

Multi-Class Classification of Guava Quality from video frames

A Major Project Submitted
in partial fulfillment of the Degree
In Master of Computer Application

Submitted By:

Sakshi Gheeyal

Ravi Kumar Raushan

Roll No: 40823210027

Roll No: 40823210033

Department of Computer Science

Department of Computer Science

MCA, 2025

MCA, 2025

Supervisor:

Dr. Atif Mahmood

Assistant Professor

Submitted to:



SRM
UNIVERSITY
DELHI-NCR, SONEPAT

SRM UNIVERSITY DELHI-NCR, SONEPAT HARYANA-131029

Department of Computer Science

May 2025

CERTIFICATE

This is to certify that the project titled “**Multi-Class Classification of Guava Quality from video frames**” submitted to **SRM University, Delhi-NCR, Sonapat, Haryana - 131029** in partial fulfilment of the requirement for the award of the degree of **Master of Computer Applications** is an original work carried out under the guidance of **Dr. Atif Mahmood** during the academic year **2024-2025**. The matter embodied in this project is a genuine work and has been submitted neither to this University nor to any other University for the fulfilment of the requirement of the course of study.

Date: _____

External Examiner’s Signature

Supervisor’s Signature
Supervisor
Dr. Atif Mahmood
Assistant Professor
Department of Computer Science
SRM University, Delhi-NCR, Sonapat

CANDIDATE'S DECLARATION

We hereby declare that the work which is being presented in the project titled “**Multi-Class Classification of Guava Quality from video frames**” submitted to **SRM University, Delhi-NCR, Sonapat, Haryana – 131029** in partial fulfilment of the requirement for the award of the degree of **Master of Computer Applications** is an original work carried out under the guidance of **Dr. Atif Mahmood** as Major Project in the 4th semester during the academic year **2024-2025**. The matter embodied in this project is a genuine work done by us and has been submitted neither to this University nor to any other University for the fulfilment of the requirement of the course of study.

Sakshi Gheeyal

Reg. No. 40823210027

Ravi Kumar Raushan

Reg. No. 40823210033

ACKNOWLEDGEMENT

Acknowledging the contributions of those who have helped us throughout the completion of this project is an honor. We would like to express our sincere gratitude to the following individuals and institutions for their unwavering support and guidance.

We would like to express our heartfelt gratitude to our project supervisor, Assistant Professor **Dr. Atif Mahmood**, whose expert guidance, constant encouragement, and care played a pivotal role in the successful completion of this project. We are truly thankful for your continuous support throughout the journey of this project.

Our sincere thanks is also extended to the Project Committee and SRM University, Delhi-NCR, Sonapat, Haryana - 131029 for providing us with the opportunity to work on the **"Multi-Class Classification of Guava Quality from video frames"** project. Without the resources, support, and suggestions from the university, this project would not have been possible. Your invaluable guidance has been instrumental in various stages of the project, and we are truly grateful for that.

Finally, we extend our thanks to everyone who has directly or indirectly contributed to this project. Special thanks to our parents and peers for their continued support and encouragement throughout the course of this project. Their patience, understanding, and faith in us have been a constant source of motivation.

We hereby affirm that this project is the result of our own hard work and dedication, and that it has not been copied or plagiarized in any form.

ABSTRACT

Ensuring availability of top-quality fruits has always been a continuous challenge faced by consumers and suppliers as traditional methods mainly rely on manual sorting or single side-view images. This study examines various deep learning models and ensemble learning approach to classify Guava into various categories based on its quality, fresh, mid-quality and rotten using video frames capturing 360° view enabling a more comprehensive approach. The frames were extracted and augmented from a collected dataset of 60 videos and then trained on nine pre-trained models. The top performing model on the augmented dataset was VGG16 with an accuracy of 96.78%. MobileNetV2, ResNet101 and ResNet50 were the second-best performing models with accuracy of 96.67%. To enhance the performance of models, ensemble learning with soft voting and feature fusion strategy was used. Soft voting ensemble model combined the strength of NASNetMobile, EfficientNetB0 and ResNet50 (NER) achieving an accuracy of 97.33% while hard voting ensemble model comprising of the same models secured an accuracy of 97.22%. Along with these two feature fusion ensemble models were also tested on the validation dataset, VRM (VGG16, ResNet50, MobileNetV2) and NER (NASNetMobile, EfficientNetB0 and ResNet50) which used feature-fusion ensemble achieved an accuracy of 96.44%. A simple web application using Flask was also developed to demonstrate the real time classification process of guava.

TABLE OF CONTENTS

ACKNOWLEDGEMENT	iii
ABSTARCT	iv
LIST OF FIGURES AND TABLES	vii
Chapter 1: Introduction.....	1
1.1 Background of the Study	1
1.2 Problem Statement	3
1.3 Research Objectives	4
1.4 Scope of the Study	4
1.5 Research Methodology Overview	5
1.6 Contribution to Sustainability Goals	5
Chapter 2: Literature Review.....	7
2.1 Related Work and Approaches	7
2.2 Research Gaps	8
Chapter 3: Research Methodology.....	10
3.1 Research Design	10
3.2 Data Collection Methods	11
3.3 Data Preprocessing Techniques	11
3.4 Tools and Technologies Used	12
3.4.1 Programming Languages	12
3.4.2 Image and Video Processing	16
3.4.3 Development Environment	16
3.4.4 Python Libraries	17
3.4.5 Data Visualization Libraries	18
3.5 Algorithms / Models Applied	19
3.5.1 Introduction to CNN	20
3.5.2 VGG16	21
3.5.3 MobileNetV2	24
3.5.4 DenseNet201	25
3.5.5 InceptionV3	28
3.5.6 EfficientNetB0	30
3.5.7 NASNetMobile	32
3.5.8 ResNet50 and ResNet101	34

3.5.9 Xception	38
3.5.10 Ensemble Learning	40
3.6 Evaluation Metrics	42
Chapter 4: System Workflow.....	43
4.1 System Architecture	43
4.2 Flow Diagram / Block Diagram	46
4.3 Module Description	47
4.3.1 Data Preparation and Frame Extraction	47
4.3.2 Model Training	51
4.3.3 Model Evaluation and Selection	57
4.3.4 Video processing and Frame prediction	57
4.3.5 Web Application	58
4.4 Implementation Details	58
4.4.1 Backend Functionality	60
4.5 Challenges Faced	64
Chapter 5: Results and Analysis.....	65
5.1 Results of Different Models / Techniques	65
5.2 Performance Comparison	67
5.3 Discussion	81
Chapter 6: Conclusion & Future Work.....	83
6.1 Conclusion	83
6.2 Future Work	84
REFERENCES.....	85

LIST OF FIGURES AND TABLES

List of Figures:

Figure 1: Original framework for classification of Guava into 3 classes: fresh, mid and rotten.....	10
Figure 2: Architecture of CNN.....	21
Figure 3: Architecture of VGG16 model.....	23
Figure 4: Architecture of MobileNetV2 model.....	25
Figure 5: Architecture of DenseNet201 model.....	27
Figure 6: Architecture of InceptionV3 model.....	30
Figure 7: Architecture of EfficientNetB0 model.....	32
Figure 8: Architecture of NASNetMobile model.....	34
Figure 9: Architecture of ResNet50 model.....	36
Figure 10: Architecture of ResNet101 model.....	38
Figure 11: Architecture of Xception model.....	40
Figure 12: Ensemble Model using soft voting.....	41
Figure 13: Ensemble Model using hard voting.....	42
Figure 14: System Architecture.....	43
Figure 15: Workflow Diagram.....	46
Figure 16: Home Page of Application.....	61
Figure 17: Uploading video/images.....	62
Figure 18: About Page.....	62
Figure 19: Contact Page.....	63
Figure 20: Prediction result summary.....	63
Figure 21: Prediction result Analysis by frame.....	64
Figure 22: (a) Confusion Matrix of VGG16 and (b) Training/Validation accuracy – VGG16.....	69
Figure 23: (a) Confusion Matrix of MobileNetV2 and (b) Training/Validation accuracy – MobileNetV2.....	70
Figure 24: (a) Confusion Matrix of DenseNet201 and (b) Training/Validation accuracy – DenseNet201.....	71
Figure 25: (a) Confusion Matrix of InceptionV3 and (b) Training/Validation accuracy – InceptionV3.....	72
Figure 26: (a) Confusion Matrix of EfficientNetB0 and (b)	

Training/Validation accuracy – EfficientNetB0.....	73
Figure 27: (a) Confusion Matrix of NASNetMobile and (b)	
Training/Validation accuracy – NASNetMobile.....	74
Figure 28: (a) Confusion Matrix of ResNet50 and (b) Training/Validation	
accuracy – ResNet50.....	75
Figure 29: (a) Confusion Matrix of ResNet101 and (b)	
Training/Validation accuracy – ResNet101.....	76
Figure 30: (a) Confusion Matrix of Xception and (b) Training/Validation	
accuracy – Xception.....	77
Figure 31: Confusion Matrix of (a) NER (Soft Voting) model and	
(b)NER (Hard Voting) model.....	78
Figure 32: (a) Confusion Matrix of NER (Feature Fusion) model and (b)	
Training/Validation accuracy – NER (Feature Fusion) model.....	79
Figure 33: (a) Confusion Matrix of VRM (Feature Fusion) model and (b)	
Training/Validation accuracy – VRM (Feature Fusion) model.....	80

List of Tables:

Table 1: Layers in Architecture of VGG16 model.	22
Table 2: Layers in Architecture of MobileNetV2 model.	24
Table 3: Layers in Architecture of DenseNet201 model.	26
Table 4: Layers in Architecture of InceptionV3 model.	28
Table 5: Layers in Architecture of EfficientNetB0 model.	31
Table 6: Layers in Architecture of NASNetMobile model.	33
Table 7: Layers in Architecture of ResNet50 model.	35
Table 8: Layers in Architecture of ResNet101 model.	36
Table 9: Layers in Architecture of Xception model.	39
Table 10: Validation accuracy for models trained on GPU per epoch.....	65
Table 11: Validation accuracy for models trained on CPU per epoch.....	66
Table 12: Validation accuracy of Ensemble Models.....	66
Table 13: Comparison of Precision, Recall and F1-score.....	68

CHAPTER I

INTRODUCTION

1.1 Background of the Study

Guava, common name *Psidium guajava*, is a tropical fruit belonging to the myrtle family (Myrtaceae). The common name is sometimes written incorrectly as *P. guava* or *P. guayaba* like the English name, comes from Spanish speaking tropical America, where the fruit is first said to be originated from. The genus *Psidium* comprises approximately 153 species of trees, including 20 species that produce edible fruits [1]. Most popular specie of Guava, Safeda is commonly grown in Allahabad, Uttar Pradesh, a state in India. Safeda is generally round, smooth skinned and white fleshed. Other popular species of guava include Chitidas, similar to Safeda but has red spots and sweeter pulp or Hafsi which is a red fleshed guava having round and smooth shape [2]. Guava is commonly known as “poor man’s apple” in India owing to its affordability and high nutritional value. It has high medical value possessing anti-viral, anti-inflammatory, anti-plaque, anti-mutagenic properties as well as rich in vitamins A, C, iron, phosphorus, calcium and minerals. Characteristically speaking, the fruit ranges from 3 cm to 6 cm in size for small to medium sized guavas. It is generally in green color but sometimes yellow in ripened condition [3] whereas the flesh color determined by the presence of carotenoids, lycopene and β -carotene can range from whitish yellow to pink or salmon red [6]. It is one of the most widely grown fruit across tropical and subtropical regions and India contributes around 45% to global guava production, the highest in the world. A high percentage of this number comes from the state of Uttar Pradesh contributing to 22.93% of the market share. The second and third largest producer of guava are Madhya Pradesh and Bihar standing at 16% and 10% of market share respectively [4]. Due to its extensive consumption in form of juices, jams and processed products the demand of fresh produce is often high. Poor quality harvest may result in financial losses along with loss of consumer trust and eventually a huge decline in customer retention. Hence, there is a growing need to automate fruit quality sorting to minimize errors and enhance productivity.

The most common methods to determine maturity levels in fruits include visual, physical, chemical, computational and physiological factors [5,6]. Along with these many quality attributes are also considered for maturity grading of fresh produce. These include

appearance, texture, flavour, nutritional and safety factors. Appearance attributes consider size, shape, color of fruits along with the presence and absence of decay and defects. Firmness, crispiness and juiciness are other important factors listed under Texture attribute. Another factor is safety which is used to evaluate whether a particular commodity is free from pesticide residue and certain fungi. The fruits and vegetables can be classified into two categories on the basis of their maturity level [5]. The first category includes fruits and vegetables that are harvested mostly when they reach a sufficient stage of development while being attached to the tree or plant, known as physiological maturity. The other one is called horticultural maturity, where harvesting is performed only when the commodity possesses the necessary characteristic desired by the customer. Hence maturity index is crucial in identifying the exact time of harvest. Maturity index can be defined as a measurement used to determine whether a particular fruit or vegetable is mature enough to be harvested or not. Various maturity indices have been defined for different fruits and vegetables. Overall, the evaluation of maturity indices can be performed using two approaches destructive and non-destructive [6]. Destructive approaches often include destructive analysis which ultimately harms the produce and make it unsuitable for further use while non-destructive approaches don't degrade the quality of the subject and mostly uses external factors to provide a maturity grade. The most apparent external factor in case of guava fruit is color. Guava is generally harvested when it has a dark green to greenish yellow color and takes almost 17 to 20 weeks to reach maturity. As the guava fruit matures it observes discoloration as result of chlorophyll degradation. The Directorate of Marketing and Inspection (DMI) divides guava into 6 categories on the basis of the weight [20] and size of the fruit whereas the Fruits and Vegetable grading and marking rules divides the fruit into three categories namely, Extra class, Class I and Class II [19]. However, most of these traditional fruit sorting methods often include manual sorting which is prone to human error and eye fatigue.

This ultimately led to the introduction of deep learning and CNN models in fruit classification tasks. But the goal is missed because the quality assessment is often performed on static images or single side view images. Now, it is important to note that single side view images shouldn't be used solely to detect the quality of fruit. Comprehensive quality assessment should include views from all angles, as a fruit that may look fresh from front may have hidden defects such as a rotten backside or a fungal

infection. These overlooked flaws make the fruit inedible, despite being classified as fresh using conventional methods. Therefore, it becomes essential to classify the fruit from all angles before assigning a particular grading to it.

1.2 Problem Statement

Although manual guava quality assessment is a common industry practice, it has significant limitations. Inspectors typically evaluate the quality of the fruit by looking at it from one surface; therefore, these assessments are objective, subject to bias, inconsistent, and prone to error. Environmental variations (e.g., lighting conditions), inspector fatigue, and subtle fruit defects render manual classification ineffective, particularly for volume sorting.

Also, all current automated systems for fruit classification in agricultural technology are based on static images at a single angle. While speed and consistency of can be achieved over an inspector with an automated quality assessment system, their assessments of guava quality will still be inaccurate since many defects are either not present or under the fruit's surface.

One common limitation amongst these systems is the inability of the system to provide products, including guavas, which entirely analyzes the surface(s) of the fruit. A guava may appear fresh when viewed from one surface and exhibit points of decay (e.g., bruising, fungal infection) in the view from the opposite side. Inverse classification of fruit has a detrimental effect on product quality resulting in significant consumer dissatisfaction, economic loss for producers, and wasted resources.

As a result, there is a strong demand for a quality assessment process that is both robust and scalable. A tool which can assess all sides of the fruit, combine visual assessment with deep learning capabilities, and provides rapid assessments of fruit quality would greatly enhance the repeatability and efficiency of guava evaluations. This research addresses these issues by presenting a deep learning quality classification model based on 360-degree video frames of guavas to ensure full-surface assessment and reliable quality classification performance.

1.3 Research Objectives

This research aims to establish an integrated system for automated guava quality classification, utilizing deep learning algorithms on 360-degree video data. This study aims to achieve the following specific aims:

- i. To obtain 360-degree video data of guava fruits at different quality classes (fresh, middle-quality, rotten) to ensure a comprehensive visual data collection.
- ii. To extract images from video data and apply image augmentations (rotation, flip, scale, etc.) to create a balanced dataset across visual features of guavas.
- iii. To develop and train a set of deep learning models including CNN architectures and pre-trained networks to perform quality classification with accurate predictions and minimal false prediction.
- iv. To apply ensemble learning methods including soft voting, hard voting, and feature fusion, to combine the collection of models so that they collectively produce better predictions.
- v. To compare model performance across different environments, particularly CPU and GPU (T4) runtimes available on Google Colaboratory to determine an appropriate configuration for training and inference.
- vi. To design and deploy a simple web application for users to upload videos or images of guavas, which will provide predictions on their quality class in real-time.

By achieving the above objectives, the research will contribute to the development of intelligent agricultural systems that are impactful, effective, and feasible for real-world implementation.

1.4 Scope of the Study

This study encompasses the entire pipeline from data collection to real-time deployment. It involves capturing 360° video data of guava fruits, frame extraction, data augmentation, deep learning model training and evaluation, and finally, deployment in a web-based interface. The system is designed to assist farmers, fruit vendors, and quality inspectors in performing fast and objective assessments. While the current study focuses on guava, the methodology can be extended to other fruits and integrated with real-time industrial sorting systems in the future.

1.5 Research Methodology Overview

This study addresses this specific problem by utilizing videos of guavas captured from all angles (360° view) to classify the subject into three categories namely, fresh-quality, mid-quality and rotten-quality. Furthermore, the working of models was tested in two different runtime environments on Google Colab: CPU and T4 GPU. Overall, T4 GPU showed better performance and accuracy over CPU primarily because GPUs are able to provide a computing environment without interruptions owing to the large-scale computation units and a relatively smaller portion of storage units and control unit [7].

In this research, we have utilized video dataset to capture quality grading of guava from all sides ensuring a systematic approach. Various CNN paradigms including VGG16 [6], MobileNetV2 [7], DenseNet201 [8], InceptionV3 [9], EfficientNetB0 [10], NASNetMobile [11], ResNet50 [12], ResNet101 [12] and Xception [13] were trained and tested on the augmented frames generated from the original dataset to perform a detailed evaluation. Among the various frameworks implemented, VGG16 achieved an accuracy of 96.78%. The second-best performing models were MobileNetV2, ResNet50 and ResNet101 with an accuracy of 96.67% on the validation set. Additionally, ensemble learning techniques including soft voting, hard voting and feature concatenation were explored to find an optimal ensemble having better accuracy and performance.

1.6 Contribution to Sustainability Goals

This project is not just about a technological innovation in agriculture; it also addresses various UN Sustainable Development Goals (UN SDGs) that promote responsible production, food quality monitoring, and digital inclusion in agricultural practices.

Relevant Sustainability Goals:

SDG 2: Zero Hunger

The system will classify the quality of guava quickly and accurately so that we can limit post-harvest loss, ensuring the only fruits supplied to consumers are market-ready. By providing the guava supply chain with better-quality fruit, people will have better access to food while limiting waste.

SDG 12: Responsible Consumption and Production

The project will provide support by helping farmers and vendors assess quality in a more reasonable way, and this, in turn, will help them avoid convenience decisions that can be wasteful. It will also help the system become more efficient with grading, and it will save farmers and vendors time with data-driven decisions regarding production and distribution.

SDG 9: Industry, Innovation, and Infrastructure

The use of AI and deep learning in agriculture demonstrates how innovation can be applied to classical industries. The project demonstrates how we can perform smart agriculture using technologies that are affordable and accessible- especially to smallholder farmers.

Overall Impact:

The overall impact of the project is how an AI-driven quality control mechanism in agriculture can lead to changes (remote or otherwise) that are more sustainable over the long term, by improving efficiencies and reducing waste while making decisions that would ultimately benefit the producers. This project aligns with these global commitments to improve food systems that are both resilient and equitable.

CHAPTER II

LITERATURE REVIEW

2.1 Related Works and Approaches

Crop maturity grading is a crucial process in the crop management process. Recent years have witnessed significant developments in the field of ML and DL leveraging them to classify fruit based on maturity grading [21]. Traditionally, it was performed manually, mainly focusing on visual inspection and non-automatic grading. But now there is a growing body of literature that recognizes implementing ML and DL technologies not only improves accuracy but also eradicates human-induced bias. A study conducted by A. Mahmood et.al. 2022 [22] explored two CNN paradigms i.e., AlexNet and VGG16 utilizing transfer learning to classify jujube fruits based on their maturity level achieving a high accuracy of 98.26% and 99.17% respectively. The author further enhanced their work by including more categories [23] and a novel approach called C-net to classify jujube on the basis of maturity level [24]. Similar trends were observed by works of other authors including T. Khatun et.al. 2023 [25] supported the ideology by utilizing Residual Network ResNet50 framework by identifying the ripeness and quality of dragon fruits. The model demonstrated a strong capability by achieving 90% accuracy in distinguishing mature and immature dragon fruits as well as 98% accuracy in case of fresh and damaged dragon fruit. While, for custard apple G.C. Wakchaure et.al. 2023 [26] proposed an image processing-based maturity stages detection prototype device. It deployed two algorithms K-Means Clustering and SVM on Raspberry Pi to achieve 100% accuracy on the captured images. Their methodology also showed future scope for other fruits with slight modifications.

Various authors have utilized deep learning for image processing and classification for quality classification of guava. J. Yeshwanth et.al. 2023 [27] emphasized use of computer vision system for automating the visual grading process of Taiwan guava. The system achieved an accuracy of 89% on static images of guava. Similarly, A. Hayat et.al. 2024 [28] and M.M. Ali et.al 2023 [29] proposed deep learning architectures called FruitVision and FruizNet respectively. Both of these models worked on different datasets classifying 8 different types of fruit including guava. FruitVision achieved an accuracy of 99.24% in case of guava and mainly categorized the fruits into three categories, good quality, bad

quality and mixed quality. While FruizNet achieved 97.78% and 96.7% precision for fresh and rotten quality guava respectively. Some studies have also focused on various aspects for classification of fruits including color, texture, and shape. A study conducted by A. Bhatt et.al. 2024 [30] focuses on categorizing fruits including guava based on similar color and texture using deep transfer learning. They integrated various models including DenseNet, MobileNet and EfficientNet in their research and found that MobileNetV1 with Adam optimizer outperforms other models. Similarly, T. Choudhary et.al. 2022 [31] demonstrated the scope of deep learning as a potential tool for assessing quality of fruits. They graded the fruit on a scale of 1 to 5 based on the 3 aspects namely, color, shape and texture. I. D. Apostolopoulos et.al. 2023 [32] introduced the use of vision transformers built and trained on mixture of various fruit datasets and can distinguish between good and rotten fruit image based on the visual appearance and not predefined quality attributes.

In conclusion, many studies have acknowledged the potential of deep learning networks in quality grading of fruits. However, the existing work mainly focuses only on static images which is not optimal. Single side view is not a comprehensive way to depict the overall quality of any fruit. To ensure accurate results, the subject must be analyzed from various angles and then graded. Therefore, in this research we have utilized a video dataset comprising of 360° videos of guavas ensuring view of the fruit from all angles before model training. The frames are then extracted from the video dataset to classify the guava fruit into three grades: fresh, mid and rotten quality.

2.2 Research Gaps

Despite significant advancements in the application of Machine Learning (ML) and Deep Learning (DL) techniques for fruit maturity and quality grading, several research gaps still exist:

1. **Over-Reliance on Static Images:** A major limitation in existing studies is the dependence on static images for fruit quality classification. Most models are trained using single-view images, which fail to capture the complete visual profile of the fruit. Since quality assessment often depends on multiple visual attributes

like shape, color consistency, and surface defects, a single image from one angle may result in incomplete or inaccurate grading.

2. **Lack of 360° Visual Analysis:** Current approaches do not consider comprehensive 360° views of the fruit. Real-world fruit inspection, whether manual or automated, requires examination from various angles to detect hidden defects or inconsistencies. The absence of such holistic analysis in prior research limits the model's reliability and robustness in practical scenarios.
3. **Limited Use of Video-Based Data:** While some research has successfully implemented DL models like CNNs, ResNet, and custom architectures (e.g., FruitVision, FruizNet) for fruit classification, they predominantly rely on image data. Very few studies explore the potential of video input for grading, which can simulate human-like inspection and provide richer, more informative data for model training.
4. **Minimal Application to Guava Quality Grading:** Although various fruits like jujube, dragon fruit, and custard apple have been studied extensively, guava remains underexplored. A few works such as those by Yeshwanth et al. (2023) and Hayat et al. (2024) have made initial contributions, but the performance on guava is still limited by the use of static imagery and small datasets. There remains substantial room for improving guava quality classification using more advanced and dynamic data sources.
5. **Underutilization of Frame-Based Aggregation Techniques:** In video-based systems, frame extraction and aggregation strategies (e.g., majority voting across frames) remain underexplored. Techniques that average predictions across multiple frames or apply temporal context for better inference could significantly enhance prediction accuracy but are seldom applied.
6. **Lack of Unified Framework Integrating End-to-End Video Processing:** Existing studies either stop at classification or separate the preprocessing and model inference stages. Very few works present an end-to-end automated framework that integrates video upload, frame extraction, preprocessing, classification, and result visualization, especially in a web-based deployment context.

CHAPTER III

METHODOLOGY

This chapter briefs about the methodology adopted to achieve objectives of this research. It presents step-by-step description of the processes involved, following from the Data collection, Preprocessing, Model training and Evaluation. The chapter opens up by providing a detailed overview about the creation of video dataset to frame extraction and moves down to other subsections. Additionally, it also outlines how a simple and interactive website was developed using flask to allow users to upload image/videos or access their device camera to predict the quality of guava in real time scenarios.

3.1 Research Design

This research follows an experimental design to classify guava into three categories based on the quality assessment i.e., fresh quality, mid quality and rotten quality. The quality assessment was performed on frames extracted from 360° videos showing the fruit from all angles. The main process comprises the following stages: Data Collection, Data Preprocessing, Model Training and Evaluation.

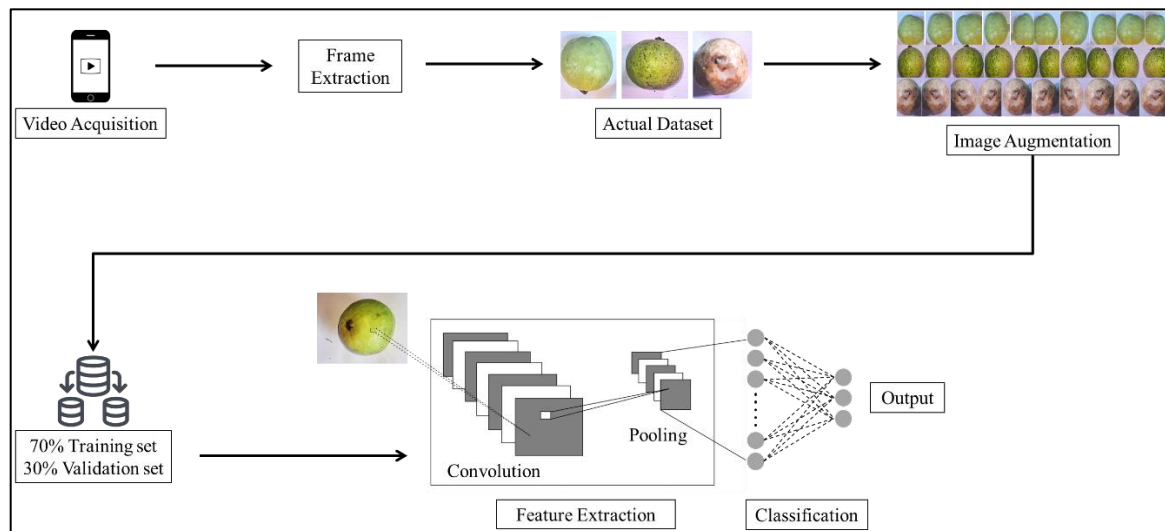


Fig 1. Original framework for classification of Guava into 3 classes: fresh, mid and rotten.

The illustration above shows the fundamental research design and methodology undertaken for the classification of guava quality into three classes namely; Fresh, Mid and Rotten. It begins with a video recording which involves a mobile device or a camera (Step 1).

In Step 2, frames are captured from the supplied video, and further processes such as rotation, flipping, zooming, and brightness adjustments are implemented to improve the overall dataset's augmentation, diversity, and robustness.

Actual and augmented data is saved in Step 3 and then split in Training Set (70%) and Validation Set (30%) in Step 4. This split assists in enhancing the model generalization as well as averting overfitting.

In the last step, images are processed by Convolutional Neural Networks (CNNs) where features are extracted in convolution and pooling layers. Other than that, the model was able to classify the guava as 'Fresh', 'Mid' or 'Rotten' based on predetermined features.

3.2 Data Collection Methods

The video dataset of guava was collected from the fruit vendors at Keshavpur Vegetable Market and Azadpur Fruit Market, Delhi. Each guava was placed on a white paper and videos were captured at 30 FPS(frame per seconds) in 1080P resolution using phone, set at a distance of 20 cm to record subject from all angles ensuring a 360° view. A total of 60 video samples were collected comprising of 22 fresh-quality videos, 26 mid-quality videos and 12 rotten-quality videos. To ensure diversity in frames, a total of 5 evenly spaced indexed frames were extracted from each video using OpenCV library, resulting in a total of 110 fresh, 130 mid-quality and 60 rotten guava frames. Hence, the original non-augmented dataset comprised of 300 images.

3.3 Data Preprocessing Techniques

Each of extracted frames from the original dataset were subjected to various augmentation to increase the dataset size and reduce overfitting. A total of six augmentations were applied consisting of rotation, width and height shifting, shearing, zooming, horizontal flipping and brightness enhancement to generate a total of 3000 augmented frames.

1. rotation_range=20: Randomly rotates images up to 20 degrees.

2. `width_shift_range=0.1` and `height_shift_range=0.1`: Shift images horizontally and vertically by up to 10% of the total width and height leading to shifting in position.
3. `shear_range=0.1`: Geometric transformation that slants the shape of an image along one axis, usually the x-axis or y-axis by 10% of the angle.
4. `zoom_range=0.2`: Randomly zooms into images by up to 20%
5. `horizontal_flip=True`: Enables random horizontal flipping
6. `fill_mode='nearest'`: Fills empty pixels left by transformations using the nearest pixel values.
7. `brightness_range=[1.2, 1.2]`: Uniformly increases the brightness of images by 20%.

All augmented frames were resized to 224x224 pixels to ensure uniformity across training. The training dataset and validation dataset split was 70% and 30% respectively. Additionally, different 'preprocess_input' function was used according to different pre trained models to ensure that the input data was in correct format.

3.4 Tools and Technologies Used

3.4.1 Programming Languages:

3.4.1.1 Python

The primary language for backend development (Flask, Deep learning models, data processing). Python is a popular programming language. It was created by Guido van Rossum, and released in 1991. Use cases of python in this project:-

1) Flask:-

Flask is a lightweight and user-friendly web framework built with Python. It is intended to make it easier to build web-based applications by providing the basic building blocks and features you need with low overhead. In this project, Flask will mainly provide the backend environment needed to support the integration of deep learning models and data processing tasks.

Flask allows you to organize your application logic in a well-structured and modular way. You can manage the routing to address various functionalities for Flask, and organize them in a way that will keep the backend clean and expandable. Its

minimalist design lets the developer have the power over the components they want to use and is a perfect solution for a research-oriented project like this.

Because it is based on Python, Flask makes it easy to integrate into data processing scripts or machine learning modules and use them in both the backend and front-end. These capabilities allow to easily execute useful tasks, such as loading models, controlling the flow of data, and managing the backend capabilities.

2) Deep Learning Models:-

In short, deep learning is a type of machine learning utilizing neural networks with multiple layers of abstraction that can learn latent structure from data. Python is currently the most widely used programming language for deep learning due to its relatively simple syntax, large functionality via many libraries, and large and active community. TensorFlow, Keras, and PyTorch are examples of libraries that provide high-level APIs to make building, training, and deploying a deep neural network very easy.

Deep learning has achieved state-of-the-art results in many fields, including image recognition, natural language processing, medical diagnosis, and autonomous systems. Because of the syntax of Python, it is easy for researchers and developers to experiment with many different model architectures (e.g. Convolutional Neural Networks [CNNs], Recurrent Neural Networks (RNNs), and Transformers) with only a few lines of code.

In the project described, deep learning models will be used to classify guava quality from visual inputs. Using Python deep learning libraries, these models will be trained quickly and efficiently using image data by finding differences in quality levels. Since it is simple to use Python, it also enables rapid prototyping, tuning the model, and optimizing performance.

3) Data Processing:-

Processing the data is one of the most important steps in your overall machine learning or data science "pipeline." Data Processing refers to the steps to clean, transform, and structure 'raw data' for analysis or model training. Python is a powerful language in this space, with many options to handle data, as well as the ability to build on existing

capabilities. The libraries of NumPy, Pandas, and OpenCV provide many tools for data manipulation and data processing in Python.

NumPy, This library provides unique tools for working with numerous, large arrays and matrices of numerical data; Pandas provides easy to understand data structures, specifically DataFrames, upon which you can structure your well-defined dataset from; OpenCV is a widely known library for image processing, resizing images, filtering images, creating and detecting features in images, etc. Various types of data such as images, text, and numbers can be easily handled.

Good data processing generally results in a better quality of input data, therefore impacting the performance of your machine learning and deep learning models. Practical techniques for data processing include data augmentation, normalization, and dimensionality reduction, all of which are important aspects of developing successful models. Data processing for this project includes generating the frames from 360° videos, organizing the raw frames into labeled datasets, and transforming the raw forms into datasets that will ultimately lead to better model training. Python allows the automation and combination of data processing tasks (in our case, many transformations workflows), reducing manual work and providing consistent results across a explicitly-structured dataset.

3.4.1.2 HTML/CSS

The main functionality of HTML/CSS is for structuring and styling the frontend web application. HTML (HyperText Markup Language) and CSS (Cascading Style Sheets) are the basic technologies for creating and designing web interfaces. HTML is responsible for the content of web pages, whereas CSS governs its appearance, including layout, colour, typography, and responsiveness.

Generally, HTML provides the semantics for grouping content into headings, paragraphs, forms, buttons, and media, to make web content organized, readable, and accessible. CSS enable users to be a maintainable and appealing media using design, so web interfaces may look good and perform similarly on any device or screen size.

For this project, we'll be using HTML and CSS to design the front-end interface that lets people interact with the system. HTML governs the inclusion of layout, the arrangement of elements on the page, including sections for image upload and model results. CSS will provide styling choices such as colour, contrast, typography, space, and patterns to keep

the interface clear and usable. This will bring together the use of HTML and CSS so that the front end will act a suitable media to present results from the processes that happen via the back end (e.g., predictions from models, data visualizations).

In addition, HTML and CSS allow the frontend to use little disk space resulting in fast load times which are very important considerations when considering media-heavy formats and rich-content (i.e. images or video frames). Their ease of use and browser compatibility make them great options for developing user interfaces both in research and production.

3.4.1.3 JavaScript

JavaScript is a high-level, interpreted programming language that is essential for enhancing interactivity and dynamic behavior in web applications. It operates on the client side (within the user's browser) and enables real-time interaction without requiring constant communication with the server. This makes web pages more responsive and user-friendly.

In general, JavaScript is widely used to handle events such as button clicks, form submissions, dynamic content updates, and interactive UI elements. It supports integration with HTML and CSS to create seamless and intuitive user experiences. Additionally, modern JavaScript is supported by powerful libraries and frameworks such as jQuery, React, and Vue.js, which simplify complex functionalities and improve development efficiency.

In this project, JavaScript is utilized to enhance the usability of the frontend interface. It manages interactions like uploading files (e.g., images or video frames), displaying loading animations, dynamically updating model results, and improving responsiveness. JavaScript ensures that users receive immediate feedback and that the interface remains smooth and interactive throughout the session.

By leveraging JavaScript, the frontend is not only visually structured (with HTML and CSS) but also functionally dynamic, allowing for a more engaging and efficient user experience. This is particularly important when integrating backend responses, such as displaying prediction outcomes from deep learning models.

3.4.2 Image and Video Processing:

3.4.2.1 OpenCV (cv2)

OpenCV (Open Source Computer Vision Library) is an open-source library software for computer vision and machine learning. It has various capabilities that can be used in real time to process images and videos. OpenCV is written in C++ with wrappers to support Python. OpenCV is suitable for multiple application areas such as education, industry, and research because of its speed, versatility, and amount of usage options.

OpenCV is useful in various ways. Image filtering, object detection, feature extraction, motion analysis, and image transformations are all operations that OpenCV can perform. OpenCV utilizes computing technologies that allow it to navigate and accomplish several file formats. It can also connect with images and video streams, which would be beneficial for multimedia data.

In this project, OpenCV (cv2 module in Python) was the library most frequently used for video processing and frame extraction corresponding to the 360° video. OpenCV has video capture and frame reading functions, which break down the 360° video into pictures. The extracted frames become the image input dataset for the deep learning model of guava quality classification.

OpenCV also provides other preprocessing tasks such as resize, convert to grayscale, image enhancement, and augmentations. These actions improve the standardization of the input data and aids in the performance of the model. The fast-processing time as well as the strong overall functionality of OpenCV allows it to use effectively in dealing with large quantities of visual data.

3.4.3 Development Environment:

3.4.3.1 Google Colaboratory (Colab)

Google Colaboratory, typically referred to as Google Colab, is a free, online development environment for writing Python code and executing code from a web browser. It is commonly used by researchers, students, and data scientists due to its connectivity to Google Drive and ability to support machine learning and deep learning workflows.

Colab is a type of Jupyter Notebook and provides free access to high-performance hardware accelerators such as GPUs (Graphics Processing Units) and TPUs (Tensor Processing Units) that are standard to training deep learning models efficiently. The advantage of Colab is that code can be executed with high-performance hardware without needing local setup or specialized hardware.

The project makes use of Google Colab as the main development environment to train, test, and experiment with models. For deep learning model building, Colab promotes rapid prototyping and interactive data analysis, as well as visualizing results. Since Colab is physically stored in the cloud and connectivity to Google Drive is utilized, models and code can be shared and collaborated on, while all work is accessible real-time.

Colab additionally provides support for using external Python packages, connectivity with GitHub repositories, and connection with Google Drive to collect and store datasets and models. Overall, Colab is simple to use, easy to access, and great high-performance computing resources, making it an excellent environment for machine learning projects.

3.4.4 Python Libraries:

Python makes itself an excellent choice for machine learning, data analysis, and deep learning projects; the specialized libraries used in this project serve to make the implementation, training, evaluation, and analysis simple.

3.4.4.1 scikit-learn

scikit-learn is one of the most popular machine learning libraries, and includes efficient tools for data mining and data analysis. It provides a number of different algorithms for classification, regression, clustering and dimension reduction. In this project, scikit-learn is mostly used for model evaluation. This library also leverages a number of utility functions for performance metrics including classification report, confusion matrix, accuracy, precision, recall, and F1 score, used for assessing the efficacy of the trained deep learning models against both the training and testing datasets.

3.4.4.2 TensorFlow/Keras

TensorFlow is an open-source depth learning framework created by Google, while Keras is a high-level API that extends TensorFlow for neural network creation and training.

Combining TensorFlow and Keras provides developers a powerful and flexible environment to develop depth learning models. In this project, TensorFlow/Keras is used to create, train, and optimize pretrained convolutional neural networks (CNN) for guava quality classification. TensorFlow/Keras supports GPU processing, model saving and loading, as well as enhancing model training and optimization through callbacks and data generators, among other features.

3.4.4.3 Pandas

Pandas is a popular and powerful Python library for data manipulation and analysis. It provides data structures (e.g., DataFrame), allowing effective handling of structured data. In this project, Pandas is used to manage metadata, organize image/frame-level information, and assist in preprocessing such as label encoding and dataset splitting. Pandas' flexibility and ease-of-use allow simple exploration and handling of large datasets.

3.4.4.4 NumPy

NumPy (Numerical Python) is the fundamental package for scientific computing with Python. It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays. NumPy has a significant role in this project as it is used for numerical operations, such as normalizing image pixel values, reshaping arrays to prepare for input to the model, and vectorized calculation. NumPy's performance and interoperability with other libraries written in Python make it vital in the workflows for preprocessing data and building models.

3.4.5 Data visualization libraries:

Data visualization is important to both understanding and interpreting the outcomes of machine learning and deep learning techniques. Visualization aids researchers and developers to convey insights, detect trends, diagnose reasons for performance changes, and direct improving model design and training.

3.4.5.1 Matplotlib

Matplotlib is one of the oldest and most flexible plotting libraries in Python and supports many different plot types, from line and pie charts to complex 3d plots, with full control

for each plotting detail. This is valuable for deep analysis to track how model performance changes on the validation set over time. During the development of this project, Matplotlib was utilized to graphically visualize training and validation metrics for accuracy, loss, precision, and recall values across epochs. The visualizations help monitor for overfitting or underfitting during training and understanding how the model is learning. Matplotlib can also export plots in many different formats (PNG, SVG, PDF) to assist with documentation and reporting of experimental results.

3.4.5.2 Seaborn

Seaborn adds a user-friendly layer on top of Matplotlib and features better looking styles by default. It is focused on statistical data visualization, allowing complex plots made simple such as heatmaps, pair plots, violin plots, and categorical plots. Seaborn is useful for creating confusion matrices, which are useful in classification problems. Confusion matrices display the predicted outcomes of your model against the true labels of test data. Seaborn generates heatmaps that effectively convey where your model is performing adequately, as well as where it is more inadequate meaning misclassification of test data points, which can then be thoroughly analyzed. It is effective because it natively integrates with Pandas DataFrames so that you can directly use tabular data without any additional implementation of objects. This is useful for exploratory data analysis and presenting your results at the end.

3.5 Algorithms / Models Applied

For this research, various CNN models were trained and tested on the augmented frames including VGG16, MobileNetV2, DenseNet201, InceptionV3, EfficientNetB0, NASNetMobile, ResNet50, ResNet101, and Xception. These models were selected based on their proven effectiveness in classification tasks and their ability to generalize.

To further boost performance, ensemble approaches were used namely, Voting-based and Concatenation-based. The voting-based approach combined NASNetMobile, ResNet50, and EfficientNetB0 using both soft and hard voting. In addition, concatenation-based approach was implemented on two ensemble models: one used NASNetMobile, ResNet50, and EfficientNetB0 and the other used VGG16, ResNet50, and MobileNetV2.

Transfer learning was applied on several pre-trained convolutional neural network architectures including VGG16, MobileNetV2, DenseNet201, InceptionV3, EfficientNetB0, NASNetMobile, ResNet50, ResNet101, and Xception. These models were selected based on their proven effectiveness in image classification tasks and their ability to generalize across varying image conditions.

3.5.1 Introduction to CNN

Convolutional Neural Network (CNN) is an updated version of artificial neural networks (ANN). These networks are unique because they are designed to extract features from grid like matrix data. CNNs were created to analyze visual data sets like images or videos when we need to locate a data pattern. CNNs are popular in computer vision applications for analyzing visual data.

CNNs consists of various layers: input layer, Convolutional layer, pooling layers and fully connected layers. CNNs image classification function takes an input image to processed into used for input image classification into various categories(Eg., Dog, Cat, Tiger, Lion). When a computer reads an image, the computer is looking at an array of pixels based on the image's resolution. Depending on the resolution of the image, the computer interprets the input image as $h \times w \times d$ (h = Height, w = Width, d = Dimension). For example, an image could be a $6 \times 6 \times 3$ array of matrix of RGB (3 refers to RGB values) and a $4 \times 4 \times 1$ array of matrix of grayscale.

Working of CNNs

A typical CNN architecture consists of the following key layers:

1. Input Layer:

This layer receives the raw pixel data of an image. For instance, a color image of size 128×128 would have dimensions of $128 \times 128 \times 3$ (width, height, and color channels).

2. Convolutional Layer:

This is the core building block of a CNN. It applies a set of learnable filters (kernels) that slide across the input image to produce feature maps. Each filter detects specific patterns such as edges, textures, or shapes.

3. Activation Function (ReLU):

The Rectified Linear Unit (ReLU) is typically used to introduce non-linearity into the model. It replaces all negative pixel values in the feature map with zero, helping the network learn complex patterns.

4. Pooling Layer:

Pooling (often max pooling) is used to downsample the spatial dimensions of the feature maps. This reduces computational complexity and helps in achieving translation invariance.

5. Fully Connected Layer (Dense Layer):

After several convolutional and pooling layers, the high-level reasoning in the network is performed via fully connected layers. These layers interpret the features extracted by the convolutional layers and perform classification or regression tasks.

6. Output Layer:

The final layer typically uses a softmax activation function (for classification tasks) to produce a probability distribution over the classes.

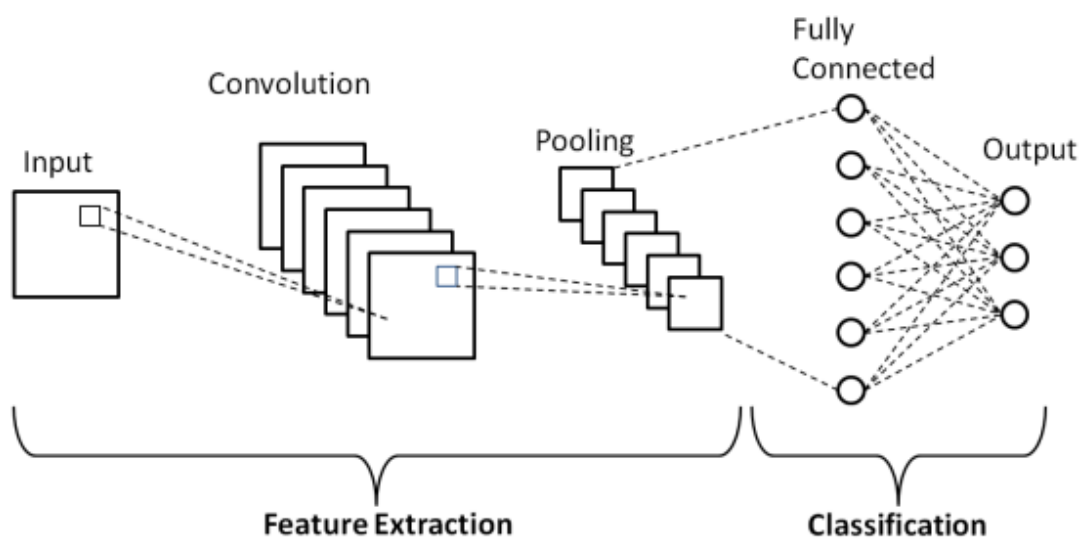


Fig. 2 Architecture of CNN

3.5.2 VGG16

VGG16 is an object detection and classification framework introduced by Karen Simonyan et.al. at the University of Oxford through the ImageNet Challenge 2014 [6]. The model originally has 21 layers: 13 convolutional layers, 5 Max pooling layers and 3 dense layers but only 16 layers have learnable parameters, hence the name VGG16. It is able to classify 1000 images belonging to 1000 different categories with an accuracy of

92.7%. The architectural model of VGG16 takes an input of size $224 \times 224 \times 3$ where the dimensions 224×224 represent the height and width of the input image in pixels while 3 represents the RGB color channels. All of the convolution layers use a 3×3 filter with stride 1 and always follow the same padding on the other hand, the max pooling layers possess a 2×2 filter with stride 2. In the original model, among the three fully connected layers, the first two possess 4096 neurons separately while the third layer performed the ILSVRC classification for 1000 classes and thus it possessed 1000 neurons, one for each class. In our case this final layer was modified and downsized to adapt to 3 neurons representing three different classes. Additionally, the preprocess input function was called to transform the RGB inputs to BGR format and zero-center each color channel with respect to the ImageNet dataset without scaling. The final architecture comprised of 14.8 million parameters corresponding to a model size of 56.64 Megabytes.

Table 1. Layers in Architecture of VGG16 model.

Layer No.	Type	Details
1	Conv2D	64 filters, 3x3, ReLU
2	Conv2D	64 filters, 3x3, ReLU
3	MaxPooling2D	2x2 stride 2
4	Conv2D	128 filters, 3x3, ReLU
5	Conv2D	128 filters, 3x3, ReLU
6	MaxPooling2D	2x2 stride 2
7	Conv2D	256 filters, 3x3, ReLU
8	Conv2D	256 filters, 3x3, ReLU
9	Conv2D	256 filters, 3x3, ReLU
10	MaxPooling2D	2x2 stride 2
11	Conv2D	512 filters, 3x3, ReLU
12	Conv2D	512 filters, 3x3, ReLU
13	Conv2D	512 filters, 3x3, ReLU

14	MaxPooling2D	2x2 stride 2
15	Conv2D	512 filters, 3x3, ReLU
16	Conv2D	512 filters, 3x3, ReLU
17	Conv2D	512 filters, 3x3, ReLU
18	MaxPooling2D	2x2 stride 2
19	Flatten	—
20	Dense	4096 neurons, ReLU
21	Dense	4096 neurons, ReLU
22	Dense	1000 neurons, Softmax (for ImageNet)

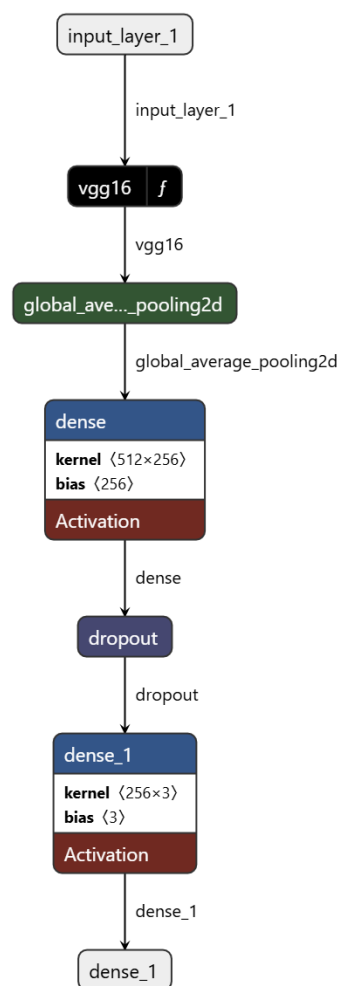


Fig 3. Architecture of VGG16 model

3.5.2 MobileNetV2

MobileNetV2, a mobile architecture was developed by researchers at Google who introduced a novel layer module consisting of an inverted residual with linear bottleneck [7]. MobileNetV2 model consists of two types of blocks, a Residual Block with stride 1 and another block with stride 2 for downsizing. Both the layers possess 3 different layers where the first layer is an expansion convolution layer with a 1×1 filter and ReLU6 activation. The second layer is called depthwise convolution with a filter of size 3×3 and the third layer is a 1×1 projection convolution layer but doesn't use any activation function hence called linear bottleneck. The architectural model of MobileNetV2 has 17 units of bottleneck with 3 layer each. Additionally, it has one initial convolution layer that accepts input in format of $224 \times 224 \times 3$ and one final 1×1 convolution layer before pooling. In our case, this final layer was modified in accordance to cater to 3 classes. Just like VGG16, we have called preprocess function in case of MobileNetV2 also to scale the input pixels between -1 and +1.

Table 2. Layers in Architecture of MobileNetV2 model.

Stage	Operator	t (expansion factor)	c (output channels)	n (repetition s)	s (stride)	Output Size
1	Conv2D 3×3	-	32	1	2	$112 \times 112 \times 3$ 2
2	Bottleneck (dw, pw- linear)	1	16	1	1	$112 \times 112 \times 1$ 6
3	Bottleneck	6	24	2	2	$56 \times 56 \times 24$
4	Bottleneck	6	32	3	2	$28 \times 28 \times 32$
5	Bottleneck	6	64	4	2	$14 \times 14 \times 64$
6	Bottleneck	6	96	3	1	$14 \times 14 \times 96$
7	Bottleneck	6	160	3	2	$7 \times 7 \times 160$

8	Bottleneck	6	320	1	1	$7 \times 7 \times 320$
9	Conv2D 1×1	-	1280	1	1	$7 \times 7 \times 1280$

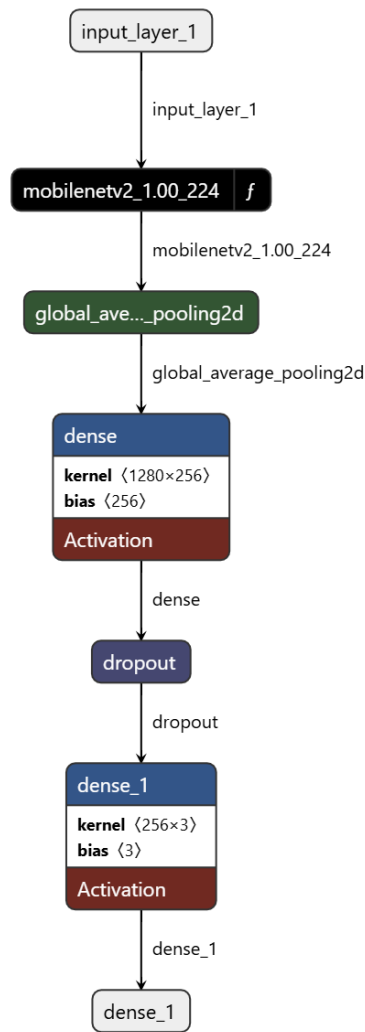


Fig 4. Architecture of MobileNetV2 model

3.5.3 DenseNet201

DenseNet201 is another convolution neural network architecture that was introduced by Gao Huang et.al. in 2017 [8]. It has a unique connectivity pattern where each block takes output as input from previous block and outputs from all previous blocks also, creating a dense connectivity network. The structure of the DenseNet201 model comprises of 201 layers where 196 convolution layers resides in 4 dense blocks. A transition layer made up of 1×1 convolution layer and 2×2 average pooling layer with stride 2 is added between

each dense block to reduce spatial dimensionality and number of feature maps. The denseNet201 architecture accepts an image of input size $224 \times 224 \times 3$. The final classification layer was modified to accommodate 3 classes. Additionally, preprocess function was used such that all the input pixels are scaled between 0 to 1 and each channel is normalized. DenseNet201 is a large model and therefore, has a high number of training parameters, in our case which was 18.8 million trainable parameters.

Table 3. Layers in Architecture of DenseNet201 model.

Stage	Operator	Layers (n)	Output Channels(c)	Stride (s)	Output Size
1	Conv2D 7×7	1	64	2	$112 \times 112 \times 64$
	MaxPool 3×3	1	-	2	$56 \times 56 \times 64$
2	Dense Block 1 (Bottleneck)	6	256	1	$56 \times 56 \times 256$
	Transition Layer 1	1	128	2	$28 \times 28 \times 128$
3	Dense Block 2 (Bottleneck)	12	512	1	$28 \times 28 \times 512$
	Transition Layer 2	1	256	2	$14 \times 14 \times 256$
4	Dense Block 3 (Bottleneck)	48	1792	1	$14 \times 14 \times 1792$
	Transition Layer 3	1	896	2	$7 \times 7 \times 896$

5	Dense Block 4 (Bottleneck)	32	1920	1	$7 \times 7 \times 1920$
6	BatchNorm → ReLU → GAP	1	1920	-	$1 \times 1 \times 1920$
7	Fully Connected (Dense)	1	1000 (classes)	-	$1 \times 1 \times 1000$

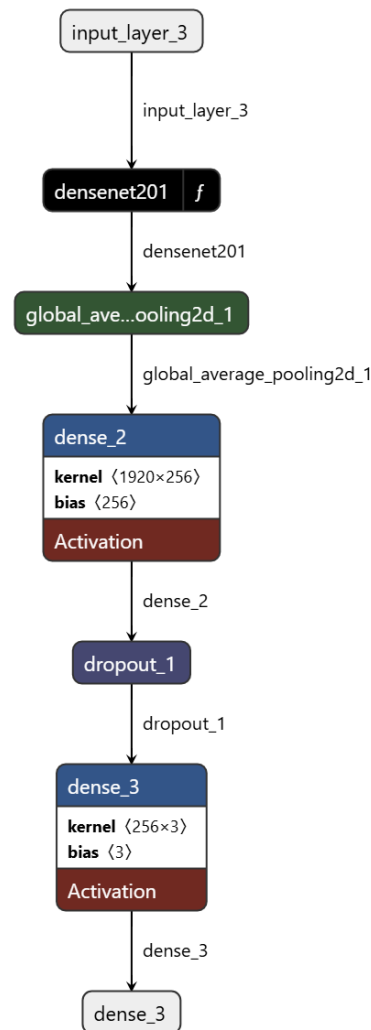


Fig 4. Architecture of DenseNet201 model

3.5.4 InceptionV3

InceptionV3 was introduced as an advancement in existing Inception networks in 2015 along with InceptionV2 [9]. It was developed to achieve computational efficiency while keeping the high performance intact. Therefore, it has less computational cost as compared to the VGG model despite the complexity in the architecture of Inception models. The InceptionV3 model is made up of 48 layers in total and comprises of various Inception and Reduction Blocks but one initial Stem block. The stem block is mainly responsible for extracting low level features from the input image while the Inception block, also called building blocks of Inception models are primarily responsible for parallel operations to capture multi-scale features. This includes 1×1 , 3×3 , 5×5 convolutions followed by 3×3 max pooling layer. The Reduction block however down samples the feature maps to maintain computational feasibility. The preprocessing used in the case of InceptionV3 model has the same functionality as preprocess function in MobileNetV2 allowing the input pixels to be scaled between -1 and +1.

Table 4. Layers in Architecture of InceptionV3 model. (For Standard Input size = $299 \times 229 \times 3$)

Stage	Operator / Block	Repetitions (n)	Output Channels (c)	Stride (s)	Output Size
1	Conv2D 3×3	1	32	2	$149 \times 149 \times 32$
	Conv2D 3×3	1	32	1	$147 \times 147 \times 32$
	Conv2D 3×3	1	64	1	$147 \times 147 \times 64$
	MaxPool 3×3	1	-	2	$73 \times 73 \times 64$
	Conv2D 1×1	1	80	1	$73 \times 73 \times 80$

	Conv2D 3×3	1	192	1	71×71×192
	MaxPool 3×3	1	-	2	35×35×192
2	Inception Module A	3	288	1	35×35×288
	Reduction A	1	768	2	17×17×768
3	Inception Module B	5	768	1	17×17×768
	Reduction B	1	1280	2	8×8×1280
4	Inception Module C	2	2048	1	8×8×2048
5	Global Average Pooling	1	-	-	1×1×2048
6	Dropout (keep_prob=0.8)	1	-	-	1×1×2048
7	Fully Connected (Dense)	1	1000 (classes)	-	1×1×1000

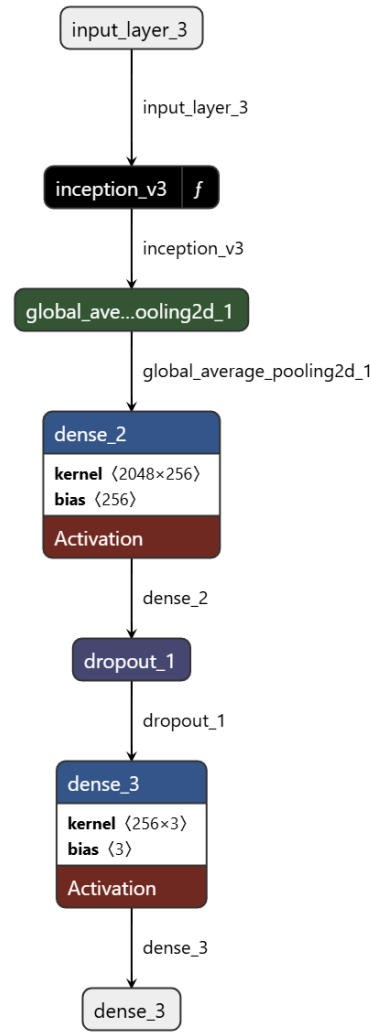


Fig 6. Architecture of InceptionV3 model

3.5.5 *EfficientNetB0*

EfficientNetB0 is a baseline architecture comprising of 82 layers belonging to the EfficientNet family introduced by Mingxing Tan and Quoc V. Le in 2019 [10] where the authors proposed a scaling method which uniformly scales depth, width, or resolution using a single yet effective compound coefficient. Contrary to other models, EfficientNetB0 has relatively less parameters and it does transfer well. The B0 model also serves as a foundation for larger variants like from B1 to B7. The architecture of EfficientNetB0 is made up of a Stem block, MBConv Block and Head block. The stem block comprises of 3*3 convolution with stride 2 and 32 filters followed by batch normalization and swish activation. The building block, MBConv (Mobile Inverted Bottleneck Convolution) includes a 3*3 or 5*5 depthwise convolution, Squeeze & Activation (SE) modules for channel attention and a 1*1 Projection layer. This is

somehow similar to the bottleneck layers observed in MobileNetV2. A total of 7 MBConv stages exist in the structure of EfficientNetB0, each having varying number of layers, kernel sizes, strides, and expansion factors. The final Head block consists of a 1×1 convolution, a global average pooling layer, and a dense layer for classification. Like other models, we have implemented preprocess, a pass-through function here also because EfficientNetB0 expects the input to be float tensors of pixels having a ranging value from 0 to 255.

Table 5. Layers in Architecture of EfficientNetB0 model.

Stage	Operator / Block	Expansion (t)	Output Channels (c)	Repeats (n)	Stride (s)	Output Size
1	Conv2D 3×3	-	32	1	2	$112 \times 112 \times 32$
2	MBConv1, 3×3	1	16	1	1	$112 \times 112 \times 16$
3	MBConv6, 3×3	6	24	2	2	$56 \times 56 \times 24$
4	MBConv6, 5×5	6	40	2	2	$28 \times 28 \times 40$
5	MBConv6, 3×3	6	80	3	2	$14 \times 14 \times 80$
6	MBConv6, 5×5	6	112	3	1	$14 \times 14 \times 112$
7	MBConv6, 5×5	6	192	4	2	$7 \times 7 \times 192$
8	MBConv6, 3×3	6	320	1	1	$7 \times 7 \times 320$
9	Conv2D 1×1 + BN + Swish	-	1280	1	1	$7 \times 7 \times 1280$
10	Global Average Pooling	-	-	1	-	$1 \times 1 \times 1280$
11	Dropout (rate = 0.2)	-	-	1	-	$1 \times 1 \times 1280$
12	Fully Connected	-	1000 (classes)	1	-	$1 \times 1 \times 1000$

	(Dense)					
--	---------	--	--	--	--	--

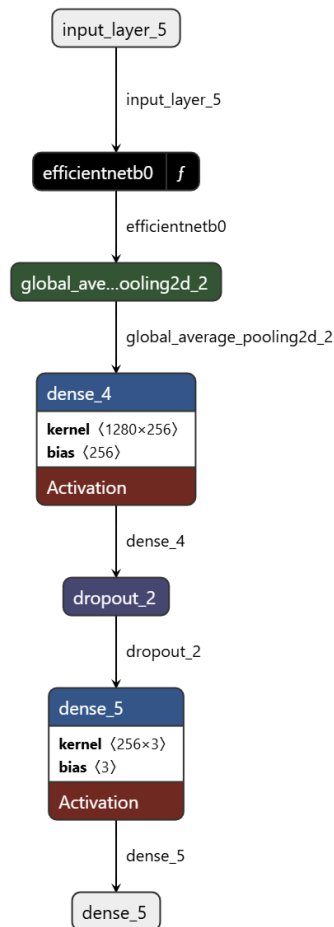


Fig 7. Customized Architecture of EfficientNetB0 model

3.5.6 NASNetMobile

NASNetMobile is a mobile-enabled, less complex version of full NASNet model which was initially derived from NAS (Neural Architecture Search) [12]. NAS is an automated ML technique proposed by B. Zoph and Q. V. Le. in 2017 with an aim to discover best possible Neural Network Architectures for any given dataset. It was a technique that instead of manually designing algorithms (like VGG, ResNet, etc.), focused on using an algorithm to search a large space of possible architectures. NAS inspired Barret Zoph et.al. [11] to design NASNet which is capable to automatically design high performing Neural Network using reinforcement learning instead of manual engineering of models. It utilized a new regularization technique called ScheduledDropPath which aimed to improve generalization in NasNet models. The mobile version, NASNetMobile is made

up of two types of cells, Normal cell and Reduction cell. Each cell contains depthwise convolutions of size 1×1 followed by 3×3 or 5×5 convolutions, max and average pooling layers, addition or concatenation of feature maps and ReLU activation and batch normalization. The Normal cell preserves spatial dimensions of feature maps while the Reduction cell decreases spatial resolution (height & width) while increasing the number of filters. Preprocessing input samples include scaling the pixels value between -1 and +1.

Table 6. Layers in Architecture of NASNetMobile model.

Stage	Operator / Block	Repetitions (n)	Output Channels (c)	Stride (s)	Output Size
1	Conv2D 3×3	1	32	2	$112 \times 112 \times 32$
2	Stem Cell	1	44	2	$56 \times 56 \times 44$
3	Normal Cell	1	88	1	$56 \times 56 \times 88$
	Normal Cell	3	88	1	$56 \times 56 \times 88$
4	Reduction Cell	1	176	2	$28 \times 28 \times 176$
5	Normal Cell	4	176	1	$28 \times 28 \times 176$
6	Reduction Cell	1	352	2	$14 \times 14 \times 352$
7	Normal Cell	4	352	1	$14 \times 14 \times 352$
8	Reduction Cell	1	704	2	$7 \times 7 \times 704$
9	Normal Cell	3	704	1	$7 \times 7 \times 704$
10	Global Average Pooling (GAP)	1	-	-	$1 \times 1 \times 704$
11	Dropout (rate = 0.5)	1	-	-	$1 \times 1 \times 704$
12	Fully Connected	1	1000	-	$1 \times 1 \times 1000$

	(Dense)		(classes)		
--	---------	--	-----------	--	--

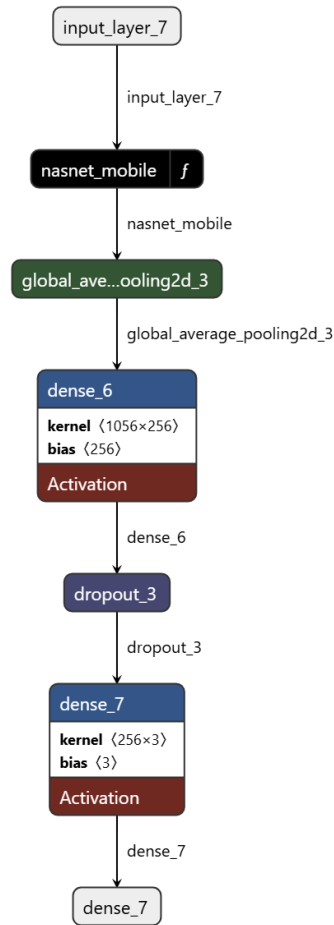


Fig 8. Customized Architecture of NASNetMobile model

3.5.7 ResNet50 and ResNet101

Residual Networks or ResNet were first introduced by a team of Microsoft Researchers in 2015 [13]. The ResNet architecture won the first place on ILSVRC 2015 classification task which ultimately led to the breakthrough of residual learning and the use of residual blocks which allows Skip connections mechanism to address the issue of vanishing gradient. Skip connections allow networks to skip layers and pass information directly forward. This not only helps the model to train deeper but also keeps the performance intact. Therefore, instead of learning $H(x)$ from input x , the network learns a residual function:

$$F(x) = H(x) - x \Rightarrow H(x) = F(x) + x$$

This makes sure that the network only learns the residual then add it back to original input. ResNet50 and ResNet101 both are the part of ResNet family and follow bottleneck architecture which consists of a 1×1 convolution to reduce number of channels, a 3×3 convolution to extract spatial features and a 1×1 convolution for restoring original dimensions. The main difference between both models lies in their depth. While the ResNet50 model contains only 16 bottleneck blocks organized in four stages, the ResNet101 expands the blocks in the third stage resulting in a total of 33 bottleneck blocks. This difference causes the ResNet50 and ResNet101 models to have a total of 50 and 101 layers respectively. Due to their deeper architecture ResNets generally have more parameters as compared to other models but they are highly stable and generalizable. Additionally, input image preprocessing works in the same way as VGG16, converting the RGB values to BGR and then zero centering each color channel without scaling with respect to the ImageNet dataset.

Table 7. Layers in Architecture of ResNet50 model.

Stage	Block Type	Repetitions (n)	Output Channels (c)	Stride (s)	Output Size
1	Conv2D 7×7	1	64	2	$112 \times 112 \times 64$
	MaxPool 3×3	1	-	2	$56 \times 56 \times 64$
2	Bottleneck (1×1 , 3×3 , 1×1)	3	256	1	$56 \times 56 \times 256$
3	Bottleneck (1×1 , 3×3 , 1×1)	4	512	2	$28 \times 28 \times 512$
4	Bottleneck (1×1 , 3×3 , 1×1)	6	1024	2	$14 \times 14 \times 1024$
5	Bottleneck (1×1 , 3×3 , 1×1)	3	2048	2	$7 \times 7 \times 2048$
6	Global Average Pooling	1	-	-	$1 \times 1 \times 2048$
7	Fully Connected	1	1000 (classes)	-	$1 \times 1 \times 1000$

	(Dense)				
--	---------	--	--	--	--

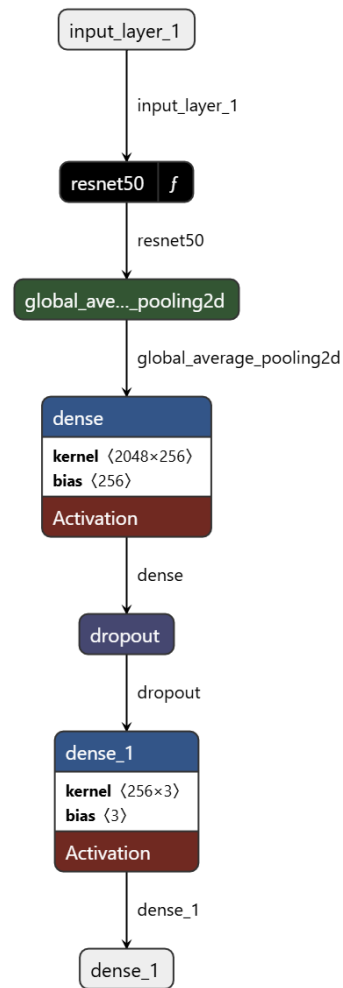


Fig 9. Customized Architecture of ResNet50 model

Table 8. Layers in Architecture of ResNet101 model.

Stage	Block Type	Repetitions (n)	Output Channels (c)	Stride (s)	Output Size
1	Conv2D 7×7	1	64	2	112×112×64
	MaxPool 3×3	1	-	2	56×56×64

2	Bottleneck (1×1 , 3×3 , 1×1)	3	256	1	$56 \times 56 \times 256$
3	Bottleneck (1×1 , 3×3 , 1×1)	4	512	2	$28 \times 28 \times 512$
4	Bottleneck (1×1 , 3×3 , 1×1)	23	1024	2	$14 \times 14 \times 1024$
5	Bottleneck (1×1 , 3×3 , 1×1)	3	2048	2	$7 \times 7 \times 2048$
6	Global Average Pooling	1	-	-	$1 \times 1 \times 2048$
7	Fully Connected (Dense)	1	1000 (classes)	-	$1 \times 1 \times 1000$

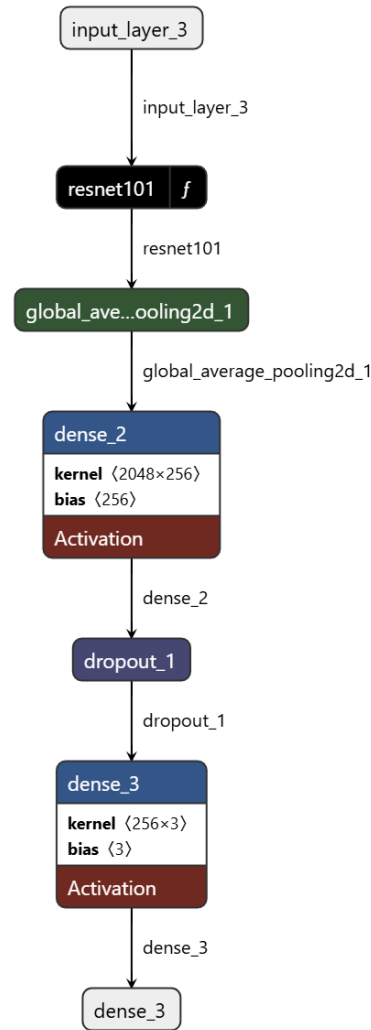


Fig 10. Customized Architecture of ResNet101 model

3.5.8 Xception

The development of Xception was inspired by Inception's architecture and was brought into light in 2017 by Francois Chollet [14] where the author replaced the inception modules with depthwise separable convolutions which make the model simpler and efficient. The Xception model surpassed InceptionV3 on the ImageNet dataset despite having nearly equal number of parameters indicating greater performance. The main idea behind Xception's architecture is that spatial correlation and cross-channel correlation in feature maps can be learned or mapped separately. This can be done by replacing traditional convolutions by depthwise convolution followed by a 1*1 partitioning convolution to capture cross-channel interaction. Xception uses 36 convolution layers for feature extraction which are organized into three flows. The first flow is called Entry Flow having 9 of the convolution layers, followed by the Middle Flow with 24

convolution layers. The middle flow is the part where most of the model's depth resides. The last flow known as Exit Flow comprises of 2 depthwise separable convolutions and 1 projection convolution. This is followed by the global average pooling layer and a fully connected dense layer for classification which was modified for 3 neurons in our case.

Table 9. Layers in Architecture of Xception model. (Standard Input size = 299x229x3)

Stage	Block	Description	Repeats	Output Size
0	Input	Input image	—	299×299×3
1	Conv2D	3×3 conv, 32 filters, stride 2	1	149×149×32
2	Conv2D	3×3 conv, 64 filters, stride 1	1	149×149×64
3	Entry Flow	Depthwise Separable Conv + Residual (to 128), stride 2	1	74×74×128
4	Entry Flow	Depthwise Separable Conv + Residual (to 256), stride 2	1	37×37×256
5	Entry Flow	Depthwise Separable Conv + Residual (to 728), stride 2	1	19×19×728
6	Middle Flow	3× Depthwise Separable Conv + Residual, all stride 1	8	19×19×728
7	Exit Flow	Depthwise Separable Conv + Residual (to 1024), stride 2	1	10×10×1024
8	Exit Flow	Separable Conv 3×3 → 1536	1	10×10×1536
9	Exit Flow	Separable Conv 3×3 → 2048	1	10×10×2048

10	Global AvgPool	Average over 10×10	1	1×1×2048
11	Dense	Fully Connected → Softmax (e.g., 1000 classes for ImageNet)	1	1×1×1000

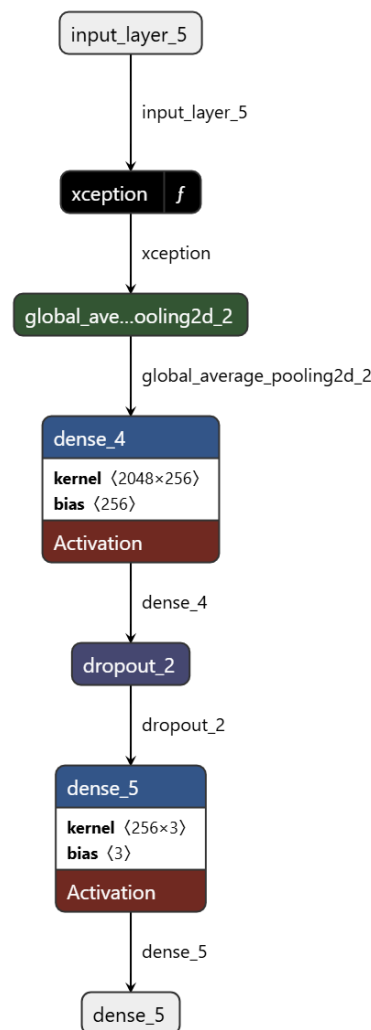


Fig 11. Customized Architecture of Xception model

3.5.9 Ensemble Learning

Ensemble Learning combines strength of multiple models often referred as “base or weak learners” to build a more robust and accurate model. The motive behind ensemble learning is that a single model may have bias or limitations, however they can be overcome by combining the models can improve overall prediction process. This is

effective in cases where different models perform differently over various subsets of dataset. A common method used to implement ensemble approach is using the voting method which is often used to aggregate predictions. Generally, there are two types of voting mechanisms, soft voting and hard voting. The hard voting method selects the final prediction using the class label with most votes. Hence, each model makes a class prediction but final output is always determined by majority voting. Hard voting treats all classifiers equally and doesn't really consider probabilities associated with each prediction. Another voting method, soft voting takes predicted class probabilities from each model. Therefore, it doesn't really count votes instead the focus is on computing the average of predicted probabilities for each class and select the class with highest average probability. Generally, soft voting is better than hard voting in terms of performance. The third ensemble strategy is feature-based ensemble also called feature fusion, it integrates intermediate feature representation extracted from multiple models. Unlike voting strategies, this operates on a deeper level and aggregates learned features prior to classification. These features are either combined using addition or concatenation function. In addition-based fusion, corresponding feature maps from different models are added element wise while concatenation-based fusion stacks the feature maps along the channel dimension. In this study, ensemble of NASNetMobile, EfficientNetB0, and ResNet50 was trained and validated for all three strategies. The corresponding weights used in soft voting for NASNetMobile, EfficientNetB0, and ResNet50 are 0.4, 0.31, and 0.29. Furthermore, one more ensemble including models VGG16, ResNet50 and MobileNetV2 was trained under addition-based feature fusion.

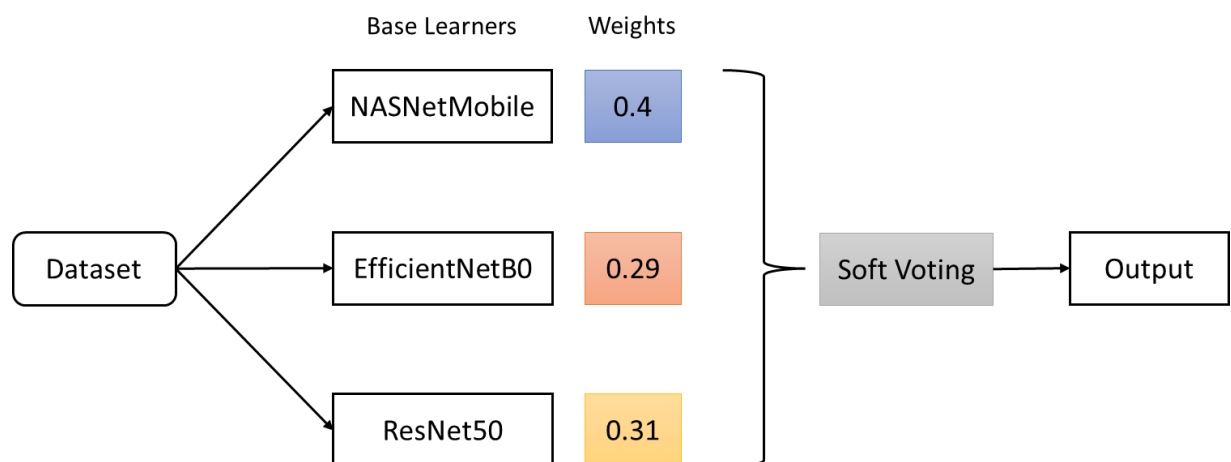


Fig. 12 Ensemble Model using soft voting

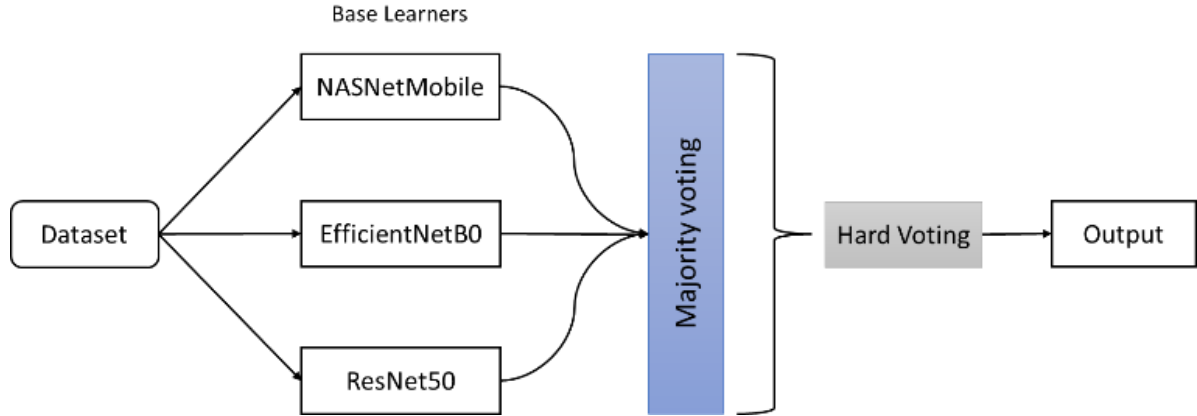


Fig. 13 Ensemble Model using hard voting

3.6 Evaluation Metrics

To assess the performance of the pretrained models and ensemble approaches, multiple evaluation metrics were used:

1. Accuracy: It is measure of how well a model correctly classifies both instances, positive and negative.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. Precision: Indicates how many of the instances predicted as positive are actually positive.

$$Precision = \frac{TP}{TP + FP}$$

3. Recall: It is also called sensitivity and shows the proportion of true positives detected out of all the actual positive instances.

$$Recall = \frac{TP}{TP + FN}$$

4. F1 Score: A harmonic mean of precision and recall, especially useful when dealing with imbalanced datasets.

$$f1 - score = \frac{2 * Precision * Recall}{Precision + Recall}$$

5. Confusion Matrix: Provides a detailed breakdown of true positives, false positives, true negatives, and false negatives for each class.
6. Training and Validation Curves: Plotted to observe trends in loss and accuracy over epochs to ensure the model is learning and not overfitting.

CHAPTER IV

SYSTEM WORKFLOW

This chapter briefs about the methodology adopted to achieve objectives of this research. It presents step-by-step description of the processes involved, following from the Data collection, Preprocessing, Model training and Evaluation. The chapter opens up by providing a detailed overview about the creation of video dataset to frame extraction and moves down to other subsections. Additionally, it also outlines how a simple and interactive website was developed using flask to allow users to upload image/videos or access their device camera to predict the quality of guava in real time scenarios.

4.1 System Architecture

In the System Architecture section, we discussed the functional aspect of the system, including how it behaves as a system; how it functions and processes data. This encompassed designing the data flow as well as the overall function of the application. This section will explore the individual components used, with some emphasis placed on the selection and suitability of those components for this particular task.

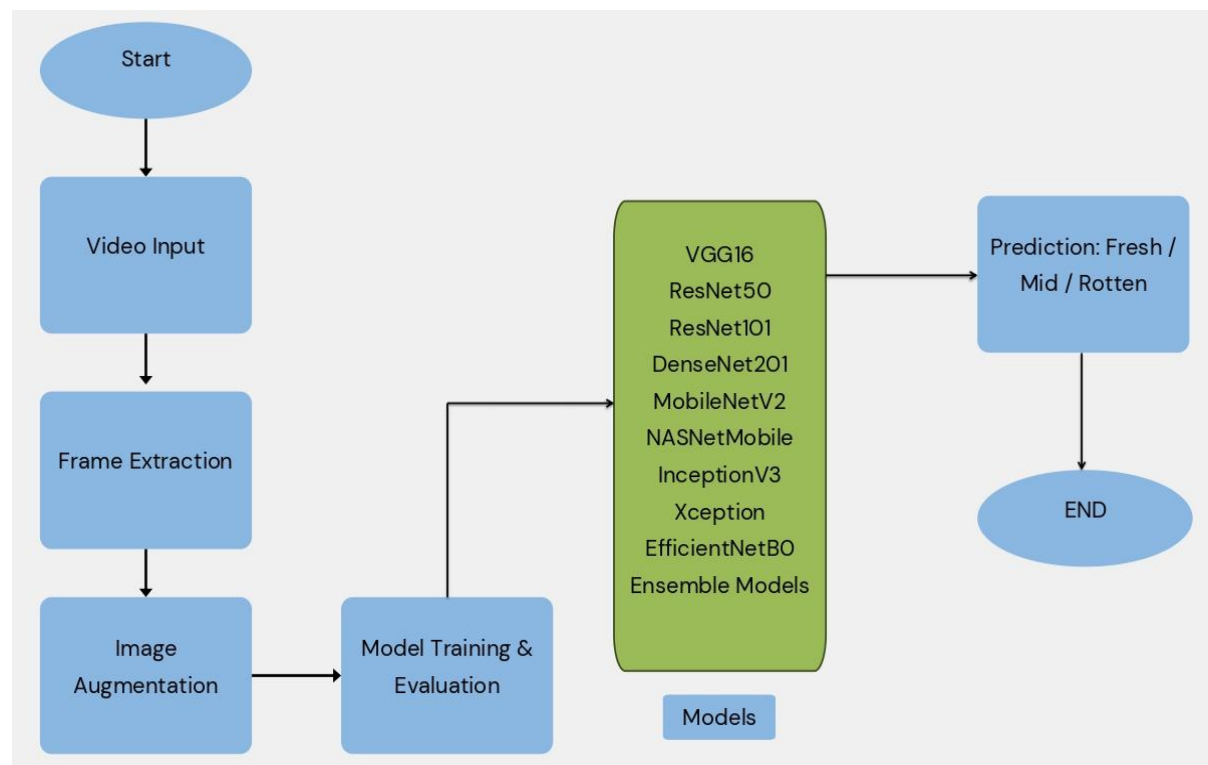


Fig. 14 System Architecture

In this project, we will be using several deep learning models and compare the performance of each to identify the most efficient model. The optimal model will be similar to one with good accuracy, but will allow for the most efficient processing time, making it compatible for use in a real-time web application. The main goal will be focused on training a model capable of classifying the quality of guava fruits using a video input.

This project is based on mainly three components: -

1. Deep Learning Model (MobileNetV2) :-

In regards to our project, we have trained a total of 9 different pretrained deep learning models as well as 4 ensemble models and reported on the models' performance based on accuracy and inference time. After evaluating the models with regard to balance, we determined that the MobileNetV2 model achieves the optimal balance between accuracy and performance.

MobileNetV2 was a suitable model as it is designed specifically for mobile and embedded vision applications. It is a lightweight, efficient convolutional neural network that performs reasonably well and does not require high computational power. It is generally found in real-world cases such as detection of objects, classification of fine-grained objects (recalling specimen A), face recognition, object localization, and a variety of other applications. It employs depthwise separable convolutions, which reduce computational costs without much loss of quality. It also uses inverted residual blocks and linear bottlenecks, which are very useful for mobile and real-time applications.

MobileNetV2 was fine-tuned using a custom guava dataset with three output classes labeled Fresh, Mid, and Rotten, during which the last fully connected layers were modified. Transfer learning effectively utilized pretrained ImageNet weights, which sped up convergence and improved generalization on small datasets.

2. User Interface (Frontend):-

For our front end, we designed an intuitive web interface that allows users to easily upload video files to be analyzed. The interface was developed using standard web technologies:

- HTML was used to describe the structure and form of the web pages.
- CSS controlled the visual style to make sure all pages look pleasing and consistent.
- JavaScript implemented the application logic so users would have a responsive experience based on their inputs.

The end result is a simple and intuitive interface, allowing an easy upload video or image file, while providing a satisfying experience overall.

3. *Backend:-*

For the backend, we have utilized Flask (a Python web framework) in order to receive video or image input from the user and run it through the trained MobileNetV2 model. The backend does several things:

- It receives the uploaded video via the web interface.
- It creates a temporary folder to place the uploaded videos and perform actions.
- It takes action by splitting the uploaded video into multiple frames.
- It preprocesses the frames, before sending them through the model for quality classification.

All of the application's backend logic is housed in the app.py file. This includes routing HTML pages, handling file uploads, segmenting videos into frames, preprocessing the frames, and sending the data through the model. We also incorporated email into the backend, to send results, or can send if people need to be notified via email.

4.2 Flow Diagram / Block Diagram

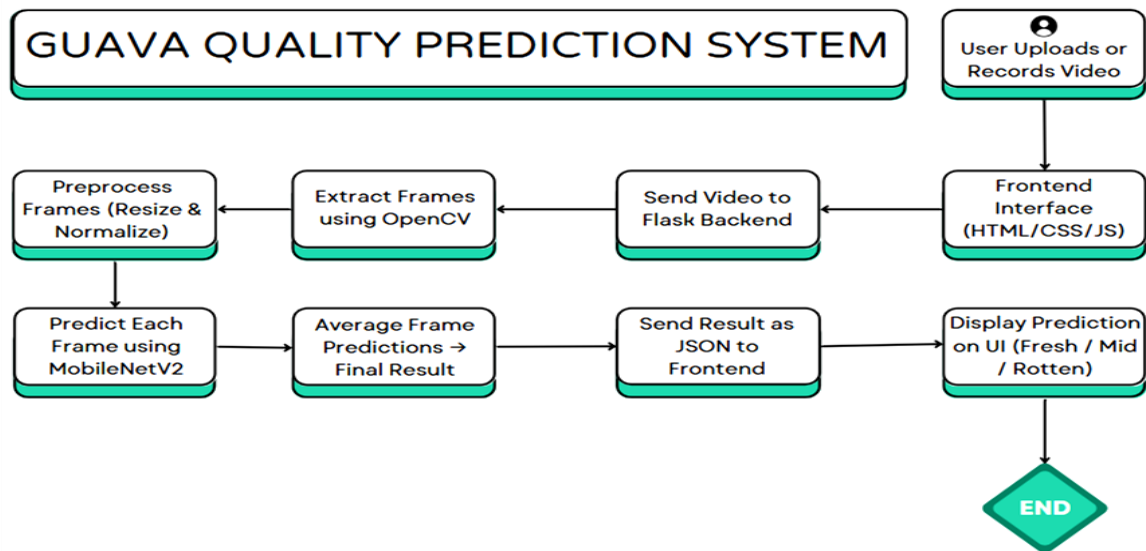


Fig. 15 Workflow Diagram

Workflow Diagram of the Guava Quality Prediction System :-

The whole architecture of the guava quality prediction system is represented in the above flowchart. As indicated in the use case, the process starts with the user uploading or recording a video of a guava by accessing the web application developed using HTML, CSS, and JavaScript, which together provided the user with a familiar and seamless user experience.

After the user submits the video, the video is sent to the Flask backend server and the prediction class is executed. At the backend, the OpenCV library extracts a series of frames from the video. Each frame is preprocessed by resizing and normalizing the pixel statistics to meet the input requirements of the deep learning model.

Each of the frames is then passed to a pre-trained MobileNetV2 model which makes a classification of either Fresh, Mid, or Rotting guava. Once predictions for the frames are obtained, the results for the frames are aggregated to form an average "cleanliness" measurement used to identify the final quality classification of the guava.

In the penultimate step of the process, the classification response is sent from backend server to frontend server for ease of human interpretation and display on the screen to the user. The backend-to-frontend server and previous step form an end-to-end pipeline, providing the user with a seamless user experience by enabling them to receive a ready-to-go automated guava quality prediction classification from a video input example.

4.3 Modules Description

This project's objective is to categorize Cleanliness of guava fruits quality into three levels of cleanliness: Fresh, Mid, and Rotten. We trained several deep learning models for the categorization, and picked the most accurate and efficient one.

The project was broken down into following modules to improve the developmental progress. Below, each module is given in more detailed format:

4.3.1 Data Preparation & Frame Extraction Module:-

For this module, we manually collected data by recording and compiling video samples of guavas in all three quality categories of Fresh, Mid and Rotten. These videos are first-hand input for our model. Using OpenCV, we captured every frame of each video that we collected, and annotated each quality class.

The '*extract_frames*' function was developed to automate the steps of extracting a fixed number of frames evenly at intervals from each guava quality video. This was a crucial step in converting the video dataset we collected to an image-based dataset for use in deep learning model training.

The function takes in four parameters:

1. `video_path`: The file path to the input video.
2. `output_folder`: The path to save the extracted frames.
3. `category`: The video label (Fresh, Mid, Rotten) that will be used in the filename to help organize the data.
4. `num_frames`: The number of frames to extract from each video (default is 5).

Working of '*extract_frames*' function:

1. Loading the Video: The method will load the video path using OpenCV's `VideoCapture`.
2. Getting Number of Frames: It will get the total number of frames in the video using `CAP_PROP_FRAME_COUNT`.
3. Selecting Frame Index: It will use NumPy's `linspace` to get frame indices that are evenly spaced out at different time intervals.

4. Extracting Frames: Read and save the frames sequentially, and only save the frames that match with the calculated positions.
5. Saving and Naming: Each saved frame will be named, using the category, video name, and frame index; this will allow for well-organized datasets that are easy to backtrack.
6. Freeing up resources: The method will terminate the video capture, which will free up resources on the system.

Code:-

```
# Function to extract frames
def extract_frames(video_path, output_folder, category, num_frames=5):
    video_name = os.path.splitext(os.path.basename(video_path))[0]
    cap = cv2.VideoCapture(video_path)
    total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
    if total_frames < num_frames:
        print(f"Video {video_name} has fewer than {num_frames} frames.")
        num_frames = total_frames

    # Evenly spaced frame indices
    frame_indices = np.linspace(0, total_frames - 1, num_frames, dtype=int)
    current_index = 0
    saved_count = 0
    frame_id = 0
    while cap.isOpened() and saved_count < num_frames:
        success, frame = cap.read()
        if not success:
            break
        if frame_id == frame_indices[saved_count]:
            frame_filename = f"{category}_{video_name}_frame_{saved_count}.jpg"
            cv2.imwrite(os.path.join(output_folder, frame_filename), frame)
            saved_count += 1
            frame_id += 1
    cap.release()
```

Additionally, to improve the variability and size of our dataset, we have used data augmentation. This technique provides various perspectives and simulated conditions to the model, thereby allowing it extract more generalized features. The end result was our dataset grew in size, improved the model's ability classify the guava quality from video input, and improved the reduced risk of overfitting.

This function '*augment_images*' is created using the ImageDataGenerator class from the TensorFlow Keras API, which provides a useful utility for real-time data augmentation of image data.

The main components in augmentation strategy: -

1. **Augmentation Strategy:** The input images underwent certain transformations:
 - i. Rotate: The images were randomly rotated within a range of ± 20 degrees for rotation. Width Shift and Height Shift: Images were translated up to 10% horizontally and vertically.
 - ii. Shear and Zoom: Shearing and zooming were done for 20%. Horizontal Flip: The images were horizontally flipped.
 - iii. Brightness: Brightness was increased slightly by 20%.
 - iv. Fill Mode: The missing pixels after image transformations were filled in using the 'nearest' strategy.
2. **Directory Set-up:** When the function is executed: It will accept an input directory that contains the original frames. It then creates an output directory (if it does not exist), in which augmented images will be saved to.
3. **Image Processing:** For each image in the input directory: It is loaded and converted to a NumPy array. The image is reshaped to add a batch dimension and a loop is executed to generate a defined number of augmented images (n_augmented). These images are saved with a prefix (aug_) in the filename. The prefix will distinguish the augmented images from the original frames.
4. **Multiple Class Handling:** The script is executed three times for each guava quality class: Fresh, Mid and Rotten.

Code: -

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator, load_img,  
img_to_array, save_img
```

```
def augment_images(input_dir, output_dir, n_augmented=10):
```

```

#Augmentations
datagen = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.1,
    height_shift_range=0.1,
    shear_range=0.1,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    brightness_range=[1.2, 1.2]
)

#Output dir to save aug images
os.makedirs(output_dir, exist_ok=True)
print(f'Saving augmented images to: {output_dir}')
for img_name in os.listdir(input_dir):
    img_path = os.path.join(input_dir, img_name)
    try:
        img = load_img(img_path)
        x = img_to_array(img)
        x = x.reshape((1,) + x.shape)
        i = 0
        for batch in datagen.flow(x, batch_size=1):
            aug_img_name = f'aug_{i}_{img_name}'
            aug_img_path = os.path.join(output_dir, aug_img_name)
            save_img(aug_img_path, batch[0])
            i += 1
        if i >= n_augmented:
            break
    except Exception as e:
        print(f'Error processing {img_name}: {e}')

#Augmented images folder

```

```
base_path = "/content/drive/MyDrive/data"
os.makedirs(os.path.join(base_path, "augmented"), exist_ok=True)
augment_images(os.path.join(base_path, "fresh_frames"), os.path.join(base_path,
"augmented", "augmented_fresh_frames"))
augment_images(os.path.join(base_path, "mid_frames"), os.path.join(base_path,
"augmented", "augmented_mid_frames"))
augment_images(os.path.join(base_path, "rotten_frames"), os.path.join(base_path,
"augmented", "augmented_rotten_frames"))
```

4.3.2 Model Training Module:-

During the training step, we constructed and evaluated 13 models.

They consisted of:

- 9 pretrained CNN models
- 1 Soft Voting Ensemble model
- 1 Hard Voting Ensemble model
- 2 Feature Concatenation Ensemble models

The pretrained models were chosen for their popularity in previous image classification efforts, and for the fact that they worked reasonably well in that capacity. All models were trained upon the augmented dataset, and we simply reported which model achieved the best accuracy. The ensemble models were built to combine the strengths of the individual models above, for the potential to increase the accuracy of prediction.

The models were evaluated based on the accuracy of the classification, the time taken for prediction, and the computational efficiency. The evaluation indicated that the model achieving the most accurate predictions was VGG16. However, MobileNetV2 provides better trade off between accuracy and inference time making it a better option for deployment. Implementation of both of these models are explained in detail below:

4.3.2.1 VGG16 Implementation and Customization

Among the many deep-learning models that were tested VGG16 was one of the most accurate guava quality predictors with a validation accuracy of 96.78%. For this project, we used the VGG16 architecture pre-trained on the standard ImageNet dataset as the base model. VGG16 is a well-known convolutional neural network trained on a large dataset

(ImageNet) which is well-known in the deep-learning community, mainly, because of the model's simplicity and depth. VGG16 consists of 16 layers with small 3x3 filters and same architecture throughout.

In this implementation, we load the pre-trained VGG16 network using the Keras Applications module while setting `weights='imagenet'`. By setting `include_top=False`, we were able to exclude the old fully connected layers from the original VGG16 (which was created for ImageNet classification with 1000 classes), and use the model for our own custom classification of three guava quality types (Fresh, Mid, and Rotten) while only utilizing parts of the pre-trained model. The input shape was defined as (224, 224, 3) to accommodate the VGG16 model.

In order to keep the powerful learned features from ImageNet, all layers of the base VGG16 model were frozen (i.e., made non-trainable). A custom classifier was added on top of this base using the Keras Sequential API. The custom head included a GlobalAveragePooling2D layer to reduce the spatial dimensions, a fully connected Dense layer with 256 neurons (ReLU activation), a Dropout layer with a dropout rate of 0.4 to mitigate overfitting, and finally, a Dense output layer with 3 neurons, using softmax activation to output class probabilities for the three guava categories.

Code: -

```
from tensorflow.keras.applications import VGG16

#Loading vgg16
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224,
3))

# Freezing the base layers
for layer in base_model.layers:
    layer.trainable = False

#Model Building
model_vgg = Sequential([
    base_model,
    GlobalAveragePooling2D(),
    Dense(256, activation='relu'),
    Dropout(0.4),
    Dense(3, activation='softmax')
```

```
])  
#Compilation  
model_vgg.compile(optimizer=Adam(learning_rate=1e-4),  
loss='categorical_crossentropy', metrics=['accuracy'])  
  
#Model summary  
model_vgg.summary()
```

4.3.2.2 MobileNetV2 Implementation and Customization

To find the right balance between having classification accuracy and computation, the MobileNetV2 model was chosen as the final deployed model. MobileNetV2 is a lightweight, pre-trained convolutional neural network model for mobile and real-time applications. In this project, the MobileNetV2 model was initialized with pre-trained ImageNet weights and with `include_top=False` to avoid the original model's classification layers so the researcher can add their own specific to the task. The input shape was also set to the dimensions of the images after preprocessing.

Each base model's layers were frozen to avoid overfitting and lower training complexity; essentially, when the weights were frozen, they were not trained. A custom classification head was added, which was composed of a `GlobalAveragePooling2D` layer to reduce spatial dimensions of the output; a dense layer with 256 neurons and ReLU activation to learn complex patterns; a dropout layer to apply a 0.4 dropout rate to reduce overfitting through random neuron deactivation during training; and a dense output layer with three neurons and softmax activation to classify images into the three target classes: Fresh, Mid, and Rotten.

Code: -

```
base_model = MobileNetV2(input_shape=(img_size[0], img_size[1], 3),  
include_top=False, weights='imagenet')  
base_model.trainable = False  
  
model_mobilenetv2 = Sequential([  
    base_model,  
    GlobalAveragePooling2D(),  
    Dense(256, activation='relu'),
```

```
Dropout(0.4),
Dense(3, activation='softmax')
])
model_mobilenetv2.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])
model_mobilenetv2.summary()
```

4.3.2.3 Ensemble Soft Voting Model - NER

To further improve classification robustness, a soft voting ensemble method was applied by summing the output probabilities from the three highest-performing models, NASNetMobile, ResNet50, and EfficientNetB0. Not only does this method choose from only one final prediction from a model, but rather, the predictions of all three models are brought together by summing the predicted class probabilities in a weighted average. Empirically, weights were assigned based on the individual model's accuracies whereby NASNetMobile contributed 40%, ResNet50 contributed 29%, and EfficientNetB0 contributed 31% to the final ensemble prediction.

Code: -

```
avg_pred_weighted = (0.4*pred_nasnet + 0.29*pred_resnet + 0.31*pred_efficientnet) /
3.0
ensemble_prediction_weighted = np.argmax(avg_pred_weighted, axis=1)
```

4.3.2.4 Ensemble Hard Voting Model - NER

Aside from the soft voting strategy, a hard voting ensemble was also utilized to assess prediction consistency among models. In this approach, the predicted classes were used from three pretrained classifiers - NASNetMobile, EfficientNetB0, and ResNet50 - by utilizing the argmax of their output probability vectors. These individual class labels were then stacked, and the same majority voting strategy was applied using the statistical mode function. Thus, for each input instance, the final ensemble prediction was the class that had been predicted the most across the three model predictions.

Code: -

```
from scipy.stats import mode
```

```
stacked_pred = np.stack([
    np.argmax(pred_nasnet, axis=1),
    np.argmax(pred_efficientnet, axis=1),
    np.argmax(pred_resnet, axis=1)], axis=1)

hard_vote_pred, count = mode(stacked_pred, axis=1) #mode, count
final_preds = hard_vote_pred.flatten()
```

4.3.2.5 Feature Fusion Ensemble Model - VRM

To increase the robustness and generalization capabilities of the classification system, we implemented a feature-level fusion ensemble model. While soft or hard voting combines outputs from several pre-trained models, feature-level fusion ensembles combine intermediate feature representations from multiple pre-trained models at the architectural level.....Three of the highest performing models, VGG16, ResNet50, and MobileNetV2 were loaded using the best-trained model weights, and then the internal structure of the models was modified to accept a common input using a custom renaming wrapper function. The renaming wrapper function essentially made a new identical version of each model while allowing the models to operate in parallel on a common input tensor.

The `rename_model()` function was used to wrap each model in a new scope with its own unique input layers:

```
def rename_model(model, prefix, input_shape=(224, 224, 3)):
    # Freeze original model
    model.trainable = True
    # Create new input
    new_input = Input(shape=input_shape, name=f'{prefix}_input')
    # Apply model to the new input
    new_output = model(new_input)
    # Create a new model with renamed scope
    new_model = Model(inputs=new_input, outputs=new_output, name=f'{prefix}_model')
    return new_model
```


All three networks were provided the same input image, and the output feature maps were merged with an elementwise addition layer. This merging strategy allowed the ensemble to take advantage of complementary features that were learned by each model:

```
commonInput = Input((224, 224, 3), name="ensemble_input")
out1 = vgg(commonInput)
out2 = resnet(commonInput)
out3 = mobilenet(commonInput)
# Merged outputs
merged = Add(name='merge_outputs')([out1, out2, out3])
# Ensemble model
newModel = Model(inputs=commonInput, outputs=merged, name='ensemble_model')
```

The resulting model is a parallel fusion of several deep architectures, at the feature level, resulting in a more robust set of features in terms of diversity. This type of architecture can be particularly useful if the individual models have learned different aspects of the input data, and the additive nature allows the ensemble to take advantage of the individual strengths of each network.

4.3.2.6 Feature Fusion Ensemble Model - NER

In addition to the previous feature fusion strategy, a second ensemble model was constructed to further take advantage of three additional performances of deep-learning architecture models: EfficientNetB0, ResNet50, and NASNetMobile. This variation continued the conceptualization of architectural level fusion but leveraged diversity by integrating internal designs, and computational efficiency. Each of the models was loaded with its respective best trained weights, then wrapped using the same custom `rename_model()` function to isolate input scopes and facilitate parallel integration.

Each of the three models were passed a shared input tensor of (224, 224, 3) dimensions. Feature outputs were merged into a unified feature using a Merge layer encasing an element-wise addition operation:

```
# common input
commonInput = Input((224, 224, 3), name="ensemble_input")
out1 = efficientnet(commonInput)
out2 = resnet(commonInput)
```

```

out3 = nasnet(commonInput)
# Merged outputs
merged = Add(name='merge_outputs')([out1, out2, out3])
# Ensemble model
newModel2 = Model(inputs=commonInput, outputs=merged, name='ensemble_model')

```

In combining the three various models, EfficientNet provides optimized scaling, ResNet delivers deep residual learning, and NASNet brings in adaptability based on neural architecture search - so we have now a great hybrid model in guava quality classification that balances efficiency and accuracy.

4.3.3 Model Evaluation & Selection Module :-

In this module we focused on all 13 trained models, in order to identify an applied model that has the greatest capacity to maximize performance in as little prediction time as possible. Several performance indicators were used to conduct this evaluation , accuracy, loss confusion, matrix, validation scores.

During this analysis, it was clear that the VGG16 yielded the highest accuracy of 96.78%, but was too slow to be viable in real-time applications and was computationally expensive. Aside from the VGG16 model, it was clear that MobileNetV2 offered a trade-off: it yielded high accuracy, and it was much faster in prediction time making it more suitable for real-time deployment conditions. To summarize in this module, it is clear that many models were deployable, but MobileNetV2 was the optimal model based on performance and speed of model.

4.3.4 Video Processing & Frame Prediction Module :-

In this module, we are going to talk about the backend operations that occur after the model has been selected and deployed on the frontend, namely the operations that are launched when the user uploads a video file from the user interface.

The backend operation has a mechanism to extract frames from the video file while the user uploads, using optimized techniques depending on the length of the video and time

to process. By extracting frames at regular intervals and evaluating only a simple subset of frames rather than troubled hours of video files, the working time is minimized. After extracting the frames, the model needs to preprocess each extracted frame to share the same size, normalized, and format as the model requires in input. The pre-processed frames pass through the trained model to predict the quality of guava for each individual frame.

4.3.5 Web Application Module :-

After developing and selecting the best model, the next step would be to create an easy-to-use web app. we created the web app using Flask, which is a web framework written in Python. For the environment, we used Visual Studio Code (VS Code) and installed all the required libraries and frameworks, including Flask, TensorFlow, OpenCV, etc. so that the backend was equipped to handle video processing and model prediction workloads.

For the frontend, we have used HTML, CSS and Javascript for a clean, interactive UI that have three main features:

- Video Upload: to allow users to upload pre-recorded videos.
- Image Upload: to allow users to upload images.
- Live Capture: to allow users to predict live using webcam input.

By either uploading a video or selecting the live capture option, the user hands the work over to the backend. The backend processes the video, extracts frames, preprocesses frames, and sends the frames to the model for prediction tasks.

10 frames from each video were extracted and the model makes a prediction on each of the 10 extracted frames. Finally, we use majority voting to classify the quality of the video. The class with the most votes from the 10 predictions, is returned to the user as the final predictions.

4.4 Implementation Details

The Guava Quality Prediction project is a machine learning-based web application that takes video input and classifies the quality of guava fruits. After the user uploads videos,

the videos will be extracted into individual frames, and different deep learning models will be trained on each frame. After reviewing the performance of some of these models, the best model will be selected, and we will develop a user interface to make the project suitable for real-life uses. The ultimate goal of the project is to develop a scalable system that will be able to be used for at least the quality analysis of other fruits in the future. The architecture of the project follows a web-based deep learning architecture. The backend is implemented with the Python programming language, while the frontend consists of external markup languages HTML, CSS, and Javascript. The system works by taking user runtime input which will be processed on the backend, and then predictions will be made with the final deep learning model.

The complete system was implemented as a configuration of several independent modules that make up a single pipeline that can automate guava quality classification using video input. The system architecture is composed of steps to extract frames, perform deep learning inference, aggregate predictions, and user interaction with a web-based interface.

Our implementation began with our dataset creation, which involved capturing 360° videos of guava samples spanning three quality grades, and subsequently, preprocessing and data augmentation were completed before the frames were input into the various deep learning models we trained in the previous phase. In terms of which model architecture framework to deploy, based on our assessment which re-evaluated different models against similar dimensions, we decided to deploy MobileNetV2. This decision was based primarily on the trade-off among accuracy and inference speed.

Once we had a trained model, we embedded it into a Flask web app. The back-end will process video uploads from users, which will then be broken down frame-by-frame using OpenCV and real-time inference will be made on each frame using our trained MobileNetV2 model. The predicted quality class of the guava fruit was determined by utilising a majority voting approach among the predictions of the individual frames.

HTML and CSS were used to develop the front-end of the application that has functionality for users to upload their guava videos and view the results. The updates to the application ensure that model loading and predictions are made efficiently, without users noticing a significant loading time. T4 GPU runtime from Google Colab was used

in the training of the models to lower the training time comparison to CPU and provide increased performance.

All component parts of the system, including frame extraction, model inference, result aggregation, and interaction with the user were tested to ensure a cohesive integration of each component and reliable functionality within different user inputs. The application implementation is successful in demonstrating that theoretical design can be realized in a working, real-time decision support system that would be usable in the agriculture domain.

4.4.1 Backend Functionality

App.py controls most of the back-end functionality for the system. This is the main Python script that routes all HTML pages through the Flask framework. It handles other functionalities such as testing, rapid page redirects, and emailing functionalities. A temporary directory called Uploads is used to temporarily store video files before they are processed. The core logic for the backend, such as convert video to frames, preprocessing, and prediction is also handled by this script.

Key Backend Features: -

1. Flask Framework Setup

The backend is comprised of Python's Flask framework. We initialize the application with:

```
app = Flask(__name__)
```

Every HTML file is routed to the app.py file. This file will help us to carry out the server-side actions for the web app.

2. Model Loading

The specified .h5 model file—MobileNet, which was selected after going through 13 different trained models is loaded. If the model file did not exist or was not loaded, the application will log appropriately.

3. Video Upload Handling

The video upload will be handled with the Flask route: /predict. The uploaded video files are saved temporarily in the Uploads directory for processing. To account for performance and reliability, the maximum video size is 50 MB.

4. File Types Allowed

The application accepts the following file types:

Videos: .mp4, .avi, .mov, .mkv, .webm

Images: .jpg, .jpeg, .png, .gif

5. Routes

The backend has the following routes:

- / → Home Page

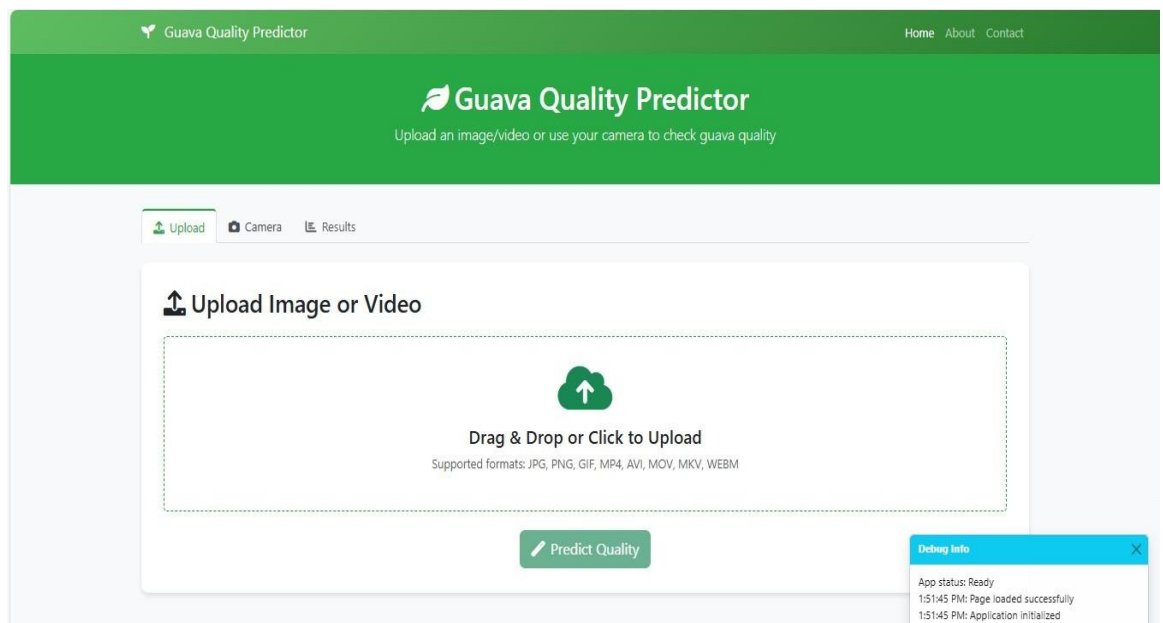


Fig. 16 Home Page of Application

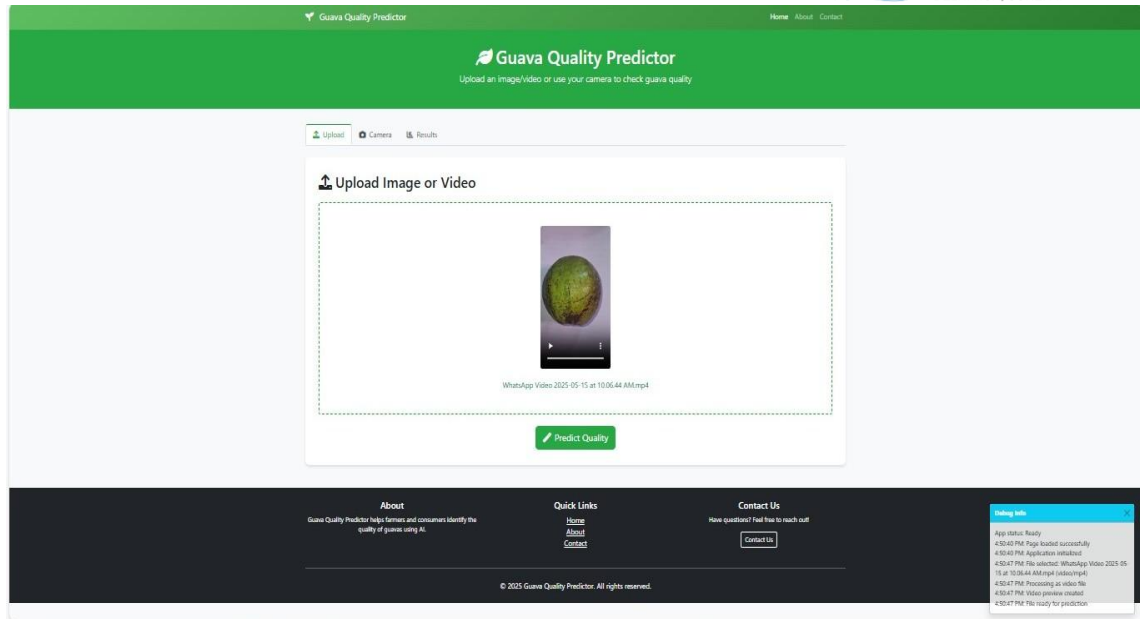


Fig. 17 Uploading video/images

- /about → About Page

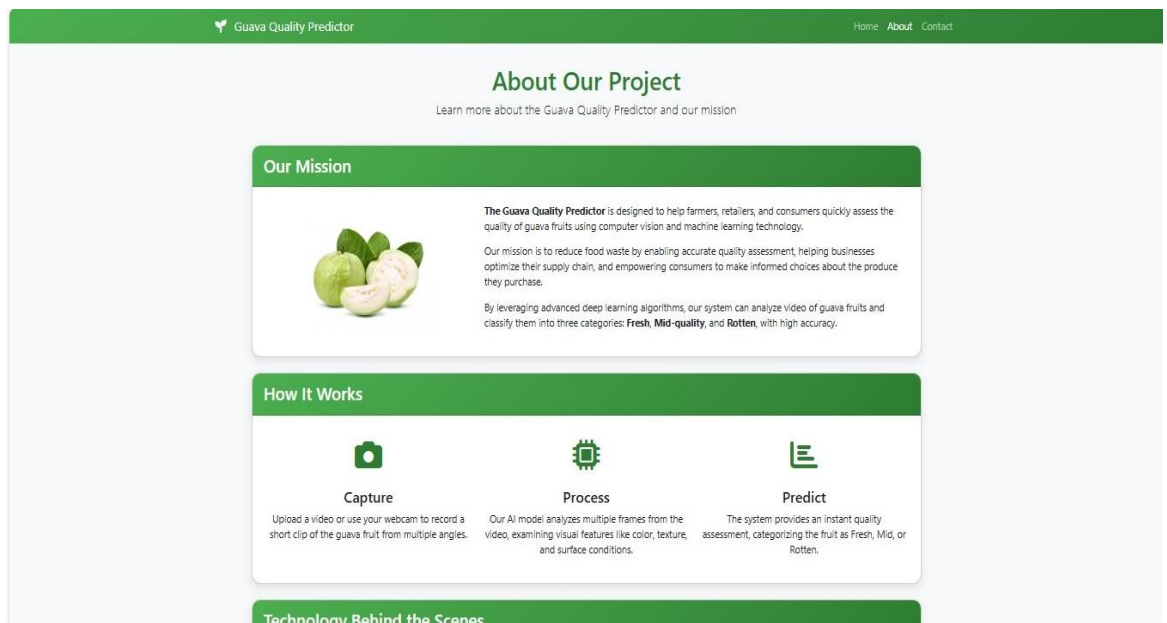


Fig. 18 About Page

- /contact → Contact Page

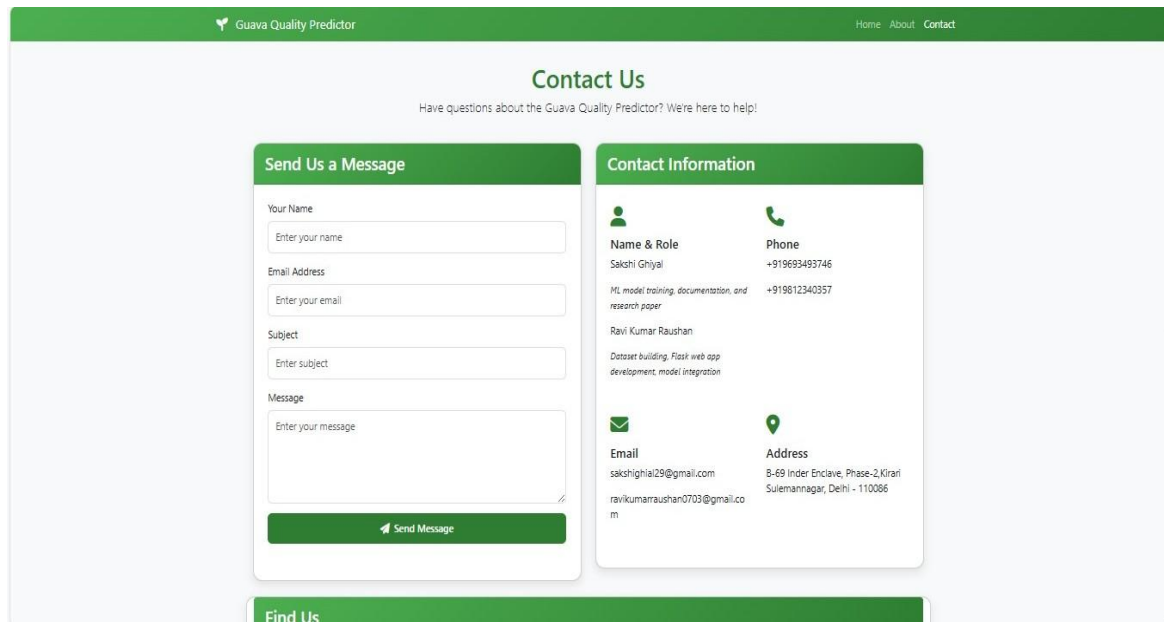


Fig. 19 Contact Page

- /submit-contact → Handles submission of contact form and sends the message to a built email system
- /predict → Handles file upload (image/video) and returns prediction as an output

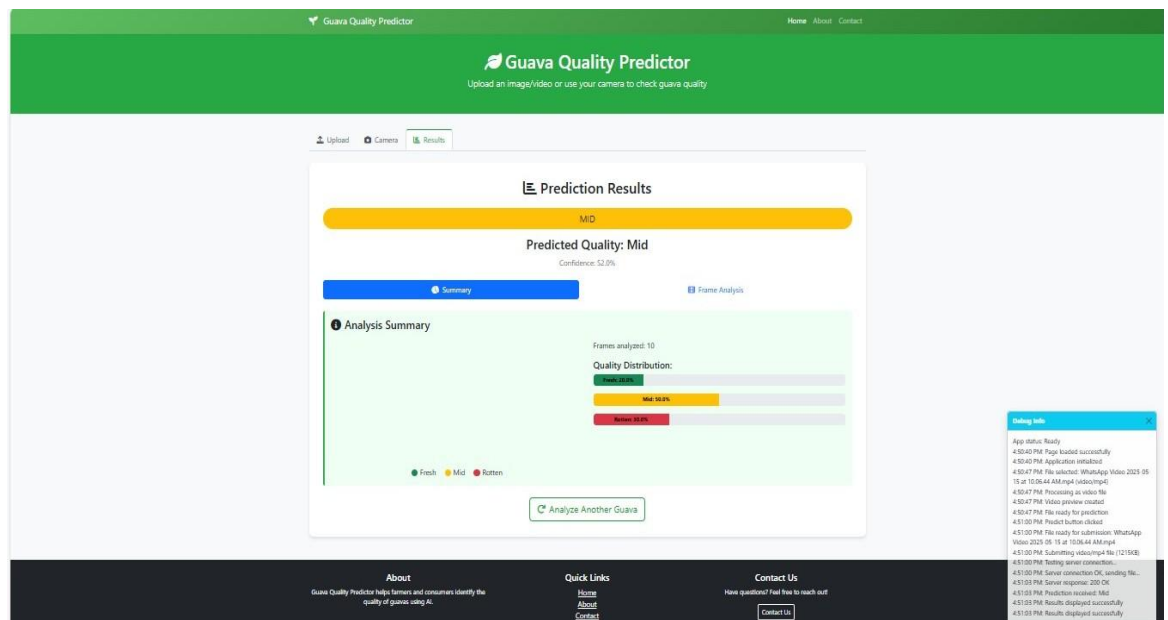


Fig. 20 Prediction result summary

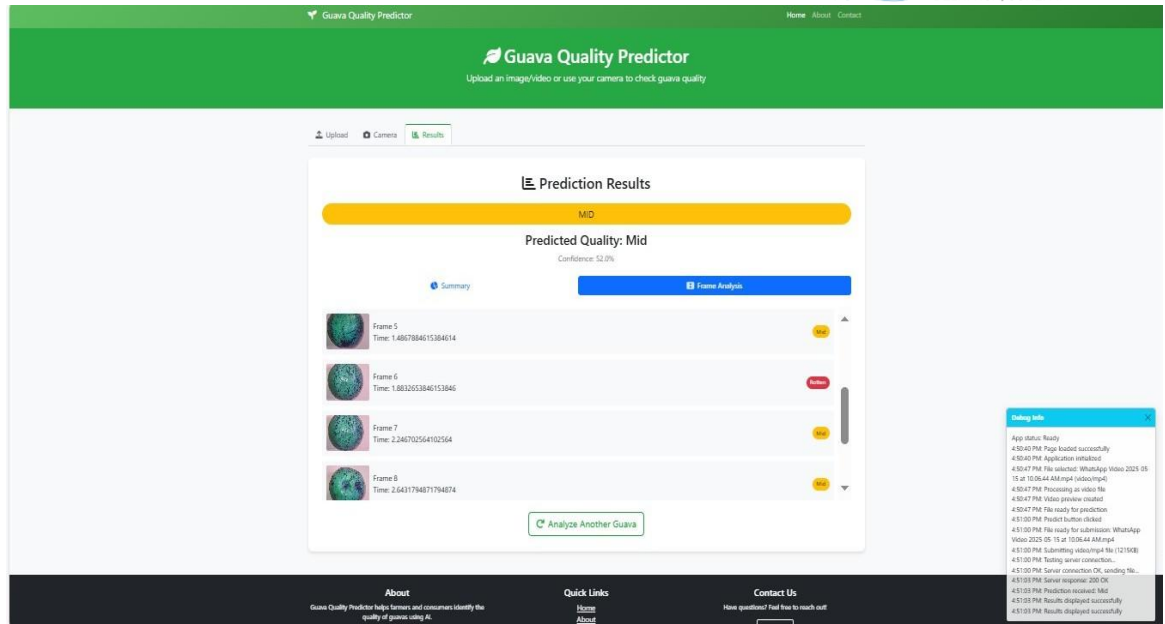


Fig. 21 Prediction result Analysis by frame

6. Frame Extraction

The video input type is using OpenCV to extract frames every 1-2 seconds. We extract 10 frames per video to be representative yet still efficient for the prediction.

7. Frame Preprocessing

Each frame which is extracted is resized to 224x224 for input to the MobileNet model. In the case of video predictions, the predict_video() method extracts all frames and takes the average prediction.

8. Model Prediction

- Image Prediction: make use of the predict_image() method.
- Video Prediction: make use of the predict_video() method. It predicts every frame, and the final output is determined by majority vote over 10 frames.

9. Prediction Response

After prediction, the Flask backend provides a structured JSON response with:

- prediction (final class label)
- confidence
- file_type (image vs video)
- class_counts (number of frames per predicted class)
- class_percentages (percentage of predictions per class)

CHAPTER V

RESULTS AND ANALYSIS

In this chapter, we analysed the performance of 13 deep learning models trained by us. After implementing each model in detail, we evaluated the models based on metrics such as accuracy, loss, confusion matrix and inference speed. Some models such as VGG16 showed the highest accuracy but their speed was slow. That is why we used MobileNetV2 model for final development as its accuracy was good and its speed was also good compared to other models. In this chapter, result summary, reason for final model selection and prediction output of web app are discussed.

5.1 Results of Different Models / Techniques

Below I have given the performance of all 13 deep learning models and each model is run for 50 epochs and then based on accuracy, speed, and deployment suitability, we want to see which is the best model for development among the models given below.

Table 10. Validation accuracy for models trained on GPU per epoch

Model Name	Number of Epochs					T4 GPU Runtime(s)
	10	20	30	40	50	
VGG16	94.88	96.11	96.77	96.77	96.78	1439.94
MobileNet	96.22	96.66	96.66	96.66	96.66	1228.60
DenseNet201	96.33	96.33	96.33	96.44	96.44	1541.89
Inceptionv3	93.77	95.11	95.33	95.55	95.56	1600.73
EfficientNetB0	96.22	96.22	96.55	96.77	96.77	1472.03
NASANetMobile	95.33	95.44	95.55	95.77	95.78	1540.09
ResNet 50	96.22	96.55	96.55	96.55	96.67	1379.02
ResNet101	96.22	96.33	96.44	96.66	96.67	1428.18
Xception	95.33	95.77	96.00	96.00	96.44	1328.47
NER (Soft Voting)	97.33					-
NER (Hard Voting)	97.22					-

Table 11. Validation accuracy for models trained on CPU per epoch

Model Name	Number of Epochs					CPU Runtime(s)
	10	20	30	40	50	
VGG16	94.55	95.11	95.33	95.33	95.33	12h+
MobileNet	96.11	96.22	96.22	96.22	96.22	5057.62
DenseNet201	96.22	96.33	96.33	96.33	96.33	23347.67
Inceptionv3	94.11	95.44	95.55	95.56	95.56	12004.58
EfficientNetB0	96.11	96.33	96.33	96.33	96.33	7206.66
NASNetMobile	94.66	95.11	95.11	95.11	95.11	5566.91
ResNet 50	96.33	96.44	96.44	96.44	96.44	12247.64
ResNet101	95.11	96.11	96.11	96.11	96.11	25120.62
Xception	95.88	96.11	96.11	96.11	96.11	16008.80

Table 12. Validation accuracy of Ensemble Models

S. No.	Ensemble Model	Strategy	Val accuracy	T4 GPU Runtime (in seconds)
1	NASNet + ResNet50 + EfficientNetB0	Soft Voting	97.33%	-
2	NASNet + ResNet50 + EfficientNetB0	Hard Voting	97.22%	-
3	NASNet + ResNet50 + EfficientNetB0	Feature Fusion	96.44%	1557.98
4	VGG16 + ResNet50 + MobileNetV2	Feature Fusion	96.44%	1519.00

As we evaluated the accuracy, speed, and deployment suitability of all the above models, we found that VGG16 model has the best accuracy of 96.78%. However VGG16 is a heavy model hence slow for real time computations.

Next, we have 3 such models whose accuracy is good as well as speed.

MobileNetV2, EfficientNetB0, NASNetMobile are these three models whose accuracy is good as well as their speed is also good.

Now after evaluating all the 13 models, MobileNetV2 was selected for deployment because it is a perfect balance of accuracy and real time speed.

Final Deployed Model : MobileNetV2

5.2 Performance Comparison

In this section we did overall comparison of models. As we saw in the result section above, a table of all the models was made in which their performance was shown. We will analyze them and here we will see a description of total 13 models and discussed their performance.

In this study, several models of deep learning were implemented and tested in order to forecast the quality of guava fruit into three distinct classes: Fresh, Mid, and Rotten. The performance of every model was evaluated in terms of validation accuracy, validation loss, and training time in CPU and GPU modes. Among the models that were evaluated, VGG16 was the most accurate with 96.78% at 50 epochs but also the longest training time - 18753.85 seconds on CPU and 1439.94 seconds on GPU- so it might not be the best option for real-time deployment. MobileNetV2, on the other hand, had the accuracy of 96.67% with significantly lower GPU training time of 1228.60 seconds and the lowest validation loss of 0.0980, and thus it is another candidate that can be deployed.

DenseNet201 also had good accuracy (96.44%) and low loss (0.0983) but took longer to train (1541.89 seconds on GPU). ResNet50, ResNet101, and Xception all had 96.67% and 96.44% accuracy, respectively, and decent GPU times (1328 to 1428 seconds), all of which are in the high-performance category. InceptionV3 and NASNetMobile had lower but similar accuracy rates (95.56% and 95.78%) with moderate training time (1600 seconds), while EfficientNetB0, although quick (1472.03 seconds on GPU), only managed to score 92.33% and hence was less appealing for precise classification purposes.

Table 13 Class-wise Comparison of Precision, Recall and F1-score

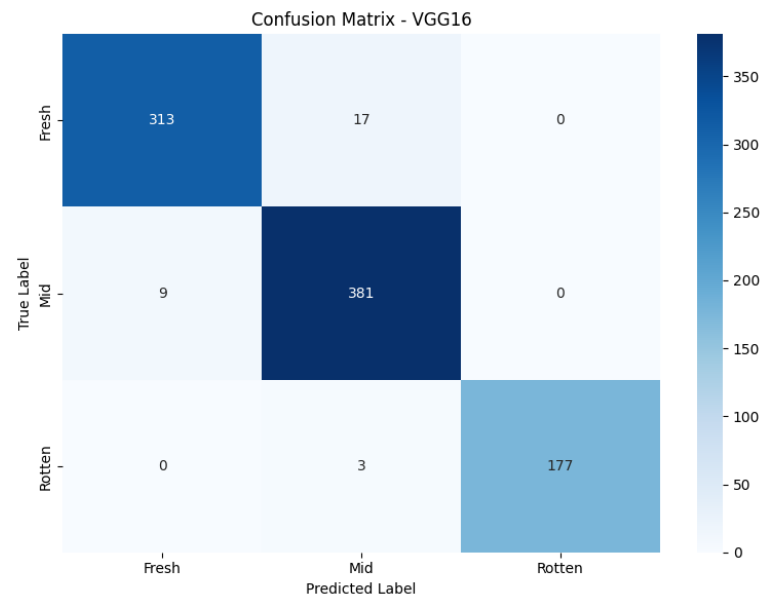
Model Name	Fresh			Mid			Rotten		
	<i>Precision</i>	<i>Recall</i>	<i>F1-Score</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-Score</i>	<i>Precision</i>	<i>Recall</i>	<i>F1-Score</i>
VGG16	0.97	0.95	0.96	0.95	0.98	0.96	1.00	0.98	0.99
MobileNet	0.95	0.97	0.96	0.97	0.96	0.96	0.99	0.98	0.99
DenseNet 201	0.93	0.98	0.95	0.98	0.94	0.96	1.00	1.00	1.00
Inception v3	0.97	0.94	0.95	0.93	0.97	0.95	0.98	0.96	0.97
EfficientNetB0	0.83	1.00	0.91	1.00	0.83	0.90	0.99	1.00	1.00
NASNet Mobile	0.97	0.94	0.96	0.94	0.96	0.95	0.98	0.97	0.97
ResNet 50	0.94	0.98	0.96	0.98	0.94	0.96	0.98	0.99	0.99
ResNet101	0.98	0.94	0.96	0.94	0.98	0.96	0.99	0.98	0.99
Xception	0.95	0.96	0.96	0.96	0.96	0.96	0.99	0.98	0.99
NER (Soft Voting)	0.97	0.96	0.96	0.96	0.97	0.97	1.00	1.00	1.00
NER (Hard Voting)	0.97	0.96	0.96	0.96	0.97	0.97	1.00	0.99	1.00
NER (Feature Fusion)	1.00	0.91	0.95	0.93	0.99	0.96	0.97	0.99	0.98
VRM (Feature Fusion)	0.99	0.92	0.95	0.93	0.99	0.96	0.99	0.99	0.99

Overall, upon comparison of accuracy, training time, and validation loss, the choice of the final deployed model came down to MobileNetV2 since it provided the best tradeoff between performance and efficiency, particularly in the context of a real-time web application using Flask where speed and reliability are paramount. Secondly, the relatively low resource requirements of MobileNetV2 made it fit easily into web applications without requiring high-end hardware.

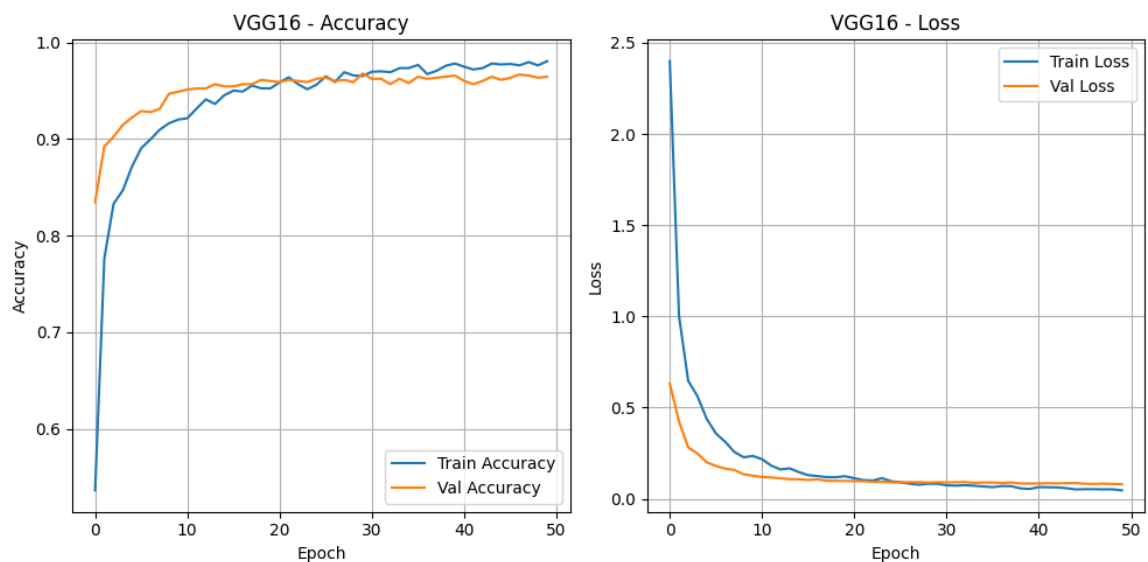
It showed consistent performance even with video input, making accurate predictions on different frames through majority voting method. The shorter GPU training time also equates to quicker retraining or model update in future runs. Based on all the major

parameters - computational expense, accuracy, training time, and model size - MobileNetV2 proved to be the most pragmatic and production-friendly option for our guava quality prediction system.

Performance of VGG16



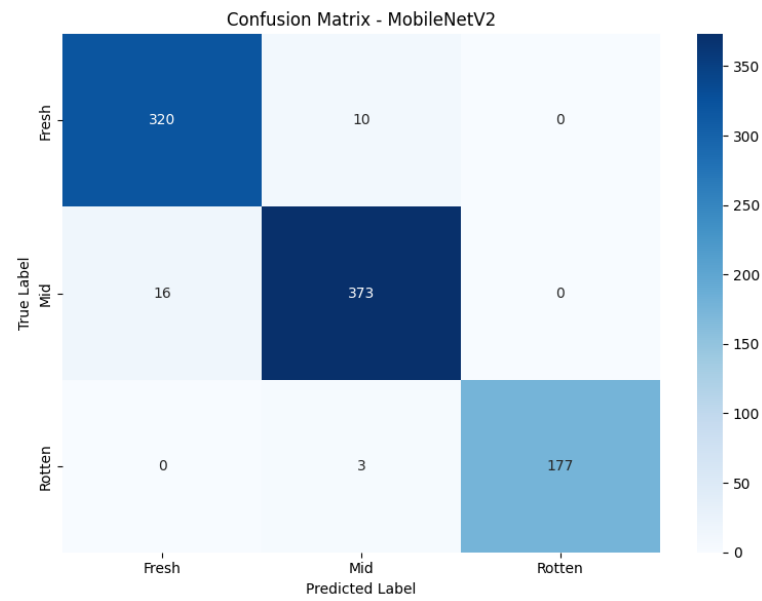
(a)



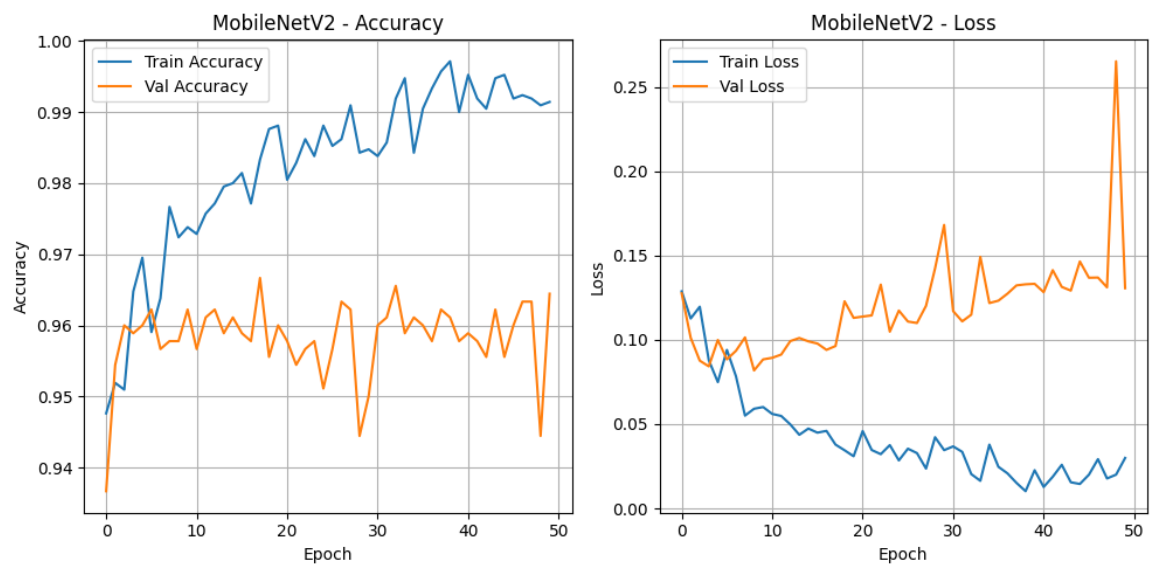
(b)

Fig. 22 (a) Confusion Matrix of VGG16 and (b) Training/Validation accuracy – VGG16

Performance of MobileNetV2



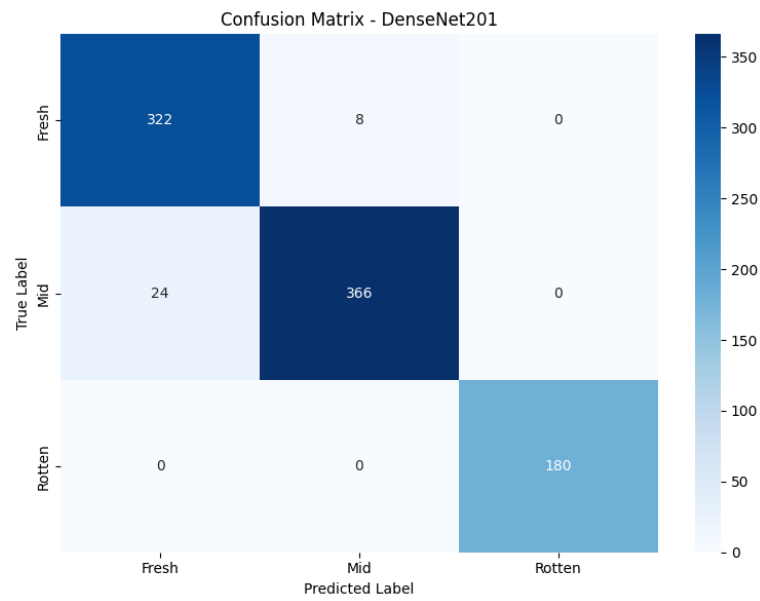
(a)



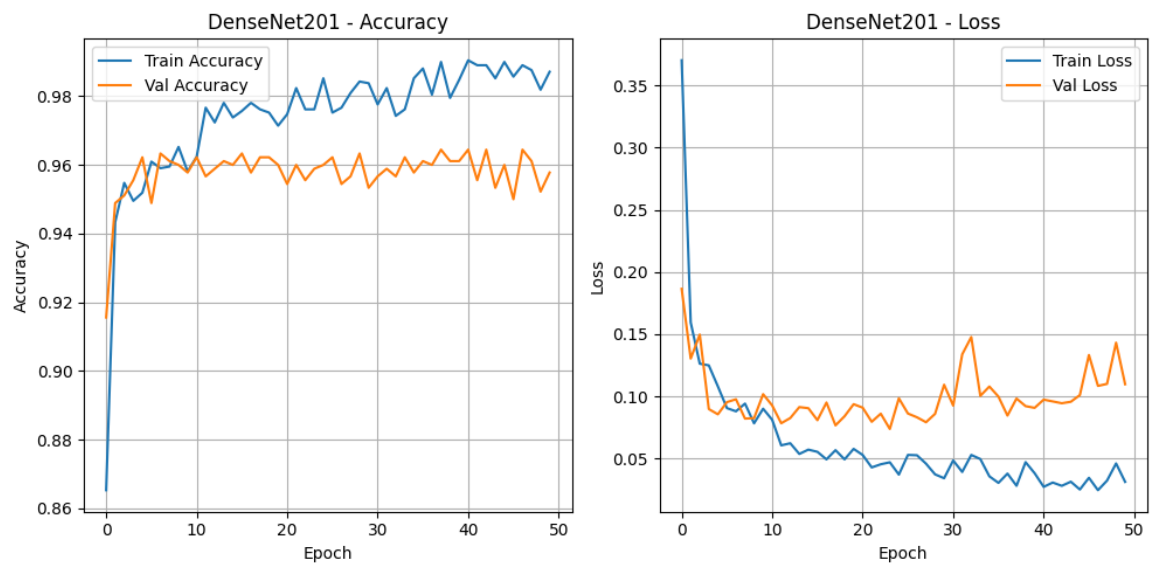
(b)

Fig. 23 (a) Confusion Matrix of MobileNetV2 and (b) Training/Validation accuracy – MobileNetV2

Performance of DenseNet201



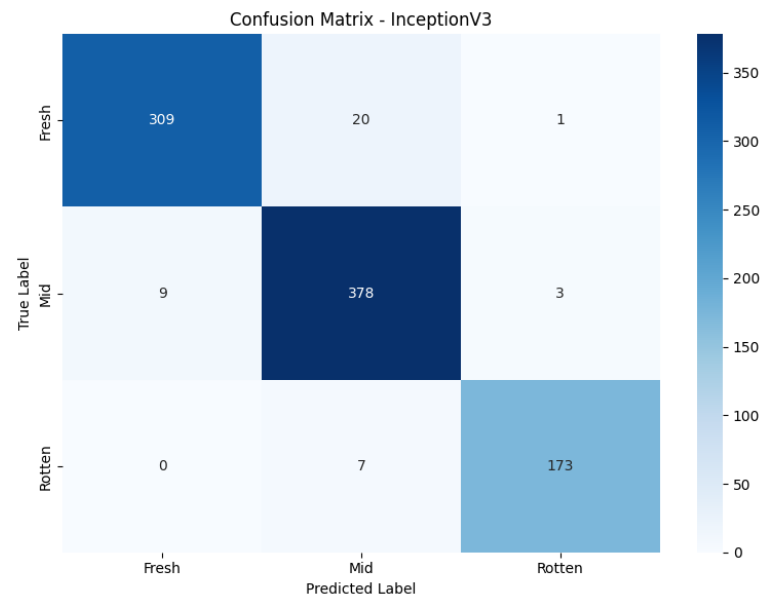
(a)



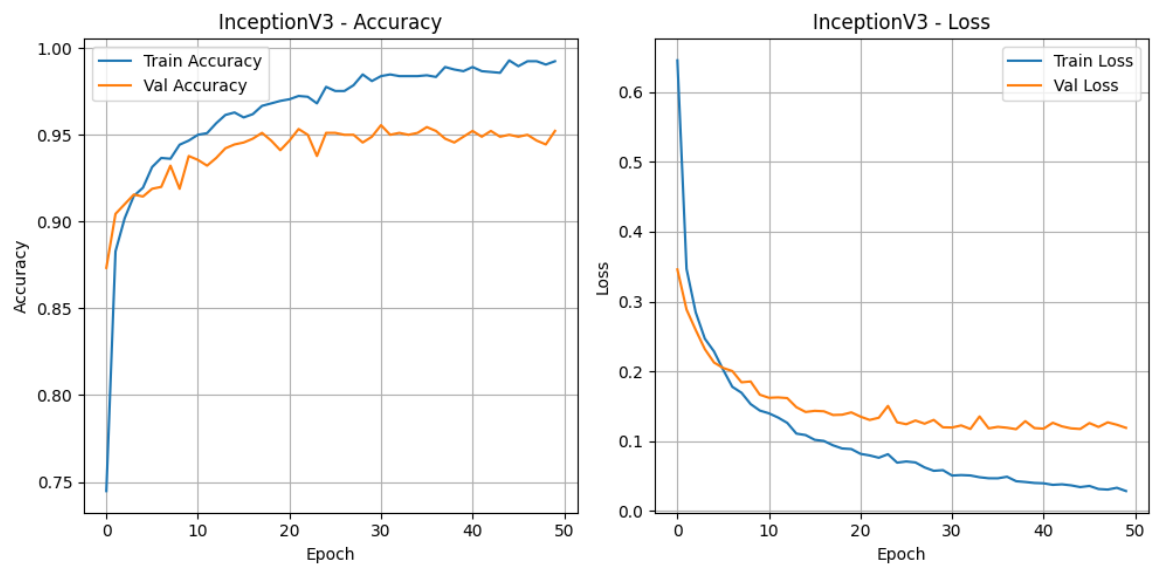
(b)

Fig. 24 (a) Confusion Matrix of DenseNet201 and (b) Training/Validation accuracy – DenseNet201

Performance of InceptionV3



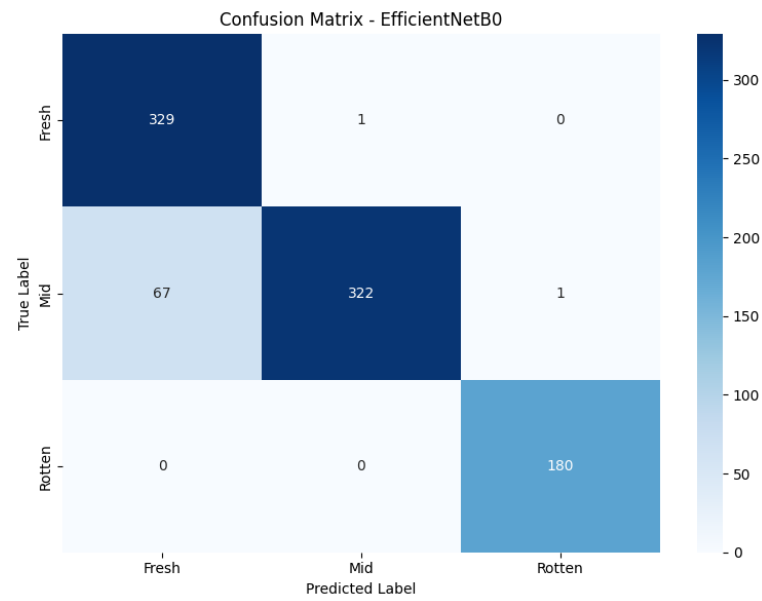
(a)



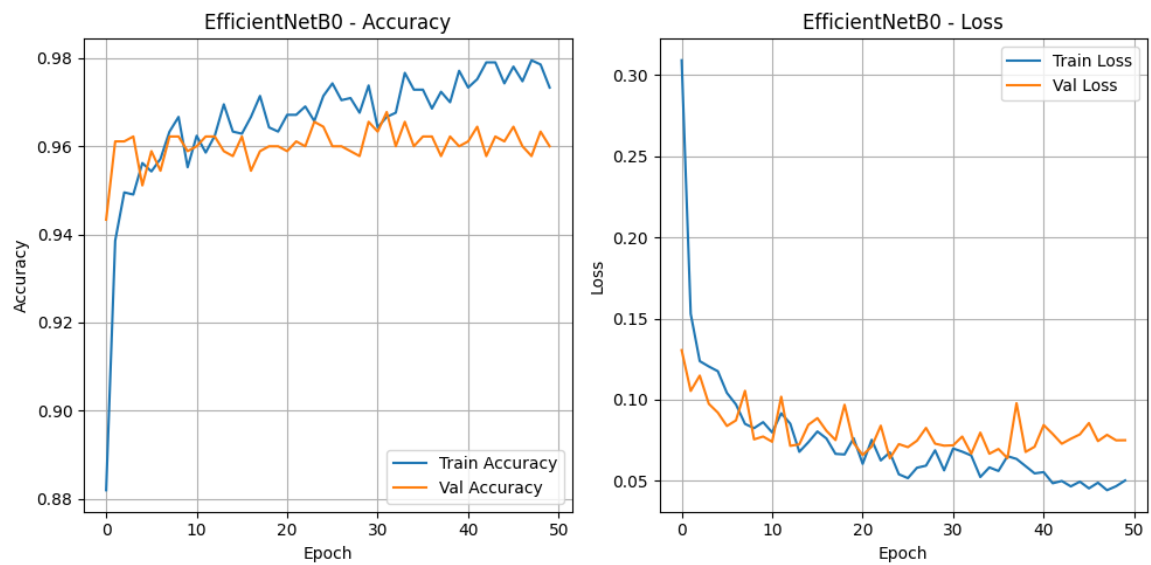
(b)

Fig. 25 (a) Confusion Matrix of InceptionV3 and (b) Training/Validation accuracy – InceptionV3

Performance of EfficientNetB0



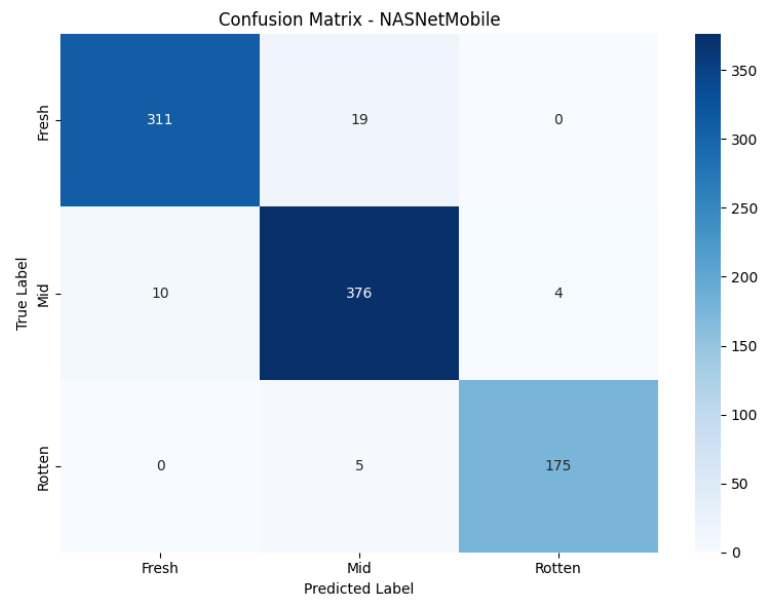
(a)



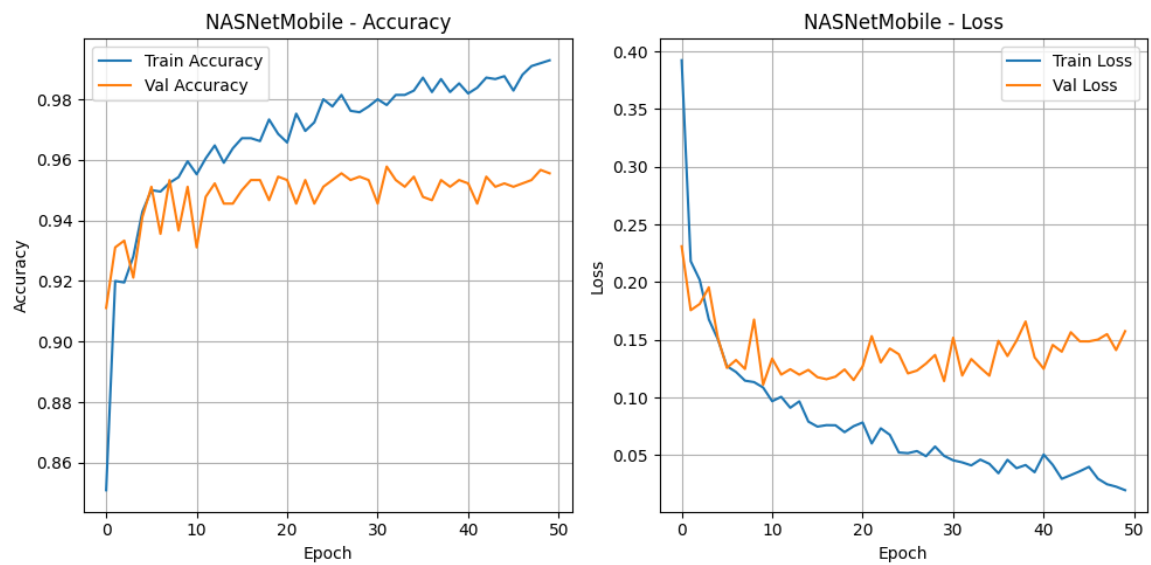
(b)

Fig. 26 (a) Confusion Matrix of EfficientNetB0 and (b) Training/Validation accuracy – EfficientNetB0

Performance of NASNetMobile



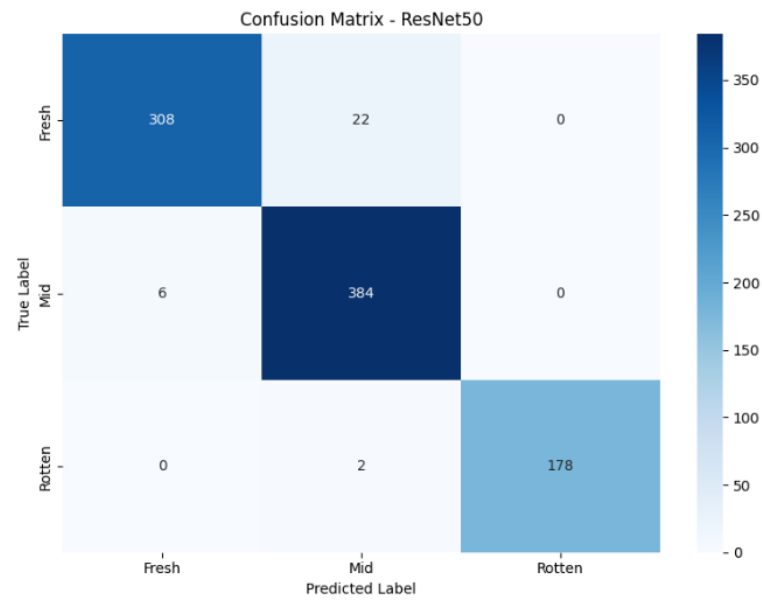
(a)



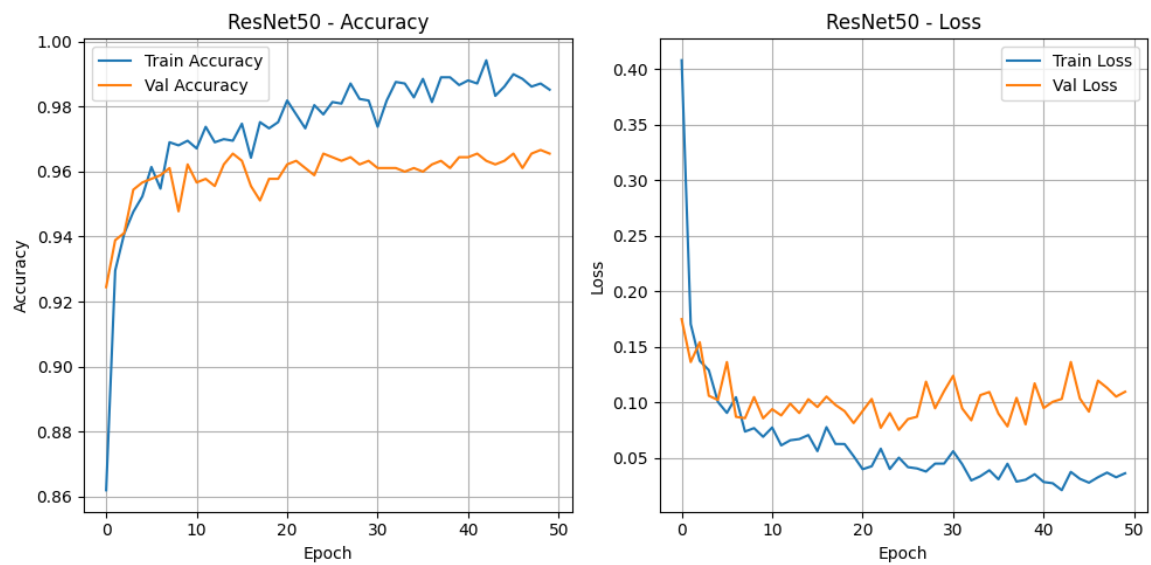
(b)

Fig. 27 (a) Confusion Matrix of NASNetMobile and (b) Training/Validation accuracy – NASNetMobile

Performance of ResNet50



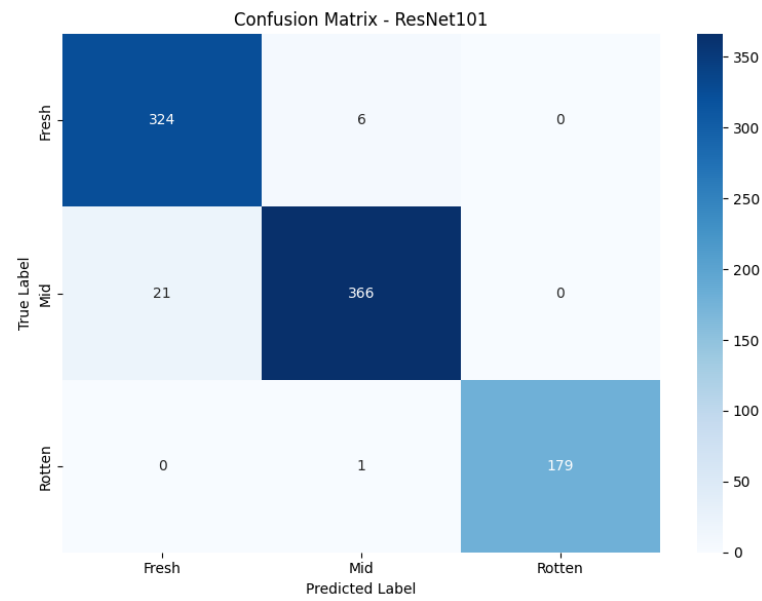
(a)



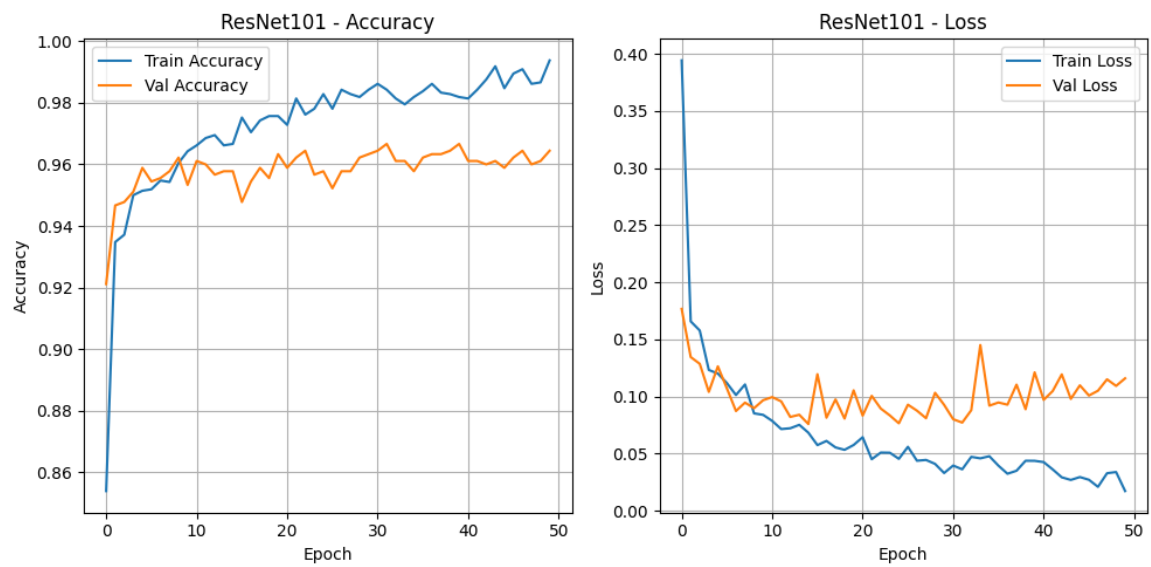
(b)

Fig. 28 (a) Confusion Matrix of ResNet50 and (b) Training/Validation accuracy – ResNet50

Performance of ResNet101



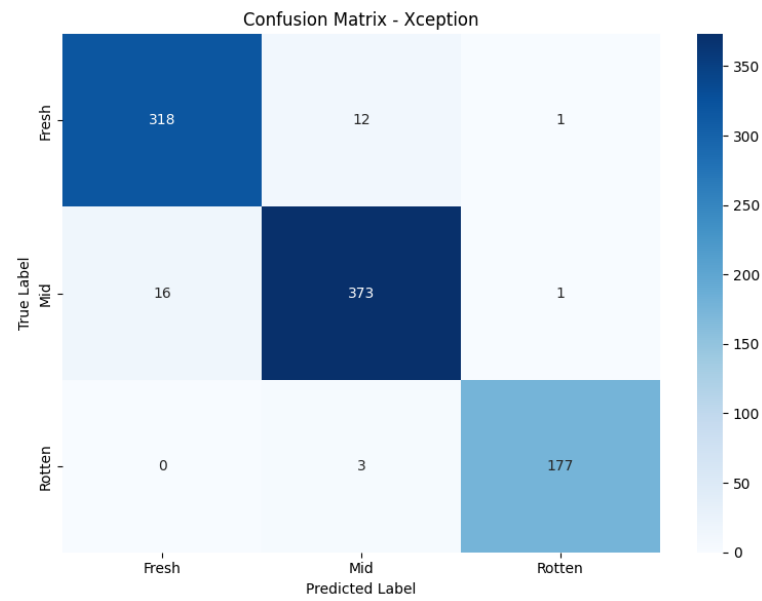
(a)



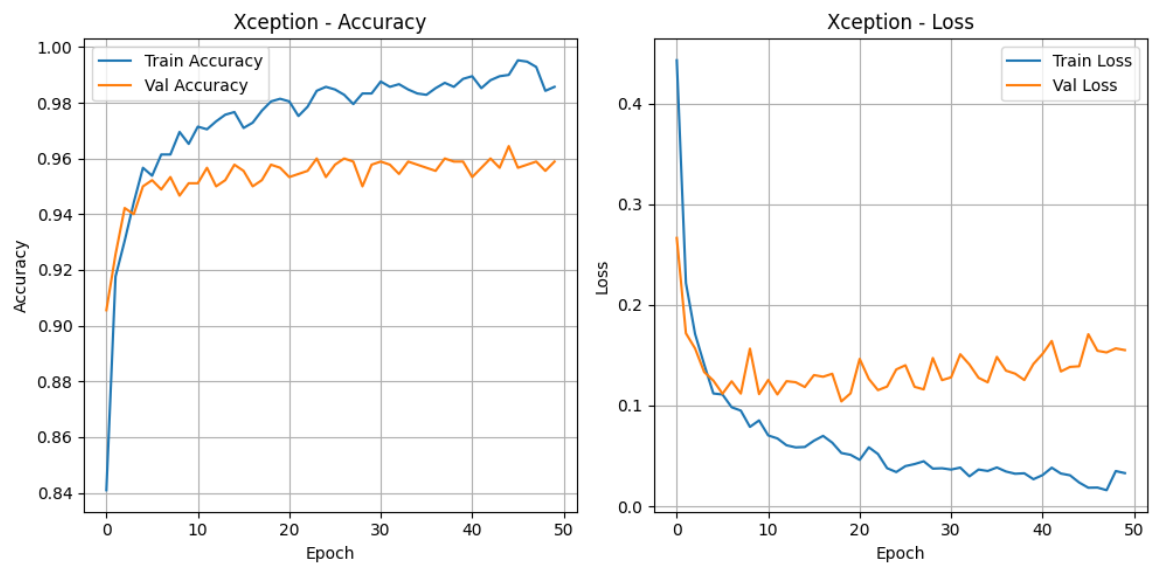
(b)

Fig. 29 (a) Confusion Matrix of ResNet101 and (b) Training/Validation accuracy – ResNet101

Performance of Xception



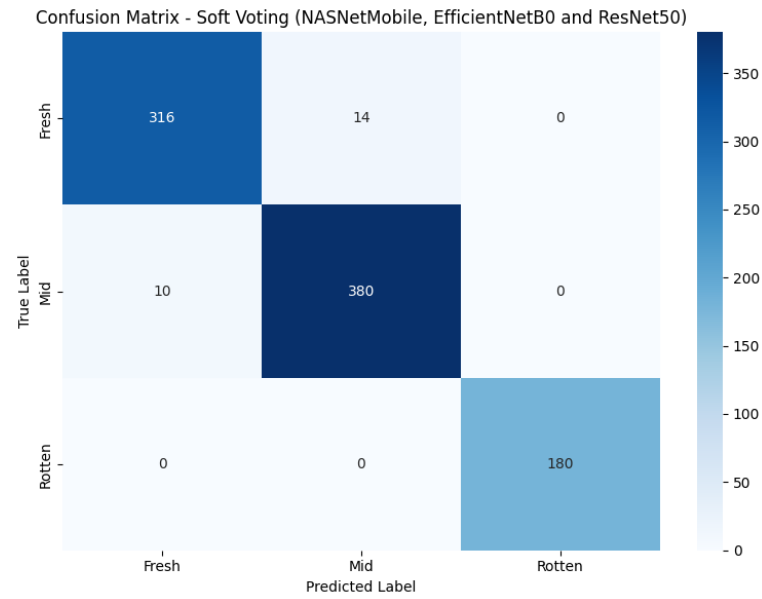
(a)



(b)

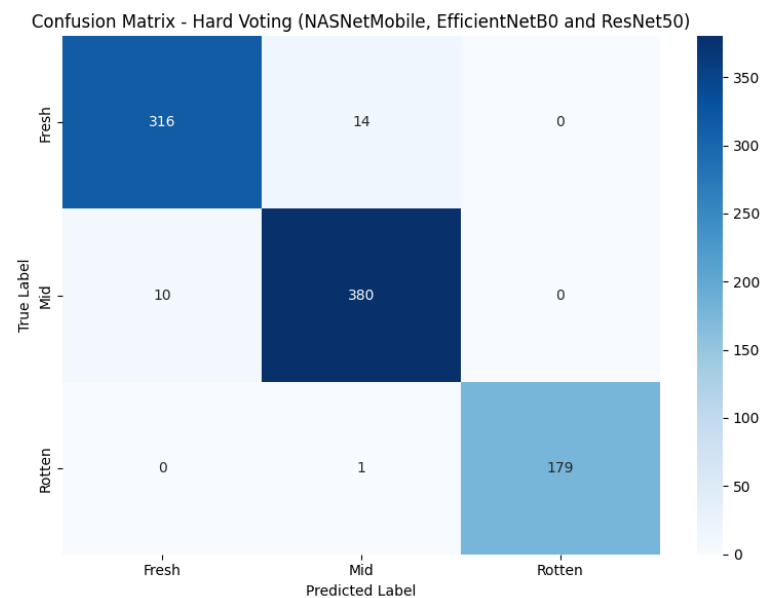
Fig. 30 (a) Confusion Matrix of Xception and (b) Training/Validation accuracy – Xception

Performance of NER (Soft voting) model



(a)

Performance of NER (Hard voting) model

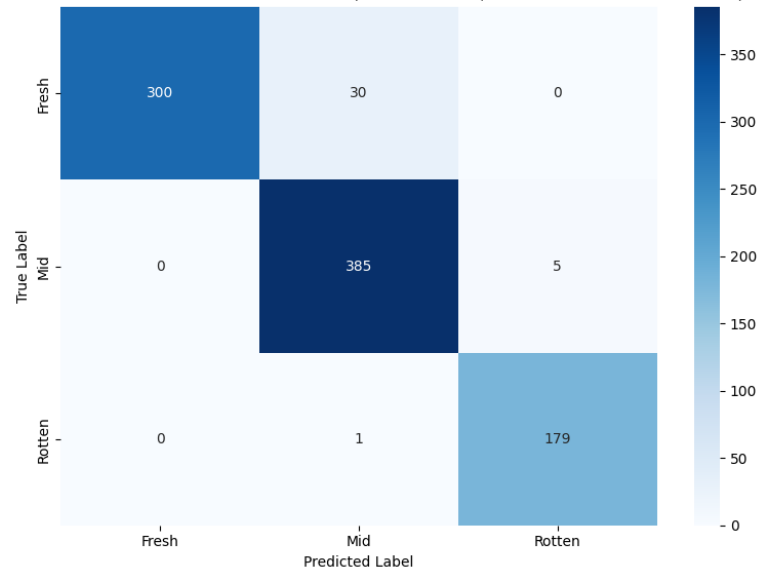


(b)

Fig. 31 Confusion Matrix of (a) NER (Soft Voting) model and (b) NER (Hard Voting) model

Performance of NER (Feature Fusion) model

Confusion Matrix - Feature Fusion Ensemble (NASNetMobile, EfficientNetB0 and ResNet50)



(a)

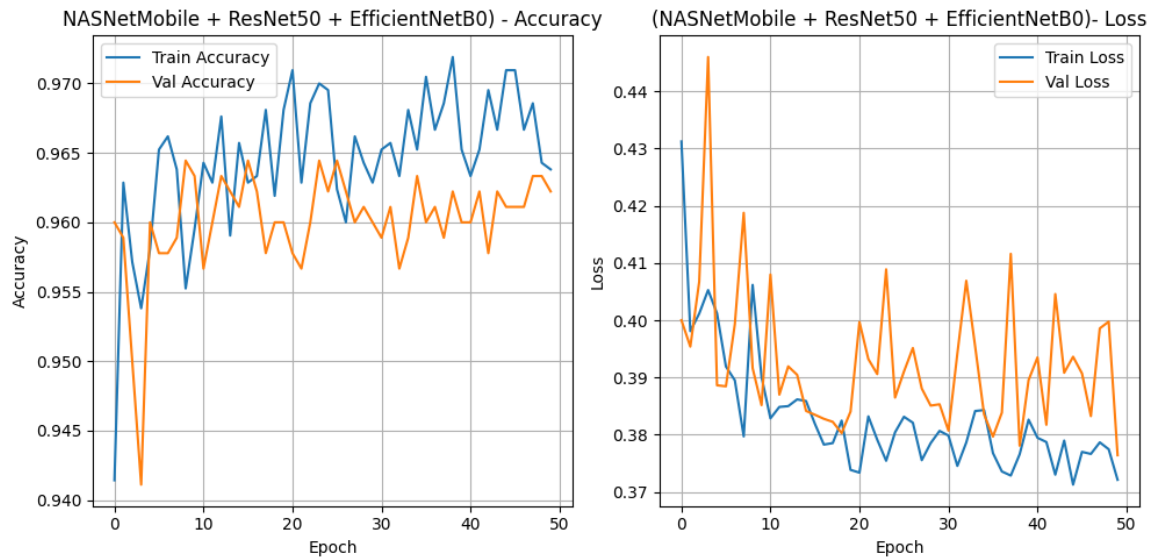
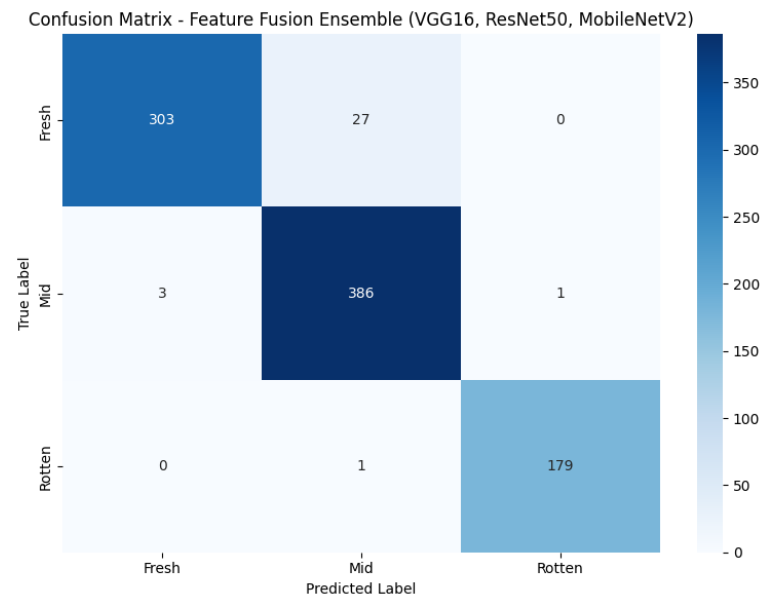
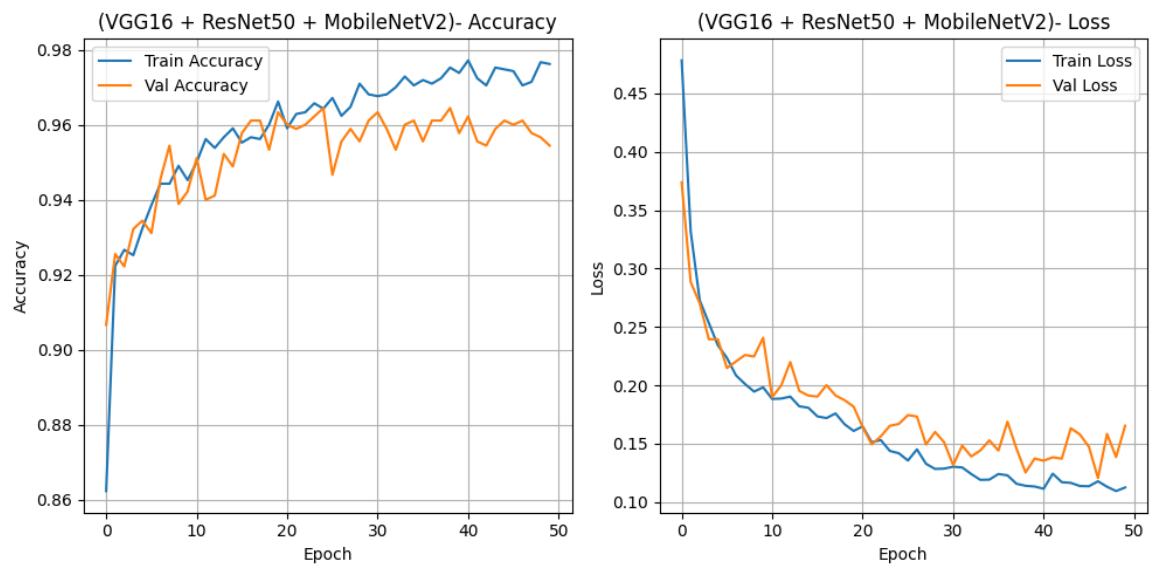


Fig. 32 (a) Confusion Matrix of NER (Feature Fusion) model and (b) Training/Validation accuracy – NER (Feature Fusion) model

Performance of VRM (Feature Fusion) model



(a)



(b)

Fig. 33 (a) Confusion Matrix of VRM (Feature Fusion) model and (b) Training/Validation accuracy – VRM (Feature Fusion) model

After analyzing all models based on accuracy, validation loss, and GPU training time, MobileNetV2 emerged as the most balanced model. It provided high accuracy (96.67%), low validation loss, and the fastest inference speed among top-performing models, making it ideal for real-time deployment in the web application.

5.3 Discussion

This study aimed to devise a deep learning system, based on video input, for classifying guava fruit in three quality levels: Fresh, Mid, and Rotten. The aim was to create both an accurate system and a system that could be reasonably utilized in practice by users in a real-time decision-making application with an intuitive web-based interface. The first step was dataset collection and preparation.

There was no public dataset, so we took videos of all three guava categories and obtained as many frames as possible. To diversify and increase dataset size, we temporally augmented the extracted video frames. This step was critical since the performance of any deep learning model will heavily depend on quality and balance in the training dataset.

In the second phase, the model training phase, we trained a total of 13 different models which included 9 pretrained CNN architectures (VGG16, MobileNetV2, ResNet, and others), and 5 ensemble option (soft voting, hard voting, feature concatenating). These models were trained for up to 50 epochs and evaluated on validation accuracy, validation loss, and time to train.

In our analysis, we found that some models, specifically VGG16 and ResNet101, had very high accuracy. However, VGG16 and ResNet101 were too heavy and slow to use in a real-time application. MobileNetV2 produced nearly the same accuracy at 96.67% accuracy while being lightweight and fast. Thus, it was more appropriate to use for deployment in a web environment.

The final system we built utilized the Flask web framework providing a clean front-end interface for users to upload videos. The backend handled the videos, extracted frames with OpenCV, and passed the frames to the MobileNetV2 model. The final output was generated through majority voting on the frame predictions, which was then printed back to the user.

Overall, the project proved that deep learning could be successfully harnessed for real-world agricultural use cases. It also showed that it is not just about the most accurate model, but a balance between performance, speed and ability to deploy. The web application we built could serve as a working prototype that can be adapted and expanded for farms, supply chains, and marketplaces to quickly and reliably assess the quality of fruit.

4.5 Challenges Faced

Although deep learning models produced promising accuracy for guava quality classification, there were a number of challenges met through the course of this study:

1. **Small Data Set:** The video data set contained 60 samples with a small number of unique frames. Whilst augmentation helped create a larger data set, the limited diversity of the original data set may hinder the models ability to generalize.
2. **Limited Hardware / GPU:** The training of deep models like DenseNet201, Xception, and ensemble architectures, requires high computational resources. The implementation of Google Colab resulted in hardware limitations for runtime, particularly with CPU, which impeded training as the training sessions were prematurely terminated or left incomplete.
3. **Class Imbalance:** The data set had an imbalanced class ratio which limited representations of the category 'rotten'. This can impair the ability of the model to learn any features belonging to marginalized classes, leading to biased prediction.
4. **Extracted Redundant Features from Video Frames:** The frame extraction method of obtaining several images from the same video would create redundant images or very similar images which would contribute to overfitting and reduce robustness of model.
5. **Accuracy Vs. Runtime:** The models with the best accuracy with regards to training time and parameter size were VGG16. Its long training time and parameter size would not be practical for real time without optimization.
6. **The Risk of Overfitting in Deep Networks:** The deep architectures trained on relatively small augmented data showed overfitting behavior when longer training durations were employed, especially when trained for long epochs with insufficient regularization.

CHAPTER VI

CONCLUSION AND FUTURE WORK

6.1 Conclusion

This study focused on utilizing 360° videos of guava to classify them into three major categories: fresh, mid and rotten. Various deep learning architectures were trained and tested on the frames extracted from the video dataset. This process ensured a comprehensive evaluation of guava quality which was often missed in the case of standalone images or images captured from only one angle. It was found that ensemble strategies work better than individual pretrained models. Among all the models, soft voting ensemble NER achieved the highest validation accuracy of 97.33%. Hard voting ensemble of NER also showed promising results with an accuracy of 97.22%. The soft voting ensemble of NER especially showed exceptional results by obtaining 100% accuracy for the frames under rotten category. Though, the feature fusion ensemble of VRM and NER lacked in comparison to other ensemble strategies but they showed great potential when compared to individual models, achieving same accuracy as that of the second best performing individual models. The best performing individual models were the pretrained models like VGG16 followed by MobileNetV2 and ResNets. All of these findings suggest the scope of deep learning technologies in classifying fruits based on maturity grading. It suggests that incorporating 360° angle view of a fruit helps in a comprehensive assessment of the fruit as well as proves to be quality classification method. Future work could include increasing the dataset size and incorporating different varieties of guava. There is need for even deeper assessment due to the fact that some fruits graded as good quality on visual inspection may still have defects inside. Therefore, incorporating multimodal data along with video dataset can provide better analysis regarding the quality. This study can also be scaled for other fruits and integrated in real time scenarios to ease the process to crop maturity grading. Overall, this study underscores the potential of integrating deep learning networks with video dataset to revolutionize the crop assessment and quality grading.

6.2 Future Work

The present work provides many opportunities for future research and development:

1. **Dataset Expansion:** Collecting a higher volume and diversity of data with different lighting conditions, backgrounds, and many varieties of guava will help with model robustness and scalability.
2. **Multimodal Data Integration:** Incorporate algorithms that use multiple modalities to assess quality, e.g. visual data with near-infrared imaging or fruit firmness sensors to achieve a more complete quality assessment, as measuring for both internal and external defects is important.
3. **Real-time Deployment:** Evaluate how those models can be optimized for integration into edge devices or mobile applications in order to make this system useful to markets and farms.
4. **Enhancing Class Balance:** Future datasets will need to improve the distribution of quality by class in order to allow for fairer predictions, and to allow for reduced bias as well.
5. **Testing Advanced Ensemble Techniques:** Advanced ensemble techniques other than voting and feature fusion could be tested, including stacking and boosting.
6. **Transfer to Other Fruits:** We could transfer this framework to and validate it on other fruits, allowing us to check for generality and develop a more complete systematic method for multi-fruit quality classification.
7. **Estimate internal quality:** Potentially implement non-destructive internal quality estimating techniques (such as hyperspectral imaging) to allow us to classify the fruit holistically, beyond simply what is seen on the outside.

REFERENCES

- [1] D. P. Semwal et al., “Diversity distribution analysis of guava (*Psidium guajava* L.) populations in cultivated and wild habitats in the mid-hills of Uttarakhand, India,” *Agriculture*, vol. 14, p. 575, 2024. [Online]. Available: <https://doi.org/10.3390/agriculture14040575>
- [2] C. A. Pontikis, “*Psidium guajava* L. (Guava),” in *Trees IV. Biotechnology in Agriculture and Forestry*, Y. P. S. Bajaj, Ed. Berlin/Heidelberg, Germany: Springer, 1996, vol. 35.
- [3] S. Naseer, S. Hussain, N. Naeem, et al., “The phytochemistry and medicinal value of *Psidium guajava* (guava),” *Clin. Phytosci.*, vol. 4, p. 32, 2018. [Online]. Available: <https://doi.org/10.1186/s40816-018-0093-8>
- [4] ABC Fruits, *Guava: The Tropical Apple*, Tamil Nadu, India. [Online]. Available: <https://abcfruits.com/guava-the-tropical-apple/>
- [5] K. Prasad, S. Jacob, and M. Siddiqui, “Fruit maturity, harvesting, and quality standards,” in *Postharvest Biology and Technology of Tropical and Subtropical Fruits*, 2017. doi: 10.1016/B978-0-12-809807-3.00002-0.
- [6] P. Bakshi, “Maturity indices of Guava,” 2014.
- [7] Y. S. Voon, Y. Wu, X. Lin, and K. Siddique, “Performance analysis of CPU, GPU and TPU for deep learning applications,” *CICET*, vol. 16, no. 12, 2021.
- [8] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” in *Proc. ICLR*, 2015, pp. 1–14.
- [9] M. Sandler, A. Howard, M. Zhu, et al., “Mobilenetv2: Inverted residuals and linear bottlenecks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Salt Lake City, UT, USA, 2018, pp. 4510–4520. doi: 10.1109/CVPR.2018.00474.
- [10] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 2261–2269. doi: 10.1109/CVPR.2017.243.
- [11] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” in *Proc. IEEE Conf. Comput. Vis. Pattern*

Recognit. (CVPR), Las Vegas, NV, USA, 2016, pp. 2818–2826. doi: 10.1109/CVPR.2016.308.

[12] M. Tan and Q. V. Le, “EfficientNet: Rethinking model scaling for convolutional neural networks,” arXiv preprint arXiv:1905.11946, 2019.

[13] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, “Learning transferable architectures for scalable image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Salt Lake City, UT, USA, 2018, pp. 8697–8710. doi: 10.1109/CVPR.2018.00907.

[14] B. Zoph and Q. V. Le, “Neural architecture search with reinforcement learning,” in Proc. Int. Conf. Learn. Representations (ICLR), 2017.

[15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Las Vegas, NV, USA, 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.

[16] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Honolulu, HI, USA, 2017, pp. 1800–1807. doi: 10.1109/CVPR.2017.195.

[17] L. K. Hansen and P. Salamon, “Neural network ensembles,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 12, no. 10, pp. 993–1001, Oct. 1990. doi: 10.1109/34.58871.

[18] D. Opitz and R. Maclin, “Popular ensemble methods: An empirical study,” J. Artif. Intell. Res., vol. 11, pp. 169–198, 1999. doi: 10.1613/jair.614.

[19] Directorate of Marketing and Inspection, Fruits and Vegetables Grading and Marking Rules, Ministry of Agriculture & Farmers Welfare, Government of India, Faridabad, India, 2004. [Online]. Available: <https://dmi.gov.in>

[20] Directorate of Marketing and Inspection, Agmark Standards for Fruits and Vegetables under the Agricultural Produce (Grading and Marking) Act, 1937, Ministry of Agriculture & Farmers Welfare, Government of India, Faridabad, India. [Online]. Available: <https://agmarknet.gov.in>

[21] A. Mahmood, A. Tiwari, S. Singh, and S. Udmale, “Contemporary machine learning applications in agriculture: Quo Vadis?,” Concurrency and Computation: Practice and Experience, vol. 34, p. e6940, 2022, doi: 10.1002/cpe.6940.

- [22] A. Mahmood, S. K. Singh, and A. K. Tiwari, “Pre-trained deep learning-based classification of jujube fruits according to their maturity level,” *Neural Comput. Appl.*, vol. 34, no. 16, pp. 13925–13935, 2022. doi: 10.1007/s00521-022-07213-5.
- [23] A. Mahmood, A. K. Tiwari, and S. K. Singh, “Maturity grading of jujube for industrial applications harnessing deep learning,” 2023. [Online]. Available: <https://www.researchsquare.com/article/rs-2561485/latest.pdf>
- [24] A. Mahmood, A. Tiwari, and S. Singh, “C-net: A deep learning-based jujube grading approach,” *J. Food Meas. Charact.*, vol. 18, pp. 7794–7805, 2024. doi: 10.1007/s11694-024-02765-7.
- [25] T. Khatun, M. A. S. Nirob, P. Bishshash, M. Akter, and M. S. Uddin, “A comprehensive dragon fruit image dataset for detecting the maturity and quality grading of dragon fruit,” *Data Brief*, vol. 52, 2024, Art. no. 109936. doi: 10.1016/j.dib.2023.109936.
- [26] G. C. Wakchaure, S. B. Nikam, K. R. Barge, S. Kumar, K. K. Meena, V. J. Nagalkar, J. D. Choudhari, V. P. Kad, and K. S. Reddy, “Maturity stages detection prototype device for classifying custard apple (*Annona squamosa* L) fruit using image processing approach,” *Smart Agric. Technol.*, vol. 7, 2024, Art. no. 100394. doi: 10.1016/j.atech.2023.100394.
- [27] J. Yeshwanth and J. D. Sweetlin, “Enhancing Taiwan Guava Grading through Advanced Image Processing and Deep Learning Techniques,” pp. 1–6, Dec. 2023, doi: 10.1109/icacic59454.2023.10435292.
- [28] A. Hayat, F. Morgado-Dias, T. Choudhury, T. P. Singh, and K. Kotecha, “FruitVision: A deep learning based automatic fruit grading system,” *Open Agriculture*, Jan. 2024, doi: 10.1515/opag-2022-0276.
- [29] M. M. Ali, M. Raj, and D. Vatsa, “FruizNet Using an Efficient Convolutional Neural Network,” pp. 287–292, Mar. 2023, doi: 10.1109/ICCIKE58312.2023.10131865.
- [30] A. Bhatt and M. Joshi, “Leveraging Deep Transfer Learning for Precision in Similar Color and Texture-Based Fruit Classification,” *International journal of next-generation computing*, Dec. 2024, doi: 10.47164/ijngc.v15i3.1592.

- [31] T. Choudhury et al., “Quality Evaluation in Guavas using Deep Learning Architectures: An Experimental Review,” 2022 International Congress on Human-Computer Interaction, Optimization and Robotic Applications (HORA), pp. 1–6, Jun. 2022, doi: 10.1109/hora55278.2022.9799824.
- [32] I. D. Apostolopoulos, M. Tzani, and S. I. Aznaouridis, “A General Machine Learning Model for Assessing Fruit Quality Using Deep Image Features,” Sep. 2023, doi: 10.3390/ai4040041.