AI & SUSTAINABILITY IN VET EDUCATION

Name of the Author/s: Sucessos Criativos, Lda

Title of WP: WP3 – AI-SAVES@ProjectHub

Activity Reference: 3.2 – Project template for AI-SAVES@ProjectHub

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Table of Contents

1. Project Title: 4

2. Project Category: 4

3. Project Description: 4

4. Keywords: 4

5. Target Audience: 5

6. Project Objectives: 5

7. Project Timeline: 5

8. Resources Provided: 5

9. Tools and Technologies Used: 6

10. Step-by-Step Implementation Plan: 6

Step 1: MOUNT GOOGLE DRIVE 7

Step 2: DATA COLLECTION 8

Step 3: CHECKING AND HANDLING IMBALANCED DATA 8

Step 4: DATA AUGMENTATION TO BALANCE CLASSES 11

11. Evaluation Criteria: 37

12. Languages Available: 37

13. Contributors: 37

14. Feedback and Continuous Improvement: 38

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

1. Project Title:

Image Classification with MobileNet and Transfer Learning

---

2. Project Category:

AI in Education

---

3. Project Description:

In this micro-project, learners will develop a custom image classification model capable of distinguishing between three flower types: **roses**, **sunflowers**, and **tulips**. The project uses **MobileNetV3**, a lightweight deep learning model well-suited for edge devices, and uses **transfer learning** to fine-tune the model using a small, curated dataset.

This project is ideal for learners interested in computer vision, image processing, and model training using pre-trained architectures. It demonstrates best practices for preparing datasets, handling class imbalance, and structuring data for training with TensorFlow and Keras.

---

4. Keywords:

Transfer Learning, MobileNetV3, Image Classification, Imbalanced Dataset, ImageDataGenerator, TensorFlow / Keras

---

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

5. Target Audience:

Educators, learners, and makers with or without prior experience who are beginning their journey in AI.

---

6. Project Objectives:

After completing this project, learners will be able to:

- Understand how to organize and preprocess datasets for image classification.
- Handle imbalanced datasets using data augmentation techniques.
- Use TensorFlow's `ImageDataGenerator` for loading, augmenting, and batching images.
- Apply **transfer learning** using MobileNetV3 to build a custom classifier.
- Train, validate, and test a deep learning model on custom flower datasets.
- Evaluate the model's performance using accuracy, precision, recall, and confusion matrices.
- Make predictions on new images and interpret model outputs.

---

7. Project Timeline:

The estimated duration is approximately **2.5 to 3 hours**.

---

8. Resources Provided:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- For these tasks you can use either a computer or your raspberry pi.

---

9. Tools and Technologies Used:

Google Colab: Cloud-based development environment

TensorFlow / Keras: Deep learning framework

Matplotlib, NumPy: Visualization and numerical operations

Kaggle: Source of labeled flower image dataset

---

10. Step-by-Step Implementation Plan:

BEFORE YOU START

This project will guide you through training an image classification model to recognize three types of flowers: **roses**, **sunflowers**, and **tulips**. You'll use **Google Colab**, **TensorFlow**, and **MobileNetV3 with transfer learning**.

Requirements

- A Google Account
- Internet connection
- Access to Kaggle
- Google Colab (no installation needed)

Folder Setup

1. Go to Kaggle Flowers Dataset
   Download the dataset and unzip it.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

2. Keep only the following folders from the dataset:
    - `rose/`
    - `sunflower/`
    - `tulip/`
3. In your **Google Drive**, create the folder structure:

```
/My Drive/
└── Training/
    ├── datasets/
    │   └── flowers/
    │       ├── rose/
    │       ├── sunflower/
    │       └── tulip/
    ├── models/
    └── SupportImages/
```

4. Upload your three flower folders to `/datasets/flowers/`.
5. Create an empty `/models/` folder (to save the model).
6. Open this notebook on Colab and save it in `/Training/`.
7. Activate Table of Contents in Colab:
   View > Table of contents

Step 1: MOUNT GOOGLE DRIVE

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Mount your Google Drive to Colab so you can read and save files:

```
from google.colab import drive
drive.mount('/content/drive')
```

Step 2: DATA COLLECTION

We're building a model to classify **roses**, **sunflowers**, and **tulips**. Images are organized in folders like this:

```
/Training/datasets/flowers/
├── rose/
├── sunflower/
└── tulip/
```

Each folder must contain images only for that class.

Step 3: CHECKING AND HANDLING IMBALANCED DATA

The next step is to check for balanced data as part of the data preparation and preprocessing stage. This check should ideally occur after loading and organizing the dataset but before training begins, meaning this is the moment to do it.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Verifying data balance is essential because an imbalanced dataset — where some classes have significantly more images than others — can lead to a biased model that favors the overrepresented classes.

A balanced dataset ensures that the model has equal opportunity to learn the features of each class, improving its ability to classify each category accurately.

If we find that the dataset is imbalanced, we can take corrective actions, such as augmenting images in underrepresented classes or applying class weights during training, to address this issue before the model starts learning.

```
import os

from collections import Counter

import matplotlib.pyplot as plt



# Path to the main dataset directory

data_dir                    =                    '/content/drive/My
Drive/TrainingGreeceCV/datasets/flowers/'



# Count images in each subdirectory (i.e., each class)

class_counts = Counter()

for class_name in os.listdir(data_dir):

    class_path = os.path.join(data_dir, class_name)

    if os.path.isdir(class_path):

        class_counts[class_name] = len(os.listdir(class_path))

# Print the number of images in each class
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
for class_name, count in class_counts.items():

    print(f"Class '{class_name}': {count} images")



# Plotting the distribution of images per class

plt.figure(figsize=(8, 6))

plt.bar(class_counts.keys(),                    class_counts.values(),
color='skyblue')

plt.xlabel('Class')

plt.ylabel('Number of Images')

plt.title('Class Distribution in Dataset')

plt.show()
```

```
Class 'sunflowers': 699 images
Class 'roses': 641 images
Class 'tulips': 799 images
```



Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Step 4: DATA AUGMENTATION TO BALANCE CLASSES

The dataset we are using is somewhat imbalanced, meaning that certain flower classes have more images than others, as showed on the graph.

An imbalance can affect the model's learning, making it more likely to favor the classes with more images, which can lead to biased predictions.

One approach to handling this imbalance would be to delete images from the overrepresented classes to make the dataset sizes equal across classes. However, removing images would reduce the amount of information the model can learn from, which would be a loss in terms of data quality and diversity.

Instead of deleting images, we will use data augmentation to balance the dataset. Data augmentation is a technique that generates new images by applying transformations to existing images. This allows us to create more samples of the underrepresented classes without gathering new images.

Common transformations used in data augmentation include:

Flipping (horizontal or vertical)

Rotation

Zooming

Color Adjustments

To perform data augmentation on the dataset, we will use TensorFlow's ImageDataGenerator to generate and save augmented images for the underrepresented classes. This code will apply transformations like rotations, flips, and zooms to create more images, which will help balance the dataset.

```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
import os



# Path to the main dataset directory and target number of images
per class

data_dir = '/content/drive/My
Drive/TrainingGreeceCV/datasets/flowers/'

target_num_images = 799  # Desired images per class







# Define data augmentation transformations

datagen = ImageDataGenerator(      # Initialize an
ImageDataGenerator instance for augmenting images

    rotation_range=40,             # Randomly rotate images within
a 40-degree range to create varied angles

    width_shift_range=0.2,         # Randomly shift images
horizontally by up to 20% of the width

    height_shift_range=0.2,        # Randomly shift images
vertically by up to 20% of the height

    shear_range=0.2,               # Apply random shearing
transformations with a 20% range for variety

    zoom_range=0.2,                # Randomly zoom in on images by
up to 20% to create different scales

    horizontal_flip=True,          # Randomly flip images
horizontally to create mirror images

    fill_mode='nearest'            # Fill any newly created pixels
after transformations with nearest pixel values

)
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# Loop through each class folder
for class_name in os.listdir(data_dir):
    class_path = os.path.join(data_dir, class_name)

    # Check if this is a folder and count current images
    if os.path.isdir(class_path):
        current_num_images = len(os.listdir(class_path))
        images_needed = target_num_images - current_num_images

        # Only augment if more images are needed
        if images_needed > 0:
            # Generate and save augmented images in the same class
folder
            generator = datagen.flow_from_directory(
                data_dir,
                classes=[class_name],
                target_size=(224, 224),
                batch_size=1,
                save_to_dir=class_path,  # Save augmented images in
the same folder as original images
                save_prefix='augmented',
                save_format='jpeg'
            )
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
            # Generate the exact number of images needed

            for _ in range(images_needed):

                next(generator)
```

Now, we can execute the same code as before to apply data augmentation and ensure that the classes are balanced.

```
import os

from collections import Counter

import matplotlib.pyplot as plt



# Path to the main dataset directory

data_dir = '/content/drive/My
Drive/TrainingGreeceCV/datasets/flowers/'



# Count images in each subdirectory (i.e., each class)

class_counts = Counter()

for class_name in os.listdir(data_dir):

    class_path = os.path.join(data_dir, class_name)

    if os.path.isdir(class_path):

        class_counts[class_name] = len(os.listdir(class_path))



# Plotting the distribution of images per class

plt.figure(figsize=(8, 6))

plt.bar(class_counts.keys(), class_counts.values(),
color='skyblue')

plt.xlabel('Class')
```

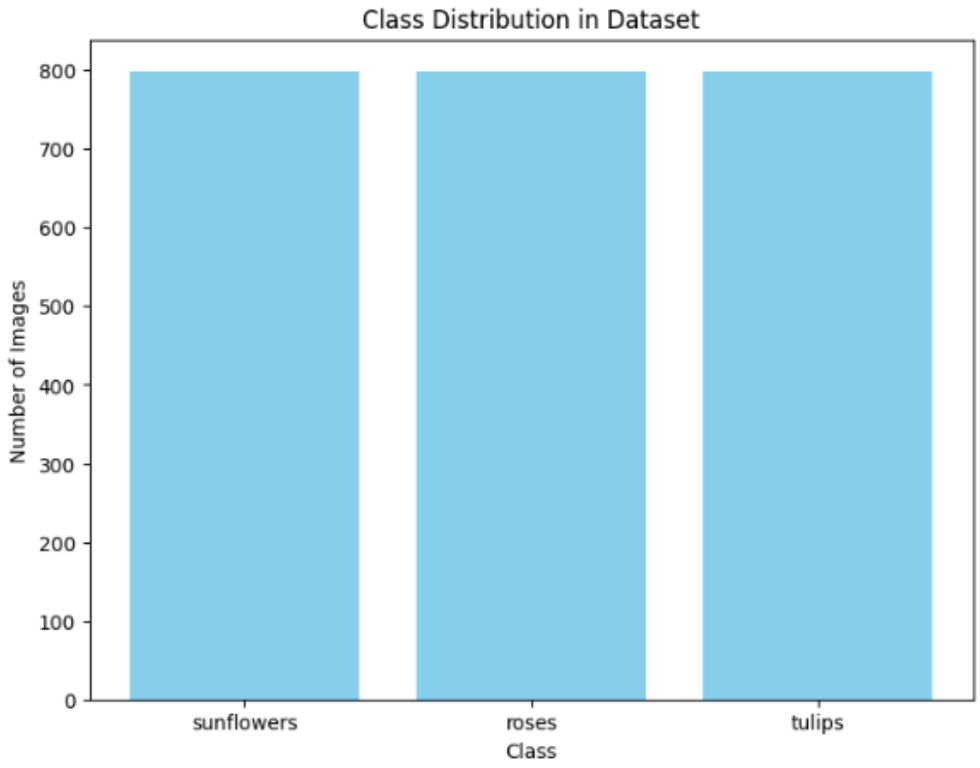
Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
plt.ylabel('Number of Images')

plt.title('Class Distribution in Dataset')

plt.show()
```



Step 5: ORGANIZING THE DATASET FOR TRAINING

Organizing the Dataset for Training means arranging data in a structured way to align with the requirements of the machine learning model. Different models require different dataset structures due to their different architectures and training approaches.

For this project, we are using MobileNetV3. MobileNet's structure is relatively simple compared to other models and it requires a straightforward dataset structure.

MobileNet expects images to be organized by class, where each class has its own folder containing all its images.

Example of the structure:

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
/dataset/
    ├── train/                  # 70% of data
    │   ├── class_1/            # Class 1 images for training
    │   │   ├── image1.jpg
    │   │   ├── image2.jpg
    │   │   └── ...
    │   ├── class_2/            # Class 2 images for training
    │   │   ├── image1.jpg
    │   │   ├── image2.jpg
    │   │   └── ...
    │   └── ...
    ├── validation/             # 15% of data
    │   ├── class_1/            # Class 1 images for validation
    │   │   ├── image1.jpg
    │   │   ├── image2.jpg
    │   │   └── ...
    │   ├── class_2/            # Class 2 images for validation
    │   │   ├── image1.jpg
    │   │   ├── image2.jpg
    │   │   └── ...
    │   └── ...
    ├── test/                   # 15% of data
    │   ├── class_1/            # Class 1 images for testing
    │   │   ├── image1.jpg
    │   │   ├── image2.jpg
    │   │   └── ...
    │   ├── class_2/            # Class 2 images for testing
    │   │   ├── image1.jpg
    │   │   ├── image2.jpg
    │   │   └── ...
    │   └── ...
```

In MobileNet, folder names are used to automatically label images, with each folder representing a specific class. The model associates all images in a folder with that class label, eliminating the need for separate label files.

Splitting a dataset into training, validation and test sets helps the model learn patterns from one part of the data, fine-tune its parameters, and evaluate its performance on unseen data.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Training Set: This portion, typically 70% of the data, is used by the model to learn patterns, features, and relationships in each class, providing it with enough examples to build a robust understanding.

Validation Set: Around 15% of the data is used for validation, allowing the model to adjust its parameters and avoid overfitting on the training data. The validation set helps to tune the model without influencing the final performance measurement.

Test Set: The remaining 15% is kept for testing. This data is not seen by the model during training or validation, allowing an unbiased evaluation of how well the model performs on new data. A 15% test set typically provides a reliable estimate of the model's real-world accuracy.

This process is very important because:

Preventing Overfitting If the model trains and tests on the same data, it may "memorize" the dataset instead of learning patterns that apply to new data. Splitting the data helps the model learn in a way that prepares it for new examples.

Reliable Evaluation The test set provides a way to check the model's accuracy and other metrics on data it hasn't seen before. This helps ensure the model's performance is realistic and not just good on familiar data.

Better Generalization The model trains on a variety of examples but is tested on separate data. This setup helps the model learn patterns that work well on new data rather than just the training examples.

```
import os

import shutil

import random



# Define the source directory and target directories for train and
test

source_dir = '/content/drive/My Drive/Training/datasets/flowers/'

train_dir                    =                    '/content/drive/My
Drive/Training/datasets/flowersToTrain/train/'

test_dir                     =                    '/content/drive/My
Drive/Training/datasets/flowersToTrain/test/'

val_dir                      =                    '/content/drive/My
Drive/Training/datasets/flowersToTrain/val/'
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# Define split ratio
train_ratio = 0.7  # 70% for training
test_ratio = 0.15   # 15% for testing
validation_ratio = 0.15   # 15% for validation



# Define the split ratios
train_ratio = 0.7
validation_ratio = 0.15
test_ratio = 0.15



# Create target directories if they don't exist
os.makedirs(train_dir, exist_ok=True)
os.makedirs(val_dir, exist_ok=True)
os.makedirs(test_dir, exist_ok=True)


# Iterate through each class folder in the source directory
for class_name in os.listdir(source_dir):
    class_path = os.path.join(source_dir, class_name)
    if os.path.isdir(class_path):
        # Create subdirectories in train, validation, and test
folders for each class
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
        os.makedirs(os.path.join(train_dir,             class_name),
exist_ok=True)

        os.makedirs(os.path.join(val_dir,               class_name),
exist_ok=True)

        os.makedirs(os.path.join(test_dir,              class_name),
exist_ok=True)


        # Get list of all images in the class folder and shuffle them

        images = os.listdir(class_path)

        random.shuffle(images)


        # Calculate split counts

        train_count = int(len(images) * train_ratio)

        validation_count = int(len(images) * validation_ratio)


        # Split images into train, validation, and test sets

        train_images = images[:train_count]

        validation_images    =    images[train_count:train_count    +
validation_count]

        test_images = images[train_count + validation_count:]


        # Move images to the train folder

        for img in train_images:

            src_path = os.path.join(class_path, img)

            dst_path = os.path.join(train_dir, class_name, img)

            shutil.copy(src_path, dst_path)
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
        # Move images to the validation folder
        for img in validation_images:
            src_path = os.path.join(class_path, img)
            dst_path = os.path.join(val_dir, class_name, img)
            shutil.copy(src_path, dst_path)

        # Move images to the test folder
        for img in test_images:
            src_path = os.path.join(class_path, img)
            dst_path = os.path.join(test_dir, class_name, img)
            shutil.copy(src_path, dst_path)

print("Dataset successfully split into train, validation, and test sets.")
```

Check the number of files

```
def count_images(directory):
    total_count = 0
    for class_name in os.listdir(directory):
        class_folder = os.path.join(directory, class_name)
        if os.path.isdir(class_folder):
            num_images = len(os.listdir(class_folder))
            total_count += num_images
            print(f"{class_name}    -    {directory.split('/')[-1]}: {num_images} images")
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
    print(f"Total images in {directory.split('/')[-1]} set: {total_count}\n")


print("\nCounting images in each set:")

count_images(train_dir)

count_images(val_dir)

count_images(test_dir)
```

Step 6: Import the libraries

Before building and training our model, we need to import essential libraries from TensorFlow and Keras. These libraries provide tools for image preprocessing, model building, and optimization:

```
import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV3Small

from tensorflow.keras.layers import Dense, GlobalAveragePooling2D, Dropout

from tensorflow.keras.models import Model

from tensorflow.keras.optimizers import Adam
```

What each import does:

- `tensorflow` – Core deep learning framework used throughout this project.
- `ImageDataGenerator` – Helps load and augment images in batches during training.
- `MobileNetV3Small` – A lightweight, pre-trained model we'll adapt for flower classification.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

- `Dense`, `GlobalAveragePooling2D`, `Dropout` – Layers used to build the custom classifier.
- `Model` – Allows us to define a model that combines the MobileNet base with our new layers.
- `Adam` – An adaptive optimizer used to train the network efficiently.

Step 7:Set Up Data Generators

Setting up data generators is essential for training machine learning models for several reasons.

First, data generators manage efficient data loading and memory by handling images in batches, which prevents memory overflow. Instead of loading all images at once, data generators load only a few images at a time, allowing training on large datasets without using too much memory.

Second, data generators enable real-time data augmentation, which helps improve model robustness, expanding the dataset through transformations like rotations, shifts, zooms, and flips. This means the model sees slightly different versions of each image during each training epoch, which helps reduce overfitting and improves the model's ability to handle variations in new data.

Third, data generators automatically assign labels and perform basic preprocessing, such as resizing and normalizing images, based on the folder structure.

```
# Define directories for training and validation datasets

train_dir                    =                    '/content/drive/My
Drive/Training/datasets/flowersToTrain/train/'

val_dir                      =                    '/content/drive/My
Drive/Training/datasets/flowersToTrain/val/'
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# Set up ImageDataGenerators for training and testing with data augmentation

train_datagen = ImageDataGenerator(rotation_range=15)

train_generator = train_datagen.flow_from_directory(train_dir, target_size=(224, 224), batch_size=32)




val_datagen = ImageDataGenerator(rotation_range=15)

val_generator = val_datagen.flow_from_directory(val_dir, target_size=(224, 224), batch_size=32)
```

The number of images created in this stage through data augmentation depends on the number of epochs and batches used during training, rather than a fixed count.

Data augmentation generates new variations of images in each batch, producing an "infinite" supply of training data without actually increasing the physical dataset size. When using an ImageDataGenerator with augmentation, each batch provides a modified "view" of the original images, with transformations like rotation, zoom, and shifts. This means that the total number of unique image variations the model encounters is determined by the number of epochs and batches.

These augmentations should be used carefully because excessive or inappropriate transformations can distort the images, leading to poor model learning. For example, extreme rotations, zooms, or flips may create unrealistic versions of images that do not represent the actual classes, confusing the model instead of improving it. Additionally, if certain augmentations (like flips) are applied to classes where orientation matters, the model may struggle to distinguish between correct and incorrect orientations, reducing accuracy. Properly balanced augmentations help the model generalize by introducing useful variations, but overuse or misapplication can negatively affect its performance on real-world data.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Step 8: Load Pre-trained MobileNetV3 and Modify for Transfer Learning

We'll load the MobileNetV3 model, remove its final classification layer, and add new layers for our custom classes. Freeze the pre-trained layers to retain learned features and add a custom output layer for our dataset's specific classes.

This following code creates a custom classification model based on MobileNetV3Small, a pre-trained model, which is used as a feature extractor by setting include_top=False to remove its original classification layer.

The base model outputs feature maps, which are then passed through a series of custom layers: a global average pooling layer to reduce dimensionality and three fully connected layers with ReLU activation to help the model learn complex patterns.

The final output layer has 3 units with a softmax activation, enabling multi-class classification for three categories. The model is configured by connecting the base model's input to this new output layer.

All layers except the last five are frozen, so only the new layers are updated during training, which helps retain MobileNet's learned features and reduces the likelihood of overfitting.

```
baseModel = MobileNetV3Small(weights="imagenet", include_top=False, input_shape=(224, 224, 3))



# Add custom layers on top of the base model

x = baseModel.output

x = GlobalAveragePooling2D()(x)                 # Add global average pooling layer

x = Dense(512, activation='relu')(x)           # Add fully connected layer with 512 units

x = Dense(256, activation='relu')(x)           # Add fully connected layer with 256 units

x = Dense(128, activation='relu')(x)           # Add fully connected layer with 128 units
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# Output layer with 3 classes and softmax activation for multi-class classification

predictionLayer = Dense(3, activation='softmax')(x)



# Define the full model by connecting base model input and custom prediction layer

model = Model(inputs=baseModel.input, outputs=predictionLayer)



# Freeze the layers of MobileNetV3 (already trained layers)

for layer in model.layers[:-5]:  # Freeze all layers except the last 5

    layer.trainable = False
```

Step 9:Compile the Model

To prepare the model for training, we need to compile it by specifying an optimizer, a loss function, and evaluation metrics.

Optimizer: The optimizer adjusts the model's weights during training to minimize the loss function. Here, we use the Adam optimizer, a popular choice for deep learning models, as it adapts the learning rate for each parameter individually, making training efficient. For transfer learning, we set a smaller learning rate that allows fine-tuning of the new layers added on top of the pre-trained model without making drastic changes to the pre-trained weights. This approach helps retain the beneficial features learned from the original ImageNet dataset while adapting the model to new data.

Loss Function: The loss function measures how well the model's predictions match the true labels. We use categorical crossentropy, which compares the predicted probability distribution across classes with the true distribution.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

Metrics: Metrics are used to evaluate the model's performance during training and testing. We use accuracy as a primary metric for classification tasks, as it indicates the percentage of correctly classified images.

```
# Compile the model with optimizer, loss function, and metrics

optimizer = Adam(learning_rate=0.0001)

model.compile(loss="categorical_crossentropy", optimizer=optimizer,
metrics=['accuracy'])
```

Step 10 :Train and save the Model

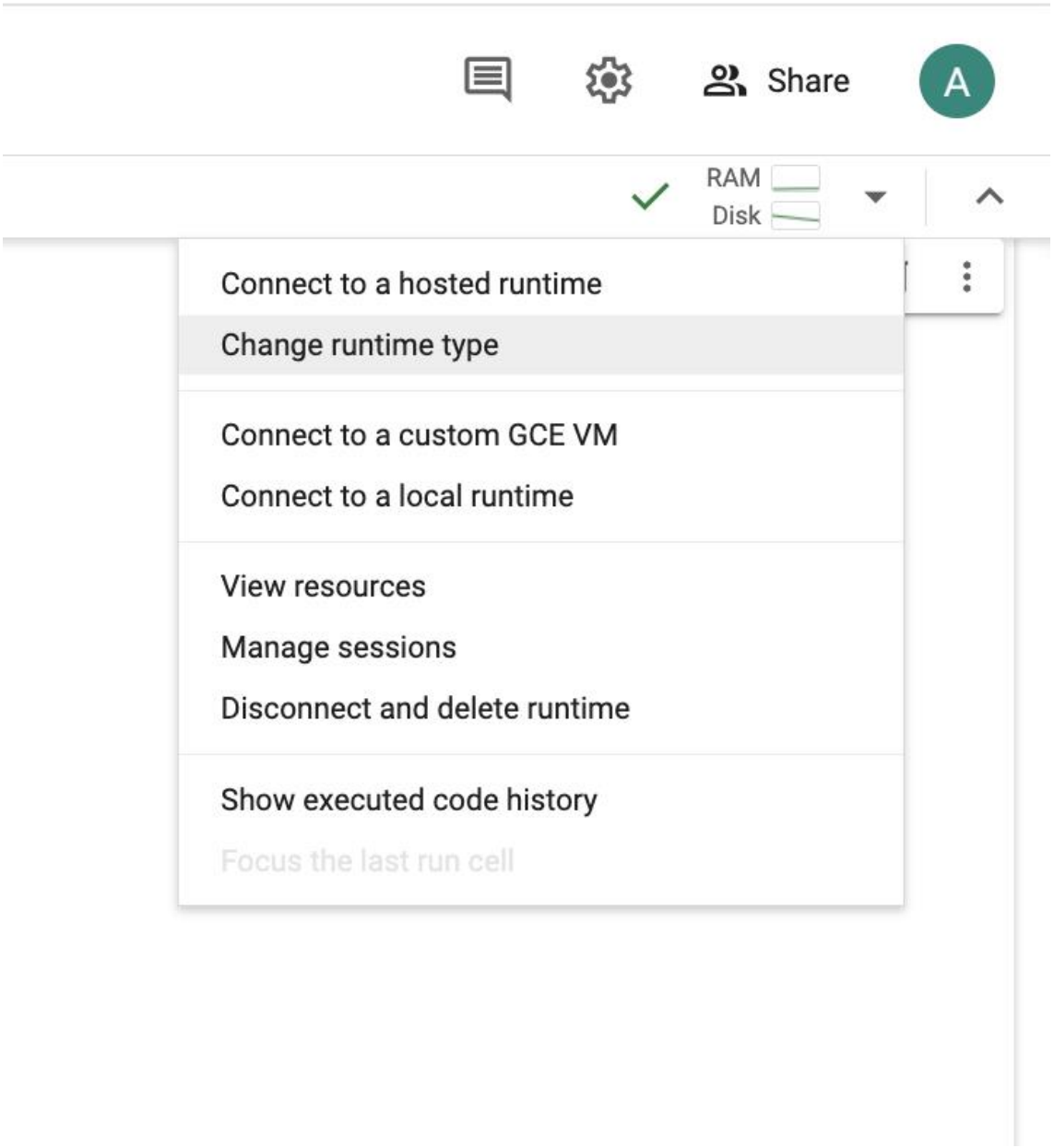
In Colab enviroment make sure that you have access to a T4 GPU (Graphics Processing Unit). The T4 GPU is used for deep learning and model training because it can handle the parallel computations required for tasks like matrix operations and tensor manipulations, which are common in neural networks.

To switch from CPU to GPU in Google Colab, follow these steps:

1- In your Colab notebook, go to the top menu and select Runtime and select Change runtime type.

Co-funded by the European Union

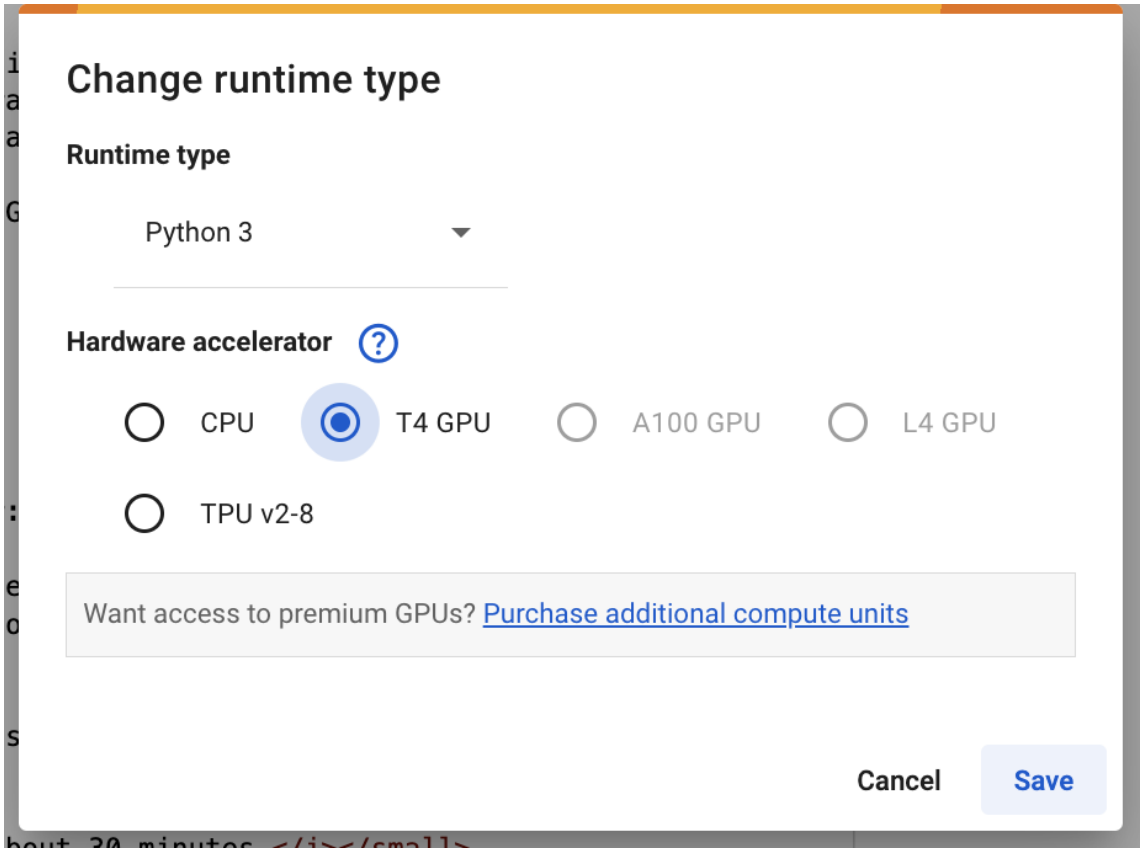
This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



2- In the pop-up window, find the Hardware accelerator dropdown menu. Select GPU from the dropdown options (you may also see None and TPU as other options). Click Save to confirm your selection.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



This task takes about 30 minutes.

```
model.fit(train_generator,            validation_data=val_generator,
epochs=10)


model.save('/content/drive/My
Drive/TrainingGreeceCV/models/flower_classifier_model_AUG_FV2.keras
')
```

Step 11 :Evaluate the model

The following code evaluates our model on the test set by loading the test data, predicting the classes, and calculating various performance metrics.

The test data is loaded with ImageDataGenerator in a non-augmented format, which feeds images to the model without shuffling to ensure consistent results.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

The model evaluation step provides loss and accuracy metrics, where loss reflects the prediction error, and accuracy shows the proportion of correct predictions over the total samples.

- **The confusion matrix** visually displays the model's performance across each class, showing how often the model correctly or incorrectly predicted each class, which helps identify patterns of misclassification.
- **Precision** measures how many of the predicted positive results are actually positive (useful for reducing false positives)
- **recall** measures how many actual positives were correctly identified (useful for reducing false negatives)
- **F1 score** combines precision and recall to provide a balanced measure, especially helpful for imbalanced datasets.

The classification report gives a per-class summary of precision, recall, and F1 score, providing deeper insight into model performance for each class.

- **ROC curve** (Receiver Operating Characteristic curve) plots the true positive rate against the false positive rate, helping assess the model's ability to distinguish between classes, with the AUC (Area Under the Curve) score indicating overall classification performance—values closer to 1.0 suggest better distinction between classes.

```
import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.metrics import confusion_matrix

from tensorflow.keras.models import load_model

from tensorflow.keras.preprocessing.image import ImageDataGenerator


from sklearn.metrics import precision_score, recall_score, f1_score,
roc_curve, auc, classification_report

import matplotlib.pyplot as plt

import numpy as np



model                 =                load_model('/content/drive/My
Drive/Training/models/flower_classifier_model_AUG_FV2.keras')
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# Set up the ImageDataGenerator for the test set
# Define the test directory
test_dir                    =                    '/content/drive/My
Drive/Training/datasets/flowersToTrain/test/'

# Set up ImageDataGenerator for the test data without augmentation
test_datagen = ImageDataGenerator()  # No data augmentation

# Create the test generator
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='categorical',  # Change to 'binary' if it's a binary
classification task
    shuffle=False  # Don't shuffle test data
)

# Calculate the number of steps for evaluation based on the test
generator
evaluation_steps         =         test_generator.samples         //
test_generator.batch_size

# Evaluate model to confirm loss and accuracy
loss, accuracy = model.evaluate(test_generator)
print(f"Test Loss: {loss:.4f}")
print(f"Test Accuracy: {accuracy:.4f}")

# Generate predictions
predictions = model.predict(test_generator)
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
y_pred = np.argmax(predictions, axis=1)
y_true = test_generator.classes
y_pred_prob = predictions if predictions.shape[1] == 2 else predictions[:, 1]  # For ROC

# Confusion matrix
cm = confusion_matrix(y_true, y_pred)
class_names = list(test_generator.class_indices.keys())

# Plot the confusion matrix
plt.figure(figsize=(10, 8))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=class_names, yticklabels=class_names)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Compute precision, recall, and F1 scores
precision = precision_score(y_true, y_pred, average='weighted')
recall = recall_score(y_true, y_pred, average='weighted')
f1 = f1_score(y_true, y_pred, average='weighted')

# Print classification report
class_names = list(test_generator.class_indices.keys())
report = classification_report(y_true, y_pred, target_names=class_names)
print("Classification Report:\n", report)
print(f"Precision: {precision:.4f}, Recall: {recall:.4f}, F1 Score: {f1:.4f}")
```

Co-funded by the European Union

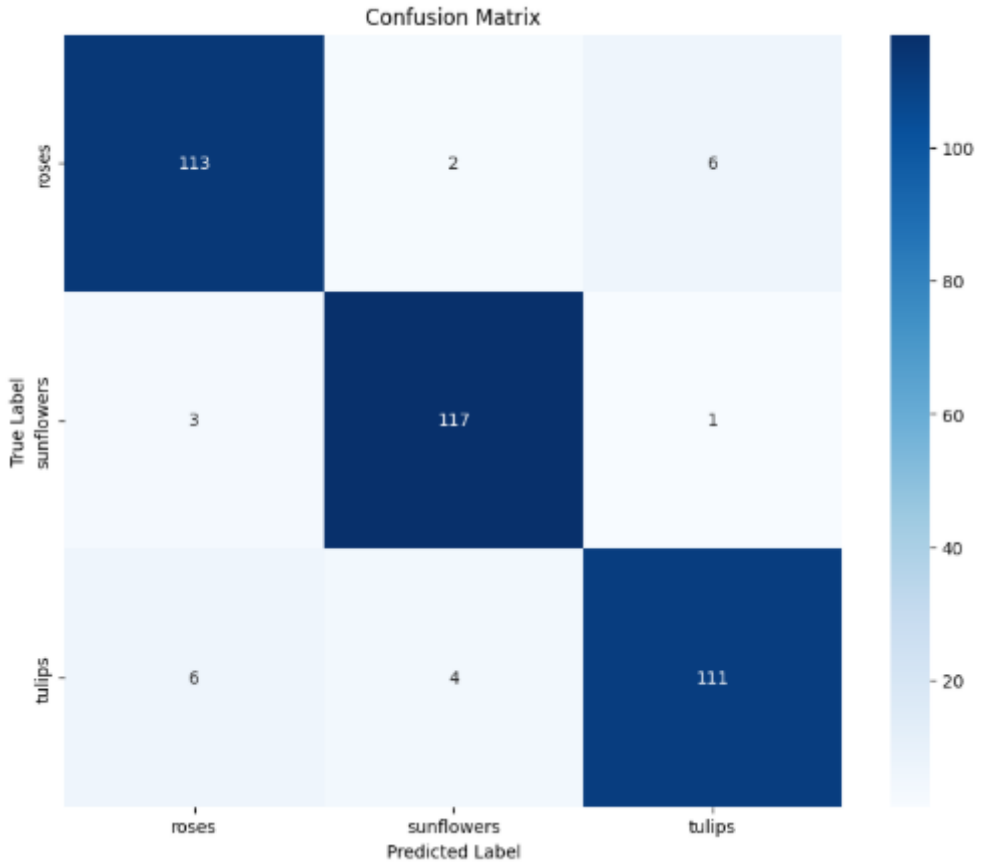
This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# ROC Curve Plotting
plt.figure(figsize=(8, 6))
if len(class_names) == 2:  # Binary classification
    fpr, tpr, _ = roc_curve(y_true, y_pred_prob)
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, color='blue', label=f'ROC curve (area = {roc_auc:.2f})')
else:  # Multiclass classification
    for i in range(len(class_names)):
        fpr, tpr, _ = roc_curve(y_true == i, predictions[:, i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'ROC curve of class {class_names[i]} (area = {roc_auc:.2f})')

plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.



Co-funded by the European Union

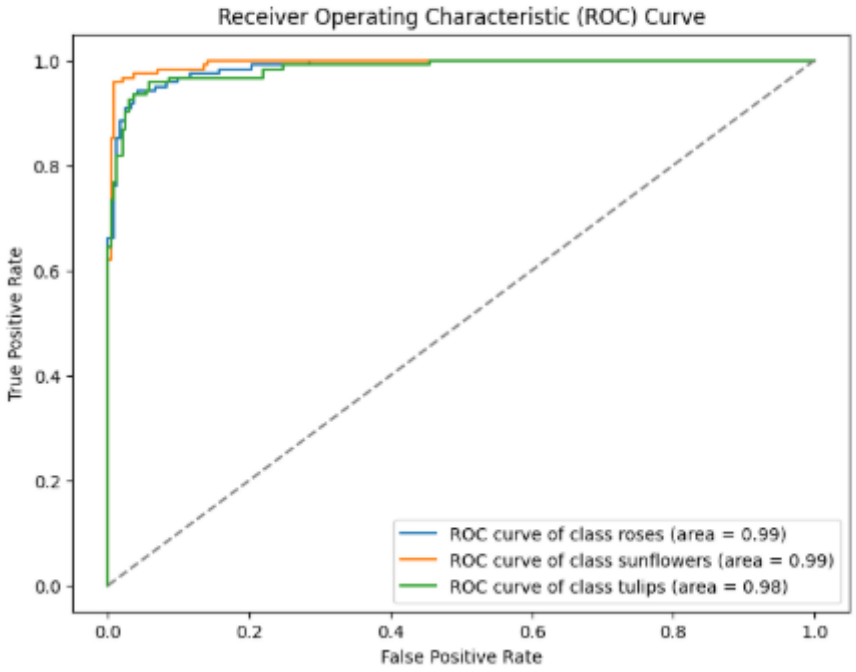
This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
Classification Report:
              precision    recall  f1-score   support

       roses       0.93      0.93      0.93       121
  sunflowers       0.95      0.97      0.96       121
      tulips       0.94      0.92      0.93       121

    accuracy                           0.94       363
   macro avg       0.94      0.94      0.94       363
weighted avg       0.94      0.94      0.94       363

Precision: 0.9394, Recall: 0.9394, F1 Score: 0.9393
```



Step 12 :Predict on new image

After training and saving our flower classification model, we can use it to predict the class of a new, unseen image. This is an important step to verify that the model works correctly outside the training and validation sets.

How to Get the Image

Choose any image of a flower (rose, tulip, or sunflower) from your computer or from an online source like Google Images.

Rename the image (e.g., t.jpeg) and upload it manually to your Google Drive under: /Training/

Make sure it's a real-world photo — this helps assess how well the model generalizes.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

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

import numpy as np



model = load_model('/content/drive/My Drive/Training/models/flower_classifier_model_AUG_FV2.keras')

# Set up ImageDataGenerators for training and testing with data augmentation



train_dir = '/content/drive/My Drive/Training/datasets/flowersToTrain/train/'

train_datagen = ImageDataGenerator()

train_generator = train_datagen.flow_from_directory(train_dir, target_size=(224, 224), batch_size=32)

train_dir = '/content/drive/My Drive/Training/datasets/flowersToTrain/train/'







# Path to the single image

img_path = '/content/drive/My Drive/Training/t.jpeg'



# Load and preprocess the image

img = load_img(img_path, target_size=(224, 224))

img_array = img_to_array(img)

img_array = np.expand_dims(img_array, axis=0)  # Add batch dimension
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
# Predict the class

prediction = model.predict(img_array)

predicted_class = np.argmax(prediction, axis=1)[0]

print(prediction)



# Map index to class label

class_labels = {v: k for k, v in train_generator.class_indices.items()}



# Print predictions for all classes with percentages

print("Predictions for all classes:")

for index, probability in enumerate(prediction[0]):

    label = class_labels[index]

    percentage = probability * 100  # Convert to percentage

    print(f"Class: {label}, Probability: {percentage:.2f}%")



# Print the top predicted class

predicted_class = np.argmax(prediction, axis=1)[0]

predicted_label = class_labels[predicted_class]

predicted_probability = prediction[0][predicted_class] * 100

print(f"\nPredicted class: {predicted_label} with confidence: {predicted_probability:.2f}%")
```

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

```
Found 1677 images belonging to 3 classes.
1/1 ━━━━━━━━━━━━━━━━━━━━ 3s 3s/step
[[7.7244986e-05 1.1240940e-05 9.9991155e-01]]
Predictions for all classes:
Class: roses, Probability: 0.01%
Class: sunflowers, Probability: 0.00%
Class: tulips, Probability: 99.99%

Predicted class: tulips with confidence: 99.99%
```

---

11. Evaluation Criteria:

Dataset Setup : Images collected, labeled, and organized correctly

Data Augmentation: Underrepresented classes augmented using ImageDataGenerator

MobileNet Integration: Model uses MobileNet with transfer learning

Accuracy and Evaluation: Confusion matrix, precision, recall, F1 score reported

Final Prediction: Model makes predictions on new images |

---

12. Languages Available:

English, **Portuguese** (translations in progress)

---

13. Contributors:

Fátima Pais, Luís Soares – Project Designers and Facilitators

---

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

14. Feedback and Continuous Improvement:

- **Surveys**: At the end of this milestone, students and educators will complete feedback surveys to assess the effectiveness of the project.
- **Project Reviews**: After project completion, a final review meeting will be held to discuss potential improvements for future AI-based projects.
- **Data Analysis**: Feedback data will be analyzed to identify strengths and areas for improvement, informing future projects.

Co-funded by the European Union

This project has been funded with support from the European Commission. This communication reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.