***Faculty of Science and Technology***

**Assignment Coversheet**

|  |  |
| --- | --- |
| **Student ID number &**  **Student Name** | U3261011  Advaith Ramakrishnan |
| **Unit name** | Software Technology 1 |
| **Unit number** | 4483 |
| **Unit Tutor** | Pranav Gupta |
| **Assignment name** | ST1 Capstone Project |
| **Due date** | 29th October 2023 |
| **Date submitted** | 28/10/2023 |

**You must keep a photocopy or electronic copy of your assignment.**

**Student declaration**

I certify that the attached assignment is my own work. Material drawn from other sources has been appropriately and fully acknowledged as to author/creator, source and other bibliographic details.

**Signature of student: \_\_AR\_\_\_\_\_\_\_\_\_**

**Date: \_\_\_\_\_28/10/2023\_\_\_\_\_\_\_**

Contents

[Introduction 3](#_Toc149429593)

[Dataset Description 3](#_Toc149429594)

[Methodology 4](#_Toc149429595)

[Algorithm Design Stage 4](#_Toc149429596)

[Exploratory Data Analysis 4](#_Toc149429597)

[Predictive Data Analysis 14](#_Toc149429599)

[Software Implementation & Deployment Stage 20](#_Toc149429600)

[Streamlit Image Classification Learner 20](#_Toc149429601)

[Home Page 23](#_Toc149429602)

[Image Classification Learner 23](#_Toc149429603)

[About Page 23](#_Toc149429604)

[Deployment Screenshots 24](#_Toc149429605)

[Output 24](#_Toc149429606)

[Git Hub Repository 24](#_Toc149429607)

[Journal 25](#_Toc149429608)

# Introduction

This report describes the details of Python Capstone Project for ST1 unit within the scope of the project requirements provided in the assignment handout [1]. I have decided to work on an Astronomy Image Classification Dataset that can be found on Kaggle and can be accessed publicly.

In this report I will be approaching my dataset through data preprocessing, visualization, and analysis techniques.

This capstone project report will discuss the findings found in this dataset through the use of Exploratory Data Analysis (using Pandas, plt etc), Predictive Data Analysis (Keras, Teachable Machine) and Implementation as a desktop app (Streamlit/Tkinter).

The details of the dataset used will be described in Section 2, the methodology will be presented in Section 3, the details of the code development, performance evaluation and deployment are discussed in Section 4. The report will end with conclusions and recommendations in Section 5.

# Dataset Description

Astronomy Image Dataset

Link - <https://www.kaggle.com/datasets/abhikalpsrivastava15/space-images-category/data>

* Type of Data – 1107 images
* 6 classes
  + Constellations
  + Cosmos Space
  + Galaxies
  + Nebula
  + Planet
  + Stars
* Image format – JPG files
* How data was acquired – Google Images
* Applicability – Suitable for exploring space images and performing analysis on different types of space images.

# Methodology

The methodology used for developing the software platform involves 3 stages as outlined below:

1. Stage 1: Exploratory Data Analysis for astronomy images from the dataset (Google Colab)
2. Stage 2: Predictive Data Analysis using machine learning platforms (Keras, teachable machine with Google)/Google Colab.
3. Stage 3: Implementation as Desktop app (Streamlit)/Pycharm

## Algorithm Design Stage

1. Collect Database from Kaggle
2. Preprocess if required.
3. Mount database in google colab
4. Basic Analysis using matplotlib, cv, NumPy etc.
5. Building Model
6. Make Implementation from predictive model.

### 

### Exploratory Data Analysis

The first phase of the assignment was to understand the dataset a bit more through visualizations and basic exploratory data analysis. Google Colab was the chosen environment for EDA. Below were the steps carried out to perform EDA:

* Insert Database on google drive.
* Mapping drive to colab notebook
* Importing numerous libraries to perform analysis.

**Mount Function and Libraries**

|  |
| --- |
| **from** **google.colab** **import** drive  drive.mount('/content/drive')  **import** **numpy** **as** **np**  **from** **tensorflow.keras.optimizers** **import** RMSprop  **from** **tensorflow.keras.preprocessing** **import** image  **from** **tensorflow.keras.preprocessing.image** **import** ImageDataGenerator  **import** **numpy** **as** **np**  %matplotlib inline  **from** **collections** **import** defaultdict  **import** **matplotlib.pyplot** **as** **plt**  **from** **IPython.display** **import** Image  **import** **tensorflow** **as** **tf**  **import** **cv2**  **import** **os**  **from** **PIL** **import** Image  **import** **skimage**  **import** **skimage.color** **as** **skic**  **import** **skimage.filters** **as** **skif**  **import** **skimage.data** **as** **skid**  **import** **skimage.util** **as** **sku** |

Getting details of image through cv2.imread

|  |
| --- |
| cv2.imread('/content/drive/MyDrive/Capstone\_Databse/space images/galaxies - Google Search/1.jpg') |

A screenshot of a computer

Description automatically generated

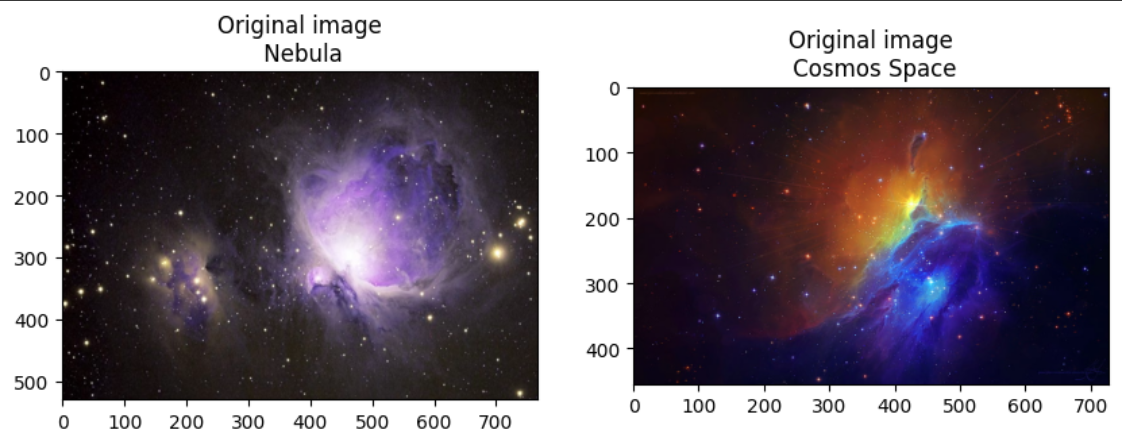
Basic Analysis of how many images per class

|  |
| --- |
| data\_dir = '/content/drive/MyDrive/Capstone\_Databse/Data/train'  class\_counts = defaultdict(int)  **for** i **in** os.listdir(data\_dir):  class\_dir = os.path.join(data\_dir, i)  **if** os.path.isdir(class\_dir):  class\_counts[i] = len(os.listdir(class\_dir))  plt.figure(figsize=(**10**, **5**)) # Adjust size  plt.bar(class\_counts.keys(), class\_counts.values(), width=**0.5**)  # Rotate labels  plt.xticks(rotation=**45**)  plt.title("Number of Images by Class in Train Folder")  plt.xlabel("Class name")  plt.ylabel("Images")  plt.tight\_layout()  plt.show() |

### A graph of blue rectangular objects Description automatically generated

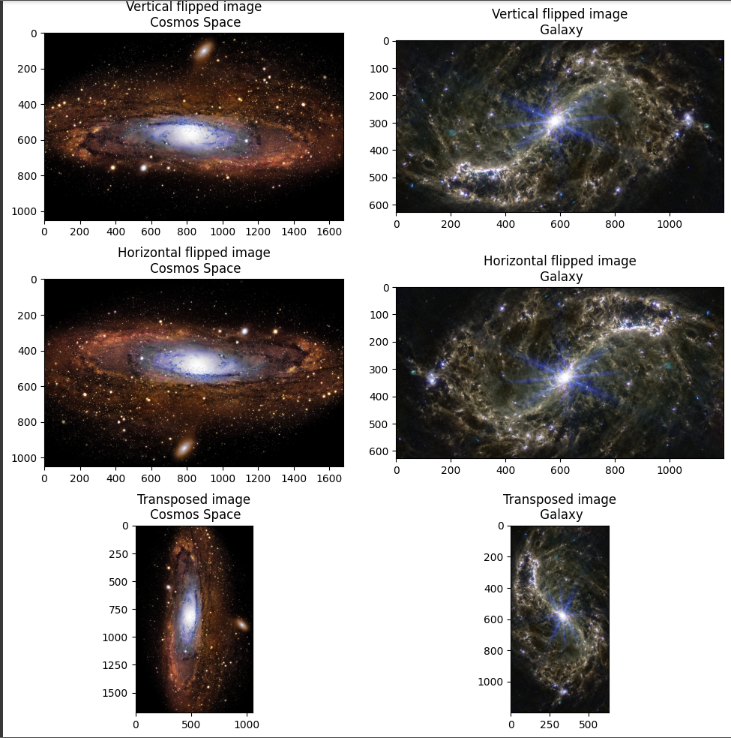
Basic Image display function from each class

|  |
| --- |
| **import** **cv2**  **import** **matplotlib.pyplot** **as** **plt**  %matplotlib inline  img\_path\_1 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/nebula - Google Search/45.jpg'  img\_1 = cv2.imread(img\_path\_1)  img\_path\_2 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/cosmos space - Google Search/144.jpg'  img\_2 = cv2.imread(img\_path\_2)  plt.figure(figsize=(**10**, **10**))  plt.subplot(**121**)  plt.imshow(img\_1),plt.title('Original image**\n** Nebula')  plt.subplot(**122**)  plt.imshow(img\_2),plt.title('Original image**\n** Cosmos Space') |



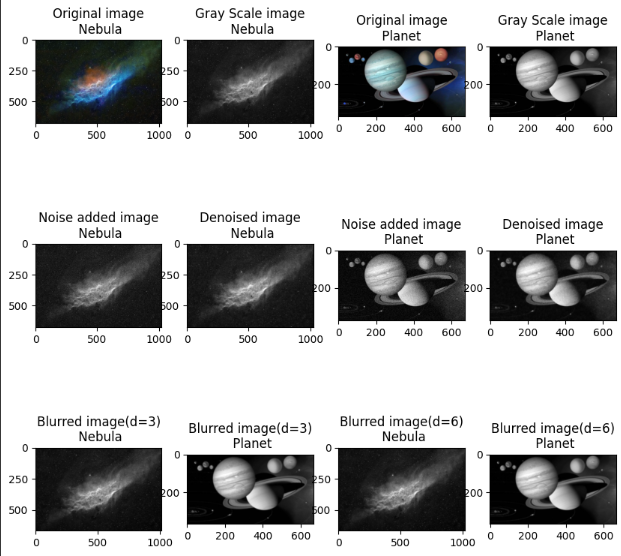
Geometric Transformation of Images

|  |
| --- |
| **import** **cv2**  **import** **matplotlib.pyplot** **as** **plt**  %matplotlib inline  img\_path\_1 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/cosmos space - Google Search/102.jpg'  img\_1 = cv2.imread(img\_path\_1)  img\_path\_2 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/galaxies - Google Search/10.jpg'  img\_2 = cv2.imread(img\_path\_2)  #Basic image manipulation (rotating/flipping/transpose)  flip\_img\_v1=cv2.flip(img\_1,**0**) # vertical flip  flip\_img\_v2=cv2.flip(img\_2,**0**) # vertical flip  #horizontal flip  flip\_img\_h1=cv2.flip(img\_1,**1**) # horizontal flip  flip\_img\_h2=cv2.flip(img\_2,**1**) # horizontal flip  #transpose  transp\_img\_1=cv2.transpose(img\_1,**1**) # transpose  transp\_img\_2=cv2.transpose(img\_2,**1**) # transpose  plt.figure(figsize=(**10**,**10**))  plt.subplot(**321**)  plt.imshow(flip\_img\_v1),plt.title('Vertical flipped image**\n** Cosmos Space')  plt.subplot(**322**)  plt.imshow(flip\_img\_v2),plt.title('Vertical flipped image**\n** Galaxy')  plt.subplot(**323**)  plt.imshow(flip\_img\_h1), plt.title('Horizontal flipped image**\n** Cosmos Space')  plt.subplot(**324**)  plt.imshow(flip\_img\_h2), plt.title('Horizontal flipped image**\n** Galaxy')  plt.subplot(**325**)  plt.imshow(transp\_img\_1),plt.title('Transposed image**\n** Cosmos Space')  plt.subplot(**326**)  plt.imshow(transp\_img\_2),plt.title('Transposed image**\n** Galaxy')  plt.tight\_layout()  plt.show() |



Colour and Texture Analysis

|  |
| --- |
| **import** **cv2**  **import** **numpy** **as** **np**  **import** **matplotlib.pyplot** **as** **plt**  **import** **skimage**  **import** **skimage.color** **as** **skic**  **import** **skimage.filters** **as** **skif**  **import** **skimage.data** **as** **skid**  **import** **skimage.util** **as** **sku**  %matplotlib inline  img\_path\_1 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/nebula - Google Search/113.jpg'  img\_1 = cv2.imread(img\_path\_1)  img\_path\_2 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/planets - Google Search/101.jpg'  img\_2 = cv2.imread(img\_path\_2)  #gray scale conversion  img\_1\_gray = skic.rgb2gray(img\_1)  img\_2\_gray = skic.rgb2gray(img\_2)  # We add Gaussian noise and denoise using denoise\_tv\_bregman approach  #for img\_1 and img\_2  img\_1\_n = sku.random\_noise(skic.rgb2gray(img\_1))  img\_1\_d = skimage.restoration.denoise\_tv\_bregman(img\_1\_n, **5.**)  img\_2\_n = sku.random\_noise(skic.rgb2gray(img\_2))  img\_2\_d = skimage.restoration.denoise\_tv\_bregman(img\_2\_n, **5.**)  #Noise reduction using Gaussian Blur  d=**3**  img\_1\_blur3 = cv2.GaussianBlur(skic.rgb2gray(img\_1), (**2**\*d+**1**, **2**\*d+**1**), -**1**)[d:-d,d:-d]  img\_2\_blur3 = cv2.GaussianBlur(skic.rgb2gray(img\_2), (**2**\*d+**1**, **2**\*d+**1**), -**1**)[d:-d,d:-d]  img\_1\_blur6 = cv2.GaussianBlur(skic.rgb2gray(img\_1), (**2**\*d+**1**, **2**\*d+**1**), -**1**)[d:-d,d:-d]  img\_2\_blur6 = cv2.GaussianBlur(skic.rgb2gray(img\_2), (**2**\*d+**1**, **2**\*d+**1**), -**1**)[d:-d,d:-d]  plt.figure(figsize=(**10**,**10**))  #VisualisingGray scale images visualisation  plt.subplot(**341**), plt.imshow(img\_1),plt.title('Original image**\n** Nebula')  plt.subplot(**342**), plt.imshow(img\_1\_gray, cmap = 'gray'),plt.title('Gray Scale image**\n** Nebula')  plt.subplot(**343**), plt.imshow(img\_2),plt.title('Original image**\n** Planet ')  plt.subplot(**344**), plt.imshow(img\_2\_gray, cmap = 'gray'),plt.title('Gray Scale image**\n** Planet')  #Visualising Noising-Denoising images  plt.subplot(**345**), plt.imshow(img\_1\_n,cmap = 'gray'), plt.title('Noise added image**\n** Nebula')  plt.subplot(**346**), plt.imshow(img\_1\_d,cmap = 'gray'),plt.title('Denoised image**\n** Nebula')  plt.subplot(**347**), plt.imshow(img\_2\_n,cmap = 'gray'),plt.title('Noise added image**\n** Planet')  plt.subplot(**348**), plt.imshow(img\_2\_d,cmap = 'gray'),plt.title('Denoised image**\n** Planet')  #Visualising Noise Reduction with Gaussian Blurring  plt.subplot(**349**), plt.imshow(img\_1\_blur3,cmap = 'gray'), plt.title('Blurred image(d=3)**\n** Nebula')  plt.subplot(**3**,**4**,**10**), plt.imshow(img\_2\_blur3,cmap = 'gray'),plt.title('Blurred image(d=3)**\n** Planet')  plt.subplot(**3**,**4**,**11**), plt.imshow(img\_1\_blur6,cmap = 'gray'),plt.title('Blurred image(d=6)**\n** Nebula')  plt.subplot(**3**,**4**,**12**), plt.imshow(img\_2\_blur6,cmap = 'gray'),plt.title('Blurred image(d=6)**\n** Planet') |



Feature Analysis of Images using edges and Conor detection

|  |
| --- |
| img\_path\_1 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/stars - Google Search/101.jpg'  img\_1 = cv2.imread(img\_path\_1)  img\_path\_2 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/cosmos space - Google Search/104.jpg'  img\_2 = cv2.imread(img\_path\_2)  #Sobel edge detector  #edge detector works on gray scale images  sobel\_img\_1=cv2.cvtColor(img\_1,cv2.COLOR\_BGR2GRAY)  sobel\_img\_2=cv2.cvtColor(img\_2,cv2.COLOR\_BGR2GRAY)  sobelx\_img\_1 = cv2.Sobel(sobel\_img\_1,cv2.CV\_64F,**1**,**0**,ksize=**9**)  sobely\_img\_1 = cv2.Sobel(sobel\_img\_1,cv2.CV\_64F,**0**,**1**,ksize=**9**)  sobelx\_img\_2 = cv2.Sobel(sobel\_img\_2,cv2.CV\_64F,**1**,**0**,ksize=**9**)  sobely\_img\_2 = cv2.Sobel(sobel\_img\_2,cv2.CV\_64F,**0**,**1**,ksize=**9**)  #Canny edge detector  #threshold selection  th1=**30**  th2=**60**  # Canny recommends threshold 2 is 3 times threshold 1  # you could try experimenting with this...  d=**3**  # gaussian blur  # this takes pixels in edgeresult where edge non-zero and colours them bright green  edgeresult\_1=img\_1.copy()  edgeresult\_1 = cv2.GaussianBlur(edgeresult\_1, (**2**\*d+**1**, **2**\*d+**1**), -**1**)[d:-d,d:-d]  gray\_1 = cv2.cvtColor(edgeresult\_1, cv2.COLOR\_BGR2GRAY)  edge\_1 = cv2.Canny(gray\_1, th1, th2)  edgeresult\_1[edge\_1 != **0**] = (**0**, **255**, **0**)  edgeresult\_2=img\_2.copy()  edgeresult\_2 = cv2.GaussianBlur(edgeresult\_2, (**2**\*d+**1**, **2**\*d+**1**), -**1**)[d:-d,d:-d]  gray\_2 = cv2.cvtColor(edgeresult\_2, cv2.COLOR\_BGR2GRAY)  edge\_2 = cv2.Canny(gray\_2, th1, th2)  edgeresult\_2[edge\_2 != **0**] = (**0**, **255**, **0**)  #Corner detector  #detecting corners for image\_1  harris\_1=img\_1.copy()  #greyscale it  gray = cv2.cvtColor(harris\_1,cv2.COLOR\_BGR2GRAY)  gray = np.float32(gray)  blocksize=**4** #  kernel\_size=**3** # sobel kernel: must be odd and fairly small  # run the harris corner detector  dst = cv2.cornerHarris(gray,blocksize,kernel\_size,**0.05**) # parameters are blocksize, Sobel parameter and Harris threshold  #result is dilated for marking the corners, this is visualisation related and just makes them bigger  dst = cv2.dilate(dst,**None**)  #we then plot these on the input image for visualisation purposes, using bright red  harris\_1[dst>**0.01**\*dst.max()]=[**0**,**0**,**255**]  #detecting corners for image\_2  harris\_2=img\_2.copy()  #greyscale it  gray = cv2.cvtColor(harris\_2,cv2.COLOR\_BGR2GRAY)  gray = np.float32(gray)  blocksize=**4** #  kernel\_size=**3** # sobel kernel: must be odd and fairly small  # run the harris corner detector  dst = cv2.cornerHarris(gray,blocksize,kernel\_size,**0.05**) # parameters are blocksize, Sobel parameter and Harris threshold  #result is dilated for marking the corners, this is visualisation related and just makes them bigger  dst = cv2.dilate(dst,**None**)  #we then plot these on the input image for visualisation purposes, using bright red  harris\_2[dst>**0.01**\*dst.max()]=[**0**,**0**,**255**]  #Visualisng Edges and Corners  plt.figure(figsize=(**10**,**10**))  #Visualising Sobel Edges  plt.subplot(**341**), plt.imshow(sobelx\_img\_1, cmap = 'gray'),plt.title('Horizontal edges**\n** Stars')  plt.subplot(**342**), plt.imshow(sobely\_img\_1, cmap = 'gray'),plt.title('Horizontal edges**\n** Cosmos Space')  plt.subplot(**343**), plt.imshow(sobelx\_img\_2, cmap = 'gray'),plt.title('Vertical edges**\n** Star')  plt.subplot(**344**), plt.imshow(sobely\_img\_2, cmap = 'gray'),plt.title('Vertical edges**\n** Cosmos')  #Visualising Canny Edges  plt.subplot(**345**), plt.imshow(img\_1),plt.title('Original image**\n** Star')  plt.subplot(**346**), plt.imshow(edgeresult\_1, cmap = 'gray'),plt.title('Canny edges**\n** Star')  plt.subplot(**347**), plt.imshow(img\_1),plt.title('Original image**\n** star')  plt.subplot(**348**), plt.imshow(edgeresult\_2, cmap = 'gray'),plt.title('Vertical edges**\n** Cosmos Space')  #Visualising Corners  plt.subplot(**349**), plt.imshow(cv2.cvtColor(img\_1, cv2.COLOR\_BGR2RGB)),plt.title('Original image**\n** Star')  plt.subplot(**3**,**4**,**10**), plt.imshow(cv2.cvtColor(harris\_1, cv2.COLOR\_BGR2RGB)),plt.title('Image with Corners**\n** Star')  plt.subplot(**3**,**4**,**11**), plt.imshow(cv2.cvtColor(img\_2, cv2.COLOR\_BGR2RGB)),plt.title('Original image**\n** Cosmos Space')  plt.subplot(**3**,**4**,**12**), plt.imshow(cv2.cvtColor(harris\_2, cv2.COLOR\_BGR2RGB)),plt.title('Image with Corners**\n** Cosmos Space') |

A group of images of stars

Description automatically generated

Analysing Lighting artifacts in images

|  |
| --- |
| img\_path\_1 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/galaxies - Google Search/122.jpg'  img\_1 = cv2.imread(img\_path\_1)  img\_path\_2 = '/content/drive/MyDrive/Capstone\_Databse/Data/train/nebula - Google Search/78.jpg'  img\_2 = cv2.imread(img\_path\_2)  **def** **show**(img):  # Display the image.  fig, (ax1, ax2) = plt.subplots(**1**, **2**,  figsize=(**12**, **3**))  ax1.imshow(img, cmap=plt.cm.gray)  ax1.set\_axis\_off()  # Display the histogram.  ax2.hist(img.ravel(), lw=**0**, bins=**256**)  ax2.set\_xlim(**0**, img.max())  ax2.set\_yticks([])  plt.show()  show(img\_1)  # adaptive histogram equalisation  show(skie.equalize\_adapthist(img\_1))  show(img\_2)  # adaptive histogram equalisation  show(skie.equalize\_adapthist(img\_2))  #class 1 image  img = skic.rgb2gray(img\_1)  sobimg\_nheq= skif.sobel(img)  show(sobimg\_nheq)  img = skic.rgb2gray(skie.equalize\_adapthist(img\_1))  sobimg\_heq\_1 = skif.sobel(img)  show(sobimg\_heq\_1)  #class 2 image  img = skic.rgb2gray(img\_2)  sobimg\_nheq= skif.sobel(img)  show(sobimg\_nheq)  img = skic.rgb2gray(skie.equalize\_adapthist(img\_2))  sobimg\_heq\_2 = skif.sobel(img)  show(sobimg\_heq\_2) |

A group of images of stars and galaxies

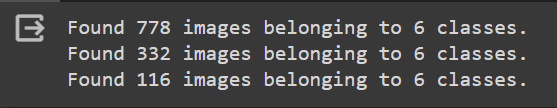
Description automatically generated

### Predictive Data Analysis

For the Predictive Data Analysis (PDA) I created a neural network to use epoch to train the model. I selected 11 epochs to track the train and validation folder and then created a plot to show the accuracy and loss comparison between both folders. A confusion matrix was also created to analyse images in both folders (train, validation)

View images in all directories

|  |
| --- |
| train = ImageDataGenerator(rescale = **1**/**255**)  val = ImageDataGenerator(rescale = **1**/**255**)  test = ImageDataGenerator(rescale= **1**/**255**)  d1 = train.flow\_from\_directory('/content/drive/MyDrive/Capstone\_Databse/Data/train', target\_size = (**400**,**400**),  batch\_size = **5**,  class\_mode = "binary")  d2 = train.flow\_from\_directory('/content/drive/MyDrive/Capstone\_Databse/Data/val', target\_size=(**400**,**400**),  batch\_size=**5**,  class\_mode = "binary")  d3 = train.flow\_from\_directory('/content/drive/MyDrive/Capstone\_Databse/Data/test',target\_size=(**400**,**400**),  batch\_size=**5**,  class\_mode = "binary") |



|  |
| --- |
| data\_model = tf.keras.models.Sequential([tf.keras.layers.Conv2D(**16**,(**3**,**3**),activation = 'relu',input\_shape = (**400**,**400**,**3**))  ,tf.keras.layers.MaxPool2D(**2**,**2**),  ##  tf.keras.layers.Conv2D(**32**,(**3**,**3**),activation = 'relu'),  tf.keras.layers.MaxPool2D(**2**,**2**),  ##  tf.keras.layers.Conv2D(**64**,(**3**,**3**),activation = 'relu'),  tf.keras.layers.MaxPool2D(**2**,**2**),  ##  tf.keras.layers.Flatten(),  ##  tf.keras.layers.Dense(**512**,activation="relu"),  ##  tf.keras.layers.Dense(**1**,activation='sigmoid')  ]) |

The code above represents the construction of the prediction neural network model, this is the model from which the epoch will output results, as shown above it uses different activation modes such as “relu”, “sigmod”.

Outputting the Epoch

|  |
| --- |
| **from** **keras.src.callbacks** **import** History  data\_model.compile(loss = "binary\_crossentropy",  optimizer = RMSprop(lr=**0.001**),  metrics = ['accuracy'])  History=data\_model.fit(d1,steps\_per\_epoch=**5**,epochs = **11**,  validation\_data = d2) |

A black and white screen

Description automatically generated

As shown above the model for the epoch was made through the “data\_model.compile”, data\_model was the name of variable used in the code before to create the model and that has been referenced again when outputting the epoch.

Train and Validation Accuracy

|  |
| --- |
| **import** **matplotlib.pyplot** **as** **plt**  l3, = plt.plot(History.history['val\_accuracy'], color='orange', label="validation\_accuracy")  l4, = plt.plot(History.history['accuracy'], color='red', label="train ccuracy")  plt.legend(handles=[l3, l4], loc='lower left')  plt.ylabel("accuracy")  plt.xlabel("epoch")  plt.title("Model Accuracy")  plt.figure().set\_figwidth(**15**)  plt.show() |

Please note that as the predictive model was run again, results might be different when comparing the Collaboratory file to the output shown below.

A graph with red lines

Description automatically generated

Train and Validation Loss

|  |
| --- |
| l1, = plt.plot(History.history['val\_loss'], color='orange', label="validation\_loss")  l2, = plt.plot(History.history['loss'], color='red', label=" train loss")  plt.legend(handles=[l1, l2], loc='upper right')  plt.ylabel("loss")  plt.xlabel("epoch")  plt.title("Model Loss")  plt.show() |

A graph with red and orange lines

Description automatically generated

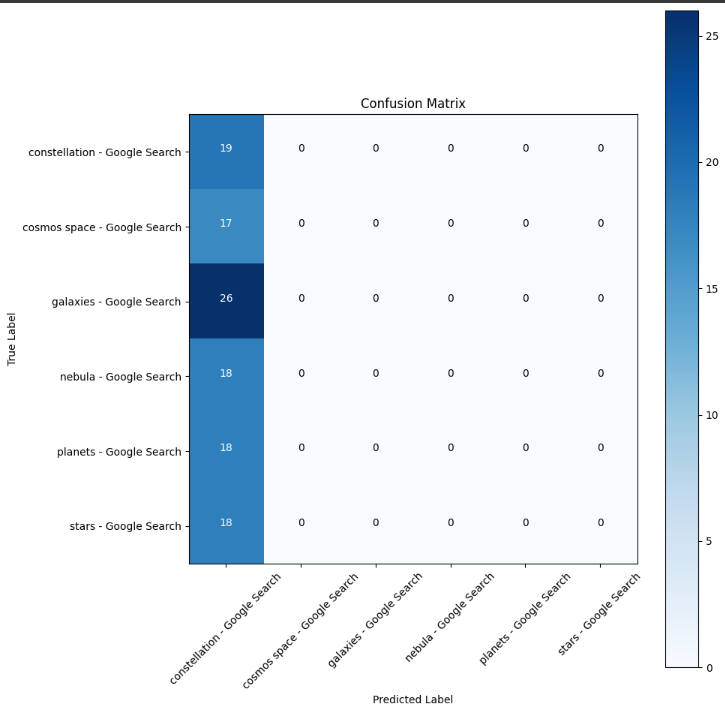
Prediction Generator

|  |
| --- |
| **from** **sklearn.metrics** **import** confusion\_matrix  **from** **tensorflow.keras.models** **import** load\_model  **from** **tensorflow.keras** **import** Model  preds = data\_model.predict\_generator(d3)  y\_pred = np.argmax(preds, axis=**1**) |

The code above is what we must create before making the confusion matrix. We are using the predict generator for the model to make predictions as it runs or tests the folder. The variable d3 represents the test folder.

Confusion Matrix

|  |
| --- |
| **from** **sklearn.metrics** **import** confusion\_matrix  **import** **itertools**  # Confusion Matrix  g\_dict = d3.class\_indices  classes = list(g\_dict.keys())  # Confusion matrix  cm = confusion\_matrix(d3.classes, y\_pred)  plt.figure(figsize= (**10**, **10**))  plt.imshow(cm, interpolation= 'nearest', cmap= plt.cm.Blues)  plt.title('Confusion Matrix')  plt.colorbar()  tick\_marks = np.arange(len(classes))  plt.xticks(tick\_marks, classes, rotation= **45**)  plt.yticks(tick\_marks, classes)  thresh = cm.max() / **2.**  **for** i, j **in** itertools.product(range(cm.shape[**0**]), range(cm.shape[**1**])):  plt.text(j, i, cm[i, j], horizontalalignment= 'center', color= 'white'  **if** cm[i, j] > thresh **else** 'black')  plt.tight\_layout()  plt.ylabel('True Label')  plt.xlabel('Predicted Label')  plt.show() |



Classification Report

|  |
| --- |
| # Classification report  **from** **sklearn.metrics** **import** classification\_report  print(classification\_report(d3.classes, y\_pred, target\_names=  classes)) |

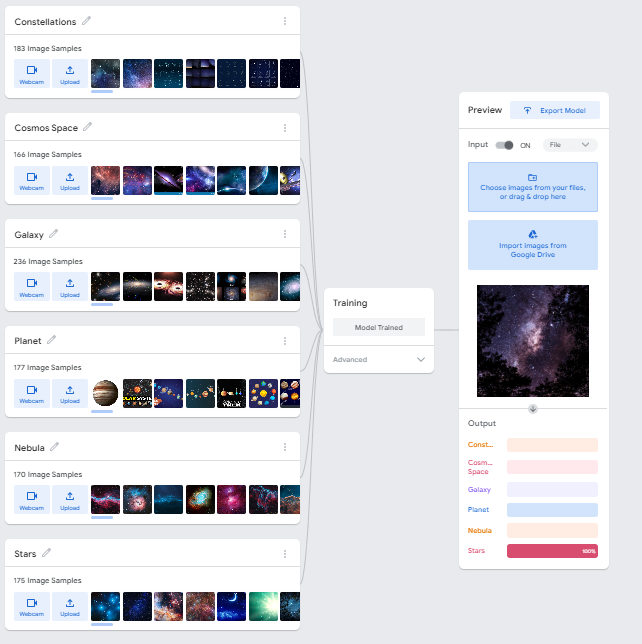
A screenshot of a computer screen

Description automatically generated

**Teachable Machine from Google**

For more accurate results from my dataset, I also used teachable machine to predict my model with 50 epochs and a batch size of 32. This is the model that I will input into my deployment stream lit web app.

Below is a screenshot of the teachable machine analysing my dataset and outputting a model, in the left it is shown that after uploading an image the model can properly output what class it belongs to, in the scenario below an image of stars has been uploaded and the model’s output is 100% stars.



## Software Implementation & Deployment Stage

### Streamlit Image Classification Learner

For the implementation & deployment stage I developed a streamlit web app to classify an image. After training the model in teachable machine, I exported the model into the same file where I have my code files relating to the capstone project. I exported both, the .h5 file and the txt file stating all my classes. After this I made a web app, which prompts users to attach a file relating to the 6 categories I was working on(Constellations, Cosmos Space, Galaxy, Planet, Nebula and Star). After the user attaches the image, with the use of TensorFlow and PIL the code runs the model and predicts the class the image belongs to and outputs the confidence score.

|  |
| --- |
| **import** **streamlit** **as** **st** # Libraries  **import** **tensorflow** **as** **tf**  **from** **PIL** **import** Image, ImageOps  **import** **numpy** **as** **np**  **from** **streamlit\_option\_menu** **import** option\_menu  st.set\_page\_config(page\_title="Image Classification", layout="wide")  # Create navigation bar  selected = option\_menu(  menu\_title=**None**,  options=["Home", "Image Classification Learner", "About"],  icons=["house", "clipboard-data", "person"],  default\_index=**1**,  orientation="horizontal"  )  #Creating the Home Page  **def** **home\_page**():  **with** st.container():  st.title("Image Classifcation Web App")    st.write("This image classigication learner uses a pre built prediction learner using teachable machine to predict what the image is!")  st.write("1. Upload an image from your device."  )  st.write("2. Make sure its an image from these 6 classes (Planets, Galaxy, Cosmos Space, Nebula, Stars and Constellations)")  st.write("3. Make sure it is in JPEG format")  st.write("4. Watch as the web app calculates what the picture is(Make sure its only related to space)")  #Creating the classification learner page  **def** **class\_learner**():  **global** model  model = tf.keras.models.load\_model("model.h5",compile=**False**)  class1 = open("labels.txt","r").readlines()  data = np.ndarray(shape=(**1**, **224**, **224**, **3**), dtype=np.float32)  st.header("Astronomy Image Classification Model")  up\_img = st.file\_uploader("Upload an image", type=["jpg","jpeg"])  **if** up\_img **is** **not** **None**:  data = np.ndarray(shape=(**1**, **224**, **224**, **3**), dtype=np.float32)  image = Image.open(up\_img).convert("RGB")  st.image(up\_img,use\_column\_width=**True**)  size = (**224**,**224**)  image = ImageOps.fit(image, size, Image.Resampling.LANCZOS)  image\_array = np.asarray(image)  image\_array1 = (image\_array.astype(np.float32)/**127.5**) - **1**  data[**0**] = image\_array1  prediction = model.predict(data)  index = np.argmax(prediction)  class\_name = class1[index]  confidence\_score = prediction[**0**][index]  st.write("Predicted Class: ", class\_name[**2**:])  st.write(f"Confidence Score:",confidence\_score \* **100**, "%")  **else**:  st.write("")  #creating the about page  **def** **about**():  st.header("About me")  st.write("")  **with** st.container():  st.write("My name is Advaith Ramakrishnan")  st.write("I am a student studying a double degree in software engineering and business informatics in UC.")  st.write("Contact Details: u3261011@uni.canberra.edu.au ")  **if** selected == "Home":  home\_page()  **elif** selected == "Image Classification Learner":  class\_learner()  **if** selected == "About":  about() |

### Home Page

**A screenshot of a computer

Description automatically generated**

### Image Classification Learner

**A screenshot of a computer

Description automatically generated**

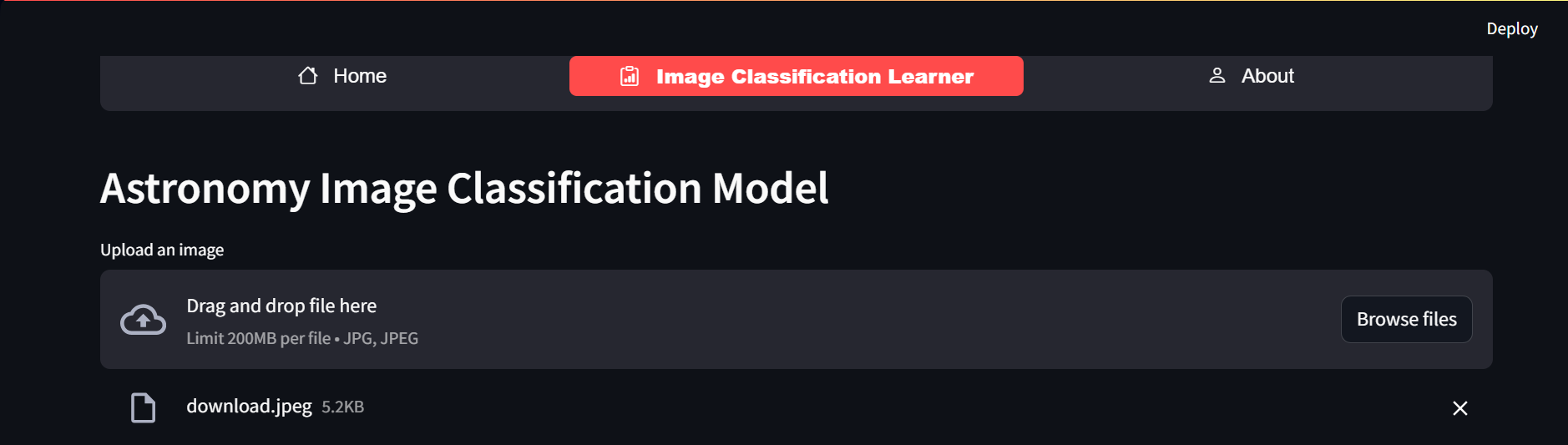
### About Page

**A screenshot of a computer

Description automatically generated**

### Deployment Screenshots

Below is an example of me inputting a planet image file into the learner, and the result given by the model.



### Output

**A close-up of a planet

Description automatically generated**

The code then prints the image out.

A screenshot of a video game

Description automatically generated

Finally, the code outputs the predicted class, and the confidence score. As shown in the image above, the prediction model can be considered as accurate as it has a high confidence score and it able to properly predict the class.

## Git Hub Repository

[**https://github.com/advaithram12**](https://github.com/advaithram12)

## Journal

|  |  |
| --- | --- |
| Week | Activities Done |
| Week 10 | Started working on my Predictive Data Analysis (EDA), starting on building neural network, mainly building epoch. |
| Week 11 | Continued working on Predictive Data Analysis, completed my neural network, by building epoch and a graph to plot model accuracy and loss. Also finished my St1 Quiz 3. |
| Week 12 | Started my Exploratory Data Analysis (EDA), started building basic visualisations such as analysing light, illumination, analysing edge detection and corners and using ski image I was able to analyse images using denoise functions.  Also worked on my capstone presentation. |
| Week 13 | Presented my capstone presentation, and started my implementation using stream lit, planning to make an image classification app. Imported all files into git hub repository. |