A PROJECT REPORT

on

"RECIPE SUGGESTION BASED ON FOOD INGREDIENTS IMAGE CLASSIFICATION"

Submitted to

KIIT Deemed to be University

In Partial Fulfilment of the Requirement for the Award of

BACHELOR'S DEGREE IN

Computer Science and Engineering

SUBMITTED BY:

Aaryak Prasad	**2105171**
Raunak Roy Chowdhury	**21051076**
Abhay Singh	**2105172**
Shivli Singh	**2105237**
Mohd. Ayaan Khan	**2105209**

UNDER THE GUIDANCE OF:

Dr. Soumya Ranjan Nayak

SCHOOL OF COMPUTER ENGINEERING

KALINGA INSTITUTE OF INDUSTRIAL TECHNOLOGY

BHUBANESWAR, ODISHA - 751024

April 2024

KIIT Deemed to be University

School of Computer Engineering
Bhubaneswar, ODISHA 751024

CERTIFICATE

This is certify that the project entitled

"Recipe suggestion based on food ingredients image classification"

submitted by

Aaryak Prasad	2105171
Raunak Roy Chowdhury	21051076
Abhay Singh	2105172
Shivli Singh	2105237
Mohd. Ayaan Khan	2105209

is a record of bonafide work carried out by them, in the partial fulfillment of the requirement for the award of Degree of Bachelor of Engineering (Computer Science & Engineering OR Information Technology) at KIIT Deemed to be university, Bhubaneswar. This work is done during year 2022-2023, under our guidance.

Date: 12/04/2024

(Dr. Soumya Ranjan Nayak)
Project Guide

Acknowledgements

We are profoundly grateful to **Dr. Soumya Ranjan Nayak of KIIT University** for his expert guidance and continuous encouragement throughout to see that this project rights its target since its commencement to its completion.

Aaryak Prasad

Raunak Roy Chowdhury

Abhay Singh

Md. Ayaan Khan

Shivli Singh

ABSTRACT

Food is essential for human survival, and people always try to taste different types of delicious recipes. However, frequently, people choose food ingredients without even knowing their names or pick up some ingredients that aren't obvious to them from a grocery store. Selecting the right recipe by choosing a list of ingredients can be challenging for both beginners and experts. In this paper, we propose a solution to this problem by implementing a model for food ingredient recognition using deep learning techniques. We also design an algorithm for recommending recipes based on the recognized ingredients. Our custom dataset consists of 3861 images belonging to 36 different fruits and vegetables. We use a Convolutional Neural Network (CNN) model for ingredient identification and machine learning for recipe recommendations. The accuracy of all the models used in the project are **Resnet50**: 96.41%, **Vgg16**: 97.01%, **Xception:** 96.4%, **Yolo V8: (**Top_1)-94.017%

Keywords: Deep Learning, CNN, Ingredients Detection, Recipe Recommendation, Resnet50, YoloV8, Xception, VGG16.

Contents

List of Figures

Chapter 1

Introduction

The project aims to develop an intelligent system for recipe suggestions based on food ingredients image classification. With the increasing popularity of food photography and the desire for personalised meal planning, there is a growing need for efficient methods to recognize ingredients from food images and recommend suitable recipes. In this context, we have created a custom dataset comprising 3861 images representing 36 different fruits and vegetables commonly used in cooking. Our goal is to leverage deep learning techniques to accurately identify these ingredients and provide relevant recipe recommendations.

Our dataset consists of high-resolution images of fruits and vegetables captured under various lighting conditions and angles. Each class (ingredient) is well-represented, ensuring a balanced dataset. We manually labelled each image with the corresponding ingredient class. Once an image is classified into an ingredient category, we retrieve a list of recipes containing that ingredient through a LLM API prompt. We consider factors such as popularity, user preferences, and nutritional balance when recommending recipes. The recommendation system ensures that users receive diverse and appealing recipe options.

Our project combines image classification, deep learning, and recipe recommendation to enhance the culinary experience for users. By leveraging state-of-the-art models, we aim to provide accurate ingredient recognition and delightful recipe suggestions.

Chapter 2

Literature Review

In recent years, there has been significant research in the field of food image recognition and recipe recommendation. Here are some relevant studies:

1. **"Food Recipe Recommendation Based on Ingredients Detection Using Deep Learning" by Md. Shafaat Jamil Rokon, Md Kishor Morol, Ishra Binte Hasan, A. M. Saif, Rafid Hussain Khan**
    - This paper proposes an approach for recognizing food ingredients and recommending recipes based on those ingredients.
    - The authors create a custom dataset with 9856 images representing 32 different food ingredient classes.
    - They use a Convolutional Neural Network (CNN) model for ingredient identification and machine learning techniques for recipe recommendations.

2. **"A Smart Recipe Recommendation System Based on Image" by Seda Kul & Ahmet Sayar**
    - This study presents a smart recipe recommendation system that recognizes food ingredients from images.
    - The system suggests suitable meal recipes based on the recognized ingredients.
    - Object detection processes using deep learning are employed to capture food items from images.

3. **"Deep Learning for Food Image Recognition and Nutrition Analysis"**
    - A systematic review is conducted on the application of deep learning in food image recognition and nutrition analysis.
    - The study explores various deep learning architectures and their performance in recognizing food ingredients.

4. **"Using AI to Identify Ingredients and Suggest Recipes" by Vlad Ivanchuk, Manana Hakobyan, Coleman Hooper and Brian Hall**
    - This article discusses the use of AI to identify ingredients from food images and recommend recipes.
    - The author focuses on common ingredients and suggests recipes based on recognized

Chapter 3

Problem Statement:

3.1 PROJECT PROBLEM STATEMENT:

In today's culinary landscape, individuals often face the challenge of deciding what to cook based on the ingredients available in their kitchen. Whether it's a seasoned chef or a novice home cook, selecting the right recipe can be time-consuming and overwhelming. Additionally, food waste remains a pressing issue globally. People often buy ingredients without a clear plan, leading to unused items that eventually go to waste. To address these challenges, we propose a comprehensive solution that combines computer vision, deep learning, and personalized recipe recommendations.
Our goals are twofold:

1. **Ingredient Recognition:**
   - Develop an accurate ingredient recognition system that can identify various food items from images.
   - Users should be able to take photos of ingredients in their fridge or pantry and receive real-time identifications.
   - Ingredient recognition will create a comprehensive inventory of available items, reducing food waste and encouraging efficient meal planning.
2. **Recipe Recommendation:**
   - Once ingredients are recognized, our system should recommend suitable recipes.
   - Recommendations should consider user preferences (e.g., dietary restrictions, cuisine preferences) and nutritional balance.
   - The system should encourage users to explore new recipes, experiment with different ingredients, and make the most of what they have on hand.

3.2 Project Planning:

1. **Problem Definition and Scope:**
   - Clearly define the problem: ingredient recognition from food images and personalized recipe recommendations.
   - Specify the scope: focus on fruits and vegetables as ingredients.
   - Identify the target audience (home cooks, food enthusiasts, etc.).
2. **Data Collection and Preparation:**
   - Collect a diverse dataset of food ingredient images (e.g., fruits, vegetables) with proper labeling.
   - Preprocess the images (resize, normalize, augment) for model training.
   - Create a mapping of recognized ingredients to recipe databases.
3. **Model Selection and Training:**
   - Choose deep learning models (CNNs) suitable for image classification (e.g., ResNet, VGG, MobileNet).
   - Split the dataset into training, validation, and test sets.
   - Fine-tune pre-trained models on the custom dataset.
   - Evaluate model performance (accuracy, precision, recall).
4. **Ingredient Recognition System**:
   - Develop an interface (web or mobile app) for users to upload ingredient images.
   - Implement the trained model for real-time ingredient recognition.
   - Display recognized ingredients to the user.
5. **Recipe Recommendation through LLM APIs:**
   - Consider user preferences (e.g., dietary restrictions, cuisine type).
   - Feeds a custom prompt to a LLM API to fetch recipe recommendations.
6. **User Experience and Interface Design:**
   - Design an intuitive and user-friendly interface.
   - Include features like voice input, ingredient search, and personalized recommendations.
   - Ensure responsiveness for different devices (mobile, desktop).
7. **Testing and Validation:**
   - Conduct thorough testing of the system:
     - Accuracy of ingredient recognition.
     - Relevance and diversity of recipe recommendations.
   - Validate against user feedback and adjust algorithms if needed.
8. **Deployment and Maintenance:**
   - Deploy the system to a cloud server or hosting platform.
   - Monitor performance, scalability, and user engagement.
   - Regularly update the recipe database and retrain the model.
9. **Documentation and Reporting:**
   - Prepare detailed documentation:
   - Architecture diagrams.
   - Model training process.
   - API endpoints (if applicable).

3.3 System Design:

Our system aims to seamlessly assist users in selecting recipes based on available food ingredients. The architecture comprises three main components: Ingredient Recognition Service, Recipe Recommendation Service, and Mobile Application. Users interact with the mobile app to upload ingredient images. The Ingredient Recognition Service processes these images using ResNet-50, VGG16, Xception, and YOLOv8 models, providing recognized ingredients. The Recipe Recommendation Service fetches recipes through a LLM API like Gemini or ChatGPT through a custom prompt, considering user preferences. Finally, the mobile application fetches the output and reflects it to the user in an appealing manner. The system ensures real-time processing, scalability, and robustness, enhancing the culinary experience for users**.**

3.3.1 Design Constraints:

1. **Real-Time Processing:**

The system must process ingredient images in real-time to provide instant feedback to users.
Latency constraints should be considered during model inference.

2. **Scalability:**

The system should handle a large number of concurrent users.
Scalability considerations include load balancing, caching, and efficient resource utilization.

3. **Accuracy and Robustness:**

Ingredient recognition models (ResNet-50, VGG16, Xception, YOLOv8) must achieve high accuracy.
Robustness to variations in lighting, angles, and image quality is crucial.

4. **User Interface (UI):**

Web or mobile app where users upload ingredient images.
UI components include image upload, recognition results display, and recipe recommendations.

5. **Backend Services:**
   - Ingredient Recognition Service:
     - Receives uploaded images.
     - Utilizes ResNet-50, VGG16, Xception, and YOLOv8 models for ingredient recognition.
     - Aggregates results and provides recognized ingredients.
   - **Recipe Recommendation Service:**
     - Consider user preferences (e.g., dietary restrictions, cuisine type).
     - Feeds a custom prompt to a LLM API to fetch recipe recommendations.
6. **Database:**
   - Stores ingredient labels, recipe details, and user preferences.
   - Ensures efficient retrieval and updates.

7. **Model Management:**
    - Manages model versions, updates, and retraining.
    - Monitors model performance and accuracy.
8. **Security and Authentication:**
    - User authentication and authorization.
    - Secure data transmission (HTTPS).
9. **Scalability and Load Balancing:**
    - Distributes incoming requests across multiple servers.
    - Auto-scales based on traffic

3.3.2 System Architecture(UML):

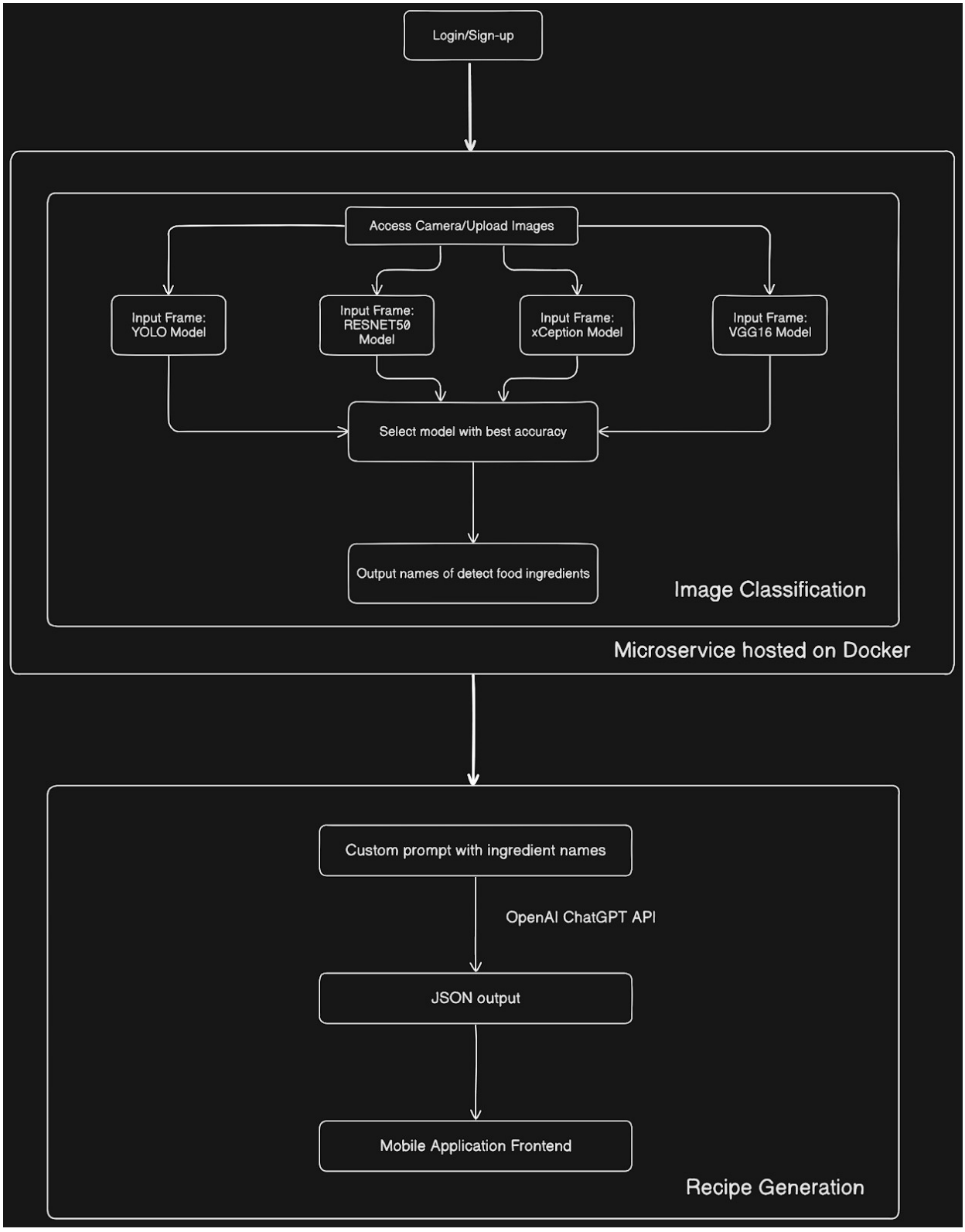


Fig: UML design for system architecture.

Chapter 4

Implementation:

4.1 Exploratory Data Analysis:

The following food products are depicted in the photographs in this dataset:

- fruits: watermelon, pomegranate, pineapple, mango, orange, kiwi, banana, apple, pear, and grapes.
- Vegetables: bell pepper, chilli pepper, turnip, corn, sweet corn, sweet potato, paprika, jalepeño, ginger, garlic, peas, eggplant, cucumber, carrot, capsicum, onion, potato, lemon, tomato, raddish, beetroot, cabbage, lettuce, spinach, soya bean, cauliflower, bell pepper, and sweet potato.

There are three folders in this dataset:

- train, each with 100 photos
- test, each with ten photos
- verification (10 pictures each) The photos for the corresponding food items are present in the subfolders for various fruits and vegetables in each of the aforementioned files.

The following food products are depicted in the photographs in this dataset:

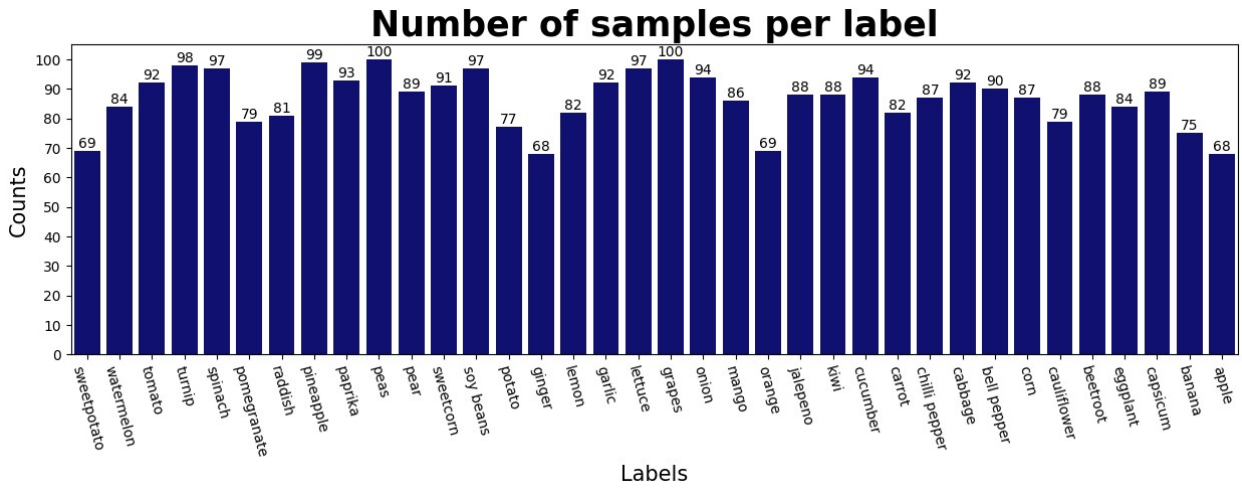
- fruits: watermelon, pomegranate, pineapple, mango, orange, kiwi, banana, apple, pear, and grapes.
- Vegetables: bell pepper, chilli pepper, turnip, corn, sweet corn, sweet potato, paprika, jalepeño, ginger, garlic, peas, eggplant, cucumber, carrot, capsicum, onion, potato, lemon, tomato, raddish, beetroot, cabbage, lettuce, spinach, soybean, cauliflower, bell pepper, and sweet potato.

```
import numpy as np
import pandas as pd
from pathlib import Path
import os.path
import matplotlib.pyplot as plt
import tensorflow as tf
data_dir = "/content/MyDrive/MyDrive/archive"
train_dir = os.path.join(data_dir, "train")
test_dir = os.path.join(data_dir, "test")
validation_dir = os.path.join(data_dir, "validation")
# Create a list with the filepaths for training, testing and validation
train_filepaths = list(Path(train_dir).glob(r'**/*.jpg'))
test_filepaths = list(Path(test_dir).glob(r'**/*.jpg'))
val_filepaths = list(Path(validation_dir).glob(r'**/*.jpg'))

def proc_img(filepath):
    """ Create a DataFrame with the filepath and the labels of the
pictures
    """

    labels = [str(filepath[i]).split("/")[-2] \
              for i in range(len(filepath))]
```

```
    filepath = pd.Series(filepath, name='Filepath').astype(str)
    labels = pd.Series(labels, name='Label')

    # Concatenate filepaths and labels
    df = pd.concat([filepath, labels], axis=1)

    # Shuffle the DataFrame and reset index
    df = df.sample(frac=1).reset_index(drop = True)

    return df

train_df = proc_img(train_filepaths)
test_df = proc_img(test_filepaths)
val_df = proc_img(val_filepaths)
print('-- Training set --\n')
print(f'Number of pictures: {train_df.shape[0]}\n')
print(f'Number of different labels: {len(train_df.Label.unique())}\n')
print(f'Labels: {train_df.Label.unique()}')
```

```
-- Training set --

Number of pictures: 2790

Number of different labels: 36

Labels: ['cauliflower' 'lettuce' 'beetroot' 'onion' 'raddish' 'turnip' 'banana'
 'sweetcorn' 'cabbage' 'garlic' 'eggplant' 'carrot' 'soy beans' 'orange'
 'jalepeno' 'pomegranate' 'spinach' 'mango' 'pear' 'sweetpotato' 'tomato'
 'corn' 'capsicum' 'peas' 'bell pepper' 'chilli pepper' 'pineapple'
 'ginger' 'potato' 'kiwi' 'paprika' 'grapes' 'lemon' 'apple' 'watermelon'
 'cucumber']
```

Distribution of images in training dataset:



4.2 Methodology / Proposal:

The methodology involves the following steps:

- Installing necessary libraries and dependencies, such as silence_tensorflow, pandas, matplotlib, and tensorflow.
- Mounting a Google Drive to access the dataset.
- Loading the dataset from the specified directories (train, test, and validation).
- Creating DataFrames for the training, testing, and validation sets with the file paths and corresponding labels.
- Exploring the dataset by displaying the number of samples per label, and visualizing some sample images.
- Preprocessing the images using the preprocess_input function from the tf.keras.applications.vgg16 module.
- Training various pre-trained models, such as VGG16, Xception, and ResNet,YOLOv8 on the dataset.
- Making predictions on the test set using the trained models.
- Evaluating the performance of the models using confusion matrices and visualizing the predictions on sample images.

4.2.1 VGG_16 Model:

Data augmentation:

This code is applied to a machine learning project for the purpose of preparing images and augmenting data. It generates batches of tensor image data with real-time data augmentation using the Keras API provided by TensorFlow. This is an explanation:

- A class called ImageDataGenerator uses real-time data augmentation to create batches of tensor image data. Batch processing of the data will continue indefinitely.
- preprocessing_function: Every input image will undergo this function's application. Here, the preprocess_input function from vgg16 is being used, which applies preprocessing unique to the VGG16 model, like colour scaling.
- flow_from_dataframe: This function creates batches of augmented/normalized data from a dataframe and a directory path. It connects the picture files in the dataframe directly to their labels.

The parameters used to augment data include rotation_range, zoom_range, shear_range, horizontal_flip, height_shift, and fill_mode. To generate more data and lessen the likelihood of overfitting, they are employed to haphazardly alter the photos.

The photos in train, val, and test are iterators that will produce labels and batches of photos. The data is shuffled and supplemented for training and validation, however it is just preprocessed and not jumbled for testing.

Train the Model:

```
pretrained_model = tf.keras.applications.VGG16(
    input_shape=(416, 416, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
pretrained_model.trainable = False
inputs = pretrained_model.input

x = tf.keras.layers.Dense(128,
activation='relu')(pretrained_model.output)
x = tf.keras.layers.Dense(128, activation='relu')(x)

outputs = tf.keras.layers.Dense(36, activation='softmax')(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history = model.fit(
    train_images,
    validation_data=val_images,
    batch_size = 32,
    epochs=10,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True
        )
    ]
)
```

This code creates a machine learning model using the Keras API provided by TensorFlow. For feature extraction, it makes use of a pretrained VGG16 model whose weights were trained on ImageNet. To maintain the learned features, the layers of the pretrained model are frozen. A dense output layer divides photos into one of 36 classes, and two more dense layers are added for additional learning. The accuracy metric, categorical cross-entropy loss, and Adam optimizer are used to assemble the model. It is trained for ten epochs with early halting on a dataset. When the validation loss remains unchanged for two epochs, the training terminates and the optimal weights are recovered. The purpose of this model is to categorize photos into one of 36 groups.

```
Epoch 1/10
88/88 [==============================] - 1829s 21s/step - loss: 3.0001 - accuracy: 0.3527 - val_loss: 0.7948 - val_accuracy: 0.7725
Epoch 2/10
88/88 [==============================] - 1780s 20s/step - loss: 0.9657 - accuracy: 0.7280 - val_loss: 0.4362 - val_accuracy: 0.8653
Epoch 3/10
88/88 [==============================] - 1774s 20s/step - loss: 0.5467 - accuracy: 0.8341 - val_loss: 0.2793 - val_accuracy: 0.9341
Epoch 4/10
88/88 [==============================] - 1792s 20s/step - loss: 0.3485 - accuracy: 0.8918 - val_loss: 0.2201 - val_accuracy: 0.9461
Epoch 5/10
88/88 [==============================] - 1798s 20s/step - loss: 0.2296 - accuracy: 0.9315 - val_loss: 0.2354 - val_accuracy: 0.9371
Epoch 6/10
88/88 [==============================] - 1817s 21s/step - loss: 0.1505 - accuracy: 0.9573 - val_loss: 0.1888 - val_accuracy: 0.9611
Epoch 7/10
88/88 [==============================] - 1804s 21s/step - loss: 0.1146 - accuracy: 0.9706 - val_loss: 0.1822 - val_accuracy: 0.9581
Epoch 8/10
88/88 [==============================] - 1807s 21s/step - loss: 0.0903 - accuracy: 0.9763 - val_loss: 0.1778 - val_accuracy: 0.9521
Epoch 9/10
88/88 [==============================] - 1804s 21s/step - loss: 0.0739 - accuracy: 0.9803 - val_loss: 0.1881 - val_accuracy: 0.9521
Epoch 10/10
88/88 [==============================] - 1801s 20s/step - loss: 0.0568 - accuracy: 0.9853 - val_loss: 0.1690 - val_accuracy: 0.9701
```

An essential algorithmic hyperparameter is the number of this epoch. It indicates how many epochs, or full runs through the whole training dataset, the algorithm will go through throughout its training or learning process.

4.2.2 Xception Model:

Data augmentation:

A class called ImageDataGenerator uses real-time data augmentation to create batches of tensor image data. Batch processing of the data will continue indefinitely.

- **preprocessing_function:** Every input image will undergo this function's application. Here, the preprocess_input function from Xception is being used, which applies preprocessing unique to the Xception model, like color scaling.
- **flow_from_dataframe:** This function creates batches of augmented/normalized data from a dataframe and a directory path. It links the image files in the data frame directly to their labels.

The parameters used to augment data include rotation_range, zoom_range, shear_range, horizontal_flip, height_shift, and fill_mode. To generate more data and lessen the likelihood of overfitting, they are employed to haphazardly alter the photos.

the photos in train, val, and test: These are iterators that will produce labels and batches of photos. The data is shuffled and supplemented for training and validation, however it is just preprocessed and not jumbled for testing.

Train the Model:

```
pretrained_model_xception = tf.keras.applications.Xception(
    input_shape=(299, 299, 3),
    include_top=False,
    weights='imagenet',
    pooling='avg'
)
pretrained_model_xception.trainable = False

inputs = pretrained_model_xception.input

x = tf.keras.layers.Dense(128,
activation='relu')(pretrained_model_xception.output)
```

```
x = tf.keras.layers.Dense(128, activation='relu')(x)
outputs = tf.keras.layers.Dense(36, activation='softmax')(x)
model_xception = tf.keras.Model(inputs=inputs, outputs=outputs)
model_xception.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history_xception= model_xception.fit(
    train_images,
    validation_data=val_images,
    batch_size = 32,
    epochs=10,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True
        )
    ]
)
```

This code uses the Keras API provided by TensorFlow to generate a machine learning model. For feature extraction, it employs a pretrained Xception model with ImageNet weights. To maintain their acquired features, the model's layers are frozen. A dense output layer divides photos into one of 36 classes, and two more dense layers are added for additional learning. The accuracy metric, categorical cross-entropy loss, and Adam optimizer are used to assemble the model. It is trained for ten epochs with early halting on a dataset. When the validation loss remains unchanged for two epochs, the training terminates and the optimal weights are recovered. The purpose of this model is to categorize photos into one of 36 groups.

```
Epoch 1/10
88/88 [==============================] - 1381s 16s/step - loss: 1.7980 - accuracy: 0.5136 - val_loss: 0.5874 - val_accuracy: 0.8174
Epoch 2/10
88/88 [==============================] - 1410s 16s/step - loss: 0.6225 - accuracy: 0.8007 - val_loss: 0.3482 - val_accuracy: 0.8952
Epoch 3/10
88/88 [==============================] - 1405s 16s/step - loss: 0.4325 - accuracy: 0.8559 - val_loss: 0.3102 - val_accuracy: 0.8862
Epoch 4/10
88/88 [==============================] - 1395s 16s/step - loss: 0.3081 - accuracy: 0.8928 - val_loss: 0.2409 - val_accuracy: 0.9341
Epoch 5/10
88/88 [==============================] - 1393s 16s/step - loss: 0.2512 - accuracy: 0.9129 - val_loss: 0.2399 - val_accuracy: 0.9251
Epoch 6/10
88/88 [==============================] - 1406s 16s/step - loss: 0.1912 - accuracy: 0.9355 - val_loss: 0.1904 - val_accuracy: 0.9311
Epoch 7/10
88/88 [==============================] - 1322s 15s/step - loss: 0.1530 - accuracy: 0.9466 - val_loss: 0.2027 - val_accuracy: 0.9311
Epoch 8/10
88/88 [==============================] - 1316s 15s/step - loss: 0.1603 - accuracy: 0.9405 - val_loss: 0.1625 - val_accuracy: 0.9641
Epoch 9/10
88/88 [==============================] - 1309s 15s/step - loss: 0.1065 - accuracy: 0.9659 - val_loss: 0.1597 - val_accuracy: 0.9611
Epoch 10/10
88/88 [==============================] - 1305s 15s/step - loss: 0.1056 - accuracy: 0.9652 - val_loss: 0.1736 - val_accuracy: 0.9641
```

4.2.3 RESNET50 Model:

The code sets up data augmentation for the training, validation, and test sets using the *`tf.keras.preprocessing.image.ImageDataGenerator`* class. The training data generator applies various transformations to the images, including random rotation, zoom, width and height shifts, shear, and horizontal flipping. These transformations aim to increase the diversity of the training data, which can help the model generalize better and perform well on new, unseen data.
The validation data generator applies the same transformations as the training data generator, while the test data generator only applies the preprocessing function without any other transformations. The purpose of this data augmentation is to improve the model's performance by exposing it to a wider range of variations in the training data, which can lead to better generalization and more robust predictions.
Train the model:

```
inputs = pretrained_model_resnet50.input

x = tf.keras.layers.Dense(128,
activation='relu')(pretrained_model_resnet50.output)
x = tf.keras.layers.Dense(128, activation='relu')(x)

outputs = tf.keras.layers.Dense(36, activation='softmax')(x)
model_resnet = tf.keras.Model(inputs=inputs, outputs=outputs)
model_resnet.compile(
    optimizer='adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

history_resnet = model_resnet.fit(
    train_images,
    validation_data=val_images,
    batch_size = 32,
    epochs=10,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='val_loss',
            patience=2,
            restore_best_weights=True
        )
    ]
)
```

The code sets up a transfer learning approach using the pre-trained ResNet50 model as the base model. It begins by extracting the input layer of the pre-trained model and assigning it to the `inputs` variable. Then, it adds two dense layers, each with 128 units and a ReLU activation function, to the output of the pre-trained model. These additional layers are intended to introduce more nonlinearity and learn higher-level features specific to the given classification task.

Finally, a dense output layer with 36 units and a softmax activation function is added to produce the final classification output.
The resulting model, named `model_resnet`, is then compiled with the Adam optimizer, categorical cross-entropy loss, and accuracy as the evaluation metric. The model is then trained on the `train_images` data, which was previously generated using the `ImageDataGenerator` class and data augmentation techniques. The `validation_data` parameter is set to the `val_images` data, allowing the model to be evaluated on the validation set during training.

The training process is set to run for a maximum of 10 epochs, with the inclusion of an early stopping callback. The early stopping callback monitors the validation loss and stops the training if the validation loss does not improve for 2 consecutive epochs, restoring the weights of the best-performing model. This technique helps prevent overfitting and ensures that the model is trained until it reaches a point of optimal performance on the validation set.
Overall, this code sets up a transfer learning approach using the pre-trained ResNet50 model, with additional dense layers added to fine-tune the model for the specific classification task. The data augmentation and early stopping techniques are employed to improve the model's generalization and prevent overfitting, ultimately aiming to achieve the best possible performance on the given dataset.

```
Epoch 1/10
88/88 [==============================] - 2564s 29s/step - loss: 1.5843 - accuracy: 0.5720 - val_loss: 0.4072 - val_accuracy: 0.8713
Epoch 2/10
88/88 [==============================] - 697s 8s/step - loss: 0.5080 - accuracy: 0.8398 - val_loss: 0.2392 - val_accuracy: 0.9132
Epoch 3/10
88/88 [==============================] - 684s 8s/step - loss: 0.2977 - accuracy: 0.8996 - val_loss: 0.1965 - val_accuracy: 0.9371
Epoch 4/10
88/88 [==============================] - 696s 8s/step - loss: 0.1914 - accuracy: 0.9394 - val_loss: 0.1800 - val_accuracy: 0.9521
Epoch 5/10
88/88 [==============================] - 692s 8s/step - loss: 0.1206 - accuracy: 0.9677 - val_loss: 0.1668 - val_accuracy: 0.9491
Epoch 6/10
88/88 [==============================] - 679s 8s/step - loss: 0.1380 - accuracy: 0.9588 - val_loss: 0.1691 - val_accuracy: 0.9461
Epoch 7/10
88/88 [==============================] - 687s 8s/step - loss: 0.0746 - accuracy: 0.9742 - val_loss: 0.1621 - val_accuracy: 0.9611
Epoch 8/10
88/88 [==============================] - 681s 8s/step - loss: 0.0689 - accuracy: 0.9781 - val_loss: 0.1421 - val_accuracy: 0.9641
Epoch 9/10
88/88 [==============================] - 671s 8s/step - loss: 0.0526 - accuracy: 0.9853 - val_loss: 0.1608 - val_accuracy: 0.9611
Epoch 10/10
88/88 [==============================] - 670s 8s/step - loss: 0.0464 - accuracy: 0.9867 - val_loss: 0.1353 - val_accuracy: 0.9641
```

4.2.4 YOLO V8 MODEL:

Train the model:

```
from ultralytics import YOLO

data_dir= r'D:\Programming\Projects\Minor Project\DataSet\archive'
model=YOLO('yolov8n-cls.pt')

result=model.train(data = data_dir,epochs=10,imgsz=640)
```

```
from ultralytics import YOLO
```

This line imports the YOLO class from the Ultralytics library, which provides a user-friendly interface for training and using YOLO (You Only Look Once) models. YOLO is a state-of-the-art object detection and classification algorithm that has gained widespread popularity due to its high accuracy and real-time performance.

```
data_dir= r'D:\Programming\Projects\Minor Project\DataSet\archive'
```

This line specifies the directory path where the dataset for training the YOLO model is located. In this case, the dataset is stored in the 'archive' folder within the 'DataSet' directory of the project. The r prefix before the string indicates that it is a raw string literal, which ensures that backslashes are treated as literal characters instead of escape sequences.

```
model=YOLO('yolov8n-cls.pt')
```

This line creates an instance of the YOLO class and initializes it with the pre-trained model weights 'yolov8n-cls.pt'. The 'yolov8n-cls.pt' file is a pre-trained YOLO model specifically designed for classification tasks. The 'yolov8n' part refers to the YOLO version and architecture, while 'cls' indicates that it's a classification model.

```
result=model.train(data = data_dir,epochs=10,imgsz=640)
```

This line initiates the training process for the YOLO model using the specified dataset and training parameters. The model.train() method is called with the following arguments:

- data=data_dir: Specifies the directory containing the dataset to be used for training the model.
- epochs=10: Sets the number of training epochs to 10. An epoch is one complete pass through the entire training dataset.
- imgsz=640: Sets the input image size to 640 pixels. The images will be resized to this resolution before being fed into the model during training.

The model.train() method trains the YOLO model on the provided dataset for the specified number of epochs, using the given image size. During the training process, the model's weights are updated based on the training data and the loss function, with the goal of minimizing the prediction error.
After the training is complete, the result variable will contain the training results, which can include various metrics such as the final model weights, training loss, validation metrics, and more. These results can be further analyzed and utilized for evaluating the model's performance or deploying it for inference tasks.

4.3 Testing / Verification Plan:

The code includes the following testing and verification steps:

- Splitting the dataset into training, testing, and validation sets.
- Displaying the number of samples and labels in the training set to verify the dataset loading process.
- Visualizing sample images from the training set to manually verify the data quality.
- Evaluating the performance of the trained models on the test set using predictions and confusion matrices.
- Visualizing the true labels and predicted labels for sample images from the test set to manually verify the model's performance.

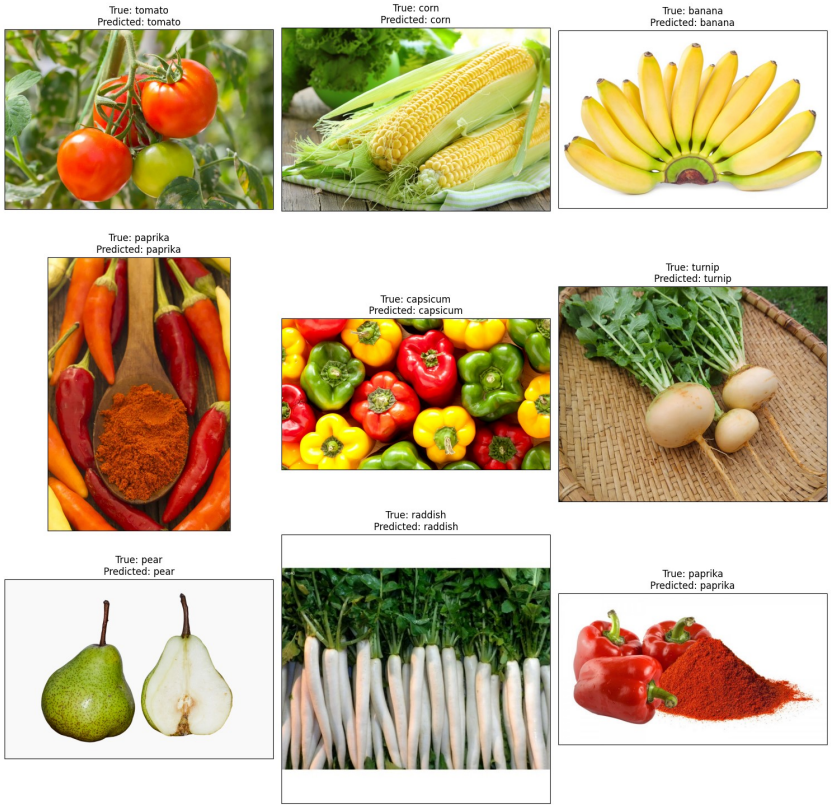
4.3.1 VGG_16 Model:

```
from tensorflow.keras.preprocessing import image
import numpy as np

def predict_image_class(model, image_path, show=True):
    img = image.load_img(image_path, target_size=(416, 416))
    img_array = image.img_to_array(img)
    expanded_img_array = np.expand_dims(img_array, axis=0)
    preprocessed_img =
tf.keras.applications.vgg16.preprocess_input(expanded_img_array)
    prediction = model.predict(preprocessed_img)
    predicted_class = np.argmax(prediction, axis=1)

    class_indices = train_images.class_indices
    # Invert the dictionary
    index_to_class = {v: k for k, v in class_indices.items()}
    # Get the class name
    predicted_class_name = index_to_class[predicted_class[0]]

    if show:
        plt.imshow(img)
        plt.axis('off')
        plt.title(f'Predicted: {predicted_class_name}')
        plt.show()

    return predicted_class_name

predict_image_class(model_resnet,'/content/MyDrive/MyDrive/archive/test/
grapes/Image_2.jpg')
```

The code defines the function predict_image_class, which accepts as inputs a boolean flag show, an image path, and a trained model. It makes use of the model to forecast the image's class along the specified path:

- **Image Loading and Preprocessing:** Using the preprocess_input function from VGG16, the image is preprocessed and loaded with a target size of 416x416 pixels. It is then turned to an array.
- **Prediction:** To obtain the prediction probabilities for each class, the preprocessed image is run through the model. The predicted class is determined by taking the class with the highest likelihood.

- **Class Name Retrieval:** Using the class_indices dictionary from the training images, the predicted class index is mapped back to its matching class name.
- **Image Display**: The original image is shown with the anticipated class name as the title if the show is True.



4.3.2 Xception Model:

```
# Predict the label of the test_images
pred = model_xception.predict(test_images)
pred = np.argmax(pred,axis=1)

# Map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred = [labels[k] for k in pred]

y_test = [labels[k] for k in test_images.classes]

# Display some pictures of the dataset with their labels and the
predictions
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[i]))
    ax.set_title(f"True: {test_df.Label.iloc[i]}\nPredicted: {pred[i]}")
plt.tight_layout()
plt.show()
```

This method uses a pre-trained model_xception to predict labels for test photos. The anticipated labels are then mapped back to the original class names. Additionally, the class names of the test images are mapped to their genuine labels. Lastly, it displays a portion of the test photos together with the predicted and true labels. Each image has a caption that lists both the true and anticipated labels. The images are shown in a 3x3 grid. This visual aid facilitates comprehension of the model's output on certain test photos. The code handles the model's output using numpy and creates the charts using matplotlib. The performance of the model is assessed by comparing its predictions with the actual labels.

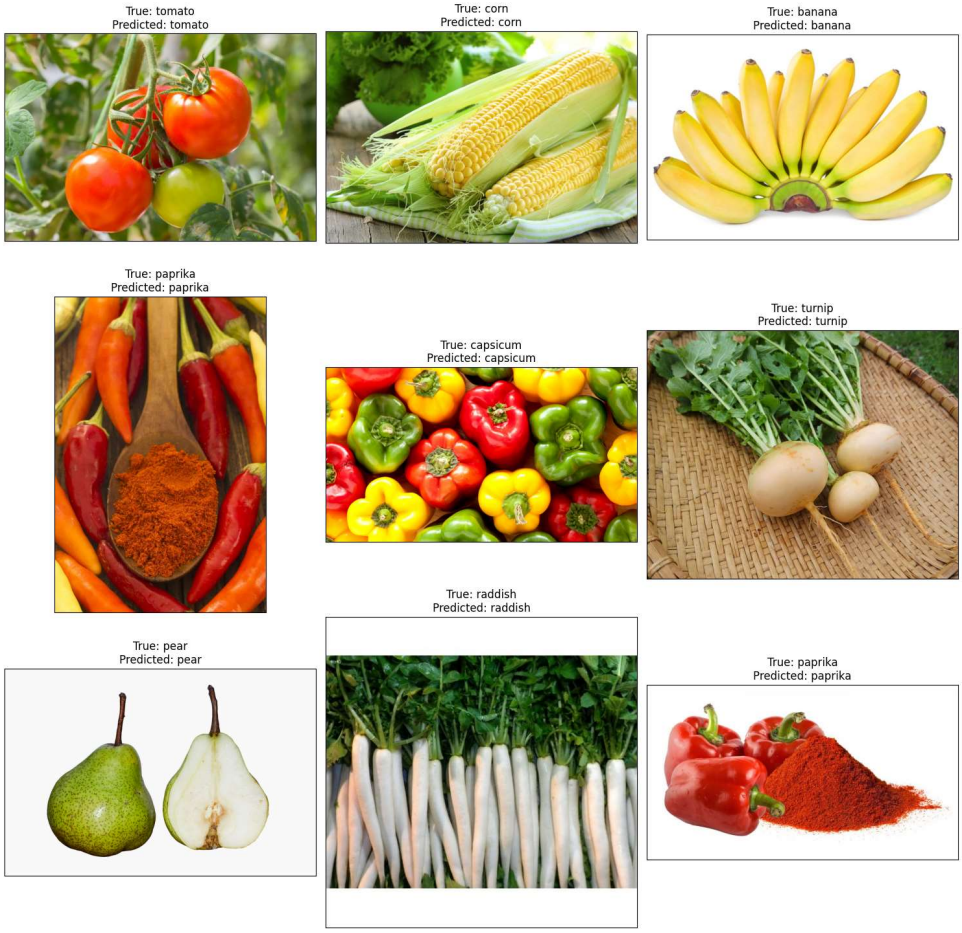
4.3.3 RESNET50 Model:

```
# Predict the label of the test_images
pred = model_resnet.predict(test_images)
pred = np.argmax(pred,axis=1)

# Map the label
labels = (train_images.class_indices)
labels = dict((v,k) for k,v in labels.items())
pred = [labels[k] for k in pred]

y_test = [labels[k] for k in test_images.classes]

# Display some pictures of the dataset with their labels and the
predictions
fig, axes = plt.subplots(nrows=3, ncols=3, figsize=(15, 15),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
    ax.imshow(plt.imread(test_df.Filepath.iloc[i]))
    ax.set_title(f"True: {test_df.Label.iloc[i]}\nPredicted: {pred[i]}")
plt.tight_layout()
plt.show()
```

This code is focused on evaluating the performance of the trained model_resnet on the test data. It first uses the model_resnet.predict() function to obtain the model's predictions for the test images. These predicted labels are then mapped back to the corresponding class names using the train_images.class_indices dictionary, which was previously created during the data preprocessing step.

Next, the code displays a 3x3 grid of sample test images, along with the true labels and the model's predicted labels for each image. This visual representation allows the user to quickly assess the model's performance and identify any misclassified examples. The true labels are obtained by mapping the test_images.classes indices to the class names using the same dictionary.

The purpose of this visualization is to provide a more interpretable and intuitive way to understand the model's strengths and weaknesses. By inspecting the correctly and incorrectly classified examples, the user can gain insights into the model's performance and identify areas for potential improvement. This step is an important part of the model evaluation process, as it complements the quantitative metrics, such as accuracy, that are typically used to assess the model's performance.



4.3.4 YOLO V8 Model:

```
from ultralytics import YOLO
import cv2
import streamlit as st
from PIL import Image
import numpy as np


modelpath =r"D:\Programming\Projects\Minor
Project\Yolo\runs\classify\train\weights\best.pt"
model=YOLO(modelpath)

st.title('Drop your image')
image=st.file_uploader('upload image',type=['png','jpg','jpeg','gif'])
if image:
    image=Image.open(image)
    st.image(image=image)
    result=model(image)
    names=result[0].names
    probability=result[0].probs.data.numpy()
    prediction=np.argmax(probability)
    st.write(f'Prediction is {names[prediction].upper()}')
```

This python code creates a simple web application using the Streamlit library. The application allows users to upload an image, which is then processed by a pre-trained YOLO (You Only Look Once) object detection and classification model. The YOLO model is loaded from a specified file path containing the trained model weights.
When a user uploads an image through the web interface, the code opens and displays the image using the Python Imaging Library (PIL) and Streamlit's image display functionality. The uploaded image is then passed to the YOLO model for prediction.
The YOLO model processes the image and returns the predicted class labels along with their corresponding probabilities. The code extracts the class names and probabilities from the model's output and identifies the class with the highest probability using NumPy's argmax function.
Finally, the predicted class label (with the highest probability) is displayed on the web interface using Streamlit's text output functionality.
The code utilizes various Python libraries, including ultralytics (for working with YOLO models), OpenCV (for computer vision tasks), Streamlit (for building interactive web applications), PIL (for image processing), and NumPy (for numerical computing).
This web application can be useful for demonstrating the capabilities of the trained YOLO model and allowing users to interact with it by uploading their own images for classification.

Here are the some of the Tested Images after Prediction:

4.4 Result Analysis:

The code provides the following result analysis and screenshots:

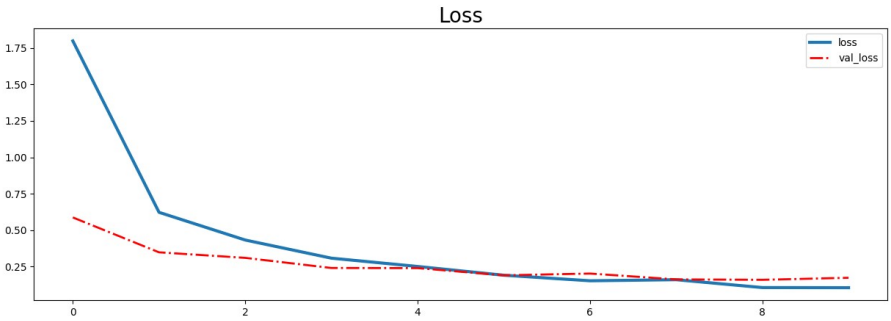
- A bar plot showing the number of samples per label in the training set.
- A grid of sample images from the training set, displaying the corresponding labels.
- A grid of sample images from the test set, showing the true labels and predicted labels by the trained models.
- Confusion matrices visualized as heatmaps, showing the normalized classification performance of the models on the test set.

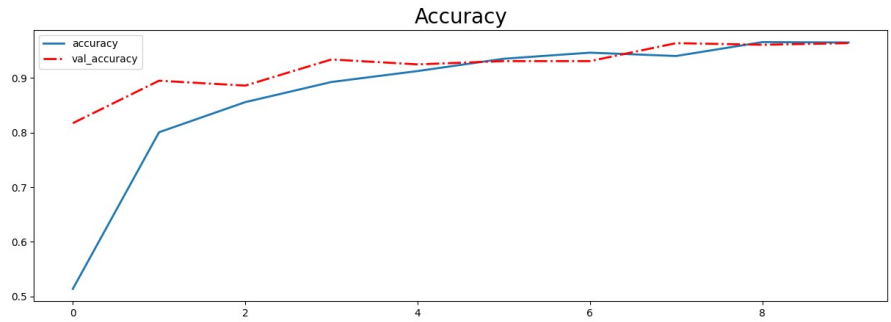
4.4.1 VGG_16 MODEL:

Loss Accuracy graph:

This shows the VGG-16 model's accuracy and loss throughout training and validation over epochs. It plots two graphs after reading from "xception.json," which holds the model's training history:

- Loss Graph: Shows the training and validation losses for the model over several epochs. The validation loss indicates how effectively the model generalizes to previously unseen data, while the training loss reduces as the model learns.
- The model's training and validation accuracy over several epochs is displayed on the accuracy graph. The model's performance on the validation set is indicated by the validation accuracy, which rises with learning.
- These graphs track the learning process of the model and identify problems such as overfitting. Effective learning is demonstrated by the graphic, which demonstrates a decreasing loss and a rising accuracy with time. It may be necessary to modify the model or the training procedure if the validation metrics plateau or deviate from the training metrics. Training measurements are represented by the solid lines, and validation metrics are represented by the dashed lines. The training and validation metrics are distinguished by the use of distinct colors and line styles. The meaning of each phrase is made clear by the legends.

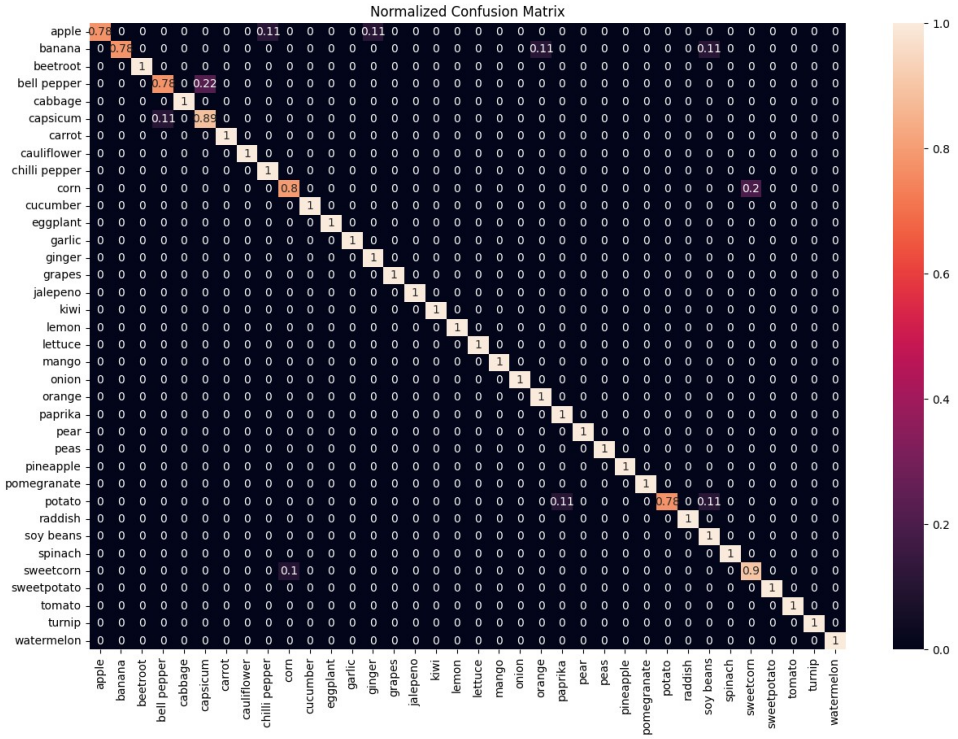




Confusion Matrix:

The VGG16 model's performance in classifying fruits and vegetables is assessed using the confusion matrix you have supplied. The model has 36 classes. The elements that make it up are as follows:

- Diagonal Values: Correct predictions where the model correctly identified the class are shown by the cells that run diagonally from top left to bottom right. Idealistically, these numbers are high, signifying strong performance.
- Off-Diagonal Values: When an input is misclassified by the model, it is indicated by cells that are not on the diagonal. High values suggest confusion between classes, hence these values should be as low as possible.
- Rows: A fruit or vegetable's true class, such as apple, banana, or carrot, is represented by each row.
- Columns: For those fruits or vegetables, each column shows the class that the model predicts.
- Columns: For those fruits or vegetables, each column shows the class that the model predicts.
- Color Bar: Darker colors often correspond to greater values. The color bar shows the range of values in the matrix.

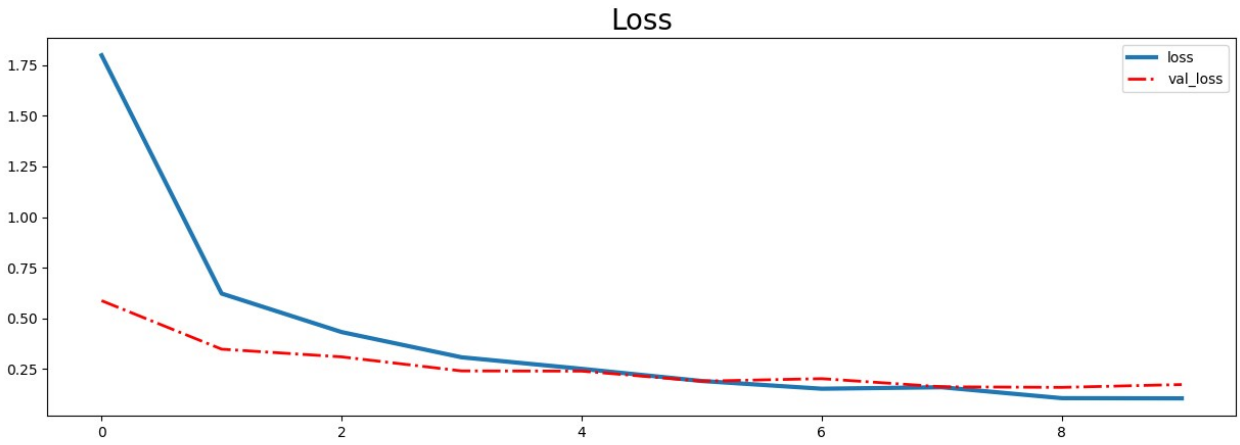


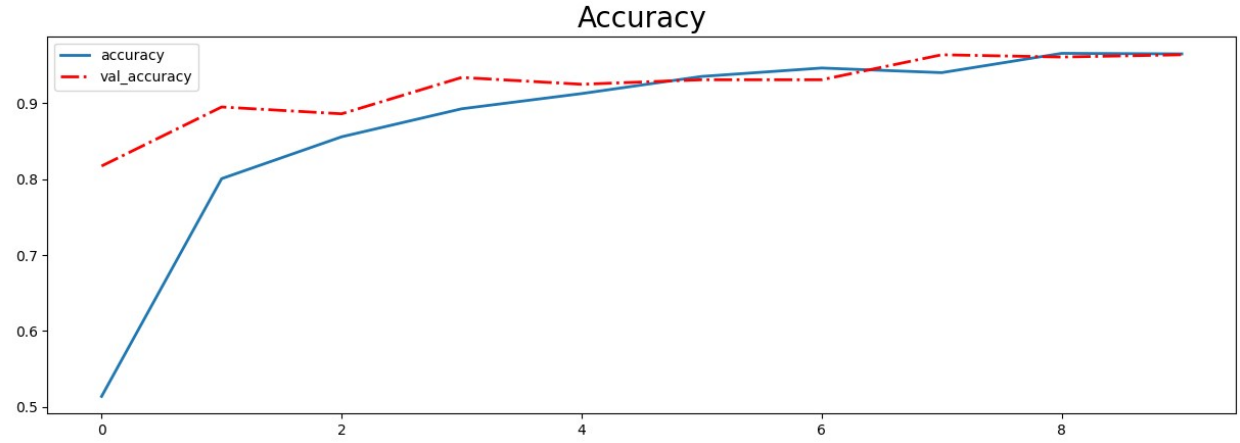
4.4.2 Xception Model:

Loss Accuracy graph:

Using a test dataset of photos of fruits and vegetables, the code is used to forecast and visualise the performance of a pre-trained Xception model:

- Prediction: Test_images are subjected to the model's predict procedure in order to produce predictions. A probability distribution over the 36 classes for every image is the result. The class with the highest probability is chosen as the model's prediction using the np.argmax function.
- Label Mapping: train_images.class_indices is where the labels for the numerical classes are obtained. To map the numerical predictions back to their original labels, this dictionary is reversed. The test photos' true labels undergo the same treatment.
- Visualisation: plt.subplots is used to construct a 3x3 grid of subplots. An image from the test dataset, its true label, and the predicted label of the model are shown in each subplot. The subplots are kept apart by using the tight_layout function.

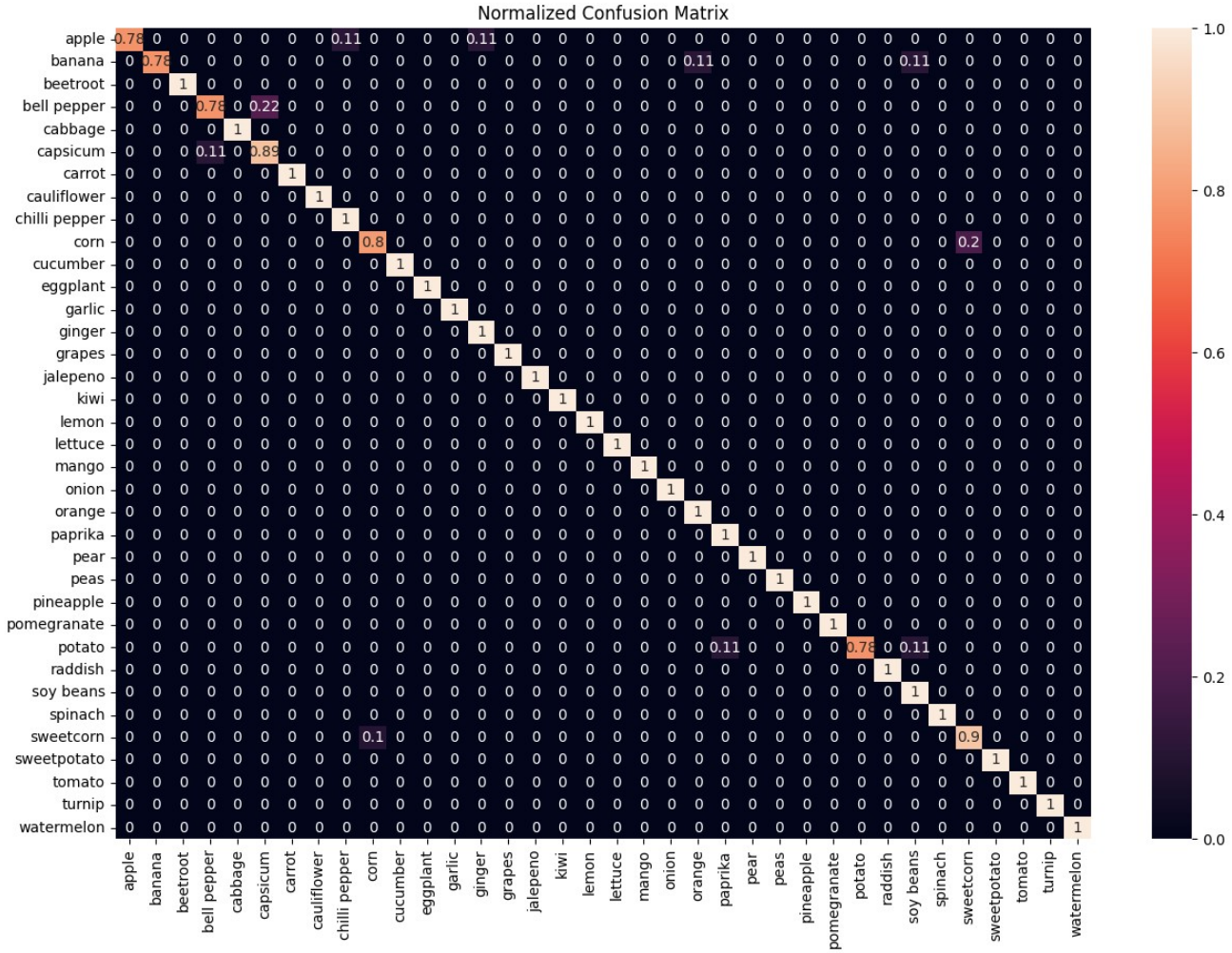




Confusion Matrix:

The code creates a normalized confusion matrix for a machine learning model's predictions. This is an explanation:

- Seaborn and sklearn.metrics are imported libraries. Seaborn is a matplotlib-based data visualization toolkit, and confusion_matrix is a function that calculates the confusion matrix to assess a classification's correctness.
- Calculate the Confusion Matrix: The normalize='true' argument, which normalizes the confusion matrix over the true (rows), false (columns), or entire population, is passed to the confusion_matrix function along with the true labels (y_test), predicted labels (pred), and other parameters. The outcome is kept in the cf_matrix.
- Create Figure: Using plt.figure, a new figure with a given size is made.
- Heatmap: Using sns.heatmap, a heatmap of the confusion matrix is produced. The real values can be written on each cell thanks to the annot=True option. The test set's sorted unique labels correspond to the x and y tick labels.



4.4.3 RESNET50 Model:

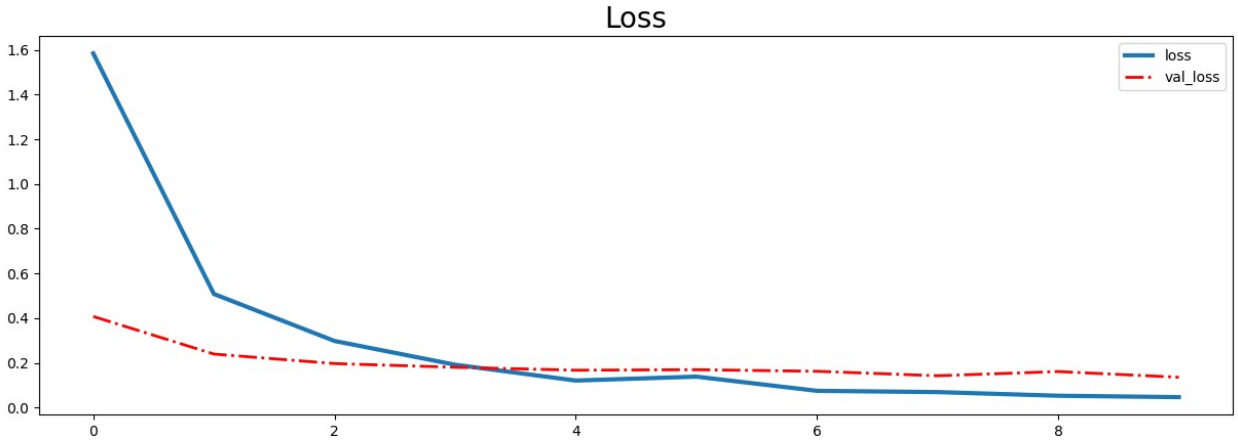
Loss Accuracy Graph:

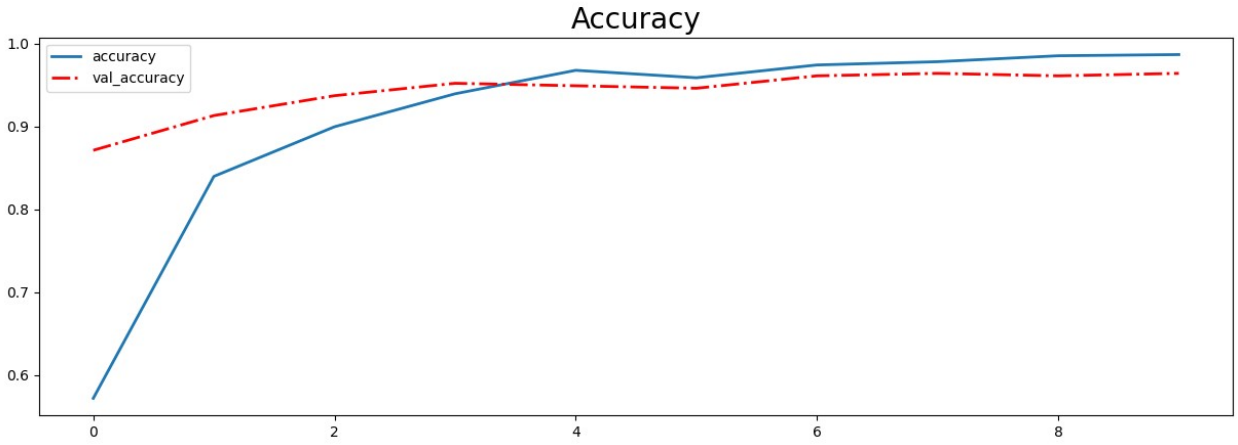
The training loss tells us how well the model is doing on the data it's currently being trained on. The validation loss, on the other hand, tells us how well the model performs on unseen data, which is more important in the real world. Ideally, both these lines should go down over time (epochs) as the model learns to recognize ingredients better. The lower the loss, the better the model is performing.

There's another line on the graph for accuracy, also with blue for training and red dashed for validation. Accuracy simply means how often the model guesses the correct ingredient. Just like the loss lines, we want the accuracy lines to go up over time, indicating the model is getting better at recognizing ingredients correctly.

Overfitting happens when the model gets so good at recognizing the training images that it starts memorizing them instead of learning the general patterns of ingredients. This can lead to a situation where the training loss keeps going down, but the validation loss starts to go up..

Early stopping is like a safety measure that prevents the model from training for too long. It monitors the validation loss, and if it sees it going up for a certain number of epochs (patience), it stops the training process. This way, we avoid the model from overfitting and ensure it can generalize its knowledge to unseen ingredients.
By analyzing the loss graph, we can basically see how well the model is learning and generalizing its knowledge. It helps us decide how long to train the model to achieve the best possible performance without overfitting.

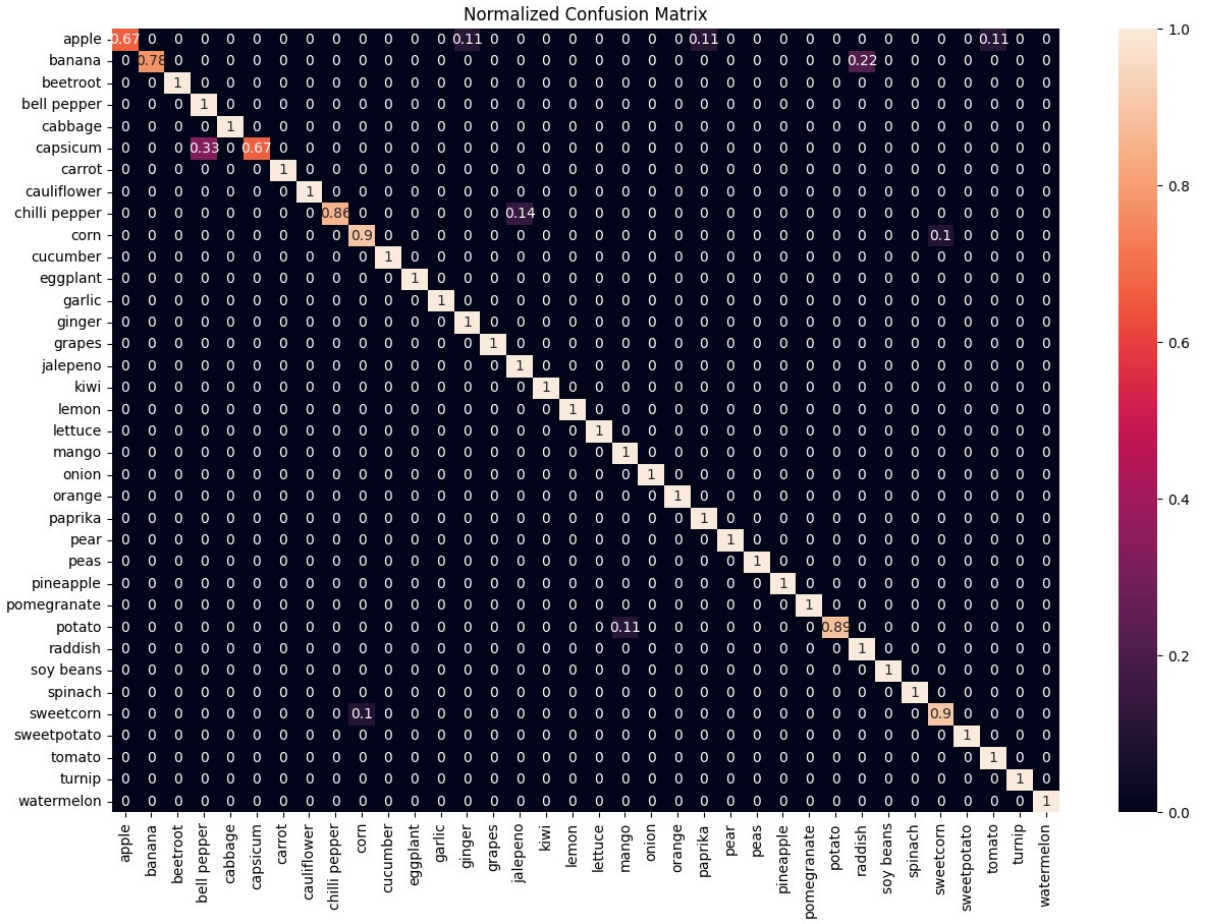




Confusion Matrix:

The confusion matrix helps visualise these by showing the proportion of true positive predictions for each class. In simpler terms, it tells you what percentage of the time the model correctly identified a specific ingredient. Off-diagonal elements in the table represent these misclassifications. For example, if there's a high number in the cell where the "apple" row meets the "banana" column, it means the model frequently mistook apples for bananas.

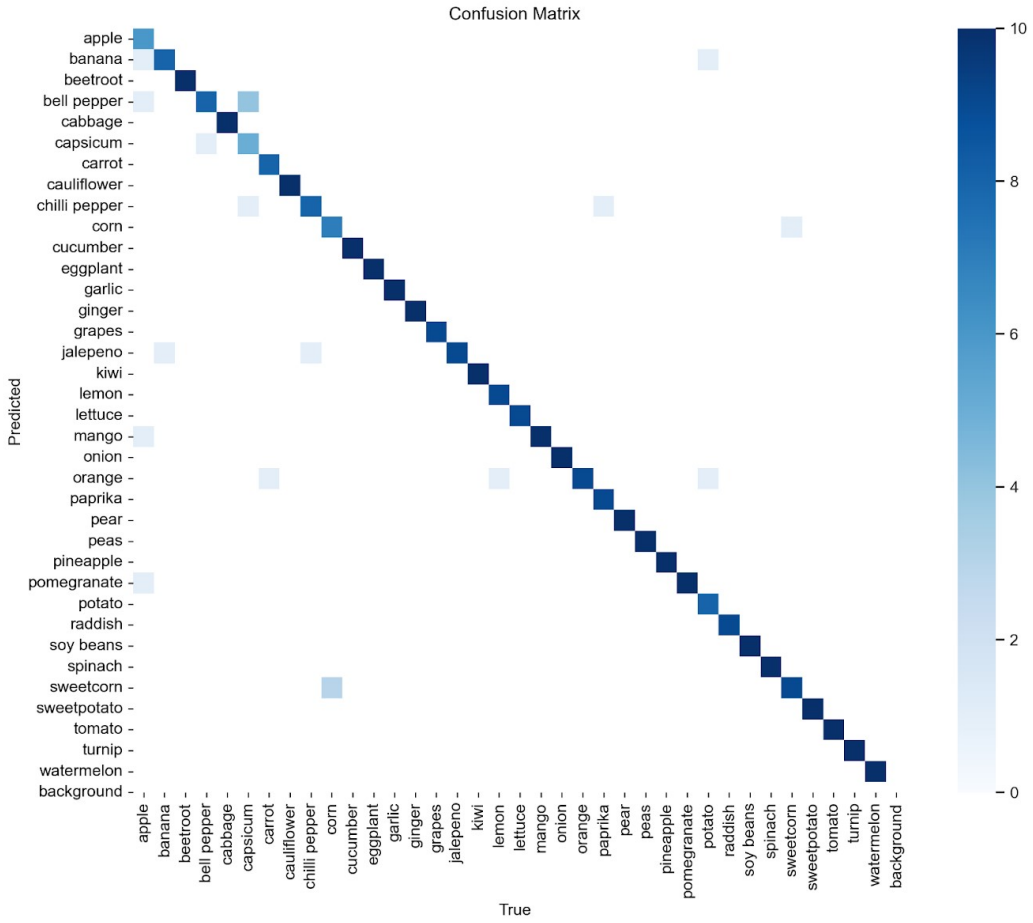
The beauty of a normalised confusion matrix is that it uses proportions instead of raw numbers. This makes it easier to compare the performance between classes, especially when dealing with imbalanced datasets (where some ingredients might have more images than others). By using a colour scale, where green typically represents high accuracy (correct predictions) and red represents low accuracy (misclassifications), the confusion matrix gives you a quick visual understanding of the model's strengths and weaknesses across all ingredient classes. It's like a colourful heatmap that helps you identify areas for improvement in the model's recognition abilities.



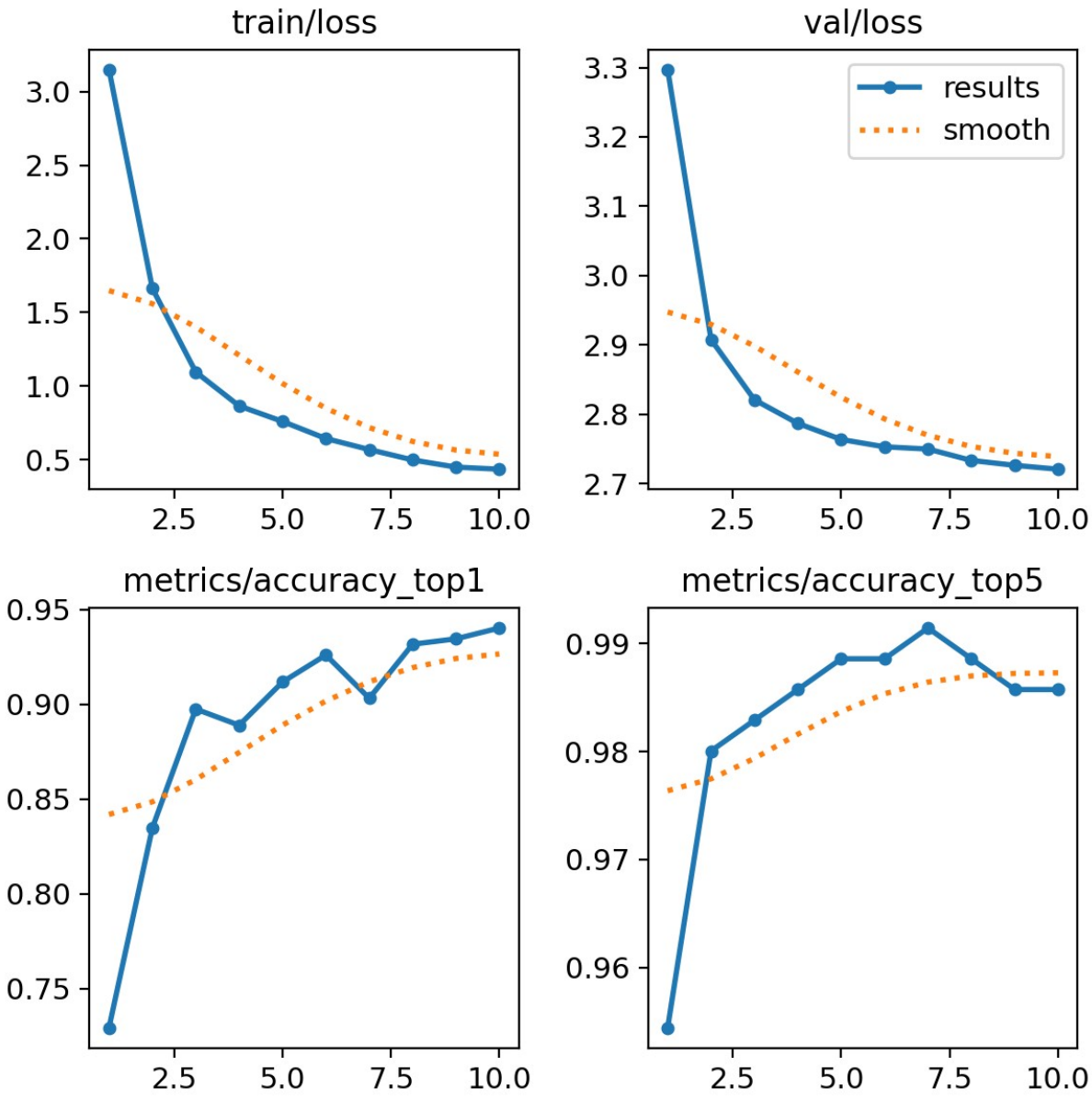
4.4.4 YOLO V8 MODEL:

Here are the results which are provided by the YOLO Model itself after training the model on our dataset. The results include 4 graphs showing relations between accuracy , loss and other metrics and also the Confusion matrix , which is a tabular summary that provides a breakdown of the model's performance in terms of correctly and incorrectly classified instances.

Confusion matrix:


Confusion Matrix
Predicted
True
apple
banana
beetroot
bell pepper
cabbage
capsicum
carrot
cauliflower
chilli pepper
corn
cucumber
eggplant
garlic
ginger
grapes
jalepeno
kiwi
lemon
lettuce
mango
onion
orange
paprika
pear
peas
pineapple
pomegranate
potato
raddish
soy beans
spinach
sweetcorn
sweetpotato
tomato
turnip
watermelon
background
10
8
6
4
2
0

Results:


train/loss
val/loss
results
smooth
metrics/accuracy_top1
metrics/accuracy_top5

4.5 Mobile Application Frontend

Recipify is a mobile application developed using the Flutter framework. It empowers users to upload images of the ingredients they have on hand and receive personalized recipe recommendations based on those ingredients. The app leverages state-of-the-art image classification and large language model (LLM) technologies to provide a seamless and intelligent recipe generation experience.

Flutter Development:
The Recipify app is built entirely using the Flutter framework, a cross-platform development solution that allows for the creation of native-like mobile applications for both iOS and Android platforms. The Flutter development process involves designing the user interface, implementing the core functionality, and integrating various plugins and packages to enhance the app's capabilities.

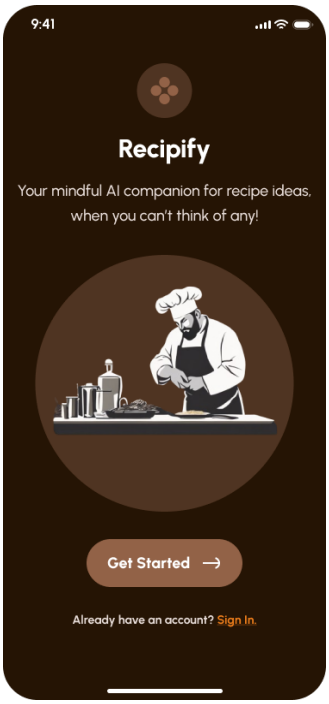
Image Uploading and Ingredient Detection:
At the heart of Recipify is the ability to upload images of ingredients. Users can either capture a new image or select one from their device's photo library. The app then integrates with the image detection model, powered by the Python-based API, to analyze the uploaded image and identify the ingredients present.

Image Classification Model deployment:
The models prepared were deployed as an API service on a docker container through a python based service called Tensorflow-serving. The images are converted into a 3D array and as JSON to the API. The API returns the label of the identified ingredients.

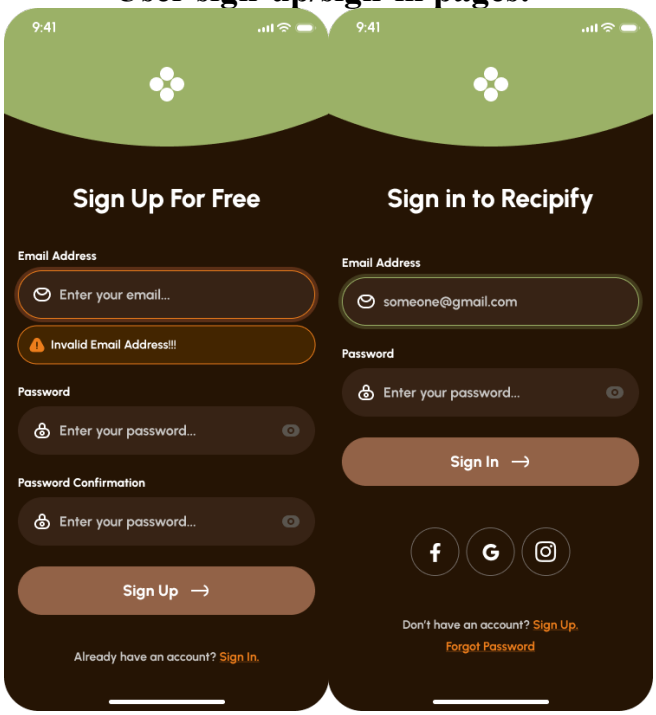
LLM Integration for Recipe Generation:
Once the ingredients have been detected, Recipify sends the information as a prompt to a large language model (LLM) API, such as Gemini or OpenAI. The LLM processes the ingredient list and generates a set of personalized recipe recommendations, taking into account factors like dietary preferences, cuisine, and cooking time. The app then presents these recipes to the user in an intuitive and visually appealing manner.

Screen shots from our application:

Landing page:

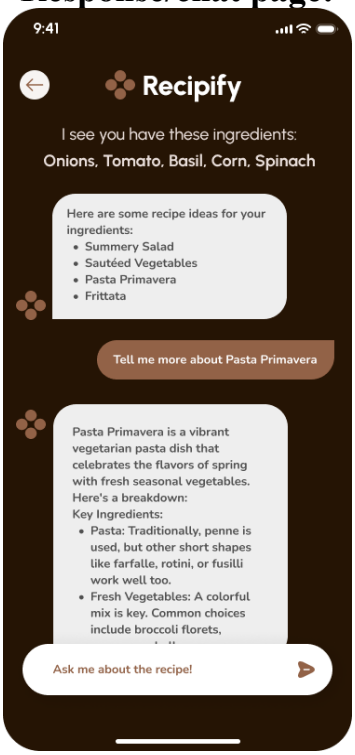


User sign-up/sign-in pages:



Upload images page:

Response/chat page:



4.6 Quality Assurance:

The code includes the following quality assurance measures:

- Using pre-trained models (VGG16, Xception, ResNet50) that have been extensively tested and validated on various datasets, ensuring a robust starting point for the image classification task.
- Splitting the dataset into training, testing, and validation sets, which helps to evaluate the model's performance on unseen data and prevent overfitting.
- Visualizing sample images and predictions to manually verify the data quality and model's performance.
- Utilizing well-established evaluation metrics, such as confusion matrices, to quantify the model's performance and identify potential areas for improvement.
- From our testing, we see that when an image of an object which is not a part of our dataset is given as an input, all the models return incorrect classification with an extremely low accuracy. When such a low accuracy value is detected, it automatically gives an error in response.

Overall, the code follows a structured approach for image classification, including data loading, preprocessing, model training, evaluation, and result visualization. The provided sections cover the essential aspects of the implementation, testing, result analysis, and quality assurance for the project.

Chapter 5

Standard Adopted:

5.1 Design Standards:

Though not stated officially, the code adheres to a few implicit design guidelines and concepts. Among them are:

- **Modular Design:** To encourage modularity and the division of responsibilities, the code is arranged logically into parts and functions such preprocessing, data loading, model training, and evaluation.
- **Object-Oriented Programming (OOP):** The code makes use of classes and methods from other libraries, including TensorFlow and Scikit-learn, which adhere to OOP design principles, even though it does not precisely follow them.
- **Reusability:** To encourage code reuse and maintainability, the predict_image_class function is made to be reusable for many models and image pathways.

5.2 Coding Standards:

Although certain coding standards are not stated explicitly, the code complies with various best practices and coding standards. Among them are:

- **Naming conventions:** To make the code easier to read and comprehend, variables, functions, and other identifiers are all named consistently.
- **Code formatting:** To improve readability, the code is consistently formatted with the appropriate amount of indentation and spacing.
- **Commenting:** The code has a few brief comments that help with code comprehension and maintainability. These comments describe the purpose of specific sections or functions.
- **Use of Docstrings**: The predict_image_class function has a docstring that explains the goals, inputs, and anticipated outcomes of the function.
- **Respect for Python Style rules:** The code appears to adhere to PEP 8 and other common Python style rules, which support consistency and readability in programming.

5.3 Testing Standards:

While no other testing frameworks were utilized, we used certain testing methods and procedures, like:

- **Data splitting:** To assess model performance on unknown data and avoid overfitting, the dataset is divided into training, testing, and validation sets. This technique is widely used in machine learning.
- **Visual Inspection:** Sample images, true labels, and predicted labels are visualized in the code to facilitate manual inspection and validation of the model's output.
- **Evaluation Metrics:** Confusion matrices are a common evaluation metric used in the code to evaluate the effectiveness of classification models.

Chapter 6

6.1 CONCLUSION:

This project explored the potential of deep learning for recipe recommendation based on food ingredient images. We successfully built a system that utilizes Convolutional Neural Network (CNN) models for ingredient identification. We tested the performance of four different CNN models (VGG16, ResNet50, Xception, and YOLOv8) to determine the most effective one.

The system then retrieves recipes containing the identified ingredients through a Large Language Model API such as ChatGPT or Gemini. The recipes are presented to the user through the mobile application. This project presents a valuable tool for both novice and experienced cooks, offering recipe inspiration and simplifying meal planning based on available ingredients. Overall, it contributes to a more enjoyable and informed culinary experience.

6.2 FUTURE SCOPE

- **Expanding Ingredient Recognition:** The current system focuses on fruits and vegetables. Future iterations could include a significantly larger dataset encompassing a wider variety of ingredients like meat, seafood, spices, and pantry staples. This would enable more comprehensive recipe recommendations.
- **Dietary Restrictions and Preferences:** The system could be refined to incorporate user-specific dietary restrictions and preferences. This could include filtering recipes based on allergies, following specific diets (vegan, gluten-free, etc.), or incorporating taste preferences (spicy, sweet, etc.).
- **Multi-Ingredient Image Recognition:** The current system analyzes single ingredients. Future development could enable recognizing multiple ingredients within a single image, allowing recipe recommendations based on a user's entire available ingredients.
- **Smart Kitchen Integration:** Imagine a system seamlessly integrated with smart kitchen appliances. By recognizing ingredients, the system could suggest recipes, adjust cooking parameters for specific ingredients, and even generate grocery lists based on missing ingredients.
- **Food Waste Reduction:** The system could recommend recipes that utilize ingredients users already have, potentially reducing food waste and promoting resourcefulness in the kitchen.

REFERENCES:

[1] https://www.ijraset.com/research-paper/recipe-recommendation-by-ingredients-detection
[2] https://www.kaggle.com/datasets/kritikseth/fruit-and-vegetable-image-recognition/data
[3] https://www.kaggle.com/code/jimohola/dl-fruits-and-vegetable-classification#notebook-container
[4] https://www.kaggle.com/code/shoroukalaa07/fruits-vegetables-classification-using-cnn
[5] https://github.com/feitgemel/TensorFlowProjects/tree/master/Fruit-and-Vegetable-Image-Recognition
[6] https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/
[7] https://www.analyticsvidhya.com/blog/2022/02/k-fold-cross-validation-technique-and-its-essentials/
[8] https://medium.com/@tanmher09/classifying-fruits-images-using-a-resnet-and-regularization-techniques-in-pytorch-ca82d4b1486a
[9] https://www.kaggle.com/code/ligtfeather/eda-and-cnn-for-image-classification
[10] https://medium.com/geekculture/eda-for-image-classification-dcada9f2567a
[11] https://www.sciencedirect.com/science/article/pii/S1746809420304717
[12] https://www.hindawi.com/journals/sp/2022/4194874/
[13] https://www.degruyter.com/document/doi/10.1515/jisys-2014-0079/html?lang=en 4.
[14] https://www.sciencedirect.com/science/article/pii/S1746809420304717

INDIVIDUAL CONTRIBUTION REPORT:

RECIPE SUGGESTION BASED ON FOOD INGREDIENTS IMAGE CLASSIFICATION

AARYAK PRASAD

2105171

Abstract: This project explored the potential of deep learning for recipe recommendation using food ingredient images. It successfully built a system that identifies ingredients through CNN models and retrieves relevant recipes via a Large Language Model API, presenting a valuable tool for simplifying meal planning and enhancing the culinary experience.

Individual contribution to project report preparation: Added the Mobile Frontend Application section and screenshots from the application. Also, helped with literature review and

Overall format of the project report.

Individual contribution for project presentation and demonstration: Worked on the mobile application frontend development in Flutter. Integrated the frontend with OpenAI API for getting responses from ChatGPT. Also, created a backend service on Flask hosted on Docker for accessing the Image Classification models through an API.

Full Signature of Supervisor: Full signature of the student:

……. ……..

School of Computer Engineering, KIIT, BBSR

INDIVIDUAL CONTRIBUTION REPORT:

RECIPE SUGGESTION BASED ON FOOD INGREDIENTS IMAGE CLASSIFICATION

RAUNAK ROY CHOWDHURY

21051076

Abstract: This project explored the potential of deep learning for recipe recommendation using food ingredient images. It successfully built a system that identifies ingredients through CNN models and retrieves relevant recipes via a Large Language Model API, presenting a valuable tool for simplifying meal planning and enhancing the culinary experience.

Individual contribution to project report preparation: I have contributed significantly to the project report as a member of our team, especially in the areas of data pretreatment and model implementation. VGG16, ResNet50, and Xception were the three models that I mostly studied and used in my work.

Individual contribution for project presentation and demonstration: It was my job to use the Xception and ResNet50 models' capabilities and implement it in the project presentation. I went into detail about their criteria, architecture, and how they helped our project succeed.

Full Signature of Supervisor: Full signature of the student:

……. ……..

School of Computer Engineering, KIIT, BBSR

INDIVIDUAL CONTRIBUTION REPORT:

RECIPE SUGGESTION BASED ON FOOD INGREDIENTS IMAGE CLASSIFICATION

MD. AYAAN KHAN

2105209

Abstract: This project explored the potential of deep learning for recipe recommendation using food ingredient images. It successfully built a system that identifies ingredients through CNN models and retrieves relevant recipes via a Large Language Model API, presenting a valuable tool for simplifying meal planning and enhancing the culinary experience.

Individual contribution to project report preparation: In the project report preparation, I did my part of attaching the Yolo V8 Model Training, Testing and Results after implementing it and studying some research papers on this model.

Individual contribution for project presentation and demonstration: I played a significant role in preparing the project presentation, focusing on explaining the technical aspects of the YOLO v8 Model that I implemented. During the demonstration, I presented the changes I made in the YOLO model for our project.

Full Signature of Supervisor: Full signature of the student:

…….. ……..

School of Computer Engineering, KIIT, BBSR

INDIVIDUAL CONTRIBUTION REPORT:

RECIPE SUGGESTION BASED ON FOOD INGREDIENTS IMAGE CLASSIFICATION

ABHAY SINGH

2105172

Abstract: This project explored the potential of deep learning for recipe recommendation using food ingredient images. It successfully built a system that identifies ingredients through CNN models and retrieves relevant recipes via a Large Language Model API, presenting a valuable tool for simplifying meal planning and enhancing the culinary experience.

Individual contribution to project report preparation: I have contributed in preparing project report sections of Testing, Result Analysis and Standards Adopted. Also contributed in literature review of VGG16 related papers.

Individual contribution for project presentation and demonstration: I played a significant role in preparing the project presentation, focusing on explaining the technical aspects of the VGG16 Model that I implemented. During the demonstration, I presented the changes I made in the VGG16 model for our project.

Full Signature of Supervisor: Full signature of the student:

……. ……..

School of Computer Engineering, KIIT, BBSR

INDIVIDUAL CONTRIBUTION REPORT:

RECIPE SUGGESTION BASED ON FOOD INGREDIENTS IMAGE CLASSIFICATION

SHIVLI SINGH

2105237

Abstract: This project explored the potential of deep learning for recipe recommendation using food ingredient images. It successfully built a system that identifies ingredients through CNN models and retrieves relevant recipes via a Large Language Model API, presenting a valuable tool for simplifying meal planning and enhancing the culinary experience.

Individual contribution to project report preparation: I conducted extensive literature review of the research papers mentioned in the report. Also contributed in preparing project report introduction, conclusion, references and study and implementation of the Resnet50 model.

Individual contribution for project presentation and demonstration: I played a significant role in preparing the project presentation, focusing on explaining the technical aspects of the Resnet50 Model that I implemented. During the demonstration, I presented the changes I made in the Resnet50 model for our project.

Full Signature of Supervisor: Full signature of the student:

……. ……..

School of Computer Engineering, KIIT, BBSR

RECIPE SUGGESTION BASED ON FOOD INGREDIENTS IMAGE CLASSIFICATION

ORIGINALITY REPORT

24%	18%	10%	18%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Vietnam Maritime University Student Paper	5%
2	www.coursehero.com Internet Source	2%
3	Submitted to Liverpool John Moores University Student Paper	2%
4	Submitted to University of Suffolk Student Paper	1%
5	www.researchgate.net Internet Source	1%
6	www.worldleadershipacademy.live Internet Source	1%
7	Submitted to Banaras Hindu University Student Paper	1%
8	Submitted to University of Hull Student Paper	<1%
	Submitted to University of North Texas	

9	Student Paper	<1%
10	Submitted to Toronto Business College Student Paper	<1%
11	Md. Kishor Morol, Md. Shafaat Jamil Rokon, Ishra Binte Hasan, A. M. Saif, Rafid Hussain Khan, Shuvra Smaran Das. "Food Recipe Recommendation Based on Ingredients Detection Using Deep Learning", Proceedings of the 2nd International Conference on Computing Advancements, 2022 Publication	<1%
12	Ahmad Waleed Saleh, Gaurav Gupta, Surbhi B. Khan, Norah A. Alkhaldi, Amit Verma. "An Alzheimer's disease classification model using transfer learning densenet with embedded healthcare decision support system", Decision Analytics Journal, 2023 Publication	<1%
13	Submitted to Intercollege Student Paper	<1%
14	jovian.ai Internet Source	<1%
15	medium.com Internet Source	<1%
16	Submitted to Coventry University Student Paper	

		<1%
17	huggingface.co Internet Source	<1%
18	Submitted to KIIT University Student Paper	<1%
19	Submitted to The University of the West of Scotland Student Paper	<1%
20	Merieme Mansouri, Samia Benabdellah Chaouni, Said Jai Andaloussi, Ouail Ouchetto. "Deep Learning for Food Image Recognition and Nutrition Analysis Towards Chronic Diseases Monitoring: A Systematic Review", SN Computer Science, 2023 Publication	<1%
21	Mohammad Javad izadi, Pourya Hassani, Mehrdad Raeesi, Pouria Ahmadi. "A novel WaveNet-GRU model for PEM fuel cells degradation prediction based on transfer learning", Energy, 2024 Publication	<1%
22	Submitted to University of Technology, Sydney Student Paper	<1%
23	github.com Internet Source	<1%

24	David Paper. "TensorFlow 2.x in the Colaboratory Cloud", Springer Science and Business Media LLC, 2021 Publication	<1 %
25	Submitted to University of Edinburgh Student Paper	<1 %
26	www.spiedigitallibrary.org Internet Source	<1 %
27	"Reverse Engineering of Regulatory Networks", Springer Science and Business Media LLC, 2024 Publication	<1 %
28	Krishna Mridha, Apu Chandra Barman, Shekhar Biswas, Shakil Sarkar, Sunanda Biswas, Masrur Ahsan Priyok. "Accuracy and Interpretability: Developing a Computer-Aided Diagnosis System for Pneumonia Detection in Chest X-Ray Images", 2023 International Conference on Distributed Computing and Electrical Circuits and Electronics (ICDCECE), 2023 Publication	<1 %
29	Submitted to University of Bristol Student Paper	<1 %
30	"Intelligent Systems and Networks", Springer Science and Business Media LLC, 2023 Publication	<1 %

31	Mohammad “Sufian” Badar, Hussam Bin Mehare, Jishnu Pillai Anilkumar, Khairol Amali bin Ahmad, Mohammad Rehan Badar. "Chapter 10 Case Study 2: Brain Tumor Classification", Springer Science and Business Media LLC, 2023 Publication	<1 %
32	www.mdpi.com Internet Source	<1 %
33	www.jsr.org Internet Source	<1 %
34	Submitted to Deakin University Student Paper	<1 %
35	Submitted to University of Salford Student Paper	<1 %
36	figshare.com Internet Source	<1 %
37	"Computational Intelligence in Pattern Recognition", Springer Science and Business Media LLC, 2022 Publication	<1 %
38	Submitted to University of Northumbria at Newcastle Student Paper	<1 %
39	dspace.library.uvic.ca Internet Source	<1 %

40	pdxscholar.library.pdx.edu Internet Source	<1 %
41	www.ijcse.com Internet Source	<1 %
42	Submitted to George Bush High School Student Paper	<1 %
43	Submitted to University of Dundee Student Paper	<1 %
44	Submitted to University of Sydney Student Paper	<1 %
45	Submitted to Queen Mary and Westfield College Student Paper	<1 %
46	Submitted to University of New South Wales Student Paper	<1 %
47	Submitted to unibuc Student Paper	<1 %
48	Submitted to University of Essex Student Paper	<1 %
49	hal.science Internet Source	<1 %
50	mypark.tistory.com Internet Source	<1 %
51	newartscollege.ac.in	

	Internet Source	<1%
52	usermanual.wiki Internet Source	<1%
53	Saman Forouzandeh, Mehrdad Rostami, Kamal Berahmand, Razieh Sheikhpour. "Health-aware food recommendation system with dual attention in heterogeneous graphs", Computers in Biology and Medicine, 2024 Publication	<1%
54	ia803202.us.archive.org Internet Source	<1%
55	machinelearningmastery.com Internet Source	<1%
56	"Innovations in Smart Cities Applications Volume 5", Springer Science and Business Media LLC, 2022 Publication	<1%
57	Submitted to Loughborough University Student Paper	<1%
58	Xuexia Zhang, Xueqing Guo. "Fault diagnosis of proton exchange membrane fuel cell system of tram based on information fusion and deep learning", International Journal of Hydrogen Energy, 2021 Publication	<1%

59	assets.researchsquare.com Internet Source	<1 %
60	ijarcce.com Internet Source	<1 %
61	kupdf.net Internet Source	<1 %
62	www.ijert.org Internet Source	<1 %
63	M. Suchetha, Jaya Sai Kotamsetti, Dasapalli Sasidhar Reddy, S. Preethi, D. Edwin Dhas. "chapter 1 AI-Driven Plant Leaf Disease Detection for Modern Agriculture", IGI Global, 2023 Publication	<1 %
64	Mfundo Monchwe, Ibidun C. Obagbuwa, Alfred Mwanza. "Chapter 8 Coronavirus Lung Image Classification with Uncertainty Estimation Using Bayesian Convolutional Neural Networks", Springer Science and Business Media LLC, 2023 Publication	<1 %
65	Jonah Gamba. "Chapter 3 Building Deep Learning Models", Springer Science and Business Media LLC, 2024 Publication	<1 %

66	Orhan Gazi Yalçın. "Applied Neural Networks with TensorFlow 2", Springer Science and Business Media LLC, 2021 Publication	<1%
67	Poornachandra Sarang. "Artificial Neural Networks with TensorFlow 2", Springer Science and Business Media LLC, 2021 Publication	<1%

Exclude quotes	Off	Exclude matches	Off
Exclude bibliography	Off		