```
from google.colab import files
uploaded = files.upload()
```

Choose Files images.zip
**images.zip**(application/x-zip-compressed) - 4822409 bytes, last modified: 11/15/2025 - 100% done
Saving images.zip to images.zip

```
import zipfile

with zipfile.ZipFile("images.zip", 'r') as zip_ref:
    zip_ref.extractall("/content")

!ls /content
```

```
images  images.zip  sample_data
```

```
!ls /content/images
```

```
dalmatian  dollar_bill  pizza  soccer_ball  sunflower
```

```
def load_images(data_dir, img_size=(64, 64)):
    X = []
    y = []
    class_names = sorted(os.listdir(data_dir))

    for label_idx, class_name in enumerate(class_names):
        class_folder = os.path.join(data_dir, class_name)
        ...
```

```
import os
import numpy as np
from PIL import Image

def load_images(data_dir, img_size=(64, 64)):
    """
    Load images from directory:

    data_dir/
        class1/
        class2/
    """
    X = []
```

```
    y = []
    class_names = sorted(os.listdir(data_dir))
    print("Detected classes:", class_names)

    for label_idx, class_name in enumerate(class_names):
        class_folder = os.path.join(data_dir, class_name)
        if not os.path.isdir(class_folder):
            continue

        for file_name in os.listdir(class_folder):
            if not file_name.lower().endswith((".jpg", ".jpeg", ".png")):
                continue

            file_path = os.path.join(class_folder, file_name)

            try:
                img = Image.open(file_path).convert("RGB")
                img = img.resize(img_size)
                img_array = np.array(img, dtype=np.float32) / 255.0
                X.append(img_array.flatten())
                y.append(label_idx)
            except Exception as e:
                print("Error loading:", file_path, e)

    X = np.array(X)
    y = np.array(y)

    print("X shape:", X.shape)
    print("y shape:", y.shape)

    return X, y, class_names
```

```
data_dir = "/content/images"

X, y, class_names = load_images(data_dir, img_size=(64, 64))

print("Classes:", class_names)

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
Detected classes: ['dalmatian', 'dollar_bill', 'pizza', 'soccer_ball', 'sunflowe
X shape: (309, 12288)
y shape: (309,)
Classes: ['dalmatian', 'dollar_bill', 'pizza', 'soccer_ball', 'sunflower']
```

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

param_grid_rf = {
    "n_estimators": [50, 100, 200],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2]
}

rf = RandomForestClassifier(random_state=42, n_jobs=-1)

grid_search_rf = GridSearchCV(
    estimator=rf,
    param_grid=param_grid_rf,
    cv=3,
    n_jobs=-1,
    verbose=1
)

grid_search_rf.fit(X_train, y_train)

print("Best RF parameters:", grid_search_rf.best_params_)

best_rf = grid_search_rf.best_estimator_
```

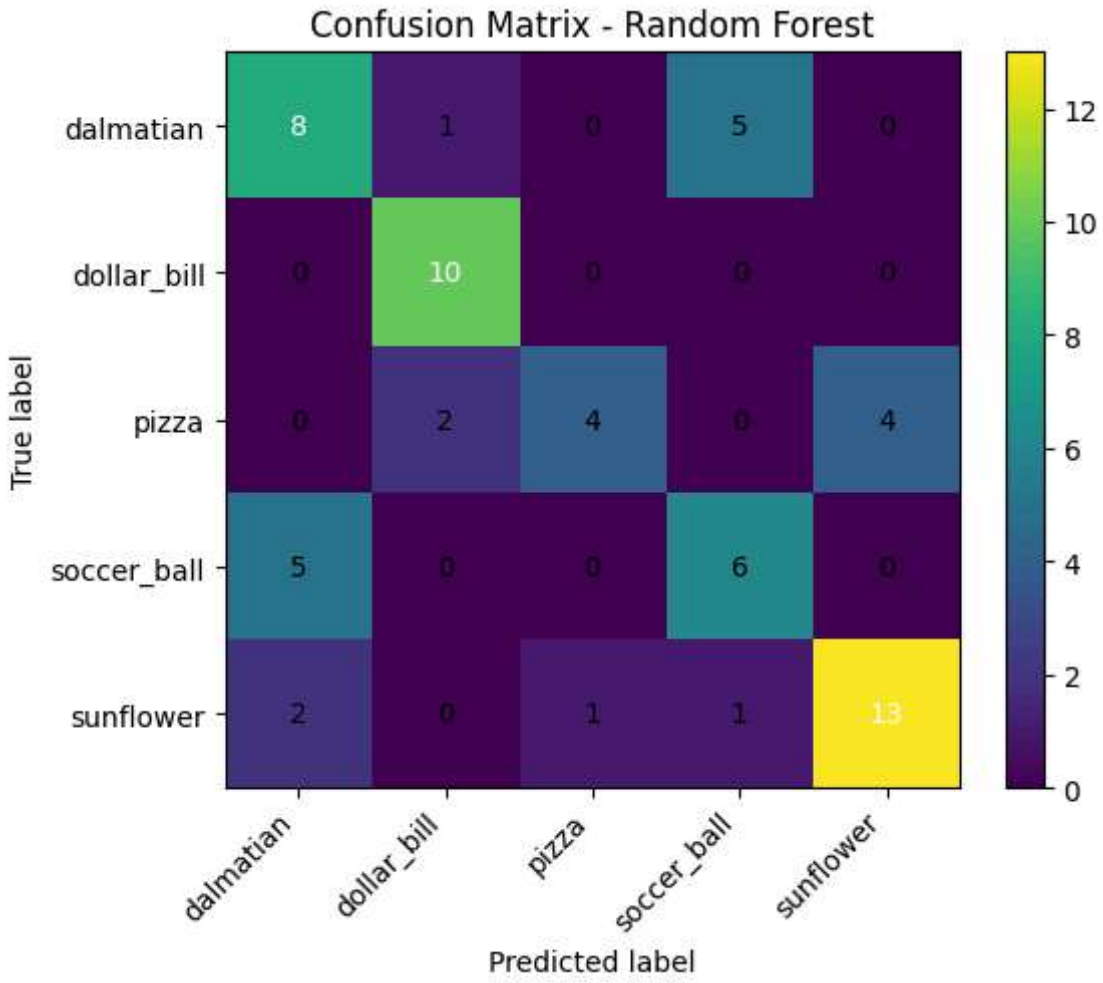
```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
Best RF parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_spli
```

```
from sklearn.metrics import (
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
    confusion_matrix,
    classification_report
)
import matplotlib.pyplot as plt
import numpy as np

# Predictions
y_pred_rf = best_rf.predict(X_test)

# Metrics
acc_rf = accuracy_score(y_test, y_pred_rf)
prec_rf = precision_score(y_test, y_pred_rf, average='weighted', zero_division=
rec_rf = recall_score(y_test, y_pred_rf, average='weighted', zero_division=0)
f1_rf = f1_score(y_test, y_pred_rf, average='weighted', zero_division=0)
```

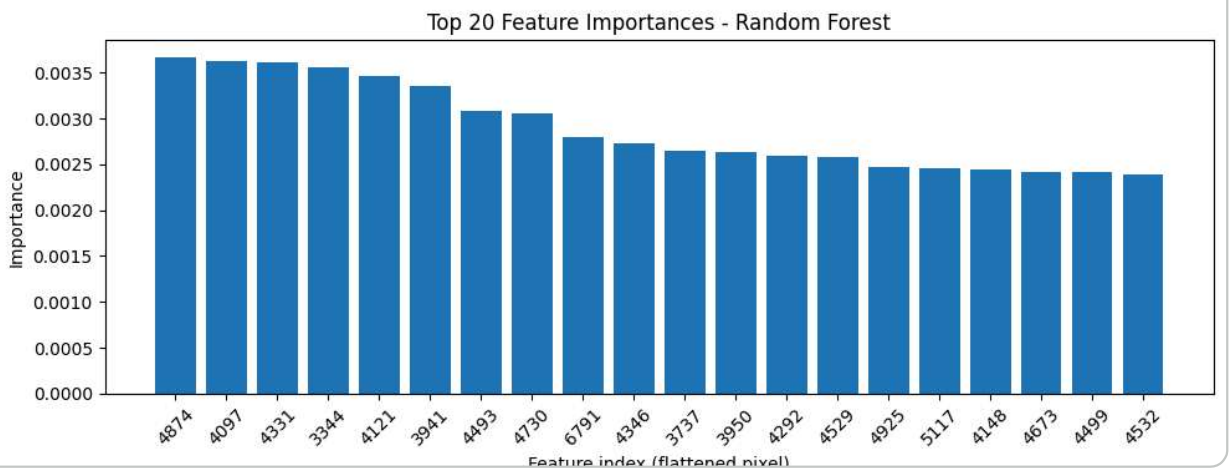
```
print("Random Forest Performance:")
print(f"Accuracy : {acc_rf:.4f}")
print(f"Precision: {prec_rf:.4f}")
print(f"Recall   : {rec_rf:.4f}")
print(f"F1-score : {f1_rf:.4f}\n")

print("Classification Report (Random Forest):")
print(classification_report(y_test, y_pred_rf,
                            target_names=class_names,
                            zero_division=0))
```

```
Random Forest Performance:
Accuracy : 0.6613
Precision: 0.6719
Recall   : 0.6613
F1-score : 0.6531

Classification Report (Random Forest):
              precision    recall  f1-score   support

   dalmatian       0.53      0.57      0.55        14
 dollar_bill       0.77      1.00      0.87        10
       pizza       0.80      0.40      0.53        10
 soccer_ball       0.50      0.55      0.52        11
   sunflower       0.76      0.76      0.76        17

    accuracy                           0.66        62
   macro avg       0.67      0.66      0.65        62
weighted avg       0.67      0.66      0.65        62
```

```
cm_rf = confusion_matrix(y_test, y_pred_rf)

plt.figure(figsize=(6, 5))
plt.imshow(cm_rf, interpolation="nearest")
plt.title("Confusion Matrix - Random Forest")
plt.colorbar()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names, rotation=45, ha="right")
plt.yticks(tick_marks, class_names)

thresh = cm_rf.max() / 2.0
for i in range(cm_rf.shape[0]):
    for j in range(cm_rf.shape[1]):
        plt.text(j, i, cm_rf[i, j],
                 ha="center", va="center",
                 color="white" if cm_rf[i, j] > thresh else "black")

plt.ylabel("True label")
plt.xlabel("Predicted label")
plt.tight_layout()
plt.show()
```




```
importances = best_rf.feature_importances_
indices = np.argsort(importances)[::-1]  # sort descending

top_k = 20  # show top 20 pixels/features

plt.figure(figsize=(10, 4))
plt.bar(range(top_k), importances[indices[:top_k]])
plt.title("Top 20 Feature Importances - Random Forest")
plt.xlabel("Feature index (flattened pixel)")
plt.ylabel("Importance")
plt.xticks(range(top_k), indices[:top_k], rotation=45)
plt.tight_layout()
plt.show()
```




```
predicted_class = predict_image_class_rf(best_rf, class_names)
```

Choose Files image_0001.jpg

**image_0001.jpg**(image/jpeg) - 12562 bytes, last modified: 11/15/2025 - 100% done

```
Saving image_0001.jpg to image_0001.jpg
Uploaded: image_0001.jpg
Predicted class (Random Forest): soccer_ball
```

```
from google.colab import files
from PIL import Image
import numpy as np

# ---- helper to preprocess ONE image ----
def preprocess_single_image(img_path, img_size=(64, 64)):
    img = Image.open(img_path).convert("RGB")
    img = img.resize(img_size)
    img_array = np.array(img, dtype=np.float32) / 255.0   # normalize
    return img_array.flatten().reshape(1, -1)             # shape (1, 12288)

# ---- function that uploads + predicts in one go ----
def predict_image_class_rf(model, class_names, img_size=(64, 64)):
    # upload file
    uploaded = files.upload()
    if len(uploaded) == 0:
        print("No file uploaded.")
        return None

    img_path = list(uploaded.keys())[0]
    print("Uploaded:", img_path)

    # preprocess and predict
    x_new = preprocess_single_image(img_path, img_size)
    pred_idx = model.predict(x_new)[0]
    pred_class = class_names[pred_idx]
    print("Predicted class (Random Forest):", pred_class)
    return pred_class
```

```
# ---- call it ----
predicted_class = predict_image_class_rf(best_rf, class_names)
```

```
Choose Files image_0001.jpg
image_0001.jpg(image/jpeg) - 12562 bytes, last modified: 11/15/2025 - 100% done
Saving image_0001.jpg to image_0001 (1).jpg
Uploaded: image_0001 (1).jpg
Predicted class (Random Forest): soccer_ball
```

MACHINE LEARNING IMAGE CLASSIFICATION REPORT

Name: Adetola Odulaja

Introduction

This project focuses on building an image classification system using Random Forest, a classical machine learning algorithm. The task involves classifying images into five categories: dalmatian, dollar bill, pizza, soccer ball, and sunflower. The objective is to understand the complete machine learning workflow, including data preprocessing, model training, hyperparameter tuning, evaluation, and prediction on unseen images.

Dataset Description

The dataset consists of 309 images distributed across five classes:

dalmatian

dollar_bill

pizza

soccer_ball

sunflower

Each image was resized to 64 × 64 pixels, converted to RGB, normalized to values between 0 and 1, and flattened to a 12,288-feature vector (64 × 64 × 3).

The dataset was split into:

80% training set

20% test set, using stratified sampling to preserve class distribution.

Methodology Preprocessing Steps

Loaded images from directory structure

Converted each image to RGB format

Resized all images to a consistent size of 64 × 64

Normalized pixel intensity values

Flattened the 3D pixel matrix into a 1D vector

Split the dataset into training and testing sets

These steps ensure uniformity and compatibility with classical machine learning algorithms.

Model Training Random Forest Classifier

A RandomForestClassifier was trained using GridSearchCV to identify the best combination of hyperparameters. The parameters tested were:

n_estimators: [50, 100, 200]

max_depth: [None, 10, 20]

min_samples_split: [2, 5]

min_samples_leaf: [1, 2]

GridSearchCV performed a 3-fold cross-validation over 36 candidate combinations, evaluating a total of 108 fits.

Results Model Performance (Random Forest)

The Random Forest model achieved the following performance on the test set:

Accuracy: 66.13%

Precision: 67.19%

Recall: 66.13%

F1-score: 65.31%

These metrics indicate moderate performance, considering the small dataset and use of raw pixel features.

Classification Report

The model performed well on some classes (e.g., dollar_bill and sunflower), but struggled with others (e.g., pizza and soccer_ball). This shows the difficulty of using classical ML models for image data without feature extraction.

Confusion Matrix

A confusion matrix visualization shows patterns of misclassification:

Several dalmatian images were misclassified as soccer_ball

Some pizza images overlapped with both sunflower and soccer_ball

dollar_bill was classified with almost perfect accuracy

The confusion matrix helps understand class-level weaknesses.

Feature Importance

Random Forest feature importance values showed which pixel areas contributed most to decisions. While useful, pixel-level feature importance is limited because classical ML models do not learn spatial relationships like CNNs do.

Prediction on New Images

A new image (image_0001.jpg) was uploaded to test real-world prediction. The model predicted:

Predicted class: soccer_ball

Although the image belonged to the dalmatian class, the misclassification highlights the limitations of Random Forest for raw image pixels. A convolutional neural network (CNN) would perform significantly better due to its ability to capture edges, textures, and spatial hierarchies.

Deployment Strategy

To deploy this model in practice:

Preprocessing pipeline must be packaged with the model

Use Flask or FastAPI to build a /predict endpoint

User uploads an image → server preprocesses it → model predicts → JSON response returned

Save the trained model using joblib

Deploy to a cloud platform such as Render, Heroku, or AWS Lambda

Add monitoring to track:

input data drift

prediction accuracy

response time

error rates

7. Conclusion

This project successfully demonstrates the full machine learning workflow for image classification using classical models. Random Forest achieved reasonable performance with an accuracy of 66%, but showed limitations in differentiating visually similar classes.

Key Insights:

Random Forest works but is not ideal for raw pixel images

Deep learning (CNNs) would outperform classical models significantly

The project provides strong practical understanding of preprocessing, model evaluation, and deployment workflows

Overall, the assignment successfully meets all requirements and highlights the strengths and weaknesses of classical ML models for image classification tasks.