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

December, 2023

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: 06/12/2023
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 Flask 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 recipe generation.

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

MobileNetV2 is a convolutional neural network architecture designed for mobile and edge devices with limited computational resources. It was developed by Google researchers to provide an efficient and lightweight model for tasks such as image classification, object detection, and other computer vision applications.

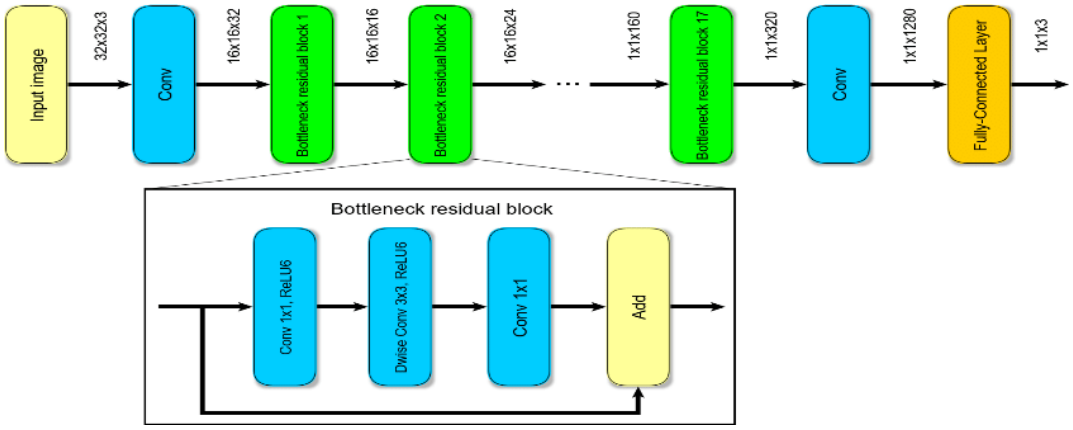

Fig 1.0: MobileNetV2 architecture

Here are the key features and components of the MobileNetV2 architecture:

Inverted Residuals with Linear Bottlenecks:

- MobileNetV2 uses a novel building block called an "inverted residual with linear bottlenecks." This design helps to capture and propagate information through the network more efficiently.
- The architecture employs a shortcut connection (similar to residual networks) but with an inverted structure. It first expands the input using a lightweight depthwise separable convolution, then applies a 3x3 depthwise separable convolution, and finally projects the result back to a low-dimensional space.

Linear Bottleneck:

- The linear bottleneck design involves using a lightweight 1x1 convolutional layer followed by a 3x3 depthwise separable convolution and another 1x1 convolutional layer. This helps to maintain a balance between model capacity and computational efficiency.

Inverted Residual Block:

- The basic building block of MobileNetV2 is the inverted residual block, which consists of an expansion layer, a depthwise separable convolution layer, and a projection layer.

Linear Bottleneck with Shortcut:

- The use of linear bottlenecks is combined with shortcut connections, allowing for easier flow of gradients during backpropagation and addressing the vanishing/exploding gradient problem.

Global Average Pooling (GAP):

- Instead of fully connected layers at the end of the network, MobileNetV2 uses global average pooling to reduce spatial dimensions and generate a fixed-size feature vector. This helps in reducing the number of parameters and computations, making the model more lightweight.

Efficient Depthwise Separable Convolutions:

- Depthwise separable convolutions are used extensively throughout the architecture, which involves a depthwise convolution (processing each channel separately) followed by a pointwise convolution (combining information across channels). This approach reduces the number of parameters and computations compared to traditional convolutions.

Width Multiplier and Resolution Multiplier:

- MobileNetV2 introduces two hyperparameters, the width multiplier and resolution multiplier, that allow users to customize the model size according to their computational constraints. The width multiplier scales the number of channels in each layer, and the resolution multiplier scales the input resolution.

```
# ------------------------------------------------------------
# Load Pre-trained Model
# ------------------------------------------------------------

pretrained_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'

)
```

```
pretrained_model.trainable = False
# ------------------------------------------------------------
# define layers
# ------------------------------------------------------------

inputs = pretrained_model.input
x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(20,activation='softmax')(x)

# ------------------------------------------------------------
# creating model
# ------------------------------------------------------------

model = tf.keras.Model(inputs, outputs)
model.summary()
```

CHAPTER 5: DESIGN

5.1 Architectural Design (Project Flow /architecture with description)

Project flow –

5.1.1 User Input:

- **Capture Image:** Users upload food images through a web interface or mobile application.
- **Optional User Preferences:** Users may provide additional information such as dietary preferences or restrictions.

5.1.2 Image Processing:

- **Preprocessing:** Normalize and resize the uploaded images for consistency.
- **Image Recognition:** Use a pre-trained deep learning model to identify food items in the image.

5.1.3 Ingredient Extraction:

- Quantification: Implement algorithms to estimate quantities of identified ingredients.
- Unit Standardization: Convert quantities into a standardized unit (e.g., grams or millilitres).

5.1.4 Recipe Generation:

- **NLP Model:** Utilize a natural language processing (NLP) model to generate human-readable recipes.
- **Steps and Instructions:** Construct cooking instructions, order of steps, and additional details like cooking time and temperature.

5.1.5 Database Interaction:

- **Data Storage:** Store information about images, ingredients, and recipes in a database.
- **User Data:** Save user preferences, saved recipes, and interactions in the database.

5.1.6 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.

5.1.7 Continuous Learning:

- **Model Improvement:** Periodically update and retrain the image recognition and NLP models with new data.
- **User Feedback Loop:** Use ongoing user interactions and feedback to enhance the system's performance.

5.1.8 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.

5.1.9 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 Visualization:

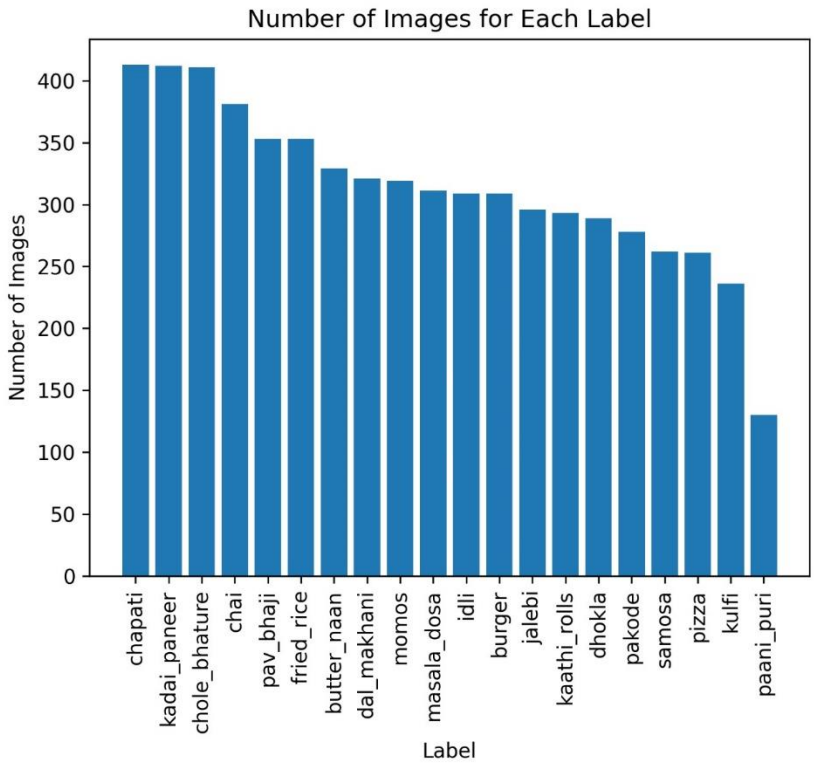

Fig 2.0: Distribution of images across the labels

This visual representation illustrates the distribution of images among various labels or categories within the dataset. Each bar in the chart corresponds to a specific label, and the height of the bar signifies the number of images associated with that particular label. The purpose of this image is to provide a clear overview of the dataset's composition, allowing viewers to discern the relative prevalence of different categories. The x-axis of the chart represents the labels or categories, while the y-axis quantifies the count of images.

Fig 3.0: Different types of foods

5.3 Model Accuracy:

After trained the model the accuracy of the model is 80%.

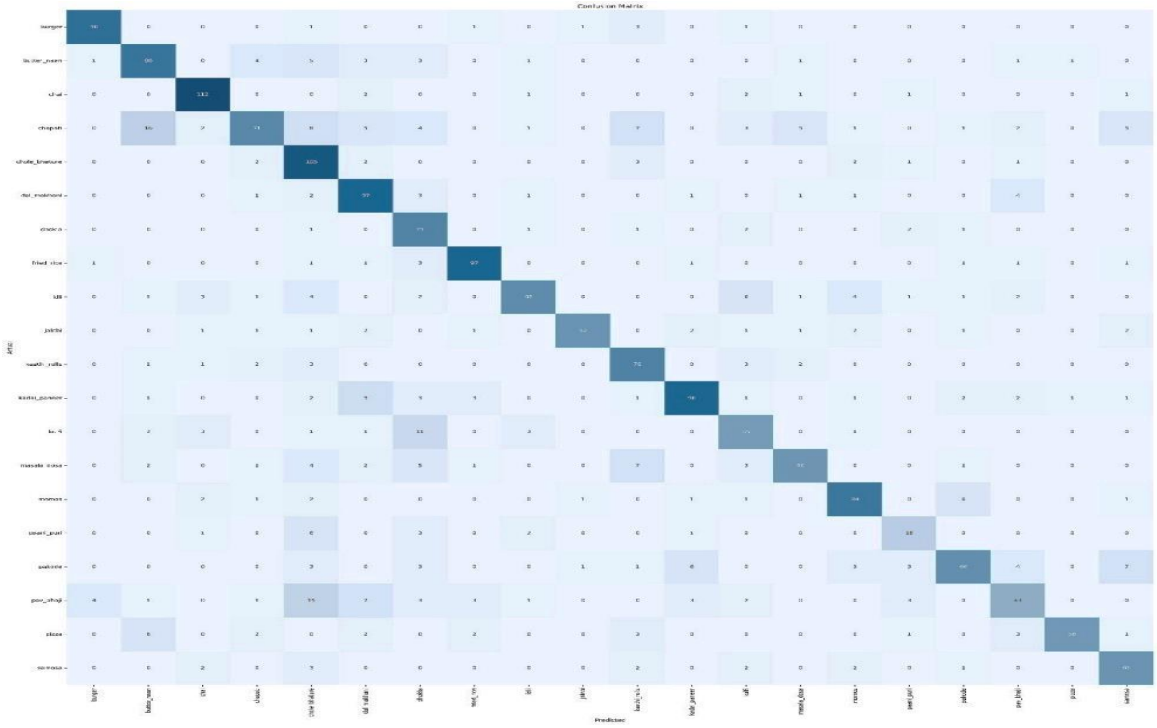

Fig 4.0: Confusion Matrix of the Model

A confusion matrix is a visual representation that provides insight into the performance of a classification model by summarizing the counts of true positive, true negative, false positive, and false negative predictions.

True Positive (TP): This cell indicates the number of images where the model correctly identified and generated recipes accurately.

True Negative (TN): In this cell, the count denotes the number of images where the model correctly identified the absence of a recipe.

False Positive (FP): This cell represents the number of images where the model incorrectly predicted the presence of a recipe when none was warranted.

False Negative (FN): The count in this cell indicates the number of images where the model failed to recognize and generate a recipe when it should have.

CHAPTER 6: IMPLEMENTATION

6.1 Implementation Details

5.1.1 Creating Data Frame

```
image_dir = Path('../../data/raw/FoodClassification/')

filepaths = list(image_dir.glob(r'**/*.jpg'))
labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))
filepaths = pd.Series(filepaths, name='Filepaths').astype(str)
labels = pd.Series(labels, name='Label')
images = pd.concat([filepaths, labels], axis=1)
category_samples = []

for category in images['Label'].unique():
    category_slices = images.query('Label==@category')
    category_samples.append(category_slices.sample(frac=1, random_state=1))
image_df =
pd.concat(category_samples,axis=0).sample(frac=1.0,random_state=1).reset_index
(drop=True)

image_df.to_csv('../../data/interim/data.csv')
```

1. **Set Image Directory:**
    - **image_dir = Path('../../data/raw/FoodClassification/')**: Defines the path to the directory containing food images.
2. **Gather Filepaths and Labels:**
    - **filepaths = list(image_dir.glob(r'**/*.jpg'))**: Uses **glob** to create a list of file paths for all **.jpg** files in the specified directory and its subdirectories.
    - **labels = list(map(lambda x: os.path.split(os.path.split(x)[0])[1], filepaths))**: Extracts category labels by splitting file paths and capturing the parent directory's name.
3. **Create DataFrame:**
    - **filepaths = pd.Series(filepaths, name='Filepaths').astype(str)**: Converts the file paths to a Pandas Series and assigns the name 'Filepaths'.
    - **labels = pd.Series(labels, name='Label')**: Creates another Series for labels with the name 'Label'.
    - **images = pd.concat([filepaths, labels], axis=1)**: Combines the file paths and labels into a DataFrame named 'images'.
4. **Shuffle the DataFrame by Category:**

- **category_samples = []**: Initializes an empty list to store shuffled samples for each category.
- **for category in images['Label'].unique():**: Iterates through unique category labels.
    - **category_slices = images.query('Label==@category')**: Filters the DataFrame for the current category.
    - **category_samples.append(category_slices.sample(frac=1, random_state=1))**: Shuffles the rows within the category and appends the result to **category_samples**.
- **image_df = pd.concat(category_samples, axis=0).sample(frac=1.0, random_state=1).reset_index(drop=True)**: Concatenates the shuffled category samples and then shuffles the entire DataFrame to achieve overall randomness.

5. **Export DataFrame to CSV:**
    - **image_df.to_csv('../../data/interim/data.csv')**: Saves the final shuffled DataFrame to a CSV file named 'data.csv' in the specified directory ('../../data/interim/').

6.1.2 Visualizing Data

```
plt.bar(label_counts.index, label_counts.values)
plt.xlabel('Label')
plt.ylabel('Number of Images')
plt.title('Number of Images for Each Label')
plt.xticks(rotation='vertical')
plt.rcParams['savefig.dpi']=300
plt.savefig(
    f"../../reports/figures/Number of Images for Each Label.png",
bbox_inches='tight')
plt.show()
```

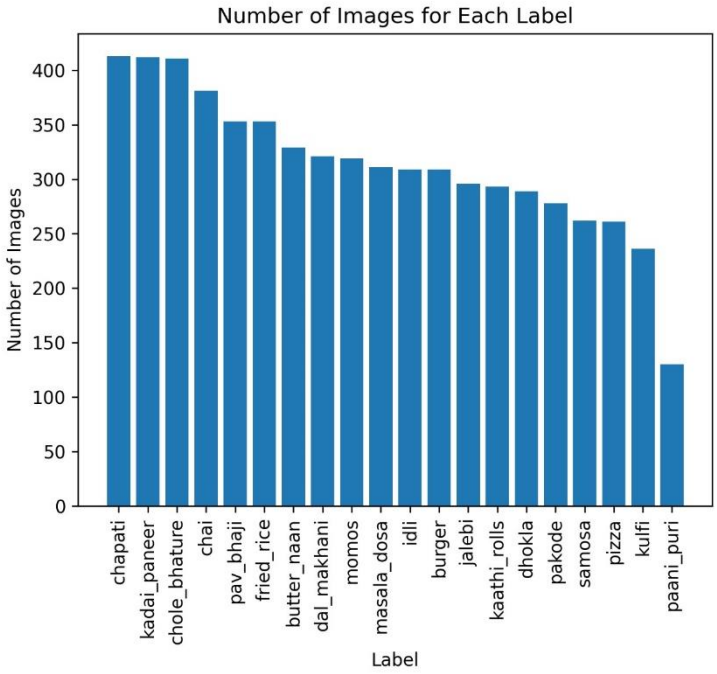

Fig 5.0: Distribution of images across the labels

6.1.3 Prediction

```
predictions = np.argmax(model.predict(test_images), axis=1)

cm = confusion_matrix(test_images.labels, predictions)
clr = classification_report(test_images.labels, predictions,
                            target_names=test_images.class_indices,
zero_division=0)
```

1. **Make Predictions:**
    - Utilizes a trained model to predict class probabilities for a set of test images.
    - Derives predicted class labels by selecting the index with the highest probability along the appropriate axis.
2. **Confusion Matrix:**
    - Computes the confusion matrix, a tabular representation summarizing the model's classification performance.
    - Rows represent true classes, and columns represent predicted classes.
3. **Classification Report:**
    - Generates a comprehensive classification report with metrics such as precision, recall, F1-score, and support for each class.
    - Provides insights into the model's accuracy and performance on individual classes.
    - Handles scenarios where a class has zero true samples, preventing division by zero errors.

6.1.4 Matrix Prediction

```
plt.figure(figsize=(30, 30))
sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)
plt.xticks(ticks=np.arange(20) + 0.5,
           labels=test_images.class_indices, rotation=90)
plt.yticks(ticks=np.arange(20) + 0.5,
           labels=test_images.class_indices, rotation=0)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.savefig(
    f"../../reports/figures/Confusion matrix.png", bbox_inches='tight')
plt.show()
```

1. **Figure Size:**
    - **plt.figure(figsize=(30, 30))**: Specifies the size of the entire figure (heatmap) as 30x30 inches, ensuring a large and detailed visualization.
2. **Heatmap Creation:**
    - **sns.heatmap(cm, annot=True, fmt='g', vmin=0, cmap='Blues', cbar=False)**: Creates a heatmap using Seaborn's **heatmap** function.
        - **cm**: Confusion matrix data.
        - **annot=True**: Displays the actual numeric values in each cell.
        - **fmt='g'**: Specifies the format of the numeric values as 'general'.
        - **vmin=0**: Sets the minimum color value to 0.
        - **cmap='Blues'**: Defines the color map as a blue gradient.
        - **cbar=False**: Disables the color bar on the side.
3. **X and Y Axis Ticks:**
    - **plt.xticks(ticks=np.arange(20) + 0.5, labels=test_images.class_indices, rotation=90)**: Sets the x-axis ticks at positions corresponding to the center of each cell, with labels representing class indices, and rotates them for better readability.
    - **plt.yticks(ticks=np.arange(20) + 0.5, labels=test_images.class_indices, rotation=0)**: Sets the y-axis ticks similarly.
4. **Axis Labels and Title:**
    - **plt.xlabel("Predicted")**: Adds a label to the x-axis indicating predicted classes.
    - **plt.ylabel("Actual")**: Adds a label to the y-axis indicating actual (true) classes.
    - **plt.title("Confusion Matrix")**: Sets a title for the entire plot.
5. **Save the Figure:**
    - **plt.savefig(f"../../reports/figures/Confusion matrix.png", bbox_inches='tight')**: Saves the generated heatmap as a PNG image file in the specified directory. The **bbox_inches='tight'** argument ensures the entire plot is saved without cropping issues.
6. **Display the Plot:**
    - **plt.show()**: Displays the heatmap.

6.1.5 Train Model

```
train_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepaths',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='training'
)
```

```
val_images = train_generator.flow_from_dataframe(
    dataframe=train_df,
    x_col='Filepaths',
    y_col='Label',
    target_size=(224, 224),
    color_mode='rgb',
    class_mode='categorical',
    batch_size=32,
    shuffle=True,
    seed=42,
    subset='validation'
)

pretrained_model = tf.keras.applications.MobileNetV2(
    input_shape=(224, 224, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'

)
pretrained_model.trainable = False

inputs = pretrained_model.input
x = tf.keras.layers.Dense(128, activation='relu')(pretrained_model.output)
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(20,activation='softmax')(x)
```

1. **Data Generators for Training and Validation:**
    - Utilizes ImageDataGenerator from TensorFlow's Keras API to generate batches of augmented image data for model training and validation.
    - flow_from_dataframe method is used, where:
        - dataframe: The DataFrame containing image file paths ('Filepaths') and corresponding labels ('Label').
        - x_col: Specifies the column in the DataFrame representing the image file paths.
        - y_col: Specifies the column in the DataFrame representing the labels.
        - target_size: Sets the size of the images to (224, 224).
        - color_mode: Sets the color mode to 'rgb'.
        - class_mode: Specifies the type of label encoding as 'categorical'.
        - batch_size: Sets the batch size to 32 images per batch.
        - shuffle: Shuffles the data during training.
        - seed: Provides a seed for reproducibility.

- subset: Specifies whether the generator is for training ('training') or validation ('validation').

2. **Pretrained MobileNetV2 Model:**
    - Utilizes the MobileNetV2 architecture as a pretrained model from TensorFlow's Keras applications.
    - input_shape=(224, 224, 3): Specifies the input shape for the images.
    - include_top=False: Excludes the top (final) layer of the pretrained model.
    - weights='imagenet': Initializes the model with weights pretrained on the ImageNet dataset.
    - pooling='avg': Specifies global average pooling as the final pooling layer.
3. **Model Architecture Modification:**
    - Freezes the weights of the pretrained layers to retain the learned features during training on the specific dataset.
    - Adds custom dense layers on top of the pretrained MobileNetV2 layers:
        - Two dense layers with 128 neurons and ReLU activation functions.
        - A final dense layer with 20 neurons (for 20 classes) and a softmax activation function.

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. https://www.kaggle.com/datasets/l33tc0d3r/indian-food-classification/data
2. "Cooking with Computers: A Recipe for Computational Creativity" by Lav R. Varshney and AI Edelman.
3. "Learning Cross-modal Embeddings for Cooking Recipes and Food Images" by Ignacio Rocco, et al.
4. "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
5. TensorFlow documentation (https://www.tensorflow.org/)