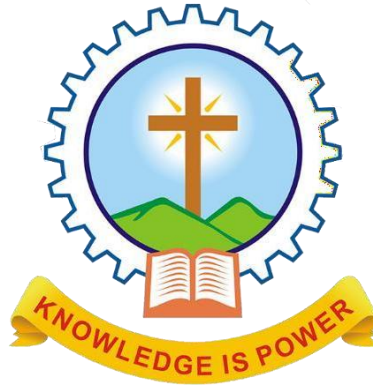


MAR ATHANASIUS COLLEGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University, TVM)

KOTHAMANGALAM



Department of Computer Applications

Mini Project Report

FRUIT CLASSIFICATION AND CALORIE COUNTER

Done by

Sandra S Santhosh

Reg No: MAC23MCA-2048

Under the guidance of

Prof. SONIA ABRAHAM

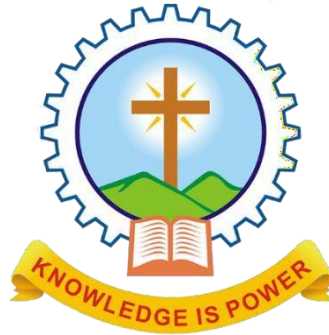
2023-2025

MAR ATHANASIUS COLLEGE OF ENGINEERING

(Affiliated to APJ Abdul Kalam Technological University, TVM)

KOTHAMANGALAM

CERTIFICATE



FRUIT CLASSIFICATION AND CALORIE COUNTER

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

SANDRA S SANTHOSH

Reg No: MAC23MCA-2048

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

Master of Computer Applications

of

APJ Abdul Kalam Technological University Thiruvananthapuram

Faculty Guide

Prof. Sonia Abraham

Head of the Department

Prof. Biju Skaria

Project Coordinator

Prof. Sonia Abraham

Internal Examiner

ACKNOWLEDGENT

I am profoundly grateful for the blessings and guidance bestowed upon me by the Almighty throughout this journey. His divine presence has been the driving force behind my accomplishments.

I extend my heartfelt appreciation to Prof. Biju Skaria, who serves as the Head of the Computer Applications Department and our esteemed project coordinator, for his exceptional leadership, unwavering support, and invaluable mentorship. His profound insights and encouragement have been instrumental in shaping my academic and professional endeavors.

My deepest gratitude goes to my project guide, Prof. Sonia Abraham for his invaluable guidance, expert advice, and relentless dedication to excellence. His unwavering commitment and encouragement have been a constant source of inspiration.

I extend my sincere thanks to all the faculty members and staff of MACE for their unwavering support, encouragement, and guidance. Their collective efforts have created an environment conducive to learning and growth.

I am indebted to my beloved parents for their unconditional love, unwavering support, and endless sacrifices. Their guidance and encouragement have been my pillars of strength throughout this journey. I express my heartfelt thanks to my friends and peers for their camaraderie, support, and encouragement. Their companionship has made this journey both enriching and enjoyable.

Lastly, I extend my gratitude to all those who have contributed to my personal and professional growth. Your encouragement, expertise, and support have been deeply appreciated and will always be cherished.

ABSTRACT

Fruit classification and calorie estimation are critical for personalized nutrition, quality inspection, and automated sorting in the food industry. This project focuses on developing a deep learning model using a ResNet-50 Convolutional Neural Network (CNN) for fruit classification and calorie estimation. The model leverages a curated dataset of fruit images, enriched with nutritional data through web scraping techniques, to provide accurate calorie content estimations.

The ResNet-50 architecture, pre-trained on ImageNet, is fine-tuned on the Fruit-360 dataset, which consists of 94,110 images across 98 fruit classes. The dataset is augmented with nutritional information through web scraping tools such as Scrapy and Selenium, enhancing the classifier's ability to estimate calorie content accurately. The model's architecture incorporates custom layers, including global average pooling, dropout, and a dense prediction layer, to optimize performance.

The model is trained for 12 epochs using the Adam optimizer with a learning rate of 0.0001, achieving 100% accuracy on both training and validation sets. Despite its high performance, challenges were encountered in distinguishing certain fruit classes, such as papaya, indicating the need for further dataset augmentation or model adjustments to improve generalization.

Overall, the findings of this project underscore the effectiveness of deep learning models in fruit classification and calorie estimation. The use of a ResNet-50 architecture highlights its robustness in image recognition tasks, providing accurate and reliable results. The insights gained from this study can inform future research directions and contribute to the development of more refined and scalable solutions for real-world applications in nutrition, food industry automation, and health management.

LIST OF TABLES

2.1 Literature Summary Table.....	7
3.1 Model Summary of ResNet50.....	19
3.2 Dimensionality table of ResNet50.....	21

LIST OF FIGURES

3.1	Dataset Folders.....	10
3.2	Different classes of fruits in the dataset.....	11
3.3	Train images of the dataset.....	11
3.4	Test images of the dataset.....	11
3.5	Number of images in each class of Training and Testing.....	12
3.6	Batch size and image dimensions.....	13
3.7	Preparing images for the ResNet model.....	13
3.8	Dataset images.....	14
3.9	Code for Visualization of Train dataset.....	14
3.10	Code for Visualization of Test dataset.....	15
3.11	Code for Visualization of Test dataset.....	15
3.12	Train and Test visualization.....	18
3.13	ResNet50 Network Architecture.....	18
3.14	Project Pipeline.....	22
4.1	Loading ResNet50 pretrained model.....	32
4.2	Model compiling.....	35
4.3	Training ResNet50 model.....	36
4.4	ResNet50 12 epochs.....	37
4.5	Testing with Test dataset.....	38
4.6	Testing with Validation dataset.....	38
4.7	Prediction 1.....	38
4.8	Testing with unseen data.....	39
4.9	Prediction 2.....	39
5.1	Accuracy and Loss curve of ResNet50.....	40
5.2	Confusion Metrix.....	40
5.3	Classification Report.....	41

6.1	Home page.....	43
6.2	About page.....	44
6.3	Prediction Page 1.....	44
6.4	Prediction page 2.....	45
6.5	Prediction and calorie displayed.....	45
7.1	Git history of 3 sprint.....	46
7.2	Sprint 3.....	46

CONTENTS

1	Introduction	1
2	Supporting Literature	2
2.1	Literature Review.....	2
2.1.1	Summary Table.....	5
2.2	Findings and Proposals.....	6
3	System Analysis	7
3.1	Analysis of Dataset.....	7
3.1.1	About the Dataset.....	7
3.1.2	Explore the dataset.....	9
3.2	Data Pre-processing.....	10
3.2.1	Data preprocessessing for ResNet50.....	10
3.3	Data Visualization.....	15
3.4	Analysis of Algorithm.....	18
3.4.1	ResNet50.....	21
3.5	Project Plan.....	22
3.5.1	Project Pipeline.....	22
3.5.2	Project Implementation Plan.....	23
3.6	Feasibility Analysis.....	24
3.6.1	Technical Feasibility.....	24
3.6.2	Economic Feasibility.....	25
3.6.3	Operational Feasibility.....	26
3.7	System Environment.....	27
3.7.1	Software Environment.....	27
3.7.2	Hardware Environment.....	29
4	System Design	30
4.1	Model Building.....	30

4.1.1	Model Planning	30
4.1.2	Model Training	34
4.1.3	Model Testing	34
5	Results and Discussion	35
6	Model Deployment	37
6. 1	UI Design.....	40
7	Git History	43
8	Conclusion	44
9	Future Work	45
10	Appendix	47
10.1	Minimum Software Requirements	47
10.2	Minimum Hardware Requirements	49
11	References	50

1. INTRODUCTION

The automatic classification of fruits is an essential task with diverse applications in the food industry, agriculture, and health management. Accurate fruit classification plays a pivotal role in areas such as sorting, quality inspection, personalized nutrition, and pricing strategies. However, this task presents significant challenges due to similarities in shape and color among different fruits and considerable intra-class variations. Existing models often struggle to achieve high accuracy and practical relevance when applied to real-world scenarios.

This project, titled "Fruit Classification and Calorie Counter," addresses these challenges by focusing on a deep learning-based system that classifies fruits and estimates their calorie content. We employ the ResNet-50 Convolutional Neural Network (CNN) architecture, which is highly effective in image recognition tasks. The model is trained on a subset of the Fruit-360 dataset, sourced from Kaggle, which contains 94,110 images in 141 classes (fruits, vegetables, and nuts). For this project, we focus specifically on 8 fruit classes: Apple, Banana, Blueberry, Watermelon, Strawberry, Papaya, Pineapple, and Orange. The training set for this project consists of 3,853 images, while the testing set includes 1,288 images. Each image is scaled to 100x100 pixels. Additionally, nutritional data for calorie estimation is obtained through web scraping, adding another layer of functionality to the system by providing users with accurate calorie information based on the classified fruits.

Accurate fruit classification and calorie estimation are vital for the agriculture and food processing sectors, as well as for consumer health management. Automated tools that can perform these tasks efficiently help meet industry demands for quality control, sorting, and economic efficiency. Moreover, by providing precise calorie data, such systems can assist users in managing their dietary intake, which is crucial for weight management, nutrition planning, and overall health monitoring.

This project focuses solely on ResNet-50, known for its balance of depth and computational efficiency. By narrowing the scope to 8 fruit classes, this study aims to demonstrate the model's effectiveness in fruit classification and its potential applications in both industry and consumer health solutions. The findings from this project are expected to contribute to advancements in food industry automation and personalized nutrition tools.

2. SUPPORTING LITERATURE

2.1 Literature Review

Paper1: Mehenag Khatun, Forhad Ali, Nakib Aman Turzo, and Julker Nin, "Fruits Classification using Convolutional Neural Network," GRD Journals - Global Research and Development Journal for Engineering, 2020.

The paper "Fruits Classification using Convolutional Neural Network"* explores the automation of fruit classification using advanced computer vision techniques, addressing the inefficiencies of traditional manual methods. By leveraging image processing techniques, the authors propose a more efficient and accurate approach to tasks like quality inspection and sorting based on attributes such as color, size, and shape. Using the Fruits 360 dataset from Kaggle, which consists of 1400 images across seven fruit classes (apple, banana, orange, pear, mango, pineapple, and strawberry), the images were resized to 100x100 pixels to ensure consistency. The dataset was split into 1260 training images and 140 testing images, each class containing around 200 images.

The methodology involves pre-processing images to remove noise and adjust contrast, followed by segmentation and feature extraction, focusing on key visual aspects such as color, size, and texture. The classification was performed using CNN architectures like MobileNet and Inception v3, with MobileNet proving particularly suitable for real-time applications due to its lightweight design. The results showed a remarkable 98.74% accuracy using the MobileNet architecture, demonstrating the effectiveness of CNNs in fruit classification tasks.

In conclusion, the paper presents a practical solution for automating fruit classification, highlighting its potential applications in agriculture and retail. The study's future directions include expanding the dataset to cover more fruits and vegetables, testing additional CNN models for comparative accuracy, and integrating features to detect fruit diseases. This research lays the groundwork for more advanced, automated systems that could enhance operational efficiency in various sectors.

Paper 2: Hanshu Tomar, "Multi-Class Image Classification of Fruits and Vegetables Using Transfer Learning Techniques," Master's dissertation, Journal of Physics: Conference Series, 2021.

The paper "Multi-Class Image Classification of Fruits and Vegetables Using Transfer Learning Techniques" by Hanshu Tomar explores the challenges and advancements in classifying fruits and vegetables using deep learning. It highlights the complexity of distinguishing different fruits and vegetables through convolutional neural networks (CNNs), aiming to improve the accuracy and real-world applicability of these models. The research uses a diverse dataset of fruit and vegetable images to train and evaluate several deep learning models, ensuring robustness and inclusivity in classification tasks.

The study compares several CNN architectures, including ResNet-50, InceptionV3, and VGG16. These models are evaluated for their performance in classifying the dataset, with a focus on transfer learning—where pre-trained models are re-trained for specific tasks, improving performance without requiring large amounts of data. ResNet-50 and InceptionV3 performed the best, each achieving an accuracy of 96.5%, demonstrating their effectiveness in fruit and vegetable classification.

In conclusion, the paper provides a comprehensive review of CNN architectures in fruit and vegetable classification, emphasizing the role of transfer learning in enhancing model performance. The comparative analysis of these architectures provides valuable insights into their strengths and weaknesses, offering practical solutions for deploying accurate image recognition systems in real-world applications. ResNet-50 and InceptionV3, in particular, show strong potential for improving classification accuracy in automated systems.

Paper 3 :Yonis Gulzar, "Fruit Image Classification Model Based on MobileNetV2 with Deep Transfer Learning Technique," Sustainability, 2023.

The paper "Fruit Image Classification Model Based on MobileNetV2 with Deep Transfer Learning Technique" by Yonis Gulzar investigates the application of deep learning for fruit classification, a vital component of precision agriculture. The research aims to enhance the accuracy of fruit identification using advanced machine learning models, which are crucial for quality analysis, yield estimation, and disease prediction. This study emphasizes the growing significance of deep learning in agriculture, striving to contribute to more effective fruit classification methodologies.

The dataset utilized in this study consists of images of 40 different fruit types, which is essential for training and evaluating the model. The diversity in the dataset enables the model to learn distinct features and patterns associated with each fruit, which is critical for achieving high classification accuracy. The methodology employs a deep learning model based on the MobileNetV2 architecture, enhanced through transfer learning. By removing the original classification layer and adding five new layers, the researchers optimized the TL-MobileNetV2 model through various preprocessing steps and tuning techniques to effectively manage the dataset while minimizing overfitting.

The results of the study are impressive, with the TL-MobileNetV2 model achieving a remarkable 99% accuracy in fruit classification. This high level of accuracy underscores the model's effectiveness and reliability. The paper concludes with plans for future work, including the development of a mobile-based application for broader fruit classification and comparative analysis with other CNN models to identify the most efficient approach. This future research aims to enhance the model's applicability and usability in real-world scenarios, making it a valuable tool for users with varying expertise levels.

Paper4:Chaitanya A, Jayashree Shetty, and Priyamvada Chiplunkar, "Food Image Classification and Data Extraction Using Convolutional Neural Network and Web Crawlers," Procedia Computer Science, Volume 218, 2023, Pages 143-152.

The paper "Food Image Classification and Data Extraction Using Convolutional Neural Network and Web Crawlers" investigates the application of transfer learning to enhance food image classification accuracy. Utilizing the extensive Food-101 dataset, which comprises 101,000 images across 101 food categories, the research fine-tunes pre-trained CNN models, InceptionV3 and ResNet50. These models achieved impressive accuracy rates of 94.85% and 95.16%, respectively, underscoring their effectiveness in identifying various food items.

A notable aspect of this study is the incorporation of web scraping techniques to enrich the dataset with additional metadata and nutritional information. Using tools like Scrapy and Selenium, the researchers gathered detailed information about each food item, including ingredients, calories, and serving suggestions. This enriched dataset not only aids in achieving more accurate classifications but also provides valuable contextual information for dietary monitoring and other applications.

In conclusion, while the study emphasizes the advantages of using transfer learning with advanced CNN architectures for food image classification, the innovative use of web scraping represents a significant contribution. This technique enhances the dataset and paves the way for creating more informative models. Future research could explore further applications of these methods, potentially incorporating real-time data acquisition to maintain the dataset's relevance. Overall, this paper offers valuable insights into leveraging advanced machine learning techniques and innovative data collection methods for food image classification, presenting promising directions for future research and practical applications.

2.1. Literature Summary Table

PAPER	TITLE	YEAR	PUBLISHER	DATASET	ARCHITECTURE	ACCURACY
1	"Fruits Classification using Convolutional Neural Network" by Mehenag Khatun, Forhad Ali, Nakib Aman Turzo, and Julker Nine	2020	IEEE	Fruits 360	MobileNet Inception v3	98.74% above 78.1%
2.	"Multi-Class Image Classification of Fruits and Vegetables Using Transfer Learning Techniques" By Hanshu Tomar .	2021	IEEE	Fruits 360	VGG16 ResNet50	99% 99.35%

3.	"Fruit Image Classification Model Based on MobileNetV2 with Deep Transfer Learning Technique " by Yonis Gulzar .	2023	IEEE	Fruits 360	TL-MobileNetV2	99%
4.	"Food Image Classification and Data Extraction Using Convolutional Neural Network and Web Crawlers" by Chaitanya A, Jayashree Shetty, and Priyamvada Chiplunkar.	2021	IEEE	Food-101	InceptionV3 ResNet50	94.85%. 95.16%.

Table 2.1 Literature Summary Table

2.2. Findings and Proposal

I have referred to four papers that utilize CNN architectures for fruit classification and nutritional information extraction. Accurate fruit classification plays a significant role in various applications, from dietary management to grocery store automation. These four research papers highlight advancements in using CNN models like MobileNet, InceptionV3, and ResNet50, and explore the benefits of transfer learning and web scraping for dataset enhancement. Each of these papers presents different methodologies, performance evaluations, and strengths.

The papers demonstrate the utility of CNN architectures in achieving high accuracy for image classification tasks, with MobileNet offering a lightweight solution suitable for mobile applications, while InceptionV3 excels in handling complex image features. ResNet50, the most robust of the models, has been particularly effective due to its deep architecture and the use of residual connections, which mitigate the vanishing gradient problem. Moreover, the integration of web scraping to collect additional nutritional information from online sources has proven to enhance the functionality of fruit classification systems, making them more useful in real-world applications.

Paper 1 focuses on MobileNet, highlighting its efficiency in mobile-based fruit classification with minimal computational requirements. Paper 2 uses InceptionV3, showing high accuracy in classifying a wide range of fruits but requiring substantial computational power. Paper 3 leverages ResNet50, demonstrating superior performance in classification accuracy due to its deeper architecture and ability to learn more complex features. Paper 4 introduces a combination of CNNs with web scraping techniques to enrich datasets with nutritional information, emphasizing the practical utility of linking classification tasks to real-world data.

The findings from these papers have informed the direction of my project, guiding the choice of ResNet50 for its proven performance in fruit classification tasks. My project proposal aims to build an efficient fruit classification system utilizing the ResNet50 architecture. The model will be trained on a subset of the Fruit-360 dataset, which includes eight fruit classes: Apple, Banana, Blueberry, Watermelon, Strawberry, Papaya, Pineapple, and Orange. ResNet50 was selected due to its deep learning capabilities, which enable it to handle large-scale image recognition tasks effectively.

In addition to fruit classification, the project integrates web scraping techniques to gather calorie information for each identified fruit, linking nutritional data with the classified images. This dual-purpose system not only identifies fruits but also provides users with valuable dietary information, making it highly useful in settings such as grocery stores, restaurants, and home kitchens.

3. SYSTEM ANALYSIS

3.1. Analysis of the dataset

The analysis of the dataset, comprising a myriad of images, involved sophisticated image processing techniques and deep learning algorithms, unveiling intricate patterns, features, and relationships embedded within the visual data, thus providing a comprehensive understanding of the underlying structure and content.

3.1.1. About the dataset

The Fruit-360 dataset, sourced from Kaggle, offers a comprehensive and well-structured collection of fruit images, specifically designed for the task of fruit classification using computer vision techniques. This dataset is a valuable asset for researchers and developers aiming to create robust models capable of accurately identifying and classifying different types of fruits.

The dataset comprises high-quality images of various fruits, each resized to 100x100 pixels to ensure uniformity and ease of processing during model training. The images are divided into training and testing sets, with the training set consisting of 70,491 images and the test set containing 23,619 images. However, for this project, a subset of the Fruit-360 dataset is used, focusing on eight fruit classes: Apple, Banana, Blueberry, Watermelon, Strawberry, Papaya, Pineapple, and Orange. The training subset includes 3,853 images, while the test set has 1,288 images.

Each image in the dataset captures fruits in different orientations and lighting conditions, which helps simulate real-world scenarios. The diversity in image perspectives ensures that models trained on this dataset are better equipped to handle variations that occur during actual usage, such as recognizing fruits in different settings or environments.

This dataset provides an excellent foundation for training and evaluating convolutional neural networks (CNNs) such as ResNet50. Its structured format, clear labeling, and high image quality make it a widely-used benchmark for fruit classification tasks. Researchers and developers in the fields of computer vision and machine learning can leverage this dataset to build models capable of high-accuracy classification and identification of fruits.

Additionally, the Kaggle platform fosters collaboration by making the dataset readily accessible to the global research community. This facilitates the exchange of ideas, methodologies, and innovations that can contribute to advancements in fruit classification systems. By utilizing this dataset, developers can explore new techniques in image recognition and push the boundaries of what's possible in automatic food identification and calorie counting.

Dataset:<https://www.kaggle.com/datasets/moltean/fruits>

3.1.2 Explore the dataset

Before diving into model development, it is essential to thoroughly explore the Fruit-360 dataset, which has been sourced from Kaggle and modified to suit the requirements of this project. This exploration includes examining the distribution of images across the eight different fruit classes and identifying potential challenges, such as class imbalances. Furthermore, it is crucial to assess the overall quality and diversity of the images. Exploratory Data Analysis (EDA) techniques, such as data visualization and statistical summaries, can provide deeper insights into the characteristics and patterns within the dataset. Exploring sample images and annotations will aid in understanding the data, which is essential for making informed decisions in the model development process.

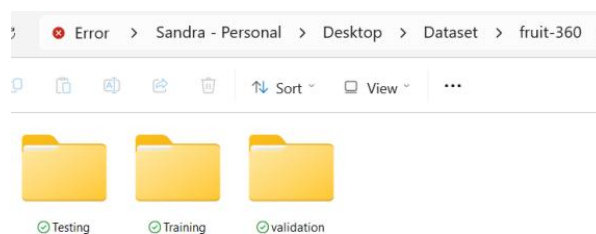


Fig 3.1 Dataset folders

The Fruit-360 dataset, which originally consisted of a vast array of fruit images, has been customized to include 5141 images spread across 8 classes: Apple, Banana, Blueberry, Watermelon, Strawberry, Papaya, Pineapple, and Orange. The dataset was methodically partitioned into three main subsets: training set, test set, and validation set.

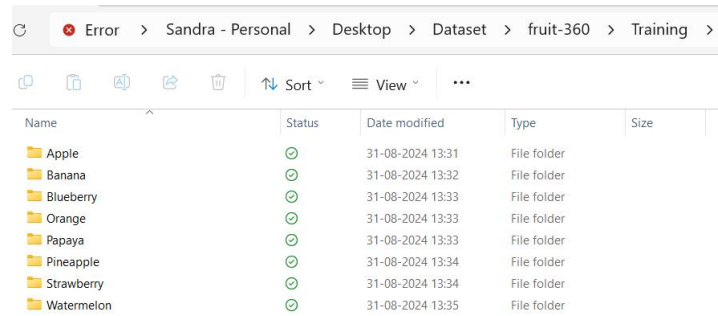


Fig 3.2 Different classes of fruits in the dataset.

The training set consists of 3853 images labeled into the eight different classes. These images form the backbone of the machine learning model, providing sufficient data to enable the model to learn from various fruit shapes, colors, and textures. To further improve model performance and avoid overfitting, 20% of the training data will be allocated as a validation set during the model testing phase. This additional validation set will serve as an internal check to fine-tune the hyperparameters of the model.

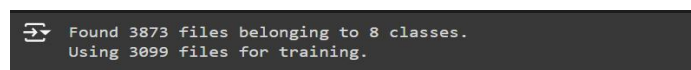


Fig 3.3 Train images of the dataset

The test set contains 1288 images distributed among the eight classes. These unseen images serve as an independent benchmark to evaluate how well the trained model performs on completely new data. By gauging the model's effectiveness on this test set, developers can assess its real-world application and predictive capabilities.

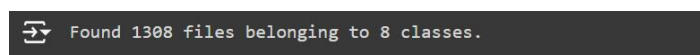


Fig 3.4 Test images of the dataset

A validation set consisting of 16 images was generated by extracting two images from each class before the training process. This subset enables the fine-tuning of model

parameters and offers an additional layer of validation during the early stages of training. By using this small validation set, developers can monitor the performance of the model and make adjustments to ensure that it generalizes effectively across various fruit classes.

```
↗ Training set class counts:  
{'Training': 0, 'Strawberry': 490, 'Papaya': 490, 'Blueberry': 460, 'Banana': 488, 'Watermelon': 473, 'Pineapple': 488, 'Orange': 477, 'Apple': 469}  
  
Testing set class counts:  
{'Testing': 0, 'Watermelon': 157, 'Strawberry': 164, 'Orange': 160, 'Blueberry': 154, 'Banana': 166, 'Apple': 146, 'Papaya': 164, 'Pineapple': 166}
```

Fig 3.5 Number of images in each class of Training and Testing

The dataset's labels contain categorical annotations corresponding to the eight classes of fruits. Each image is labeled with the fruit it represents, allowing the model to learn and predict the correct fruit class during training. This structured labeling system ensures consistency and accuracy during the training and evaluation processes. By organizing the dataset into these three distinct subsets, the project ensures a structured and thorough evaluation process.

3.2 Data Preprocessing

3.2.1 Data Preprocessing for ResNet50

In the Kaggle notebook, the Fruit-360 dataset underwent minimal preprocessing and augmentation to prepare it for utilization within the ResNet50 model. Since the dataset consists of 5141 images across 8 classes, with extensive prior augmentation, only essential transformations were applied to ensure compatibility with the ResNet50 architecture. First, caching and shuffling operations were performed, enabling efficient loading and randomization of the dataset during training. Both the training and validation sets were processed using TensorFlow's AUTOTUNE, optimizing pipeline performance.

Following this, a series of minor augmentations were applied to introduce variability and improve the model's generalization capabilities.

- **Random Horizontal Flip:** Horizontal flipping was applied randomly to training images, introducing orientation variability that mimics real-world conditions. This technique helps the model learn to classify fruits from different angles.
- **Random Rotation:** Each image was subjected to a random rotation of up to 20 degrees. This subtle rotation augments the data further by simulating minor changes in image orientation, which can be common in real-world image capture scenarios.

Through minimal preprocessing and targeted augmentation, the Fruit-360 dataset was tailored to suit the requirements of ResNet50-based models, ensuring optimal performance in classifying different fruit types. By resizing the images to the required dimensions, applying slight augmentations to introduce variability, and simulating minor real-world conditions, the dataset was optimized for robust model training and evaluation. This careful approach enhances the model's ability to generalize effectively across the eight fruit classes, ultimately supporting the development of a reliable fruit classification system.

- First, we determine the `batch_size` and dimension of the image.

```
[ ] batch_size = 32  
    img_height = 100  
    img_width = 100
```

Fig 3.6 Batch size and image dimensions

Before feeding images into the ResNet50 model, a consistent batch size and image dimension were determined. Each image was resized to match the ResNet50 input dimensions, providing uniformity across inputs.

- ResNet models expect input images to be preprocessed in a specific way to ensure that the model performs optimally.

```
[ ] preprocess_input = tf.keras.applications.resnet.preprocess_input
```

Fig 3.7 Preparing images for the ResNet model

Using `tf.keras.applications.resnet.preprocess_input`, images were normalized and pixel values scaled as required for the ResNet50 model. This critical preprocessing function ensures input compatibility by adjusting pixel values to match ResNet50's expected input range, ultimately boosting performance and stability.

3.3 Data Visualization

Data visualization is the process of presenting data in graphical or visual format to convey information, patterns, and insights more effectively and intuitively.



Fig 3.8 Dataset images

```
import matplotlib.pyplot as plt
class_counts = {'Papaya 1': 512, 'Watermelon 1': 475, 'Apple 6': 473, 'Orange 1': 479, 'Banana 1': 490, 'Blueberry 1': 462, 'Pineapple 1': 490, 'Strawberry 1': 475}
class_names = list(class_counts.keys())
image_counts = list(class_counts.values())
plt.figure(figsize=(4, 4))
plt.bar(class_names, image_counts)
plt.xlabel("Class Name")
plt.ylabel("Image Count")
plt.title("Class Frequency Distribution in Training")
plt.xticks(rotation=45, ha="right") # Rotate x-axis labels for better readability
plt.tight_layout()
plt.show()
```

Fig 3.9 Code for Visualization of Train dataset

```

import matplotlib.pyplot as plt
class_counts = {'Apple': 157, 'Banana': 176, 'Blueberry': 154, 'Orange': 160, 'Papaya': 164, 'Pineapple': 175, 'Strawberry': 165, 'Watermelon': 157}
class_names = list(class_counts.keys())
image_counts = list(class_counts.values())
plt.figure(figsize=(5, 5))
plt.bar(class_names, image_counts)
plt.xlabel("Class Name")
plt.ylabel("Image Count")
plt.title("Class Frequency Distribution in Testing")
plt.xticks(rotation=45, ha="right")
plt.tight_layout()
plt.show()

```

Fig 3.10 Code for Visualization of Test dataset

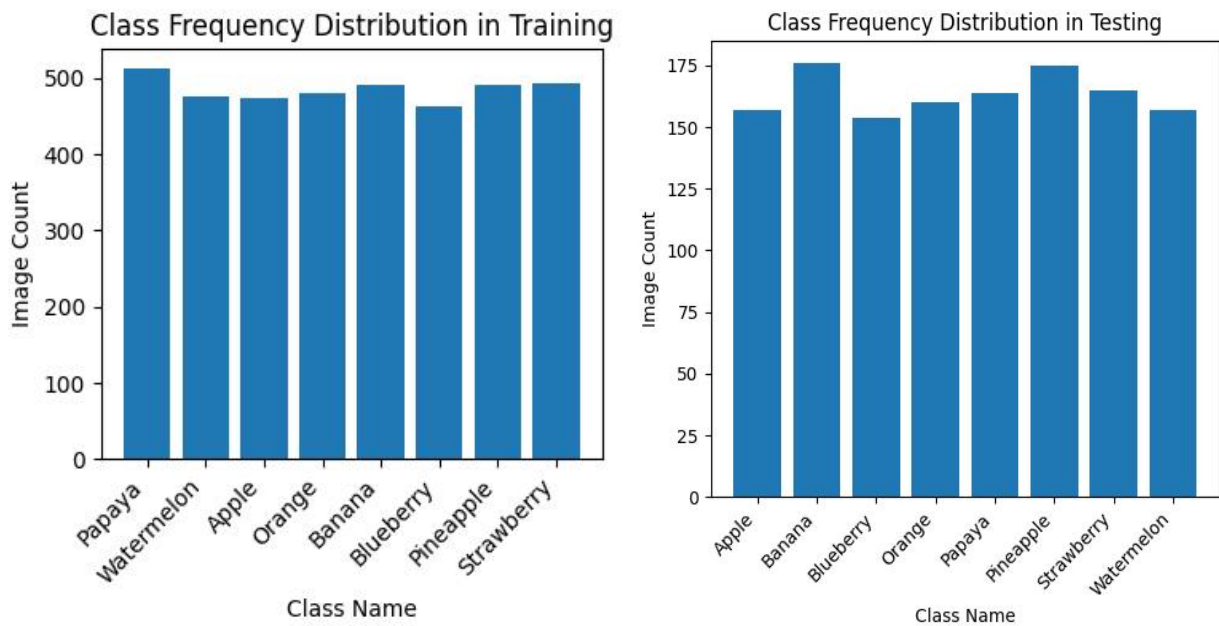


Fig 3.11 Train and Test visualization

3.4 Analysis of Architecture

In my project focused on fruit classification and nutritional information extraction, we utilize the ResNet50 architecture, a pre-trained convolutional neural network (CNN) model from the ImageNet dataset. ResNet50 is known for its deep residual network design, which addresses the vanishing gradient problem by introducing residual connections, enabling efficient training of deeper layers. With 50 layers, including convolutional layers, pooling layers, and residual blocks, ResNet50 excels at learning complex visual features, making it ideal for accurate fruit classification. By leveraging the pre-trained ResNet50 model, we benefit from its proven generalization capabilities and strong performance in image recognition tasks. This analysis focuses on ResNet50's ability to handle the complexity of fruit classification tasks, its computational demands, and how its deep architecture aligns with our project's goals of providing accurate and efficient classification while integrating nutritional information through web scraping.

3.4.1 ResNet50

- ResNet50 is a deep convolutional neural network architecture designed for image classification tasks.
- Composed of 50 layers, including convolutional layers, pooling layers, fully connected layers, and identity/residual blocks.
- Typically processes images of size 224x224x3, representing 224x224 pixel images with three color channels (RGB).
- The network progressively extracts features through multiple layers, ultimately producing a classification output based on learned features.
- In deep networks, gradients can diminish during backpropagation, leading to slow or stalled learning, particularly in very deep networks.

Residual networks (ResNets) were introduced to address the vanishing gradient problem, a common challenge in deep neural networks where exceedingly small gradients occur during backpropagation, leading to slow or stalled learning. The problem is particularly pronounced in very deep networks, as gradients diminish across many layers. The vanishing gradient issue is mitigated by ResNet50 through the incorporation of skip connections, also known

as identity shortcuts, which allow the network to bypass one or more layers and pass the input directly to a later layer. By these skip connections, it is ensured that the gradient can flow more easily through the network during training, effectively preserving the gradient magnitude and enabling the training of much deeper networks.

Key Components of ResNet50

Convolutional Layers: These layers apply a set of filters to the input data to extract features such as edges, textures, and patterns. In ResNet50, convolutional layers are used extensively, with filters of varying sizes (e.g., 3x3, 1x1) to capture different levels of detail. The output of a convolutional layer is passed through an activation function (usually ReLU) to introduce non-linearity.

Residual Blocks: The core innovation of ResNet50 is the residual block, which consists of multiple convolutional layers followed by skip connections. Each block computes a residual mapping by learning the difference between the input and the desired output rather than the full transformation. This allows the network to focus on learning residual functions, which are often easier to optimize. The skip connection in a residual block bypasses the block's layers and adds the input directly to the block's output, effectively enabling the network to learn identity mappings when necessary.

Pooling Layers: Pooling layers, specifically max pooling, are used in ResNet50 to downsample the spatial dimensions of the feature maps, reducing the computational complexity and controlling overfitting. These layers select the maximum value from a defined region (e.g., 2x2) of the feature map, preserving the most salient features while discarding less important information.

Fully Connected Layer: The final layer of the network is a fully connected layer that takes the output of the last residual block and maps it to the output classes. The number of neurons in the fully connected layer is equal to the number of output classes.

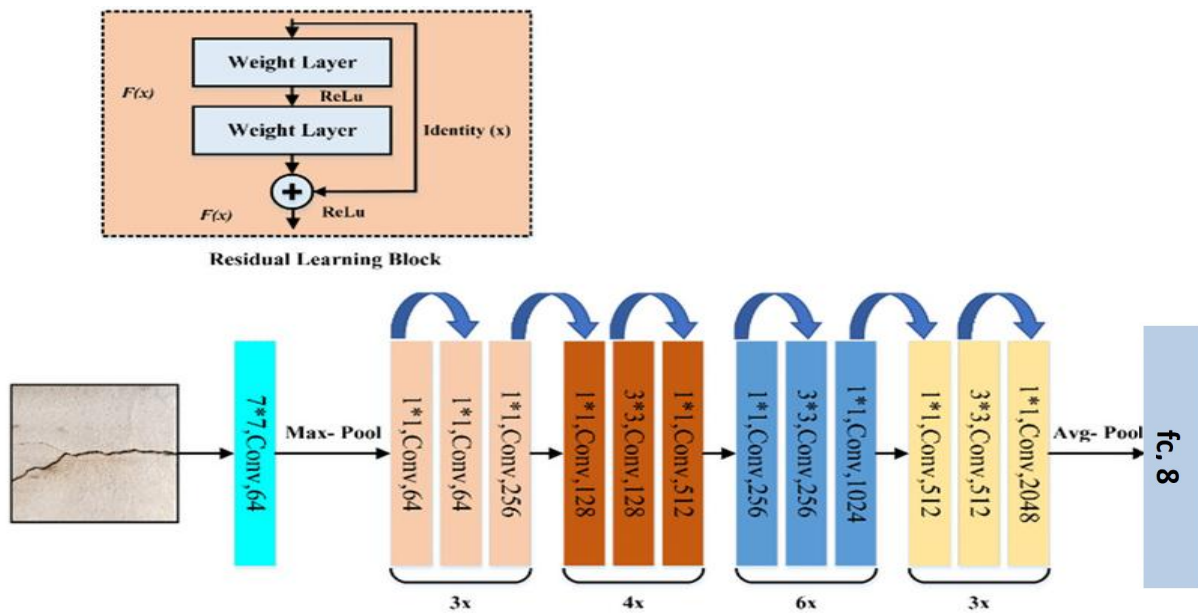


Fig 3.12 ResNet50 Network Architecture

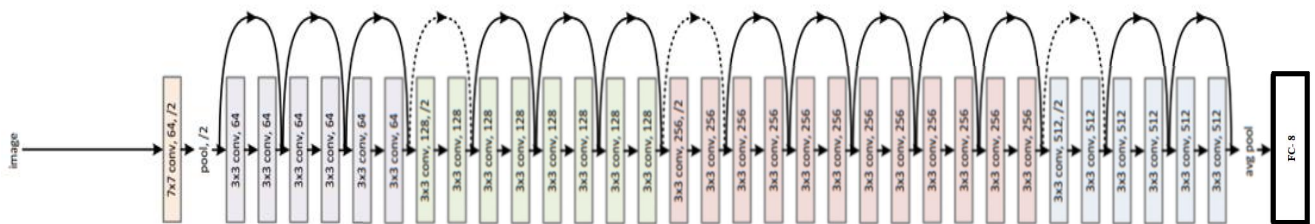


Fig 3.13 ResNet50 Architecture diagram

- A convolution with a kernel size of $7 * 7$ and 64 different kernels all with a stride of size 2 giving us 1 layer.
- Next we see max pooling with also a stride size of 2.
- In the next convolution there is a $1 * 1, 64$ kernel following this a $3 * 3, 64$ kernel and at last a $1 * 1, 256$ kernel, These three layers are repeated in total 3 time so giving us 9 layers in this step.
- Next we see kernel of $1 * 1, 128$ after that a kernel of $3 * 3, 128$ and at last a kernel of $1 * 1, 512$ this step was repeated 4 time so giving us 12 layers in this step.
- After that there is a kernel of $1 * 1, 256$ and two more kernels with $3 * 3, 256$ and $1 * 1, 1024$ and this is repeated 6 time giving us a total of 18 layers.
- And then again a $1 * 1, 512$ kernel with two more of $3 * 3, 512$ and $1 * 1, 2048$ and this was repeated 3 times giving us a total of 9 layers.

- After that we do a average pool and end it with a fully connected layer containing 1000 nodes and at the end a softmax function so this gives us 1 layer.

Each residual block typically consists of three convolutional layers, and the filters in these layers are progressively increased in number as the network depth increases. The final stage of the network includes a global average pooling layer, which reduces each feature map to a single value, followed by a fully connected layer that outputs the classification scores.

Layer (type)	Output Shape	Param #	Connected to
input_layer_1 (InputLayer)	(None, 100, 100, 3)	0	-
sequential (Sequential)	(None, 100, 100, 3)	0	input_layer_1[0][0]
get_item (GetItem)	(None, 100, 100)	0	sequential[0][0]
get_item_1 (GetItem)	(None, 100, 100)	0	sequential[0][0]
get_item_2 (GetItem)	(None, 100, 100)	0	sequential[0][0]
stack (Stack)	(None, 100, 100, 3)	0	get_item[0][0], get_item_1[0][0], get_item_2[0][0]
add (Add)	(None, 100, 100, 3)	0	stack[0][0]
resnet50 (Functional)	(None, 4, 4, 2048)	23,587,712	add[0][0]
global_average_pooling2d (GlobalAveragePooling2D)	(None, 2048)	0	resnet50[0][0]
dropout (Dropout)	(None, 2048)	0	global_average_poolin...
dense (Dense)	(None, 8)	16,392	dropout[0][0]

Total params: 23,604,104 (90.04 MB)
 Trainable params: 16,392 (64.03 KB)
 Non-trainable params: 23,587,712 (89.98 MB)

Table 3.1 Model Summary of ResNet50

Here's a breakdown of the ResNet50 architecture based on the provided details:

1) Input Layer:

- a) Input dimensions: (100, 100, 3)

2) Sequential Layer:

- a) Sequential structure to process input layers.
- b) Connected to the input layer.

3) GetItem Layers:

- a) Three consecutive 'GetItem' layers, each having an output shape of (100, 100).
- b) These are intermediate layers, each connected sequentially.

4) Stack Layer:

- a) Combines outputs of 'GetItem' layers.
- b) Output shape: (100, 100, 3)

5) Add Layer:

- a) Adds outputs from the 'Stack' layer.
- b) Output shape remains (100, 100, 3).

6) ResNet50 Layer (Functional):

- a) Pre-trained ResNet50 model from ImageNet.
- b) Output shape: (4, 4, 2048), indicating the feature maps extracted by ResNet50.
- c) Parameters: 23,587,712

7) Global Average Pooling Layer (GlobalAveragePooling2D):

- a) Reduces the spatial dimensions to a single value per feature map.
- b) Output shape: (2048)

8) Dropout Layer:

- a) Applies dropout regularization to prevent overfitting.
- b) Output shape remains (2048).

9) Dense Layer:

- a) Fully connected layer with 8 output neurons (corresponding to the 8 fruit classes).
- b) Output shape: (8), representing the classification result.

This architecture uses the ResNet50 model as the backbone for feature extraction, followed by additional pooling, dropout, and dense layers to perform the final classification into 8 categories.

Module Name	Output Size	50 Layers
Conv1	112×112	$7 \times 7, 64 \text{ stride}=2$
Conv2_x	56×56	$\left\{ \begin{array}{cc} 1 \times 1, & 64 \\ 3 \times 3 & 64 \\ 1 \times 1 & 256 \end{array} \right\} \times 3$
Conv3_x	28×28	$\left\{ \begin{array}{cc} 1 \times 1, & 128 \\ 3 \times 3 & 128 \\ 1 \times 1 & 512 \end{array} \right\} \times 4$
Conv4_x	14×14	$\left\{ \begin{array}{cc} 1 \times 1, & 256 \\ 3 \times 3 & 256 \\ 1 \times 1 & 1024 \end{array} \right\} \times 6$
Conv5_x	7×7	$\left\{ \begin{array}{cc} 1 \times 1, & 512 \\ 3 \times 3 & 512 \\ 1 \times 1 & 2048 \end{array} \right\} \times 3$
Pooling	1×1	Average Pool, 8-d FC

Table 3.2 Dimensionality table of ResNet50

The table illustrates the architecture of the ResNet-50 model, a deep convolutional neural network that is widely used for tasks like image classification. ResNet-50, part of the ResNet (Residual Network) family, introduces a unique technique called "residual learning," which helps mitigate the vanishing gradient problem in very deep networks.

- The first layer in ResNet-50 is denoted as Conv1. Here, the input image is processed with a large 7×7 convolution filter that has 64 filters and is applied with a stride of 2. This results in an output feature map size of 112×112 , effectively reducing the spatial dimensions of the input while extracting low-level features such as edges and textures. This step is typically followed by max pooling, which further reduces the spatial resolution and helps to summarize important features from local regions.
- The Conv2_x to Conv5_x blocks represent the core of ResNet-50. Each of these stages consists of multiple residual blocks, where each block contains three layers of convolutions:
 - A 1×1 convolution that reduces the number of channels.
 - A 3×3 convolution that processes the reduced channels.
 - Another 1×1 convolution that restores the original number of channels.

This design allows for deep feature extraction while maintaining efficiency in terms of computation. The number of filters increases progressively from 64 in Conv2_x to 2048 in

Conv5_x. The spatial resolution of the feature maps is progressively reduced, starting from 56×56 after Conv2_x, down to 7×7 after Conv5_x. This decreasing spatial dimension helps the network focus on increasingly abstract and high-level features of the input.

- **Residual Connections:** A key innovation in ResNet is the use of residual connections, which help combat the problem of vanishing gradients in deep networks. In each residual block, a skip connection bypasses the convolutional layers and directly adds the input of the block to the output. This skip connection allows the network to learn the "residual" mapping rather than the full mapping, making it easier to train deeper networks and improving overall performance. These residuals help ResNet-50 maintain high accuracy even with 50 layers, which would typically make training difficult in traditional deep networks.
- After passing through the convolutional blocks, the feature map size is reduced to 7×7 . Before feeding this data into the fully connected layers for classification, a global average pooling operation is applied. This converts the 7×7 feature map into a 1×1 feature vector, which reduces the amount of data while retaining important information. Finally, the pooled features are passed to a 8-dimensional fully connected layer, corresponding to the 8 output classes for the dataset.

3.5 Project Plan

3.5.1 Project Pipeline

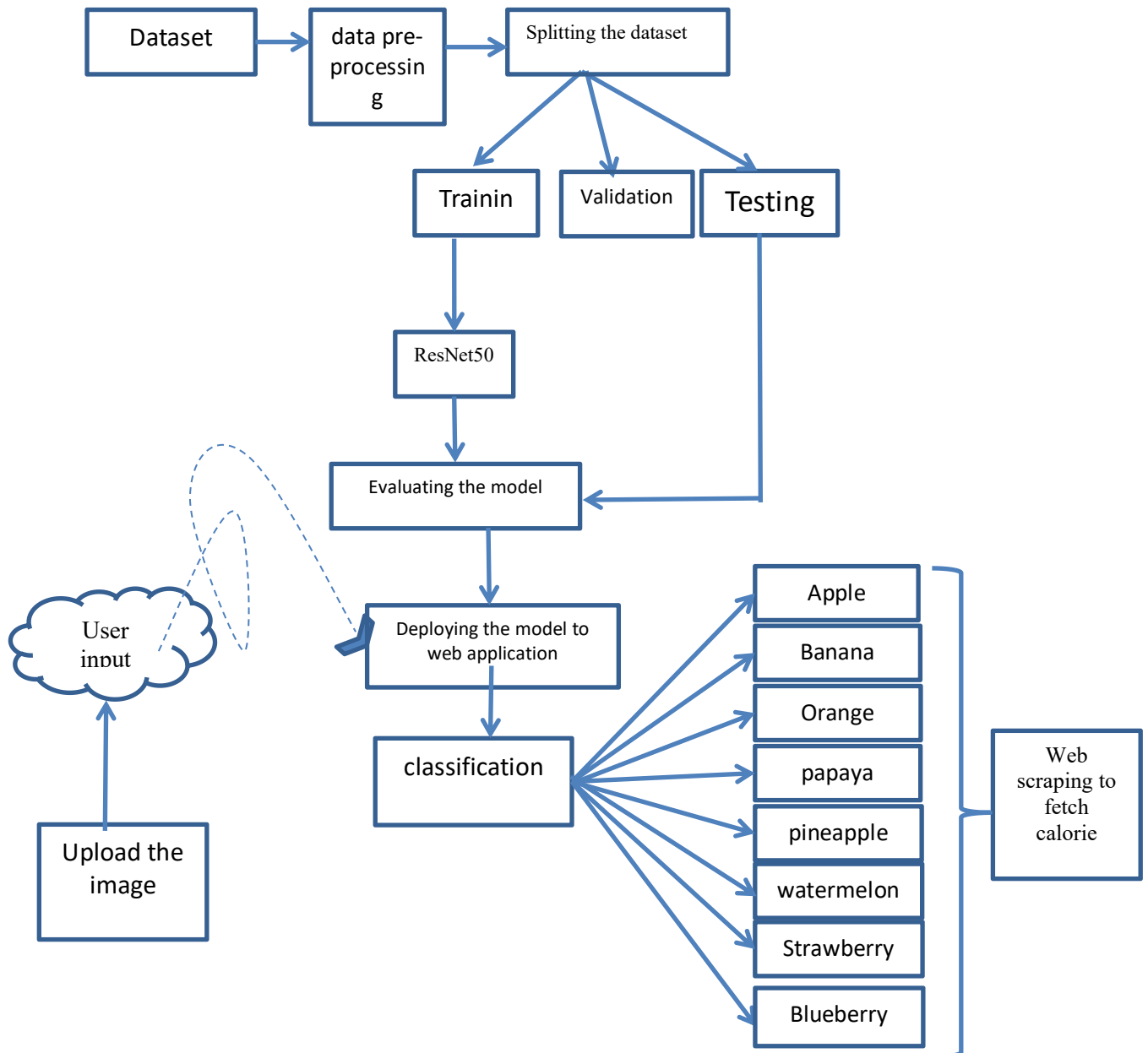


Fig 3.14 Pipeline

The steps involved in training a deep learning model for a web application. Here's a breakdown of the process:

- **Dataset Creation:** The initial step involves compiling a dataset containing images of the eight fruit classes (Apple, Banana, Blueberry, Watermelon, Strawberry, Papaya, Pineapple, Orange). This dataset will be used for training the model.
- **Data Preprocessing:** Once the dataset is assembled, it undergoes preprocessing. This includes augmenting the data, resizing them, and converting them to a suitable format that can be effectively used by the ResNet-50 model.
- **Splitting the Dataset:** After preprocessing, the dataset is divided into three distinct sets: training, validation, and testing. The training set is utilized for model training, the validation set helps monitor performance and fine-tune hyperparameters, and the testing set evaluates the model's performance on unseen data.
- **Model Selection:** In this phase, the ResNet-50 pretrained model is selected for training. Pre-trained models like ResNet-50 leverage prior knowledge gained from training on a large dataset, enabling them to perform well on specific tasks with less training time.
- **Training the Model:** The ResNet-50 model is trained using the training dataset. During this phase, the model learns to classify images into one of the eight fruit classes.
- **Evaluation:** Once the model has been trained, it is evaluated using the testing dataset. This evaluation assesses the model's accuracy and overall performance in classifying the images.
- **Deployment:** After evaluation, the model is deployed to a web application, allowing users to interact with it.
- **User Input:** When a user uploads an image of a fruit, the image is processed and fed into the deployed ResNet-50 model.
- **Prediction:** The model classifies the uploaded image into one of the eight fruit classes.
- **Calorie Fetching:** Upon classification, the application scrapes relevant information from the web to fetch the calorie content of the identified fruit class and presents this information to the user.

3.5.2 Project Development Plan

- Submission of project synopsis with Journal Papers - 22.07.2024
- Project proposal approval - 26.07.2024
- Approval Committee - 29.07.2024 & 30.07.2024
- Initial report submission - 12.08.2024
- Analysis and design report submission - 16.08.2024
- First project presentation - 21.08.2024 & 23.08.2024
- Sprint Release I - Building the base model and completing the visualization on 30-09-2024
- Sprint Release II - Completeing the training and interface designing on 4-10-2024
- Interim project presentation - 30.09.2024 & 01.10.2024
- Sprint Release III - the complete project by integrating the model with interface and calorie fetching .
- Submission of the project report to the guide - 28.10.2024
- Final project presentation - 28.10.2024 & 29.10.2024
- Submission of project report after corrections - 01.11.2024

3.6 Feasibility Analysis

A feasibility study for the Fruit Classification and Calorie Counter Web Application aims to assess the viability and practicality of implementing such a system. The feasibility of the system is evaluated in terms of the following categories:

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

3.6.1 Technical Feasibility

The proposed Fruit Classification and Calorie Counter system is technically feasible, leveraging widely available tools and technologies. The core technology involves deep learning frameworks such as TensorFlow, which are robust and well-supported. The ResNet-50 pretrained model can be easily integrated, capitalizing on the extensive community and resources available for these technologies.

The hardware requirements are minimal since the model can be deployed on cloud platforms, eliminating the need for specialized local servers. Furthermore, the necessary image processing libraries for data preprocessing and web scraping tools for calorie fetching are readily accessible, enhancing the system's implementation potential. Advances in deep learning have significantly improved classification accuracy, ensuring the model can effectively differentiate between the eight fruit classes.

3.6.2 Economical Feasibility

From an economic standpoint, the Fruit Classification and Calorie Counter system presents favorable feasibility. While initial development costs may involve expenses related to cloud hosting and software development, many essential tools are open-source, significantly reducing overall costs. The use of pre-trained models like ResNet-50 decreases the need for extensive training data and computational resources. Additionally, as the model can be hosted on affordable cloud platforms, ongoing operational costs remain low. The potential benefits include increased user engagement and satisfaction, which could translate into revenue through advertisements or premium features, further enhancing economic viability.

3.6.3 Operational Feasibility

Operational feasibility focuses on the system's practicality and usability in real-world applications. The Fruit Classification and Calorie Counter system offers significant advantages by providing users with an intuitive interface for classifying fruits and accessing nutritional information seamlessly.

The system's reliance on advanced technologies such as deep learning and web scraping ensures that users receive accurate and timely results. The user-friendly web application can be easily accessed on various devices, promoting widespread adoption. Moreover, integrating real-time image classification enhances the user experience by allowing instant feedback, which is crucial for maintaining user engagement.

The Fruit Classification and Calorie Counter system demonstrates strong technical, economic, and operational feasibility, making it a viable solution for fruit classification and nutritional awareness. By leveraging computer vision for image classification, machine learning for model training, and web scraping for calorie information retrieval, the system effectively addresses the needs of users interested in healthy eating.

Machine Learning and Artificial Intelligence: Machine learning techniques are pivotal in the development of the classification model. The ResNet-50 pretrained model will be fine-tuned using the training dataset, allowing the model to learn specific features and characteristics of the fruit classes.

Web Scraping: Web scraping methods will be employed to gather nutritional information about the classified fruits. By automating data retrieval from reliable sources, the application can provide users with accurate calorie counts and other nutritional details, enhancing the application's functionality and value.

3.7 System Environment

Understanding the system environment is crucial for ensuring optimal performance and compatibility. This section outlines the operating systems, hardware specifications, and software dependencies necessary for the system to function efficiently. By providing clarity on these factors, users can make informed decisions regarding deployment, configuration, and maintenance.

3.7.1 Software Environment

The development of this application required a variety of software tools and frameworks to ensure its functionality, performance, and scalability. Each of these tools contributed to the overall robustness, user-friendliness, and efficiency of the system. Below is a detailed overview of the primary technologies employed in the development of the fruit classification and calorie counter web application:

- **Python:** Python, a versatile and high-level programming language, was the backbone of this application. Known for its simplicity and extensive support for libraries, Python enabled rapid development and system integration. Various Python libraries were essential to handle the data processing, model training, and the creation of visualizations required in this application:
 - **NumPy:** NumPy is a fundamental library for numerical computations in Python. It supports the creation and manipulation of arrays and matrices, which are crucial for handling large sets of image data used in machine learning tasks. In this application, NumPy was essential for efficient data preprocessing and manipulation tasks during model training and evaluation.
 - **Matplotlib:** Matplotlib is a plotting library that helps in visualizing data. It was used in this project to generate graphs and plots, enabling the analysis of model performance, including loss curves and accuracy metrics. These visualizations are key to understanding how the deep learning model behaves during the training and testing phases.
- **Pillow (PIL):** Pillow, a fork of the original Python Imaging Library (PIL), is used for opening, manipulating, and saving image files in various formats. This was particularly

useful for preprocessing the fruit images, such as resizing, cropping, and normalizing them before they were fed into the deep learning model. The library also supports image augmentation techniques that can help improve the robustness of the model by generating additional training examples.

- **TensorFlow:** TensorFlow is an open-source deep learning framework developed by Google, widely used for machine learning and artificial intelligence tasks. In this application, TensorFlow was used to implement and train the ResNet-50 model—a powerful convolutional neural network for image classification. TensorFlow's capabilities, such as GPU acceleration, made it easier to train the model efficiently on large datasets like the Fruit-360 dataset. TensorFlow also provided tools for model evaluation, deployment, and optimization, ensuring that the system could perform well in real-world scenarios. The framework supports both research and production environments, making it a solid choice for building and deploying scalable machine learning models.
- **Streamlit:** Streamlit is an open-source Python library designed for creating fast and interactive web applications. It was chosen for this project to build the user interface for the fruit classification system. Streamlit simplifies the process of building UIs by providing a high-level API to design interactive elements such as buttons, sliders, and forms. This allowed the development team to quickly implement the frontend of the application, which includes sections for real-time fruit detection, uploading images for classification, and visualizing the results. With Streamlit, the application can display predictions and even render graphs that show model performance, making it user-friendly and visually appealing.
- **Selenium:** Selenium is a web automation tool used to scrape data from web pages. In this application, Selenium was integrated to fetch calorie information related to the classified fruit from various web sources. By automating web browsers, Selenium was able to retrieve up-to-date calorie data and display it to users after the classification process. This feature enhances the overall functionality of the application, offering users additional insights beyond the classification result.
- **HTML and CSS:** To ensure the frontend of the application is both functional and visually appealing, HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) were

used alongside Streamlit. HTML provides the structure for the web pages, while CSS is responsible for styling the user interface. Together, they ensure that the application is easy to navigate and visually coherent. Though Streamlit generates much of the interface automatically, custom HTML and CSS elements were employed to fine-tune the layout and design, resulting in a polished and professional appearance.

- **GitHub:** GitHub is a web-based platform that supports version control using Git, making it a crucial tool for managing the project's codebase. In this development process, GitHub served as a centralized repository where developers could push code updates, track changes, and collaborate. Features like issue tracking and pull requests made it easier to organize development tasks and review contributions, ensuring that the project progressed smoothly. GitHub also served as a platform for showcasing the project, allowing contributors to collaborate efficiently, manage code revisions, and maintain proper documentation.
- **Visual Studio Code:** Visual Studio Code (VS Code) is a lightweight yet powerful source code editor used throughout the project. It provided an integrated development environment (IDE) with features like syntax highlighting, debugging, Git integration, and code refactoring tools. The versatility of VS Code made it a perfect choice for writing Python code, managing dependencies, and debugging both backend and frontend components of the application. Its extensions for Python and TensorFlow further streamlined the development workflow by offering real-time linting, auto-completion, and debugging support for machine learning applications.
- **Other Tools and Technologies:** In addition to the core technologies mentioned above, other utilities played a supporting role in the development and deployment of the project. These include Python libraries for data manipulation, testing, and deployment, as well as cloud services for hosting the web application and managing infrastructure.

3.7.2 HARDWARE ENVIRONMENT

Hardware configuration significantly impacts software development, influencing performance and scalability. Choices regarding CPU, RAM, storage, and network infrastructure directly affect application execution and data handling. Moreover, considerations extend to system architecture and compatibility, ensuring efficiency and reliability in modern software solutions.

- **Processor:** The application was developed on a machine equipped with an 11th Gen Intel® Core™ i3-1115G4 processor. This dual-core processor operates at a base frequency of 3.00 GHz, with the ability to dynamically boost speeds using Intel® Turbo Boost technology. Although this processor is not among the highest-end CPUs, its performance is sufficient for running development tasks, such as code compilation, debugging, and moderate machine learning model training.
- **RAM:** The machine is equipped with 8.00 GB of RAM, which is essential for handling multiple tasks simultaneously, such as running IDEs, web browsers, and background services. This amount of memory is also adequate for working with moderate-sized datasets and performing image classification tasks without significant slowdowns. However, for very large datasets or highly complex models, upgrading to a higher RAM capacity might be beneficial to prevent memory bottlenecks during training or inference.
- **Memory:** The machine uses a 512 GB SSD (Solid State Drive), which provides fast read/write speeds compared to traditional HDDs. This significantly improves the performance of the system in tasks such as booting, opening applications, and loading large datasets, which is essential for machine learning projects. The SSD also ensures quicker model loading times and efficient file handling during data preprocessing tasks.
- **System Type:** The development environment is a 64-bit operating system running on an x64-based processor. A 64-bit system allows for efficient memory addressing and is capable of running modern machine learning frameworks like TensorFlow, which often leverage 64-bit processing for faster computations and better utilization of hardware resources.
- **Good internet connectivity.**

4. SYSTEM DESIGN

System design in the context of a deep learning project involves planning and structuring the components and workflows essential for building, training, and deploying deep learning models.

4.1 Model Building

Model building in the context of a deep learning project refers to the process of creating and training a neural network architecture to perform a specific task, such as image classification, object detection, or natural language processing. This process involves selecting an appropriate neural network architecture, defining its structure and parameters, preparing the training data, optimizing the model's performance through iterative training, and evaluating its performance on validation or test datasets. Model building is a crucial step in developing deep learning solutions and requires expertise in neural network design, optimization techniques, and domain-specific knowledge.

4.1.1 Model Planning

The model was built using the ResNet50 architecture, a deep learning model pre-trained on ImageNet. The base ResNet50 model was utilized for feature extraction, and the top layers were replaced to fit the 8-class fruit classification task.

```
base_model = tf.keras.applications.resnet.ResNet50(  
    input_shape=(img_height, img_width, 3),  
    include_top=False,  
    weights='imagenet'  
)  
base_model.trainable = False  
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()  
prediction_layer = tf.keras.layers.Dense(num_classes)  
inputs = tf.keras.Input(shape=(100, 100, 3))  
x = data_augmentation(inputs)  
x = preprocess_input(x)  
x = base_model(x, training=False)  
x = global_average_layer(x)  
x = tf.keras.layers.Dropout(0.2)(x)  
outputs = prediction_layer(x)  
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

Fig 4.1 Loading ResNet50 pretrained model

Here's a breakdown of the provided code:

1. Load ResNet50 Model:

```
base_model = tf.keras.applications.resnet.ResNet50( input_shape=(img_height, img_width, 3),
include_top=False,weights='imagenet')
```

- `tf.keras.applications.resnet.ResNet50`: This line initializes the ResNet50 model without the fully connected (top) layers.
- `input_shape=(img_height, img_width, 3)`: Specifies the expected input shape for the model, where `img_height` and `img_width` represent the dimensions of the input images, and 3 indicates the RGB color channels.
- `include_top=False`: Omits the top layers of the network, allowing for customization based on your specific classification task.
- `weights='imagenet'`: Loads the pre-trained weights from the ImageNet dataset, which enhances the model's performance by leveraging learned features.

2. Freeze Base Model Layers:

```
base_model.trainable = False
```

- This line sets the trainable property of the base model to False, effectively freezing all layers of the ResNet50 model. This prevents the weights of these layers from being updated during training, which is useful when fine-tuning the model.

3. Define Global Average Pooling Layer:

```
global_average_layer = tf.keras.layers.GlobalAveragePooling2D()
```

- `GlobalAveragePooling2D()`: This layer reduces the spatial dimensions of the feature maps to a single vector per feature map by calculating the average of all values. It converts the 3D output from the ResNet model into a 1D feature vector, which is suitable for the subsequent dense layers.

4. Define the Output Layer:

```
prediction_layer = tf.keras.layers.Dense(num_classes)
```

- `Dense(num_classes)`: Creates a dense layer with `num_classes` output units. Each unit corresponds to a class in the classification task. However, to interpret the output as probabilities for multi-class classification, you should include a softmax activation function in this layer.

5. Define Input Layer:

```
inputs = tf.keras.Input(shape=(100, 100, 3))
```

- `tf.keras.Input(shape=(100, 100, 3))`: This line defines the input layer of the model, specifying the shape of the input images, which are 100x100 pixels with 3 channels (RGB).

6. Data Augmentation and Preprocessing:

```
x = data_augmentation(inputs)
```

```
x = preprocess_input(x)
```

- `data_augmentation(inputs)`: This function applies data augmentation techniques (such as rotation, scaling, and flipping) to the input images. It helps increase the diversity of the training data and reduces the risk of overfitting.
- `preprocess_input(x)`: Standardizes the input data according to the requirements of the ResNet50 model, typically scaling the pixel values to a range suitable for the model (like zero-centering).

7. Forward Pass through the Base Model:

```
x = base_model(x, training=False)
```

- This line passes the augmented and preprocessed input data `x` through the ResNet50 model. The `training=False` argument indicates that the model should run in inference mode, which affects the behavior of certain layers like dropout and batch normalization.

7. Apply Global Average Pooling:

```
x = global_average_layer(x)
```

- The output from the ResNet50 model (which is a 4D tensor) is passed through the global average pooling layer, producing a 2D tensor that represents a summary of the features.

9. Add Dropout Layer:

```
x = tf.keras.layers.Dropout(0.2)(x)
```

- `Dropout(0.2)`: This layer introduces dropout regularization with a dropout rate of 20%. It randomly sets 20% of the input units to 0 during training to help prevent overfitting by ensuring that the model does not rely too heavily on any specific feature.

10. Generate Model Outputs:

```
outputs = prediction_layer(x)
```


- The processed data `x` is passed through the prediction layer to generate logits for each class.

As mentioned earlier, adding an activation function such as softmax is essential for multi-class classification.

11. Create the Final Model:

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

- This line constructs a Keras model by specifying the inputs and outputs. It connects the input layer to the output layer, integrating all the defined layers into a complete model ready for training.

```
 optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001)

model.compile(
    optimizer=optimizer,
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=['accuracy']
)
```

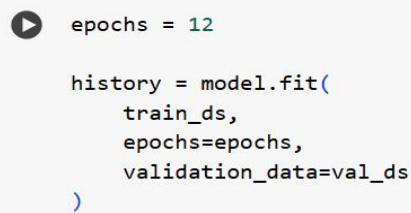
Fig 4.2 Model compiling

- a) Loss Function: `loss='categorical_crossentropy'`: This is the loss function used for multi-class classification tasks. It measures the difference between the model's predicted probabilities and the actual class labels.
- b) Optimizer: `optimizer='adam'`: Specifies the optimization algorithm used to update the model weights during training. Adam (Adaptive Moment Estimation) is a widely used optimizer known for its efficiency and effectiveness.
- c) Metrics: `metrics=['accuracy']`: Defines the evaluation metric used to monitor the model's performance during training and validation. Accuracy measures the proportion of correctly classified samples.

By compiling the model with these settings, it is prepared for training. The model will aim to minimize the categorical crossentropy loss while maximizing accuracy on the training data.

4.1.2 Model Training

The model training phase involves utilizing the ResNet50 deep learning architecture, which was pre-trained on ImageNet, to effectively train on the fruit classification dataset. The model was fine-tuned for classifying images of eight different fruits, ensuring optimal performance for this specific task.



```
epochs = 12

history = model.fit(
    train_ds,
    epochs=epochs,
    validation_data=val_ds
)
```

Fig 4.3 Training ResNet50 model

This line invokes the fit method of the ResNet50 model to begin the training process.

- The training data is loaded from the specified directory containing the fruit images, with the dataset divided into training and validation sets.
- **epochs=12** sets the number of training epochs to 12, indicating how many times the entire dataset will be passed through the model during training.
- **batch_size=32** sets the batch size to 32, determining how many samples are processed before the model's weights are updated.
- **input_size=(100, 100)** specifies the input image dimensions, with all images resized to 100x100 pixels to ensure consistency during training.

The dataset consisted of 3,835 training images and 1,277 testing images, organized into 8 classes.

The split ratio was roughly 75% training data and 25% testing data.

Training Configuration:

- Optimizer: Adam
- Loss Function: Sparse Categorical Crossentropy
- Metrics: Accuracy
- Batch Size: 32
- Epochs: 12

Epochs 12

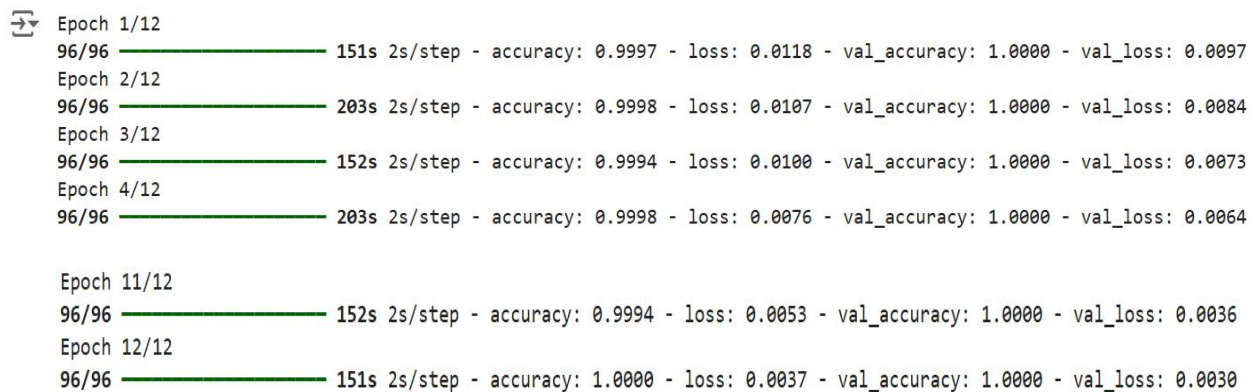


Fig 4.4 ResNet50 12 epochs

- **accuracy:** These are the accuracy and loss values of the model after the each epoch. Accuracy is a measure of how well the model performs on a set of data, while loss is a measure of how far the model's predictions are from the actual values.

Intermediate Training Results:

- During training, the model's performance was monitored on both the training and validation datasets. Some key intermediate results from the training phase:
- Training Accuracy: 99.94%
- Validation Accuracy: 100%
- Loss: 0.0037

4.1.3 Model Testing

Model testing involves evaluating the trained ResNet50 model on unseen test data from the fruit classification dataset to assess its performance and generalization ability. This phase includes measuring metrics such as accuracy to determine how well the model classifies different fruits. Additionally, a confusion matrix is generated to provide deeper insights into the model's classification performance across the eight fruit classes, helping to identify potential miss classifications.

After the training phase, the model was evaluated on the test dataset, which was not used during training. This helped measure the model's ability to generalize to unseen data.

```
[ ] # Evaluate the model on the test dataset
    loss, accuracy = model.evaluate(test_ds)
    print('Test accuracy:', accuracy)
```

40/40 ————— 145s 4s/step - accuracy: 1.0000 - loss: 0.0029
Test accuracy: 1.0

Fig 4.5 Testing with Testing dataset

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications.resnet50 import preprocess_input
model = tf.keras.models.load_model('/content/drive/MyDrive/Dataset/models/my_model.keras')
img_height, img_width = 100, 100
img = tf.keras.preprocessing.image.load_img(
    '/content/drive/MyDrive/Dataset/fruit-360/validation/Banana/40_100.jpg',
    target_size=(img_height, img_width)
)
img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)
img_array = preprocess_input(img_array)
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions, axis=1)[0]
class_names = ['Apple', 'Banana', 'Blueberry', 'Orange', 'Papaya', 'Pineapple', 'Strawberry', 'Watermelon']
predicted_label = class_names[predicted_class]
plt.imshow(img)
plt.title(f'Predicted Class: {predicted_label}')
plt.axis('off')
plt.show()
```

1/1 ————— 4s 4s/step

Fig 4.6 Testing with Validation dataset



Fig 4.7 Prediction 1

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications.resnet50 import preprocess_input
model = tf.keras.models.load_model('/content/drive/MyDrive/Dataset/models/my_model.keras')
img_height, img_width = 100, 100
img = tf.keras.preprocessing.image.load_img(
    '/content/drive/MyDrive/Dataset/fruit-360/download.jpg',
    target_size=(img_height, img_width)
)
img_array = tf.keras.preprocessing.image.img_to_array(img)
img_array = tf.expand_dims(img_array, 0)
img_array = preprocess_input(img_array)
predictions = model.predict(img_array)
predicted_class = np.argmax(predictions, axis=1)[0]
class_names = ['Apple', 'Banana', 'Blueberry', 'Orange', 'Papaya', 'Pineapple', 'Strawberry', 'Watermelon']
predicted_label = class_names[predicted_class]
plt.imshow(img)
plt.title(f'Predicted Class: {predicted_label}')
plt.axis('off')
plt.show()
```

1/1 — 4s 4s/step

Fig 4.8 Testing with unseen data



Fig 4.9 Prediction 2

5.RESULTS AND DISCUSSION

ResNet50 pretrained model

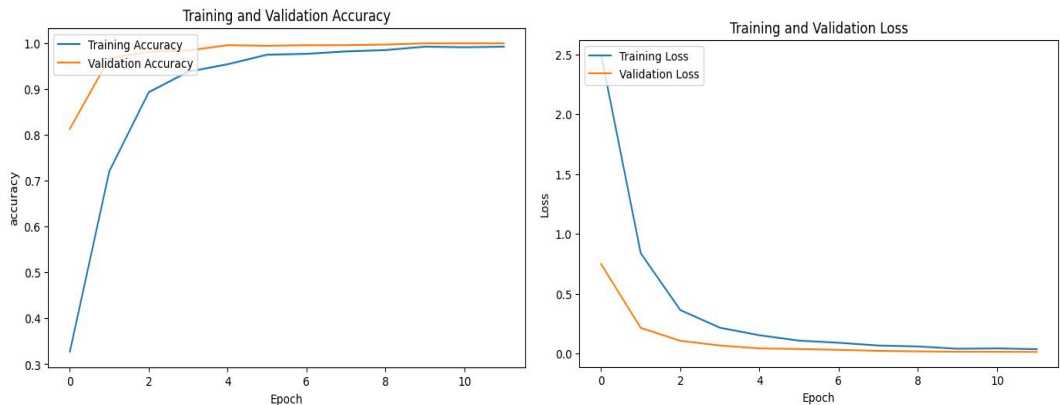


Fig 5.1 Accuracy and Loss curve of ResNet50

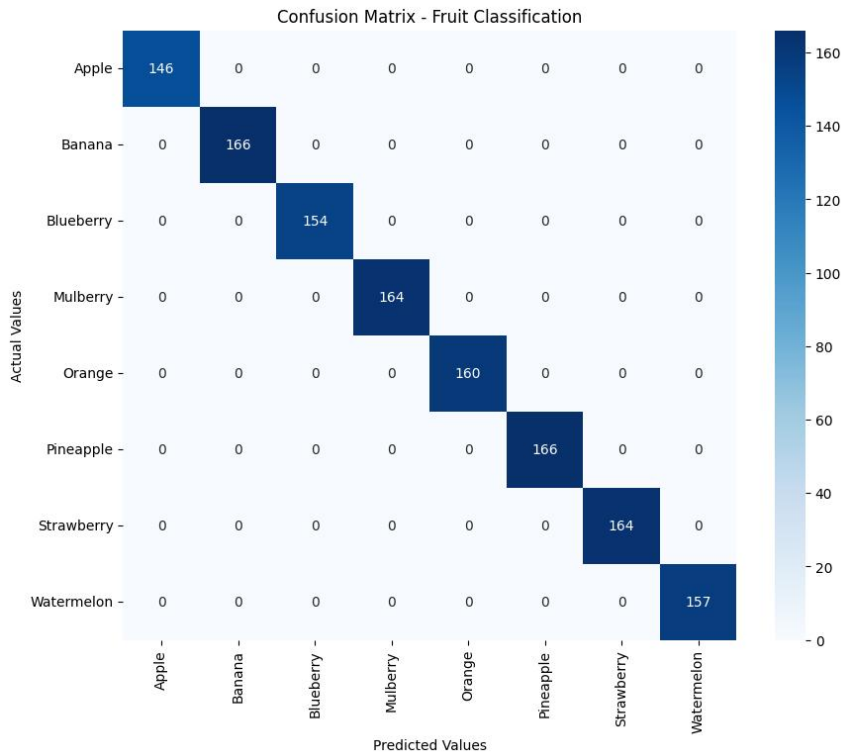


Fig 5.2 Confusion Metrix

Classification Report:				
	precision	recall	f1-score	support
Apple	1.00	1.00	1.00	146
Banana	1.00	1.00	1.00	166
Blueberry	1.00	1.00	1.00	154
Orange	1.00	1.00	1.00	160
Papaya	1.00	1.00	1.00	164
Pineapple	1.00	1.00	1.00	166
Strawberry	1.00	1.00	1.00	164
Watermelon	1.00	1.00	1.00	157
accuracy			1.00	1277
macro avg	1.00	1.00	1.00	1277
weighted avg	1.00	1.00	1.00	1277

Fig 5.3 Classification Report

6.MODEL DEPLOYMENT

For the deployment of my project, I have utilized a Streamlit web application, designed to be interactive and user-friendly. The main interface is structured around two key components: About and Prediction.

The Home page serves as the central hub for accessing the project's functionalities and contains a dashboard with the following sections:

- **About Section:** The About section provides detailed information about the project, including the dataset used, the project's applications, and technical aspects of the underlying deep learning model. This section helps users gain insight into the methodology and goals of the fruit classification and calorie prediction system.
- **Prediction Section:** The Prediction page enables users to easily upload images in JPG or PNG format for classification. Once an image is uploaded, the original image and augmented images (such as rotated or flipped versions) are displayed on the screen.

Users can then click the Predict button to classify the fruit image using the trained model. After the prediction is made, the application employs Selenium to fetch the calorie information of the predicted fruit from the Nutritionix website. The calorie details along with the URL to the source are displayed, providing users with easy access to nutritional information for their predicted fruit. The streamlined process and integration of web scraping for calorie data ensure that users not only get fruit classification results but also relevant nutritional insights, making the application practical for dietary tracking and analysis.

6.1UI DESIGN

Under the UI Design section, showcasing screenshots of the user interface enhances the project's visual presentation and provides users with a glimpse of the application's layout and functionality. By including images of the Home, About and prediction result pages, users can visually understand the navigation flow and design aesthetics. These screenshots offer a quick overview of the application's appearance and enable users to familiarize themselves with its interface before interacting with the live application.

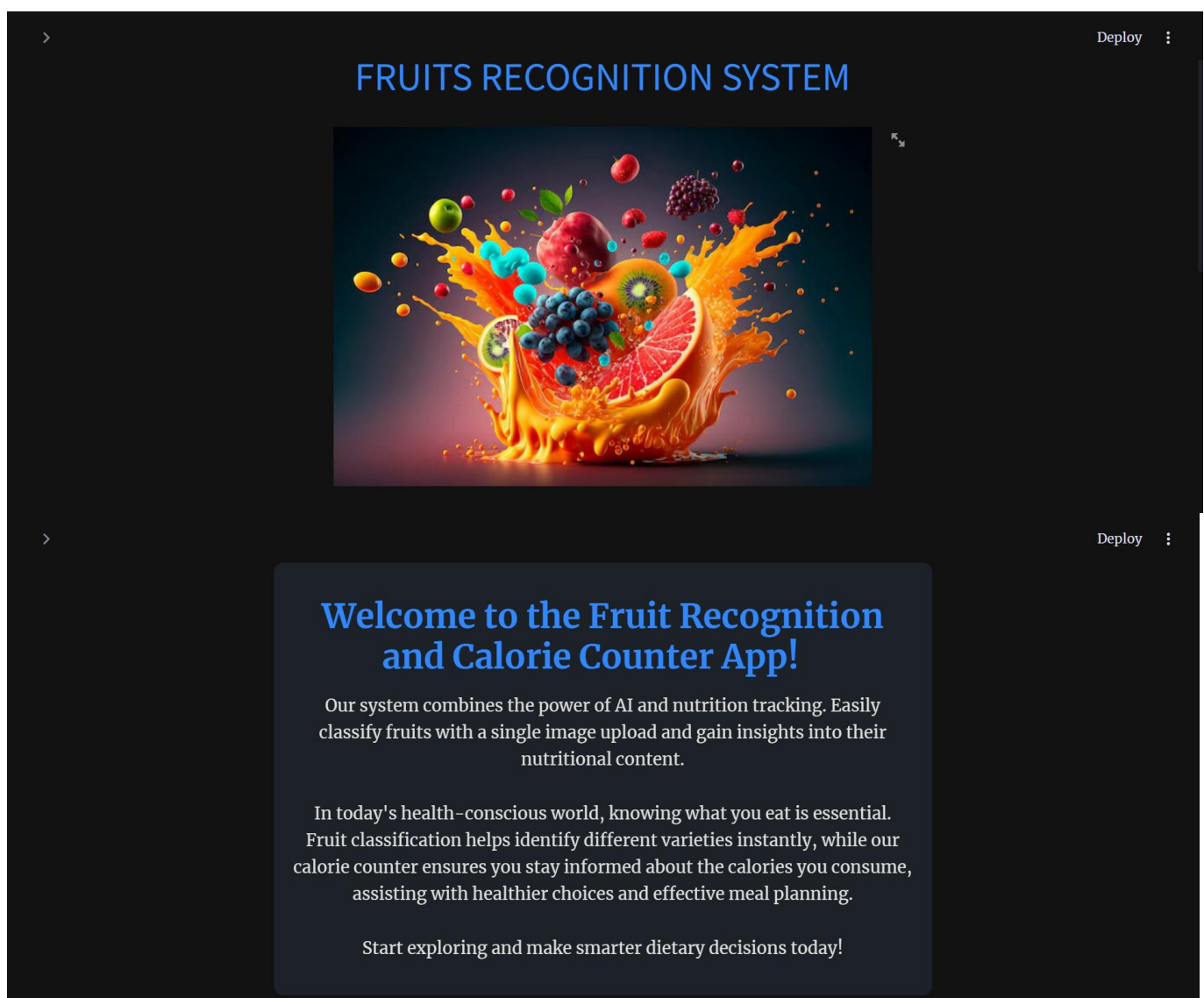


Fig 6.1 Home page 1

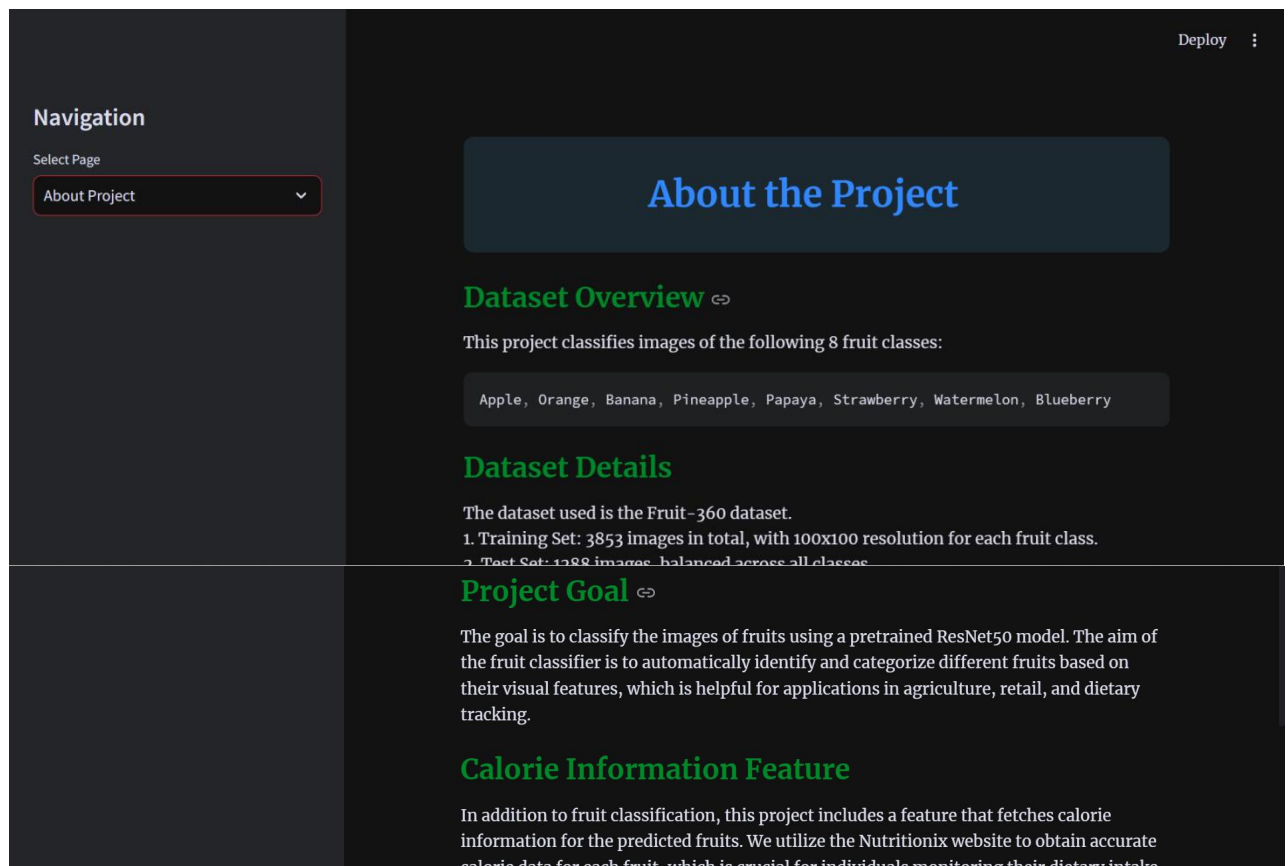


Fig 6.2 About page

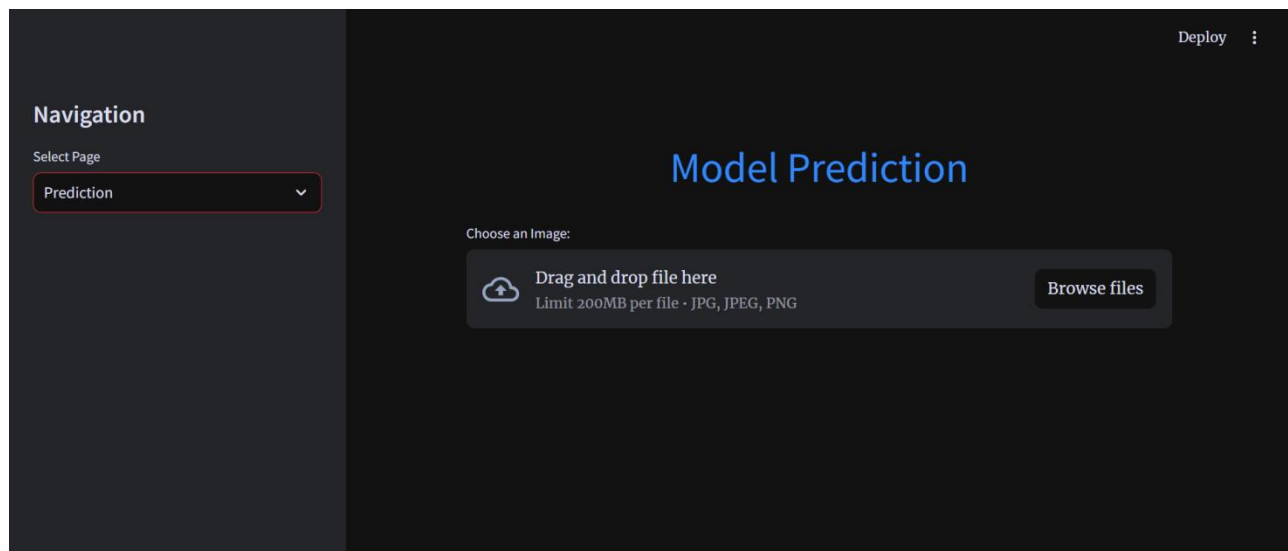


Fig 6.3 Prediction page 1

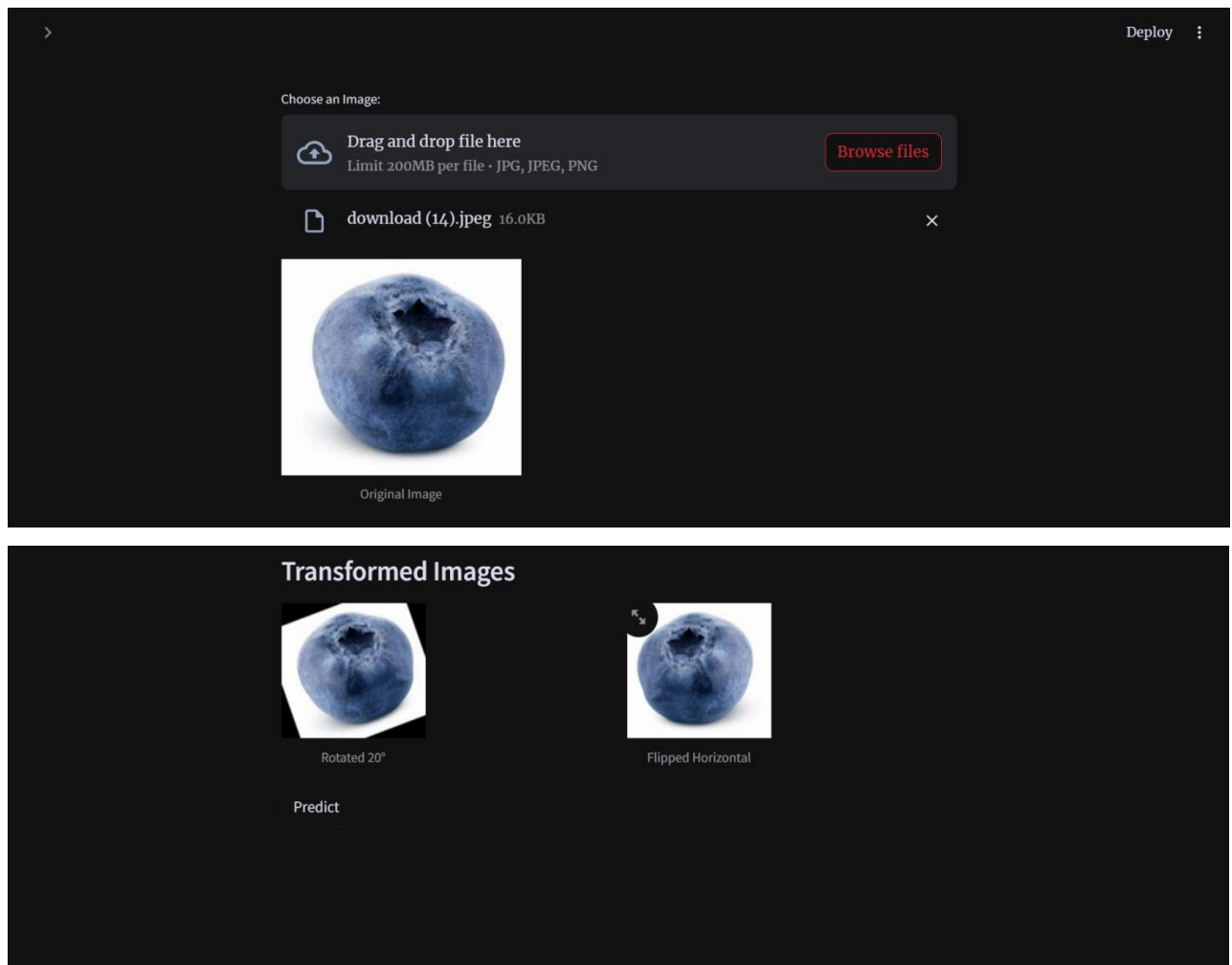


Fig 6.4 Prediction page 2

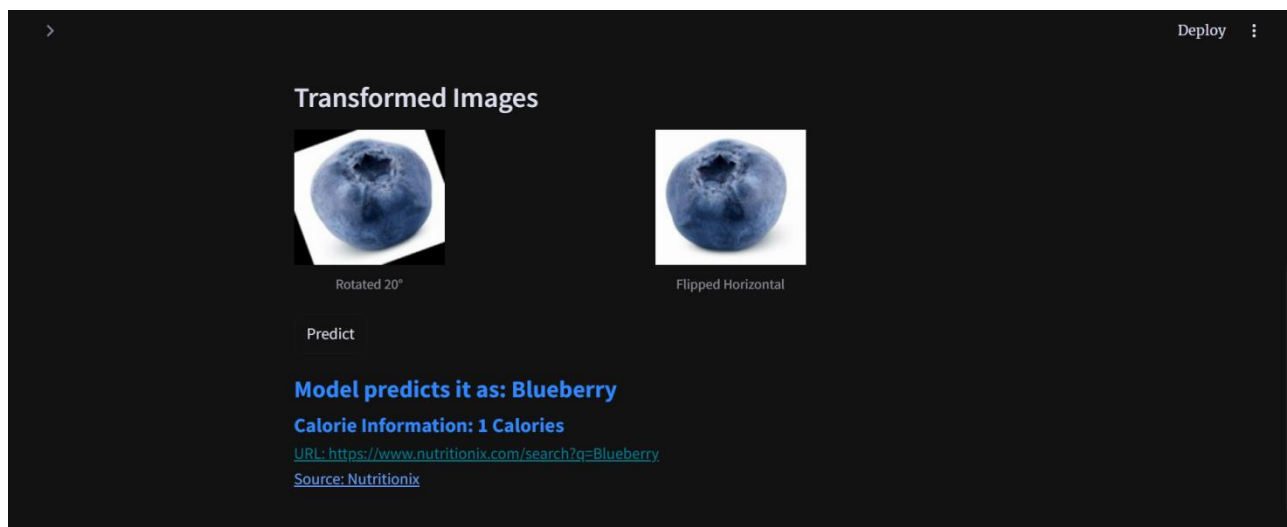


Fig 6.5 Prediction and calorie displayed

7.GIT HISTORY

Git Repository “Fruit classification and calorie counter” contains all colab files, dataset, py files, and three related research papers. It is maintained for systematic way of project presentation and mainly for future reference. The colab files are created separately for three sprint releases during the project.

PROJECTS / Fruit Recognition and calorie counter /

sandra202002 Add files via upload f16ae6f · last week History

Name	Last commit message	Last commit date
..		
Sprint1	Delete Fruit Recognition and calorie counter/Sprint1/web scraping1...	last week
refrence papers	Add files via upload	last week
sprint2	Add files via upload	3 weeks ago
sprint3/web_app	Add files via upload	last week
README.md	Create README.md	3 weeks ago
desktop.ini	Add files via upload	3 weeks ago
fruit360.zip	Add files via upload	last week

Fig 7.1 Git History of 3 Sprints

PROJECTS / Fruit Recognition and calorie counter / sprint3 / web_app /

sandra202002 Delete Fruit Recognition and calorie counter/sprint3/web_app/Mainpage.py ddbf143 · now History

Name	Last commit message	Last commit date
..		
Fruit_classification.py	Add files via upload	now
home_img.jpg	Add files via upload	last week
labels.txt	Add files via upload	last week
webscraping.py	Add files via upload	last week

Fig 7.2 Git History sprint 3

8.CONCLUSIONS

In conclusion, the Fruit Classification and Calorie Counter project has successfully met its objective of delivering a user-friendly tool for recognizing fruits and providing nutritional insights. By harnessing the power of the ResNet-50 pretrained model, the system excels at accurately identifying various fruit types from images, contributing to real-time dietary monitoring. The integration of web scraping through Selenium enables the retrieval of calorie information, enhancing the practical value of the application.

The project's technical achievements, including the seamless incorporation of image processing, machine learning, and a web interface using Streamlit, demonstrate its scalability and readiness for real-world use. The system's intuitive design ensures smooth user interaction, while its back-end robustness provides reliable and timely results.

The production-ready deployment, alongside real-time calorie tracking, positions this project as a significant contribution to the field of health monitoring and AI-driven nutritional tracking. Looking forward, the lessons learned from the development of this system will guide future improvements, including expanding the dataset, refining the prediction model, and integrating additional nutritional metrics. The Fruit Classification and Calorie Counter system, a successful blend of machine learning and practical application, is well-positioned to aid users in making informed dietary choices in today's health-conscious environment.

9.FUTURE WORK

- Expansion of Dataset: Incorporating a larger and more diverse dataset of fruits, including exotic varieties, to improve the model's ability to classify a wider range of fruits.
- Improved Accuracy with Advanced Models: Exploring more advanced deep learning architectures like EfficientNet or Vision Transformers (ViT), which could offer better performance and higher accuracy in fruit classification compared to ResNet-50.
- Nutritional Information Expansion: Extending the calorie counter feature to provide more detailed nutritional information such as macronutrient breakdown (carbohydrates, proteins, fats), vitamins, and minerals for each fruit.
- Mobile Application Integration: Developing a mobile application that leverages the current classification and calorie counting capabilities, making the system more accessible for users on the go.
- Real-Time Object Detection: Adding real-time object detection using YOLO or Faster R-CNN for fruit classification through a live camera feed, improving the app's usability in settings like grocery stores or kitchens.
- Personalized Recommendations: Implementing a recommendation system based on user preferences, suggesting healthier fruit alternatives or meal suggestions based on dietary goals like weight loss or muscle gain.
- Offline Functionality: Enabling offline capabilities by deploying the model on edge devices or mobile phones, allowing users to classify fruits and access calorie data without an internet connection.
- Multi-language Support: Adding multi-language support to reach a wider global audience and improve accessibility in non-English speaking regions.
- User Data and Analytics: Incorporating user data tracking to provide insights over time, such as how frequently a user consumes certain fruits, helping them track their diet patterns and make better health decisions.
- Sustainability and Local Produce Suggestions: Providing users with recommendations for locally sourced fruits or seasonal fruits, promoting sustainability and supporting local agriculture.

Each of these future works would contribute to making the system more robust, user-friendly, and widely applicable in the areas of nutrition, health, and personal well-being.

10 APPENDIX

10.1 Minimum Software Requirements:

To deploy and run the Fruit Classification and Calorie Counter system, the following minimum software requirements must be met:

- Operating System: Windows, Linux, Mac
- Python: Version 3.6 or above is required to execute the project code and its dependencies.
- Colab Notebook: If using Colab notebooks for development and testing, having Colab installed is recommended.
- Visual Studio Code v 1.71.2
- TensorFlow: Essential for building and deploying the deep learning model (ResNet-50) used for fruit classification.
- NumPy: Necessary for numerical operations, especially for handling image arrays.
- Pillow (PIL): Needed for image processing, including resizing and transformations (such as rotation and flipping) of fruit images.
- Selenium: Used for web scraping to fetch real-time calorie data from websites such as Nutritionix.
- Streamlit: The framework used to build the web application interface, which allows users to upload fruit images and view results.
- Chrome WebDriver: Necessary for automating web interactions using Selenium to fetch calorie information.
- Other Dependencies: Additional libraries and modules specified in the project code, including TensorFlow's preprocessing and image handling utilities.

10.2 Minimum Hardware Requirements:

The Fruit Classification and Calorie Counter system is moderately lightweight but involves computational processes, particularly during model training and image processing. The minimum hardware requirements to ensure smooth execution are as follows:

- **Processor (CPU):** A modern multi-core processor (e.g., Intel Core i5 or equivalent) is recommended to efficiently handle the computations involved in deep learning model inference and image transformations.

- **RAM:**A minimum of 8 GB RAM is recommended to manage the dataset and perform image processing and machine learning operations smoothly.
- **Storage:**Adequate storage space is required for storing datasets, model files, code, and additional resources. While the system itself doesn't demand extensive storage, having sufficient space for datasets (e.g., the Fruit-360 dataset) and model persistence is essential.
- **Internet Connection:**An internet connection is necessary for downloading datasets, libraries, and dependencies during the setup phase. It is also required for fetching real-time calorie information from external websites using Selenium.

11.REFERENCES

- 1) Tomar, Hanshu. "Multi-Class Image Classification of Fruits and Vegetables Using Transfer Learning Techniques." Master's dissertation, Journal of Physics: Conference Series, 2021.
- 2) Khatun, Mehenag, Forhad Ali, Nakib Aman Turzo, and Julker Nin. "Fruits Classification using Convolutional Neural Network." GRD Journals - Global Research and Development Journal for Engineering, 2020.
- 3)Gulzar, Yonis. "Fruit Image Classification Model Based on MobileNetV2 with Deep Transfer Learning Technique." Sustainability, 2023.
- 4)A, Chaitanya, Jayashree Shetty, and Priyamvada Chiplunkar. "Food Image Classification and Data Extraction Using Convolutional Neural Network and Web Crawlers." Procedia Computer Science, vol. 218, 2023, pp. 143-152.