National College of Ireland

Configuration Manual

MSc Research Project
Data Analytics

Ketaki Dabekar
Student ID: x22149619

School of Computing
National College of Ireland

Supervisor: Professor Anant Aloka

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Ketaki Suhas Dabekar.

Student ID: X22149619

Programme: Data Analytics **Year:** 2023-24

Module: Research Project

Lecturer: Professor Anant Aloka

Submission Due Date: 31/01/2024

Project Title: Vegan Ingredient Image Classification using Deep Learning and Transfer Learning.

Word Count: 1165 **Page Count:** 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Ketaki Dabekar.

Date: 31/01/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Ketaki Dabekar
Student ID: 22149619

1 Introduction

The configuration manual covers a variety of topics related to the implementation process. This includes the hardware specifications for the system on which the implementation was performed, as well as the software required. It also discusses the various stages of implementation and the overall evaluation of the research "Vegan Ingredient Image Classification using Deep Learning and Transfer Learning".

In below section we will discuss the implementation steps which are carried out during project implementation. And which computational resources used in the project development phase all details we discussed here:

2 System Configuration

Here we will discuss which hardware and software we needed to run code. Let's start with hardware requirements:

2.1 Hardware Requirements:

- Processor 12th Gen Intel(R) Core (TM) i5-1235U 1.30 GHz
- RAM 16.00 GB
- System Type 64-bit Windows Operating System
- GPU Intel(R) Iris(R) Xe Graphics
- Storage 512 GB SSD

2.2 Software Requirements:

- Anaconda navigator for Windows 2.5.1
    - It includes many design frameworks such as Jupyter Notebook, Spyder, R Studio, and others.
- Python –
    - In the whole project I utilized Python as a programming language and 3.8 version I used.
- Jupyter Notebook: We can download Jupyter from anaconda navigator.It is open-source platform. The Jupyter notebook built into the operating system was primarily used project implementation. Jupyter Notebook was used for most of the implementation, from exploratory data analysis to image classification model implementation. The following Fig.1.shows the libraries included in the project:

Libraries

```
In [490]: import os
import random
from PIL import Image
import matplotlib.pyplot as plt
from tabulate import tabulate
import tensorflow as tf
import random
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.resnet50 import ResNet50, preprocess_input
from tensorflow.keras import layers, models, optimizers
from tensorflow.keras.callbacks import ReduceLROnPlateau
from tensorflow.keras.applications import EfficientNetB0
import numpy as np
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
from tensorflow.keras.preprocessing import image
from sklearn.metrics import classification_report, confusion_matrix
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
import json
import zipfile
```

Fig.1. Libraries used in the project.

- Once you download jupyter notebook from anaconda navigator. It will open new tab in browser I have attached image below Fig.2. Also using same tab, you need upload code. So, it will be ready to run.

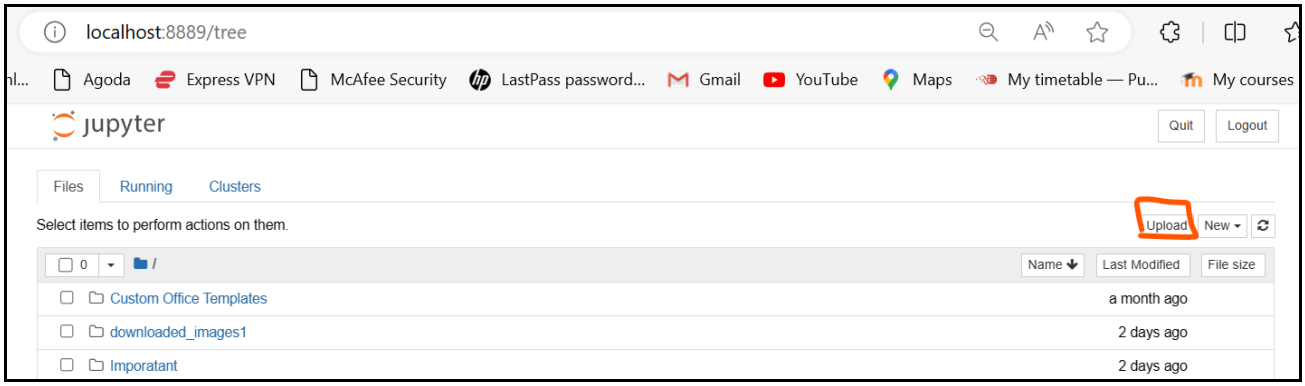


Fig.2.Jupyter software.

- Below code snippets shows that I have taken dataset from Kaggle website. I downloaded zip file of the dataset through Kaggle API. Generally, API we will get under account setting -> API section -> Click on 'Create New API Token.' It will generate Json file. Using that just create environment. Then run Kaggle command which will download dataset zip folder. Fig.3. shows the implementation. Once I finish with downloading. I extracted dataset from zip file Fig.4. demonstrates that. Then unzip dataset and used further in project.

Load data from kaggle folders

```
# Load Kaggle API key
with open('C:/Users/ketak/kaggle.json') as f:
    kaggle_api_key = json.load(f)

# Set Kaggle API key environment variables
os.environ['KAGGLE_USERNAME'] = kaggle_api_key['username']
os.environ['KAGGLE_KEY'] = kaggle_api_key['key']

# Run Kaggle command to download the dataset
!kaggle datasets download -d ketakidabekar/vegan-ingredients -p extracted_data
```

Fig.3. Download dataset from kaggle

```
# Specify the path to the downloaded zip file
zip_file_path = 'extracted_data/vegan-ingredients.zip'

# Specify the path where you want to extract the contents
extract_path = 'extracted_data'

# Extract the dataset
with zipfile.ZipFile(zip_file_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

# Remove the downloaded zip file if needed
os.remove(zip_file_path)

# dataset path
dataset_path = 'extracted_data/Vegan-Food Dataset - Copy'
```

Fig.4.Extract dataset from zip folder

3 Dataset Description

The vegan ingredients dataset was constructed using one dataset which took from Kaggle website. It is an open-source public dataset that can be used for research. The dataset contains 95 food item categories, each of which is labelled with at least 60 images. 5577 images collected from both datasets . Fig.6. shows random images from dataset within grid. Fig.5. Shows total 95 number of classification names which I printed while implementing code.

Print class names within dataset

```
# print the class names
class_folders = os.listdir(dataset_path)
class_names = [class_folder for class_folder in class_folders if os.path.isdir(os.path.join(dataset_path, class_folder))]

print(class_names)
```

```
['Almonds', 'Apples', 'Avocado', 'Bananas', 'Barley', 'Basil', 'Bay Leaf', 'beetroot', 'Black Beans', 'Black Lentils (Beluga Le
ntils)', 'Black-Eyed Peas', 'Brazil Nuts', 'broccoli', 'Brown Lentils', 'Brussel sprouts', 'Buckwheat', 'cabbage', 'capsicum',
'Carrots', 'Cashews', 'cauliflower', 'Cherries', 'ChiaSeeds', 'Chickpeas', 'Cinnamon', 'Clove', 'Coriander', 'corn', 'courgette
s', 'cucumber', 'cummin seeds', 'dried seaweeds', 'Drum Stick', 'Edamame', 'eggplant', 'Flaxseeds', 'garlic', 'Ginger', 'grapef
ruit', 'Grapes', 'green beans', 'Green Lentils', 'Hazelnuts', 'jackfruit', 'Kale', 'Kidney Beans', 'Kiwi', 'leeks', 'lemon', 'L
emongrass', 'Lima Beans', 'Macadamia Nuts', 'Mangoes', 'Mung Beans', 'Muster Seeds', 'Nutmeg', 'Oats', 'onions', 'Oranges', 'Or
egano', 'Parsley', 'parsnips', 'Peaches', 'Peanuts', 'Pears', 'peas', 'Pecans', 'Pine Nuts', 'Pineapples', 'Pinto Beans', 'Pist
achios', 'Plums', 'Pomegranate', 'potato', 'Pumpkin seeds', 'Quinoa', 'raddish', 'Red Lentils', 'Rice', 'Rosemary', 'Sage', 'Se
same seeds', 'soya', 'spinach', 'strawberries', 'Sunflower seed', 'sweetpotato', 'Tempeh', 'Thyme', 'Tofu', 'tomato', 'Turmeri
c', 'Walnuts', 'Watermelon', 'Yellow Lentils']
```

Fig.5. Total classification classes.

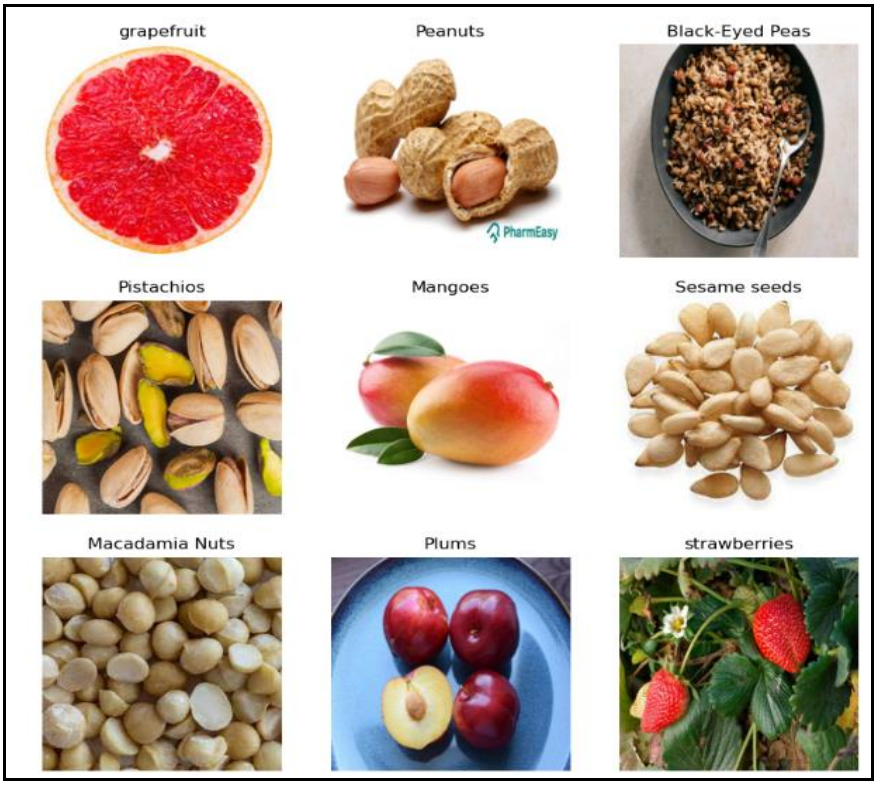


Fig.6. Dataset

4 Data Modelling for Vegan Ingredients Classification

This section describes how the data was prepared before the model was applied. Also, this is step by step execution of cells and the sequence is same which I am mentioning. As I am using 3 transfer learning model the preparation for building those models is also described below:

4.1 Pre-processing dataset:

For pre-processing dataset preprocess_input function used. The Keras library provide this function. This function used with transfer learning model so that's why I choose this function. It will perform normalization, scaling, mean subtraction functionality and many more. Fig.7. shows the implementation of preprocess_input.

```
# Data preprocessing using ImageDataGenerator with augmentation
datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
```

Fig.7. preprocess_input

4.2 Data augmentation:

For data augmentation used ImageDataGenerator. ImageDataGenerator this is substitute of data augmentation. All operation performed by data augmentation the similar operation this functionally also performed on dataset. Data preprocessing and augmentation goes hand in hand here I have applied both things together. In Fig.8. we can see that and operations which I am using for augmentation is also mentioned in below figure.

```python
# Data preprocessing using ImageDataGenerator with augmentation
datagen = ImageDataGenerator(
    preprocessing_function=preprocess_input,
    validation_split=0.2,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    vertical_flip=True,
)
```

Fig.8. Data augmentation

4.3 Splitting Data:

After performing preprocessing and augmentation on dataset. Next step which I performed was split data into two sections. Training and Validation. Below in Fig.9. we can see I have divided 4496 images in training dataset and 1084 images in validation dataset.

```python
# Create data generators for training and validation
train_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=True,
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    dataset_path,
    target_size=img_size,
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False,
    subset='validation'
)
```

```
Found 4494 images belonging to 95 classes.
Found 1084 images belonging to 95 classes.
```

Fig.9. Splitting augmented data

4.4 EfficientNet B0:

This is the first model which I applied on augmented dataset. Fig.10. shows EfficientNet B0 model implementation which I performed in notebook. From image we can see I have applied Sequential model then adds GlobalAveragePooling2D and Dense layer into model. After designing I compiled model and fit model on divided dataset.

```python
# Build the EfficientNetB0 model
model_efficientnet = models.Sequential()
model_efficientnet.add(EfficientNetB0(input_shape=(224, 224, 3), include_top=False, weights='imagenet'))
model_efficientnet.add(layers.GlobalAveragePooling2D())
model_efficientnet.add(layers.Dense(95, activation='softmax'))  # Assuming 95 classes

# Compile the model
model_efficientnet.compile(optimizer=optimizers.Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=['accuracy'])

# Reduce learning rate if the validation loss plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-7)

# Train the EfficientNetB0 model with data augmentation and learning rate scheduling
history_efficientnet = model_efficientnet.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10,  # Adjust the number of epochs as needed
    callbacks=[reduce_lr]
)
```

```
Epoch 1/10
141/141 [==============================] - ETA: 0s - loss: 4.1840 - accuracy: 0.1195
```

Fig.10. EfficientNet B0

4.5 ResNet 50:

This is the second model which I applied on augmented dataset. Fig.11. shows ResNt 50 model implementation. From below image we can see that I have followed similar architecture as EfficientNet B0. After designing I compiled model and fit model on divided dataset with 20 as epoch value. 20 is last epoch value I considered after performing fine tuning on model.

```
# Build the ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

model = models.Sequential()
model.add(base_model)
model.add(layers.GlobalAveragePooling2D())
model.add(layers.Dense(95, activation='softmax'))

# Compile the model with a lower learning rate
model.compile(optimizer=optimizers.Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Reduce learning rate if the validation loss plateaus
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=2, min_lr=1e-7)

# Train the model with data augmentation and learning rate scheduling
history = model.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=20,  # Increase the number of epochs
    callbacks=[reduce_lr]
)
```

```
Epoch 1/20
141/141 [==============================] - ETA: 0s - loss: 2.6498 - accuracy: 0.4097
```

Fig.11. ResNet 50

4.6 Inception V3:

This is the third model which I applied on augmented dataset. Fig.9. shows Inception V3 model implementation. From below image we can see that here also I have followed similar architecture as EfficientNet B0. After designing I compiled model and fit model on divided dataset with 12 as epoch value.

```
# Build the InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

model_inception = models.Sequential()
model_inception.add(base_model)
model_inception.add(layers.GlobalAveragePooling2D())
model_inception.add(layers.Dense(95, activation='softmax'))

# Compile the model with a lower learning rate
model_inception.compile(optimizer=optimizers.Adam(learning_rate=1e-4), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the EfficientNetB0 model with data augmentation and learning rate scheduling
history_inception = model_inception.fit(
    train_generator,
    validation_data=validation_generator,
    epochs=10,  # Adjust the number of epochs as needed
    callbacks=[reduce_lr]
)
```

```
Epoch 1/10
141/141 [==============================] - ETA: 0s - loss: 3.2814 - accuracy: 0.3013
```

Fig.12. Inception V3

4.7 Inception V3 on Original Dataset:

Before applying Inception V3 model on Original Dataset I performed pre-processing. Fig.13. shows implementation of preprocessing. In below image we can see that what are things which I have used for image preprocessing. After that convert classification class names into integer values because before this it was in string format. For that I have used label encoder. Fig.14. shows the implementation of label encoder which performed on all label within dataset.

```
def preprocess_image(image_path, label):
    img = tf.io.read_file(image_path)
    img = tf.image.decode_jpeg(img, channels=3)
    img = tf.image.resize(img, img_size)
    img = preprocess_input(img)
    return img, label
```

Fig.13. Preprocessing

```
# Use LabelEncoder to convert string labels to integers
label_encoder = LabelEncoder()
all_labels_encoded = label_encoder.fit_transform(all_labels)

# One-hot encode labels
all_labels_encoded = to_categorical(all_labels_encoded, num_classes=95)
```

Fig.14. Label encoder

Then I divide dataset into train and test. Fig.15. shows splitting code. After that performed Inception V3 model on original dataset. I used previous code of Inception V3 here just change the parameters of compile and fit.

```
# Create datasets
train_dataset = tf.data.Dataset.from_tensor_slices((train_paths, train_labels))
validation_dataset = tf.data.Dataset.from_tensor_slices((validation_paths, validation_labels))

# Load and preprocess images
train_dataset = train_dataset.map(preprocess_image, num_parallel_calls=tf.data.experimental.AUTOTUNE)
validation_dataset = validation_dataset.map(preprocess_image, num_parallel_calls=tf.data.experimental.AUTOTUNE)

# Batch and shuffle the datasets
train_dataset = train_dataset.shuffle(buffer_size=len(train_paths)).batch(batch_size).prefetch(tf.data.experimental.AUTOTUNE)
validation_dataset = validation_dataset.batch(batch_size)
```

Fig.15. Splitting dataset into train and validation.

5 Model Evaluation

Below We will discuss evaluation of model performed on which things:

5.1 Accuracy:

Below Fig.16 shows the code implementation which I used to showcase validation accuracy. This code I have used in all models to showcase validation accuracy.

```
# Evaluate the model on the test set
eval_result_efficientnet = model_efficientnet.evaluate(validation_generator)
print("EfficientNetB0 Test accuracy:", eval_result_efficientnet[1])
```

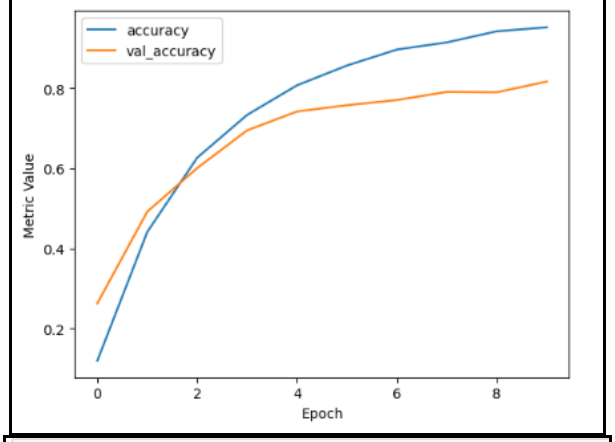
```
34/34 [==============================] - 44s 1s/step - loss: 0.7880 - accuracy: 0.8081
EfficientNetB0 Test accuracy: 0.8081181049346924
```

Fig.16. Accuracy implementation.

5.2 AUC and loss graph:

Below Fig.17 shows the code implementation of AUC and loss graph to showcase both graphs for train and validation dataset. Similar code I have used in all models to showcase accuracy and loss graph.

```
# Plot training history
plt.plot(history_efficientnet.history['accuracy'], label='accuracy')
plt.plot(history_efficientnet.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Metric Value')
plt.legend()
plt.show()
```




```
# Plot training history
plt.plot(history_efficientnet.history['loss'], label='loss')
plt.plot(history_efficientnet.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Metric Value')
plt.legend()
plt.show()
```

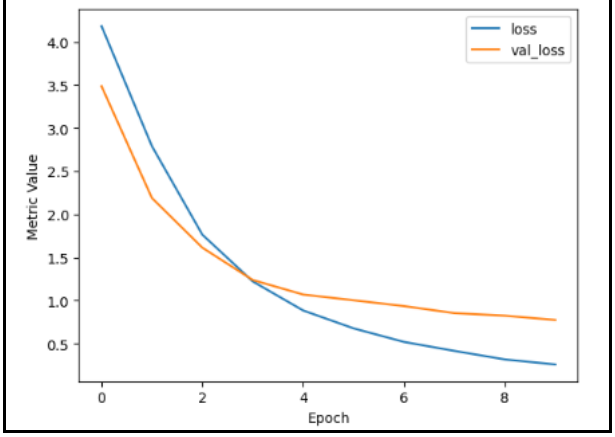


Fig.17. Graph of AUC and loss

5.3 Precision, Recall and F1 - Score:

Below Fig.18.shows the implementation as well as evaluation of precision, recall and F1-score. Which I have used in all models for evaluation purpose.

```
In [170]: # Calculate precision and F1 score manually
precision = np.diag(conf_mat) / np.sum(conf_mat, axis=0)
recall = np.diag(conf_mat) / np.sum(conf_mat, axis=1)
f1_score = 2 * (precision * recall) / (precision + recall)

print("Precision per class:")
print(precision)

Precision per class:
[1.         0.875      0.875      0.8125     0.73333333 0.92307692
 1.         0.86666667 0.63157895 0.88888889 0.84615385 0.72222222
 0.92307692 0.5625     0.83333333 0.68421053 0.58823529 0.83333333
 0.83333333 0.92307692 0.9        1.         0.84210526 0.81818182
 1.         1.         0.90909091 0.8        0.66666667 0.81818182
 0.88888889 1.         0.66666667 0.9        0.8        0.66666667
 0.58333333 0.66666667 1.         1.         0.83333333 1.
 0.73333333 0.75       0.7        0.76190476 0.85714286 0.77777778
 1.         0.75       0.85714286 0.81818182 0.90909091 0.76923077
 1.         1.         0.46666667 0.64285714 0.9        0.6
 0.7        0.61538462 0.58333333 0.66666667 0.8        0.81818182
 0.77777778 0.68421053 0.875      1.         0.90909091 0.92307692
 1.         1.         0.75       0.88888889 0.76923077 0.90909091
 0.85714286 1.         0.75       1.         0.8        0.85714286
 0.8        0.84615385 0.69230769 0.8        0.58823529 0.875
 0.54545455 1.         0.71428571 1.         0.92307692]

In [171]: print("Recall per class:")
print(recall)

Recall per class:
[0.72727273 0.63636364 1.         0.92857143 1.         1.
 0.90909091 1.         1.         1.         1.         0.92857143
 1.         1.         0.625      0.92857143 0.90909091 0.5
 0.41666667 1.         0.9        0.5        1.         0.9
 0.44444444 0.90909091 0.83333333 0.44444444 0.88888889 1.
 0.8        1.         0.88888889 0.69230769 0.44444444 0.72727273
 0.77777778 0.6        0.76923077 0.54545455 0.90909091 0.22222222
 0.84615385 0.33333333 0.7        1.         0.92307692 0.77777778
 0.45454545 1.         0.66666667 0.69230769 0.66666667 0.71428571
 0.57142857 0.69230769 0.63636364 1.         1.         0.46153846
 0.7        0.66666667 0.7        0.66666667 0.88888889 0.81818182
 0.63636364 1.         0.93333333 1.         1.         0.85714286
 1.         1.         0.9        0.8        1.         1.
 0.66666667 0.88888889 1.         0.7        0.88888889 0.6
 0.8        0.91666667 0.9        1.         0.71428571 0.7
 0.5        0.875      1.         0.2        0.85714286]

In [172]: print("F1 Score per class:")
print(f1_score)

F1 Score per class:
[0.84210526 0.73684211 0.93333333 0.86666667 0.84615385 0.96
 0.95238095 0.92857143 0.77419355 0.94117647 0.91666667 0.8125
 0.96       0.72       0.71428571 0.78787879 0.71428571 0.625
 0.55555556 0.96       0.9        0.66666667 0.91428571 0.85714286
 0.61538462 0.95238095 0.86956522 0.57142857 0.76190476 0.9
 0.84210526 1.         0.76190476 0.7826087  0.57142857 0.69565217
 0.66666667 0.63157895 0.86956522 0.70588235 0.86956522 0.36363636
 0.78571429 0.46153846 0.7        0.86486486 0.88888889 0.77777778
 0.625      0.85714286 0.75       0.75       0.76923077 0.74074074
 0.72727273 0.81818182 0.53846154 0.7826087  0.94736842 0.52173913
 0.7        0.64       0.63636364 0.66666667 0.84210526 0.81818182
 0.7        0.8125     0.90322581 1.         0.95238095 0.88888889
 1.         1.         0.81818182 0.84210526 0.86956522 0.95238095
 0.75       0.94117647 0.85714286 0.82352941 0.84210526 0.70588235
 0.8        0.88       0.7826087  0.88888889 0.64516129 0.77777778
 0.52173913 0.93333333 0.83333333 0.33333333 0.88888889]
```

Fig.18. Precision,Recall and F1 – Score Implementation

5.4 Image Predication:

Below Fig.19. shows the implementation as well as evaluation of image predication code. Which are important evaluation criteria telling predication about image whether it's comes under vegan or not. This implementation as well I have used in all models for evaluation purpose.

Image Predication

```
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt

# Load and preprocess the image
img_path = 'download.jpg'
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.

predictions = model_inception1.predict(img_array)

# Threshold for classification
threshold = 0.05

is_vegan = predictions[0][0] > threshold

# Display the image
plt.imshow(img)
plt.axis('off')

# Print the result
if is_vegan:
    plt.title("The image is classified as vegan.")
else:
    plt.title("The image is not classified as vegan.")

# Show the image with the result
plt.show()
```

```
1/1 [==============================] - 0s 132ms/step
```


```
from tensorflow.keras.preprocessing import image
import numpy as np
import matplotlib.pyplot as plt

# Load and preprocess the image
img_path = '100.jpg'  # Replace with the path to your image
img = image.load_img(img_path, target_size=(224, 224))
img_array = image.img_to_array(img)
img_array = np.expand_dims(img_array, axis=0)
img_array /= 255.

predictions = model_inception1.predict(img_array)

# Threshold for classification
threshold = 0.05

is_vegan = predictions[0][0] > threshold

# Display the image
plt.imshow(img)
plt.axis('off')

# Print the result
if is_vegan:
    plt.title("The image is classified as vegan.")
else:
    plt.title("The image is not classified as vegan.")

# Show the image with the result
plt.show()
```

```
1/1 [==============================] - 0s 189ms/step
```

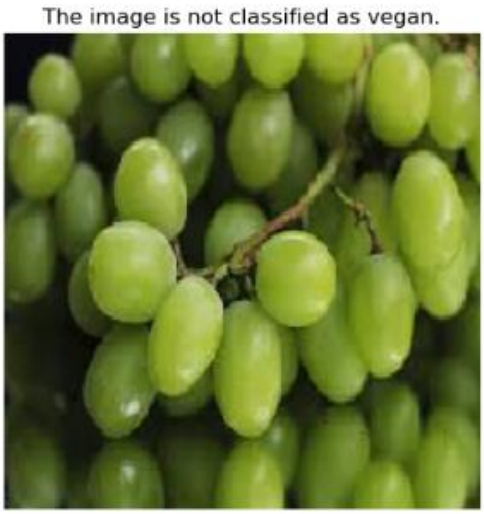


Fig.19. Image predication implementation with evaluation.