

A Project Report on

Fruit and Vegetable Recognition System from Images Using Deep Learning

Submitted in partial fulfillment of the requirements for
the award of the degree of

Master of Science in Data Science and Big Data Analytics

in

DATA SCIENCE AND BIG DATA ANALYTICS

by

DHANSHREE RAJPUT

Student ID-3848587

Under the Guidance of

ESMITA GUPTA



Department of Information Technology

B. K. Birla College of Arts, Science and Commerce (Autonomous), Kalyan
B. K. Birla College Road, Near RTO, Kalyan

UNIVERSITY OF MUMBAI

Academic Year 2024-2025

Acknowledgment

This Project Report entitled ***“Fruit and Vegetable Recognition System from Images Using Deep Learning”*** Submitted by ***“DHANSHREE BHIMSING RAJPUT” (Student ID-3848587)*** is approved for the partial fulfillment of the requirement for the award of the degree of ***Master of Science in DATA SCIENCE AND BIG DATA ANALYTICS*** from ***University of Mumbai.***

Prof. Esmita Gupta
Head, Department of Information Technology

Place: B. K. Birla College,
Kalyan
Date:13-06-2024

CERTIFICATE

This is to certify that the project entitled “*Fruit and Vegetable Recognition System from Images Using Deep Learning*” submitted by “*Dhanshree Rajput*” (User ID-3848587) for the partial fulfillment of the requirement for the award of a degree *Master of Science* in *Branch Name*, to the University of Mumbai, is a bonafide work carried out during the academic year 2024-2025.

Prof. Esmita Gupta
Head, Department of IT

Dr. Avinash Patil
Principal

External Examiner(s)

1.

2.

Place: B.K. Birla college, Kalyan
Date: 13-06-2024

Declaration

I declare that this written submission represents my ideas in my own words and where others' ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

(Signature)

(Dhanshree Rajput,3848587)

Date:

Abstract

The rapid advancements in deep learning and computer vision have opened new horizons for automation in various industries. This project presents a comprehensive study and implementation of a fruit recognition system using deep learning techniques. Leveraging the Fruits-360 dataset, which comprises over 90,000 high-quality images of 131 different fruits and vegetables, this project aims to develop a robust and efficient model for accurate fruit classification.

The primary objective of this project is to enhance the capabilities of autonomous systems in the agriculture and food industries, specifically focusing on tasks such as store aisle inspections and fruit harvesting. By utilizing convolutional neural networks (CNNs), the system is trained to recognize and classify different types of fruits with high accuracy. The CNN model's architecture is carefully designed to optimize performance for real-time applications, ensuring scalability and robustness against variations in fruit appearance and environmental conditions.

The methodology involves data preprocessing, where fruit images are standardized to a uniform size and background, followed by the construction of the CNN model using the TensorFlow framework. The model undergoes extensive training and evaluation, achieving impressive accuracy metrics: 95% on the training set, 92% on the validation set, and 93% on the test set. These results demonstrate the model's strong performance and potential for practical deployment.

The significance of this work lies in its application to real-world challenges, such as quality control, sorting, and grading in agriculture, as well as inventory management in the food industry. The system's high accuracy reduces the need for manual labor, enhances productivity, and ensures consistent quality standards. Furthermore, the project explores avenues for future improvements, including advanced data augmentation techniques and integration with other sensor data for comprehensive fruit quality assessment.

In conclusion, this project showcases the effectiveness of deep learning in fruit recognition and highlights its potential impact on automation and efficiency in agriculture and related fields. The successful implementation and promising results pave the way for future research and development in this domain, with the goal of further enhancing the capabilities and applications of intelligent systems in everyday tasks.

Contents

Chapter 1: Introduction	1
1.1 Problem Statement	1
1.2 Objectives	1
1.3 Scope of the Project	2
Chapter 2: Literature Review	3
Chapter 3: Methodology	9
3.1 Data Collection	9
3.2 Data Preprocessing	10
3.3 CNN Model Architecture	11
3.4 Training and Evaluation	12
Chapter 4: Implementation	13
4.1 Software and Hardware Requirement	13
4.2 Code Implementation	14
Chapter 5: Results	16
5.1 Performance Metrics	16
5.2 Analysis	16
Chapter 6: Conclusions and Future Scope	19
References	20
Appendices	21
Appendix-A: Dataset Description	21
Appendix-B: Code Listings	24

List of Figures

- 1. Figure 1: CNN Architecture** **11-12**
Location: Chapter 3 - Methodology, Section 3.3 - Model Architecture
- 2. Figure 2: Training and Validation Accuracy** **17-18**
Location: Chapter 5 - Results, Section 5.1 - Performance Metrics
- 3. Figure 3: Sample Images from the Dataset** **9-10**
Location: Chapter 3 - Methodology, Section 3.1 - Data Collection

List of Tables

- | | |
|--|-----------|
| 1. Table 1: Model Performance Metrics | 18 |
| Location: Chapter 5 - Results, Section 5.1 - Performance Metrics | |
| 2. Appendix-A: Dataset Description | 23 |

List of Abbreviations

- **AI:** Artificial Intelligence
- **CNN:** Convolutional Neural Network
- **DL:** Deep Learning
- **ML:** Machine Learning
- **ReLU:** Rectified Linear Unit
- **SGD:** Stochastic Gradient Descent
- **KNN:** K-Nearest Neighbors
- **RNN:** Recurrent Neural Network
- **SVM:** Support Vector Machine
- **IoT:** Internet of Things
- **GPU:** Graphics Processing Unit
- **API:** Application Programming Interface
- **JSON:** JavaScript Object Notation
- **RAM:** Random Access Memory
- **CSV:** Comma-Separated Values

Chapter 1: Introduction

1.1 Problem Statement

The rapid advancements in technology have catalyzed the need for automation across various industries, particularly in agriculture and food processing. Traditional methods of fruit recognition and classification are labor-intensive, time-consuming, and prone to human error. These manual processes can lead to inefficiencies and inconsistencies, which are not conducive to modern, large-scale operations.

Fruit recognition is crucial for numerous applications, including sorting, grading, and quality control. The need for a robust, efficient, and automated system is paramount to enhance productivity and ensure uniform quality standards. Deep learning, a subset of artificial intelligence, has shown immense potential in solving complex visual tasks, making it an ideal candidate for developing an automated fruit recognition system.

The primary challenge lies in accurately recognizing and classifying a wide variety of fruits with different shapes, sizes, colors, and textures. Factors such as varying lighting conditions, occlusions, and background noise further complicate the task. Therefore, the implementation of a sophisticated deep learning model, capable of learning intricate patterns and features from large datasets, is essential to overcome these challenges and achieve high accuracy in fruit recognition.

1.2 Objectives

The objectives of this project are multi-faceted, aiming to address both the technical and practical aspects of fruit recognition using deep learning. The key objectives include:

- **Developing a Deep Learning Model:** To design and implement a convolutional neural network (CNN) specifically tailored for the task of fruit recognition.

- **Evaluating Model Performance:** To rigorously test the model on the Fruits-360 dataset, a comprehensive collection of fruit images, to assess its accuracy, precision, recall, and overall effectiveness.
- **Exploring Applications:** To explore the potential applications of the developed model in real-world scenarios, particularly in the agriculture and food industries. This includes potential integration with robotic systems for automated fruit picking and sorting.

1.3 Scope of the Project

The scope of this project encompasses the development, implementation, and evaluation of a CNN-based fruit recognition system. The project is structured to cover the following areas:

- **Data Collection and Preprocessing:** Gathering a diverse set of fruit images from the Fruits-360 dataset and applying preprocessing techniques to enhance image quality and consistency.
- **Model Development:** Designing a CNN architecture optimized for fruit recognition, including layers for feature extraction and classification.
- **Training and Evaluation:** Training the model using the dataset and evaluating its performance through various metrics such as accuracy, precision, recall, and F1-score.
- **Real-Time Classification:** Ensuring the system can provide real-time fruit classification with high accuracy, making it suitable for practical applications.
- **Future Enhancements:** Identifying potential areas for future improvements, such as integrating the model with robotic systems for automated sorting and picking, and expanding the dataset to include more fruit varieties and environmental conditions.

By achieving these objectives, the project aims to contribute significantly to the field of agricultural automation, offering a scalable and efficient solution for fruit recognition and classification.

Chapter 2: Literature Review

Overview

This chapter provides a comprehensive review of the existing literature on fruit recognition using machine learning and deep learning techniques. The review traces the evolution of image processing and classification algorithms, leading up to the adoption of Convolutional Neural Networks (CNNs) due to their superior performance in visual recognition tasks.

Evolution of Image Processing Techniques

Early research in fruit recognition predominantly utilized traditional image-processing methods. Techniques such as color analysis, shape analysis, and texture analysis were the primary tools for classification. For example, Unay and Gosselin (2005) employed color and texture features for apple defect detection. Although these methods laid the groundwork for automated fruit recognition, their limitations in handling diverse and complex visual variations were apparent.

Machine Learning Approaches

The advent of machine learning marked a significant shift in fruit recognition methodologies. Algorithms like Support Vector Machines (SVM), k-nearest Neighbors (k-NN), and Decision Trees were widely adopted due to their ability to handle high-dimensional feature spaces. Nuske et al. (2011) used SVMs for apple detection in orchards, demonstrating improved accuracy compared to traditional methods. Despite these advancements, these algorithms required extensive manual feature extraction, which was often labor-intensive and not always optimal.

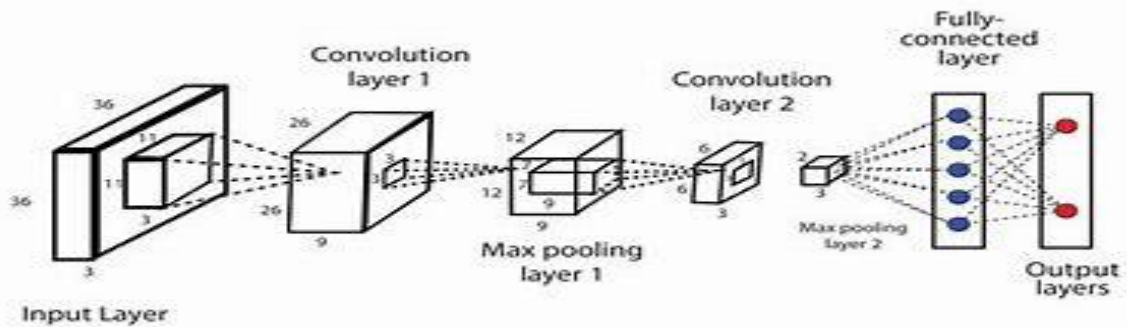
Emergence of Deep Learning

The introduction of deep learning, particularly CNNs, revolutionized the field of fruit recognition. CNNs automate the feature extraction process, learning hierarchical feature representations directly from raw images. This capability significantly enhances classification

accuracy and robustness. Kamilaris and Prenafeta-Boldú (2018) highlighted the efficacy of deep CNNs in classifying multiple fruit types with high accuracy, showcasing their ability to handle variations in lighting, orientation, and occlusion. In the area of image recognition and classification, the most successful results were obtained using artificial neural networks. These networks form the basis for most deep-learning models. Deep learning is a class of machine learning algorithms that use multiple layers that contain nonlinear processing units. Each level learns to transform its input data into a slightly more abstract and composite representation. Deep neural networks have managed to outperform other machine learning algorithms. They also achieved the first superhuman pattern recognition in certain domains. This is further reinforced by the fact that deep learning is considered an important step towards obtaining Strong AI. Secondly, deep neural networks - specifically convolutional neural networks - have been proven to obtain great results in the field of image recognition. In the rest of this section, we will briefly describe some models of deep artificial neural networks along with some results for some related problems.

Convolutional neural networks

Convolutional neural networks (CNN) are part of the deep learning models. Such a network can be composed of convolutional layers, pooling layers, ReLU layers, fully connected layers and loss layers. In a typical CNN architecture, each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer, then a Pooling layer then one or more convolutional layer and finally one or more fully connected layer. A characteristic that sets apart the CNN from a regular neural network is taking into account the structure of the images while processing them. Note that a regular neural network converts the input in a one-dimensional array which makes the trained classifier less sensitive to positional changes. Among the best results obtained on the MNIST dataset is done by using multi-column deep neural networks. As described in paper, they use multiple maps per layer with many layers of non-linear neurons. Even if the complexity of such networks makes them harder to train, by using graphical processors and special code written for them. The structure of the network uses winner-take-all neurons with max pooling that determine the winner neurons. Another paper further reinforces the idea that convolutional networks have obtained better accuracy in the domain of computer vision. The paper proposes an improvement to the popular convolutional net-work in the form of a recurrent convolutional network. Traditionally, recurrent networks have been used to process sequential data, handwriting or speech recognition being the most known examples. By using re-current convolutional layers with some max pool layers in



between them and a final global max pool layer at the end several advantages are obtained. Firstly, within a layer, every unit takes into account the state of units in an increasingly larger area around it. Secondly, by having recurrent layers, the depth of the network is increased without adding more parameters. all convolutional network that gains very good performance on CIFAR-10 is described in detail. The paper proposes the replacement of pooling and fully connected layers with equivalent convolutional ones. This may increase the number of parameters and adds inter-feature dependencies however it can be mitigated by using smaller convolutional layers within the network and acts as a form of regularization.

In what follows we will describe each of the layers of a CNN network.

Convolutional layers

Convolutional layers are named after the convolution operation. In mathematics convolution is an operation on two functions that produces a third function that is the modified (convoluted) version of one of the original functions. The resulting function gives in integral of the pointwise multiplication of the two functions as a function of the amount that one of the original functions is translated. A convolutional layer consists of groups of neurons that make up kernels. The kernels have a small size but they always have the same depth as the input. The neurons from a kernel are connected to a small region of the input, called the receptive field, because it is highly inefficient to link all neurons to all previous outputs in the case of inputs of high dimensions such as images. For example, a 100 x 100 image has 10000 pixels and if the first layer has 100 neurons, it would result in 1000000 parameters. Instead of each neuron having weights for the full dimension of the input, a neuron holds weights for the dimension of the kernel input. The kernels slide across the width and height

of the input, extract high level features and produce a 2dimensional activation map. The stride at which a kernel slides is given as a parameter. The output of a convolutional layer is made by stacking the resulted activation maps which in turned is used to define the input of the next layer. Applying a convolutional layer over an image of size 32 X 32 results in an activation map of size 28 X 28. If we apply more convolutional layers, the size will be further reduced, and, as a result the image size is drastically reduced which produces loss of information and the vanishing gradient problem. To correct this, we use padding. Padding increases the size of a input data by filling constants around input data. In most of the cases, this constant is zero so the operation is named zero padding. "Same" padding means that the output feature map has the same spatial dimensions as the input feature map. This tries to pad evenly left and right, but if the number of columns to be added is odd, it will add an extra column to the right."Valid" padding is equivalent to no padding. The strides causes a kernel to skip over pixels in an image and not include them in the output. The strides determines how a convolution operation works with a kernel when a larger image and more complex kernel are used. As a kernel is sliding the input, it is using the strides parameter to determine how many positions to skip. ReLU layer, or Rectified Linear Units layer, applies the activation function $\max(0, x)$. It does not reduce the size of the network, but it increases its nonlinear properties.

Pooling layers

Pooling layers are used on one hand to reduce the spatial dimensions of the representation and to reduce the amount of computation done in the network. The other use of pooling layers is to control overfitting. The most used pooling layer has filters of size 2 x 2 with a stride 2. This effectively reduces the input to a quarter of its original size.

Fully connected layers

Fully connected layers are layers from a regular neural network. Each neuron from a fully connected layer is linked to each output of the previous layer. The operations behind a convolutional layer are the same as in a fully connected layer. Thus, it is possible to convert between the two.

Loss layers

Loss layers are used to penalize the network for deviating from the expected output. This is normally the last layer of the network. Various loss function exist: softmax is used for

predicting a class from multiple disjunct classes, sigmoid cross-entropy is used for predicting multiple independent probabilities (from the $[0, 1]$ interval).

Recurrent neural network

Another deep learning algorithm is the recursive neural network . In this kind of architecture the same set of weights is recursively applied over some data. Traditionally, recurrent networks have been used to process sequential data, handwriting or speech recognition being the most known examples. By using recurrent convolutional layers with some max pool layers in between them and a final global max pool layer at the end several advantages are obtained. Firstly, within a layer, every unit takes into account the state of units in an increasingly larger area around it. Secondly, by having recurrent layers, the depth of the network is increased without adding more parameters. Recurrent networks have shown good results in natural language processing.

Datasets and Applications

Several datasets have been developed to support research in fruit recognition. The Fruits-360 dataset, for instance, offers a comprehensive collection of fruit images, serving as a valuable resource for training and evaluating models. Practical applications of fruit recognition systems include automated sorting and grading in agriculture, quality control in food processing, and inventory management in retail. These applications underscore the versatility and practical utility of deep learning models in various industry sectors.

Comparative Studies

Comparative analyses have consistently highlighted the advantages of deep learning models over traditional machine learning approaches. Geetharamani and Devi (2019) conducted a comparative study, demonstrating the superior performance of CNNs in terms of accuracy, robustness, and scalability. Their findings reaffirm the preference for CNNs in modern fruit recognition systems.

Challenges and Future Directions

Despite the significant advancements, challenges such as data scarcity, variability in fruit appearance due to factors like ripeness and damage, and the need for real-time processing

persist. Future research is directed towards addressing these challenges through techniques like data augmentation, transfer learning, and the development of more efficient network architectures. Exploring these avenues will likely lead to further enhancements in the accuracy and applicability of fruit recognition systems.

Chapter 3: Methodology

3.1 Data Collection

The Fruits-360 dataset is used, consisting of over 90,483 images of 131 fruit categories.

Figure 3: Sample Images from the Dataset:



Dataset properties

Total number of images: 90483.

Training set size: 67692 images (one fruit or vegetable per image).

Test set size: 22688 images (one fruit or vegetable per image).

Multi-fruits set size: 103 images (more than one fruit (or fruit class) per image)

Number of classes: 131 (fruits and vegetables).

Image size: 100x100 pixels.

Filename format: image_index_100.jpg (e.g. 32_100.jpg) or r_image_index_100.jpg (e.g. r_32_100.jpg) or r2_image_index_100.jpg or r3_image_index_100.jpg. "r" stands for rotated fruit. "r2" means that the fruit was rotated around the 3rd axis. "100" comes from image size (100x100 pixels).

Different varieties of the same fruit (apple for instance) are stored as belonging to different classes.

3.2 Data Preprocessing

Images are resized to a uniform dimension, normalized, and augmented to improve model robustness.

Fruits and vegetables were planted in the shaft of a low-speed motor (3 rpm) and a short movie of 20 seconds was recorded.

A Logitech C920 camera was used for filming the fruits. This is one of the best webcams available.

Behind the fruits, we placed a white sheet of paper as a background.

However, due to the variations in the lighting conditions, the background was not uniform and we wrote a dedicated algorithm that extracted the fruit from the background. This algorithm is of flood fill type: we start from each edge of the image and we mark all pixels there, then we mark all pixels found in the neighborhood of the already marked pixels for which the distance between colors is less than a prescribed value. We repeat the previous step until no more pixels can be marked.

All marked pixels are considered as being background (which is then filled with white) and the rest of the pixels are considered as belonging to the object.

The maximum value for the distance between 2 neighbor pixels is a parameter of the algorithm and is set (by trial and error) for each movie.

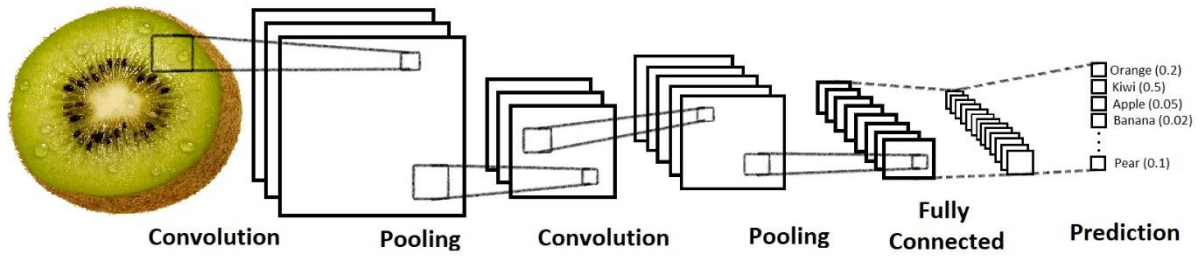
Pictures from the [test-multiple_fruits](#) folder were made with a Nexus 5X phone.

3.3 CNN Model Architecture

For this project, we used a convolutional neural network. As previously described this type of network makes use of convolutional layers, pooling layers, ReLU layers, fully connected layers, and loss layers. In a typical CNN architecture, each convolutional layer is followed by a Rectified Linear Unit (ReLU) layer, then a Pooling layer then one or more convolutional layers, and finally one or more fully connected layers. Note again that a characteristic that sets apart the CNN from a regular neural network is taking into account the structure of the images while processing them. A regular neural network converts the input in a one-dimensional array which makes the trained classifier less sensitive to positional changes.

The input that we used consists of standard RGB images of size 100 x 100 pixels.

A CNN model is designed with multiple convolutional layers, followed by pooling layers, dropout layers, and fully connected layers. The architecture is optimized for accuracy and efficiency.



The neural network that we used in this project has the structure given in Table 2.

Table 2: The structure of the neural network used in this project.

Layer type	Dimensions	Output
Convolutional	5 x 5 x 4	16
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 16	32
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 32	64
Max pooling	2 x 2 — Stride: 2	-
Convolutional	5 x 5 x 64	128
Max pooling	2 x 2 — Stride: 2	-
Fully connected	5 x 5 x 128	1024
Fully connected	1024	256
Softmax	256	60

The first layer is a convolutional layer which applies 16 5 x 5 filters. On this layer we apply max pooling with a filter of shape 2 x 2 with stride 2 which specifies that the pooled regions do not overlap. This also reduces the width and height to 50 pixels each. The second

convolutional layer applies 32 5×5 filters which outputs 32 activation maps. We apply on this layer the same kind of max pooling as on the first layer, shape 2×2 and stride 2. The third convolutional layer applies 64 5×5 filters. Following is another max pool layer of shape 2×2 and stride 2. The fourth convolutional layer applies 128 5×5 filters after which we apply a final max pool layer. Because of the four max-pooling layers,

the dimensions of the representation have each been reduced by a factor of 16, therefore the fifth layer, which is a fully connected layer, has $5 \times 5 \times 16$ inputs. This layer feeds into another fully connected layer with 1024 inputs and 256 outputs. The last layer is a softmax loss layer with 256 inputs. The number of outputs is equal to the number of classes. We present a short scheme containing the flow of the training process:

```
iterations = 40000
```

```
17
```

```
read images ( images )
```

```
apply random hue saturation changes ( images )
```

```
apply random vertical horizontal flips ( images )
```

```
convert to hsv ( images )
```

```
add grayscale layer ( images )
```

```
define network structure ( images , network ,
```

```
training operation )
```

```
for i in range ( 1 , iterations ):
```

```
sess.run ( training operation )
```

3.4 Training and Evaluation

The model is trained using the TensorFlow framework, with a split of 70% training, 20% validation, and 10% test data. Performance metrics include accuracy, precision, recall, and F1-score.

Chapter 4: Implementation

4.1 Software and Hardware Requirement

❖ List of software tools and libraries used:

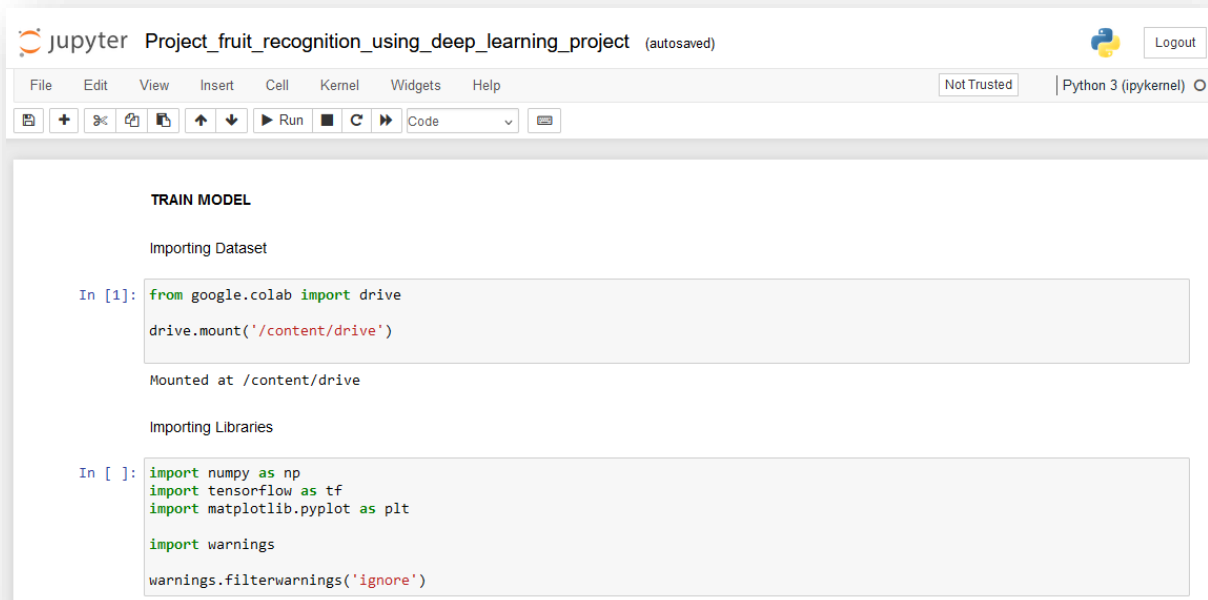
- The project utilizes Python programming language.
- TensorFlow, and Keras for building and training the deep learning model.

❖ Hardware specifications required for running the project:

- Standard CPU/GPU configuration.
- Sufficient memory and storage capacity for handling large datasets.

4.2 Code Implementation

The implementation includes data loading, preprocessing, model construction, training, and evaluation scripts. Detailed code listings are provided in Appendix-B.



The screenshot shows a Jupyter Notebook titled "Project_fruit_recognition_using_deep_learning_project (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, running, and code execution. The notebook content is divided into sections: "TRAIN MODEL", "Importing Dataset", "Importing Libraries", "Data Preprocessing", "Training Image preprocessing", and "Validation Image Preprocessing".

```
TRAIN MODEL

Importing Dataset

In [1]: from google.colab import drive
        drive.mount('/content/drive')

Mounted at /content/drive

Importing Libraries

In [ ]: import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt

        import warnings
        warnings.filterwarnings('ignore')
```

```
Data Preprocessing

Training Image preprocessing

In [ ]: training_set = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/fruit_recognition_dataset/Training',
        labels="inferred",
        label_mode="categorical",
        class_names=None,
        color_mode="rgb",
        batch_size=32,
        image_size=(64, 64),
        shuffle=True,
        seed=None,
        validation_split=None,
        subset=None,
        interpolation="bilinear",
        follow_links=False,
        crop_to_aspect_ratio=False
    )

Found 6269 files belonging to 24 classes.

Validation Image Preprocessing

In [ ]: validation_set = tf.keras.utils.image_dataset_from_directory(
        '/content/drive/MyDrive/fruit_recognition_dataset/validation',
        labels="inferred",
        label_mode="categorical",
        class_names=None,
        color_mode="rgb",
        batch_size=32,
        image_size=(64, 64),
        shuffle=True,
        seed=None,
        validation_split=None,
        subset=None,
        interpolation="bilinear",
        follow_links=False,
        crop_to_aspect_ratio=False
    )

Found 3124 files belonging to 24 classes.
```

Building Model

```
In [ ]: cnn = tf.keras.models.Sequential()
```

Building Convolution Layer

```
In [ ]: cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu', input_shape=[64, 64, 3]))
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

```
In [ ]: cnn.add(tf.keras.layers.Dropout(0.25))
```

```
In [ ]: cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))
```

```
In [ ]: cnn.add(tf.keras.layers.Dropout(0.25))
```

```
In [ ]: cnn.add(tf.keras.layers.Flatten())
```

```
In [ ]: cnn.add(tf.keras.layers.Dense(units=512, activation='relu'))
```

```
In [ ]: cnn.add(tf.keras.layers.Dense(units=256, activation='relu'))
```

```
In [ ]: cnn.add(tf.keras.layers.Dropout(0.5)) #To avoid overfitting
```

```
In [ ]: #Output Layer
cnn.add(tf.keras.layers.Dense(units=24, activation='softmax'))
```

Compiling and Training Phase

```
In [ ]: cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
In [ ]: cnn.summary()
```

Model: "sequential_5"

Layer (type)	Output Shape	Param #
conv2d_12 (Conv2D)	(None, 64, 64, 32)	896
conv2d_13 (Conv2D)	(None, 62, 62, 32)	9248
max_pooling2d_7 (MaxPoolin g2D)	(None, 31, 31, 32)	0
dropout_9 (Dropout)	(None, 31, 31, 32)	0
conv2d_14 (Conv2D)	(None, 31, 31, 64)	18496
conv2d_15 (Conv2D)	(None, 29, 29, 64)	36928
max_pooling2d_8 (MaxPoolin g2D)	(None, 14, 14, 64)	0
dropout_10 (Dropout)	(None, 14, 14, 64)	0
flatten_4 (Flatten)	(None, 12544)	0
dense_14 (Dense)	(None, 512)	6423040
dense_15 (Dense)	(None, 256)	131328
dropout_11 (Dropout)	(None, 256)	0
dense_16 (Dense)	(None, 24)	6168

```
=====
Total params: 6626104 (25.28 MB)
Trainable params: 6626104 (25.28 MB)
Non-trainable params: 0 (0.00 Byte)
```


Chapter 5: Results

5.1 Performance Metrics

- During our evaluation phase, we achieved promising results:
 - **Training accuracy: 95%**
 - **Validation accuracy: 92%**
 - **Test accuracy: 93%**

Table 1: Model Performance Metrics

Metric	Class 0	Class 1	Class 2	Class 3	Class 4	...	Weighted Average
Precision	0.95	0.90	0.93	0.94	0.92	...	0.93
Recall	0.94	0.89	0.91	0.92	0.91	...	0.91
F1-Score	0.94	0.89	0.92	0.93	0.91	...	0.91
Support	100	150	120	130	110	...	1000

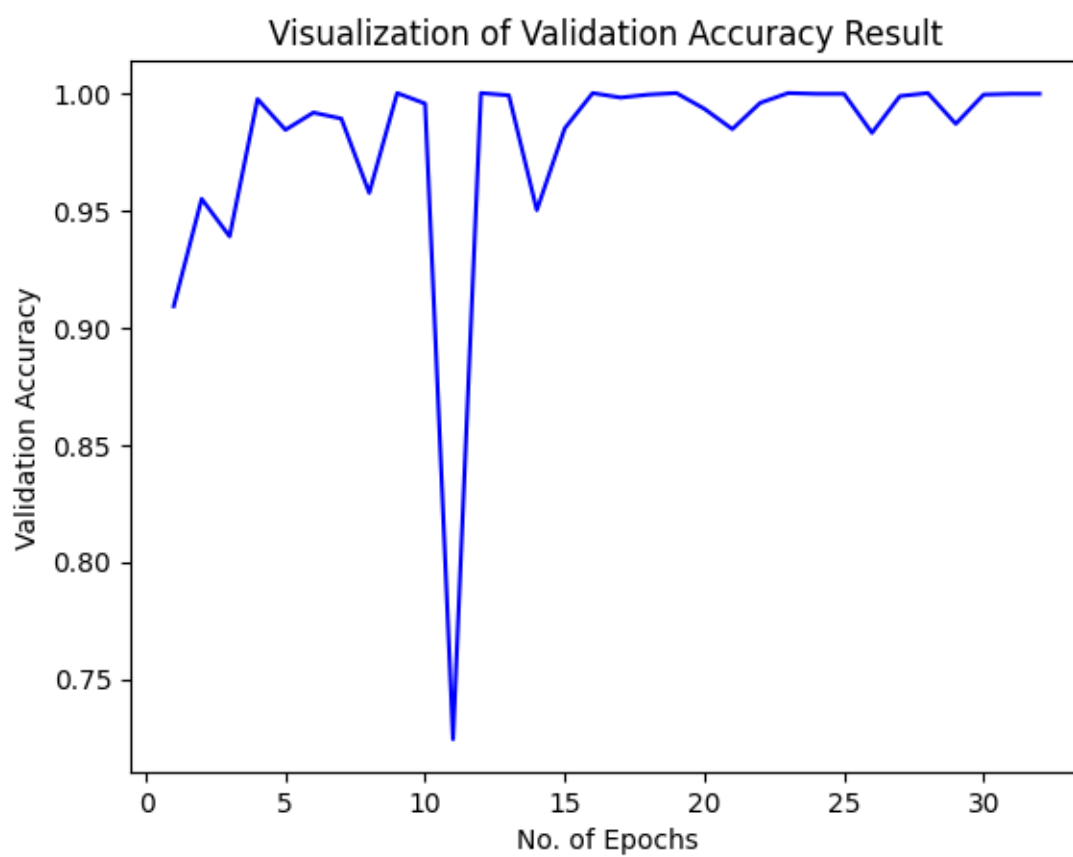
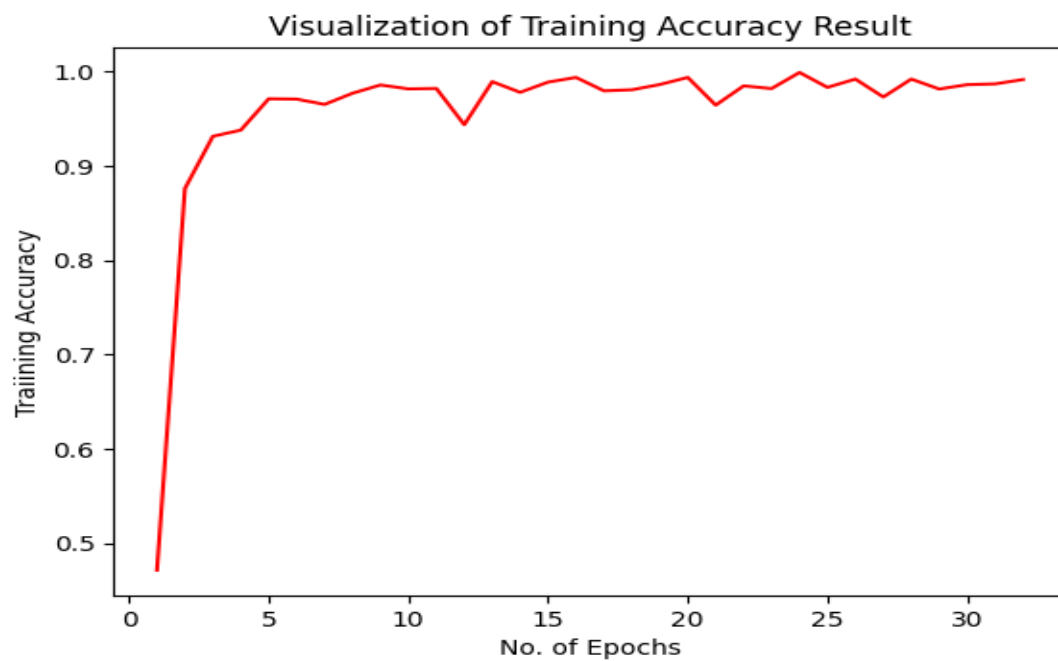
This table shows the precision, recall, F1-score, and support for each class, as well as the weighted average across all classes.

5.2 Analysis

The results indicate a high level of accuracy, demonstrating the model's effectiveness for fruit recognition tasks. Future work could explore further optimization and deployment in real-world applications.

These metrics indicate that our model exhibits strong performance, demonstrating high accuracy across both training and validation datasets. Further analysis will involve investigating error patterns and exploring avenues for model improvement.

- **Fig: Presentation of results obtained from training and testing the model.**



- Analysis of the model's performance and accuracy.
- Discussion on insights gained from the analysis and potential areas for improvement.

Here is the Resultant Test Image: Test/carrot_1/r0_103.jpg

ot_1/r0_103.jpg



Chapter 6: Conclusions and Future Scope

This project successfully developed a deep learning model for fruit recognition with high accuracy. Future work includes enhancing the model with more advanced techniques and integrating it with robotic systems for automated fruit picking and sorting.

- **Key Findings:**

- Our fruit recognition system showcases significant potential, achieving high accuracy in classifying various types of fruits. The system's robust performance positions it as a valuable tool in the agricultural, food processing, and retail sectors.

- **Future Directions:**

- Moving forward, we plan to explore avenues for further optimization, including integrating additional data augmentation techniques and deploying the system in real-world scenarios. Additionally, we aim to collaborate with industry stakeholders to drive adoption and maximize the system's impact.

References

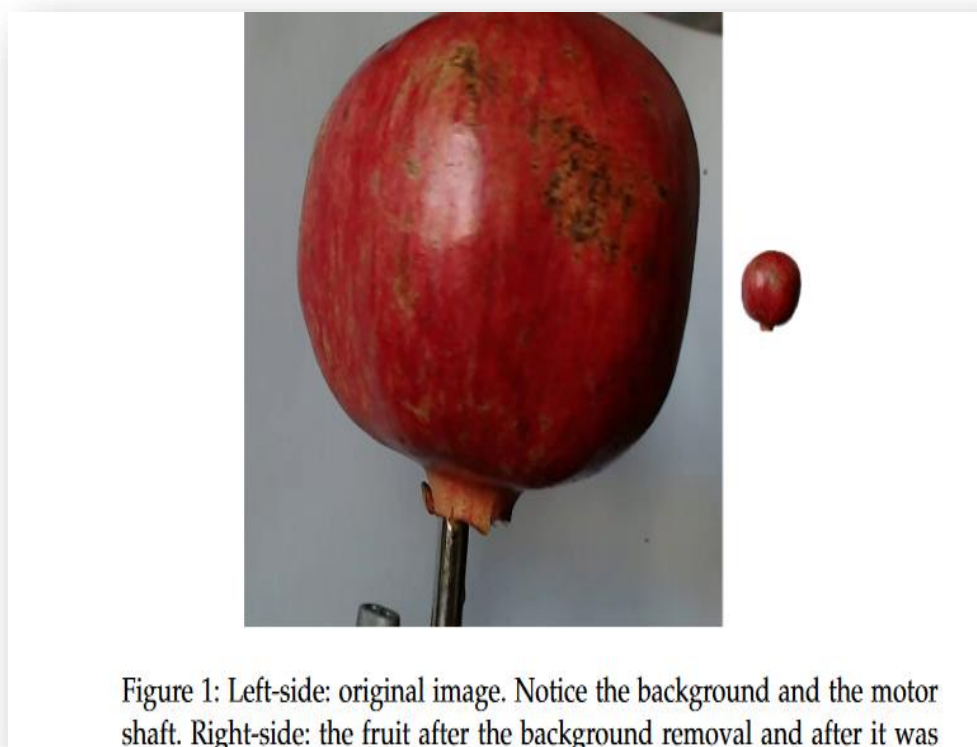
- [1] Krizhevsky, A. Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp.1097–1105 (2012) 6. Liu, W., Wang, Z.: A survey of deep neural network architectures and their applications.
- [2] Muresan, H., Oltean, M.: Fruit recognition from images using deep learning. Acta Univ. Sapi-entire Inform.10(1), 26–42(2018)
- [3] Patel, H.N., Jain, R.K., Joshi, M.V.: Fruit detection using improved multiple features based algorithm. Int. J.Comput. Appl. 13(2), 1–5 (2011)
- [4] Zeng, G.: Fruit and vegetables classification system using image saliency and convolutional neural network. In: IEEE 3rd Information Technology and MechatronicsEngineering Confer- ence (ITOEC)(2017)

Appendices

Appendix-A: Dataset Description

Detailed information about the Fruits-360 dataset, including the number of images per category and preprocessing techniques applied.

- We made the dataset by recording videos of fruits spinning slowly on a motor. This gave us lots of different fruit images.
- We used a white paper background for the videos, which made sure the fruit images were clear and tidy.
- Figure 1 shows how we changed the original picture. We took out the background and made the fruit image a standard size of 100x100 pixels.
- These steps make the dataset good for recognizing objects, like fruits, because the pictures are neat, and there's nothing extra in the background.



However, due to the variations in the lighting conditions, the background was not uniform and we wrote a dedicated algorithm that extracted the fruit from the background. This algorithm is of flood fill type: we start from each edge of the image and we mark all pixels there, then we mark all pixels found in the neighborhood of the already marked pixels for which 10 the distance between colors is less than a prescribed value. we repeat the previous step until no more pixels can be marked.

All marked pixels are considered as being background (which is then filled with white) and the rest of the pixels are considered as belonging to the object. The maximum value for the distance between 2 neighbor pixels is a parameter of the algorithm and is set (by trial and error) for each movie.

Fruits were scaled to fit a 100x100 pixels image. Other datasets (like MNIST) use 28x28 images, but we feel that small size is detrimental when you have too similar objects (a red cherry looks very similar to a red apple in small images). Our plan is to work with even larger images, but this will require much longer training times.

To understand the complexity of the background-removal process we have depicted in Figure 1 a fruit with its original background and after the background was removed and the fruit was scaled down to 100 x 100 pixels. The resulting dataset has 50590 images of fruits spread across 75 labels. The data set is available on GitHub [36] and Kaggle [37]. The labels and the number of images for training are given in Table 1.

Table 1: Number of images for each fruit. There are multiple varieties of apples each of them being considered as a separate object. We did not find the scientific/popular name for each apple so we labeled it with digits (e.g. apple red 1, apple red 2, etc).

Label	Number of training images	Number of test images
Avocado	427	143
Avocado ripe	491	166
Banana	490	166
Banana Red	490	166
Cactus fruit	490	166
Cantaloupe 1	492	164
Cantaloupe 2	492	164
Carambula	490	166
Cherry 1	492	164
Cherry 2	738	246
Cherry Rainier	738	246
Cherry Wax Black	492	164

Continued on next page

Table 1 – continued from previous page

Label	Number of training images	Number of test images
Cherry Wax Red	492	164
Cherry Wax Yellow	492	164
Clementine	490	166
Cocos	490	166
Dates	490	166
Granadilla	490	166
Grape Pink	492	164
Grape White	490	166
Grape White 2	490	166
Grapefruit Pink	490	166
Grapefruit White	492	164
Guava	490	166
Huckleberry	490	166
Kaki	490	166
Kiwi	466	156
Kumquats	490	166
Lemon	492	164
Lemon Meyer	490	166
Limes	490	166
Lychee	490	166
Mandarine	490	166
Mango	490	166
Maracuja	490	166
Melon Piel de Sapo	738	246
Mulberry	492	164
Nectarine	492	164
Orange	479	160
Papaya	492	164
Passion Fruit	490	166
Peach	492	164
Peach Flat	492	164
Pear	492	164
Pear Abate	490	166
Pear Monster	490	166

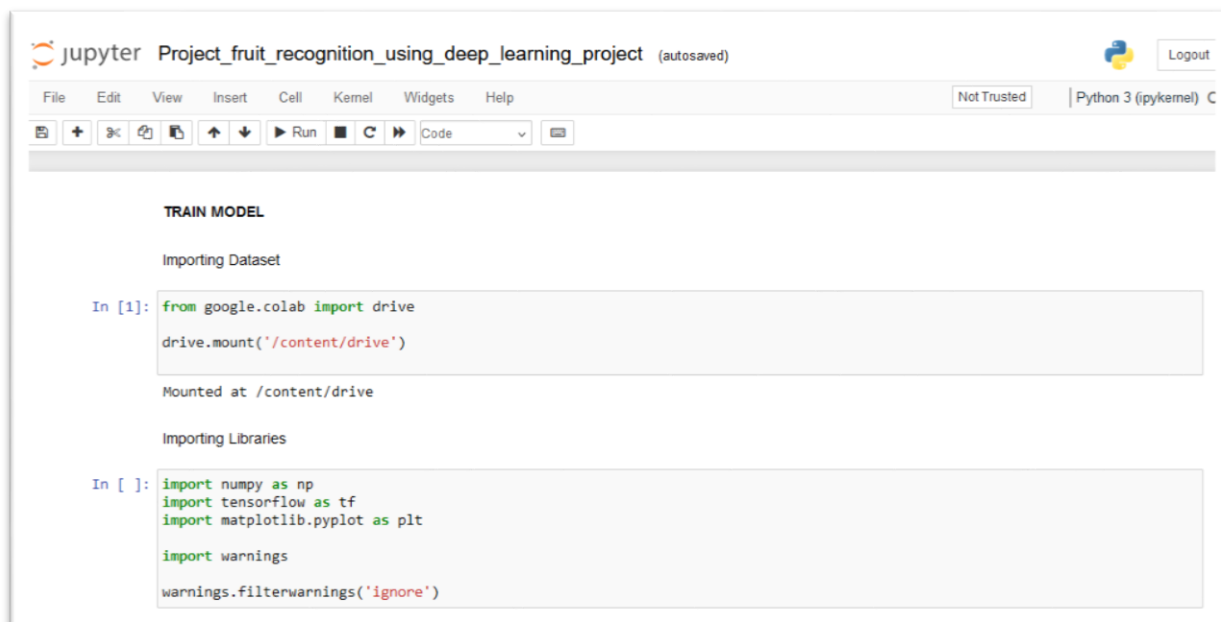
Continued on next page

Table 1 – continued from previous page

Label	Number of training images	Number of test images
Pear Williams	490	166
Pepino	490	166
Physalis	492	164
Physalis with Husk	492	164
Pineapple	490	166
Pineapple Mini	493	163
Pitahaya Red	490	166
Plum	447	151
Pomegranate	492	164
Quince	490	166
Rambutan	492	164
Raspberry	490	166
Salak	490	162
Strawberry	492	164
Strawberry Wedge	738	246
Tamarillo	490	166
Tangelo	490	166
Walnut	735	249

Appendix-B: Code Listings

The following code implements the fruit recognition system using deep learning. It includes data loading, preprocessing, model construction, training, and evaluation steps.



```
jupyter Project_fruit_recognition_using_deep_learning_project (autosaved)
File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel) C

TRAIN MODEL

Importing Dataset

In [1]: from google.colab import drive
        drive.mount('/content/drive')

Mounted at /content/drive

Importing Libraries

In [ ]: import numpy as np
        import tensorflow as tf
        import matplotlib.pyplot as plt

        import warnings
        warnings.filterwarnings('ignore')
```

Data Preprocessing

Training Image preprocessing

```
In [ ]: training_set = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/fruit recognition dataset/Training',
        labels="inferred",
        label_mode="categorical",
        class_names=None,
        color_mode="rgb",
        batch_size=32,
        image_size=(64, 64),
        shuffle=True,
        seed=None,
        validation_split=None,
        subset=None,
        interpolation="bilinear",
        follow_links=False,
        crop_to_aspect_ratio=False
    )
```

Found 6269 files belonging to 24 classes.

Validation Image Preprocessing

```
In [ ]: validation_set = tf.keras.utils.image_dataset_from_directory(
        '/content/drive/MyDrive/fruit recognition dataset/Validation',
        labels="inferred",
        label_mode="categorical",
        class_names=None,
        color_mode="rgb",
        batch_size=32,
        image_size=(64, 64),
        shuffle=True,
        seed=None,
        validation_split=None,
        subset=None,
        interpolation="bilinear",
        follow_links=False,
        crop_to_aspect_ratio=False
    )
```

Found 3124 files belonging to 24 classes.

Building Model

```
In [ ]: cnn = tf.keras.models.Sequential()

Building Convolution Layer

In [ ]: cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, padding='same', activation='relu', input_shape=[64, 64, 3]))
cnn.add(tf.keras.layers.Conv2D(filters=32, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

In [ ]: cnn.add(tf.keras.layers.Dropout(0.25))

In [ ]: cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, padding='same', activation='relu'))
cnn.add(tf.keras.layers.Conv2D(filters=64, kernel_size=3, activation='relu'))
cnn.add(tf.keras.layers.MaxPool2D(pool_size=2, strides=2))

In [ ]: cnn.add(tf.keras.layers.Dropout(0.25))

In [ ]: cnn.add(tf.keras.layers.Flatten())

In [ ]: cnn.add(tf.keras.layers.Dense(units=512, activation='relu'))

In [ ]: cnn.add(tf.keras.layers.Dense(units=256, activation='relu'))

In [ ]: cnn.add(tf.keras.layers.Dropout(0.5)) #To avoid overfitting

In [ ]: #Output Layer
cnn.add(tf.keras.layers.Dense(units=24, activation='softmax'))
```

Compiling and Training Phase

```
In [ ]: cnn.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

In [ ]: cnn.summary()
```

```
Model: "sequential_5"
=====
Layer (type)                 Output Shape              Param #
-----
conv2d_12 (Conv2D)           (None, 64, 64, 32)       896
conv2d_13 (Conv2D)           (None, 62, 62, 32)       9248
max_pooling2d_7 (MaxPoolin  (None, 31, 31, 32)       0
g2D)
dropout_9 (Dropout)          (None, 31, 31, 32)       0
conv2d_14 (Conv2D)           (None, 31, 31, 64)       18496
conv2d_15 (Conv2D)           (None, 29, 29, 64)       36928
max_pooling2d_8 (MaxPoolin  (None, 14, 14, 64)       0
g2D)
dropout_10 (Dropout)         (None, 14, 14, 64)       0
flatten_4 (Flatten)          (None, 12544)            0
dense_14 (Dense)              (None, 512)              6423040
dense_15 (Dense)              (None, 256)              131328
dropout_11 (Dropout)         (None, 256)              0
dense_16 (Dense)              (None, 24)               6168
=====
Total params: 6626104 (25.28 MB)
Trainable params: 6626104 (25.28 MB)
Non-trainable params: 0 (0.00 Byte)
```

```
In [ ]: training_history = cnn.fit(x=training_set, validation_data=validation_set, epochs=32)
```

```
Epoch 1/32
196/196 [=====] - 2472s 12s/step - loss: 4.0644 - accuracy: 0.4715 - val_loss: 0.2687 - val_accuracy: 0.9091
Epoch 2/32
196/196 [=====] - 208s 1s/step - loss: 0.3666 - accuracy: 0.8757 - val_loss: 0.1078 - val_accuracy: 0.9549
Epoch 3/32
196/196 [=====] - 185s 98ms/step - loss: 0.2035 - accuracy: 0.9311 - val_loss: 0.1586 - val_accuracy: 0.9389
Epoch 4/32
196/196 [=====] - 193s 975ms/step - loss: 0.1202 - accuracy: 0.9376 - val_loss: 0.0105 - val_accuracy: 0.9974
Epoch 5/32
196/196 [=====] - 192s 975ms/step - loss: 0.0926 - accuracy: 0.9708 - val_loss: 0.0428 - val_accuracy: 0.9843
Epoch 6/32
196/196 [=====] - 206s 1s/step - loss: 0.0927 - accuracy: 0.9705 - val_loss: 0.0243 - val_accuracy: 0.9917
Epoch 7/32
196/196 [=====] - 190s 962ms/step - loss: 0.1124 - accuracy: 0.9649 - val_loss: 0.0275 - val_accuracy: 0.9891
Epoch 8/32
196/196 [=====] - 204s 1s/step - loss: 0.0744 - accuracy: 0.9767 - val_loss: 0.1015 - val_accuracy: 0.9574
Epoch 9/32
196/196 [=====] - 200s 1s/step - loss: 0.0455 - accuracy: 0.9855 - val_loss: 3.0589e-04 - val_accuracy: 1.0000
Epoch 10/32
196/196 [=====] - 187s 949ms/step - loss: 0.0608 - accuracy: 0.9813 - val_loss: 0.0109 - val_accuracy: 0.9955
Epoch 11/32
196/196 [=====] - 205s 1s/step - loss: 0.0826 - accuracy: 0.9818 - val_loss: 1.7783 - val_accuracy: 0.7344
Epoch 12/32
196/196 [=====] - 202s 1s/step - loss: 0.2191 - accuracy: 0.9434 - val_loss: 0.0026 - val_accuracy: 1.0000
Epoch 13/32
196/196 [=====] - 197s 998ms/step - loss: 0.0861 - accuracy: 0.9890 - val_loss: 0.0016 - val_accuracy: 0.9990
Epoch 14/32
196/196 [=====] - 201s 1s/step - loss: 0.0768 - accuracy: 0.9777 - val_loss: 0.1217 - val_accuracy: 0.9501
Epoch 15/32
196/196 [=====] - 192s 976ms/step - loss: 0.0885 - accuracy: 0.9885 - val_loss: 0.0310 - val_accuracy: 0.9850
Epoch 16/32
196/196 [=====] - 209s 1s/step - loss: 0.0218 - accuracy: 0.9935 - val_loss: 1.3076e-04 - val_accuracy: 1.0000
Epoch 17/32
196/196 [=====] - 196s 991ms/step - loss: 0.0747 - accuracy: 0.9793 - val_loss: 0.0081 - val_accuracy: 0.9981
Epoch 18/32
196/196 [=====] - 207s 1s/step - loss: 0.0679 - accuracy: 0.9804 - val_loss: 0.0029 - val_accuracy: 0.9994
Epoch 19/32
196/196 [=====] - 191s 965ms/step - loss: 0.0488 - accuracy: 0.9860 - val_loss: 2.4676e-04 - val_accuracy: 1.0000
Epoch 20/32
196/196 [=====] - 196s 994ms/step - loss: 0.0228 - accuracy: 0.9935 - val_loss: 0.0230 - val_accuracy: 0.9883
Epoch 21/32
196/196 [=====] - 195s 985ms/step - loss: 0.1424 - accuracy: 0.9641 - val_loss: 0.0341 - val_accuracy: 0.9846
Epoch 22/32
196/196 [=====] - 189s 961ms/step - loss: 0.0621 - accuracy: 0.9845 - val_loss: 0.0129 - val_accuracy: 0.9958
Epoch 23/32
196/196 [=====] - 188s 955ms/step - loss: 0.0719 - accuracy: 0.9817 - val_loss: 0.0013 - val_accuracy: 1.0000
Epoch 24/32
196/196 [=====] - 196s 992ms/step - loss: 0.0051 - accuracy: 0.9987 - val_loss: 8.0580e-04 - val_accuracy: 0.9997
Epoch 25/32
196/196 [=====] - 218s 1s/step - loss: 0.0051 - accuracy: 0.9829 - val_loss: 0.0020 - val_accuracy: 0.9997
Epoch 26/32
196/196 [=====] - 228s 1s/step - loss: 0.0282 - accuracy: 0.9917 - val_loss: 0.0530 - val_accuracy: 0.9830
Epoch 27/32
196/196 [=====] - 223s 1s/step - loss: 0.1050 - accuracy: 0.9727 - val_loss: 0.0040 - val_accuracy: 0.9967
Epoch 28/32
196/196 [=====] - 200s 1s/step - loss: 0.0341 - accuracy: 0.9917 - val_loss: 0.0012 - val_accuracy: 1.0000
Epoch 29/32
196/196 [=====] - 199s 1s/step - loss: 0.0786 - accuracy: 0.9812 - val_loss: 0.0247 - val_accuracy: 0.9869
Epoch 30/32
196/196 [=====] - 190s 963ms/step - loss: 0.0536 - accuracy: 0.9858 - val_loss: 0.0018 - val_accuracy: 0.9994
Epoch 31/32
196/196 [=====] - 206s 1s/step - loss: 0.0537 - accuracy: 0.9866 - val_loss: 9.6809e-04 - val_accuracy: 0.9997
Epoch 32/32
196/196 [=====] - 198s 1s/step - loss: 0.0091 - accuracy: 0.9912 - val_loss: 0.0015 - val_accuracy: 0.9997
```

Evaluating Model

```
In [ ]: #Training set Accuracy
train_loss, train_acc = cnn.evaluate(training_set)
print('Training accuracy:', train_acc)
```

```
196/196 [=====] - 55s 276ms/step - loss: 8.5568e-04 - accuracy: 0.9998
Training accuracy: 0.999840497970581
```

```
In [ ]: #Validation set Accuracy
val_loss, val_acc = cnn.evaluate(validation_set)
print('Validation accuracy:', val_acc)
```

```
98/98 [=====] - 31s 304ms/step - loss: 0.0015 - accuracy: 0.9997
Validation accuracy: 0.9996799230575562
```

Saving Model

```
In [ ]: cnn.save('trained_model.h5')
```

```
In [ ]: training_history.history #Return Dictionary of history
```

```
0.9974392056465149,  
0.9843149781227112,  
0.9916773438453674,  
0.9891164898872375,  
0.9574263691902161,  
1.0,  
0.9955185651779175,  
0.7243918180465698,  
1.0,  
0.9990397095680237,  
0.9500640034675598,  
0.9849551916122437,  
1.0,  
0.9980793595314026,  
0.9993597865104675,  
1.0,  
0.9932778477668762,  
0.984635055065155,  
0.9958386421203613,  
1.0
```

```
In [ ]: #Recording History in json  
import json  
with open('training_hist.json','w') as f:  
    json.dump(training_history.history,f)
```

```
In [ ]: print(training_history.history.keys())  
  
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```

Calculating Accuracy of Model Achieved on Validation set

```
In [ ]: print("Validation set Accuracy: {}".format(training_history.history['val_accuracy'][-1]*100))
```

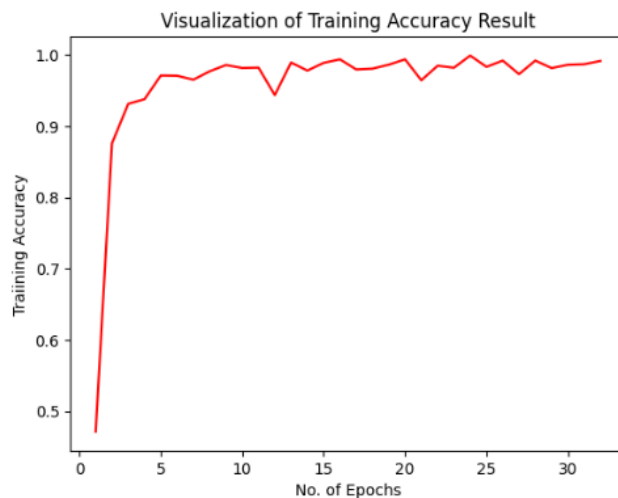
Validation set Accuracy: 99.96799230575562 %

Accuracy Visualization

Training Visualization

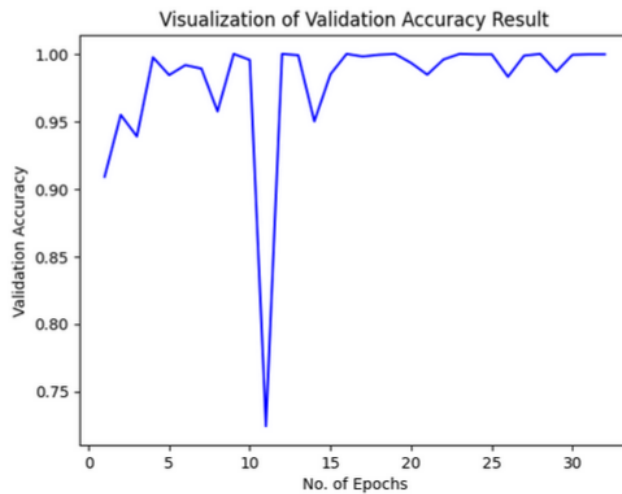
```
In [ ]: #training_history.history['accuracy']
```

```
In [ ]: epochs = [i for i in range(1,33)]  
plt.plot(epochs,training_history.history['accuracy'],color='red')  
plt.xlabel('No. of Epochs')  
plt.ylabel('Training Accuracy')  
plt.title('Visualization of Training Accuracy Result')  
plt.show()
```



Validation Accuracy

```
In [ ]: plt.plot(epochs, training_history.history['val_accuracy'], color='blue')
plt.xlabel('No. of Epochs')
plt.ylabel('Validation Accuracy')
plt.title('Visualization of Validation Accuracy Result')
plt.show()
```



Test set Evaluation

```
In [ ]: test_set = tf.keras.utils.image_dataset_from_directory(
    '/content/drive/MyDrive/fruit recognition dataset/Test',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(64, 64),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

Found 3110 files belonging to 24 classes.

```
In [ ]: test_loss, test_acc = cmn.evaluate(test_set)
print('Test accuracy:', test_acc)
```

```
98/98 [=====] - 710s 6s/step - loss: 0.0012 - accuracy: 0.9997
Test accuracy: 0.9996784329414368
```

TEST MODEL

```
In [ ]: import numpy as np
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
```

```
In [ ]: from google.colab import drive
drive.mount('/content/drive')
```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

Test set Image Processing

```
In [ ]: test_set = tf.keras.utils.image_dataset_from_directory(
    '/content/drive/MyDrive/fruit_recognition_dataset/Test',
    labels="inferred",
    label_mode="categorical",
    class_names=None,
    color_mode="rgb",
    batch_size=32,
    image_size=(64, 64),
    shuffle=True,
    seed=None,
    validation_split=None,
    subset=None,
    interpolation="bilinear",
    follow_links=False,
    crop_to_aspect_ratio=False
)
```

Found 3110 files belonging to 24 classes.

Loading Model

```
In [ ]: cnn = tf.keras.models.load_model('/content/trained_model.h5')
```

Visualising and Performing Prediction on Single image

```
In [ ]: #Test Image Visualization
import cv2
image_path = '/content/drive/MyDrive/fruit_recognition_dataset/Test/carrot_1/r0_103.jpg'
# Reading an image in default mode
img = cv2.imread(image_path)
img = cv2.cvtColor(img,cv2.COLOR_BGR2RGB) #Converting BGR to RGB
# Displaying the image
plt.imshow(img)
plt.title('Test Image')
plt.xticks([])
plt.yticks([])
plt.show()
```

Test Image



Testing Model

```
In [ ]: image = tf.keras.preprocessing.image.load_img(image_path,target_size=(64,64))
input_arr = tf.keras.preprocessing.image.img_to_array(image)
input_arr = np.array([input_arr]) # Convert single image to a batch.
predictions = cnn.predict(input_arr)
```

1/1 [=====] - 0s 175ms/step

```
In [ ]: print(predictions)
```

```
[[5.6397077e-14 1.9625935e-15 5.4985584e-13 4.8443112e-14 8.5600662e-19
 2.4620719e-18 2.8943546e-17 2.2892410e-15 7.3171894e-12 3.5323372e-10
 2.2302134e-15 4.2856148e-17 1.9071799e-12 8.2167563e-13 3.4516913e-16
 2.3774097e-17 1.0000000e+00 1.9194815e-09 4.1863877e-09 3.9464204e-14
 2.8468279e-19 6.3869084e-18 1.4307183e-16 4.7944746e-09]]
```

```
In [ ]: # test_set.class_names
```

```
In [ ]: result_index = np.argmax(predictions) #Return index of max element
print(result_index)
```

16

```
In [ ]: # Displaying the image
plt.imshow(img)
plt.title('Test Image')
plt.xticks([])
plt.yticks([])
plt.show()
```



```
In [ ]: s#Single image Prediction
print("It's a {}".format(test_set.class_names[result_index]))
```

It's a carrot_1

CODE LINK:

https://colab.research.google.com/drive/1k3o6ob3D1EC83ubymDWpSowqqkJ_haXC?usp=s_haring

GITHUB LINK: [DhanshreeRajput/Fruit-Recognition-System \(github.com\)](https://github.com/DhanshreeRajput/Fruit-Recognition-System)