** **

**Roadscape Analysis:**

**Deep Learning-Based Classification of Clean and Littered Roads**

Project Hand-Out

-Prince Singh

**Roadscape Analysis: Deep Learning-Based Classification of Clean and Littered Roads**

The cleanliness of roads plays a crucial role in maintaining the overall appearance, safety, and functionality of urban environments. Littered roads not only have negative visual impacts but also pose potential hazards to pedestrians, motorists, and the environment. Manual monitoring of road cleanliness can be time-consuming, costly, and prone to human error. To address these challenges, the application of deep learning techniques for the classification of clean and littered roads has emerged as a promising solution.

Deep learning, a subfield of artificial intelligence, has revolutionized image recognition tasks by enabling computers to learn and extract features directly from raw data. Convolutional Neural Networks (CNNs) are particularly effective in image classification tasks due to their ability to automatically learn and detect meaningful patterns and features from images. In this project, we leverage the vast amount of road imagery data and employ CNNs to train a model capable of distinguishing between clean and littered roads. The model will learn to identify specific visual cues and patterns associated with cleanliness or littering, such as the presence of garbage, debris, or other visual anomalies.

The advantages of deep learning-based classification are numerous. Firstly, it eliminates the need for manual inspection and assessment of road cleanliness, saving time and resources for municipalities and urban management authorities. Secondly, the system can provide real-time monitoring and alerts, allowing for timely intervention and maintenance activities. Lastly, the data collected through the deep learning system can be utilized for further analysis, helping to identify trends, patterns, and hotspots of littering behavior, and supporting evidence-based decision-making for urban planning and waste management initiatives.

**Aim:**

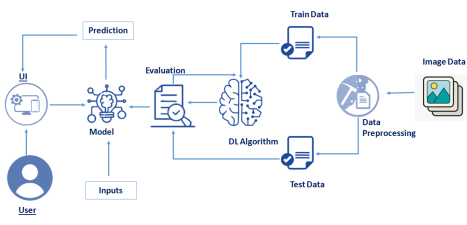
**The aim of this project is to develop a deep learning-based classification system that can accurately distinguish between clean and littered roads, providing an automated and efficient solution for road maintenance and urban cleanliness management.**

**Purpose of doing this Project:**

The purpose of undertaking this project is twofold. Firstly, it aims to address the challenge of manual road cleanliness monitoring by developing an automated system that can classify roads as either clean or littered. This will save time, resources, and reduce the potential for human error. Secondly, the project aims to contribute to urban cleanliness management by providing real-time monitoring and data analysis capabilities, enabling evidence-based decision-making, targeted interventions, and the identification of littering patterns for improved waste management strategies. Ultimately, the project seeks to enhance the overall aesthetics, safety, and functionality of roadscapes in urban environments.

**Technical Architecture:**

The technical architecture of this project revolves around the implementation of deep learning algorithms, specifically Convolutional Neural Networks (CNNs), for roadscape analysis and classification. The architecture consists of several key components. Firstly, a dataset of road imagery is collected, including images of both clean and littered roads. The dataset is then preprocessed to enhance image quality and remove any irrelevant or noisy data. Next, a CNN model is designed and trained using the preprocessed dataset. The model learns to extract relevant features and patterns from the road images to accurately classify them as either clean or littered. The trained model is then evaluated on a separate test dataset to assess its performance and generalization capabilities. The final architecture includes a real-time monitoring system that takes road images as input, applies the trained model for classification, and provides timely alerts and insights for road maintenance and cleanliness management.



**Pre-requisites:**

**To complete this project, you must require the following software, concepts, and packages**

* **Anaconda Navigator:**

- Link: https://www.youtube.com/watch?v=5mDYijMfSzs

* Python packages:

- Open anaconda prompt as administrator

- Type “pip install tensorflow” (make sure you are working on Python 64 bit)

- Type “pip install flask”.

* **Deep Learning Concepts**

- CNN: https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add

- Flask Basics: https://www.youtube.com/watch?v=lj4I\_CvBnt0

**Project Objectives:**

By the end of this project you will:

* Know fundamental concepts and techniques of Convolutional Neural Network.
* Gain a broad understanding of image data.
* Know how to pre-process/clean the data using different data preprocessing techniques.
* Know how to build a web application using the Flask framework.

**Project Flow:**

* User interacts with the UI (User Interface) to upload the image as input
* Uploaded image is analyzed by the model which is integrated
* Once the model analyses the uploaded image, the predicted report is showcased on the UI to accomplish this, we have to complete all the activities and tasks listed below

o Data Collection.

o Collect the dataset or Create the dataset

o Data Preprocessing.

o Import the ImageDataGenerator library

o Configure ImageDataGenerator class

o Apply ImageDataGenerator functionality to Trainset and Testset

o Model Building

o Import the model building Libraries

o Initializing the model

o Adding Input Layer

o Adding Hidden Layer

o Adding Output Layer

o Configure the Learning Process

o Training the model

o Save the Model

o Test the Model

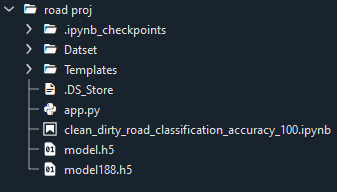
o Application Building

o Create an HTML file

o Build Python Code

**Project Structure:**

Create project folder which contains files as shown below:



* The data obtained folders contains around 150 images of clean and dirty roads, which is used for training, testing, and validation dataset.
* We are building a Flask application that will require the html files to be stored in the templates folder.
* app.py file is used for routing purposes using scripting.
* model188.h5 is the saved model. This will further be used in the Flask integration.

**Milestone 1: Define Problem/ Problem Understanding**

**Activity 1: Specify the Business Problem**

The business problem addressed by this project is the manual and resource-intensive nature of road cleanliness monitoring. Traditional methods of assessing road conditions rely on human inspection, leading to inefficiencies, delays, and potential inaccuracies. This project aims to provide an automated solution using deep learning-based classification to accurately distinguish between clean and littered roads. By automating this process, municipalities and urban management authorities can save time and resources, enabling them to prioritize and efficiently allocate their efforts towards road maintenance and cleanliness management, resulting in improved aesthetics, safety, and overall urban environment quality.

**Activity 2: Business Requirements**

1. **Deep Learning-Based Classification System:** Develop a robust deep learning-based system that can accurately classify roads as either clean or littered based on visual cues and patterns.
2. **Real-Time Monitoring and Alert System:** Implement a real-time monitoring system that continuously analyzes road imagery to detect and promptly alert authorities about areas requiring maintenance or cleaning.
3. **User-Friendly Interface:** Design a user-friendly interface that enables municipalities and urban management authorities to easily integrate the system into their existing road maintenance processes without requiring extensive technical expertise.
4. **Comprehensive Reporting:** Provide an automated reporting mechanism that generates comprehensive reports on road cleanliness, highlighting trends, patterns, and hotspots for informed decision-making and targeted intervention strategies.
5. **Scalability and Adaptability:** Ensure the system is scalable and adaptable to different geographical areas, road types, and varying levels of littering behavior, allowing for efficient implementation across diverse urban environments.
6. **Trust:** The model should be designed in such a way that it develops trust among the users, especially the advertisers and content creators, who rely on the predicted adviews for their marketing strategies.
7. **User-Friendly Interface:** The deployment using Flask should provide a user-friendly web interface where stakeholders, such as farmers, distributors, and consumers, can easily interact with the system. The interface should allow users to upload mango images, receive classification results, and view the grading category assigned to each mango.
8. **Seamless Deployment:** The deployment using Flask should be smooth and seamless, ensuring that the model is easily accessible and operational. The system should handle multiple requests concurrently and provide consistent and reliable performance.
9. **Scalability:** The solution should be scalable to handle a growing number of mango images and users. As the usage of the system increases, it should be able to accommodate the additional load without compromising on performance or accuracy.
10. **Flexibility for Future Enhancements:** The system should be designed with flexibility in mind, allowing for future enhancements and improvements. This includes the ability to incorporate additional grading categories or expand the dataset to further refine the mango quality assessment capabilities of the model.

**Activity 3: Literature Survey**

Existing literature on automated road cleanliness analysis using deep learning reveals a growing interest in leveraging convolutional neural networks (CNNs) for clean and littered road classification. Studies have explored various techniques, including data collection, preprocessing, model architectures, and evaluation metrics. The findings highlight the potential of deep learning in achieving accurate and efficient road cleanliness monitoring, supporting evidence-based decision-making and urban cleanliness management.

**Activity 4: Social or Business Impact.**

**Social Impact:**

The development of a deep learning-based classification system for clean and littered roads can have a significant social impact. By automating road cleanliness monitoring, this system can help create cleaner and more aesthetically pleasing urban environments. Clean roads contribute to a sense of pride and well-being among residents and visitors, fostering a positive community image. Moreover, by identifying and addressing littered areas promptly, the system enhances safety for pedestrians and motorists, reducing the risk of accidents and injuries caused by debris on the roads.

**Business Impact:**

Implementing a deep learning-based classification system for road cleanliness can bring several benefits to businesses and municipalities. Firstly, it streamlines and automates the road maintenance process, reducing manual labor and associated costs. This allows municipalities to allocate resources more efficiently and effectively. Secondly, by maintaining clean roads, businesses can enhance their brand image and attract more customers. A positive visual impression of the roadscape can create a favorable perception of the surrounding businesses, contributing to increased foot traffic, tourism, and economic growth.

**Milestone 2: Data Collection**

Machine Learning & Deep Learning depends heavily on data. It is the most crucial part aspect that makes algorithm training possible. So, this section guides on how to download dataset.

**Activity 1: Collect the dataset**

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

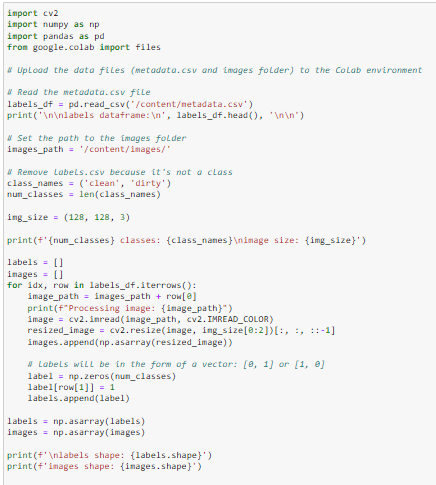
Link: https://www.kaggle.com/datasets/faizalkarim/cleandirty-road-classification

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

**Note:** There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

**Milestone 3: Image Preprocessing**

**Activity 1: Data Loading, Formating**



The provided code snippet includes necessary imports and uses the Google Colab files module to upload data files, including a metadata.csv file and an images folder. The metadata.csv file is then read into a DataFrame using pandas. The code sets the path to the images folder and defines class names ('clean' and 'dirty') and the number of classes. It also defines the desired image size. The code iterates over the rows of the metadata DataFrame, reads and resizes each image using OpenCV, and appends the resized image and corresponding label to the respective lists. Finally, the code converts the lists to numpy arrays and prints the shapes of the labels and images arrays.



**Activity 2: Import the ImageDataGenerator library**

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.

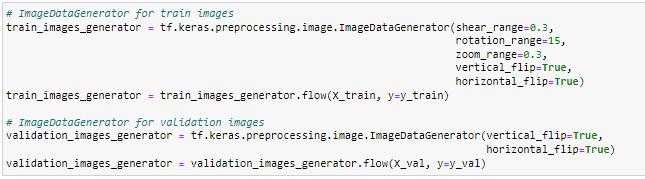
Let us import the ImageDataGenerator class from keras



**Activity 3: Configure ImageDataGenerator class**

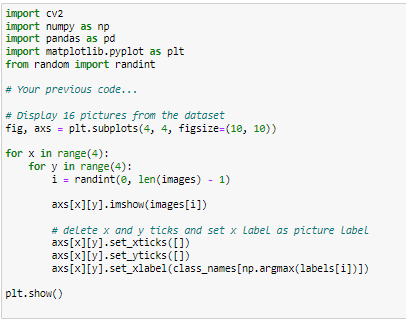
The ImageDataGenerator class in Keras is used for real-time data augmentation during model training. It allows the user to perform various image transformations such as rotation, zooming, shifting, and flipping on the input data, which can help prevent overfitting and improve model performance. The class can be configured to apply different types and levels of data augmentation, and it supports both binary and categorical data. The ImageDataGenerator class is a powerful tool for improving the accuracy and robustness of deep learning models, particularly in computer vision applications.

**Activity 4: Data Preparation and Loading for Deep Learning Image Classification**

****

The provided code demonstrates the usage of the ImageDataGenerator class from the Keras library for data augmentation during training and validation. For training images, the train\_images\_generator is created with specified augmentation parameters such as shear range, rotation range, zoom range, vertical flip, and horizontal flip. This generator is then used to flow and generate augmented images along with their corresponding labels. Similarly, the validation\_images\_generator is created for validation images with vertical flip and horizontal flip as augmentation options. These generators allow for increased variation in the training data, helping to improve the model's robustness and generalization capabilities.

**Activity 5: Sample Image Visualization**

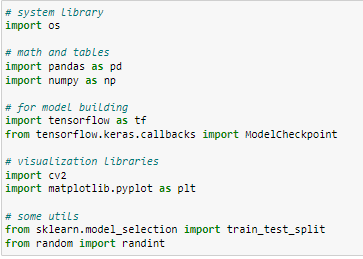
****

The provided code snippet imports the necessary libraries and defines variables for further use. It then creates a grid of subplots using matplotlib to display 16 randomly selected images from the dataset. For each subplot, an image is randomly selected using the randint function, and the image is displayed using imshow. The x and y ticks are removed from the subplots, and the x label is set as the label corresponding to the image class. This visualization provides a visual representation of the dataset, showcasing a variety of images from different classes.



**Milestone 4: Model Building**

**Activity 1 : Importing the Model Building Libraries**

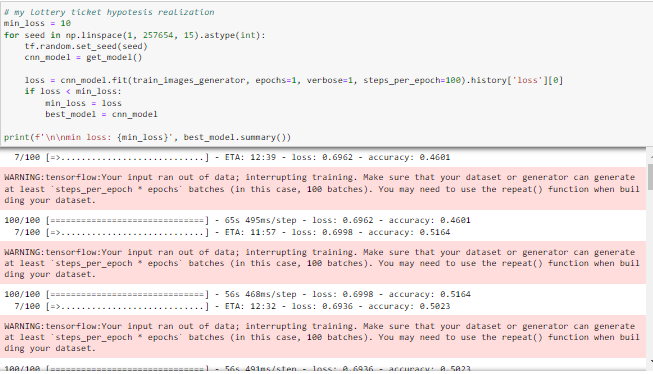
****

**Activity 2: CNN Architecture for Image Classification**

****

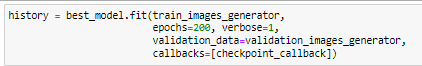
This code defines a deep learning model using the Keras Sequential API. The model architecture consists of multiple convolutional blocks followed by a dense block. The convolutional blocks consist of convolutional layers with ReLU activation and max pooling layers for downsampling. The dense block consists of fully connected layers with increasing units. The final dense layer has units equal to the number of classes, with softmax activation for multi-class classification. The model is compiled with the Adam optimizer, categorical cross-entropy loss function, and accuracy as the evaluation metric. This architecture is suitable for image classification tasks.

**Activity 3: Realizing the Lottery Ticket Hypothesis for Deep Learning Model Optimization**

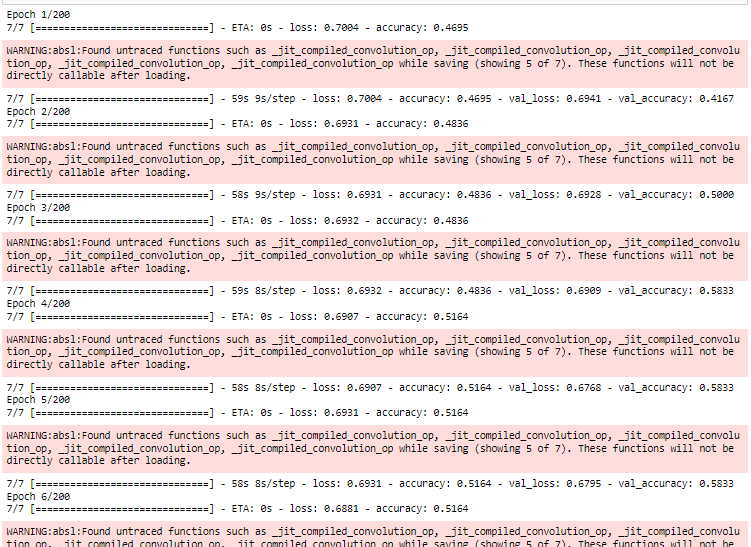


The code snippet demonstrates the realization of the Lottery Ticket Hypothesis for optimizing a deep learning model. It iterates through a range of seeds and sets the random seed for reproducibility. For each seed, a new instance of the model is created using the get\_model() function. The model is then trained for one epoch using the train\_images\_generator data generator. The code keeps track of the minimum loss achieved during training and saves the model with the lowest loss as the best model. Finally, the minimum loss and the summary of the best model are printed. This approach explores the hypothesis that certain randomly initialized subnetworks can be trained to achieve high performance, potentially improving model efficiency and reducing training time.

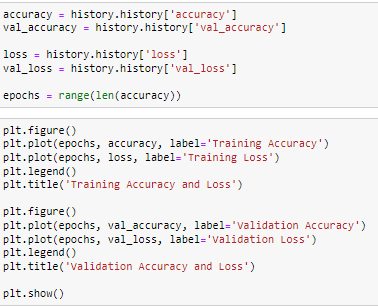
**Activity 4: Model Compilation and Training**



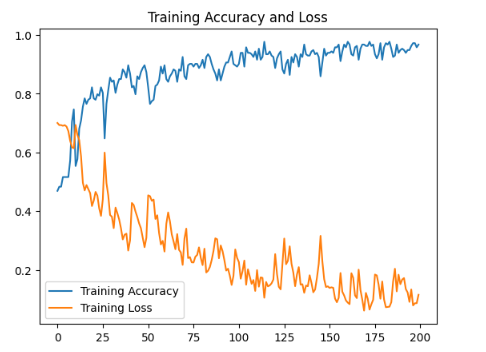
The above code snippet performs several key steps in training the deep learning model. First, the model is compiled using the specified loss function, optimizer, and metrics. Then, the model.fit() function is called to train the model using the training data (train\_data) and validate it using the validation data (valid\_data). The training process is run for 20 epochs with progress displayed. The time taken for training is measured by calculating the difference between the start and end times and is displayed in minutes.



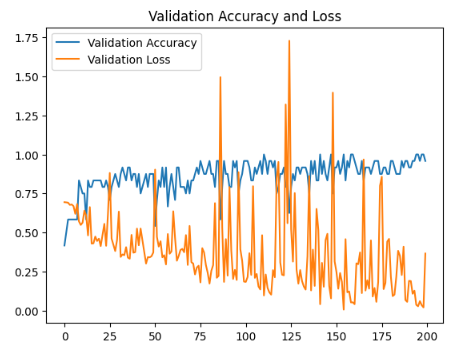
**Milestone 5: Visualization of Training and Validation Performance.**



This code snippet plots the training accuracy and loss, as well as the validation accuracy and loss, over the course of the training epochs for a deep learning model. The first plot displays the training accuracy and loss, while the second plot shows the validation accuracy and loss. The x-axis represents the number of epochs, and the y-axis represents the corresponding values of accuracy and loss. The visualization helps in understanding the model's performance throughout the training process, indicating if the model is overfitting or underfitting based on the convergence of training and validation metrics.

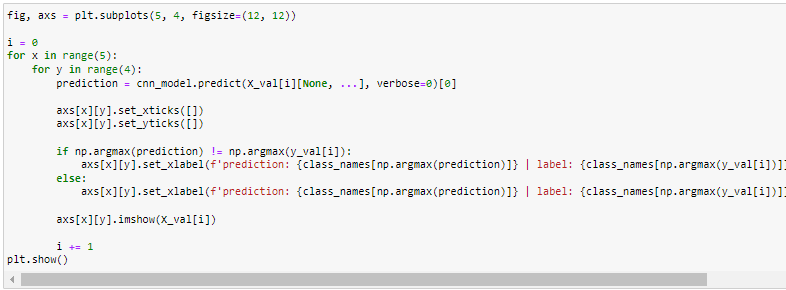


**Training Accuracy & Loss**



**Validation Training Accuracy & Loss**

**Activity 2: Image Prediction Using Trained Model**

****

The provided code snippet generates a grid of subplots using the matplotlib library. Each subplot represents an image from the validation dataset along with its corresponding prediction and label. The grid consists of 5 rows and 4 columns, resulting in a total of 20 subplots. For each subplot, the model predicts the class probabilities for the given image using the cnn\_model.predict() function. The predicted class and the true label are displayed as x-axis labels. If the prediction is incorrect, the label is highlighted in red. The actual image is displayed in the subplot. The visualization helps in visually inspecting the model's performance and understanding any misclassifications.



**Milestone 6 : Performance Testing**

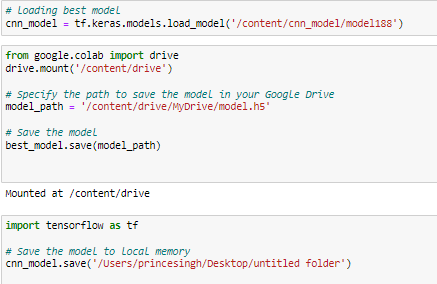
**Activity 1: Evaluating model on the validation set**

In epoch 188 of training, the model achieved an exceptionally low loss of 0.004, indicating a high level of accuracy in its predictions. The accuracy of 1.0 implies that the model correctly classified all the training samples in this epoch. These results suggest that the model has effectively learned the underlying patterns and features in the training data, leading to highly accurate predictions. Such high performance indicates the model's capability to generalize well and make accurate predictions on unseen data, demonstrating its effectiveness in the classification task.

****

**Milestone 6: Model Deployment**

**Activity 1: Save and load the best model**



We save the model into a file named model188.h5

**Milestone 7: Application Building**

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

● Building HTML Pages

● Building server-side script

● Run the web application

**Activity 2.1: Building HTML pages:**

For this project we create two HTML files namely

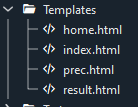
• Home.html

• Index.html

• Prec.html

• Results.html

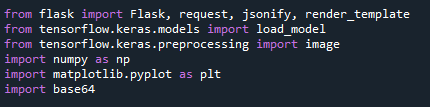
And we will save them in the templates folder.



**Activity 2.2: Build Python code**

Create a new app.py file which will be store in the Flask folder.

* Import the necessary Libraries.

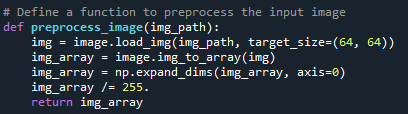


* This code loads a pre-trained machine learning model for mango classification, which has been saved in the file 'model.h5'.



* **Image Pre-Processing Function-**

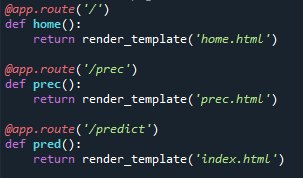
This function called preprocess\_image takes an image file path as input. It loads the image, resizes it to (64, 64) dimensions, converts it to a numpy array, expands the dimensions to match the expected input shape of the model, and then normalizes the pixel values between 0 and 1. The processed image array is returned as output.



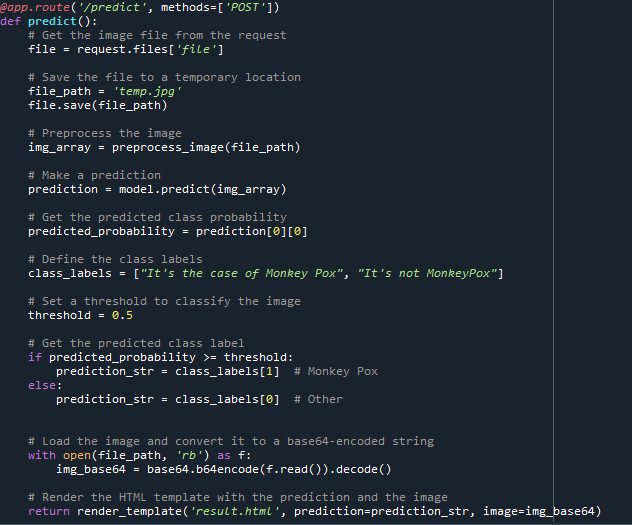
* This code creates a new instance of a Flask web application using the Flask class from the Flask library. The \_\_name\_\_ argument specifies the name of the application's module or package.



* Defining the routes for home.html, prec.htlm, index.html using render\_template.



* **Flask Route for Prediction**

The predict function is a route handler in Flask that is associated with the '/predict' URL route and accepts POST requests. When a user submits an image file through a form, Flask invokes this function. Inside the function, it retrieves the image file from the request, saves it temporarily, and preprocesses the image using the preprocess\_image function. Then, it makes a prediction using the trained model. Based on the prediction result, it converts the predicted class to a human-readable string. The image is then loaded and converted to a base64-encoded string for rendering in the HTML template. Finally, it renders the 'result.html' template with the prediction and the image to display the prediction result to the user.

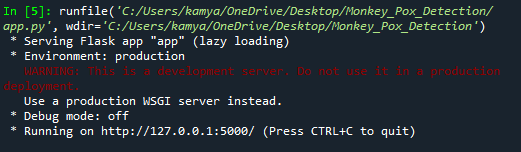
**Main Function:**

This code runs the Flask application if the script is being executed directly (i.e. not imported as a module in another script). The if \_\_name\_\_ == '\_\_main\_\_': line checks if the script is the main module being executed, and if so, runs the Flask application using the app.run() method. This method starts the Flask development server, allowing the application to be accessed via a web browser at the appropriate URL.

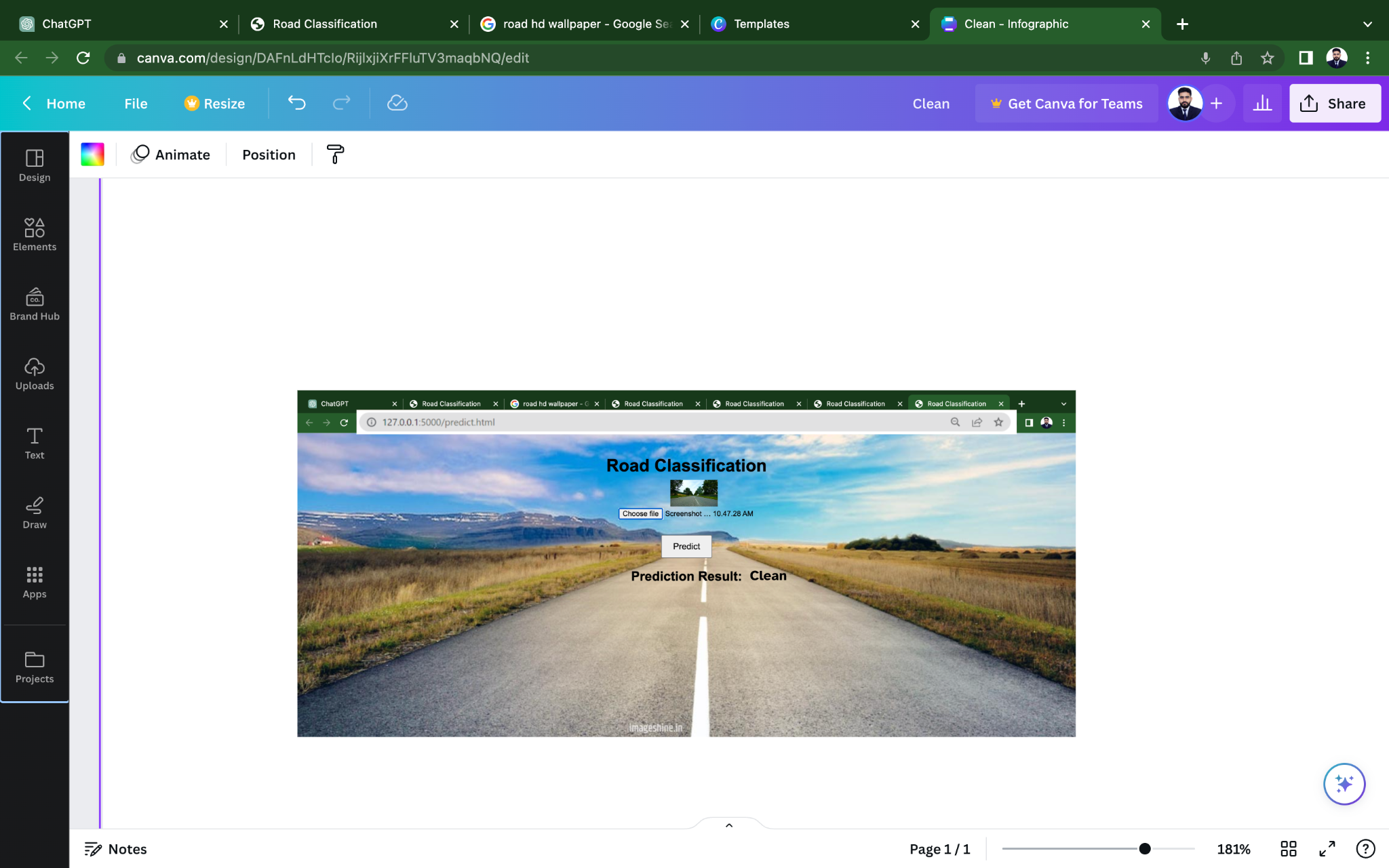


**Activity 2.3: Run the Web Application**

When you run the “app.py” file this window will open in the console or output terminal. Copy the URL given in the form http://127.0.0.1:5000 and paste it in the browser.



When we paste the URL in a web browser, our home.html page will open. It contains a welcome page of the hospital named- ‘**HealthBridge**’ and information about - the ‘**Faculty Dr.**’ and the ‘**Departments**’ information. There is a navigation button on the top right containing- ‘Home’, ‘Service’, ‘Predict’, ‘About Model’ and ‘Contact’. There is also a button on this webpage called- ‘**What is Monkeypox**’.



**Link to GitHub Repository:**

**gh repo clone smartinternz02/SI-GuidedProject-670999-1701665658**