the following code, the images were imported, and the model compiled. All models were compiled with the same optimiser function.

```
train_datagen=ImageDataGenerator(preprocessing_function=preprocess_input) #included in our dependencies
test_datagen=ImageDataGenerator(preprocessing_function=preprocess_input) #included in our dependencies

train_generator=train_datagen.flow_from_directory(train_path,
                                                  target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                  color_mode='rgb',
                                                  batch_size=BATCH_SIZE,
                                                  class_mode='categorical',
                                                  seed=None,
                                                  shuffle=True)

val_generator=train_datagen.flow_from_directory(test_path,
                                                target_size=(IMG_HEIGHT, IMG_WIDTH),
                                                color_mode='rgb',
                                                batch_size=BATCH_SIZE,
                                                class_mode='categorical',
                                                seed=None,
                                                shuffle=True)

# Details of the Classes Found for training
data_dir = pathlib.Path(train_path)
image_count = len(list(data_dir.glob('*/*.png')))
image_count
CLASS_NAMES = np.array([item.name for item in data_dir.glob('*') if item.name != "LICENSE.txt"])
CLASS_NAMES


# Compile model
model.compile(optimizer='Adam',loss='categorical_crossentropy',metrics=['accuracy'])
# Adam optimizer
# loss function will be categorical cross entropy
# evaluation metric will be accuracy
```

Figure 54 : code to select images and compile models

The models were run, using the model.fit() function

```
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir=LOG_DIR)

step_size_train=train_generator.n//train_generator.batch_size

print(datetime.datetime.now().strftime("%Y%m%d-%H:%M"))

# ensure that the training is pushed to the GPU
# this improves performance using specialised hardware
# and the processing power ofthe GPU
with tf.device('/gpu:0'):
    history = model.fit(
        train_generator,
        steps_per_epoch=step_size_train,
        epochs=EPOCHS,
        validation_data=val_generator,
        callbacks=[tensorboard_callback]
        )

print(datetime.datetime.now().strftime("%Y%m%d-%H:%M"))
InteractiveSession.close(sess)
```

Figure 55 : code to make use of GPU for running models

Once the models had completed their training and testing the model plotted out its results and also saved the model to disk as per Figure 56:

```
# Print the Training and Validation Accuracy
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss=history.history['loss']
val_loss=history.history['val_loss']
epochs_range = range(EPOCHS)
plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label='Training Accuracy')
plt.plot(epochs_range, val_acc, label='Validation Accuracy')
plt.legend(loc='lower right')
plt.title('Training and Validation Accuracy')

# Plot the Traning and Validation Loss
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label='Training Loss')
plt.plot(epochs_range, val_loss, label='Validation Loss')
plt.legend(loc='upper right')
plt.title('Training and Validation Loss')
plt.show()


# serialize weights to HDF5 and save
model.save(model_name + "_" + model_source + "_model_"+ datetime.datetime.now().strftime("%Y%m%d-%H") +".h5")
print("Saved model to disk")
```

Figure 56 : code to save results and plots as well as resulting models

8.4 Results

The following are the detailed results which were not put in the Project Report. Each model was run, and the model's accuracy was used as a measure of its performance.

8.5 MobileNet Models Results

STScI JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.8365	0.8801
Epoch #2	0.8657	0.8798
Epoch #3	0.8798	0.8830
Epoch #4	0.8750	0.8827
Epoch #5	0.8826	0.8829
Epoch #6	0.8874	0.8864
Epoch #7	0.8918	0.8873
Epoch #8	0.8955	0.8874
Epoch #9	0.8980	0.8863
Epoch #10	0.9002	0.8715

SDSS JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.8055	0.8437
Epoch #2	0.8311	0.8197
Epoch #3	0.8397	0.8440
Epoch #4	0.8417	0.8380
Epoch #5	0.8466	0.8583
Epoch #6	0.8455	0.8641
Epoch #7	0.8471	0.8382
Epoch #8	0.8507	0.8206
Epoch #9	0.8505	0.8549
Epoch #10	0.8527	0.8607

SDSS Filters Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.4281	0.3272
Epoch #2	0.4437	0.3496
Epoch #3	0.4512	0.3347
Epoch #4	0.4562	0.3502
Epoch #5	0.4575	0.3497
Epoch #6	0.4588	0.3430
Epoch #7	0.4621	0.3582
Epoch #8	0.4635	0.3483
Epoch #9	0.4658	0.3248
Epoch #10	0.4712	0.3223

SDSS RBG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.4184	0.4675
Epoch #2	0.4499	0.4892
Epoch #3	0.4532	0.4572
Epoch #4	0.4647	0.4631
Epoch #5	0.4680	0.4825
Epoch #6	0.4691	0.4630
Epoch #7	0.4703	0.5020
Epoch #8	0.4765	0.4914
Epoch #9	0.4818	0.5099
Epoch #10	0.4806	0.4822

8.6 Resnet50 Models Results

STScI JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.3703	0.3650
Epoch #2	0.3769	0.3650
Epoch #3	0.3769	0.3650
Epoch #4	0.3769	0.3650
Epoch #5	0.3769	0.3650
Epoch #6	0.3769	0.3650

Epoch #7	0.3769	0.3650
Epoch #8	0.3769	0.3650
Epoch #9	0.3769	0.3650
Epoch #10	0.3769	0.3650

Obvious that learning stopped very early in this model and no further improvements were made from epoch 2 onwards.

SDSS JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.7490	0.8460
Epoch #2	0.8258	0.8642
Epoch #3	0.8459	0.8644
Epoch #4	0.8566	0.8628
Epoch #5	0.8650	0.8774
Epoch #6	0.8745	0.8700
Epoch #7	0.8783	0.9015
Epoch #8	0.8802	0.9067
Epoch #9	0.8847	0.8751
Epoch #10	0.8838	0.8906

SDSS Filters Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.3959	0.3540
Epoch #2	0.3970	0.3514
Epoch #3	0.4023	0.3583
Epoch #4	0.4067	0.3577
Epoch #5	0.4240	0.3702
Epoch #6	0.4343	0.3497
Epoch #7	0.4448	0.3242
Epoch #8	0.4479	0.3594
Epoch #9	0.4508	0.3443
Epoch #10	0.4511	0.3579

SDSS RGB Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.3703	0.3650
Epoch #2	0.3769	0.3650
Epoch #3	0.3769	0.3650
Epoch #4	0.3769	0.3650
Epoch #5	0.3769	0.3650
Epoch #6	0.3769	0.3650
Epoch #7	0.3769	0.3650
Epoch #8	0.3769	0.3650
Epoch #9	0.3769	0.3650
Epoch #10	0.3769	0.3650

8.7 VGG16 Model Results

STScI JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.7389	0.7605
Epoch #2	0.8095	0.8046
Epoch #3	0.8189	0.8483
Epoch #4	0.8249	0.7390
Epoch #5	0.8307	0.8492
Epoch #6	0.8351	0.8536
Epoch #7	0.8337	0.8593
Epoch #8	0.8366	0.8579
Epoch #9	0.8416	0.8630
Epoch #10	0.8453	0.8393

SDSS JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.68771	0.7516
Epoch #2	0.74634	0.7665
Epoch #3	0.76061	0.7358
Epoch #4	0.76901	0.7781
Epoch #5	0.78214	0.7391
Epoch #6	0.78006	0.7807
Epoch #7	0.78214	0.7795
Epoch #8	0.78506	0.7893
Epoch #9	0.78616	0.7863
Epoch #10	0.78933	0.7951

SDSS Filter Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.4155	0.3219
Epoch #2	0.4336	0.3169
Epoch #3	0.4403	0.3343
Epoch #4	0.4381	0.3343
Epoch #5	0.4458	0.3516
Epoch #6	0.4458	0.3570
Epoch #7	0.4469	0.3584
Epoch #8	0.4481	0.3520
Epoch #9	0.4495	0.3575
Epoch #10	0.4503	0.3520

SDSS RGB Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.4052	0.4389
Epoch #2	0.4497	0.4577
Epoch #3	0.4500	0.4577
Epoch #4	0.4526	0.4730
Epoch #5	0.4503	0.5106
Epoch #6	0.4527	0.4965
Epoch #7	0.4569	0.5093
Epoch #8	0.4583	0.4679
Epoch #9	0.4562	0.5161
Epoch #10	0.4604	0.5235

8.8 Xception Model Results

STScI JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.7614	0.8227
Epoch #2	0.7961	0.8204
Epoch #3	0.8038	0.8184
Epoch #4	0.8104	0.8267
Epoch #5	0.8177	0.8418
Epoch #6	0.8215	0.8097
Epoch #7	0.8254	0.8461
Epoch #8	0.8289	0.8406
Epoch #9	0.8302	0.8434
Epoch #10	0.8316	0.8380

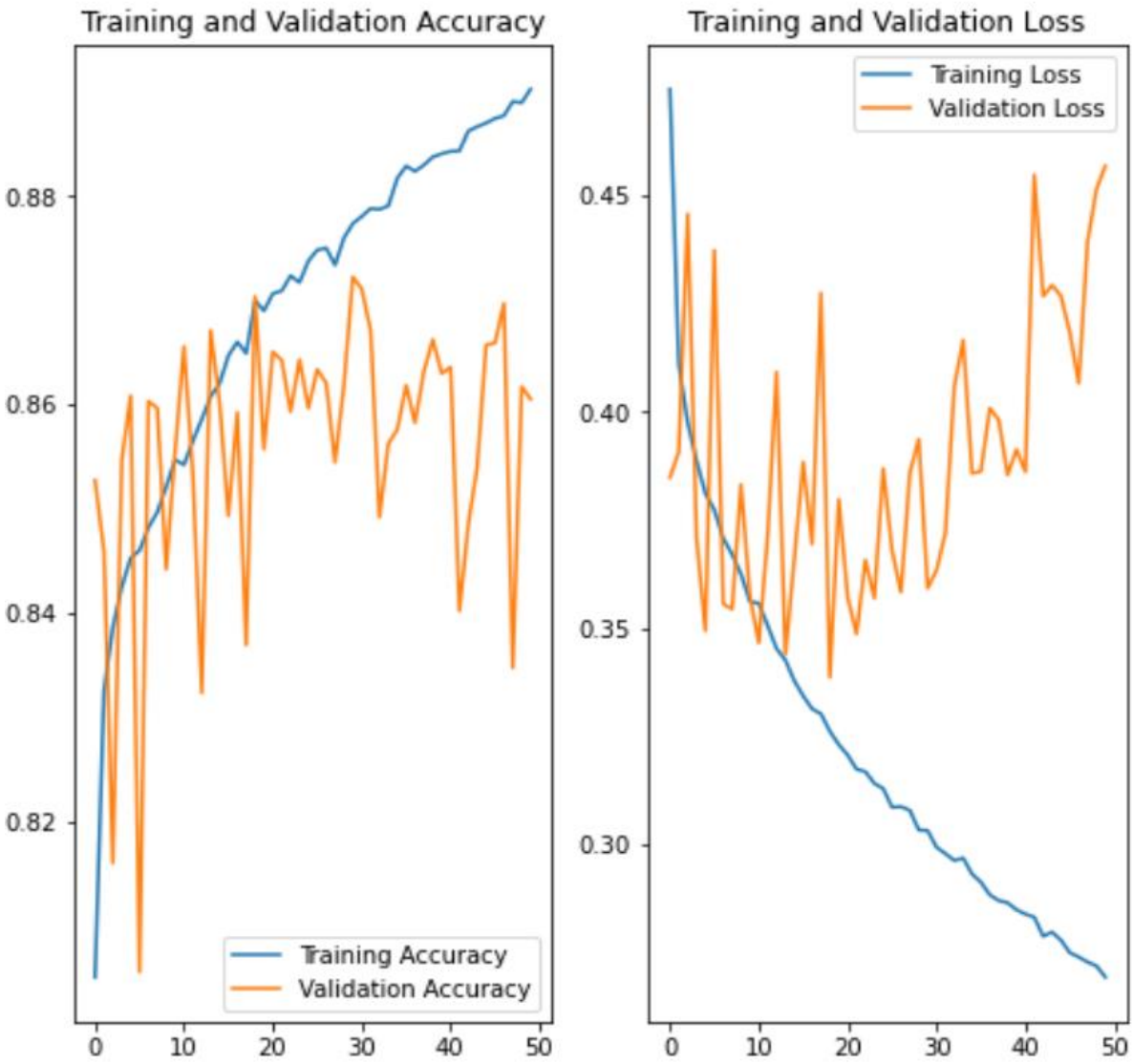
SDSS JPG Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.7732	0.7982
Epoch #2	0.8046	0.8140
Epoch #3	0.8154	0.8184
Epoch #4	0.8237	0.8145
Epoch #5	0.8285	0.8062
Epoch #6	0.8347	0.8247
Epoch #7	0.8391	0.8029
Epoch #8	0.8448	0.8072
Epoch #9	0.8474	0.8113
Epoch #10	0.8520	0.8222

SDSS Filter Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.4091	0.3823
Epoch #2	0.4277	0.4608
Epoch #3	0.4329	0.3996
Epoch #4	0.4369	0.4307
Epoch #5	0.4381	0.4077
Epoch #6	0.4396	0.4701
Epoch #7	0.4438	0.4588
Epoch #8	0.4465	0.4807
Epoch #9	0.4523	0.4560
Epoch #10	0.4532	0.4545

SDSS RGB Images	Training Epoch Results - Accuracy	Validation Epochs Results - Accuracy
Epoch #1	0.40928	0.4549
Epoch #2	0.42760	0.4656
Epoch #3	0.43486	0.4576
Epoch #4	0.43968	0.4151
Epoch #5	0.43834	0.4031
Epoch #6	0.44124	0.4556
Epoch #7	0.44552	0.4596

Epoch #8	0.44249	0.4598
Epoch #9	0.44806	0.4791
Epoch #10	0.45114	0.4656

8.9 MobileNet 50 Epoch with SDSS Jpg Images



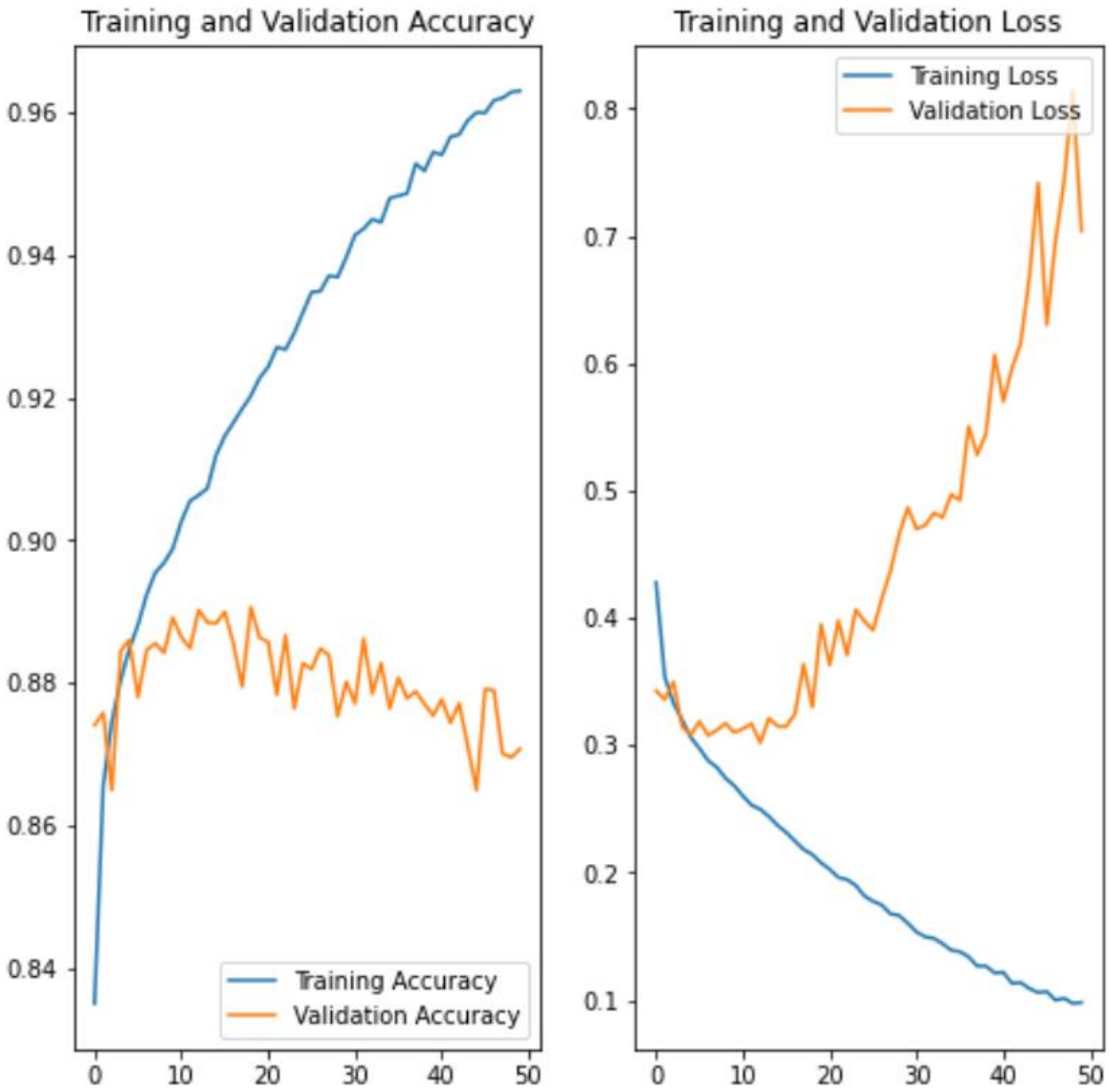
Epoch 1/50
1874/1874 [==============================] - 1866s 996ms/step - loss: 0.4744 - accuracy: 0.8050 - val_loss: 0.3848 - val_accuracy: 0.8527
Epoch 2/50
1874/1874 [==============================] - 175s 94ms/step - loss: 0.4111 - accuracy: 0.8325 - val_loss: 0.3907 - val_accuracy: 0.8459
Epoch 3/50
1874/1874 [==============================] - 176s 94ms/step - loss: 0.3975 - accuracy: 0.8386 - val_loss: 0.4456 - val_accuracy: 0.8160
Epoch 4/50

1874/1874 [==========] - 175s 93ms/step - loss: 0.3886 - accuracy: 0.8425 - val_loss: 0.3707 - val_accuracy: 0.8547
Epoch 5/50
1874/1874 [==========] - 173s 92ms/step - loss: 0.3811 - accuracy: 0.8453 - val_loss: 0.3494 - val_accuracy: 0.8609
Epoch 6/50
1874/1874 [==========] - 173s 92ms/step - loss: 0.3773 - accuracy: 0.8460 - val_loss: 0.4373 - val_accuracy: 0.8055
Epoch 7/50
1874/1874 [==========] - 173s 92ms/step - loss: 0.3708 - accuracy: 0.8482 - val_loss: 0.3556 - val_accuracy: 0.8603
Epoch 8/50
1874/1874 [==========] - 172s 92ms/step - loss: 0.3670 - accuracy: 0.8497 - val_loss: 0.3544 - val_accuracy: 0.8597
Epoch 9/50
1874/1874 [==========] - 411s 219ms/step - loss: 0.3625 - accuracy: 0.8521 - val_loss: 0.3832 - val_accuracy: 0.8442
Epoch 10/50
1874/1874 [==========] - 265s 141ms/step - loss: 0.3561 - accuracy: 0.8547 - val_loss: 0.3574 - val_accuracy: 0.8555
Epoch 11/50
1874/1874 [==========] - 174s 93ms/step - loss: 0.3557 - accuracy: 0.8542 - val_loss: 0.3466 - val_accuracy: 0.8656
Epoch 12/50
1874/1874 [==========] - 178s 95ms/step - loss: 0.3506 - accuracy: 0.8565 - val_loss: 0.3713 - val_accuracy: 0.8533
Epoch 13/50
1874/1874 [==========] - 173s 92ms/step - loss: 0.3453 - accuracy: 0.8586 - val_loss: 0.4091 - val_accuracy: 0.8323
Epoch 14/50
1874/1874 [==========] - 174s 93ms/step - loss: 0.3426 - accuracy: 0.8608 - val_loss: 0.3440 - val_accuracy: 0.8671
Epoch 15/50
1874/1874 [==========] - 173s 92ms/step - loss: 0.3378 - accuracy: 0.8620 - val_loss: 0.3660 - val_accuracy: 0.8603
Epoch 16/50
1874/1874 [==========] - 172s 92ms/step - loss: 0.3344 - accuracy: 0.8647 - val_loss: 0.3883 - val_accuracy: 0.8493
Epoch 17/50
1874/1874 [==========] - 176s 94ms/step - loss: 0.3314 - accuracy: 0.8660 - val_loss: 0.3695 - val_accuracy: 0.8593
Epoch 18/50
1874/1874 [==========] - 176s 94ms/step - loss: 0.3302 - accuracy: 0.8649 - val_loss: 0.4274 - val_accuracy: 0.8369
Epoch 19/50
1874/1874 [==========] - 173s 92ms/step - loss: 0.3262 - accuracy: 0.8700 - val_loss: 0.3387 - val_accuracy: 0.8704
Epoch 20/50
1874/1874 [==========] - 174s 93ms/step - loss: 0.3231 - accuracy: 0.8690 - val_loss: 0.3798 - val_accuracy: 0.8557
Epoch 21/50

1874/1874 [==============================] - 172s 92ms/step - loss: 0.3207 - accuracy: 0.8707 - val_loss: 0.3571 - val_accuracy: 0.8651
Epoch 22/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.3175 - accuracy: 0.8709 - val_loss: 0.3487 - val_accuracy: 0.8643
Epoch 23/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.3169 - accuracy: 0.8724 - val_loss: 0.3658 - val_accuracy: 0.8593
Epoch 24/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.3142 - accuracy: 0.8717 - val_loss: 0.3569 - val_accuracy: 0.8643
Epoch 25/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.3130 - accuracy: 0.8738 - val_loss: 0.3868 - val_accuracy: 0.8597
Epoch 26/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.3086 - accuracy: 0.8749 - val_loss: 0.3679 - val_accuracy: 0.8634
Epoch 27/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.3088 - accuracy: 0.8751 - val_loss: 0.3583 - val_accuracy: 0.8621
Epoch 28/50
1874/1874 [==============================] - 174s 93ms/step - loss: 0.3080 - accuracy: 0.8734 - val_loss: 0.3858 - val_accuracy: 0.8545
Epoch 29/50
1874/1874 [==============================] - 173s 93ms/step - loss: 0.3034 - accuracy: 0.8760 - val_loss: 0.3936 - val_accuracy: 0.8619
Epoch 30/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.3033 - accuracy: 0.8774 - val_loss: 0.3592 - val_accuracy: 0.8723
Epoch 31/50
1874/1874 [==============================] - 174s 93ms/step - loss: 0.2994 - accuracy: 0.8781 - val_loss: 0.3633 - val_accuracy: 0.8711
Epoch 32/50
1874/1874 [==============================] - 175s 93ms/step - loss: 0.2980 - accuracy: 0.8788 - val_loss: 0.3720 - val_accuracy: 0.8671
Epoch 33/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2964 - accuracy: 0.8788 - val_loss: 0.4058 - val_accuracy: 0.8492
Epoch 34/50
1874/1874 [==============================] - 174s 93ms/step - loss: 0.2969 - accuracy: 0.8791 - val_loss: 0.4165 - val_accuracy: 0.8563
Epoch 35/50
1874/1874 [==============================] - 174s 93ms/step - loss: 0.2932 - accuracy: 0.8817 - val_loss: 0.3857 - val_accuracy: 0.8576
Epoch 36/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2913 - accuracy: 0.8829 - val_loss: 0.3862 - val_accuracy: 0.8619
Epoch 37/50
1874/1874 [==============================] - 173s 93ms/step - loss: 0.2884 - accuracy: 0.8824 - val_loss: 0.4008 - val_accuracy: 0.8583
Epoch 38/50

1874/1874 [==============================] - 173s 93ms/step - loss: 0.2871 - accuracy: 0.8830 - val_loss: 0.3981 - val_accuracy: 0.8632
Epoch 39/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2866 - accuracy: 0.8838 - val_loss: 0.3854 - val_accuracy: 0.8663
Epoch 40/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2850 - accuracy: 0.8841 - val_loss: 0.3913 - val_accuracy: 0.8630
Epoch 41/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.2840 - accuracy: 0.8843 - val_loss: 0.3861 - val_accuracy: 0.8636
Epoch 42/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2833 - accuracy: 0.8844 - val_loss: 0.4546 - val_accuracy: 0.8402
Epoch 43/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2789 - accuracy: 0.8863 - val_loss: 0.4266 - val_accuracy: 0.8485
Epoch 44/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.2798 - accuracy: 0.8867 - val_loss: 0.4292 - val_accuracy: 0.8541
Epoch 45/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2780 - accuracy: 0.8870 - val_loss: 0.4267 - val_accuracy: 0.8657
Epoch 46/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.2751 - accuracy: 0.8875 - val_loss: 0.4183 - val_accuracy: 0.8659
Epoch 47/50
1874/1874 [==============================] - 173s 92ms/step - loss: 0.2741 - accuracy: 0.8877 - val_loss: 0.4065 - val_accuracy: 0.8697
Epoch 48/50
1874/1874 [==============================] - 172s 92ms/step - loss: 0.2729 - accuracy: 0.8891 - val_loss: 0.4392 - val_accuracy: 0.8347
Epoch 49/50
1874/1874 [==============================] - 175s 94ms/step - loss: 0.2720 - accuracy: 0.8890 - val_loss: 0.4514 - val_accuracy: 0.8617
Epoch 50/50
1874/1874 [==============================] - 176s 94ms/step - loss: 0.2694 - accuracy: 0.8903 - val_loss: 0.4567 - val_accuracy: 0.8605

8.10 MobileNet SDSS JPG Images - 50 Epochs



Total params: 5,854,403
Trainable params: 2,625,539
Non-trainable params: 3,228,864

Found 60000 images belonging to 3 classes.
Found 15000 images belonging to 3 classes.
20200703-12:35

Epoch 1/50
1875/1875 [==============================] - 1998s 1s/step - loss: 0.4280 - accuracy: 0.8350 - val_loss: 0.3429 - val_accuracy: 0.8741
Epoch 2/50
1875/1875 [==============================] - 197s 105ms/step - loss: 0.3537 - accuracy: 0.8656 - val_loss: 0.3361 - val_accuracy: 0.8758
Epoch 3/50
1875/1875 [==============================] - 190s 101ms/step - loss: 0.3331 - accuracy: 0.8746 - val_loss: 0.3498 - val_accuracy: 0.8649
Epoch 4/50

1875/1875 [==========] - 188s 100ms/step - loss: 0.3199 - accuracy: 0.8803 - val_loss: 0.3148 - val_accuracy: 0.8845
Epoch 5/50
1875/1875 [==========] - 183s 98ms/step - loss: 0.3063 - accuracy: 0.8847 - val_loss: 0.3082 - val_accuracy: 0.8860
Epoch 6/50
1875/1875 [==========] - 184s 98ms/step - loss: 0.2975 - accuracy: 0.8881 - val_loss: 0.3188 - val_accuracy: 0.8780
Epoch 7/50
1875/1875 [==========] - 184s 98ms/step - loss: 0.2880 - accuracy: 0.8924 - val_loss: 0.3078 - val_accuracy: 0.8846
Epoch 8/50
1875/1875 [==========] - 184s 98ms/step - loss: 0.2828 - accuracy: 0.8955 - val_loss: 0.3119 - val_accuracy: 0.8855
Epoch 9/50
1875/1875 [==========] - 184s 98ms/step - loss: 0.2741 - accuracy: 0.8968 - val_loss: 0.3172 - val_accuracy: 0.8842
Epoch 10/50
1875/1875 [==========] - 184s 98ms/step - loss: 0.2683 - accuracy: 0.8988 - val_loss: 0.3101 - val_accuracy: 0.8891
Epoch 11/50
1875/1875 [==========] - 189s 101ms/step - loss: 0.2602 - accuracy: 0.9026 - val_loss: 0.3129 - val_accuracy: 0.8864
Epoch 12/50
1875/1875 [==========] - 189s 101ms/step - loss: 0.2531 - accuracy: 0.9055 - val_loss: 0.3170 - val_accuracy: 0.8849
Epoch 13/50
1875/1875 [==========] - 193s 103ms/step - loss: 0.2498 - accuracy: 0.9063 - val_loss: 0.3021 - val_accuracy: 0.8902
Epoch 14/50
1875/1875 [==========] - 192s 102ms/step - loss: 0.2441 - accuracy: 0.9073 - val_loss: 0.3213 - val_accuracy: 0.8885
Epoch 15/50
1875/1875 [==========] - 191s 102ms/step - loss: 0.2371 - accuracy: 0.9119 - val_loss: 0.3150 - val_accuracy: 0.8883
Epoch 16/50
1875/1875 [==========] - 191s 102ms/step - loss: 0.2316 - accuracy: 0.9146 - val_loss: 0.3149 - val_accuracy: 0.8899
Epoch 17/50
1875/1875 [==========] - 192s 103ms/step - loss: 0.2248 - accuracy: 0.9165 - val_loss: 0.3247 - val_accuracy: 0.8855
Epoch 18/50
1875/1875 [==========] - 192s 102ms/step - loss: 0.2182 - accuracy: 0.9184 - val_loss: 0.3637 - val_accuracy: 0.8795
Epoch 19/50
1875/1875 [==========] - 192s 103ms/step - loss: 0.2142 - accuracy: 0.9202 - val_loss: 0.3301 - val_accuracy: 0.8906
Epoch 20/50
1875/1875 [==========] - 195s 104ms/step - loss: 0.2077 - accuracy: 0.9227 - val_loss: 0.3949 - val_accuracy: 0.8863
Epoch 21/50

1875/1875 [==============================] - 196s 105ms/step - loss: 0.2026 - accuracy: 0.9243 - val_loss: 0.3630 - val_accuracy: 0.8857
Epoch 22/50
1875/1875 [==============================] - 190s 101ms/step - loss: 0.1962 - accuracy: 0.9270 - val_loss: 0.3981 - val_accuracy: 0.8783
Epoch 23/50
1875/1875 [==============================] - 193s 103ms/step - loss: 0.1946 - accuracy: 0.9267 - val_loss: 0.3710 - val_accuracy: 0.8867
Epoch 24/50
1875/1875 [==============================] - 194s 104ms/step - loss: 0.1900 - accuracy: 0.9291 - val_loss: 0.4065 - val_accuracy: 0.8764
Epoch 25/50
1875/1875 [==============================] - 194s 103ms/step - loss: 0.1817 - accuracy: 0.9319 - val_loss: 0.3975 - val_accuracy: 0.8827
Epoch 26/50
1875/1875 [==============================] - 194s 103ms/step - loss: 0.1775 - accuracy: 0.9348 - val_loss: 0.3904 - val_accuracy: 0.8819
Epoch 27/50
1875/1875 [==============================] - 193s 103ms/step - loss: 0.1746 - accuracy: 0.9349 - val_loss: 0.4152 - val_accuracy: 0.8848
Epoch 28/50
1875/1875 [==============================] - 194s 104ms/step - loss: 0.1676 - accuracy: 0.9371 - val_loss: 0.4369 - val_accuracy: 0.8839
Epoch 29/50
1875/1875 [==============================] - 194s 104ms/step - loss: 0.1664 - accuracy: 0.9369 - val_loss: 0.4656 - val_accuracy: 0.8753
Epoch 30/50
1875/1875 [==============================] - 195s 104ms/step - loss: 0.1602 - accuracy: 0.9397 - val_loss: 0.4868 - val_accuracy: 0.8801
Epoch 31/50
1875/1875 [==============================] - 194s 103ms/step - loss: 0.1534 - accuracy: 0.9428 - val_loss: 0.4699 - val_accuracy: 0.8771
Epoch 32/50
1875/1875 [==============================] - 192s 102ms/step - loss: 0.1496 - accuracy: 0.9437 - val_loss: 0.4728 - val_accuracy: 0.8862
Epoch 33/50
1875/1875 [==============================] - 191s 102ms/step - loss: 0.1484 - accuracy: 0.9450 - val_loss: 0.4823 - val_accuracy: 0.8785
Epoch 34/50
1875/1875 [==============================] - 193s 103ms/step - loss: 0.1443 - accuracy: 0.9446 - val_loss: 0.4790 - val_accuracy: 0.8827
Epoch 35/50
1875/1875 [==============================] - 193s 103ms/step - loss: 0.1391 - accuracy: 0.9480 - val_loss: 0.4969 - val_accuracy: 0.8764
Epoch 36/50
1875/1875 [==============================] - 196s 105ms/step - loss: 0.1379 - accuracy: 0.9483 - val_loss: 0.4924 - val_accuracy: 0.8807
Epoch 37/50
1875/1875 [==============================] - 197s 105ms/step - loss: 0.1339 - accuracy: 0.9486 - val_loss: 0.5504 - val_accuracy: 0.8778
Epoch 38/50

1875/1875 [==============================] - 198s 106ms/step - loss: 0.1267 - accuracy: 0.9528 - val_loss: 0.5283 - val_accuracy: 0.8788
Epoch 39/50
1875/1875 [==============================] - 197s 105ms/step - loss: 0.1266 - accuracy: 0.9518 - val_loss: 0.5447 - val_accuracy: 0.8771
Epoch 40/50
1875/1875 [==============================] - 196s 105ms/step - loss: 0.1212 - accuracy: 0.9544 - val_loss: 0.6066 - val_accuracy: 0.8754
Epoch 41/50
1875/1875 [==============================] - 187s 100ms/step - loss: 0.1217 - accuracy: 0.9540 - val_loss: 0.5701 - val_accuracy: 0.8777
Epoch 42/50
1875/1875 [==============================] - 190s 101ms/step - loss: 0.1131 - accuracy: 0.9566 - val_loss: 0.5961 - val_accuracy: 0.8744
Epoch 43/50
1875/1875 [==============================] - 186s 99ms/step - loss: 0.1136 - accuracy: 0.9569 - val_loss: 0.6159 - val_accuracy: 0.8771
Epoch 44/50
1875/1875 [==============================] - 187s 100ms/step - loss: 0.1090 - accuracy: 0.9589 - val_loss: 0.6672 - val_accuracy: 0.8711
Epoch 45/50
1875/1875 [==============================] - 186s 99ms/step - loss: 0.1058 - accuracy: 0.9600 - val_loss: 0.7415 - val_accuracy: 0.8649
Epoch 46/50
1875/1875 [==============================] - 190s 101ms/step - loss: 0.1067 - accuracy: 0.9599 - val_loss: 0.6303 - val_accuracy: 0.8791
Epoch 47/50
1875/1875 [==============================] - 192s 103ms/step - loss: 0.0998 - accuracy: 0.9617 - val_loss: 0.6955 - val_accuracy: 0.8789
Epoch 48/50
1875/1875 [==============================] - 192s 103ms/step - loss: 0.1014 - accuracy: 0.9620 - val_loss: 0.7429 - val_accuracy: 0.8701
Epoch 49/50
1875/1875 [==============================] - 188s 100ms/step - loss: 0.0973 - accuracy: 0.9628 - val_loss: 0.8140 - val_accuracy: 0.8695
Epoch 50/50
1875/1875 [==============================] - 187s 100ms/step - loss: 0.0981 - accuracy: 0.9630 - val_loss: 0.7039 - val_accuracy: 0.8707
20200703-15:45

9 ICT Files and purpose

This section lists the files which were submitted along with the configuration manual and which are required to run the application. The purpose and manes of all files are below.

To process downloaded images FITS Images the following files are required:

ProcessGalaxiesFITS.py
ProcessquasarsFITS.py
ProcessStartFITS.py

These files use the working directory as set in the parameters section and crawl down the folders as per the parameters set.

To download images from STScI use the following file

GetImagesfromKepler.py

This file takes an input.txt file which lists all the images required in the tab delimited file with the format of RA DEC Classification, one per line.

To rename files from STScI after downloading

RemaneFilesG.py for Galaxy folder
RemaneFilesQ.py for the Quasar folder
RemaneFilesS.py for the Star folder

To combine files to make RGB files

RGBImages.py

Training for models - set working directory first

Training_for_MobileNet.py
Training_for_ResNet.py
Training_for_VG16.py
Training_for_Xception.py

Images Catalogue and URL are found in

Skysaver_SQL4_6_2020_1_30_47 PM (version4).xls

From this file take the required URL's and put in txt file.

For SDSS use "wget -I input.txt" syntax to get FITS files from site, where input.txt is a test file with the url for the FITS files to be downloaded. FITS files range in size from 2.5MB to 3.5MB.

Bibliography

A, V., & K. Lenc, K. (2015). Matconvnet: Convolutional neural networksfor matlab. Proceedings of the 23rd ACM international conferenceon Multimedia.

Azevedo, A. I. (2008). KDD, SEMMA and CRISP-DM: a parallel overview. IADIS European Conf. Data Mining.

Azhar, K., Murtaza, F., Yousaf , H., & Habib, H. a. (2016). Computer vision based detection and localization of potholes in asphalt pavement images. Vancouver: IEEE.

Ball, N., Brunner, R., Myers, A., & Tcheng, D. (2006). Robust Machine Learning Applied to Astronomical Data Sets. I. Star-Galaxy Classification of the Sloan Digital Sky Survey DR3 Using Decision Trees. *The Astrophysical Journal, 605*(1), 497-509.

Barik, D., & Mondal, M. (2010). Object Identification For Computer Vision using Image Segmentation. IEEE Xplore.
Bhattacharyya, D. K. (2006). *Research Methodology* (2nd ed.). New Delhi: Excel Books.
Brownlee, J. (2019). *A Gentle Introduction to the Rectified Linear Unit (ReLU)*. Retrieved 06 2020, from https://machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
Chapman, P. C. (2000). *CRISP-DM 1.0 Step-by-step data mining guide.* SPSS.
Clark-Carter, D. (2010). *The Complete Student's Comapnion* (3 ed.). Hove, East Susses: Psychology Press.
Diederik P. Kingma, J. B. (2015). Adam: A Method for Stochastic Optimization. San Diego: Published as a conference paper at the 3rd International Conference for Learning Representations.
du Buisson, L., Sivanandam, N., Bassett, B. A., & Smith, M. (2015). Machine learning classification of SDSS transient survey images. *Monthly Notices of the Royal Astronomical Society, 454*(2), 2026-2038.
Eisenstein D.J., W. D. (2011). SDSS-III: MASSIVE SPECTROSCOPIC SURVEYS OF THE DISTANT UNIVERSE, THE MILKY WAY, AND EXTRA-SOLAR PLANETARY SYSTEMS. *The Astronomical Journal, 142*(3).
Flemsteed, J. (1725). *Catalogus Britannicus.* London.
González, R. E., Muñoz, P. M., & Hernández, C. (2018). Galaxy detection and identification using deep learning and data augmentation. *Astronomy and Computing, 25*, 103-109.
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Back-Propagation and Other Differentiation Algorithms.* MIT Press.
Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning (Adaptive Computation and Machine Learning series).* Cambridge, Ma: The MIT Press.
Graff, P., Feroz, M., Hobson, M., & Lasenby, A. (2014). SkyNet: an efficient and robust neural network training tool for machine learning in astronomy. *Monthly Notices of the Royal Astronomical Society, 441*(2), 1741-1759.
He, K. Z. (2016). Deep residual learning for image recognition. IEEE.
Hubel, D., & Wiesel, T. (1962). Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology, 160*(1), 106-154.
Jansky, C. M. (1958). The Discovery and Identification by Karl Guthe Jansky of Electromagnetic Radiation of Extraterrestrial Origin in the Radio Spectrum. *Proceedings of the IRE, 46*(1), 13 - 15.
Kheirdastan, S., & Bazarghan, M. (2016). SDSS-DR12 bulk stellar spectral classification: Artificial neural networks approach. *Astrophysics and Space Science, 361*(9), 304.
Kingma, D., & Lei Ba, J. (2015). ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION. San Diego: International Conference on Learning Representations.
Kremer, J., Stensbo-Smidt, K., Gieseke, F., Pedersen, K., & Igel, C. (2017). Big Universe, Big Data: Machine Learning and Image Analysis for Astronomy. *IEEE Intelligent Systems, 32*(2), 16-22.
Large Synoptic Survey Telescope. (n.d.). *Legacy Survey of Space and Time*. Retrieved March 03, 2020, from https://www.lsst.org/
LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation, 1*(4), 541–551.
Li, X., & Shi, Y. (2018). Computer Vision Imaging Based on Artificial Intelligence. IEEE Xplore.
Lintott, C. J., Schawinski, K., Slosar, A., Land, L., & Steven Bamford, D. T. (2008). Galaxy Zoo: morphologies derived from visual inspection of galaxies from the Sloan Digital Sky Survey. *Monthly Notices of the Royal Astronomical Society, 389*(3), 1179-1189.

M. Abd Elfattah, N. E.-S. (2014). Galaxies image classification using empirical mode decomposition and machine learning techniques. Ciaro: 2014.
Ma, Y., Dang, J., & Li, W. (2014). Research on deep neural network's hidden layers in phoneme recognition. Singapore: IEEE.
Martín Abadi, P. B. (2016, Nov). TensorFlow: A System for Large-Scale Machine Learning. *12th {USENIX} Symposium on Operating Systems Design and Implementation*, pp. 265--283.
Mattmann, C., & Zhang, Z. (2019). Deep Facial Recognition using Tensorflow. Denvor: IEEE.
Murray, R. (2006). *How to write a Thesis* (2nd ed.). Maidenhead: Open UIniversity Press.
Nkwentsha, X., Hounkanrin, A., & Nicolls, F. (2020). Automatic classification of medical X-ray images with convolutional neural networks. IEEE.
P.H.Barchiab, Carvalhocd, R., R.R.Rosaa, R.A.Sauttera, M.Soares-Santosb, B.A.D.Marquese, . . . T.C.Mourag. (2019). Machine and Deep Learning applied to galaxy morphology - A comparative study. *Astronomy and Computing, 30*(100334).
Pasquet-Itam, J., & Pasquet, J. (2018). Deep learning approach for classifying, detecting and predicting photometric redshifts of quasars in the Sloan Digital Sky Survey stripe 82. *Astronomy & Astrophysics, 611*, A97.
Pelka O, N. F. (2018, 11 12). Annotation of enhanced radiographs for medical image retrieval with deep convolutional neural networks. *PLoS One*.
Philip Graff, F. F. (2014). SkyNet: an efficient and robust neural network training tool for machine learning in astronomy. *Monthly Notices of the Royal Astronomical Society, 441*(2), 1741–1759.
Post Grad Programme in Data Analytics. (2019). Intro to Data Mining - slide 25. Dublin: National College of Ireland.
Reber, G. &. (1947). Radio-frequency investigations of astronomical interest. *The Observatory, 67*, 15-26.
Romina Ahumada, C. A.-R. (2019, Dec 19). *The Sixteenth Data Release of the Sloan Digital Sky Surveys: First Release from the APOGEE-2 Southern Survey and Full Release of eBOSS Spectra.* Retrieved from Cornell University: https://arxiv.org/abs/1912.02905
Ruder, S. (2016). *An overview of gradient descent optimization algorithms.* NUI Galway, Insight Centre for Data Analytics. Galway: Aylien Ltd, Dublin. Retrieved from https://arxiv.org/pdf/1609.04747.pdf
Simard, P. Y., Steinkraus, D., & Platt, J. C. (2013). Best Practices for Convolutional Neural Networks. Redmond: Microsoft Research.
Szegedy, C. L. (2016). Going deeper with convolutions. In: The IEEE Conference on Computer Vision and Pattern Recognition. IEEE.
US Department of Energy Office of Science. (2018). *The Dark Energy Spectroscopic Instrument (DESI)*. Retrieved March 02, 2020, from https://www.desi.lbl.gov/
Usama Fayyad, Gregory , P.-S., & Padhraic , S. (1996). The KDD process for extracting useful knowledge from volumes of data. *Communications of the ACM*(11), 27-34.
Wikepedia. (n.d.). *Softmax function*. Retrieved 06 2020, from https://en.wikipedia.org/wiki/Softmax_function
Wray, J., & Gunn, J. (2008). A New Technique for Galaxy Photometric Redshifts in the Sloan Digital Sky Survey. *The Astrophysical Journal, 678*(1), 144-153.
Yang, X., Mo, H., Bosch, F., Pasquali, A., Li, C., & Barden, M. (2007). Galaxy Groups in the SDSS DR4. I. The Catalog and Basic Properties. *The Astrophysical Journal, 671*(1), 153-170.
Yongheng Zhao, Y. Z. (2008). Comparison of decision tree methods. *Advances in Space Research, 41*(12), 1955-1959.

Zare, M. R., Mueen, A., Awedh, M., & Seng, W. C. (2013, July 5). Automatic classification of medical X-ray images: hybrid generative-discriminative approach. *The Institution of Engineering and Technology*, pp. 523-532.