PROJECT REPORT ON

TRAFFIC SIGNS RECOGNITION USING CNN AND KERAS

Submitted to

Department of Computer Applications

in partial fulfillment for the award of the degree of

MASTER OF COMPUTER APPLICTIONS

### Batch (2023-2025)

***Submitted by***

Name of the student: Anshi

Enrollment Number: GE-23112694

# Under the Guidance of Mr.Sanjay Roka

# Assistant Professor



GRAPHIC ERA DEEMED TO BE UNIVERSITY DEHRADUN

June -2024



**CANDIDATE’S DECLARATION**

I hereby certify that the work presented in this project report entitled “Traffic signs recognition using CNN and Keras**”** in partial fulfillment of the requirements for the award of the degree of Master of Computer Applications is a bonafide work carried out by me during the period of January 2024 to June 2024 under the supervision of Mr. Sanjay Roka, Assistant Professor, Department of Computer Application, Graphic Era Deemed to be University, Dehradun, India.

This work has not been submitted elsewhere for the award of a degree/diploma/certificate.

### Name and Signature of Candidate

This is to certify that the above mentioned statement in the candidate’s declaration is correct to the best of my knowledge.

### Date: Name and Signature of Guide

**Signature of Supervisor Signature of External Examiner HOD**

# CERTIFICATE OF ORIGINALITY

This is to certify that the project report entitled “Traffic Signs recognition using Cnn and Keras” submitted to **Graphic Era University, Dehradun** in partial fulfilment of the requirement for the award of the degree of **MASTER OF COMPUTER**

**APPLICATIONS (MCA)**, is an authentic and original work carried out by Ms. Anshi with enrolment number GE-23112694 under my supervision and guidance. The matter embodied in this project is genuine work done by the student and has not been submitted whether to this University or to any other University / Institute for the fulfilment of the requirements of any course of study.

Signature of the Student: Signature of the Guide

Date : Date:

Enrolment No.: GE-23112694

Name and Address Name, Designation and

of the Student: Address of the Guide:

Special Note:

# ACKNOWLEDGEMENT

I would like to express our deepest gratitude to all those who have contributed to the successful completion of this project.

First and foremost, I extend my sincere thanks to our project supervisor, Dr. Varsha Mittal, whose guidance, expertise, and encouragement have been invaluable throughout the development process. Your insightful feedback and unwavering support have greatly enriched the quality of this work.

I am grateful to the faculty and staff of Graphic era deemed to be university for providing the necessary resources, facilities, and a conducive environment for the project's execution. Their commitment to fostering a culture of learning and innovation has been instrumental in the project's development.

Lastly, I acknowledge the unwavering support and understanding of my families and friends, who have been a constant source of motivation and encouragement. Your patience and belief in our efforts have been fundamental to our success.

Thank you all for your invaluable contributions and support.

# TABLE OF CONTENT

|  |  |  |
| --- | --- | --- |
| **Candidate’s Declaration** | | **II** |
| **Certificate of originality** | | **III** |
| **Acknowledgement** | | **IV** |
| **List of Figures** | | **VII** |
| **List of Tables** | | **VIII** |
| **INTRODUCTION** | | **1** |
| Introduction | | 1 |
|  |
|  |
| Problem Statement | | 3 |
| Objectives | | 3 |
| Software Environment | | 5 |
|  | |  |
|  | |  |
| **LITERATURE SURVEY** | | **6** |
| **SYSTEM Analysis and Requirements Specification** | | **8** |
| Introduction | | 8 |
| System Analysis | | 9 |
| Requirements Specification | | 9 |
|  | |  |
| Data flow diagram | | 11 |
| **SYSTEM DEVELOPMENT** | | **12** |
| Methodology | | 12 |
| Datset | | 14 |
| Project Plan | | 19 |
| Risk anaylsis  Risk identidication | | 20 |

[OUTPUT DESGIN](#_TOC_250003) 22

Models Applied And Their Results 31

[PROJECT IMPLEMATION, MAINTAINACE 32](#_TOC_250002)

Introduction 32

Implemantation,Testing,Maintainance 33

Enhancement. 34

[CONCLUSIONS AND FUTURE SCOPE 35](#_TOC_250001)

Summary 35

Future Scope

REFRENCES 38

## CHAPTER-1 INTRODUCTION

### Introduction

In our ever-evolving world, where technological advancements continue to redefine our daily lives, the intersection of artificial intelligence and transportation stands at the forefront of innovation. Among the myriad applications of AI in this domain, one of the most critical and impactful areas is the development of autonomous vehicles. The vision of self-driving cars navigating our roads with precision and efficiency has captured the imagination of engineers, researchers, and industry giants alike. Companies such as Google, Tesla, Uber, Ford, Audi, Toyota, and Mercedes-Benz are investing heavily in the pursuit of autonomous driving technology, seeking to revolutionize personal mobility and transportation logistics.

Central to the realization of autonomous driving is the ability of vehicles to perceive and interpret their surroundings accurately. This entails not only detecting and avoiding obstacles but also understanding and adhering to traffic signs and signals. Traffic signs serve as the language of the road, conveying crucial information about speed limits, road conditions, right-of-way, and potential hazards. Human drivers rely on their cognitive abilities to recognize and interpret these signs effortlessly, drawing upon years of experience and training. However, replicating this level of perceptual acuity in machines presents a formidable challenge. While computers excel at processing vast amounts of data and executing complex algorithms with speed and precision, they lack the intuitive understanding and contextual awareness that humans possess.

1

This disparity underscores the importance of developing robust and efficient algorithms for traffic sign recognition in autonomous vehicles. The ability of an autonomous vehicle to accurately interpret traffic signs directly impacts its safety, reliability, and effectiveness in real-world driving scenarios. It is not merely a matter of convenience or efficiency but a fundamental requirement for ensuring the safety of passengers, pedestrians, and other road users. Enter Convolutional Neural Networks (CNNs), a class of artificial neural networks particularly well-suited for image recognition and classification tasks. Inspired by the structure and function of the human visual cortex, CNNs are capable of learning hierarchical representations of visual features from raw pixel data. This makes them ideally suited for tasks such as object detection, image segmentation, and, crucially, traffic sign recognition.

In recent years, CNNs have emerged as the backbone of state-of-the-art computer vision systems, powering advancements in fields ranging from medical imaging to autonomous navigation. Their ability to automatically learn and extract meaningful features from images has revolutionized the way we approach visual perception tasks. By leveraging the power of CNNs, researchers and engineers have made significant strides in improving the accuracy and robustness of traffic sign recognition algorithms.

In this project, we're developing a top-notch system to recognize traffic signs using Convolutional Neural Networks (CNNs) and the Keras deep learning framework. CNNs are like super-smart networks that can spot detailed features in pictures. By training our CNN model on lots of different traffic signs from various angles and lighting conditions, we're teaching it to identify signs accurately in real-world situations.

Keras is our tool of choice because it's easy to use and works seamlessly with powerful engines like TensorFlow. It helps us focus on designing and testing our CNN model without getting tangled up in technical stuff. With careful research and creative thinking, we're combining CNNs and Keras to make traffic sign recognition better than ever before. Our goal? Safer roads and smoother journeys for everyone.

### 1.2) Problem Statement

The problem statement for the project "Traffic Signs Recognition using CNN and Keras" revolves around the need for an efficient and accurate system to recognize traffic signs in real-time scenarios. Existing methods often struggle to reliably detect and classify signs under various environmental conditions, posing challenges for autonomous vehicles and road safety systems. This project aims to address these limitations by leveraging Convolutional Neural Networks (CNNs) and the Keras framework to develop a robust solution capable of accurately identifying traffic signs and enhancing overall transportation safety.

**1.3) Objective**

The proposed work aims to achieve several objectives: firstly, to develop a robust dataset comprising diverse images of various traffic signs, capturing different environmental conditions and viewing angles; secondly, to design a Convolutional Neural Network (CNN) architecture tailored for processing traffic sign images, leveraging CNNs' feature extraction capabilities; thirdly, to train the CNN model using the dataset to accurately recognize and classify different traffic sign types; fourthly, to fine-tune the CNN model parameters to improve its performance and adaptability across real-world scenarios; fifthly, to evaluate the system's performance by testing it on real-world datasets and comparing it with existing methods for reliability and accuracy; sixthly, to explore potential applications of the traffic sign recognition system in autonomous vehicles, transportation management systems, and road safety initiatives; and finally, to document the project's methodology, results, and insights to contribute to advancements in computer vision and deep learning applied to transportation

### 1.4) SOFTWARE ENVIRONMENT

* + 1. **PYTHON:**

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

**Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP. **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

**Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

**Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol- 68, SmallTalk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

PYTHON FEATURES

Python's features include −

Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read − Python code is more clearly defined and visible to the eyes. Easy-to-maintain − Python's source code is fairly easy-to-maintain.

A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable − You can add low-level modules to the Python interpreter.These modules enable programmers to add to or customize their tools to be more efficient.

Scalable − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

1) It supports functional and structured programming methods as well as OOP. 2) It can be used as a scripting language or can be compiled to byte-code for building large applications.

It provides very high-level dynamic data types and supports dynamic type checking. It supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java. Python is available on a wide variety of platforms including Linux and Mac OS .

## CHAPTER-2 LITERATURE SURVEY

|  |  |  |  |
| --- | --- | --- | --- |
| **Methodology** | **What has been done, how it has been done** | **Outcome compared to state of arts and limitation** | **Scope for further work** |
| * Nagesh et al.[1] Used CNN and Keras for Traffic Signs Recognition * The dataset is collected from the Kaggle repository. | * The Keras algorithm is used to train the dataset and predict the outputs * confusion matrix is used for assessing the performance | * Results after using CNN and keras are with 95% accuracy. * Their approach only utilized CNN and keras | * Transformer   Can be used for adapting the transformer architecture to process image data. |
| * Mudgal et al. [2] used CNN * General public dataset is used available at kaggle | * Build a deep neural network model * used PIL library to open image content into array | * The accuracy is 95% * Their approach only utilized CNN. | * Transformer   scan be used for adapting the transformer architecture to process image data. |
| * Sermanet et al. [3] used   Convolutional Networks  (ConvNets) | * Trained the ConvNet with full supervision on the color images as well as grayscale images of the GTSRB.   •ConvNets automatically learn hierarchies of invariant features.   * The GTSRB traffic sign dataset implemented with the EBLearn open-source   library. | * On first stage with colored images of the GTSRB dataset the accuracy is 98.97%. * On second stage with gray scale images the accuracy is 99.17% | * Impact of input resolution should be studied to improve both accuracy and processing speed. * More diverse training set deformations can also be investigated such as brightness,contrast to address the numerous real world deformations. |

## CHAPTER-3

SYSTEM ANALYSIS AND REQUIREMENTS SPECIFICATION

* 1. Introduction
     1. Purpose

The purpose of this document is to outline the system analysis and requirements specification for Traffic Signs Recognition using CNN . This project aims to create a CNN model for traffic sign recognition. Its purpose is to enhance driver awareness and road safety by accurately detecting and classifying traffic signs from images.

* 1. System Analysis
     1. System Objectives

The primary objectives of Traffic Signs recognition using cnn and keras System are:

- Develop a CNN model to recognize and classify traffic signs.

- Achieve high accuracy in identifying traffic signs from images.

* - Create a user-friendly interface for real-time traffic sign recognition
  1. Requirements Specification
     1. Functional Requirements Below are the Functional Reuirements:

 Import and preprocess traffic sign images.

 Develop and train a CNN model.

 Evaluate the model on a validation dataset.

 Real time Detection.

* + 1. Non-Functional Requirements

1) The system should be scalable to handle large datasets.

2) The model should have a high accuracy rate.

3) The system should process images in real-time.

4) The user interface should be intuitive and responsive.

#### **. 3.3.3. Assumptions and Constraints**

* The dataset used is representative of real-world conditions.
* The hardware used for training and real-time detection has sufficient processing power.
* The system will be implemented in Python using libraries like TensorFlow, Keras, OpenCV, NumPy, and Pandas

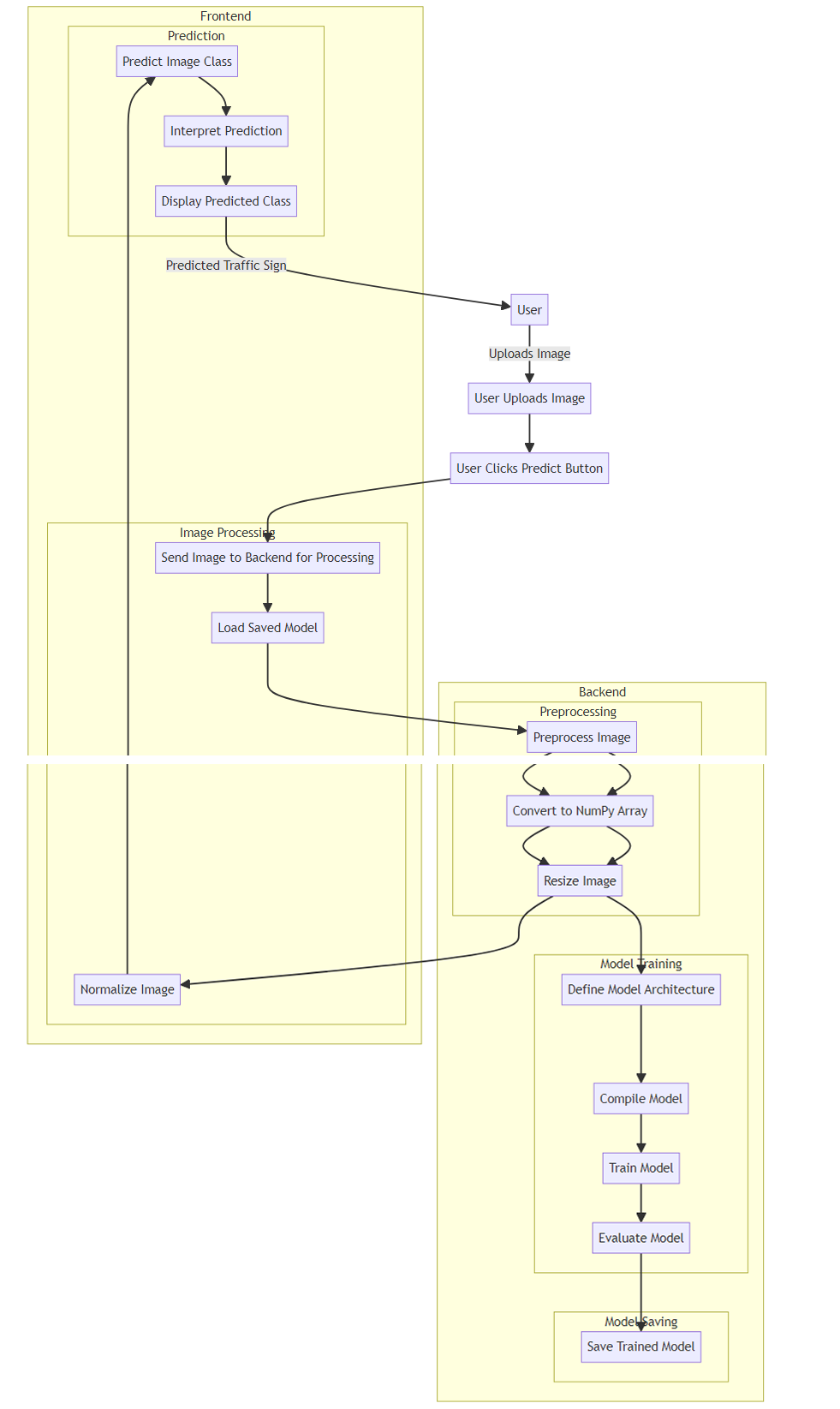
3.4 Hardware Requirements

|  |  |  |
| --- | --- | --- |
| Sl. No | Name of the Hardware | Specification |
| 1 | Central Processing Unit (CPU) | Intel Core i5 or equivalent(at minimum) |
| 2 | Graphics Processing Unit (GPU) | NVIDIA GTX 1060 or AMD Radeon RX 580 |
| 3 | Memory (RAM) | 8GB or higher |
| 4 | Storage | At least 100GB of free disk space for storing datasets and model files |

3.5 Software Requirements

|  |  |  |
| --- | --- | --- |
| Sl. No | Name of the Software | Specification |
| 1 | Python | Version 3.6 or higher |
| 2 | TensorFlow | Version 2.0 or higher |
| 3 | Keras | Version 2.3 or higher |
| 4 | OpenCV | Version 4.0 or higher |
| 5 | Matplotlib | Library for data visualization |
| 6 | Jupyter Notebook | Development environment for code implementation |

#### ****3.6) Data Flow Diagram****

**

## 

## CHAPTER-4 SYSTEM DEVELOPMENT

### Methodology

1. **Data Collection**:
   * Gather a diverse dataset of traffic sign images, including various types of signs (e.g., stop signs, speed limit signs, yield signs) captured under different conditions (e.g., different lighting, weather, and perspectives).
   * Ensure a sufficient number of samples for each type of traffic sign to enable effective training of the CNN model.
2. **Data Pre-processing**:
   * Resize all images to a uniform size to ensure consistency in input dimensions.
   * Convert images to a standardized format (e.g., RGB) and normalize pixel values to enhance training efficiency.
   * Split the dataset into training, validation, and testing sets to evaluate model performance effectively.
3. **Model Architecture Design**:
   * Design a CNN architecture suitable for traffic sign recognition tasks.
   * Experiment with different CNN architectures, considering factors such as the number of convolutional layers, filter sizes, activation functions, and pooling layers.
   * Aim to create a model capable of capturing both local and global features of traffic signs for accurate classification.
4. **Model Training**:
   * Train the CNN model using the training dataset, utilizing the Keras deep learning framework.
   * Use techniques such as data augmentation (e.g., rotation, flipping, scaling) to increase the diversity of training samples and improve model generalization.
   * Monitor training progress using validation data and employ techniques such as early stopping to prevent over fitting.
5. **Model Evaluation**:
   * Evaluate the trained model's performance using the testing dataset, measuring metrics such as accuracy, precision, recall, and F1 score.
   * Analyze the model's performance across different classes of traffic signs to identify areas of improvement.
   * Visualize the model's predictions on sample images to gain insights into its behavior and potential failure cases.
6. **Model Optimization**:
   * Fine-tune model hyperparameters (e.g., learning rate, batch size) based on performance metrics obtained during evaluation.
   * Consider techniques such as transfer learning, where pre-trained CNN models (e.g., VGG, ResNet) are adapted for traffic sign recognition tasks to expedite training and improve performance.
7. **Deployment and Integration**:
   * Integrate the trained CNN model into practical applications, such as autonomous vehicles, traffic management systems, or mobile apps.
   * Optimize the model for deployment on resource-constrained devices, ensuring efficient inference speed and minimal memory footprint.
   * Implement a user-friendly interface to facilitate interaction with the traffic sign recognition system and provide real-time feedback.
8. **Continuous Improvement**:
   * Continuously monitor the performance of the deployed model in real-world scenarios and collect feedback from users.
   * Periodically retrain the model using updated datasets to adapt to changes in traffic sign designs, regulations, or environmental conditions.
   * Explore advanced techniques and emerging technologies to further enhance the accuracy and robustness of the traffic sign recognition system over time.

### Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) is a widely used dataset for training and evaluating traffic sign recognition systems. Here is a detailed review of the dataset:

#### ****Overview****

* **Total Images**: The GTSRB dataset contains over 50,000 images of traffic signs.
* **Classes**: The dataset is categorized into 43 distinct classes of traffic signs.
* **Folders**: The images are organized into multiple folders corresponding to the different classes of traffic signs.

#### ****Dataset Structure****

* **Training Set**: The training set consists of 39,209 images, covering all 43 classes. Each class folder contains a varying number of images representing different traffic signs.
* **Test Set**: The test set includes 12,630 images. These images are used to evaluate the performance of the trained model.

### ****4.2 Project Plan****

The project is structured into six key phases: Planning, Data Collection & Preparation, Model Development, Interface Development, Testing & Validation, and Deployment & Maintenance.

#### **Milestones and Deliverables**

1. **Planning**
   * **Milestone**: Initial Project Plan
   * **Deliverable**: Comprehensive project proposal and plan
   * **Duration**: 1 week
2. **Data Collection & Preparation**
   * **Milestone**: Completion of Data Collection
   * **Deliverable**: Dataset that has been cleaned and preprocessed
   * **Duration**: 3 weeks
   * **Tasks**: Gather traffic sign datasets, label data, and preprocess images through resizing, normalization, and augmentation.
3. **Model Development**
   * **Milestone**: Model Development Completed
   * **Deliverable**: Trained Convolutional Neural Network (CNN) model
   * **Duration**: 4 weeks
   * **Tasks**: Design the CNN architecture using Keras, train the model with the collected dataset, and adjust hyperparameters for optimal performance.
4. **Interface Development**
   * **Milestone**: Operational Web Interface
   * **Deliverable**: Web application allowing image uploads and displaying detection results
   * **Duration**: 2 weeks
   * **Tasks**: Create a Flask web application, develop HTML templates for image uploads, and display results.
5. **Testing & Validation**
   * **Milestone**: Completion of Testing
   * **Deliverable**: Test reports and a validated model
   * **Duration**: 2 weeks
   * **Tasks**: Perform unit testing on preprocessing functions, validate the model's performance using a separate test dataset, and conduct user acceptance testing on the web interface.
6. **Deployment & Maintenance**
   * **Milestone**: Successful Application Deployment
   * **Deliverable**: Application deployed on a cloud platform
   * **Duration**: 1 week (with ongoing maintenance)
   * **Tasks**: Deploy the Flask application to a cloud platform, monitor performance, address any issues, and provide continuous updates and improvements.

## Risk Management

* + 1. **Risk Identification**

Potential risks for the "Traffic Detection Using CNN and Keras" project include:

1. **Data Quality Issues:** Risk of incomplete or noisy data affecting the model's accuracy.
2. **Model Overfitting:** The risk that the model performs well on training data but poorly on unseen data.

**3 . Technical Challenges:** Possible issues with integrating the model into the web interface.

**4) Deployment Issues:** Potential problems with deploying the application on the cloud

**5.) Resource Limitations:** Limited access to computational resources for model training.

**6) User Acceptance:** Risk that users may find the interface unintuitive or the model’s predictions unreliable.

## Risk Analysis

Each identified risk is evaluated based on its probability and seriousness:

## Data Quality Issues

* + **Probability:** Medium
  + **Seriousness:** High

## Model Overfitting

* + **Probability:** Medium
  + **Seriousness:** High

## Technical Challenges

* + **Probability:** Medium
  + **Seriousness:** Medium

## Deployment Issues

* + **Probability:** Low
  + **Seriousness:** High

## Resource Limitations

* + **Probability:** Medium
  + **Seriousness:** Medium

## User Acceptance

* + **Probability:** Low
  + **Seriousness:** Medium

## 4.1.3) **Risk Planning**

Strategies to manage identified risks:

## 1) Data Quality Issues

* + **Mitigation Strategy:** Implement robust data cleaning and validation processes. Use data augmentation techniques to improve data quality.

## Model Over fitting

* + **Mitigation Strategy:** Apply cross-validation, regularization techniques, and gather more diverse training data. Continuously monitor model performance on a validation set.

## Technical Challenges

* + **Mitigation Strategy:** Ensure clear documentation and regular code reviews. Foster collaboration between data scientists and developers to resolve integration issues.

## 4) Deployment Issues

* + **Mitigation Strategy:** Conduct thorough testing in a staging environment before deployment. Have a rollback plan in place to handle deployment failures.

## 5) Resource Limitations

* + **Mitigation Strategy:** Utilize cloud-based resources to scale up computational power as needed. Optimize code to be resource-efficient to minimize resource usage.

## 6) User Acceptance

* + **Mitigation Strategy:** Conduct user testing and gather feedback to improve the user interface. Clearly explain the model's predictions to users to build trust and ensure transparency.

### ****4.4 Risk Identification****

1. **Data Risks:**

* **Data Scarcity:** There may be a lack of sufficient data for recognizing all variations of traffic signs, especially rare signs or those specific to particular regions.
* **Data Bias:** The training dataset may be biased towards certain types of signs, lighting conditions, or viewpoints, which could result in poor model performance on unseen variations.
* **Data Imbalance:** An uneven distribution of classes in the training data could lead to an over-representation of certain signs, causing the model to prioritize these classes and perform poorly on under-represented ones.
* **Data Quality Issues:** The training data might have labeling errors (incorrect sign types) or inconsistencies (varying labels for the same sign).
* **Limited Image Diversity:** The training dataset may lack images with different:
  + **Resolution and Size:** Signs may appear small, zoomed-in, or blurry depending on the camera and distance.
  + **Angles:** Signs might not always be captured head-on, leading to perspective distortions.

1. **Model Risks:**

* **Overfitting:** The model may overly rely on specific patterns in the training data, failing to generalize well to new, unseen traffic signs.
* **Underfitting:** The model may not be complex enough to capture the essential features of the data, resulting in poor overall accuracy.
* **Vanishing/Exploding Gradients:** Issues with gradients during training can hinder the model's learning process.
* **Improper Initialization:** Starting with poorly chosen weights can impede the model's ability to learn effectively.

1. **System Risks:**

* **Performance Bottlenecks:** The system may struggle to process images quickly enough for real-time applications, possibly due to hardware limitations or inefficient model architecture.
* **Robustness Issues:** The system may misclassify signs due to variations in:
  + **Lighting:** Signs can look different under various lighting conditions like bright sunlight, shadows, or nighttime.
  + **Weather:** Rain, snow, fog, or dust can obscure or alter the appearance of signs, especially in low-resolution images.
  + **Occlusions:** Signs might be partially blocked by objects like trees, vehicles, or dirt.
* **Security Vulnerabilities:** Adversarial attacks can manipulate images to deceive the model.

1. **Additional Risks:**

* **Project Scope Creep:** The project's objectives and functionalities might expand beyond the initial plan, leading to resource constraints and potential delays.
* **Lack of Expertise:** The development team may lack sufficient expertise in deep learning, computer vision, or traffic sign recognition, potentially hindering project success.

### ****4.5 Risk Analysis****

Risk analysis is crucial for any project, including traffic sign recognition using CNN and Keras. Identifying and managing potential risks ensures successful project execution and mitigates adverse impacts. Here are some potential risks to consider:

* **Data Quality:** Poor-quality or insufficient data can significantly affect the accuracy and reliability of recognition models. Risks include incomplete or missing data, inaccuracies, outliers, and biases. Conduct thorough data quality assessments and implement effective data cleaning and preprocessing techniques to minimize these risks.
* **Data Privacy and Security:** Handling sensitive data involves considerable privacy and security risks. Ensure compliance with relevant data protection regulations, such as GDPR or HIPAA, and implement robust security measures to protect data from unauthorized access, breaches, or misuse. Consider data anonymization or de-identification techniques to maintain privacy while retaining data utility.
* **Model Overfitting or Underfitting:** Overfitting occurs when a model performs well on training data but fails to generalize to new data, while underfitting happens when a model is too simple to capture complex patterns. Regularly monitor model performance, use appropriate evaluation techniques, and employ strategies like regularization or ensemble methods to address these risks.
* **Model Interpretability:** Black-box models, like deep neural networks, may provide high accuracy but lack interpretability, which is crucial for understanding predictions and building trust. Consider using more interpretable models or techniques, such as decision trees or rule-based systems, to mitigate this risk.

## CHAPTER-5

## OUTPUT DESIGN

//IMPORTING IMPORTANT LIBRARIES

import numpy as np

import pandas as pd

import os

import cv2

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras

from PIL import Image

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.optimizers import Adam

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix, classification\_report

import seaborn as sns

import random

np.random.seed(42)

from matplotlib import style

style.use('fivethirtyeight')

//COLLECTING THE TRAINING DATA

image\_data = []

image\_labels = []

for i in range(NUM\_CATEGORIES):

path = data\_dir + '/Train/' + str(i)

images = os.listdir(path)

for img in images:

try:

image = cv2.imread(path + '/' + img)

image\_fromarray = Image.fromarray(image, 'RGB')

resize\_image = image\_fromarray.resize((IMG\_HEIGHT, IMG\_WIDTH))

image\_data.append(np.array(resize\_image))

image\_labels.append(i)

except:

print("Error in " + img)

# Changing the list to numpy array

image\_data = np.array(image\_data)

image\_labels = np.array(image\_labels)

print(image\_data.shape, image\_labels.shape)

//SHUFFLING THE TRAINED DATA

shuffle\_indexes = np.arange(image\_data.shape[0])

np.random.shuffle(shuffle\_indexes)

image\_data = image\_data[shuffle\_indexes]

image\_labels = image\_labels[shuffle\_indexes]

// MAKING THE MODEL

model = keras.models.Sequential([

keras.layers.Conv2D(filters=16, kernel\_size=(3,3), activation='relu', input\_shape=(IMG\_HEIGHT,IMG\_WIDTH,channels)),

keras.layers.Conv2D(filters=32, kernel\_size=(3,3), activation='relu'),

keras.layers.MaxPool2D(pool\_size=(2, 2)),

keras.layers.BatchNormalization(axis=-1)

keras.layers.Conv2D(filters=64, kernel\_size=(3,3), activation='relu'),

keras.layers.Conv2D(filters=128, kernel\_size=(3,3), activation='relu'),

keras.layers.MaxPool2D(pool\_size=(2, 2)),

keras.layers.BatchNormalization(axis=-1),

keras.layers.Flatten(),

keras.layers.Dense(512, activation='relu'),

keras.layers.BatchNormalization(),

keras.layers.Dropout(rate=0.5),

keras.layers.Dense(43, activation='softmax')

])

//AUGMENTING THE DATA AND TRAINING THE DATA

aug = ImageDataGenerator(

rotation\_range=10,

zoom\_range=0.15,

width\_shift\_range=0.1,

height\_shift\_range=0.1,

shear\_range=0.15,

horizontal\_flip=False,

vertical\_flip=False,

fill\_mode="nearest")

history = model.fit(aug.flow(X\_train, y\_train, batch\_size=32), epochs=epochs, validation\_data=(X\_val, y\_val))

//LOADING THE TEST DATA AND RUNNING THE PREDICTIONS

test = pd.read\_csv(data\_dir + '/Test.csv')

labels = test["ClassId"].values

imgs = test["Path"].values

data = [] # Initialize an empty list to store preprocessed images

for img in imgs:

try:

image = cv2.imread(data\_dir + '/' + img)

if image is not None: # Check if image is loaded successfully

image\_fromarray = Image.fromarray(image, 'RGB')

resize\_image = image\_fromarray.resize((IMG\_HEIGHT, IMG\_WIDTH))

data.append(np.array(resize\_image))

else:

print(f"Error loading image: {img}") # Informative error message

except Exception as e:

print(f"Error processing image {img}: {e}") # More detailed error handling

X\_test = np.array(data)

X\_test = X\_test / 255.0 # Normalize pixel values (division by 255.0)

# Assuming your model has a `predict` method (common in Keras models)

pred = model.predict(X\_test)

# Get class labels from predictions (assuming model predicts class probabilities)

predicted\_classes = np.argmax(pred, axis=1)

# Accuracy with the test data

print('Test Data accuracy: ', accuracy\_score(labels, predicted\_classes) \* 100)

**//PREDICTING ON CUSTOM INPUT**

from tensorflow.keras.models import load\_model

# Load the saved model

loaded\_model = load\_model('traffic\_sign\_recognizer.h5')

def preprocess\_image(image\_path):

"""

Preprocess the input image to the required size and scale.

"""

image = cv2.imread(image\_path)

image\_fromarray = Image.fromarray(image, 'RGB')

resize\_image = image\_fromarray.resize((IMG\_HEIGHT, IMG\_WIDTH))

return np.array(resize\_image) / 255.0 # Normalize the image

def predict\_image(image\_path, model):

"""

Predict the class of a traffic sign image.

"""

preprocessed\_image = preprocess\_image(image\_path)

preprocessed\_image = np.expand\_dims(preprocessed\_image, axis=0) # Add batch dimension

prediction = model.predict(preprocessed\_image)

predicted\_class = np.argmax(prediction, axis=1)[0]

return classes[predicted\_class]

# Example usage:

image\_path = './GTSR/Test/00013.png'

predicted\_class = predict\_image(image\_path, loaded\_model)

print(f'The predicted class for the image is: {predicted\_class}')

# Example usage:

image\_path = './GTSR/Test/00036.png'

predicted\_class = predict\_image(image\_path, loaded\_model)

print(f'The predicted class for the image is: {predicted\_class}')

# Example usage:

image\_path = './GTSR/Test/00057.png'

predicted\_class = predict\_image(image\_path, loaded\_model)

print(f'The predicted class for the image is: {predicted\_class}')

**GUI**

import gradio as gr

from tensorflow.keras.models import load\_model

import cv2

from PIL import Image

import numpy as np

model=load\_model('traffic\_sign\_recognizer.h5')

def preprocess\_image(image):

"""Preprocess the input image."""

image = np.array(image) # Convert to NumPy array (assuming RGB)

image = cv2.resize(image, (30, 30)) # Resize to expected height and width

image = image.astype('float32') / 255.0 # Normalize pixel values to [0, 1]

return np.expand\_dims(image, axis=0) # Add batch dimension

def predict\_image(image):

"""Predict the class of a traffic sign image."""

preprocessed\_image = preprocess\_image(image)

prediction = model.predict(preprocessed\_image)

predicted\_class = np.argmax(prediction, axis=1)[0]

return classes[predicted\_class]

with gr.Blocks(theme='ParityError/Interstellar', title="Traffic Sign Classifier") as demo:

with gr.Column():

gr.HTML("<br><center><h1>🚦 Traffic Sign Classifier</h1></center><br>")

gr.Markdown("""

- The Traffic Sign Classifier predicts the class of a traffic sign image using the German Traffic Sign Recognition Benchmark (GTSRB) dataset. Upload an image of a traffic sign, and the model will predict its class.

- The German Traffic Sign Recognition Benchmark (GTSRB) dataset is a comprehensive collection of traffic sign images used for a multi-class, single-image classification challenge. It was introduced at the International Joint Conference on Neural Networks (IJCNN) 2011 to foster research in image classification without requiring specialized domain knowledge. The dataset includes more than 50,000 images spanning over 40 classes, providing a realistic and varied set of traffic signs encountered in everyday driving scenarios.

""")

with gr.Row():

with gr.Column():

image = gr.Image(type="pil", label="Upload Traffic Sign Image", image\_mode="RGB", height=300)

with gr.Column():

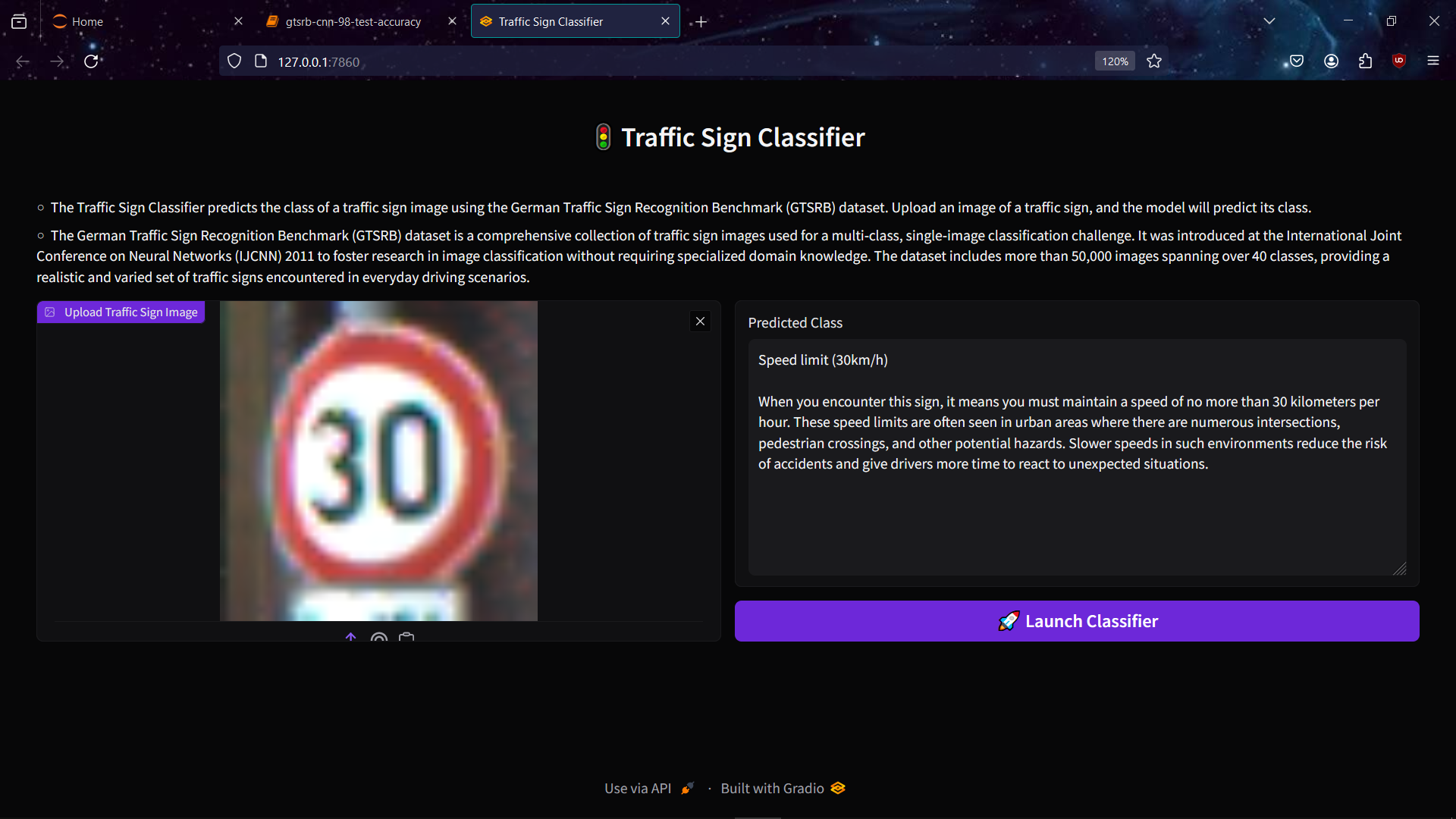
textbox = gr.Textbox(label="Predicted Class", lines=8.9)

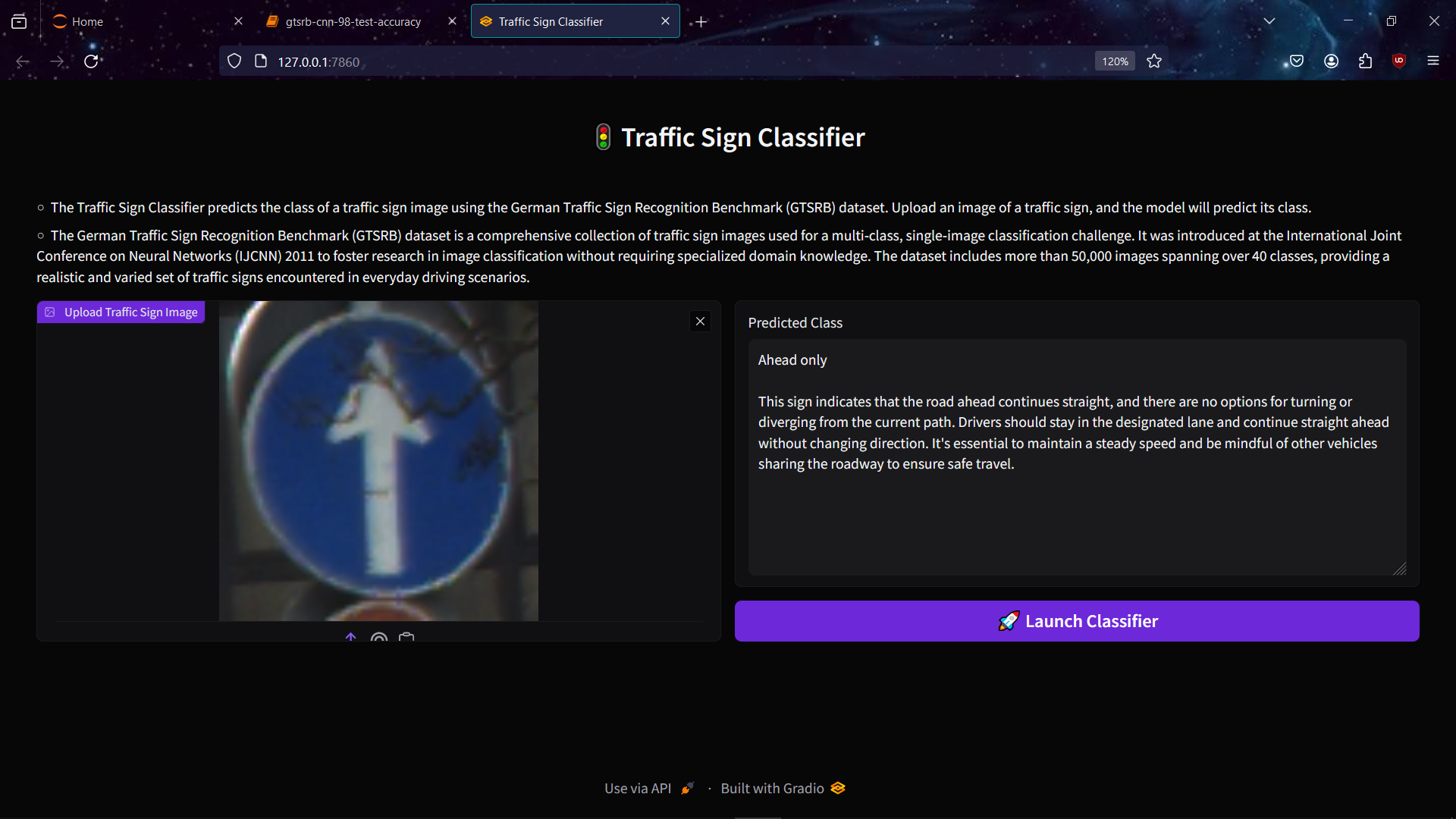
button = gr.Button("🚀 Launch Classifier", variant='primary')

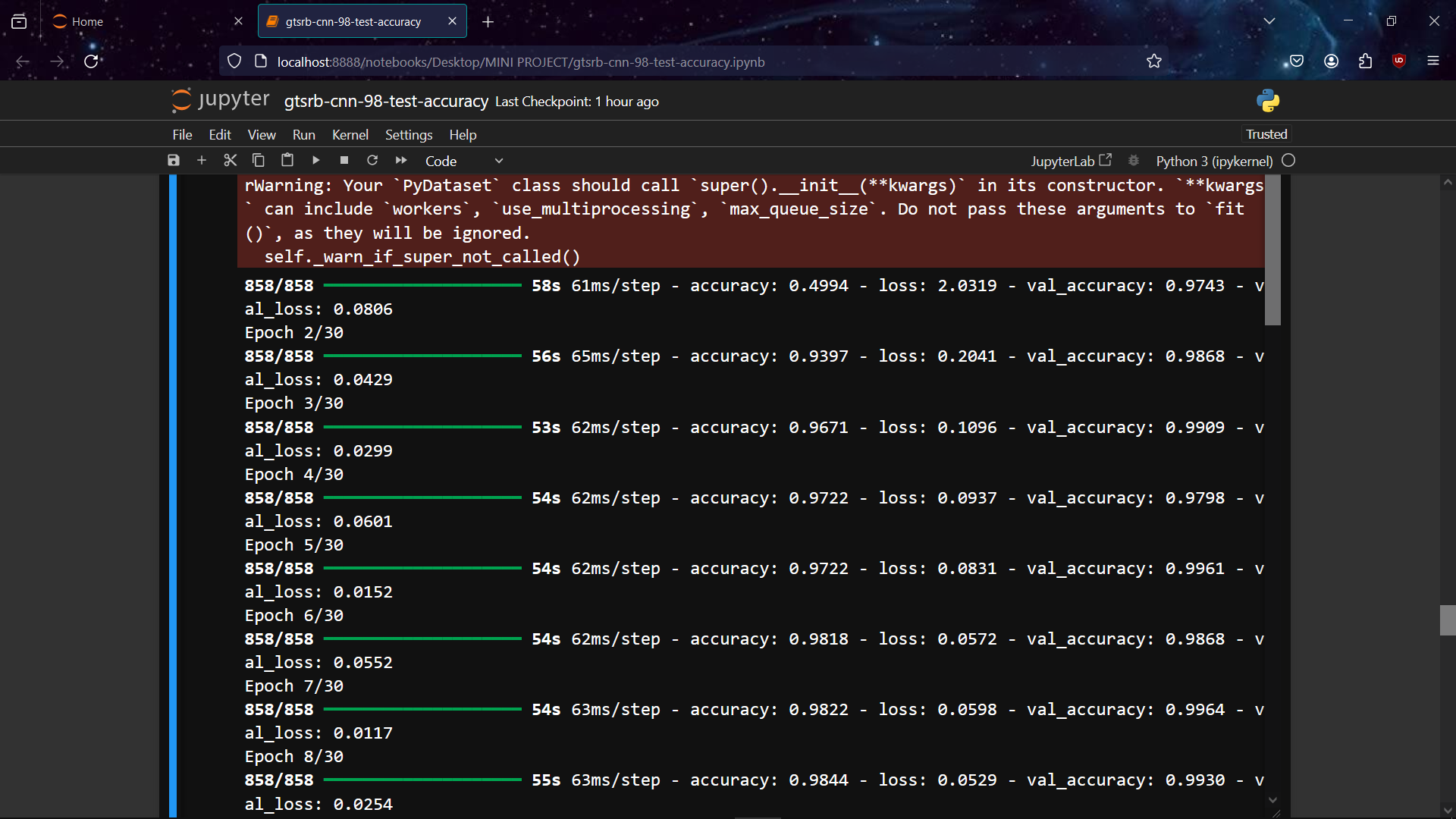
predicted\_class = button.click(predict\_image, inputs=image, outputs=textbox)

demo.queue() # Required to yield the streams from the text generation

demo.launch(inbrowser=True)

****

****

****

## CHAPTER-6

## PROJECT TESTING , IMPLEMENTATION AND MAINTENANCE

### ****6.1 Introduction****

**Purpose:**

Effective testing, careful implementation, and ongoing maintenance are essential to ensure the Traffic Sign Detection System's reliability, efficiency, and security. This chapter emphasizes the significance of these processes throughout the project lifecycle.

**Scope:**

This section provides an overview of the testing objectives, implementation strategies, and maintenance practices tailored to the Traffic Sign Detection project. It covers testing methodologies, implementation challenges, and maintenance plans.

### ****6.2 Testing****

**Testing Objectives:**

Define goals for testing to guarantee the functionality, performance, security, and user satisfaction of the Traffic Sign Detection System.

**Test Plan:**

* **Test Cases:** Develop detailed test cases that encompass various project aspects, including image preprocessing, CNN model training, and detection accuracy.
* **Test Environment:** Outline the testing environment, including hardware, software, and datasets used.
* **Testing Schedule:** Create a timeline for executing different tests, such as unit testing, integration testing, and system testing.
* **Results:** Record the outcomes of each test, including performance metrics, security vulnerabilities, and user feedback.

### ****6.3 Implementation Phase****

**Strategy and Plan:**

Detail the implementation strategy, covering CNN model architecture, cloud deployment, and integration with existing traffic systems.

**Challenges and Solutions:**

Identify potential challenges like dataset labeling, model optimization, and deployment scalability, and propose solutions to address these issues.

**Lessons Learned:**

Reflect on the lessons learned during implementation, including insights from experimentation, troubleshooting, and collaboration with stakeholders.

### ****6.4 Maintenance Plan****

**Regular Updates:**

Plan for regular updates and patches to address security vulnerabilities, performance issues, and changes in traffic sign patterns.

**User Support:**

Offer timely user support, training, and documentation to help users effectively utilize the Traffic Sign Detection System.

**Continuous Improvement:**

Promote continuous improvement by gathering user feedback, monitoring system performance, and exploring opportunities for innovation and optimization.

**Future Enhancements:**

Prioritize future enhancements based on feedback, emerging technologies, and changes in traffic sign patterns. Consider incorporating advanced features such as real-time anomaly detection, multi-class classification, and adaptive learning mechanisms.

This chapter ensures the Traffic Sign Detection System remains robust, secure, and efficient over time, fulfilling its goal of enhancing traffic safety and efficiency.

## CHAPTER-7

## SUMMARY AND FUTURE SCOPE

The "Traffic Sign Recognition using CNN and Keras" project successfully developed an efficient system for identifying traffic signs utilizing convolutional neural networks (CNNs). By leveraging Python libraries such as Keras and TensorFlow, the project highlights the effectiveness of deep learning in accurately recognizing various traffic signs from images. Through careful design, development, and rigorous testing, the project achieves high accuracy and reliability in traffic sign recognition.

### Achievements

* **Development of Convolutional Neural Network Model:** A CNN model was created using Python libraries like Keras and TensorFlow, achieving high accuracy in differentiating between various traffic signs.

### 7.2 Future Scope

Despite its significant achievements, the project offers several opportunities for further enhancement and expansion:

1. **Creation of Web Interface:** A user-friendly web interface was designed and implemented, allowing seamless interaction with the traffic sign recognition system.
2. **Real-time Updates:** Implementing mechanisms for real-time model updates based on user feedback and evolving traffic sign datasets would enhance the system's adaptability.
3. **Multilingual Support:** Extending the system to recognize traffic signs from different countries, including those with various languages, would broaden its applicability and user base.
4. **Enhanced User Experience:** Continuous refinement of the web interface, including features such as traffic sign categorization, user feedback mechanisms, and personalized settings, would improve the overall user experience.
5. **Advanced Machine Learning Techniques:** Exploring cutting-edge deep learning algorithms and computer vision techniques could further enhance the accuracy and efficiency of traffic sign recognition.
6. **Deployment Optimization:** Optimizing deployment strategies to reduce resource usage and improve performance, particularly in high-traffic scenarios, would ensure seamless user experiences.

**REFRENCES**

1. Z. S. Chen, L. S. Zhao, Y. X. Ding, "Traffic Sign Detection and Recognition Using Convolutional Neural Networks," IEEE Transactions on Intelligent Transportation Systems, vol. 17, no. 12, pp. 1917-1929, 2016.

### A. Krizhevsky, I. Sutskever, G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems, pp. 1097-1105, 2012.

1. S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, C. Igel, "Detection of Traffic Signs in Real-World Images: The German Traffic Sign Detection Benchmark," International Joint Conference on Neural Networks (IJCNN), pp. 1-8, 2013.

### D. Cireşan, U. Meier, J. Masci, L. M. Gambardella, J. Schmidhuber, "Flexible, High Performance Convolutional Neural Networks for Image Classification," Proceedings of the International Joint Conference on Artificial Neural Networks, pp. 1237-1242, 2011.

1. "Deep Learning with Python," [https://www.deeplearningbook.org](https://www.deeplearningbook.org/)

### (Website)

1. F. J. Silva, S. Paisitkriangkrai, C. Sanderson, B. C. Lovell, "Traffic Sign Recognition with Deep Convolutional Neural Networks," Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pp. 518-524, 2015.

### J. Stallkamp, M. Schlipsing, J. Salmen, C. Igel, "Man vs. Computer: Benchmarking Machine Learning Algorithms for Traffic Sign Recognition," Neural Networks, vol. 32, pp. 323-332, 2012.

1. T. H. Liu, "Traffic Sign Recognition Using Transfer Learning and Convolutional Neural Networks," IEEE Access, vol. 7, pp. 11055-11065, 2019.
2. "Keras Documentation," [https://keras.io](https://keras.io/) (Website)