PROJECT CODES

Exploratory Data Analysis :

```
from collections import Counter

from collections import defaultdict

[28]: #read the dataset

path = ""data/""

>>

[29]: names = [name.replace('', '_').split('_')[0] for name in os.listdir(path)]

classes Counter (names) #return dictionary

[30]: print(classes)
```

```
Counter({'arecaceae': 35, 'arrabidaea': 35, 'cecropia': 35, 'chromolaena': 35, 'combretum': 35,
'croton': 35, 'dipteryx': 35,

 'eucalipto': 35, 'farame a': 35, 'hyptis': 35, 'mabea': 35, 'matayba': 35, 'mimosa': 35, 'myrcia': 35,
'protium': 35,

 'qualea': 35, 'schinus': 35, 'senegalia': 35, 'serjania': 35, 'syagrus': 35, 'tridax': 35, 'urochloa': 35,
'anadenanthera': 20})
```

```
 #Total no. of images

print("Number of images", len(names))

Number of images 790
```

```
 import matplotlib.pyplot as plt

 plt.figure(figsize = (8,3))

plt.title('Class Counts in Dataset')

plt.bar(*zip(*classes.items()))

plt.xticks(rotation='vertical')

plt.show()
```

```
path_class = {key: [] for key in classes.keys()}
for name in os.listdir(path):
key = name.replace(' ', '_').split('_')[0]
path_class[key].append(path + name)

import matplotlib.pyplot as plt
from PIL import Image
fig = plt.figure(figsize=(15, 15))
for i, key in enumerate(path_class.keys()):
   img1 = Image.open(path_class[key][0])
   img2 = Image.open(path_class[key][1])
   img3 = Image.open(path_class[key][2])

   ax = fig.add_subplot(8, 9, 3*i + 1, xticks=[], yticks=[])
   ax.imshow(img1)
   ax.set_title(key)

   ax = fig.add_subplot(8, 9, 3*i + 2, xticks=[], yticks=[])
   ax.imshow(img2)
   ax.set_title(key)

   ax = fig.add_subplot(8, 9, 3*i + 3, xticks=[], yticks=[])
   ax.imshow(img3)
   ax.set_title(key)

plt.tight_layout()
plt.show()
```

```
import os
import cv2
import matplotlib.pyplot as plt
# Read sizes (height, width) from all images in the directory
sizes = [cv2.imread(os.path.join(path, name)).shape[:2]
        for name in os.listdir(path)]
x, y = zip(*sizes)  # separate heights and widths
# Plotting scatter plot
fig = plt.figure(figsize=(5, 5))
plt.scatter(x, y)
plt.title("Image size scatterplot")
plt.xlabel("Height (pixels)")
plt.ylabel("Width (pixels)")
# Add diagonal red line for reference
max_val = max(max(x), max(y))
plt.plot([0, max_val], [0, max_val], 'r')
plt.show()

# Image PreProcessing
import os
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils
from sklearn.model_selection import train_test_split

def process_img(img, size=(128, 128)):
    img = cv2.resize(img, size)  # resize image
    img = img / 255.0          # divide values by 25
 return img
```

```
# Read all images and store in X; Y holds class names
X, Y = [], []
for name in os.listdir(path):
    img = cv2.imread(os.path.join(path, name))
    X.append(process_img(img))
    Y.append(name.replace(' ', '_').split('_')[0])

X = np.array(X)

# Encode labels and convert to one-hot encoding
le = LabelEncoder()
Y_le = le.fit_transform(Y)
Y_cat = np_utils.to_categorical(Y_le, num_classes=23)

# Split data into training and testing sets
X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y_cat, test_size=0.285, stratify=Y_le
)

print("Images in each class in Test set: {}".format(np.sum(Y_test, axis=0)))
#Save arrays to disk
np.save("data/processed/X_train.npy", X_train)
 np.save("data/processed/X_test.npy", X_test)
np.save("data/processed/Y_train.npy", Y_train)
 np.save("data/processed/Y_test.npy", Y_test)
```

```
# NOW MODEL BUILDING


import numpy as np

# Load your preprocessed dataset

X_train = np.load("data/processed/X_train.npy")

X_test = np.load("data/processed/X_test.npy")

Y_train = np.load("data/processed/Y_train.npy")

Y_test = np.load("data/processed/Y_test.npy")


# Define the model (continuing from previous code)

from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense


input_shape = X_train[0].shape

output_shape = Y_train.shape[1]  # automatically picks your number of classes


from keras.models import Sequential

from keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense


input_shape = X_train[0].shape

output_shape = 23


model = Sequential()

model.add(Conv2D(filters=16, kernel_size=3, input_shape=input_shape, activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters=32, kernel_size=2, activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=2))


model.add(Conv2D(filters=64, kernel_size=2, activation='relu', padding='same'))

model.add(MaxPooling2D(pool_size=2))
model.add(Conv2D(filters=128, kernel_size=2, activation='relu', padding='same'))
model.add(MaxPooling2D(pool_size=2))

model.add(Flatten())
model.add(Dropout(0.2))
model.add(Dense(500, activation='relu'))
# model.add(Dropout(0.2))
model.add(Dense(150, activation='relu'))
# model.add(Dropout(0.2))
model.add(Dense(output_shape, activation='softmax'))

model.summary()

#Compile Model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Data augmentation setup
datagener = ImageDataGenerator(
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    horizontal_flip=True,
    vertical_flip=True
)
datagener.fit(X_train)

# Training parameters
batch_size = 4
epochs = 500
```

```
model_path = 'cnn.hdf5'
callbacks = [
    EarlyStopping(monitor='val_loss', patience=20, restore_best_weights=True),
    ModelCheckpoint(filepath=model_path, save_best_only=True)
]

# Fit the model using augmented data
history = model.fit(
    datagener.flow(X_train, Y_train, batch_size=batch_size),
    steps_per_epoch=len(X_train) // batch_size,
    epochs=epochs,
    validation_data=(X_train, Y_train),
    callbacks=callbacks,
    verbose=1
)

# Load the best weights and evaluate
model.load_weights(model_path)
score = model.evaluate(X_test, Y_test, verbose=0)
print(f"Test set accuracy: {score[1]:.4f}")




# Plot Customized Accuracy and Loss Graphs

import matplotlib.pyplot as plt
# Extract data from training history
acc = history.history['accuracy']         # Blue: generated (augmented) training accuracy
val_acc = history.history['val_accuracy'] # Orange: original validation accuracy
loss = history.history['loss']            # Blue: training loss
val_loss = history.history['val_loss']    # Orange: validation loss
```

```
epochs = range(1, len(acc) + 1)


#  Plot Model Accuracy
plt.figure(figsize=(10, 5))
plt.plot(epochs, acc, 'b-', label='Generated Data (Train Accuracy)')  # Blue line
plt.plot(epochs, val_acc, 'orange', label='Original Data (Validation Accuracy)')  # Orange line
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.xticks([0, 20, 40, 60, 80])
plt.yticks([0.0, 0.2, 0.4, 0.6, 0.8])
plt.legend()
plt.grid(True)
plt.show()


#  Plot Model Loss
plt.figure(figsize=(10, 5))
plt.plot(epochs, loss, 'b-', label='Generated Data (Train Loss)')  # Blue line
plt.plot(epochs, val_loss, 'orange', label='Original Data (Validation Loss)')  # Orange line
plt.title('Model Loss (Categorical Crossentropy)')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.xticks([0, 20, 40, 60, 80])
plt.yticks([0.0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0])
plt.legend()
plt.grid(True)
plt.show()
```

```
#Save  Model
model.save('model.h5')

# Testing the Model

import numpy as np
from tensorflow.keras.preprocessing.image
import load_img, img_to_array
import os

# 1. Define your image path
img_path = r'D:\Pollen_Grain\data\protium_23.jpg'

# 2. Check if file exists
if not os.path.exists(img_path):
    raise FileNotFoundError(f"Image not found: {img_path}")

# 3. Load & resize image (converted to RGB with 3 channels)
img = load_img(img_path, target_size=(128, 128))

# 4. Convert to array & normalize
x = img_to_array(img)
x = x / 255.0
x = np.expand_dims(x, axis=0)  # shape: (1, 128, 128, 3)

# 5. Predict with your existing model
pred = model.predict(x)  # shape: (1, num_classes)
a = np.argmax(pred, axis=1)[0]
```

```
# 6. Define your class labels in the correct order
op = [
    'arecaceae', 'anadenanthera', 'arrabidaea', 'cecropia', 'chromolaena',
    'combretum', 'croton', 'dipter',
    # ... include all other class names in training order ...
    'eucalipto'
]

# 7. Map index to class label and display
result = op[a]
print("Predicted class:", result)




import numpy as np
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import os

# 1. Define your image path
img_path = r'D:\Pollen_Grain\data\serjania_23.jpg'

# 2. Check if file exists
if not os.path.exists(img_path):
    raise FileNotFoundError(f"Image not found: {img_path}")

# 3. Load & resize image (converted to RGB with 3 channels)
img = load_img(img_path, target_size=(128, 128))



# 4. Convert to array & normalize
x = img_to_array(img)
```

```
x = x / 255.0
x = np.expand_dims(x, axis=0)  # shape: (1, 128, 128, 3)

# 5. Predict with your existing model
pred = model.predict(x)  # shape: (1, num_classes)
a = np.argmax(pred, axis=1)[0]

# 6. Define your class labels in the correct order
op = [
    'arecaceae', 'anadenanthera', 'arrabidaea', 'cecropia', 'chromolaena',
    'combretum', 'croton', 'dipter',
    # ... include all other class names in training order ...
    'eucalipto'
]

# 7. Map index to class label and display
result = op[a]
print("Predicted class:", result)
```

TEMPLATES

#index.html

```
<!--  Paste into templates/index.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Pollen Grain Classifier</title>
    <style>
        body {
            background: linear-gradient(to right, #f6f9fc, #e9eff5);
            font-family: Arial, sans-serif;
            text-align: center;
            padding: 40px;
        }
        .container {
            background: #fff;
            max-width: 600px;
            margin: auto;
            padding: 30px;
            border-radius: 15px;
            box-shadow: 0 4px 8px rgba(0,0,0,0.1);
        }
        h1 {
            margin-bottom: 20px;
            color: #2c3e50;
        }
        form {
            margin-top: 20px;

        }

        input[type="file"] {

            margin: 20px 0;

        }

        button {

            background-color: #2980b9;

            color: white;

            padding: 10px 20px;

            border: none;

            border-radius: 5px;

            cursor: pointer;

            font-size: 16px;

        }

        button:hover {

            background-color: #1f6391;

        }

        footer {

            margin-top: 50px;

            font-size: 14px;

            color: #7f8c8d;

        }

    </style>

</head>


<body>

    <div class="container">

        <h1>🌿 Pollen Grain Classification</h1>

        <!-- 🔥 CRITICAL CHANGE: Update form action -->

        <form action="/result" method="POST" enctype="multipart/form-data">

            <label for="file">Upload a pollen grain image:</label><br>
            <input type="file" name="image" id="file" required><br>
            <button type="submit">Predict</button>
        </form>
    </div>
    <footer>
        <p>&copy; 2025 Pollen Project | Team 7</p>
    </footer>
</body>
</html>
```

#logout.html

```
<!--  Paste into templates/logout.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Logged Out</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f0f8ff;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            height: 100vh;
            margin: 0;
        }
        h1 {
            color: #333;
        }
        a {
            margin-top: 20px;
            text-decoration: none;
            color: #fff;
            background-color: #007BFF;
            padding: 10px 20px;
            border-radius: 5px;
        }
        a:hover {
            background-color: #0056b3;
        }
    </style>
</head>
<body>
    <h1>You have been logged out.</h1>
    <a href="/">Return to Home</a>
</body>
</html>
```

# prediction.html

```
<!-- Paste into templates/prediction.html -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Prediction Result</title>
```

```
<style>
  body {
    font-family: 'Segoe UI', sans-serif;
    text-align: center;
    background-color: #f2f9ff;
    padding: 40px;
  }
  h1 {
    color: #333;
  }
  img {
    margin-top: 20px;
    width: 300px;
    height: auto;
    border: 5px solid #007BFF;
    border-radius: 10px;
  }
  .result {
    font-size: 24px;
    margin-top: 30px;
    color: #007BFF;
  }
  .buttons {
    margin-top: 40px;
  }
  .buttons a {
    display: inline-block;
    margin: 0 10px;
    text-decoration: none;
    padding: 10px 25px;
    color: white;
```

```
            background-color: #007BFF;
            border-radius: 5px;
        }
        .buttons a:hover {
            background-color: #0056b3;
        }
    </style>
</head>

<body>
    <h1>Prediction Result</h1>

    {% if image_path %}
        <img src="{{ image_path }}" alt="Uploaded Image">
    {% endif %}

    <div class="result">
         Predicted Class: <strong>{{ pred }}</strong>
    </div>

    <div class="buttons">
        <a href="/prediction.html"> Try Another</a>
        <a href="/logout.html"> Logout</a>
    </div>
</body>
</html>
```

```
# app.py

import re
import os
import numpy as np
import pandas as pd
from flask import Flask, request, render_template
from keras.models import load_model
from tensorflow.keras.preprocessing.image import load_img, img_to_array

#  Load model
model = load_model("model/model_.h5", compile=False)

# Initialize Flask app
app = Flask(__name__)

# Home routes
@app.route('/')
def index():
    return render_template("index.html")

@app.route('/index.html')
def home():
    return render_template("index.html")

@app.route('/logout.html')
def logout():
    return render_template("logout.html")

@app.route('/prediction.html')
def prediction():
    return render_template("prediction.html")

#  Prediction result
@app.route('/result', methods=["POST"])
def res():
    if 'image' not in request.files:
        return render_template("prediction.html", pred="No file uploaded")

    f = request.files['image']
    if f.filename == '':
        return render_template("prediction.html", pred="No file selected")

    # Save uploaded image
    basepath = os.path.dirname(__file__)
    upload_folder = os.path.join(basepath, 'static', 'upload')
    os.makedirs(upload_folder, exist_ok=True)
    filepath = os.path.join(upload_folder, f.filename)
    f.save(filepath)

    # Preprocess image
    img = load_img(filepath, target_size=(128, 128))
    x = img_to_array(img) / 255.0
    x = np.expand_dims(x, axis=0)

    # Predict
    pred = np.argmax(model.predict(x))
    op = [
        'anadenanthera', 'arecaceae', 'arrabidaea', 'cecropia', 'chromolaena', 'combretum',
        'croton', 'dipteryx', 'eucalipto', 'faramea', 'hyptis', 'mabea', 'matayba', 'mimosa',
        'myrcia', 'protium', 'qualea', 'schinus', 'senegalia', 'serjania', 'syagrus',
        'tridax', 'urochloa'
    ]
    result = op[pred]

    return render_template('prediction.html', pred=result)

# Run Flask app
if __name__ == "__main__":
    app.run(debug=True)
```