MAR ATHANASIUS COLLEGE OF ENGINEERING
(Affiliated to APJ Abdul Kalam Technological University, TVM)
KOTHAMANGALAM

Department of Computer Applications

Mini Project Report

STAR-GALAXY CLASSIFICATION

USING DEEP LEARNING

Done by

Ajay Das M

Reg No : MAC22MCA-2014

Under the guidance of
Prof. Nisha Markose

2022-2024

MAR ATHANASIUS COLLEGE OF ENGINEERING
(Affiliated to APJ Abdul Kalam Technological University, TVM)
KOTHAMANGALAM

CERTIFICATE

Star-Galaxy Classification Using Deep Learning

Certified that this is the bonafide record of project work done by

Ajay Das M
Reg No: MAC22MCA-2014

during the third semester, in partial fulfilment of requirements for award of the degree

Master of Computer Applications

of

APJ Abdul Kalam Technological University Thiruvananthapuram

Faculty Guide   **Head of the Department**

Prof. Nisha Markose   Prof. Biju Skaria

Project Coordinator   **Internal Examiners**

AsstProf. Sonia Abraham

ACKNOWLEDGEMENT

ABSTRACT

The challenge of accurately classifying astronomical objects as stars or galaxies has been a fundamental task in astrophysics for centuries. Traditional methods relied heavily on visual inspection and morphological analysis, which were labour-intensive and limited by human subjectivity and the capacity to process large data volumes. With the advent of modern sky surveys like the Sloan Digital Sky Survey (SDSS), the volume of astronomical data has grown exponentially, rendering manual classification impractical.

The literature survey across the reviewed papers highlights three algorithms Convolution Neural Network (CNN), deep convolutional neural networks (ConvNets), ContextNet where taken into consideration.

The performance of deep learning architecture Convolution Neural Network (CNN) is used to classify stars and galaxies. Steps include rejecting data with errors, correcting for extinction, aligning images, and centring objects using nMontage and SExtractor.

The Dataset is taken from the Kaggle repository, the dataset contains 3986 data which 942 galaxy 3044 Star data.

Among the three Architecture, the Convolution Neural Network (CNN) is found to be best in terms of model building and computation. Thus, Star-Galaxy Classification Using Deep learning offers significant benefits for star-galaxy classification, including reduced human error, increased scalability, and efficient handling of vast data quantities.

LIST OF TABLES

LIST OF FIGURES

CONTENTS

1. Introduction

The challenge of accurately classifying astronomical objects as stars or galaxies has been a fundamental task in astrophysics for centuries. Traditional methods relied heavily on visual inspection and morphological analysis, which were labour-intensive and limited by human subjectivity and the capacity to process large data volumes. With the advent of modern sky surveys like the Sloan Digital Sky Survey (SDSS), the volume of astronomical data has grown exponentially, rendering manual classification impractical.

The literature survey across the reviewed papers highlights three algorithms Convolution Neural Network (CNN), deep convolutional neural networks (ConvNets), ContextNet where taken into consideration.

The performance of deep learning architecture Convolution Neural Network (CNN) is used to classify stars and galaxies. Steps include rejecting data with errors, correcting for extinction, aligning images, and centring objects using nMontage and SExtractor.

The Dataset is taken from the Kaggle repository, the dataset contains 3986 data which 942 galaxy 3044 Star data.

Among the three Architecture, the Convolution Neural Network (CNN) is found to be best in terms of model building and computation. Thus, Star-Galaxy Classification Using Deep learning offers significant benefits for star-galaxy classification, including reduced human error, increased scalability, and efficient handling of vast data quantities.

2. SUPPORTING LITERATURE

2.1 Literature Review

Paper 1: *Ganesh Ranganath Chandrasekar Iyer Krishna Chaithanya Vastare (2017). Deep Learning for Star-Galaxy Classification*

This project explores a CNN-based classifier to address these limitations. The paper "Deep Learning for Star-Galaxy Classification" (2017) demonstrates that Convolutional Neural Networks (CNNs) can effectively distinguish between stars and galaxies in astronomical images, achieving higher accuracy than traditional methods.

Table 2.1.1. Summary of Paper 1

Title of the paper	Ganesh Ranganath Chandrasekar Iyer Krishna Chaithanya Vastare (2017). Deep Learning for Star-Galaxy Classification
Area of work	Using deep learning, specifically Convolutional Neural Networks (CNNs), for classifying stars or galaxies.
Dataset	Dataset was taken from the Sloan Digital Sky Survey (SDSS). The dataset contains 30 million images.
Methodology / Strategy	CNN-based binary star-galaxy classifier involves collecting labelled image data from sources like the SDSS, pre-processing the data by normalizing and resizing images, and splitting it into training, validation, and test sets. A CNN is designed with convolutional and pooling layers for feature extraction, followed by fully connected layers for classification, with a sigmoid output layer for binary classification. The model is trained using binary cross-entropy loss and the Adam optimizer, then evaluated using accuracy, precision, recall, and F1-score metrics. Finally, the trained model is deployed to classify new astronomical data.
Architecture	Convolutional Neural Networks(CNN)
Result/Accuracy	CNN(Convolutional Neural Networks) – 99.19

Paper 2 : *Kim EJ, Brunner RJ. Star-galaxy classification using deep convolutional neural networks. Monthly Notices of the Royal Astronomical Society. 2016 Oct 17:stw2672.*

Kim and Brunner (2016) developed a deep CNN approach for classifying stars and galaxies in astronomical images. Their method improves accuracy by effectively learning from the features in the images, outperforming traditional classification techniques.

Table 2.1.2. Summary of Paper 2

Title of the paper	Kim EJ, Brunner RJ. Star-galaxy classification using deep convolutional neural networks. Monthly Notices of the Royal Astronomical Society. 2016 Oct 17:stw2672.
Area of work	Star-galaxy classification using deep convolutional neural networks.
Dataset	photometric and spectroscopic data sets with different characteristics and compositions. data sets and the image pre-processing steps for retrieving cutout images
Methodology / Strategy	The research uses deep convolutional neural networks (ConvNets) to classify astronomical objects from SDSS and CFHTLenS survey data. The ConvNet, with several convolutional and fully connected layers, employs data augmentation and dropout to reduce over fitting. The study compares ConvNet performance to the Trees for Probabilistic Classifications (TPC) algorithm, focusing on accuracy and probabilistic calibration.
Architecture	Convolutional Neural Networks (ConvNets)
Result/Accuracy	ConvNet - 99.48

Paper 3 : *Kennamer N, Kirkby D, Ihler A, Sanchez-Lopez FJ. ContextNet: Deep learning for star galaxy classification. In International conference on machine learning 2018 Jul 3 (pp. 2582-2590). PMLR.*

The paper titled "ContextNet: Deep Learning for Star Galaxy Classification" presents a framework for classifying stars and galaxies in astronomical images, specifically for data from the Large Synoptic Survey Telescope (LSST)

Table 2.1.3. Summary of Paper 3

Title of the paper	Kennamer N, Kirkby D, Ihler A, Sanchez-Lopez FJ. ContextNet: Deep learning for star galaxy classification. In International conference on machine learning 2018 Jul 3 (pp. 2582-2590). PMLR.
Area of work	The work applies ContextNet Architecture to classify stars and galaxies in astronomical images from ground-based surveys like the LSST
Dataset	The dataset used in the work consists of simulated images from the Large Synoptic Survey Telescope (LSST) observations, generated using the GalSim image simulation package.
Methodology / Strategy	The methodology uses ContextNet, a three-step neural network framework. It includes a local network for individual object features, a global network for comparing features across objects to capture context, and a prediction network that combines these features for classification. This approach handles non-IID data and improves accuracy by leveraging neural network weight replication for variable object numbers in each exposure.
Architecture	**Local Network**: Convolutional Neural Networks (CNNs) **Global Network**: Recurrent Neural Networks (RNNs) **Prediction Network**: Fully Connected Neural Networks (FCNs)
Result/Accuracy	ContextNet - 95%

2.1.4 SUMMARY TABLE

Table 2.1.4. Summary of all the reference papers

REVIEW PAPER	ARCHITECTURE	ACCURACY
Deep Learning for Star-Galaxy Classification	Convolutional Neural Networks(CNN)	99.19
Star-galaxy classification using deep convolutional neural networks	Convolutional Neural Networks (ConvNets)	99.48
ContextNet: Deep Learning for Star Galaxy Classification	ContextNet	95

2.2 Findings and Proposals

From the above three papers, we get to know that different models were used for the classification of Stars and Galaxies. The initial project report on star-galaxy classification using deep learning explores three key research papers that leverage different neural network architectures for this task. The first paper, "Deep Learning for Star-Galaxy Classification" (2017), focuses on using Convolutional Neural Networks (CNNs) to classify astronomical objects. This study utilized a large dataset from the Sloan Digital Sky Survey (SDSS) and demonstrated that CNNs could achieve high accuracy in distinguishing between stars and galaxies, with the model reaching an accuracy of 99.19%. The CNN-based approach was found to be highly effective, emphasizing the strength of deep learning in handling complex classification tasks.

The second paper, "Star-Galaxy Classification Using Deep Convolutional Neural Networks" (2016), by Kim EJ and Brunner RJ, further advanced the use of deep learning by employing deep convolutional neural networks (ConvNets). This study worked with photometric and spectroscopic datasets from the SDSS and CFHTLenS surveys and incorporated techniques like data augmentation and dropout to enhance model performance. The ConvNet model outperformed traditional classification methods, achieving a remarkable accuracy of 99.48%. This research highlighted the potential of deep learning to improve the precision and reliability of astronomical classifications.

The third paper, "ContextNet: Deep Learning for Star-Galaxy Classification" (2018), introduced a more complex architecture known as ContextNet, designed to handle data from the Large Synoptic Survey Telescope (LSST). ContextNet integrates CNNs, Recurrent Neural Networks (RNNs), and Fully Connected Neural Networks (FCNs) to capture both local and global features of astronomical images. Although this model achieved a slightly lower accuracy of 95%, it offered a sophisticated approach to addressing the challenges posed by non-independent and identically distributed (non-IID) data in astronomical surveys. Together, these studies underscore the effectiveness of deep learning, particularly CNNs, in advancing star-galaxy classification.

3. SYSTEM ANALYSIS

3.1. Analysis of Dataset

3.1.1. About the Dataset

The dataset is taken from the Kaggle repository. The dataset contains 3986 sample observations with 942 Galaxies and 3044 Stars photometric data

The dataset contains a collection of astronomical images captured using a 1.3-meter telescope located in Nainital, India. These images feature stars, galaxies, and other celestial objects. Researchers and data scientists can utilize this dataset for various tasks, including star-galaxy classification, object detection, and image analysis. The dataset provides a valuable resource for exploring the cosmos through machine learning and computer vision techniques.

Dataset: https://www.kaggle.com/datasets/divyansh22/dummy-astronomy-data

3.1.2. Explore the Dataset

The dataset contains 3986 sample observations with 942 Galaxies and 3044 Stars photometric data

Fig 3.1 Snapshot of Galaxy class from the dataset

J1252p2640_r_01_10_1824_1055_3.jpg

J1252p2640_r_01_10_1864_578_3.jpg

J1252p2640_r_01_10_1864_1143_3.jpg

J1530p2310_r_01_117_754_3.jpg

J1530p2310_r_01_170_809_3.jpg

J1530p2310_r_01_171_443_3.jpg
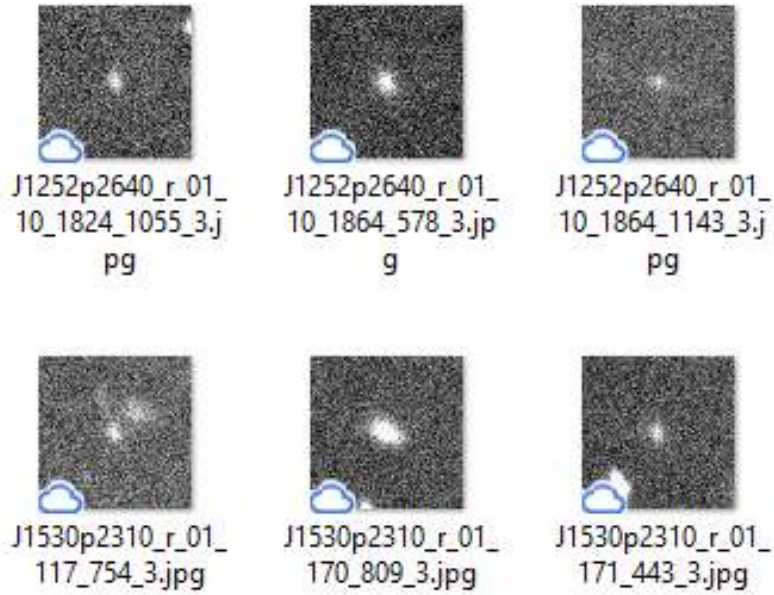
Fig 3.2 Snapshot of Star class from the dataset

grb0422a_01_68_1020_6.jpg

grb0422a_01_77_353_6.jpg

grb0422a_01_83_1747_6.jpg

grb0422a_01_149_1105_6.jpg

grb0422a_01_190_697_6.jpg

grb0422a_01_194_1113_6.jpg

3.2. Data Preprocessing

Resizing Image

Uniform Image Size: Since CNNs require fixed-size input images, all images in the dataset must be resized to a uniform size. A common choice for this type of classification task is 64x64 or 128x128 pixels, although the size can be adjusted based on the computational resources available and the complexity of the objects in the images.

Aspect Ratio Consideration: Ensure that resizing doesn't distort the images, particularly if the original images have different aspect ratios. In some cases, padding the images to maintain aspect ratios might be necessary.

Normalize

Pixel Value Scaling: CNNs perform better when input data is normalized. Typically, image pixel values are scaled from their original range (0-255 for 8-bit images) to a range of 0-1 or -1 to 1. This is done by dividing the pixel values by 255. Normalization helps in speeding up the convergence during training by ensuring that the input features have a similar scale.

Data Augmentation

Data augmentation artificially increases the size of the training dataset by creating modified versions of images in the dataset. This helps the model generalize better and become more robust to variations.

Technique:

Rotation: Randomly rotate images within a certain range to simulate different orientations of celestial objects.

Flipping: Horizontally or vertically flip the images to introduce symmetry variations.

Zooming: Randomly zoom in on images to simulate different scales.

Brightness/Contrast Adjustments: Modify the brightness and contrast of the images to account for different lighting conditions in the data.

Translation: Shift images horizontally or vertically to simulate positional variance.

Splitting the Database

Training Set: 80% of the dataset is used for training. This is the subset of data the model will learn from.

Test Set: The remaining 20% is reserved for testing the model after training to evaluate its performance on unseen data.

3.3. Analysis of Algorithm

3.3.1 Network Architecture of CNN

CNNs are a class of Deep Neural Networks that can recognize and classify particular features from images and are widely used for analyzing visual images. Their applications range from image and video recognition, image classification, medical image analysis, computer vision and natural language processing.

Convolutional Neural Networks (CNNs) are deep learning models that extract features from images using convolutional layers, followed by pooling and fully connected layers for tasks like image classification.

The main layers of CNN are:

- Input Layer
- Convolution Layer
- Pooling Layer
- Fully-Connected (dense)Layers
- Output Layer
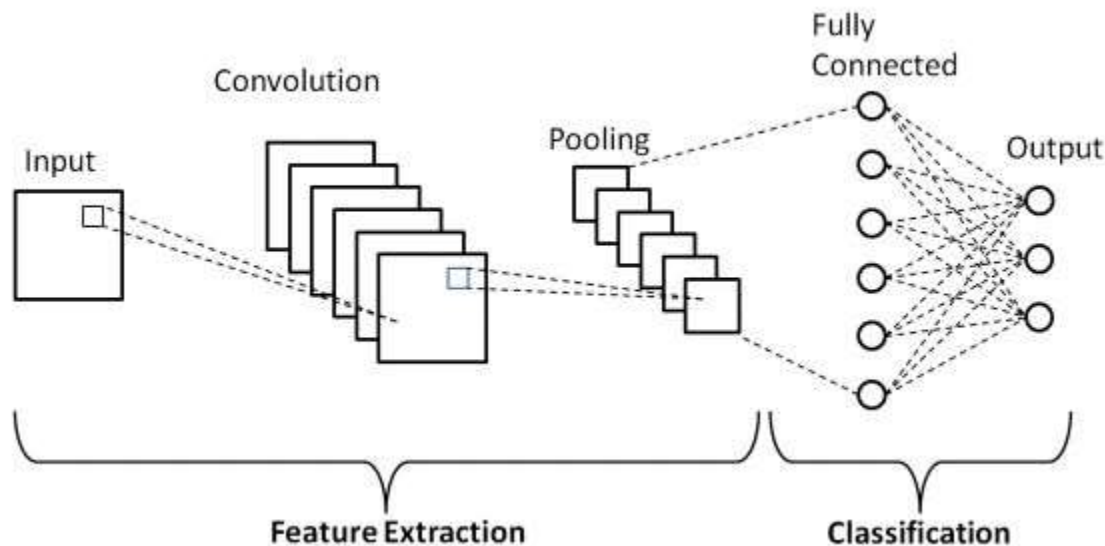
Fig 3.3 *Architecture Diagram*

Layers in CNN Architecture

<u>Input Layer</u>

This layer accepts the raw input images, which is in this case 64 x 64 pixel images in five photometric bands (u, g, r, i, z). This layer provides the initial data (images) to the network, which will be processed and analyzed to distinguish between stars and galaxies.

<u>Convolution Layer</u>

This layer Applies convolution operations to the input data, using a set of filters (kernels) to extract features such as edges, textures, and shapes. These layers detect various features at different levels of abstraction. Early layers might detect basic features like edges, while deeper layers detect more complex structures relevant to differentiating stars from galaxies.

Activation Function(Leaky ReLU)

This is Applies a non-linear transformation to the output of each convolutional layer. Leaky ReLU helps in avoiding the problem of dead neurons by allowing a small, non-zero gradient when the unit is not active. Introduces non-linearity to the model, enabling it to learn from complex data and to improve feature detection and classification.

Pooling Layer

This layer Reduces the spatial dimensions (width and height) of the feature maps, retaining the most critical information while reducing the computational load and controlling overfitting. By reducing the dimensionality, these layers help in abstracting the features detected by convolutional layers and make the network more computationally efficient.

Fully-Connected (Dense) Layers

In this layer each neuron in these layers is connected to every neuron in the previous layer, which allows the network to combine the features extracted by the convolutional and pooling layers and make final predictions. These layers are responsible for the final classification, combining all learned features to distinguish whether an object in the image is a star or a galaxy.

Output Layer

This is the final Layer of the architecture this layer Produces the final output, typically using a softmax function in classification tasks to produce probabilities for each class. This layer outputs the probability of the image belonging to either the "star" or "galaxy" class, allowing for final decision-making in the classification task.
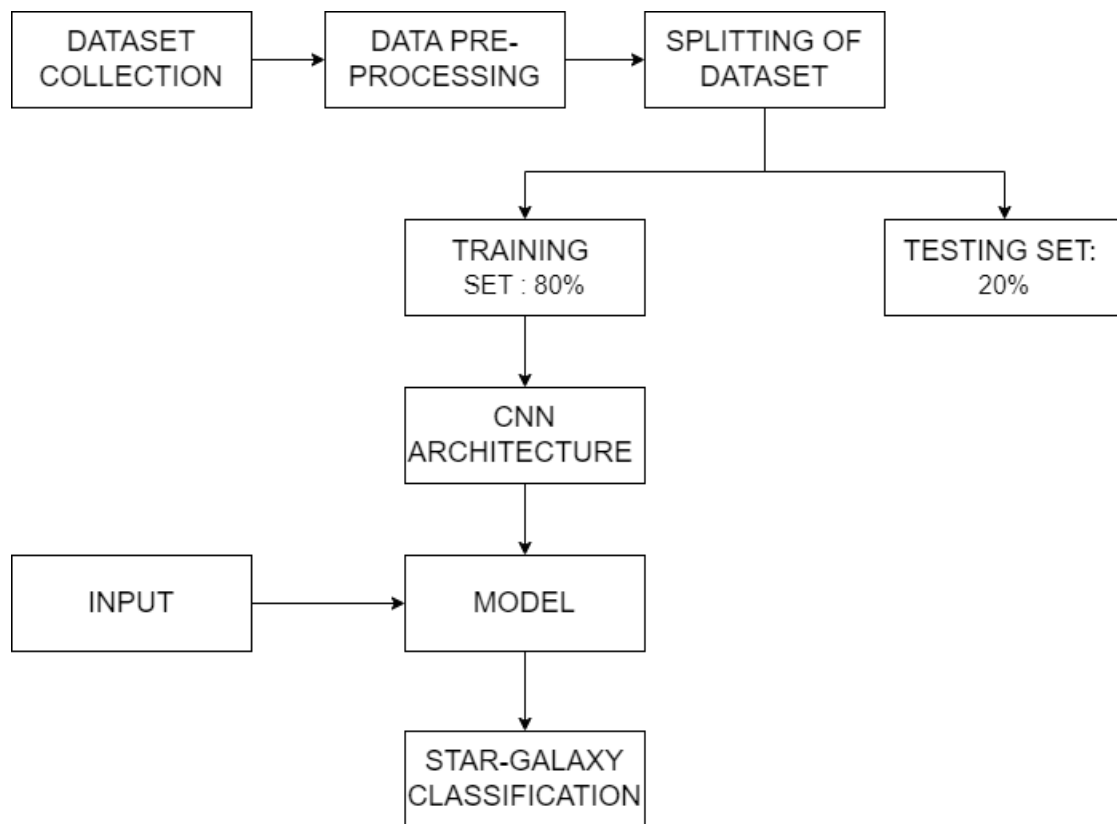
3.3.2 Dimension Table

Table 3.1 Dimension Table of CNN

Type	Filters	Filter Size	Padding	Non-linearity	Initial Weights	Initial Biases
Convolutional	32	5×5	-	Leaky ReLU	Orthogonal	0.1
Convolutional	32	3×3	1	Leaky ReLU	Orthogonal	0.1
Pooling	-	2 x 2	-	-	-	-
Convolutional	64	3 x 3	1	Leaky ReLU	Orthogonal	0.1
Convolutional	64	3 x 3	1	Leaky ReLU	Orthogonal	0.1
Convolutional	64	3 x 3	1	Leaky ReLU	Orthogonal	0.1
Pooling	-	2 x 2	-	-	-	-
Convolutional	128	3 x 3	1	Leaky ReLU	Orthogonal	0.1
Convolutional	128	3 x 3	1	Leaky ReLU	Orthogonal	0.1
Convolutional	128	3 x 3	1	Leaky ReLU	Orthogonal	0.1
Pooling	-	2 x 2	-	-	-	-
Fully-Connected	2048	-	-	Leaky ReLU	Orthogonal	0.01
Fully-Connected	2048	-	-	Leaky ReLU	Orthogonal	0.01
Fully-Connected	2	-	-	Softmax	Orthogonal	0.01

3.4. Project Plan
3.4.1. Project Pipeline

Fig 3.4 Project Pipeline

```
┌─────────────┐     ┌─────────────┐     ┌─────────────┐
│  DATASET    │ ──▶ │ DATA PRE-   │ ──▶ │ SPLITTING OF│
│ COLLECTION  │     │ PROCESSING  │     │  DATASET    │
└─────────────┘     └─────────────┘     └─────────────┘
                                              │
                          ┌───────────────────┴──────────────┐
                          ▼                                   ▼
                  ┌─────────────┐                     ┌─────────────┐
                  │  TRAINING   │                     │ TESTING SET:│
                  │  SET : 80%  │                     │    20%      │
                  └─────────────┘                     └─────────────┘
                          │
                          ▼
                  ┌─────────────┐
                  │    CNN      │
                  │ ARCHITECTURE│
                  └─────────────┘
                          │
  ┌─────────────┐         ▼
  │             │  ┌─────────────┐
  │   INPUT     │─▶│    MODEL    │
  │             │  └─────────────┘
  └─────────────┘         │
                          ▼
                  ┌─────────────┐
                  │ STAR-GALAXY │
                  │CLASSIFICATION│
                  └─────────────┘
```

3.4.2. Project Implementation Plan

➢ Submission of project synopsis with Journal Papers - 22.07.2024
➢ Project proposal approval - 26.07.2024
➢ presenting project proposal before the Approval Committee - 29.07.2024 & 30.07.2024
➢ Initial report submission - 12.08.2024
➢ Analysis and design report submission - 16.08.2024
➢ First project presentation - 21.08.2024 & 23.08.2024
➢ Sprint Release I - 30.08.2024
➢ Sprint Release II - 26.09.2024
➢ Interim project presentation - 30.09.2024 & 01.10.2024
➢ Sprint Release III - 18.10.2024
➢ Submission of the project report to the guide - 28.10.2024
➢ Final project presentation - 28.10.2024 & 29.10.2024
➢ Submission of project report after corrections - 01.11.2024

3.5. Feasibility Analysis

A feasibility study aims to objectively and rationally uncover the strengths and weaknesses of an existing system or proposed system, opportunities and threats present in the natural environment, the resources required to carry through, and ultimately the prospects for success.
Evaluated the feasibility of the system in terms of the following categories:
- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

3.5.1. Technical Feasibility

Technical feasibility evaluates whether the necessary technology for your project is readily available and whether your team has the expertise to effectively implement it.

Data Processing and Analysis

Feasibility: The project involves processing and analyzing astronomical images using libraries like NumPy, Pandas, and TensorFlow. These tools are widely recognized and supported, ensuring technical feasibility.
Expertise: Proficiency with these tools, as demonstrated by their application in the project, indicates technical competence.

Deep Learning Algorithms

Feasibility: The use of Convolutional Neural Networks (CNNs) for classifying images as either stars or galaxies is technically feasible and a common approach in image classification tasks.
Expertise: The implementation of CNN architecture demonstrates a sound understanding of deep learning concepts and methodologies.

Data Visualization

Feasibility: The project utilizes Matplotlib and Seaborn for visualizing astronomical data, indicating its reliance on well-established and technically feasible visualization tools.
Expertise: The team's capability to visualize data trends suggests proficiency in using these tools effectively.

File Handling
Feasibility: Reading and writing data to CSV files using Pandas is a standard practice, ensuring technical feasibility.
Expertise: File handling is a fundamental skill, and your team's successful implementation of it implies technical competence.

3.5.2. Economic Feasibility
Economic feasibility evaluates whether the project is financially viable, considering associated costs and potential benefits

Development Costs

Feasibility: The use of open-source tools and libraries reduces software acquisition expenses. However, potential costs related to personnel, training, and hardware should be considered.
Benefits: The benefits of a well-functioning classification system, such as accurate identification of celestial objects, may justify the development costs by enhancing the efficiency of astronomical research.

Maintenance Costs

Feasibility: Open-source tools typically have lower maintenance expenses. Nonetheless, ongoing costs related to data updates, algorithm enhancements, and support should be factored in.
Benefits: The long-term benefits, such as consistent model improvements and reliable classification, should outweigh maintenance costs by significantly contributing to the field of astronomy.

3.5.3. Operational Feasibility
Operational feasibility evaluates whether the project aligns with operational requirements and can be smoothly integrated into existing processes.

User Acceptance
Feasibility: The project aims to enhance user experience by accurately classifying celestial objects as either stars or galaxies, contributing to user acceptance among astronomers and researchers.

Integration: Seamless integration with existing astronomical data analysis platforms would enhance operational feasibility.

Scalability
Feasibility: Consider scalability issues related to the growth in the number of images processed. Implementing deep learning techniques like CNNs may face challenges with scalability.
Mitigation: Explore strategies such as distributed computing or optimizing the CNN architecture to address scalability concerns.

Ease of Use

Feasibility: If the system is designed with a user-friendly interface and easy navigation, it enhances operational feasibility.
Training: Assess the ease with which users can understand and interact with the classification system.

Software Environment

The software environment leverages the Python programming language and key libraries for data processing, deep learning, and visualization. Development can be facilitated through Colab Notebooks.

Hardware Environment

The hardware environment requires a standard computing setup with ample RAM and storage. Optional utilization of cloud services for scalability ensures a robust and flexible system environment for development, testing, and potential deployment of the classification system.

3.6. System Environment
3.6.1. Software Environment

The software environment encompasses a comprehensive array of tools, frameworks, and platforms essential for the development and execution of the StarGalaxy Classification project. This environment ensures that every aspect of the project, from data processing to model deployment, is well-supported and efficient.

Programming Languages
Python: The primary programming language employed in this project is Python. Python is renowned for its versatility and the extensive libraries it offers, making it a cornerstone in data science and deep learning. The language's syntax is both accessible and powerful, providing the necessary tools to handle complex computational tasks with ease. Its strong support community and continuous development ensure that it remains a top choice for scientific and machine learning projects.

Data Processing and Analysis
NumPy and Pandas: These libraries are integral to the project for efficient data manipulation, handling, and analysis. NumPy provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. Pandas offers data structures and data analysis tools that are easy to use and highly efficient, making it indispensable for manipulating numerical tables and time series data.

TensorFlow: A key player in the project, TensorFlow is utilized for implementing deep learning algorithms, specifically Convolutional Neural Networks (CNNs) for image classification. TensorFlow's robust capabilities in numerical computation using data flow graphs make it ideal for this project, ensuring the models are both powerful and scalable.

Deep Learning Libraries
TensorFlow: In addition to its role in data processing, TensorFlow is crucial for implementing and training the deep learning models employed in this project. The flexibility and comprehensive ecosystem of tools offered by TensorFlow facilitate the development and optimization of complex neural networks.

SciPy: Complementing NumPy, SciPy provides additional functionality for scientific computing, including modules for optimization, integration, interpolation, eigenvalue problems, algebraic equations, and other tasks. This makes SciPy a vital part of the computational toolkit used in this project.

Data Visualization
Matplotlib and Seaborn: These libraries are chosen for their powerful capabilities in creating insightful visualizations. Matplotlib is a plotting library that produces publication-quality figures in a variety of formats and interactive environments. Seaborn builds on Matplotlib and introduces additional functionality and aesthetic improvements, making it easier to generate attractive and informative statistical graphics. Together, they aid in the exploration and communication of data patterns, allowing for effective interpretation and presentation of results.

File Handling

Pandas: For reading and writing data to CSV files, Pandas is utilized. This ensures seamless data management, allowing for efficient storage, retrieval, and manipulation of the datasets involved in the project. The ability to easily convert complex data structures into manageable formats is crucial for the smooth operation of data processing workflows.

Development Environment

Google Colab: Colab is a free Jupyter notebook environment that runs entirely in the cloud. It allows users to write and execute code in Python, leveraging cloud-based resources to handle computationally intensive tasks. Colab supports many machine learning libraries and provides a platform for collaborative development, making it ideal for team-based projects and prototyping.

Visual Studio Code: Visual Studio Code is a streamlined code editor that supports various development operations, including debugging, task running, and version control. It provides the necessary tools for quick code-build-debug cycles and leaves more complex workflows to fuller-featured Integrated Development Environments (IDEs) like Visual Studio IDE. The lightweight nature and powerful extensions of Visual Studio Code make it a preferred choice for many developers.

HTML and CSS

HTML (HyperText Markup Language): HTML is used for creating web pages and describing the structure of the user interface of the project. It provides the foundation for building the visual aspects of the web application, allowing for the organization and presentation of content in a structured manner.

CSS (Cascading Style Sheets): CSS is used alongside HTML to style web pages, enabling developers to create visually appealing interfaces. By defining the look and layout of web pages, CSS helps ensure a consistent and user-friendly experience.

Flask

Flask: Flask is a micro web framework written in Python that facilitates the development of web applications. It includes features like URL routing and a template engine and is WSGI-compliant. Flask's simplicity and flexibility make it an excellent choice for developing web applications, providing a foundation for integrating backend functionality with the frontend user interface.

GitHub

Git: Git is an open-source version control system that tracks file changes efficiently and ensures file integrity. GitHub, built on Git, provides a collaborative platform where project revisions can be discussed publicly. This allows a community of experts to contribute knowledge and advance the project. GitHub's social networking aspect is one of its most powerful features, fostering collaboration and continuous improvement. The platform's ability to store file changes efficiently and ensure file integrity makes it a preferred version control system for developers.

3.7.2. Hardware Environment

The hardware environment refers to the physical infrastructure necessary to support the development and deployment of the Star-Galaxy Classification system.

Computing Resources
Processor: A powerful multi-core processor with at least a 2 GHz clock speed is recommended to handle the computational demands efficiently. For optimal performance, consider a processor with at least a quad-core architecture.

Storage: Sufficient Disk Space: A 512 GB Solid State Drive (SSD) is recommended to ensure quick data access and storage efficiency.

Memory (RAM): 8 GB RAM: Adequate RAM is crucial for efficiently handling large astronomical datasets and performing deep learning tasks.

Internet Connectivity
High-Speed Internet: A stable and high-speed internet connection is essential for accessing online resources, libraries, and datasets during development. It ensures smooth collaboration and uninterrupted access to cloud-based services and tools.

4. SYSTEM DESIGN

4.1. Model Building

4.1.1. Model Planning

The primary aim of the model planning phase in the star-galaxy classification project is to establish a comprehensive framework that guides subsequent model-building activities. This phase involves defining the scope, selecting appropriate algorithms, and outlining the overall strategy for accurately classifying astronomical objects as stars or galaxies using deep learning techniques.

1. Objective:
Develop a star-galaxy classification system using Convolutional Neural Network (CNN) to classify astronomical objects based on their photometric data

2. Approach:
Utilize CNN architecture, which has proven effective for image classification tasks. The model will be trained on a dataset containing images of stars and galaxies, preprocessing the data for improved accuracy.

2. Approach:
Utilize CNN architecture, which has proven effective for image classification tasks. The model will be trained on a dataset containing images of stars and galaxies, preprocessing the data for improved accuracy.

3. Data Preparation:
 Import and preprocess the astronomical images. Handle any errors in the data, correct for extinction, align images, and center the objects using tools like nMontage and SExtractor. Rescale images, apply random transformations for data augmentation to enhance model robustness

4.1.2 Training
The Convolutional Neural Network (CNN) model is constructed using a deep learning approach to extract features from images and classify them accordingly. Below are the detailed steps involved in the model-building process, including model architecture and training procedure using the training dataset.

Step 1: Importing Required Libraries

To commence the model-building process, it is essential to import all necessary libraries, such as TensorFlow and Keras. These libraries are fundamental for constructing and training the CNN model, offering extensive functionalities and support for deep learning applications.

Step 2: Convolutional Neural Network Model Architecture
The CNN model is meticulously designed using a structured sequence of layers, each contributing to the overall functionality and performance of the model:

1. **Input Layer**: This layer receives the input images, setting the stage for subsequent layers to process the data.

2. **Convolution Layer 1**: This layer consists of 32 filters, each with a 3x3 kernel, and employs the ReLU activation function to introduce non-linearity.

3. **MaxPooling Layer 1**: This layer utilizes a 2x2 pool size to reduce the spatial dimensions of the feature maps, thereby minimizing computational complexity.

4. **Convolution Layer 2**: This layer is composed of 64 filters, each with a 3x3 kernel, and also employs the ReLU activation function for enhanced feature extraction.

5. **MaxPooling Layer 2**: This layer again uses a 2x2 pool size to further reduce the spatial dimensions of the feature maps.

6. **Convolution Layer 3**: This layer includes 128 filters, each with a 3x3 kernel, and employs the ReLU activation function to capture more complex features.

7. **MaxPooling Layer 3**: This layer uses a 2x2 pool size to provide additional dimensionality reduction.

8. **Flatten Layer**: This layer converts the 2D feature maps into a 1D vector, facilitating the transition from convolutional layers to fully connected layers.

9. **Dense Layer 1**: This layer contains 128 units and employs the ReLU activation function to learn intricate patterns and representations.

10. **Dropout Layer 1**: Operating with a 50% dropout rate, this layer helps prevent overfitting by randomly deactivating neurons during training.

11. **Dense Layer 2**: This layer comprises 64 units and employs the ReLU activation function, further refining the learned patterns.

12. **Dropout Layer 2**: Similar to the previous dropout layer, this one also operates with a 50% dropout rate to mitigate overfitting risks.

13. **Output Layer**: This final layer consists of 2 units with the Softmax activation function, providing the classification output as probabilities for each class.

Step 3: Model Compilation and Class Weights
Upon completion of the model architecture, the next step is to compile the model. This is achieved using the Adam optimizer, which is known for its efficiency and adaptability in deep learning applications. A learning rate schedule is implemented to gradually decrease the learning rate during training, thereby enhancing the optimization process. The loss function utilized is categorical cross-

entropy, which is suitable for multi-class classification tasks. Additionally, class weights are computed to address any imbalance present within the dataset, ensuring that each class is appropriately represented during the training process..

- **Learning Rate Schedule**: Helps in gradually decreasing the learning rate during training.

- **Class Weights**: Computed to balance the influence of different classes during training.

Step 4: Model Training

The model undergoes training utilizing the dataset, with validation loss monitoring to avert overfitting through early stopping mechanisms..

- **Early Stopping**: A callback is used to stop training when validation loss does not improve for a specified number of epochs.

4.1.3 Testing

Model Performance

The Convolutional Neural Network model was evaluated using a test dataset to classify images into two categories: Star and Galaxy. The testing phase evaluates the model's performance and its ability to generalize to unseen data, which is crucial for understanding the model's effectiveness in real-world applications. The following are the key metrics and results obtained from this evaluation

Testing Metrics
- **Test Accuracy:** The model achieved an accuracy of 88.59% on the test dataset, indicating its reliability in distinguishing between stars and galaxies**.**

- **Test Loss**: The test loss was recorded at 28.75%, which is reflective of the model's performance during the testing phase.

Training Metrics
- **Training Accuracy**: During the training phase, the model achieved an accuracy of 87.72%. This demonstrates the model's ability to learn from the training data effectively.

- **Training Loss**: The training loss was measured at 29.91%, providing insight into the error rate during the model's training.

Evaluation Matrix:

The model's performance was further assessed using various evaluation metrics, as detailed in the table below

Table 4.1 Evaluation Matrix

Metric	Galaxy	Star	Accuracy	Macro Average	Weighted Average
Precision	0.24	0.76		0.50	0.64

Recall	0.21	0.79		0.50	0.66
F1-Score	0.23	0.78	0.66	0.50	0.65
Support	189	609	798	798	798

Analysis

Precision: The precision for the "Galaxy" class is 0.24, and for the "Star" class, it is 0.76. This indicates the model's ability to correctly identify positive instances of each class.

Recall: The recall for the "Galaxy" class is 0.21, and for the "Star" class, it is 0.79, showing the model's ability to capture the relevant instances of each class.

F1-Score: The F1-Score for the "Galaxy" class is 0.23, while for the "Star" class, it is 0.78. This metric provides a balance between precision and recall.

Support: The support values indicate the number of actual instances for each class: 189 for "Galaxy" and 609 for "Star."

Learning Curves:

The learning curves for the model accuracy and model loss over the training and validation datasets are depicted in Figure 4.2. These curves provide a visual representation of how the model's performance evolved during the training process.

Fig 4.2 Model Accuracy and Model Loss Curves
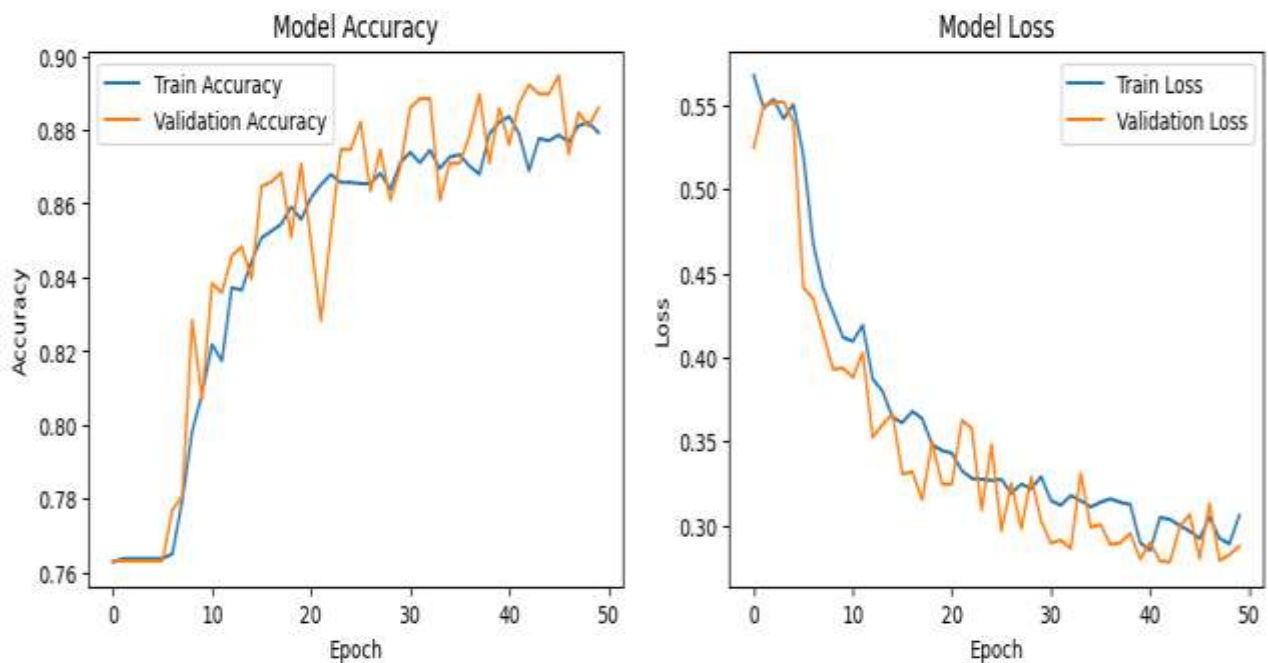
5. RESULTS AND DISCUSSION

The following code demonstrates the process of loading a pre-trained Convolutional Neural Network (CNN) model and utilizing it to classify astronomical images into two categories: stars and galaxies. The code encompasses essential steps, including model loading, image preprocessing, prediction, and result visualization.

In the initial phase, the necessary libraries are imported, including *NumPy* for numerical operations, *Matplotlib* for plotting, and *TensorFlow/Keras* for deep learning functionalities. The trained CNN model is then loaded using the *load_model* function, which retrieves the saved model from the specified file path.

Fig 5.1 Source code loading the model

```python
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model

# Load your trained model
model = load_model('star_galaxy_classifier.h5')
```

The *load_and_preprocess_image* function is defined to load and preprocess the input image for prediction. The image is resized to the target dimensions (64x64 pixels) and converted to grayscale. The image is then transformed into an array, expanded to create a batch of size one, and rescaled by dividing by 255.0 to normalize the pixel values.

Fig 5.2 Source code load_and_preprocess_image function

```python
def load_and_preprocess_image(img_path):
    """Load and preprocess the image for prediction."""
    img = image.load_img(img_path, target_size=(64, 64), color_mode='grayscale')
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)  # Create a batch of size 1
    img_array /= 255.0  # Rescale the image
    return img_array
```

The *predict_image* function is responsible for predicting the class of the input image. It preprocesses the image using the previously defined function, makes predictions using the loaded model, and determines the predicted class by selecting the class with the highest probability

Fig 5.3 Source code for predict_image function

```python
def predict_image(img_path):
    """Predict the class of the input image."""
    preprocessed_image = load_and_preprocess_image(img_path)
    predictions = model.predict(preprocessed_image)
    predicted_class = np.argmax(predictions, axis=1)
    return predicted_class, predictions
```

In this section, an example image path is specified, and the *predict_image* function is utilized to predict the class of the specified image. The prediction results, including the predicted class and the associated probabilities, are printed to the console.

Fig 5.4 Source code printing the output

```
# Example of using the prediction function
img_path = 'dataset1/test/star/grb0422a_01_1465_1314_6.jpg'
predicted_class, predictions = predict_image(img_path)

# Display the prediction results
class_labels = list(train_generator.class_indices.keys())
print(f"Predicted Class: {class_labels[predicted_class[0]]}")
print(f"Prediction Probabilities: {predictions}")
```

Finally, the optional code block displays the input image along with the predicted class. The image is loaded and displayed using Matplotlib, with the title indicating the predicted class.

Fig 5.1 Source code printing the image in the output

```
# Optional: Display the input image
plt.imshow(image.load_img(img_path, color_mode='grayscale'))
plt.title(f"Predicted: {class_labels[predicted_class[0]]}")
plt.axis('off')
plt.show()
```

The output is:
Upon running the prediction code, the model processed the input image to classify it into one of the two categories: star or galaxy. The model's inference step resulted in the following output:
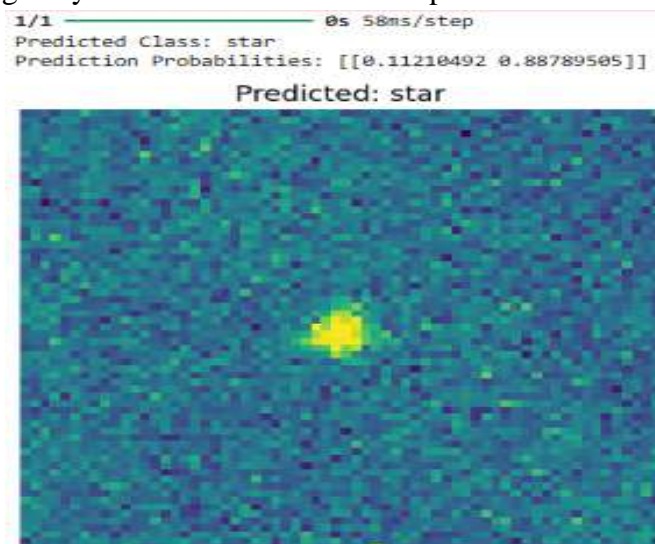
This output indicates that the model successfully processed the image within 58 milliseconds per step. The results show that the predicted class for the input image is "star," with a prediction probability of approximately 88.79% for the star class and 11.21% for the galaxy class. This high probability value suggests that the model is confident in its prediction for this particular instance.

6. MODEL DEPLOYMENT

The principal objective of model deployment in this project is to seamlessly integrate the star-galaxy classification models, which were meticulously developed, into a production environment. This critical transition enables end-users to access and utilize the classification system, thereby making informed decisions based on the model's predictions.

Model deployment is a crucial phase that effectively bridges the gap between the theoretical development of the model and its practical, real-world application. By deploying the classification models, this project aims to provide a tangible, accessible, and reliable solution that meets the needs and preferences of its target users. The deployment process ensures that the sophisticated algorithms developed during the project are fully operational and can be leveraged to their full potential by end-users.

To facilitate the deployment process, the project employs various frameworks and libraries, such as Flask for creating web applications and TensorFlow/Keras for managing the deep learning models. These tools are instrumental in ensuring the smooth and efficient integration of the models into the production environment. They contribute significantly to the overall effectiveness of the deployment, ensuring that the system is robust, scalable, and user-friendly.

By leveraging these frameworks and libraries, the deployment process is streamlined, allowing the star-galaxy classification system to be accessed seamlessly by end-users. This approach guarantees that the theoretical advancements and model optimizations translate into practical tools that can handle real-world data effectively. The deployment phase ensures that the developed models are not just theoretical constructs but are practical solutions providing meaningful insights and accurate classifications, thereby fulfilling the project's objectives and serving the end-users' requirements comprehensively.

User Interface (UI):

The star-galaxy classification system is meticulously integrated into the user interface, ensuring a seamless and user-friendly experience for its users. The system is designed to provide easy navigation and interaction with the classification tool through an intuitive web interface.

Upon accessing the application, users are greeted with a clean and user-friendly introduction page titled 'Star-Galaxy Classification System'. This introduction page provides a brief yet comprehensive overview of the system's capabilities, guiding users on how to proceed with the classification process.

Main Interface
The main interface is thoughtfully designed with several key elements to facilitate straightforward interaction. Initially, users are presented with the primary page, where they have the option to upload their image for classification. To proceed, users need to select the image file from their device using the provided file upload input.

Classification Process

Once the image file is selected, users simply need to click on the "Classify" button to initiate the classification process. The application promptly processes the uploaded image utilizing the pre-trained Convolutional Neural Network (CNN) model, and subsequently, the classification results are displayed to the user.

Result Page

The result page is designed to present the classification results in an accessible and informative manner. It showcases the predicted class of the image, indicating whether the image is classified as a star or a galaxy. Additionally, the actual image itself is displayed on this page, allowing users to visually verify the classification result.

Interactive Features

To enhance user interaction, the result page includes a convenient button labeled "Upload Another Image". This button allows users to return to the upload page, where they can repeat the classification process with a new image. This feature ensures continuous engagement and provides flexibility for users who wish to classify multiple images.

Visual and Functional Design

The overall design of the user interface is simple and easy to understand, ensuring that users, regardless of their technical expertise, can effortlessly navigate and utilize the star-galaxy classification system. The straightforward interaction design begins with the clean introduction page, leading users through a smooth workflow from image upload to result visualization.

Fig 6.1 Star-Galaxy Classification interface Home page

Fig 6.2 Star-Galaxy Classification interface Result page
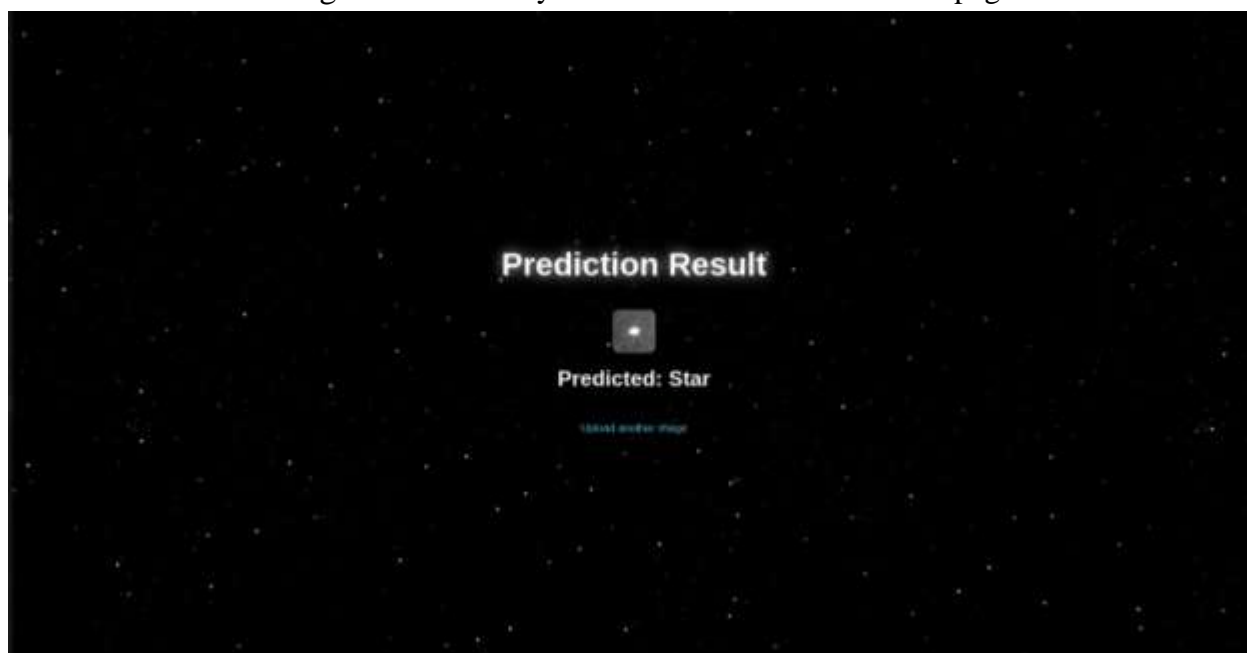
7. GIT HISTORY

The Git repository for the Star-Galaxy Classification System contains all Colab files, Python scripts, HTML files, and three related research papers. This repository is meticulously maintained to ensure a systematic presentation of the project, serving as a comprehensive reference for future work. The repository mainly contains:

Fig 7.1 The whole Git repository

This repository contains the source code and files necessary for building and running the StarGalaxy classification application. The key components of this repository include:

StarGalaxy Interface Folder: This directory holds the essential source code for developing the user interface.

App.py: This script serves as the backend code for the application, enabling its core functionality.

Star_galaxy_classifier.keras: This file contains the saved model post-training, crucial for the classification **process.**

Fig 7.2 StarGalaxy interface folder in the repository

This folder mainly contain two folders

- Static
- Template

Static Folder

The static folder typically contains all the static files used in the application, which don't change frequently. These include:

CSS files: Stylesheets that define the appearance and layout of your application.

Templates Folder

The templates folder houses the HTML files that define the structure of your web pages. These files usually contain placeholders or template tags that are rendered dynamically by the backend framework (like Flask in Python). They allow you to create a consistent look and feel across different parts of your application while inserting dynamic content where needed.

The template folder contains upload.html and result.html

Upload.html: This folder contains the source code for the front page or the home page

Result.html: This folder contains the source code for the result page which is used to view the classification result

Code Folder

The Code folder contains the collaborative notebooks utilized for data pre-processing and model building. These notebooks serve as the foundation for preparing the dataset and constructing the Deep learning models.

Data Folder

The Data folder houses the dataset employed for training the model. This repository of data is essential for feeding the architecture and ensuring the model is trained effectively.

Docs Folder

The Docs folder encompasses the primary documents used throughout the project, such as the project synopsis, presentations (PPT), and other significant documentation that supports the project's development and presentation.

Model Folder

The Model folder consists of various versions of the model, each saved at different stages of the training process. This allows for version control and ensures that all iterations of the model are accessible for review and comparison.

References Paper Folder

The References paper folder contains all the reference papers consulted during the project. These papers provide the theoretical and empirical foundation for the methodologies and approaches adopted in the project.

8. CONCLUSION

In conclusion, the StarGalaxy Classification project has successfully achieved its primary objective of accurately identifying and classifying celestial objects as either stars or galaxies using deep learning techniques. By leveraging Convolutional Neural Network (CNN) architecture, the system excels in discerning intricate patterns within astronomical images to deliver precise classifications. The project's technical contributions, including algorithmic precision and scalability, ensure the system's effectiveness in handling diverse datasets and evolving volumes of data.

The user-centric design incorporates feedback mechanisms and a user-friendly interface, fostering seamless interactions for users engaged in astronomical research and exploration. The successful deployment of production-ready models, API endpoints, and frontend integration underscores the project's transition from theoretical development to practical application. Security measures, such as authentication and authorization, along with real-time monitoring and logging, enhance the system's reliability and stability.

Lessons learned from this project will guide future improvements, with outlined plans for scalability enhancements and feature updates to ensure continuous refinement. The StarGalaxy Classification project, a tangible outcome of deep learning principles and user-centric design, is poised to contribute significantly to the field of astronomy, providing researchers with precise and efficient tools for classifying celestial objects as stars or galaxies in the dynamic landscape of astronomical discovery.

11. REFERENCES

1. Ganesh Ranganath Chandrasekar Iyer Krishna Chaithanya Vastare (2017). Deep Learning for Star-Galaxy Classification
2. Kim EJ, Brunner RJ. Star-galaxy classification using deep convolutional neural networks. Monthly Notices of the Royal Astronomical Society. 2016 Oct 17:stw2672.
3. Kennamer N, Kirkby D, Ihler A, Sanchez-Lopez FJ. ContextNet: Deep learning for star galaxy classification. In International conference on machine learning 2018 Jul 3 (pp. 2582-2590). PMLR.