A Project Report

On

RECIPE-LENS

Submitted by

Bishnu Sharma (10800220065)
Sheela Bhattacharjee (10800220066)
Biplab Gorain (10800221149)
Projjal Mitra (10800220059)

Under the Guidance of

Mr. Sudip Kumar De
Assistant Professor

Department of Information Technology

Asansol Engineering College
Asansol

Affiliated to

Maulana Abul Kalam Azad University of Technology

June, 2024

ASANSOL ENGINEERING COLLEGE

Kanyapur, Vivekananda Sarani, Asansol, Paschim Bardhaman, West Bengal - 713305
Phone: 225-3057, 225-2108 Telefax: (0341) 225-6334
E-mail: principal.aecwb@gmail.com Website: www.aecwb.edu.in

CERTIFICATE

Certified that this project report on "**Recipe-Lens**" is the bonafide work of **Bishnu Sharma (10800220065), Sheela Bhattacharjee (10800220066), Biplab Gorain (10800221149), Projjal Mitra (10800220059)** who carried out the project work under my supervision.

Mr. Sudip Kumar De
Assistant Professor
Asansol Engineering College

Anup Kumar Mukhopadhyay
Head of the Department
Information Technology
Asansol Engineering College

Department of Information Technology

Asansol Engineering College

Asansol

ACKNOWLEDGEMENT

It is our great privilege to express our profound and sincere gratitude to our Project Supervisor **Mr. Sudip Kumar De**, (Assistant Professor) for providing us with very cooperative and precious guidance at every stage of the present project work being carried out under his supervision. His valuable advice and instructions in carrying out the present study have been a very rewarding and pleasurable experience that has greatly benefitted us throughout our work.

We would also like to pay our heartiest thanks and gratitude to **Anup Kumar Mukhopadhyay**, HOD, and all the faculty members of the Information Technology Department, Asansol Engineering College for various suggestions being provided in attaining success in our work.

We would like to express our earnest thanks to our colleagues along with all technical staff of the Information Technology Department, Asansol Engineering College for their valuable assistance being provided during our project work.

Finally, we would like to express our deep sense of gratitude to our parents for their constant motivation and support throughout our work.

_ _
Bishnu Sharma (10800220065)

_ _
Sheela Bhattacharjee (10800220066)

_ _
Biplab Gorain (10800221149)

_ _
Projjal Mitra (10800220059)

Date: 15/05/2024
Place: Asansol

4th Year
Department Of
Information Technology

CONTENTS

List of Figures

CHAPTER 1: PROJECT SYNOPSIS

The "Recipe Lens" project integrates cutting-edge technologies to create a sophisticated and user-friendly tool for culinary enthusiasts. The image recognition component employs deep learning frameworks, such as TensorFlow to accurately identify various food items and extract details about their quantities and arrangements in the input image. Image processing techniques are then applied to estimate ingredient quantities, recognizing common units of measurement for conversion into a standardized format. The natural language processing model, utilizing libraries like NLTK, plays a pivotal role in constructing coherent recipes, considering cooking instructions, steps, and additional details. The seamless integration of these components, possibly facilitated by Django ensures an intuitive user interface, where users can upload food images and receive detailed recipes. The project's outcome not only offers a practical solution for recipe generation but also serves as a creative and inspiring tool for individuals seeking culinary innovation in the kitchen. The extensive testing, user feedback, and eventual deployment emphasize the commitment to delivering a reliable and enjoyable user experience.

CHAPTER 2: INTRODUCTION

In a world increasingly driven by technology and culinary innovation, the "Recipe-Lens" project stands at the intersection of computer vision and gastronomy, offering a unique solution to inspire home cooks and food enthusiasts alike. Imagine a scenario where capturing a tantalizing dish with your smartphone becomes the starting point for an instant and personalized recipe. This project envisions just that, leveraging cutting-edge techniques in image recognition and natural language processing to transform food images into detailed and comprehensible cooking instructions. Whether you're a seasoned chef looking for new ideas or an amateur cook seeking guidance, this innovative system aims to cater to a diverse audience with its intuitive and interactive approach to find recipes.

2.1 Problem Formulation

By integrating deep learning models, image processing algorithms, and linguistic patterns, the project strives to go beyond conventional recipe databases. “Recipe-Lens” aspires to analyse food images, identify ingredients, estimate quantities, and articulate step-by-step instructions seamlessly. The ultimate goal is to empower users with a tool that not only facilitates efficient meal planning but also fosters creativity in the kitchen, turning every cooking session into a personalized culinary adventure.

2.2 Motivation

The motivation behind the "Recipe-Lens" project stems from a desire to revolutionize the way individuals engage with cooking and recipe discovery. In a world increasingly driven by visual content, the project addresses the need for a seamless bridge between the visual appeal of food and the practicality of following a recipe. By harnessing the power of computer vision and natural language processing, this project aims to empower users with a unique and interactive tool that transforms the act of capturing a culinary creation into a comprehensive and personalized recipe. This endeavor is motivated by a vision to enhance the culinary experience, inspire creativity in the kitchen, and cater to a diverse range of users, from novice cooks seeking guidance to seasoned chefs looking for fresh inspiration.

2.3 Scope

The scope of the "Recipe Lens" project extends across various domains, encompassing advanced image recognition, quantitative analysis of ingredients, and the generation of coherent recipes through natural language processing. The project will delve into the intricacies of deep learning frameworks for robust image recognition, utilizing techniques to not only identify diverse food items but also discern their quantities and arrangements. The scope further includes the implementation of image processing algorithms for accurate quantification, considering various units of measurement commonly used in cooking. On the natural language processing front, the project will explore libraries to construct meaningful recipes that encapsulate cooking instructions, steps, and other relevant details. The user interface will be a crucial aspect, providing an accessible platform for users to effortlessly upload images and receive detailed recipes. Through testing, refinement, and deployment, the project aspires to make a significant impact by providing a novel, practical, and enjoyable solution to the culinary community.

CHAPTER 3: REQUIREMENT ANALYSIS

3.1 Functional Requirement

3.1.1 Image Upload: Users should be able to effortlessly upload food images through a user-friendly interface on the website without the need for authentication.

3.1.2 Image Processing: Incorporate image processing algorithms to analyze the uploaded image and identify various food items present.

3.1.3 Ingredient Quantification:

- Implement algorithms to estimate the quantities of identified ingredients in the uploaded image.
- Recognize and convert common units of measurement (e.g., cups, tablespoons) into a standardized format.

3.1.4 Deep Learning Image Recognition: Utilize a deep learning-based image recognition model to accurately identify and categorize different food items in the uploaded image.

3.1.5 Natural Language Processing (NLP):

- Develop an NLP model to generate a coherent recipe based on the identified ingredients and their quantities.
- The system should construct human-readable recipes with cooking instructions, steps, and additional details.

3.1.6 Recipe Generation: Integrate the NLP model with the identified ingredients and quantities to dynamically generate a detailed recipe based on recognized food items.

3.1.7 User Interface:

- Design an intuitive and user-friendly interface for seamless image upload and recipe display.
- Ensure responsiveness across various devices and browsers.

3.1.8 Testing and Quality Assurance: Conduct thorough testing to validate the accuracy of image recognition, ingredient quantification, and recipe generation.

3.1.9 Recipe Display: Present the generated recipe to the user in a clear and organized format, including ingredients, quantities, and step-by-step instructions

3.2 Software Requirement Specifications (SRS)

3.2.1 Hardware Requirement

- Processor – Intel(R) Core (TM) i7-3770 |CPU @ 3.40GHz
- RAM – 4 GB or above
- Hard Disk – Free disk space of above 6 GB
- Video Monitor (800 × 600 or higher resolution) with at least 256 colours (1024x768 High colour 16-bit recommended).

3.2.2 Software Requirement

- Front End: HTML, CSS
- Backend: Python, Machine Learning
- Operating System: Windows 7 & 8

3.2.3 Tools and Techniques Used

Visual Studio Code: Visual Studio Code is a source-code editor that can be used with a variety of programming languages, including Java, JavaScript, Go, Node.js and C++. It is based on the Electron framework which is used to develop Node.js Web applications that run on the Blink layout engine. Visual Studio Code employs the same editor component (codenamed "Monaco") used in Azure DevOps (formerly called Visual Studio Online and Visual Studio Team Services).

CHAPTER 4: MODEL ANALYSIS

DenseNet-201 is a convolutional neural network (CNN) architecture, part of the DenseNet family, which was proposed by Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger in their paper "Densely Connected Convolutional Networks" published in 2017. DenseNet architectures are known for their dense connections between layers, which promote feature reuse, reduce the number of parameters, and enhance gradient flow throughout the network.

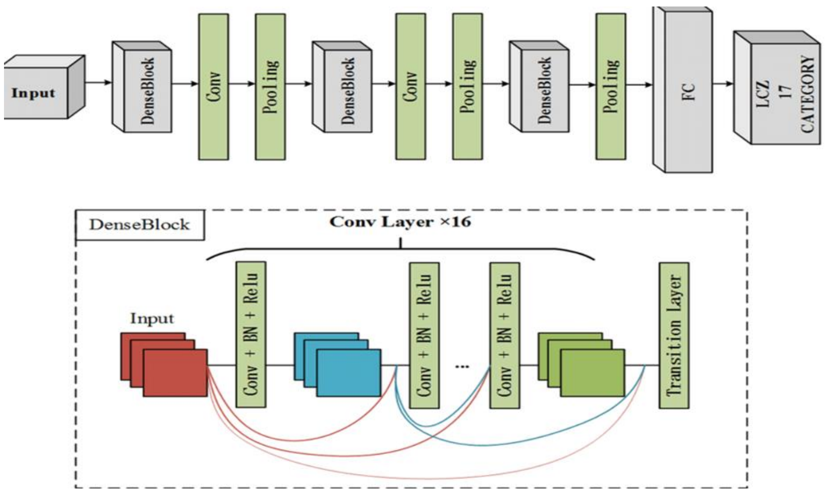


Fig 1.0: DenseNet-201 architecture

Here are the key features and components of the DenseNet-201 architecture:

- **Dense Connectivity:** DenseNet-201 employs dense connectivity between layers, where each layer receives feature maps from all preceding layers as input. This dense connectivity facilitates feature reuse, encourages gradient flow, and reduces the vanishing gradient problem commonly encountered in very deep neural networks.
- **Deep Architecture:** With 201 layers, DenseNet-201 is a deep neural network capable of learning hierarchical features from input data. The depth of the network allows it to capture intricate patterns and representations, making it suitable for tasks that demand a high level of abstraction and understanding, such as fine-grained image classification or object detection.
- **Bottleneck Layers:** Like other DenseNet variants, DenseNet-201 employs bottleneck layers within dense blocks. These bottleneck layers consist of 1x1 convolutions followed by 3x3 convolutions. The purpose of bottleneck layers is to reduce the number of input channels before the more computationally expensive

3x3 convolutions, thereby improving efficiency without sacrificing representational capacity.

- **Transition Layers:** DenseNet-201 includes transition layers between dense blocks, which serve to down sample feature maps and reduce the dimensionality of the network. These transition layers typically consist of batch normalization, 1x1 convolution, and average pooling operations. By controlling the growth of feature maps, transition layers help manage computational complexity and memory requirements.
- **Global Average Pooling (GAP):** After the last dense block, DenseNet-201 employs global average pooling to generate a fixed-size feature representation. Global average pooling computes the average value of each feature map channel across spatial dimensions, resulting in a compact feature vector. GAP reduces the spatial dimensions of the feature maps while preserving essential information, making the network more robust to spatial translations and distortions.
- **Efficient Parameter Utilization:** The dense connectivity in DenseNet-201 enables efficient parameter utilization by encouraging feature reuse. Unlike traditional architectures where each layer learns its own set of features independently, DenseNet shares features across layers, reducing redundancy and enabling the network to achieve high accuracy with fewer parameters.
- **State-of-the-Art Performance:** DenseNet-201 has demonstrated state-of-the-art performance on various benchmark datasets for image classification tasks. Its deep architecture, dense connectivity, and efficient parameter utilization contribute to its superior performance in terms of accuracy and generalization ability compared to many other CNN architectures.
- **Parameter Efficiency:** DenseNet-201 is known for its parameter efficiency, thanks to its dense connectivity pattern. Compared to traditional deep neural networks, DenseNet requires fewer parameters to achieve similar or better performance. This parameter efficiency makes DenseNet-201 more computationally affordable, particularly in scenarios where computational resources are limited.
- **Regularization:** DenseNet-201 benefits from implicit regularization due to its dense connectivity and feature reuse. The dense connections between layers act as a form of regularization by encouraging feature sharing and reducing the risk of overfitting. Additionally, techniques such as dropout or weight decay can be applied to further enhance regularization if needed.
- **Pre-Trained Models:** Pre-trained versions of DenseNet-201 are often available, trained on large-scale datasets like ImageNet. These pre-trained models can be

fine-tuned on domain-specific datasets or used as feature extractors for transfer learning tasks. Leveraging pre-trained models can significantly reduce the amount of labeled data and training time required to achieve good performance on specific tasks.

- **Memory Efficiency:** Despite its depth and parameter efficiency, DenseNet-201 can still pose challenges in terms of memory usage, especially when training on large datasets or with limited GPU memory. Techniques such as gradient checkpointing or memory-efficient implementations of convolution operations can help mitigate these memory requirements, enabling training on resource-constrained hardware.

How we used this model in our project:

- In our project, we have implemented a food recognition system that leverages a pre-trained DenseNet model for image classification. When a user uploads an image of food, the system utilizes the DenseNet model to predict the type or category of food present in the image. This prediction is made based on the features learned by the DenseNet model during its training on a large dataset of food images.
- Once the food item is predicted, our system retrieves the corresponding recipe from a curated collection of Indian recipes stored in a JSON file named 'Indian_recipes.json'. This JSON file contains a database of various Indian dishes along with their ingredients and cooking instructions.
- By integrating the DenseNet model into our project, we enable users to simply upload a food image, and our system automatically identifies the food item and provides the corresponding recipe. This functionality enhances user experience by simplifying the process of discovering and accessing recipes for different dishes.
- The use of pre-trained models like DenseNet allows us to capitalize on the knowledge and insights gained from extensive training on large datasets, leading to accurate food recognition results. Additionally, the availability of a diverse range of Indian recipes in our database enhances the utility and applicability of our system for users interested in Indian cuisine.
- Overall, the integration of the DenseNet model into our project facilitates seamless food recognition and recipe retrieval, contributing to an enhanced user experience and greater utility of the system for individuals seeking culinary inspiration and guidance.

```
# Load the pre-trained DenseNet201 model
model = tf.keras.applications.DenseNet201(include_top=False,
weights='imagenet', input_shape=(256, 256, 3), pooling='avg', classes=1000)
```

CHAPTER 5: DESIGN

5.1 Architectural Design (Project Flow /architecture with description)

Project flow –

5.1.1 User Input:

- **Upload Image:** Users upload food images through a web interface or mobile application.
- **Capture Image:** Users also can capture an image of the food through the device camera.

5.1.2 Image Processing:

- **Preprocessing:** Preprocesses an image and extracts its features using the DenseNet201 model.
- **Image Encoding:** The image is encoded to a 1-D NumPy array.

5.1.3 Similarity Checking:

- **Calculate Similarity:** Calculate similarity using cosine distance between 1-D arrays from the already encoded foods.
- **Pick the Best Match:** All the most similar foods (less distance between them) is picked up for the result.

5.1.4 Recipe Extraction:

- **Get Recipe:** Get the recipes of all the picked foods from the database.
- **Steps and Instructions:** Construct cooking instructions, order of steps, and additional details like cooking time and temperature.

5.1.5 Web Interface:

- **Display:** Show the generated recipe along with the original image and cooking instructions on the web interface.
- **User Interaction:** Allow users to interact with the system, save recipes, and provide feedback.

5.1.6 Future Enhancements (Optional):

- **Integration with Smart Appliances:** Collaborate with smart kitchen appliances for automated cooking based on generated recipes.

- **Community Features:** Enable users to share recipes, collaborate on variations, and engage in a community around the platform.

5.1.7 Deployment:

- **Public Availability:** Deploy the system for public use, whether as a web application or a standalone tool.
- **Accessibility:** Ensure that the system is easily accessible and user-friendly.

5.2 Data Flow Diagram:

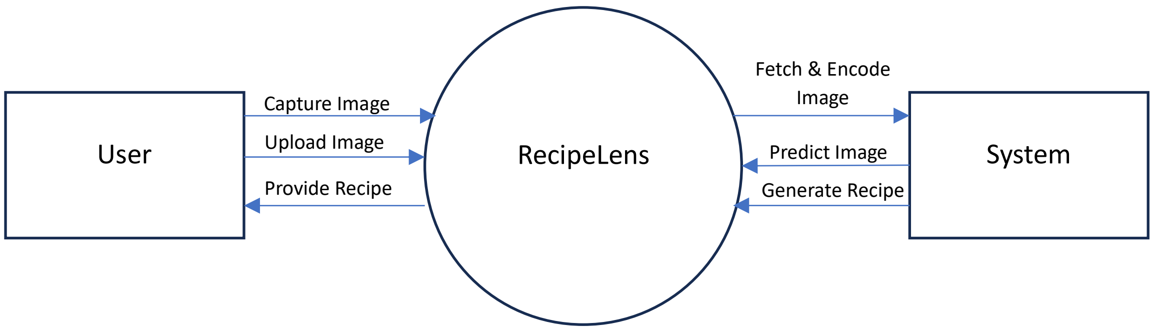


Fig 2.0: Lavel Zero DFD of RecipeLens

5.3 Use Case Diagram:

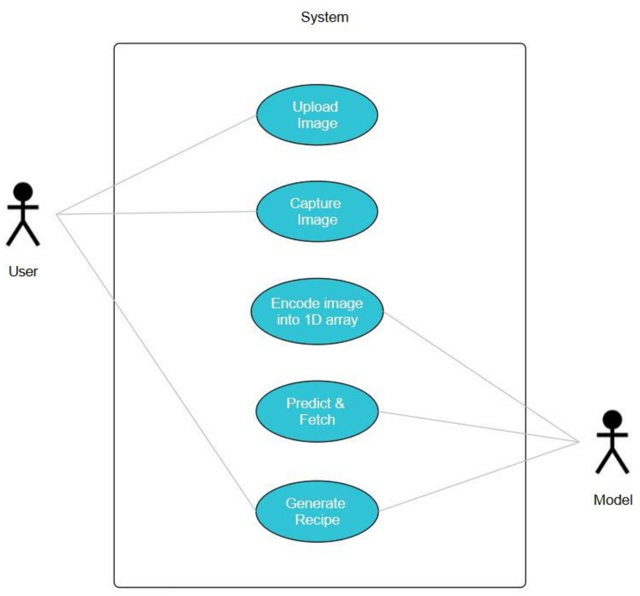


Fig 3.0: Use Case Diagram of RecipeLens

5.4 Data Scraping:

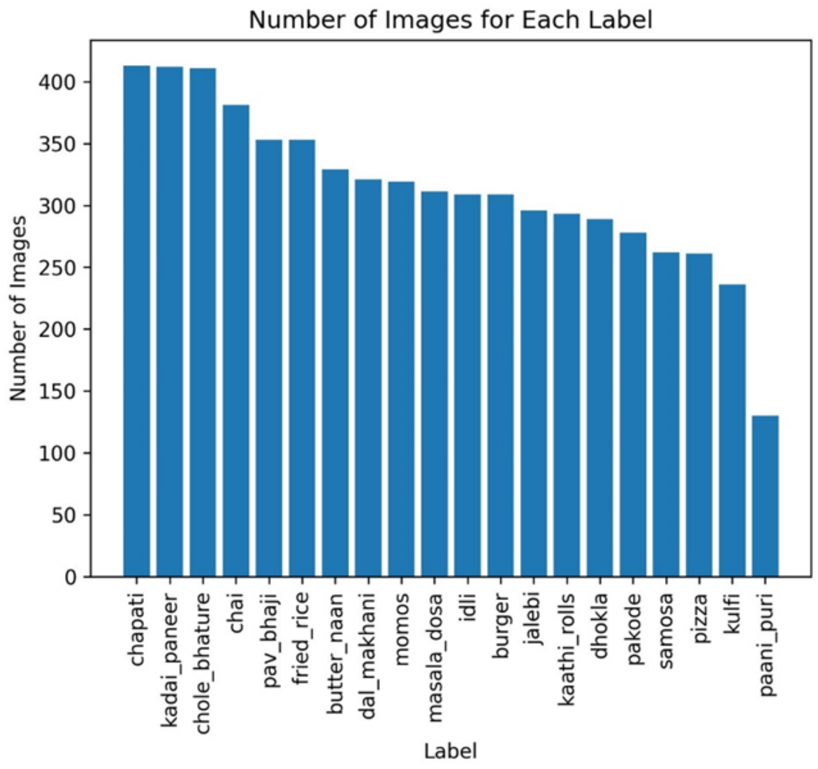
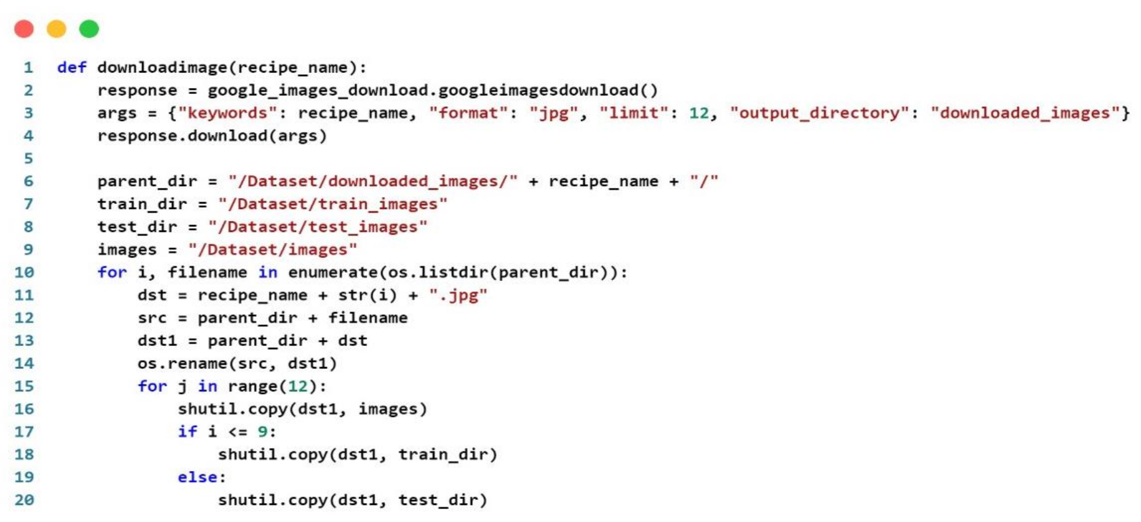


Fig 4.0: Distribution of images across the labels

- **Image Downloading Function:**

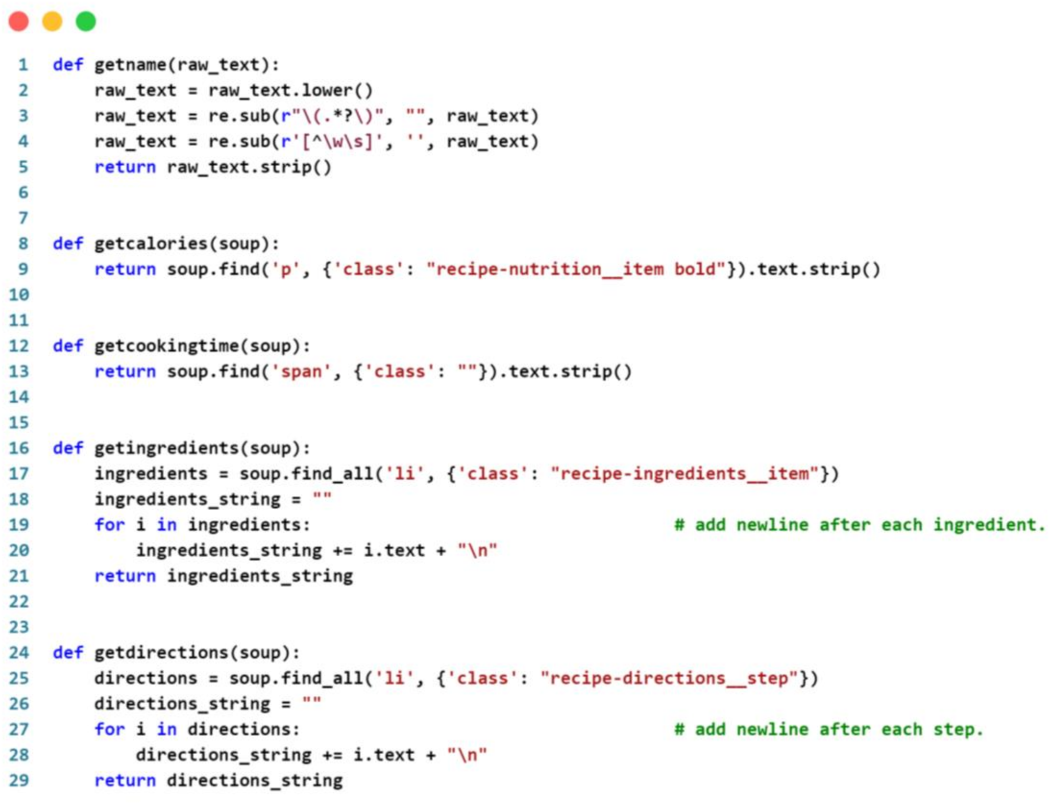


```python
def downloadimage(recipe_name):
    response = google_images_download.googleimagesdownload()
    args = {"keywords": recipe_name, "format": "jpg", "limit": 12, "output_directory": "downloaded_images"}
    response.download(args)

    parent_dir = "/Dataset/downloaded_images/" + recipe_name + "/"
    train_dir = "/Dataset/train_images"
    test_dir = "/Dataset/test_images"
    images = "/Dataset/images"
    for i, filename in enumerate(os.listdir(parent_dir)):
        dst = recipe_name + str(i) + ".jpg"
        src = parent_dir + filename
        dst1 = parent_dir + dst
        os.rename(src, dst1)
        for j in range(12):
            shutil.copy(dst1, images)
            if i <= 9:
                shutil.copy(dst1, train_dir)
            else:
                shutil.copy(dst1, test_dir)
```

Fig 5.0: Function for downloading image from google

The downloadimage function downloads images related to the given recipe name from Google using the google_images_download library. It then renames and copies these images to appropriate directories for further processing.

- **Functions for Text Scraping:**

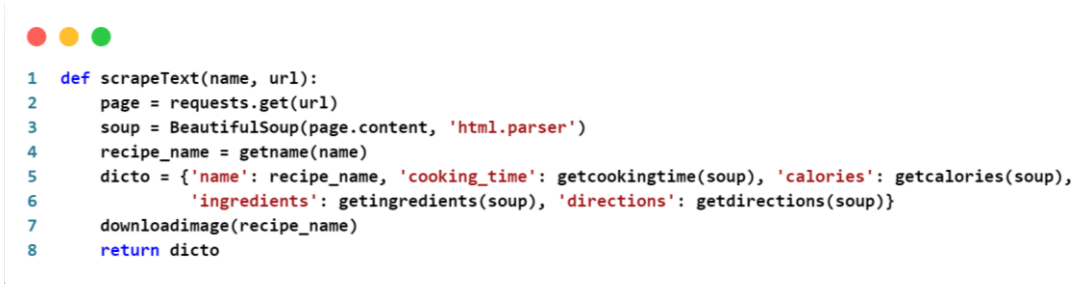


```python
def getname(raw_text):
    raw_text = raw_text.lower()
    raw_text = re.sub(r"\(.*?\)", "", raw_text)
    raw_text = re.sub(r'[^\w\s]', '', raw_text)
    return raw_text.strip()


def getcalories(soup):
    return soup.find('p', {'class': "recipe-nutrition__item bold"}).text.strip()


def getcookingtime(soup):
    return soup.find('span', {'class': ""}).text.strip()


def getingredients(soup):
    ingredients = soup.find_all('li', {'class': "recipe-ingredients__item"})
    ingredients_string = ""
    for i in ingredients:                              # add newline after each ingredient.
        ingredients_string += i.text + "\n"
    return ingredients_string


def getdirections(soup):
    directions = soup.find_all('li', {'class': "recipe-directions__step"})
    directions_string = ""
    for i in directions:                               # add newline after each step.
        directions_string += i.text + "\n"
    return directions_string
```

Fig 6.0: Function for extract specific information

These functions are used to extract specific information (recipe name, calories, cooking time, ingredients, directions) from the HTML content of a recipe webpage. They utilize Beautiful Soup, a Python library for web scraping, to parse and navigate through HTML documents.

- **Scraping Recipe Information:**

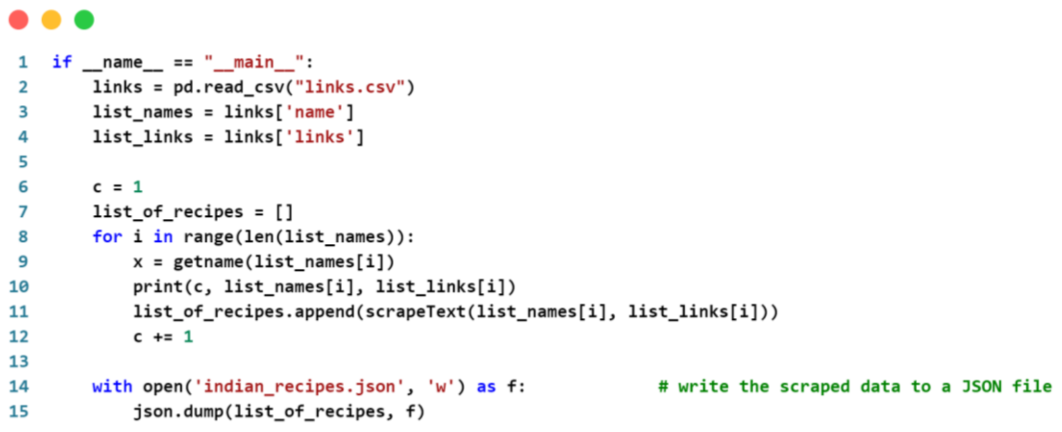


```python
def scrapeText(name, url):
    page = requests.get(url)
    soup = BeautifulSoup(page.content, 'html.parser')
    recipe_name = getname(name)
    dicto = {'name': recipe_name, 'cooking_time': getcookingtime(soup), 'calories': getcalories(soup),
             'ingredients': getingredients(soup), 'directions': getdirections(soup)}
    downloadimage(recipe_name)
    return dicto
```

Fig 7.0: Function for scraping recipe information

The scrapeText function scrapes recipe information from a given URL. It first makes an HTTP request to the URL, then parses the HTML content using Beautiful Soup. After extracting recipe name and other details, it downloads images related to the recipe and returns all the scraped information as a dictionary.

- **Main Execution:**



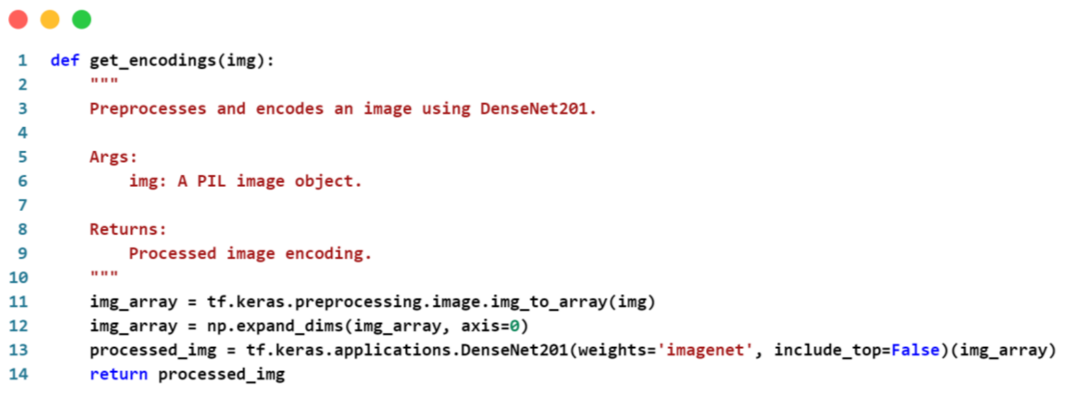
```python
if __name__ == "__main__":
    links = pd.read_csv("links.csv")
    list_names = links['name']
    list_links = links['links']

    c = 1
    list_of_recipes = []
    for i in range(len(list_names)):
        x = getname(list_names[i])
        print(c, list_names[i], list_links[i])
        list_of_recipes.append(scrapeText(list_names[i], list_links[i]))
        c += 1

    with open('indian_recipes.json', 'w') as f:            # write the scraped data to a JSON file
        json.dump(list_of_recipes, f)
```

Fig 8.0: Main function of data scraping

In the main block of the code, it reads recipe names and URLs from a CSV file. Then, it iterates through each recipe, scrapes information using the scrapeText function, and appends the scraped data to a list. Finally, it writes the collected data to a JSON file named indian_recipes.json.

5.3 Data Encoding:

- **Image Encoding Function:**

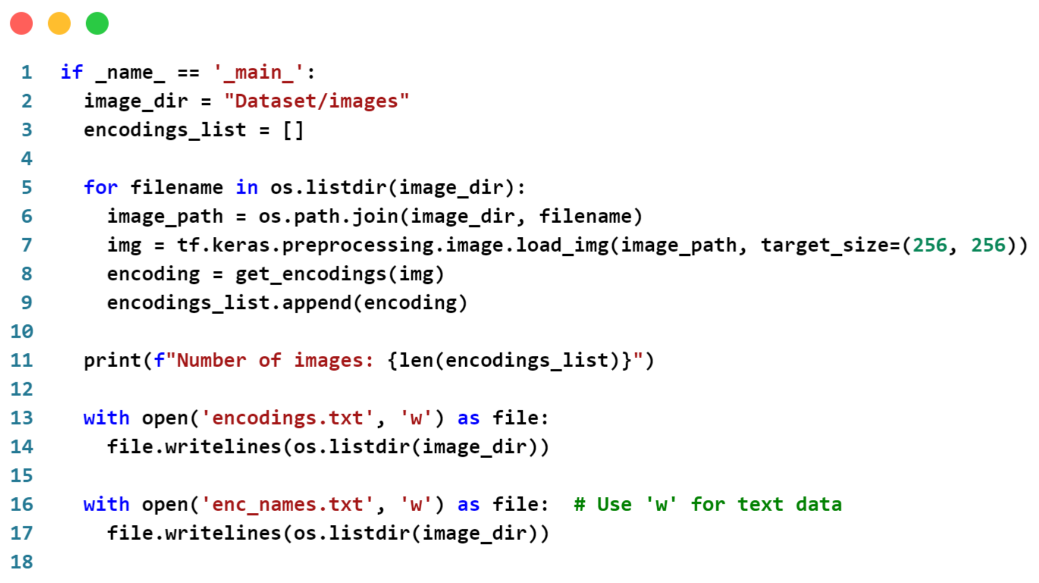


```python
def get_encodings(img):
    """
    Preprocesses and encodes an image using DenseNet201.

    Args:
        img: A PIL image object.

    Returns:
        Processed image encoding.
    """
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    processed_img = tf.keras.applications.DenseNet201(weights='imagenet', include_top=False)(img_array)
    return processed_img
```

Fig 9.0: Function for encoding images

This function preprocesses and encodes an image using the DenseNet201 model. It takes a PIL image object as input, converts it to a NumPy array, adds a batch dimension, and then passes it through the DenseNet201 model to extract features.

- **Main Execution:**



```python
if _name_ == '_main_':
  image_dir = "Dataset/images"
  encodings_list = []

  for filename in os.listdir(image_dir):
    image_path = os.path.join(image_dir, filename)
    img = tf.keras.preprocessing.image.load_img(image_path, target_size=(256, 256))
    encoding = get_encodings(img)
    encodings_list.append(encoding)

  print(f"Number of images: {len(encodings_list)}")

  with open('encodings.txt', 'w') as file:
    file.writelines(os.listdir(image_dir))

  with open('enc_names.txt', 'w') as file:  # Use 'w' for text data
    file.writelines(os.listdir(image_dir))

```

Fig 10.0: Main function of encoding images

In the main block of the code, it iterates through each image in the specified directory (Dataset/images). For each image, it loads, resizes, and then encodes it using the get_encodings function. The resulting encodings are stored in a list. After processing all images, it prints the number of images processed. Finally, it writes the filenames of the images to two separate text files (encodings.txt and enc_names.txt).

CHAPTER 6: IMPLEMENTATION

6.1 Framework

Django: Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It's designed to help developers build web applications quickly and efficiently by providing a set of tools and features for handling common web development tasks.

- **Here are some key aspects of the Django framework:**
    - ✓ **Model-View-Template (MVT) Architecture:** Django follows the MVT architecture pattern, which is similar to the Model-View-Controller (MVC) pattern. In Django, the model represents the data structure, the view handles the presentation logic, and the template manages the user interface.
    - ✓ **Object-Relational Mapping (ORM):** Django includes a built-in ORM that maps database tables to Python objects, allowing developers to interact with the database using Python code rather than SQL queries directly. This makes database interactions more intuitive and less error-prone.
    - ✓ **Admin Interface:** Django provides a powerful admin interface out-of-the-box, which allows developers to manage site content and user data dynamically without writing additional code. It's highly customizable and can be extended to suit specific project requirements.
    - ✓ **URL Routing:** Django uses a URL routing mechanism to map URL patterns to view functions, enabling developers to define clean and flexible URL structures for their web applications. This makes it easy to organize and manage different parts of a website.
    - ✓ **Template System:** Django's template system allows developers to create HTML templates with embedded Python code, making it easy to generate dynamic web pages. Templates support template inheritance, allowing developers to reuse and extend templates across multiple pages.
    - ✓ **Form Handling:** Django provides a powerful form handling library that simplifies the process of validating and processing user input from HTML forms. It includes built-in form classes and validation methods to streamline form creation and processing.
    - ✓ **Security Features:** Django comes with built-in security features to help developers protect their web applications from common security vulnerabilities, such as cross-site scripting (XSS), cross-site request forgery (CSRF), and SQL

injection. These features include built-in protection mechanisms and best practices for secure coding.

- ✓ **Scalability and Performance:** Django is designed to be scalable and performant, allowing developers to build web applications that can handle high traffic and large datasets. It provides caching mechanisms, database optimization tools, and other performance-enhancing features to improve application scalability and speed.

6.2 Frontend

- **Image Upload Area:**
  - ✓ Users can upload images or take pictures.
  - ✓ Displays the uploaded image.
  - ✓ Contains a form with options to upload or take a picture.
  - ✓ On form submission, generates a recipe based on the uploaded image.

```
main.html -
<div class="main_content">
<div class="image-upload-area">
    <center>
        <h2 style="width: 100%;">A food image to recipe converter for Indian Food</h2>
    </center>
    <div class="uploaded-image-display">
        <img id="up-image" src="data:image/png;base64,{{uploaded_image}}">
    </div>
    <div class="image_form">
        <form method="post" enctype="multipart/form-data"
            style="display:flex;flex-direction:column; justify-content:center;align-items:center;
gap:1em">
            {% csrf_token %}
            <div style="display: flex;align-items:center; gap:1em; ">
                <label class="upload_button" style="width: fit-content;">
                    Upload Image
                    <input type="file" name="image" accept="image/*" id="id_image"
                        onchange="document.getElementById('up-image').src =
window.URL.createObjectURL(this.files[0])">
                </label>&nbsp;&nbsp;&nbsp;
                <label class="upload_button" style="width: fit-content;">
                    Take a Picture
                    <input type="file" name="image" accept="image/*" capture="environment"
id="cap_image"
                        onchange="document.getElementById('up-image').src =
window.URL.createObjectURL(this.files[0])">
                </label>
            </div>
            <button class="btn" type="submit">
                <svg height="24" width="24" fill="#FFFFFF" viewBox="0 0 24 24" data-name="Layer 1"
id="Layer_1"
                    class="sparkle">
                    <path
                        d="M10,21.236,6.755,14.745.264,11.5,6.755,8.255,10,1.76413.245,6.491L19.736,11
.5l-6.491,3.245ZM18,21l1.5,3L21,21l3-1.5L21,18l-1.5-3L18,18l-3,1.5ZM19.333,4.667,20.5,7l1.167-
2.333L24,3.5,21.667,2.333,20.5,0,19.333,2.333,17,3.5Z">
                    </path>
                </svg>
                <span class="text">Generate Recipe</span>
            </button>
            <!-- <button class="process_button" type="submit" style="width: fit-content;">Generate
Recipe</button> -->
        </form>
    </div>
</div>
```

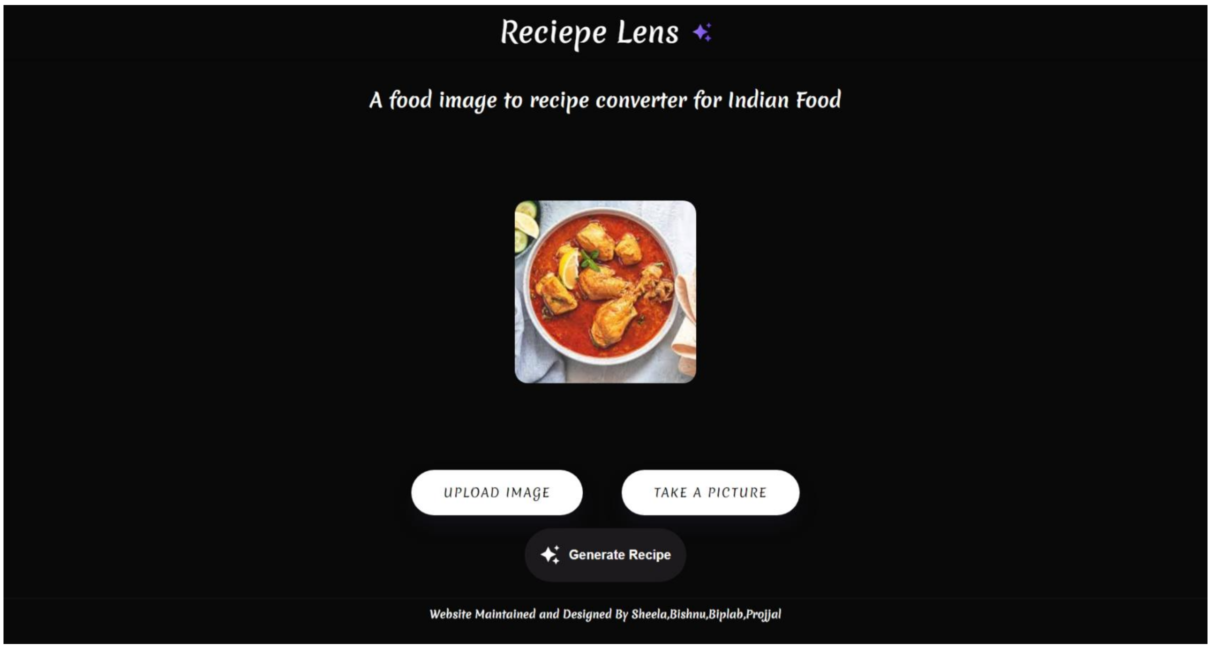
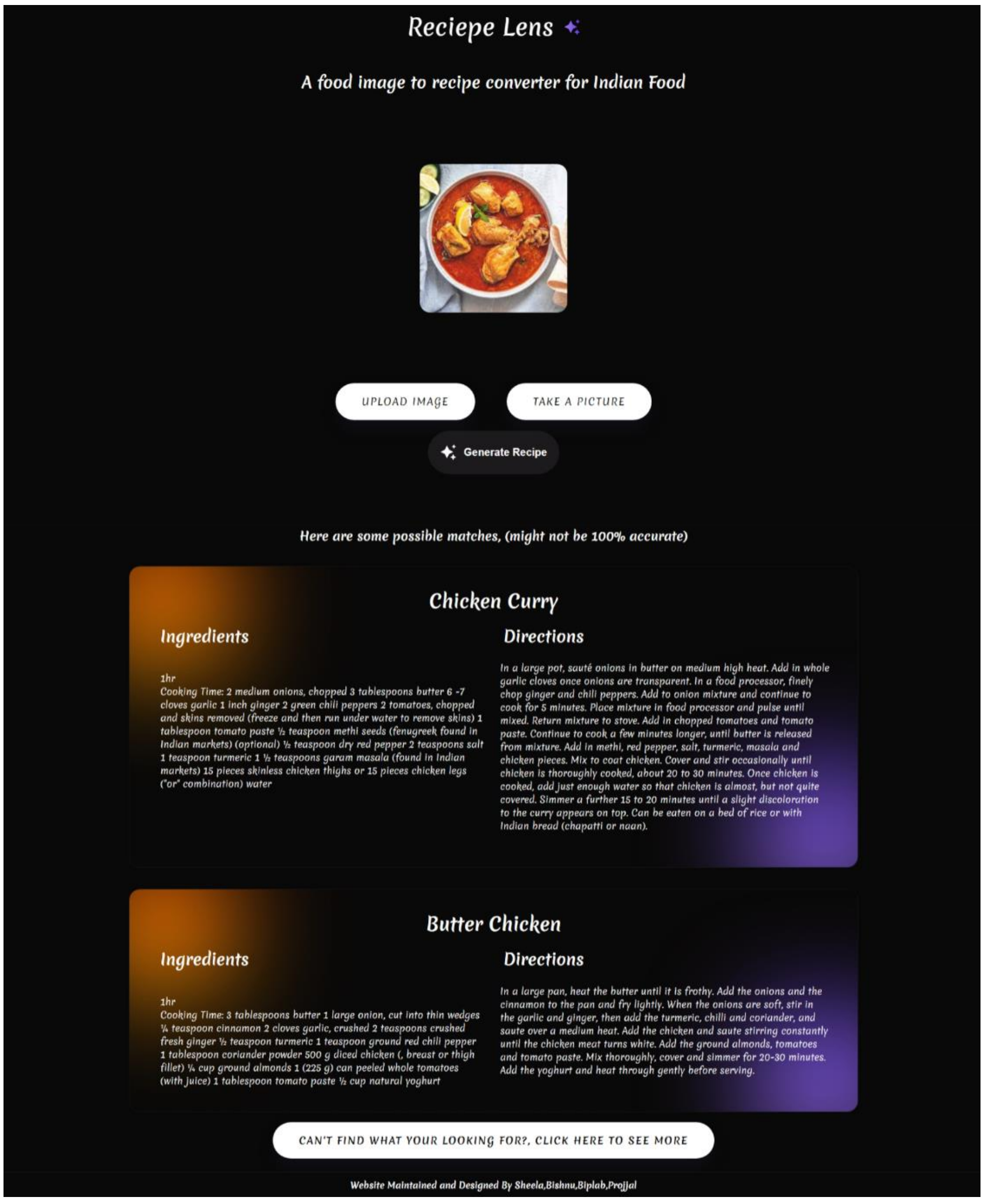

Reciepe Lens
A food image to recipe converter for Indian Food
UPLOAD IMAGE
TAKE A PICTURE
Generate Recipe
Website Maintained and Designed By Sheela,Bishnu,Biplab,Projjal

Fig 11.0: Preview of the website after uploading an image

- **Results Display Area:**
    - ✓ Shows generated recipes or search results.
    - ✓ If recipes are returned, displays each recipe with its name, ingredients, cooking time, and directions.
    - ✓ Provides an option to display more results if available.

main.html -

```
<div class="results-display-area">
    {% if recipe_list_to_return %}
    <center style="margin-top: 5em;">
        <h3 onclick="display_more()">Here are some possible matches, (might not be 100% accurate)</h3>
    </center>
    {% for x in recipe_list_to_return %}
    <div class="recipe-card">
        <div class="container-blur">
            <div class="blur-top"></div>
            <div class="blur-bottom"></div>
        </div>
        <div style="display:flex;flex-direction:column; justify-content:center;align-items:center;
gap:1em">

            <div class="first_section">
                <h2 class="recipe_name">{{ x.0 }}</h2>
            </div>
            <div style="display: flex;justify-content:center;flex-wrap:wrap; ">
                <div class="second_section"
                    style="flex:1;min-width:300px;display:flex; flex-direction:column; justify-
content:center; gap:1em">
                    <h2 style="padding-inline: 5px;">Ingredients</h2>
                    <p style="padding-inline: 5px;">{{ x.2 }}<br>Cooking Time:
                        {{ x.3 }}
                    </p>

                </div>
                <div class="third_section"
                    style="flex:1;min-width:300px;flex-direction:column; justify-content:center;align-
items:center; gap:1em">
                    <h2 style="padding-inline: 5px;">Directions</h2>
                    <p class="ing_dir_display_area">{{ x.4 }}</p>
                    <p class="ing_dir_display_area">{{ x.5 }}</p>
                </div>
            </div>
        </div>
    </div>
    {% endfor %}
    <center>
        <h3 class="upload_button" style="width: fit-content;margin-top:1em;"
onclick="display_more()">Can't
            find what your looking for?, Click here to see more</h3>
    </center>
    {% endif %}
</div>
```



Here are some possible matches, (might not be 100% accurate)

Chicken Curry

Ingredients

1hr
Cooking Time: 2 medium onions, chopped 3 tablespoons butter 6 -7 cloves garlic 1 inch ginger 2 green chili peppers 2 tomatoes, chopped and skins removed (freeze and then run under water to remove skins) 1 tablespoon tomato paste ½ teaspoon methi seeds (fenugreek found in Indian markets) (optional) ½ teaspoon dry red pepper 2 teaspoons salt 1 teaspoon turmeric 1 ½ teaspoons garam masala (found in Indian markets) 15 pieces skinless chicken thighs or 15 pieces chicken legs ("or" combination) water

Directions

In a large pot, sauté onions in butter on medium high heat. Add in whole garlic cloves once onions are transparent. In a food processor, finely chop ginger and chili peppers. Add to onion mixture and continue to cook for 5 minutes. Place mixture in food processor and pulse until mixed. Return mixture to stove. Add in chopped tomatoes and tomato paste. Continue to cook a few minutes longer, until butter is released from mixture. Add in methi, red pepper, salt, turmeric, masala and chicken pieces. Mix to coat chicken. Cover and stir occasionally until chicken is thoroughly cooked, about 20 to 30 minutes. Once chicken is cooked, add just enough water so that chicken is almost, but not quite covered. Simmer a further 15 to 20 minutes until a slight discoloration to the curry appears on top. Can be eaten on a bed of rice or with Indian bread (chapatti or naan).

Butter Chicken

Ingredients

1hr
Cooking Time: 3 tablespoons butter 1 large onion, cut into thin wedges ¼ teaspoon cinnamon 2 cloves garlic, crushed 2 teaspoons crushed fresh ginger ½ teaspoon turmeric 1 teaspoon ground red chili pepper 1 tablespoon coriander powder 500 g diced chicken (, breast or thigh fillet) ¼ cup ground almonds 1 (225 g) can peeled whole tomatoes (with juice) 1 tablespoon tomato paste ½ cup natural yoghurt

Directions

In a large pan, heat the butter until it is frothy. Add the onions and the cinnamon to the pan and fry lightly. When the onions are soft, stir in the garlic and ginger, then add the turmeric, chilli and coriander, and saute over a medium heat. Add the chicken and saute stirring constantly until the chicken meat turns white. Add the ground almonds, tomatoes and tomato paste. Mix thoroughly, cover and simmer for 20-30 minutes. Add the yoghurt and heat through gently before serving.

Website Maintained and Designed By Sheela,Bishnu,Biplab,Projjal

Fig 12.0: Preview of the website after generating recipe

- **JavaScript Functionality:**
  - ✓ display_more(): Toggles the display of additional search results.
  - ✓ load_demo_image(image_link): Loads a demo image when clicked.

```
main.html -
<script>
    function display_more() {
        var x = document.getElementById('extra_results');
        if (x.style.display === "none") {
            x.style.display = "block";
        } else {
            x.style.display = "none";
        }
    }
    function load_demo_image(image_link) {
        var frm = document.getElementById('up-image');
        frm.src = image_link;
    }
</script>
```

6.3 Backend

- **Home Page View Function:**
  - ✓ This function (home_page) defines the logic for the home page of the web application.
  - ✓ It handles both GET and POST requests.
  - ✓ When a POST request is received (typically after form submission), it processes the uploaded image, extracts recipes using the get_recipes function, matches them with data from indian_recipes.json, and prepares a list of recipes to return.
  - ✓ When a GET request is received (initial page load), it creates an empty form.
  - ✓ Finally, it renders the home page template (home.html) with the form, uploaded image (if any), and recipe lists to display.

```
Views.py -
def home_page(request):
    raw_image = None
    uploaded_image = None
    recipe_list_to_return = []

    # Check if the request method is POST
    if request.method == 'POST':
        # Create a form instance with the submitted data
        form = ImageUploadForm(request.POST, request.FILES)
        # Check if the form is valid
        if form.is_valid():
            # Retrieve the raw image data from the form
            raw_image = form.cleaned_data['image']
            # Encode the raw image data to base64 format
            uploaded_image = base64.b64encode(
                raw_image.file.read()).decode('ascii')
            # Open the raw image using PIL
            raw_image = Image.open(raw_image)
            # Get recipes from the uploaded image
            recipe_list = get_recipes(raw_image)

            # Adjust the path to indian_recipes.json
            json_file_path = os.path.join(
                current_dir, 'static', 'main', 'indian_recipes.json')
            # Load JSON data from the file
            recipes_data = json.load(open(json_file_path))
            # Iterate through the recipe list
            for i in range(len(recipe_list)):
                name = recipe_list[i]
                # Filter the recipes data to find matching recipe by name
                matching_recipes = list(
                    filter(lambda x: x["name"] == name, recipes_data))
                if len(matching_recipes) != 0:
                    # Extract recipe details if a match is found
                    matching_recipe = matching_recipes[0]
                    calories = matching_recipe['calories']
                    cooking_time = matching_recipe['cooking_time']
                    ingredients = matching_recipe['ingredients']
                    directions = matching_recipe['directions']
                    # Prepare a list of recipe details to return
                    list_to_append = [string.capwords(
                        name), calories, cooking_time, ingredients, directions]
                    recipe_list_to_return.append(list_to_append)
    else:
        # If request method is not POST, create an empty form
        form = ImageUploadForm()
    # Render the home page template with necessary data
    return render(request, 'main/home.html', {'form': form, 'uploaded_image': uploaded_image,
                                              'recipe_list_to_return': recipe_list_to_return[:4],
                                              'similar_recipe_list': recipe_list_to_return[4:10]})
```

- **Image Encoding Functions:**
  - ✓ This function preprocesses an input image and extracts its features using the pre-trained DenseNet201 model. It returns a 1-D NumPy array representing the image's encoding.

```
encoding.py -
def get_encodings(img):
    """
    Preprocesses an image and extracts its features using the DenseNet201 model.
    Args:
        img: A PIL image object.
    Returns:
        A 1-D NumPy array representing the image's encoding.
    """

    # Convert the image to a NumPy array
    img_array = tf.keras.preprocessing.image.img_to_array(img)
    # Resize the image to the model's input shape
    img_array = tf.keras.preprocessing.image.smart_resize(
        img_array, size=(256, 256))
    # Expand the dimension for batch processing (even though it's a single image)
    img_array = np.expand_dims(img_array, axis=0)
```

```python
    # Preprocess the image using the model's preprocessing function
    img_preprocessed = tf.keras.applications.densenet.preprocess_input(
        img_array)
    # Extract the encoding using the pre-trained DenseNet201 model
    encoding = model.predict(img_preprocessed)
    # Flatten the encoding to a 1-D array
    encoding = encoding.flatten()

    return encoding
```

- **Recipe Extraction Function:**
  - ✓ This function calculates the cosine similarity between the encoding of an input image and a list of stored encodings. It then returns a list of the top 10 most similar recipe names based on this similarity measure.

```python
encoding.py -
def get_recipes(img):
    """
    Calculates cosine similarity between the image encoding and the stored encodings,
    and returns a list of top 10 most similar recipe names.
    """
    enc = get_encodings(img)
    similarity_list = []
    recipe_names_list = []
    print("Shape of enc_list:", enc_list.shape)  # Add this line
    print("Shape of enc:", enc.shape)  # Add this line

    # Iterate through each encoding in the stored encodings list
    for i in range(len(enc_list)):
        # Calculate cosine similarity between the current encoding and the image encoding
        similarity = cosine(enc_list[i].flatten(), enc.flatten())
        similarity_list.append(1 - similarity)

    # Sort the similarity list along with corresponding recipe names
    sorted_list = sorted(zip(similarity_list, names_list), reverse=True)

    # Extract top 10 most similar recipe names
    for i in range(len(sorted_list)):
        name_in_list = sorted_list[i][1]
        # Remove numeric suffix from recipe name
        s = re.sub(r'[0-9]+.jpg', "", name_in_list)
        # Add unique recipe names to the list
        if s not in recipe_names_list:
            recipe_names_list.append(s)
            # Stop when 10 unique recipe names are collected
            if len(recipe_names_list) >= 10:
                break
    return recipe_names_list
```

CHAPTER 7: Conclusion & Future Scope

7.1 Conclusion

In conclusion, the "Recipe-Lens" project represents a significant stride in the realm of culinary technology, marrying the visual allure of food with the precision of artificial intelligence. Through the amalgamation of state-of-the-art image recognition and natural language processing, this project has endeavoured to transform static images into dynamic, interactive cooking experiences. The journey from recognizing ingredients in a mere snapshot to generating coherent and personalized recipes is a testament to the potential of technology in reshaping our culinary adventures. As we reflect on the development process, it becomes evident that the project not only serves as a practical tool for meal planning but also as a source of inspiration for culinary enthusiasts across diverse skill levels.

In essence, the "Recipe Lens" project not only encapsulates the capabilities of modern technology but also embodies the spirit of culinary artistry, making every cooking endeavour an exciting and unique experience. It stands as a testament to the evolving intersection of technology and gastronomy, where the boundaries between the virtual and culinary worlds blur, paving the way for a more interactive and delightful cooking future.

7.2 Future Scope

Our project opens up a myriad of possibilities for future enhancements and expansions, pushing the boundaries of technology in the culinary domain. The following avenues represent potential future developments and advancements for this innovative project:

Enhanced Recognition Capabilities:

Continual improvement in image recognition algorithms to identify a broader range of ingredients with higher accuracy.

Integration of advanced techniques, such as object detection and segmentation, for more precise identification and localization of ingredients within complex dishes.

Multi-Cuisine Support:

Expansion of the project to encompass a wider array of cuisines, including regional and international dishes.

Incorporation of cultural and regional variations in cooking styles and ingredients to provide a more diverse and inclusive recipe-generating experience.

User Personalization:

Development of personalized recommendation systems based on user preferences, dietary restrictions, and past cooking history.

Implementation of machine learning models to adapt and refine recipe suggestions over time, aligning with individual tastes and preferences.

Integration with Smart Kitchen Appliances:

Collaboration with smart kitchen appliance manufacturers to seamlessly integrate the recipe generation system with devices such as smart ovens, cooktops, and timers.

Real-time communication between the system and smart appliances to automate cooking processes based on generated recipes.

CHAPTER 8: REFERENCES

1. "Cooking with Computers: A Recipe for Computational Creativity" by Lav R. Varshney and Al Edelman.
2. "Learning Cross-modal Embeddings for Cooking Recipes and Food Images" by Ignacio Rocco, et al.
3. "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
4. TensorFlow documentation (https://www.tensorflow.org/)
5. The original paper: "Densely Connected Convolutional Networks" by Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. You can find it on arXiv: https://arxiv.org/abs/1608.06993
6. GitHub repo. : BiplabGorain/RecipeLens: Final Year Project (github.com)