

# Deep Learning Techniques for Classification of Astronomical Objects

## Configuration Manual

MSc Research Project  
Programme

Yogiraj Subhash Dalvi  
Student ID: 20205741

School of Computing

National College of Ireland

Supervisor: Abubakr Siddig

**National College of Ireland**  
**MSc Project Submission Sheet**  
**School of Computing**



**Student Name:** Yogiraj Subhash Dalvi

**Student ID:** 20205741

**Programme:** MSc in Data Analytics

**Year:** 2022

**Module:** Research Project

**Lecturer:** Abubakr Siddig

**Submission Due Date:** 19<sup>th</sup> September 2022

**Project Title:** Deep Learning Techniques for Astronomical Object Classification

**Word Count:** 2258

**Page Count:** 23

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 authors' written or electronic work is illegal (plagiarism) and may result in disciplinary action.

**Signature:** Yogiraj Subhash Dalvi

**Date:** 19<sup>th</sup> September 2022

**PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST**

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

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

<b>Office Use Only</b>	
Signature:	
Date:	
Penalty Applied (if applicable):	

# Configuration Manual

Yogiraj Subhash Dalvi

Student ID: 20205741

## 1 Introduction

This report is a component of the submission made by the student Yogiraj Subhash Dalvi for the Master of Science degree in Data Analytics. It serves as the configuration manual for the manual that describes how to implement Deep Learning Techniques for the classification of astronomical objects. As Section 2 gives us the extra details of the thesis paper submitted, section 3 goes into detail about the hardware that was utilized, and section 4 gives us a discussion about the framework that was used to create the models. The steps to choose and acquire the images which will be utilized are discussed in Section 5. The Python code that was used to obtain and pre-process the SDSS pictures is walked through in Section 6. Training of models is discussed in Section 7, which follows, and section 8 discusses the outcomes that were obtained from the training.

## 2 Additional Details from Thesis Technical Report

The following is a list of supporting documentation that was discovered or researched as part of the research effort; however, due to limitations in the document, it was not included in the final thesis report.

### 2.1 Methodology

TensorFlow is a high-scale machine learning system. TensorFlow employs dataflow graphs to describe computation, shared state, and mutations. It translates a dataflow graph over many computers in a cluster and inside a machine across several processing devices, including multicore CPUs, GPUs, and Tensor Processing Units (TPUs). This architecture allows the application developer freedom, unlike earlier "parameter server" implementations. TensorFlow lets developers try new training and optimization strategies. TensorFlow focuses on training and inferring deep neural networks. Several Google services utilize TensorFlow in production, and the open-source project is used for machine learning research. In this publication, the TensorFlow dataflow model is explained. (TensorFlow, 2022) This study implemented three pre-trained CNN models along with a base CNN model with adam optimization and it was compared against the same models but, they were trained on the ImageNet dataset.

### 2.2 Convolutional Neural Networks

Deep learning makes use of many kinds of neural networks, and one of those networks is called a convolutional neural network. Because of the work done by CNN, image recognition has made great progress in recent years. CNN's are comprised of several layers, the most

notable of which are the input layer, the output layer, and the hidden layers. The processing and classification of images is helped along in some fashion by each of these levels in some manner. Every layer is considered while calculating the hidden layers. Each one of these layers contributes significantly to the overall functionality of the network.

## 2.3 Data Acquisition

Once we get all the FITS to file URLs after querying the SDSS archive server, all those files were downloaded locally using the “shutil” library.

```
## Importing Necessary Modules
import requests # to get image from the web
import shutil # to save it locally

for ix, thisrow in df.iterrows():

    ## Set up the image URL and filename
    #image_url = "https://cdn.pixabay.com/photo/2020/02/06/09/39/summer-4823612_960_720.jpg"
    filename = str(ix) + '_' + str(thisrow['class']) + '.jpg'

    # Open the url image, set stream to True, this will return the stream content.
    r = requests.get(thisrow.imglink, stream = True)

    # Check if the image was retrieved successfully
    if r.status_code == 200:
        # Set decode_content value to True, otherwise the downloaded image file's size will be zero.
        r.raw.decode_content = True

        # Open a Local file with wb ( write binary ) permission.
        with open(filename,'wb') as f:
            shutil.copyfileobj(r.raw, f)

        print('Image successfully Downloaded: ',filename)
    else:
        print('Image Couldn\'t be retrieved')
```

Fig 1: Saving SDSS files in .jpeg format to local drive after fetching FITS file URLs

After downloading images, they were segregated into three classes.



Fig 2: Sample Images from SDSS Data Release 17

FITS files download location: <http://skyserver.sdss.org/DR17/SkyServerWS/ImgCutout/>

SDSS JPEG files Download location: <http://skyserver.sdss.org/DR17/SkyServerWS/ImgCutout/>

## 2.4 Data Pre-Processing

After the data had been obtained in the fit format, it was then processed to minimize its size and convert it to an image format so that it could be handled further down the line. This processing, along with the other portions of the model's design, training, and testing, was carried out locally. It was decided to download the FITS files and store them in individual sub-directories, one for each class. After that, the pictures were taken from each FITS image, assembled into a plot, and then saved as a PNG file. Finally, the image was closed, and the process was repeated with the subsequent image in the directory. Python was used to write this procedure, and the Config Manual that was given along with this report has comprehensive information on the various pieces of hardware and software that were deployed.

### 2.4.1 Image Augmentation

Image data augmentation is a method that entails developing changed variants of the images that are included within a training dataset. These new versions of the images are then used in the training process. After then, the updated copies are put to use for instructional reasons. This makes it feasible to artificially grow the size of the dataset without the need to collect any more data throughout the process. There are different types of augmentation techniques, such as:

1. Horizontal or vertical image flipping.
2. Choose clockwise or counterclockwise to rotate the image.
3. Rescaling pixels of the image
4. Randomly cropping an image

## 2.5 Data Extraction and Transformation

The SDSS Survey archive servers are the source of the data in this particular instance. The data that was utilized may be found on the servers at [dr17.sdss.org](http://dr17.sdss.org). DR17 referred to the 17th data release of information based on the findings of the survey. All the data has already undergone preprocessing, during which any digital noise or "poor" photos were eliminated. Before the data was made public, any images that had been negatively influenced by bad weather circumstances or unfavorable atmospheric conditions were deleted.

## 3. Hardware Specification

System: Gaming HP Laptop 15<sup>th</sup> Generation  
Processor: Intel(R) Core (TM) i5-10300H CPU @ 2.50GHz  
Installed RAM: 16.0 GB (15.8 GB usable)  
OS: Windows 10 Pro 64-bit edition

Graphics Card: NVIDIA GeForce GTX1660 Ti ,6GB DDR6

## 4. Software used

This section provides a list of all apps that must be installed in order for the project to be successfully executed, as well as screen images.

### 4.1 Applications Developed by Microsoft

The ordinary versions of Microsoft Office and the operating system that were utilized in this study do not come with any installation instructions. Any extra components that were used are: Microsoft Excel, PowerPoint, and Snipping Tool

## 4.2 Google Colab

The acronym "Colab" refers to the product that was developed by Google Research and is known as the Collaboratory. Colab is an online platform that enables users to create and run any Python code directly in their web browser. This platform is particularly useful for machine learning and data analysis.

1. Go to <https://colab.research.google.com/>
2. You need to sign up in colab to create an account in colab

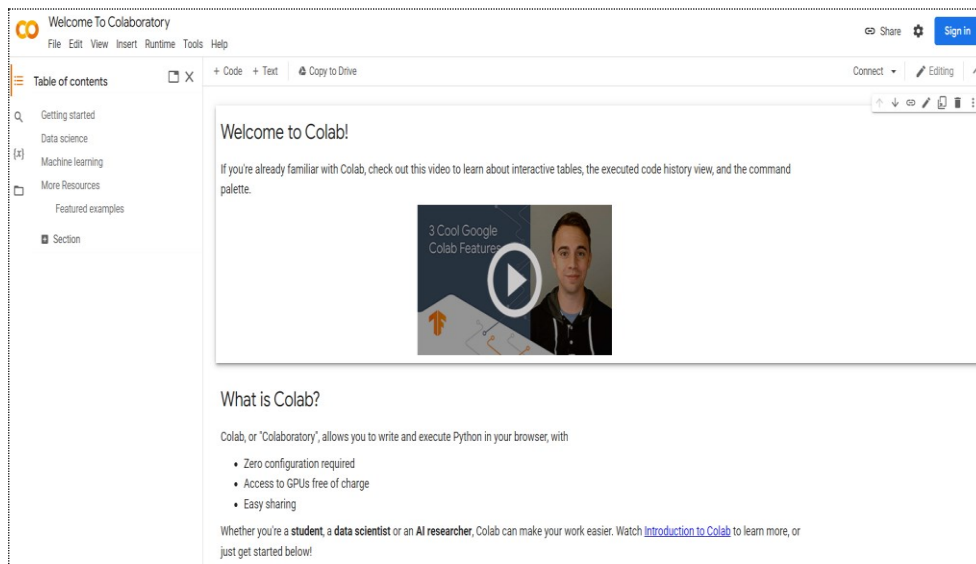


Fig 3. Google Colab Home Page

3. Login with your Gmail account credentials.

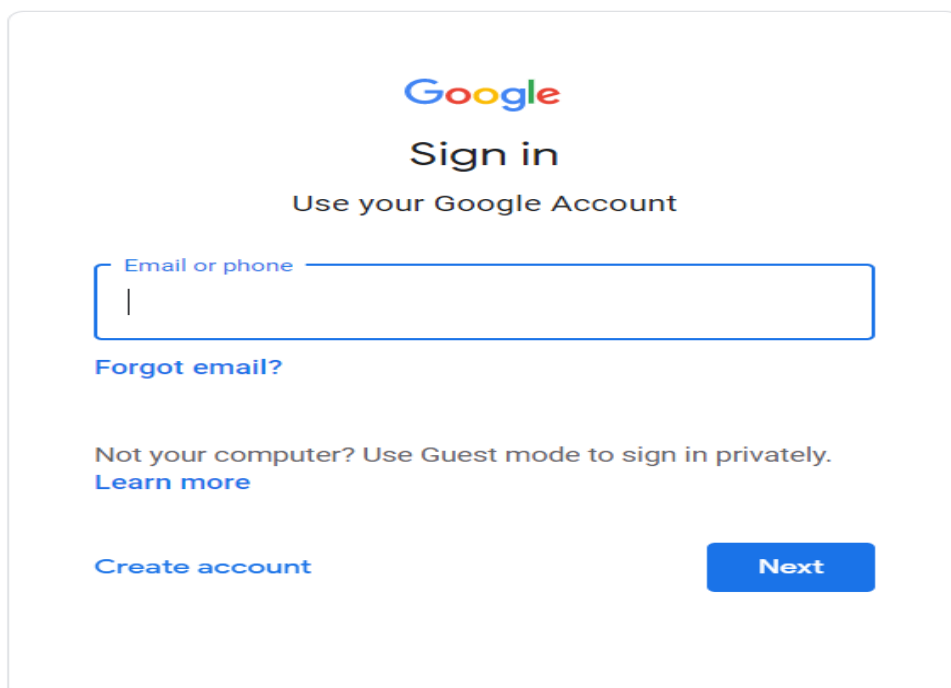


Fig 4. Google Sign-In In Colaboratory

4. Create a new notebook to start working in colab



Fig 5. New Python Notebook in Colab

5. There are three types of colab versions:

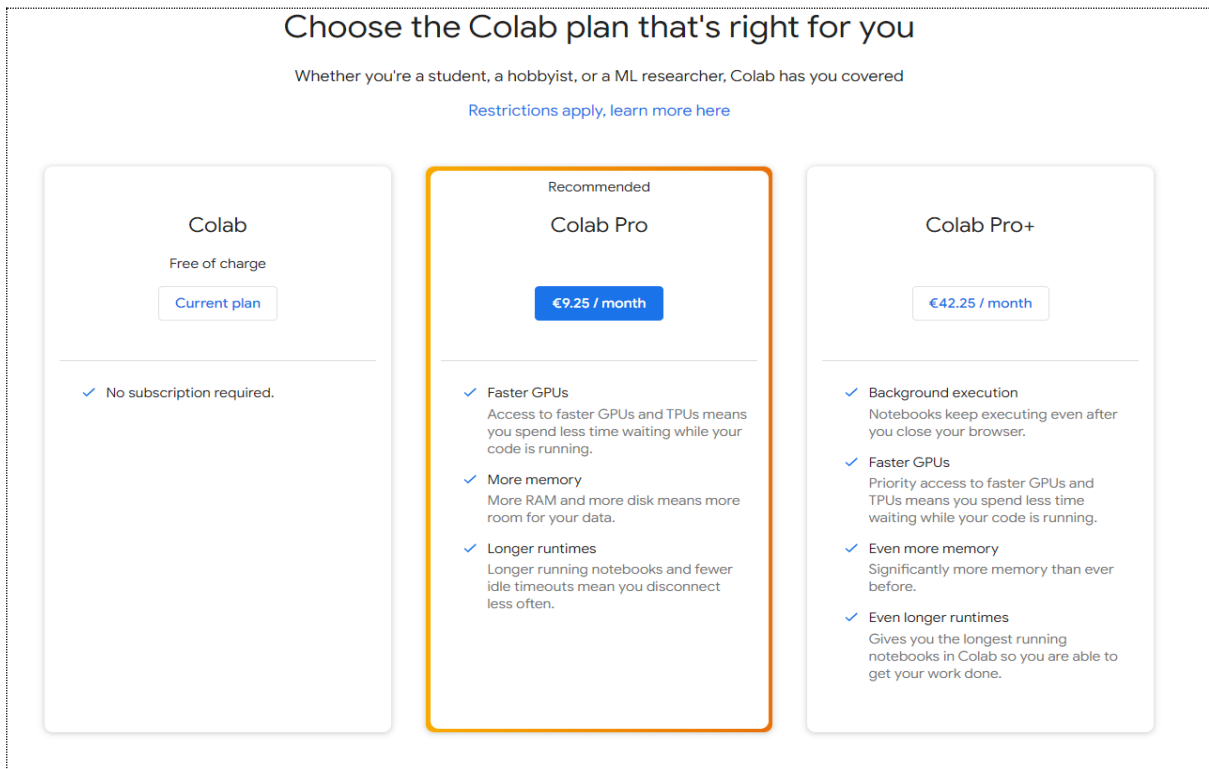


Fig 6. Google Colab Plans

The another advantage of google colab is that majority of the python dependencies have already been installed in it so, there would be no need to install any dependencies in it.

## 5. Downloading SDSS Images

Images were retrieved from the server that houses the SDSS archive. The format of the queries was determined, in both instances, by the website that was being queried.

### 5.1 Querying SDSS Catalogue Archive Server using Python and SQL

The code to extract the images from SDSS (Sdss.org. Data Release 17, 2022) archive server is as follows.

#### 5.1.1 Importing necessary libraries for Sciserver

```
# import SciServer libraries
import SciServer.CasJobs as CasJobs
import SciServer.SkyServer as skys

import urllib.request
import os

# import utility libraries
import pandas
import numpy as np
pandas.set_option('display.max_colwidth',None)
import astropy
#from astropy.io import fits
#from astropy import wcs
import skimage.io
import matplotlib
from matplotlib import pyplot as plt

dataset = 'DR17'|

print('ok')
```

Fig 7. Importing Libraries for Sciserver Casjob

#### 5.1.2 Extract Star Class images using SQL Query

ExecuteQuery() function was used to query Sciserver archive for Star Class FITS Files.

```
imgwidth = 512
imgheight = 512
imgscale = 0.024 # arcsec per pixel

sql="""
select top 3042 s.objid, s.specobjid, s.class, p.ra, p.dec, p.u, p.g, p.r, p.i, p.z, p.run, p.rerun, p.camcol, p.field,
p.petro90_r, REPLACE(dbo.fGetUrlFitsCFrame(s.fieldid, 'r'),'http://das.sdss.org','/home/idies/workspace/sdss_das/das2')
as fits_r
from specphoto s
join photoobj p on s.objid=p.objid where s.class = 'STAR'
"""
df = CasJobs.executeQuery(sql, dataset)
df = df.set_index('objid')

df = df.assign(imglink = np.nan)

for ix, thisrow in df.iterrows():
    df.loc[ix, 'imglink'] = 'http://skyserver.sdss.org/{0:}/SkyServerWS/ImgCutout/getjpeg?ra={1:.4f}&dec={2:.4f}&scale={3:.2f}&h

# imgfilename = '{0:}.jpg'.format(thisrow['objid'])
# imgfile = urllib.request.urlopen(imglink).read().decode()
# with open(imgfile, 'w') as f:
#     f.write()
# print('done')
# os.listdir()
df
```

Fig 8. SQL Query for retrieving STAR class FITS files



### 5.1.3 Saving STAR class image attributes in csv file

All information regarding STAR class images has been saved to a local drive via python data frame.

```
df.to_csv('skyserver_SQL2_10_2022_star.csv')
```

Fig 9. Save image information to CSV file

The CSV has FITS file URLs as well.

objid	specobjid	class	ra	dec	u	g	r	i	z	run	rerun	cancel	field	petror90_fits_r	imglink
1.24E+18	3.24E+17	STAR	184.9983	-0.83329	22.5168	19.86943	18.80936	18.42988	18.11434	752	301	2	277	1.886306	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/752,http://skyserver.sdss.org
1.24E+18	3.29E+17	STAR	192.3106	-0.82372	18.37756	17.23364	17.1101	17.05626	17.0541	752	301	2	326	1.53062	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/752,http://skyserver.sdss.org
1.24E+18	3.04E+17	STAR	153.5742	-0.95261	20.10137	19.07438	18.67575	18.59837	18.51201	756	301	1	252	1.549077	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	157.2287	-0.93279	22.1809	19.73117	18.30559	17.97781	17.27904	756	301	1	277	1.279921	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	157.3973	-1.01329	19.48027	18.17562	17.64455	17.44124	17.38654	756	301	1	278	1.331083	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	169.0822	-0.93856	19.79673	17.83767	16.80932	16.34618	15.86454	756	301	1	356	8.389127	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.04E+17	STAR	152.4737	-0.48151	19.63906	19.4021	19.45169	18.8509	18.25064	756	301	2	245	1.259542	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	158.1805	-0.54545	19.89175	20.08175	20.32298	20.63092	20.78814	756	301	2	283	1.370201	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.1E+17	STAR	160.9953	-0.47853	19.04517	17.87662	17.49664	17.37287	17.33316	756	301	2	302	1.377759	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.11E+17	STAR	162.9077	-0.49897	20.40022	17.78141	16.42986	15.84891	15.52807	756	301	2	314	1.51429	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.24E+17	STAR	185.6969	-0.47676	21.49655	19.27652	18.11299	17.63111	17.3677	756	301	2	467	1.377804	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	157.6916	-0.07177	20.21635	19.00464	18.91041	18.89328	18.89704	756	301	3	280	1.337706	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	158.1911	-0.07371	22.29553	19.45618	18.00819	17.24348	16.79843	756	301	3	283	1.592383	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	171.526	-0.11257	19.34245	18.44862	16.88546	18.87642	19.02608	756	301	3	372	1.212863	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3E+17	STAR	145.2928	0.376326	19.31302	18.13527	17.67289	17.49996	17.44187	756	301	4	197	1.572772	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.08E+17	STAR	157.9704	0.213096	18.71556	17.57673	17.38265	17.33843	17.32211	756	301	4	281	1.181168	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.1E+17	STAR	161.6125	0.306345	17.84553	16.82906	16.73855	16.88843	16.94442	756	301	4	306	1.278103	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.08E+17	STAR	145.2841	0.706298	20.36442	18.21354	17.38881	17.11562	17.32	756	301	5	197	1.646003	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3E+17	STAR	146.4971	0.742417	18.74275	17.26978	16.62725	16.34429	16.21385	756	301	5	205	1.416634	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	156.7058	0.65153	19.97817	19.88066	20.21217	20.42117	20.54344	756	301	5	273	1.239124	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	156.8952	0.996079	17.18925	15.98545	15.53578	15.36095	15.28022	756	301	5	274	1.280231	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.09E+17	STAR	160.9488	0.752416	19.9629	18.31689	17.90659	17.43141	17.28793	756	301	5	301	1.362178	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.12E+17	STAR	165.8836	0.744894	20.73067	20.30761	20.47252	20.71087	20.52414	756	301	5	334	1.598131	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.07E+17	STAR	158.0297	1.238775	21.13235	18.49486	17.20728	16.66124	16.30044	756	301	6	282	1.359443	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.1E+17	STAR	160.9145	1.076762	19.52762	18.34265	18.41109	18.50452	18.4938	756	301	6	301	1.312291	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.1E+17	STAR	162.1785	1.199562	23.88854	22.59401	19.85314	17.30329	15.43482	756	301	6	310	1.727914	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.15E+17	STAR	170.8474	1.111108	21.30914	20.17535	19.8846	19.41808	19.33467	756	301	6	367	1.63108	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org
1.24E+18	3.29E+17	STAR	192.3384	1.224697	20.46688	19.53189	19.15761	19.00493	18.99938	756	301	6	515	1.600869	http://dr17.sdss.org/sas/dr17/eboss/photoObj/frames/301/756,http://skyserver.sdss.org

Fig 10. CSV File Attributes of Star Images

### 5.1.4 Downloading STAR class JPEG images

Using the FITS file URLs, the file is converted into .jpg format and saved in a local drive. The same process was followed for Galaxy and Quasars images as well.

```
## Importing Necessary Modules
import requests # to get image from the web
import shutil # to save it locally

for ix, thisrow in df.iterrows():

    ## Set up the image URL and filename
    #image_url = "https://cdn.pixabay.com/photo/2020/02/06/09/39/summer-4823612_960_720.jpg"
    filename = str(ix) + '_' + str(thisrow['class']) + '.jpg'

    # Open the url image, set stream to True, this will return the stream content.
    r = requests.get(thisrow.imglink, stream = True)

    # Check if the image was retrieved successfully
    if r.status_code == 200:
        # Set decode_content value to True, otherwise the downloaded image file's size will be zero.
        r.raw.decode_content = True

        # Open a local file with wb ( write binary ) permission.
        with open(filename,'wb') as f:
            shutil.copyfileobj(r.raw, f)

            print('Image successfully Downloaded: ',filename)
    else:
        print('Image Couldn't be retrieved')
```

Fig 11. Code Snippet for downloading images in .jpg format

```

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           1  ImageHDU      6     (2048,)   float32
2   2           1  BinTableHDU   27   1R x 3C   [49152E, 2048E, 1489E]
3   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, E, E]

```

Fig 12. General structure of FITS Files

## 6. Model Training

The following bundle contains all the models that were used throughout the training process for the different datasets. The VGG16, InceptionV3, ResNet50, VGG16, and base CNN model with adam optimization were used in this project's coverage of the models.

On the first run for each model, the learned weights from the model as it was trained using Image Net images were retrieved. The format that was followed by each model was the same. As we are implementing pre-trained models, it was important to download weights for each model.

```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16\_weights\_tf\_dim\_ordering\_tf\_kernels\_notop.h5
58892288/58889256 [=====] - 0s 0us/step
58900480/58889256 [=====] - 0s 0us/step

```

Fig 13. Weights Downloading

### 6.1 Provision of necessary Python libraries

All of the models shared the same code and library components, as shown in Figure 14.

```

# Importing Libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import cv2
import os

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing import image
from keras.callbacks import ModelCheckpoint, EarlyStopping

from keras.layers import Dense, Activation, Flatten
from keras.layers.convolutional import Conv2D, MaxPooling2D
from tensorflow.keras.optimizers import Adam
from keras import models, layers, optimizers
import seaborn as sns
from keras.callbacks import History
from sklearn.metrics import classification_report, confusion_matrix
from pyarsing.core import trace_parse_action
import seaborn as sns

from sklearn.metrics import confusion_matrix

```

Fig 14. Provision of Python Libraries

The google drive was mounted in colab so that SDSS images can be accessible for model training.

```
#Mounting Google Drive in Colab  
  
from google.colab import drive  
drive.mount('/content/drive')
```

Fig 15. Mounting Google Drive in Colab

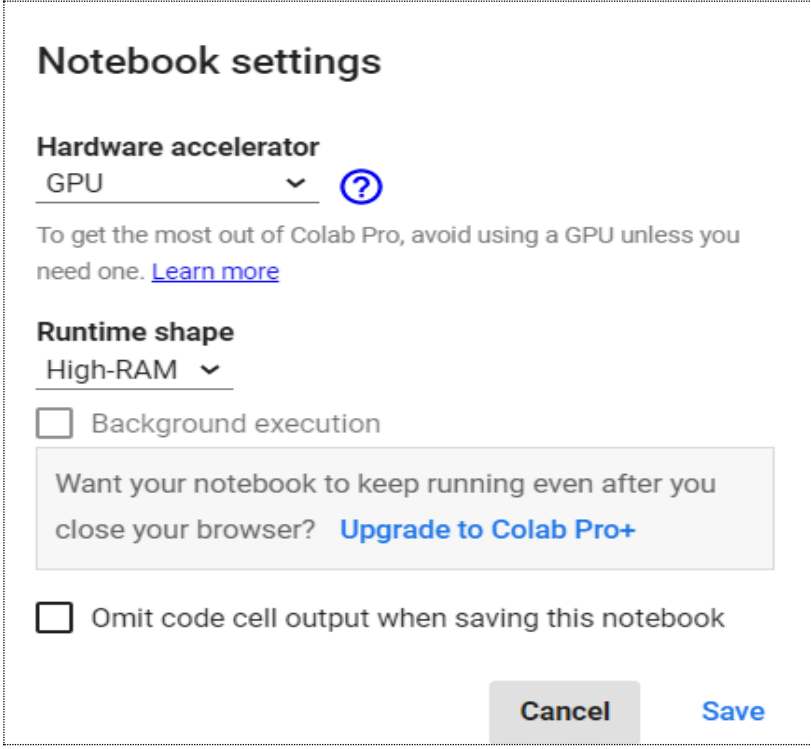
The working directory where sdss images are stored, was set for model training.

```
#Set Working directory for Input Images  
  
path = "/content/drive/MyDrive/Thesis/SDSS_Images/ImagesNew"  
  
data = tf.keras.preprocessing.image_dataset_from_directory(path)
```

Fig 16. Working directory

## 6.2 Change the notebook setting in Google colab

We changed the settings of python notebook in colab so that it would help model to train faster. The hardware accelerator needs to be changed into GPU processing and Runtime Shape needs to change into High-RAM.



**Notebook settings**

**Hardware accelerator**  
GPU ▼ ?

To get the most out of Colab Pro, avoid using a GPU unless you need one. [Learn more](#)

**Runtime shape**  
High-RAM ▼

Background execution

Want your notebook to keep running even after you close your browser? [Upgrade to Colab Pro+](#)

Omit code cell output when saving this notebook

Cancel Save

Fig 15. Notebook settings in Colab

### 6.3 Image Augmentation

The training, validation and Testing dataset were generated using image augmentation techniques.

```
#Using ImageDataGenerator Function for Pre-processing

train_datagen = ImageDataGenerator(
    rescale = 1./255,
    rotation_range = 20,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    horizontal_flip = True,
    vertical_flip = True,
    fill_mode='nearest'
)
validation_datagen = ImageDataGenerator(
    rescale = 1./255
)
test_datagen = ImageDataGenerator(
    rescale = 1./255
)
```

Fig 16. Image Augmentation on SDSS Images

### 6.4 Model Modification

The basic model was brought in; however, the topmost layer was not. After that, the models were made untrainable to locate the layers that had previously been trained; these layers were able to extract features from images. The models have had four extra layers added to them, and they are now configured to be trainable.

```
#Adding new layers to VGG16

headModel = vgg.output
headModel = Flatten(name="flatten")(headModel)
headModel = Dense(512, activation="relu")(headModel)
headModel = Dropout(0.5)(headModel)
headModel = Dense(3, activation="softmax")(headModel)

# place the head FC model on top of the base model (this will become the actual model we will train)

model = Model(inputs=vgg.input, outputs=headModel)
```

Fig 17. Adding new layers to the base model

The model was compiled using compile() function

```
# Compile the model

model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['acc'])
```

Fig 18. Model Compilation

The models were executed while using the function called fit()

```
# Train the VGG16 model

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
mc = ModelCheckpoint('VGG16 Classifier.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

history = model.fit(
    train_generator,
    steps_per_epoch=150,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples//batch_size,
    verbose=1,
    callbacks = [es, mc],)
```

Fig 19. Model Execution

Once the model training is completed, training-validation plots were plotted to review the results.

```
#Plot for Training and Validation Accuracy - VGG16

train_acc = history.history['acc']
val_acc = history.history['val_acc']
train_loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(train_acc) + 1)

plt.plot(epochs, train_acc, 'b*-', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'r', label = 'Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

#Plot for Training and Validation Loss Function - VGG16

plt.plot(epochs, train_loss, 'b*-', label = 'Training loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Fig 20. Plots for Training-Validation Accuracies and Loss function for the models

## 6.5 Model Fine-Tuning

The method of fine-tuning involves taking a model that has previously been trained for one specific job and then tweaking or otherwise modifying the model to have it execute a second task that is like the first.

In this scenario, the parameters of our already trained model were tuned meaning the few parameters of our model were unfrozen so that it can learn some new specific features which might help us in this classification task.

```
# Fine Tuning of VGG16 Model

# We chose to freeze first 15 layers of the model and rest of the layers will be trainable

for layer in vgg.layers[15:]:
    layer.trainable = True
```

Fig 21. Tuning of VGG16 Model

Now, the model had to be trained again.

```
# Train the VGG16 model

es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)
mc = ModelCheckpoint('VGG16 Classifier.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)

history = model.fit(
    train_generator,
    steps_per_epoch=150,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples//batch_size,
    verbose=1,
    callbacks = [es, mc],)
```

Fig 22. Model training after fine-tuning

Again, we had to plot training-validation plots.

```
#Plot for Training and Validation Accuracy - VGG16 : After Fine-Tuning

train_acc = vgg16_tuning.history['acc']
val_acc = vgg16_tuning.history['val_acc']
train_loss = vgg16_tuning.history['loss']
val_loss = vgg16_tuning.history['val_loss']

epochs = range(1, len(train_acc) + 1)

plt.plot(epochs, train_acc, 'b*-', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'r', label = 'Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()

#Plot for Training and Validation Loss Function - VGG16

plt.plot(epochs, train_loss, 'b*-', label = 'Training loss')
plt.plot(epochs, val_loss, 'r', label = 'Validation loss')
plt.title('Training and validation loss')
plt.legend()

plt.show()
```

Fig 23. Training-Validation Plots after Fine-Tuning

## 7. Evaluation and Results

The following is a complete record of the results, which were missing from the report on the project. Every model was put through its paces, and the accuracy of the models was used to evaluate how well they performed.

### 7.1 VGG16 Model

```
# Results of VGG16 after Fine-tuning

training_accuracy_vgg16    = vgg16_tuning.history['acc'][-1]
training_loss_vgg16       = vgg16_tuning.history['loss'][-1]
validation_accuracy_vgg16 = vgg16_tuning.history['val_acc'][-1]
validation_loss_vgg16     = vgg16_tuning.history['val_loss'][-1]
print("Training Accuracy VGG16   :", training_accuracy_vgg16 )
print("Training Loss VGG16      :", training_loss_vgg16)
print("Validation Accuracy VGG16  :", validation_accuracy_vgg16)
print("Validation Loss VGG16     :", validation_loss_vgg16)

Training Accuracy VGG16   : 0.8675330281257629
Training Loss VGG16      : 0.35223162174224854
Validation Accuracy VGG16 : 0.8604910969734192
Validation Loss VGG16     : 0.3818811774253845
```

Fig 24. Accuracies of VGG16

```
Epoch 1/10
150/150 [=====] - 69s 455ms/step - loss: 0.5315 - acc: 0.7433 - val_loss: 0.4481 - val_acc: 0.8292
Epoch 2/10
150/150 [=====] - 67s 449ms/step - loss: 0.5277 - acc: 0.7382 - val_loss: 0.4233 - val_acc: 0.8504
Epoch 3/10
150/150 [=====] - 67s 448ms/step - loss: 0.5250 - acc: 0.7648 - val_loss: 0.4224 - val_acc: 0.8482
Epoch 4/10
150/150 [=====] - 67s 447ms/step - loss: 0.4983 - acc: 0.7971 - val_loss: 0.4522 - val_acc: 0.8259
Epoch 5/10
150/150 [=====] - 67s 448ms/step - loss: 0.4606 - acc: 0.8361 - val_loss: 0.4415 - val_acc: 0.8471
Epoch 6/10
150/150 [=====] - 67s 448ms/step - loss: 0.4495 - acc: 0.8371 - val_loss: 0.4065 - val_acc: 0.8527
Epoch 7/10
150/150 [=====] - 67s 448ms/step - loss: 0.4228 - acc: 0.8487 - val_loss: 0.3918 - val_acc: 0.8605
Epoch 8/10
150/150 [=====] - 68s 450ms/step - loss: 0.4231 - acc: 0.8483 - val_loss: 0.4285 - val_acc: 0.8304
Epoch 9/10
150/150 [=====] - 68s 450ms/step - loss: 0.4028 - acc: 0.8523 - val_loss: 0.3899 - val_acc: 0.8527
Epoch 10/10
150/150 [=====] - 67s 449ms/step - loss: 0.3782 - acc: 0.8587 - val_loss: 0.3815 - val_acc: 0.8560
```

Fig 25. Epoch Run of VGG16

```
#Plot for Training and Validation Accuracy - VGG16 : After Fine-Tuning
```

```
train_acc = vgg16_tuning.history['acc']  
val_acc = vgg16_tuning.history['val_acc']  
train_loss = vgg16_tuning.history['loss']  
val_loss = vgg16_tuning.history['val_loss']  
  
epochs = range(1, len(train_acc) + 1)  
  
plt.plot(epochs, train_acc, 'b*-', label = 'Training accuracy')  
plt.plot(epochs, val_acc, 'r', label = 'Validation accuracy')  
plt.title('Training and validation accuracy')  
plt.legend()  
  
plt.figure()
```

<Figure size 432x288 with 0 Axes>



<Figure size 432x288 with 0 Axes>

Fig 26. Training-Validation Accuracy Plot - VGG16

```
#Plot for Training and Validation Loss Function - VGG16
```

```
plt.plot(epochs, train_loss, 'b*-', label = 'Training loss')  
plt.plot(epochs, val_loss, 'r', label = 'Validation loss')  
plt.title('Training and validation loss')  
plt.legend()  
  
plt.show()
```



Fig 27. Training-Validation Loss Function Plot - VGG16



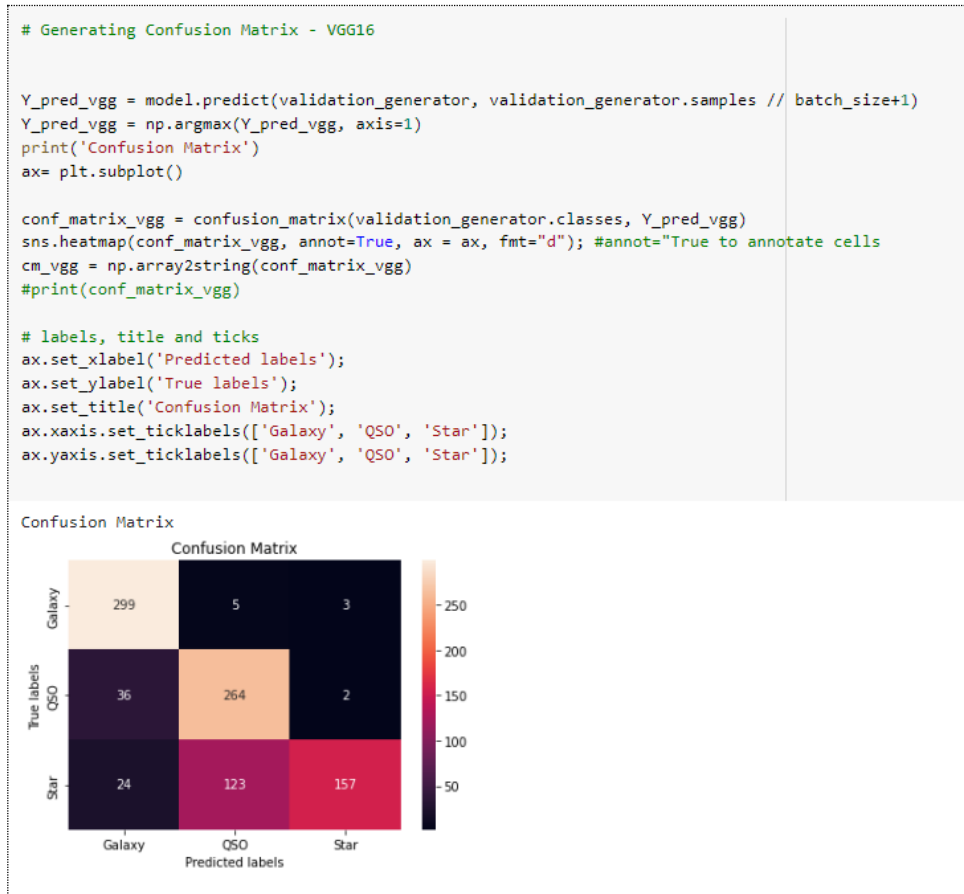


Fig 28: Confusion Matrix - VGG16

## 7.2 InceptionV3 Model

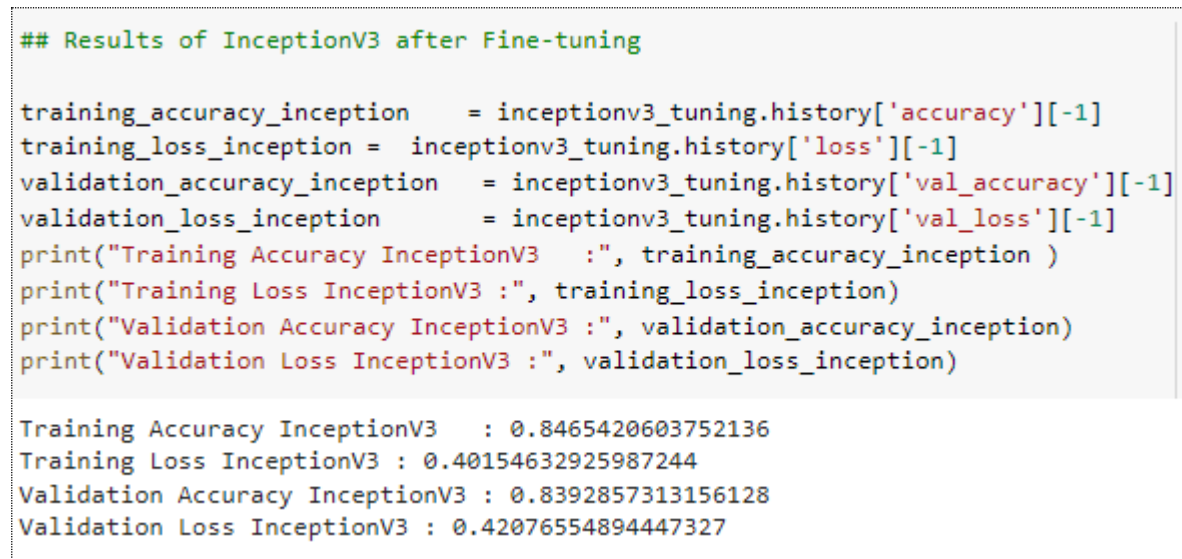


Fig 29. Accuracies of InceptionV3

```

Epoch 1/10
230/230 [=====] - 99s 432ms/step - loss: 0.4196 - accuracy: 0.8386 - val_loss: 0.4247 - val_accuracy: 0.8337
Epoch 2/10
230/230 [=====] - 99s 431ms/step - loss: 0.4270 - accuracy: 0.8394 - val_loss: 0.4345 - val_accuracy: 0.8326
Epoch 3/10
230/230 [=====] - 99s 430ms/step - loss: 0.4241 - accuracy: 0.8412 - val_loss: 0.4421 - val_accuracy: 0.8270
Epoch 4/10
230/230 [=====] - 99s 432ms/step - loss: 0.4057 - accuracy: 0.8486 - val_loss: 0.4157 - val_accuracy: 0.8304
Epoch 5/10
230/230 [=====] - 99s 429ms/step - loss: 0.4158 - accuracy: 0.8418 - val_loss: 0.3944 - val_accuracy: 0.8493
Epoch 6/10
230/230 [=====] - 100s 433ms/step - loss: 0.4116 - accuracy: 0.8448 - val_loss: 0.3783 - val_accuracy: 0.8482
Epoch 7/10
230/230 [=====] - 99s 431ms/step - loss: 0.4082 - accuracy: 0.8441 - val_loss: 0.4208 - val_accuracy: 0.8460
Epoch 8/10
230/230 [=====] - 99s 431ms/step - loss: 0.4169 - accuracy: 0.8435 - val_loss: 0.4332 - val_accuracy: 0.8504
Epoch 9/10
230/230 [=====] - 99s 429ms/step - loss: 0.4036 - accuracy: 0.8471 - val_loss: 0.4323 - val_accuracy: 0.8337
Epoch 10/10
230/230 [=====] - 99s 430ms/step - loss: 0.4015 - accuracy: 0.8465 - val_loss: 0.4208 - val_accuracy: 0.8393

```

Fig 30. Epoch Run of InceptionV3

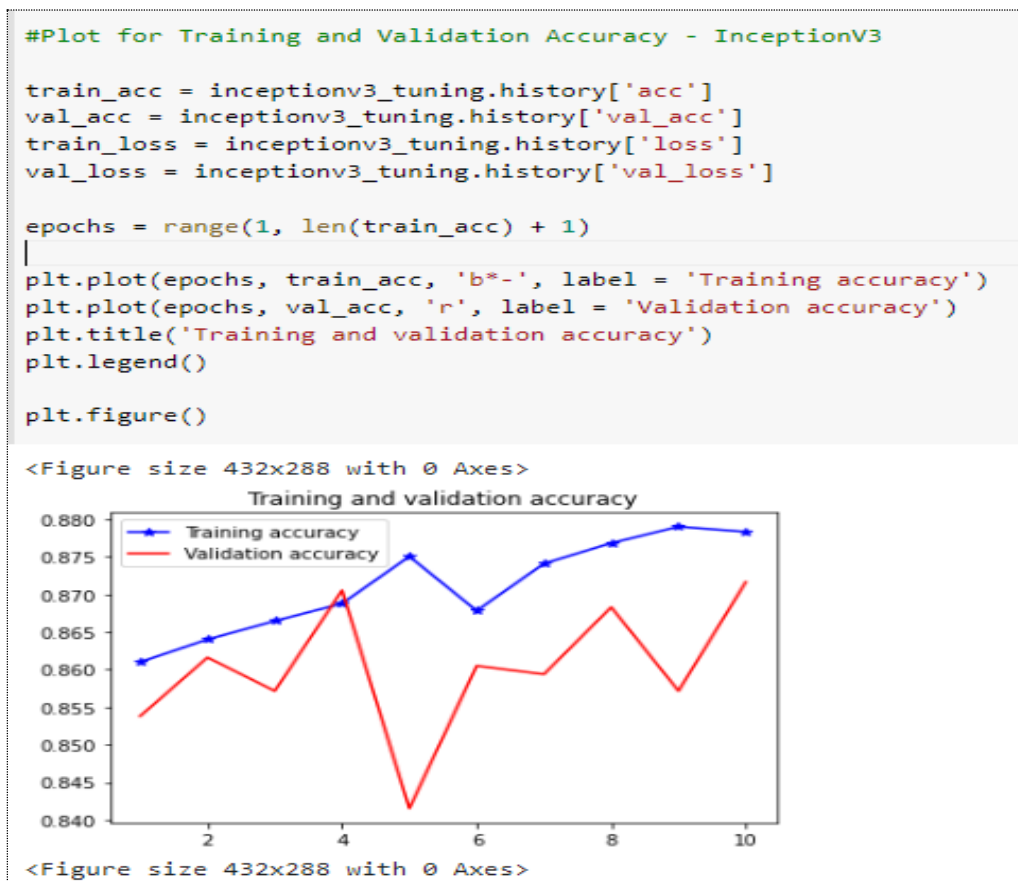


Fig 31. Training-Validation Accuracy Plot – InceptionV3

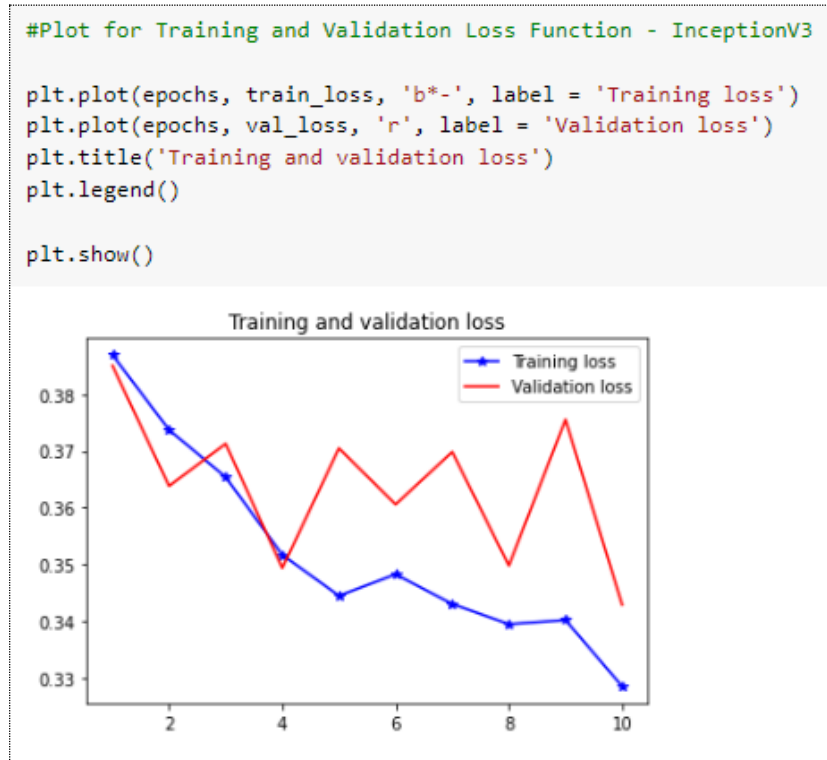


Fig 32. Training-Validation Loss Function Plot – InceptionV3

```
import seaborn as sns

# Generating Confusion Matrix - InceptionV3
Y_pred_inception = model.predict(validation_generator, validation_generator.samples // batch_size+1)
Y_pred_inception = np.argmax(Y_pred_inception, axis=1)
print('Confusion Matrix')
ax= plt.subplot()

conf_matrix_inception = confusion_matrix(validation_generator.classes, Y_pred_inception)
sns.heatmap(conf_matrix_inception, annot=True, ax = ax, fmt="d"); #annot=True to annotate cells
cm_vgg = np.array2string(conf_matrix_inception)
#print(conf_matrix_vgg)

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Galaxy', 'QSO', 'Star']);
ax.yaxis.set_ticklabels(['Galaxy', 'QSO', 'Star']);
```

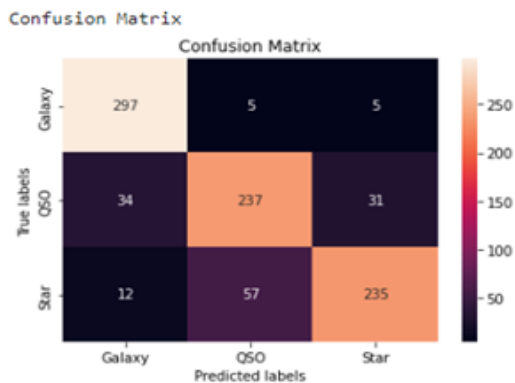


Fig 33. Confusion Matrix – InceptionV3

### 7.3 ResNet50 Model

```
## Results of ResNet50 after Fine-tuning

training_accuracy_resnet = resnet_tuning.history['accuracy'][-1]
training_loss_resnet = resnet_tuning.history['loss'][-1]
validation_accuracy_resnet = resnet_tuning.history['val_accuracy'][-1]
validation_loss_resnet = resnet_tuning.history['val_loss'][-1]
print("Training Accuracy ResNet50 :", training_accuracy_resnet )
print("Training Loss ResNet50 :", training_loss_resnet)
print("Validation Accuracy ResNet50 :", validation_accuracy_resnet)
print("Validation Loss ResNet50| :", validation_loss_resnet)
```

```
Training Accuracy ResNet50 : 0.8172145485877991
Training Loss ResNet50 : 0.5089108347892761
Validation Accuracy ResNet50 : 0.7979910969734192
Validation Loss ResNet50 : 0.6832733750343323
```

Fig 34. Accuracies of ResNet50

```
#Plot for Training and Validation Accuracy - ResNet50 - After Tuning

train_acc = resnet_tuning.history['accuracy']
val_acc = resnet_tuning.history['val_accuracy']
train_loss = resnet_tuning.history['loss']
val_loss = resnet_tuning.history['val_loss']

epochs = range(1, len(train_acc) + 1)

plt.plot(epochs, train_acc, 'b*-', label = 'Training accuracy')
plt.plot(epochs, val_acc, 'r', label = 'Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()

plt.figure()
```

<Figure size 432x288 with 0 Axes>

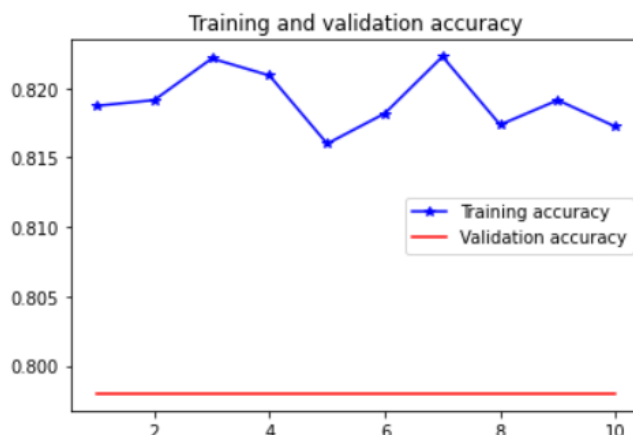


Fig 35: Training-Validation Accuracy Plot - ResNet50

```

Epoch 1/10
230/230 [=====] - 114s 485ms/step - loss: 0.5002 - accuracy: 0.8187 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 2/10
230/230 [=====] - 101s 439ms/step - loss: 0.5035 - accuracy: 0.8191 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 3/10
230/230 [=====] - 101s 437ms/step - loss: 0.5017 - accuracy: 0.8221 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 4/10
230/230 [=====] - 100s 435ms/step - loss: 0.5049 - accuracy: 0.8209 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 5/10
230/230 [=====] - 100s 435ms/step - loss: 0.5019 - accuracy: 0.8160 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 6/10
230/230 [=====] - 100s 437ms/step - loss: 0.4982 - accuracy: 0.8182 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 7/10
230/230 [=====] - 100s 435ms/step - loss: 0.5039 - accuracy: 0.8223 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 8/10
230/230 [=====] - 100s 435ms/step - loss: 0.5028 - accuracy: 0.8174 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 9/10
230/230 [=====] - 100s 436ms/step - loss: 0.4962 - accuracy: 0.8191 - val_loss: 0.6833 - val_accuracy: 0.7980
Epoch 10/10
230/230 [=====] - 100s 433ms/step - loss: 0.5089 - accuracy: 0.8172 - val_loss: 0.6833 - val_accuracy: 0.7980

```

Fig 36: Epoch Run of ResNet50

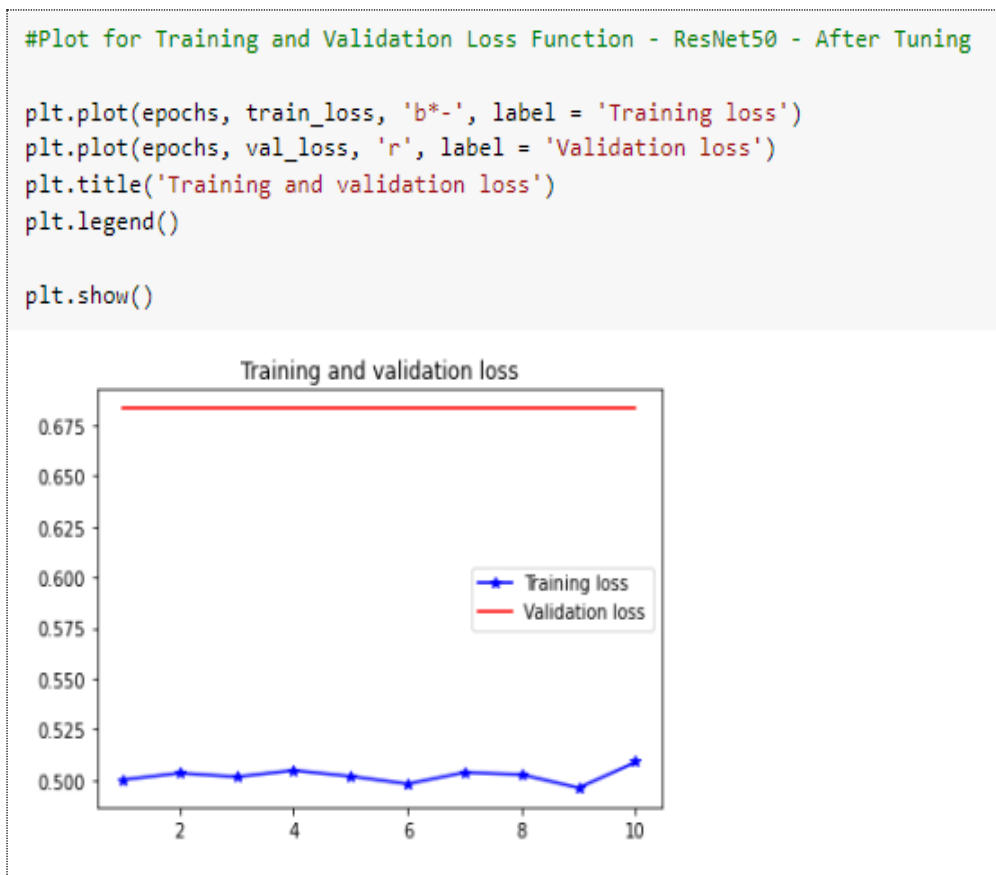


Fig 37. Training-Validation Loss Function Plot – ResNet50

```

import seaborn as sns

# Generating Confusion Matrix - ResNet50

Y_pred_resnet = model.predict(validation_generator, validation_generator.samples // batch_size+1)
Y_pred_resnet = np.argmax(Y_pred_resnet, axis=1)
print('Confusion Matrix')
ax= plt.subplot()

conf_matrix_resnet = confusion_matrix(validation_generator.classes, Y_pred_resnet)
sns.heatmap(conf_matrix_resnet, annot=True, ax = ax, fmt="d"); #annot=True to annotate cells
cm_vgg = np.array2string(conf_matrix_resnet)
#print(conf_matrix_vgg)

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['Galaxy', 'QSO', 'Star']);
ax.yaxis.set_ticklabels(['Galaxy', 'QSO', 'Star']);

```

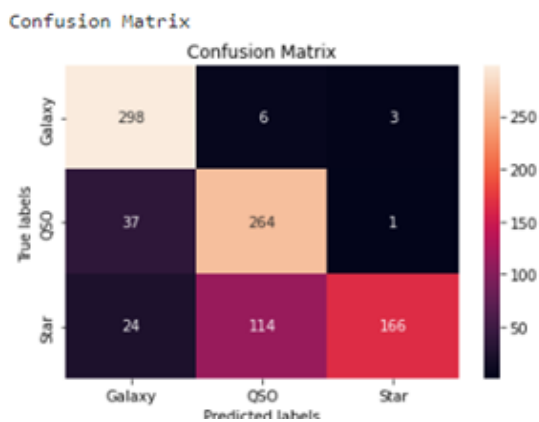


Fig. 38 Confusion Matrix – ResNet50

## 7.4 CNN with Adam Optimizer

```

## Results of CNN (Adam) after Fine-tuning

training_accuracy_cnn = history.history['acc'][-1]
training_loss_cnn = history.history['loss'][-1]
validation_accuracy_cnn = history.history['val_acc'][-1]
validation_loss_cnn = history.history['val_loss'][-1]
print("Training Accuracy CNN with adam optimizer :", training_accuracy_cnn )
print("Training Loss CNN with adam optimizer :", training_loss_cnn)
print("Validation Accuracy CNN with adam optimizer :", validation_accuracy_cnn)
print("Validation Loss CNN with adam optimizer :", validation_loss_cnn)

Training Accuracy CNN with adam optimizer : 0.8480425477027893
Training Loss CNN with adam optimizer : 0.3961517810821533
Validation Accuracy CNN with adam optimizer : 0.7957589030265808
Validation Loss CNN with adam optimizer : 0.5670728087425232

```

Fig 39. Accuracies of CNN (Adam)

```

Epoch 1/10
230/230 [=====] - 114s 495ms/step - loss: 0.3910 - acc: 0.8550 - val_loss: 0.4530 - val_acc: 0.8382
Epoch 2/10
230/230 [=====] - 113s 490ms/step - loss: 0.3956 - acc: 0.8534 - val_loss: 0.4755 - val_acc: 0.8080
Epoch 3/10
230/230 [=====] - 113s 491ms/step - loss: 0.3914 - acc: 0.8524 - val_loss: 0.4862 - val_acc: 0.8348
Epoch 4/10
230/230 [=====] - 113s 490ms/step - loss: 0.3861 - acc: 0.8591 - val_loss: 0.4701 - val_acc: 0.8382
Epoch 5/10
230/230 [=====] - 113s 491ms/step - loss: 0.3817 - acc: 0.8542 - val_loss: 0.5292 - val_acc: 0.7946
Epoch 6/10
230/230 [=====] - 113s 489ms/step - loss: 0.3789 - acc: 0.8580 - val_loss: 0.4441 - val_acc: 0.8415
Epoch 7/10
230/230 [=====] - 113s 489ms/step - loss: 0.3746 - acc: 0.8564 - val_loss: 0.4854 - val_acc: 0.8214
Epoch 8/10
230/230 [=====] - 113s 491ms/step - loss: 0.3702 - acc: 0.8605 - val_loss: 0.4840 - val_acc: 0.8348
Epoch 9/10
230/230 [=====] - 112s 489ms/step - loss: 0.3673 - acc: 0.8605 - val_loss: 0.4489 - val_acc: 0.8438
Epoch 10/10
230/230 [=====] - 113s 490ms/step - loss: 0.3610 - acc: 0.8613 - val_loss: 0.4177 - val_acc: 0.8616

```

Fig 40. Epoch Run of CNN (Adam)

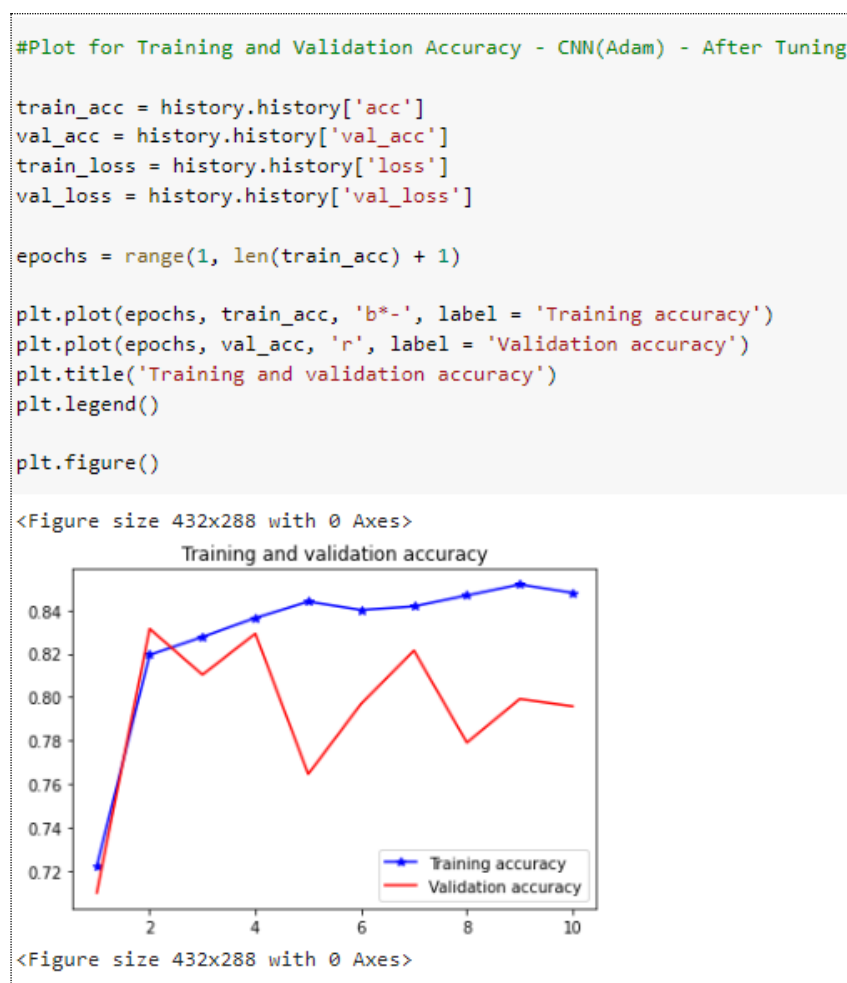


Fig 41. Training-Validation Accuracy Plot – CNN (Adam)

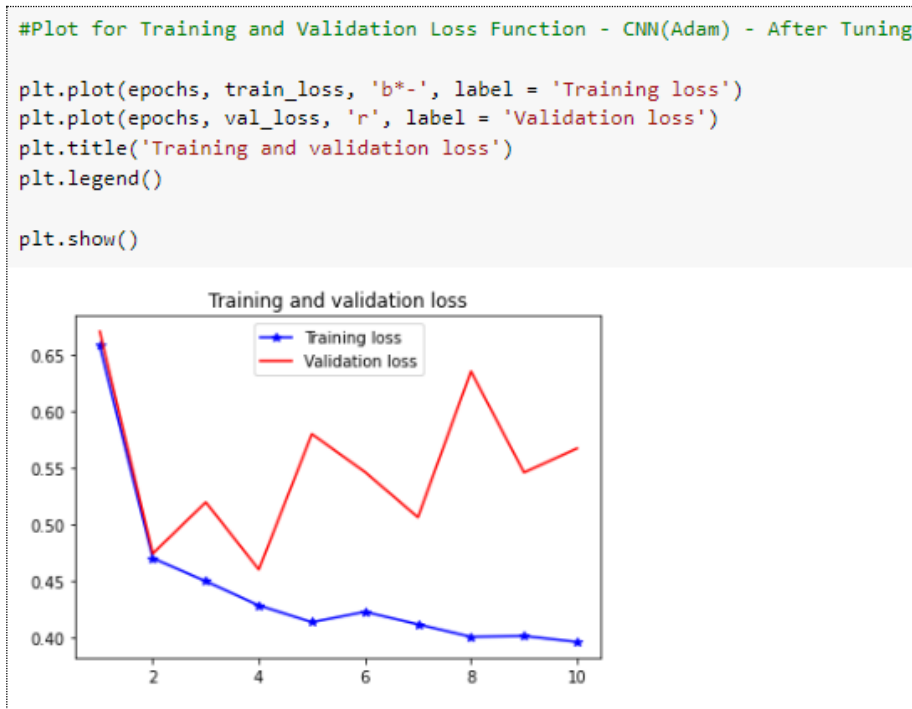


Fig 42. Training-Validation Loss Function Plot – CNN (Adam)

```

# Generating Confusion Matrix - CNN

Y_pred_cnn = model.predict(validation_generator, validation_generator.samples // batch_size+1)
Y_pred_cnn = np.argmax(Y_pred_cnn, axis=1)
print('Confusion Matrix')
ax= plt.subplot()

conf_matrix_cnn = confusion_matrix(validation_generator.classes, Y_pred_cnn)
sns.heatmap(conf_matrix_cnn, annot=True, ax = ax, fmt="d"); #annot=True to annotate cells
cm_cnn = np.array2string(conf_matrix_cnn)
#print(conf_matrix_vgg)

# labels, title and ticks
ax.set_xlabel('Predicted labels');
ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');

```

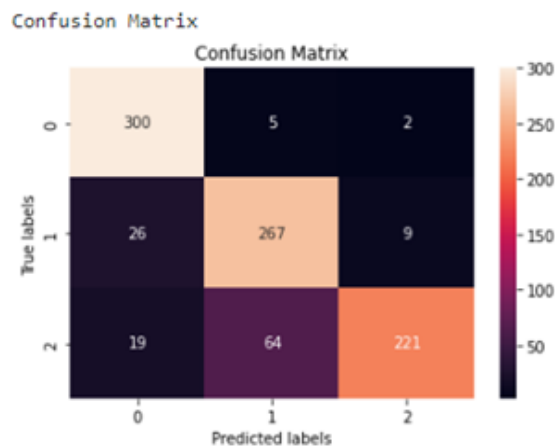


Fig. 43 Confusion Matrix – CNN (Adam)



## References

Sdss.org. 2022. Data Release 17 | SDSS. Available at: <<https://www.sdss.org/dr17/>>