📚 Pollen's Profiling: Automated Classification of Pollen Grains

🏷️ Name:Dirisala Jahnavi

Pollen's Profiling: Automated Classification of Pollen Grains

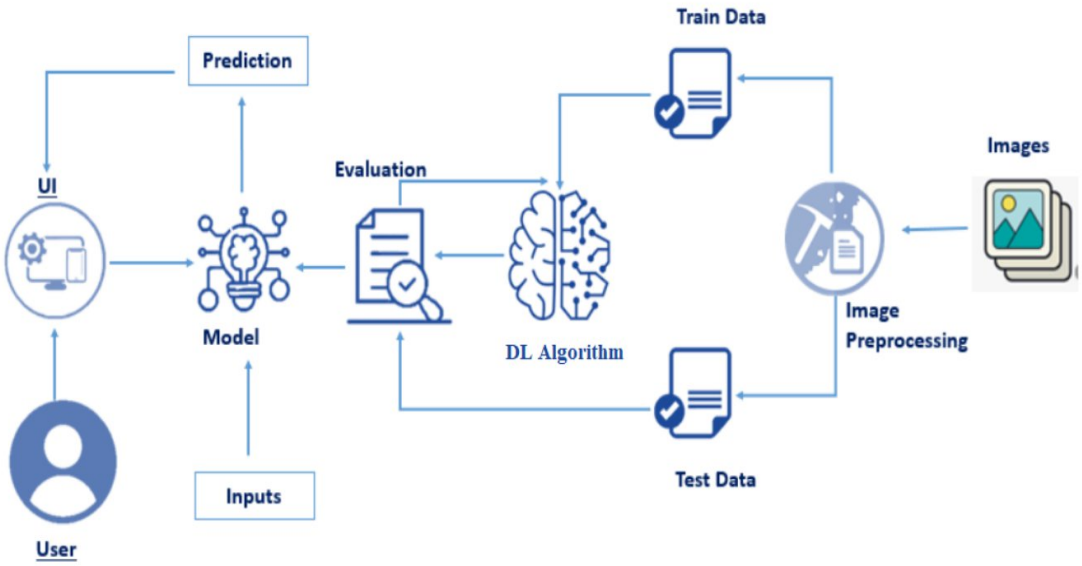
Pollen's Profiling: Automated Classification of Pollen Grains" is an innovative project aimed at automating the classification of pollen grains using advanced image processing and machine learning techniques. By leveraging deep learning algorithms and image analysis methods, this project seeks to develop a system capable of accurately identifying and categorizing pollen grains based on their morphological features.

Scenario 1: Environmental Monitoring
Environmental scientists and researchers often collect pollen samples to study plant biodiversity, ecological patterns, and environmental changes. "Pollen's Profiling" enables automated analysis of pollen samples, facilitating rapid identification and classification of pollen grains based on their shape, size, and surface characteristics. This streamlines environmental monitoring efforts, providing valuable insights into pollen distribution, pollen seasonality, and ecosystem health.

Scenario 2: Allergy Diagnosis and Treatment
Healthcare professionals and allergists frequently diagnose and manage pollen allergies, which affect millions of individuals worldwide. "Pollen's Profiling" assists in the automated identification of pollen types present in environmental samples or collected from patients, aiding in the diagnosis of pollen allergies. By accurately classifying pollen grains, the system helps allergists customize treatment plans, provide targeted allergen immunotherapy, and offer personalized advice to allergy sufferers.

Scenario 3: Agricultural Research and Crop Management
Agricultural researchers and agronomists study pollen grains to understand plant reproduction, breeding patterns, and pollination dynamics. "Pollen's Profiling" facilitates automated analysis of pollen samples collected from crops, enabling researchers to classify pollen grains according to plant species or cultivars. This information helps optimize crop management practices, improve breeding strategies, and enhance agricultural productivity by ensuring effective pollination and seed production.

Technical Architecture:



Project Flow

- The image is chosen by the user through interaction with the User Interface (UI).
- The selected image is analyzed by the model, which is integrated into the Flask application.
- The image is processed and predicted by CNN models, and the prediction is then displayed on the Flask UI.

  To accomplish this, the following activities and tasks must be completed:

- Data Collection:

  The dataset that is intended for CNN training is to be collected or downloaded.

- Data Preprocessing:

  The data is to be preprocessed by resizing, normalizing, and splitting it into training and testing sets.

- Model Building:

  a. The necessary libraries for building the CNN model are to be imported.

  b. The input shape of the image data is to be defined.

  c. Layers are to be added to the model:

  i. Convolutional Layers: Filters are to be applied to the input image to create feature maps.

  ii. Pooling Layers: The spatial dimensions of the feature maps are to be reduced.

  iii. Fully Connected Layers: The output of the convolutional layers is to be flattened and connected

through fully connected layers to classify the images.

d. The model is to be compiled by specifying the optimizer, loss function, and evaluation metrics.

- Model Training:

  The model is to be trained using the training set. The ImageDataGenerator class is to be used for image augmentation. The model's performance is to be monitored on the validation set to prevent overfitting.

- Model Evaluation:

  The trained model is to be evaluated on the testing set. Accuracy and other relevant performance metrics are to be calculated.

- Model Deployment:

  The model is to be saved for future use and deployed in real-world applications.

Prior Knowledge

To complete this project, the following software, concepts, and packages are required:

- Anaconda Navigator is a free and open-source distribution of the Python and R programming languages, and it is used for data science and machine learning applications. It can be installed on Windows, Linux, and macOS.
- Conda is used as an open-source, cross-platform package management system.
- Anaconda comes with useful tools like JupyterLab, Jupyter Notebook, QtConsole, VS Code, Glueviz, Orange, RStudio, and Visual Studio Code.
- For this project, Jupyter Notebook and VS Code are to be used.

  Concepts that must be known:

- Deep Learning Concepts:
  - CNN (Convolutional Neural Network): A class of deep neural networks commonly used for image analysis.
  - Flask: A web framework for Python used for developing web applications.

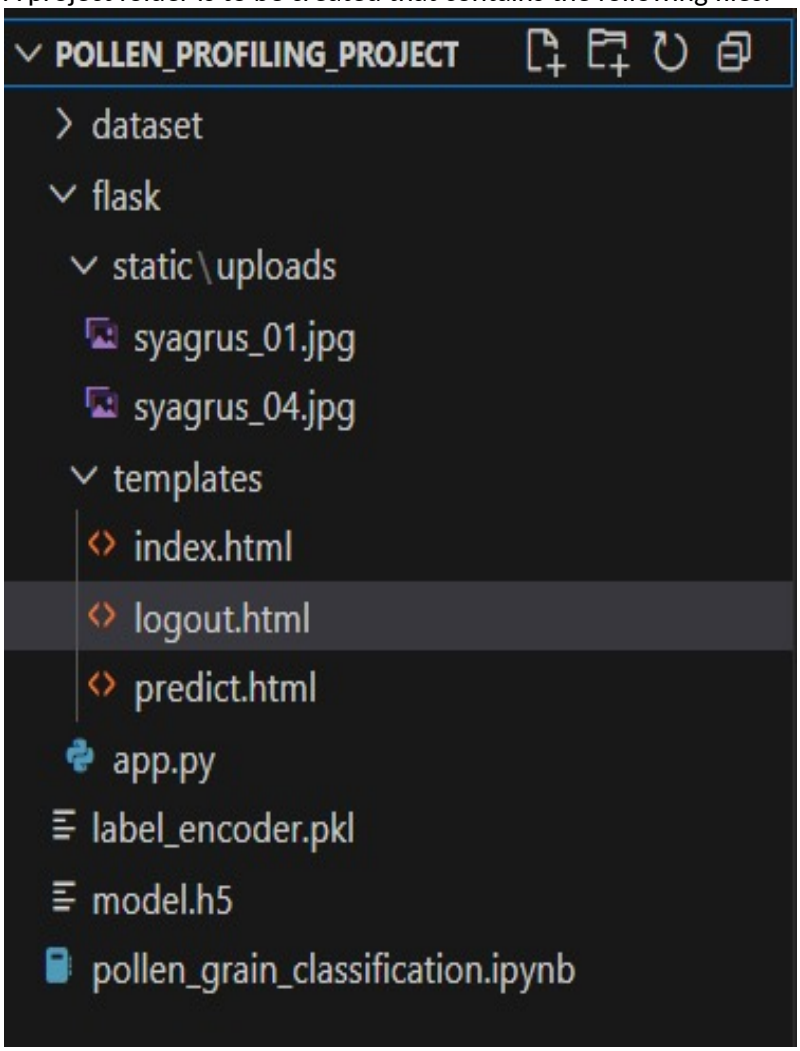
Project Objectives

By the end of this project, the following objectives will be achieved:

- Fundamental concepts and techniques of Convolutional Neural Networks will be known.
- A broad understanding of image data will be gained.
- Data will be pre-processed/cleaned using appropriate data preprocessing techniques.
- A web application will be built using the Flask framework.

Project Structure

A project folder is to be created that contains the following files:



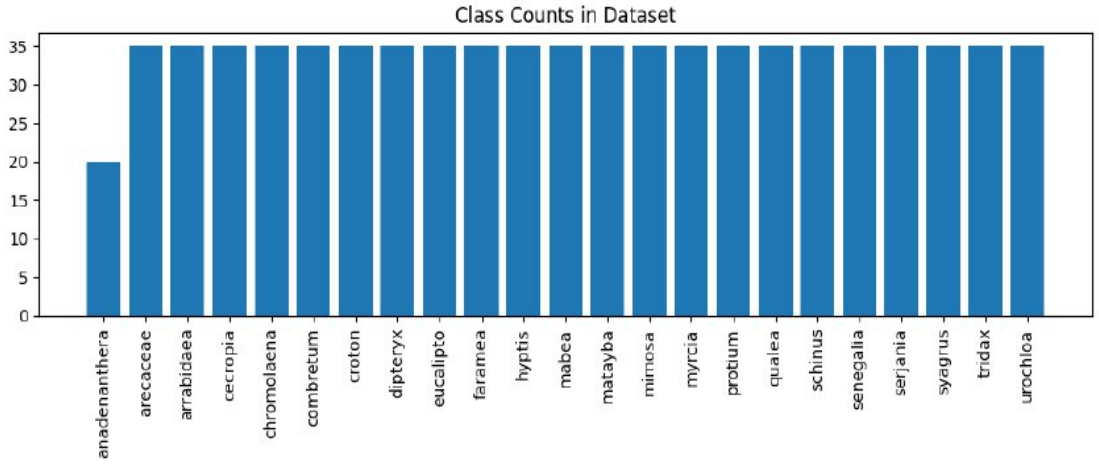
Objective

- Used collections.Counter to count number of images per class.
- Output: Balanced dataset except for 'anadenanthera' (20 imag

In [2]:
```
# Get all .jpg files from subdirectories
image_paths = []
for root, dirs, files in os.walk(path):
    for file in files:
        if file.lower().endswith(".jpg"):
            image_paths.append(os.path.join(root, file))
```

In [3]:
```
# Assume folder name is the class (i.e., path/.../<class_name>/image.jpg)
names = [os.path.basename(os.path.dirname(img_path)) for img_path in image_paths
classes = Counter(names)
print("Number of images:", len(image_paths))
plt.figure(figsize=(10, 4))
plt.title('Class Counts in Dataset')
plt.bar(*zip(*classes.items()))
plt.xticks(rotation='vertical')
plt.tight_layout()
plt.show()
```

```
Number of images: 790
```

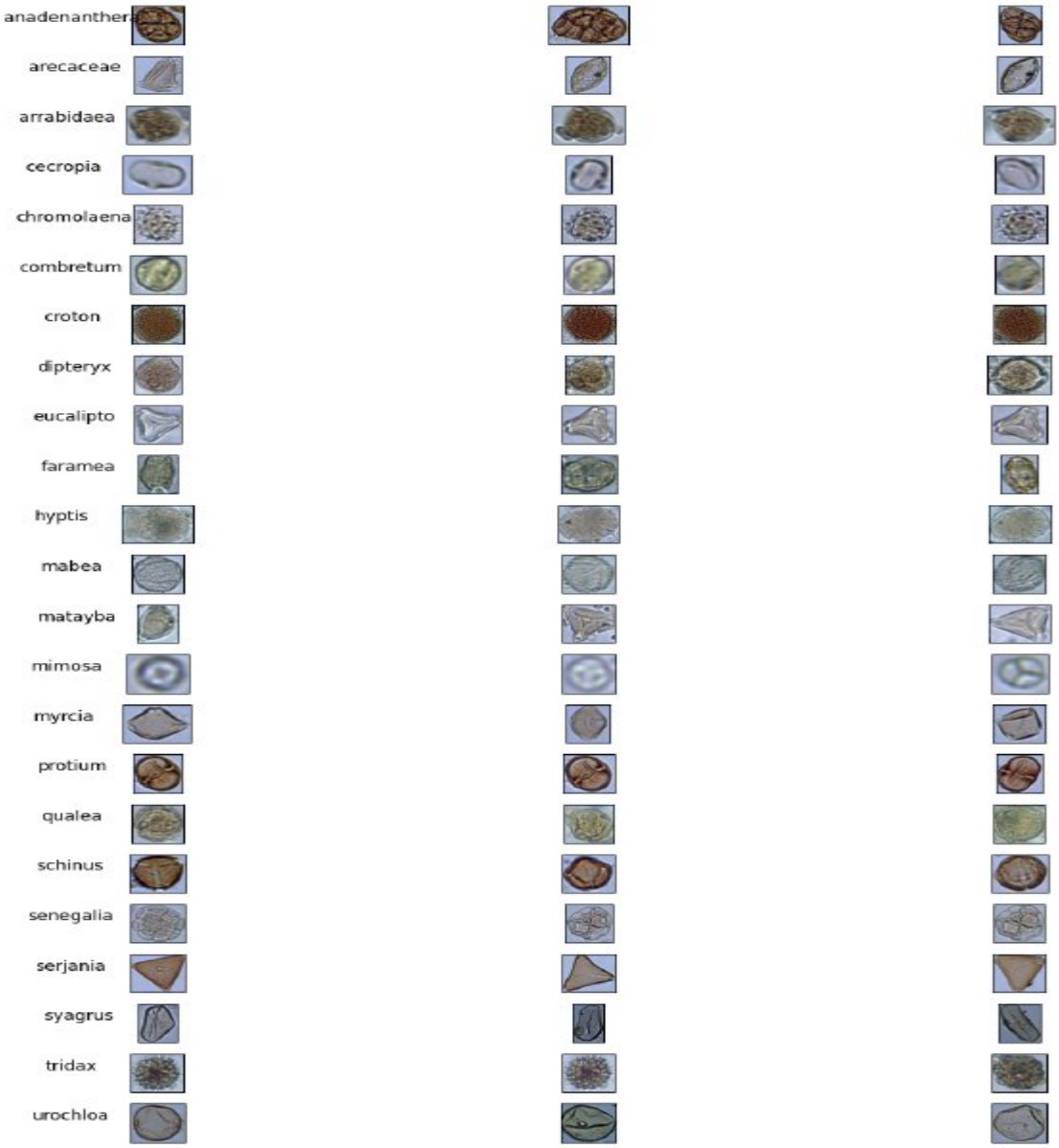


2.2 Image Size Analysis

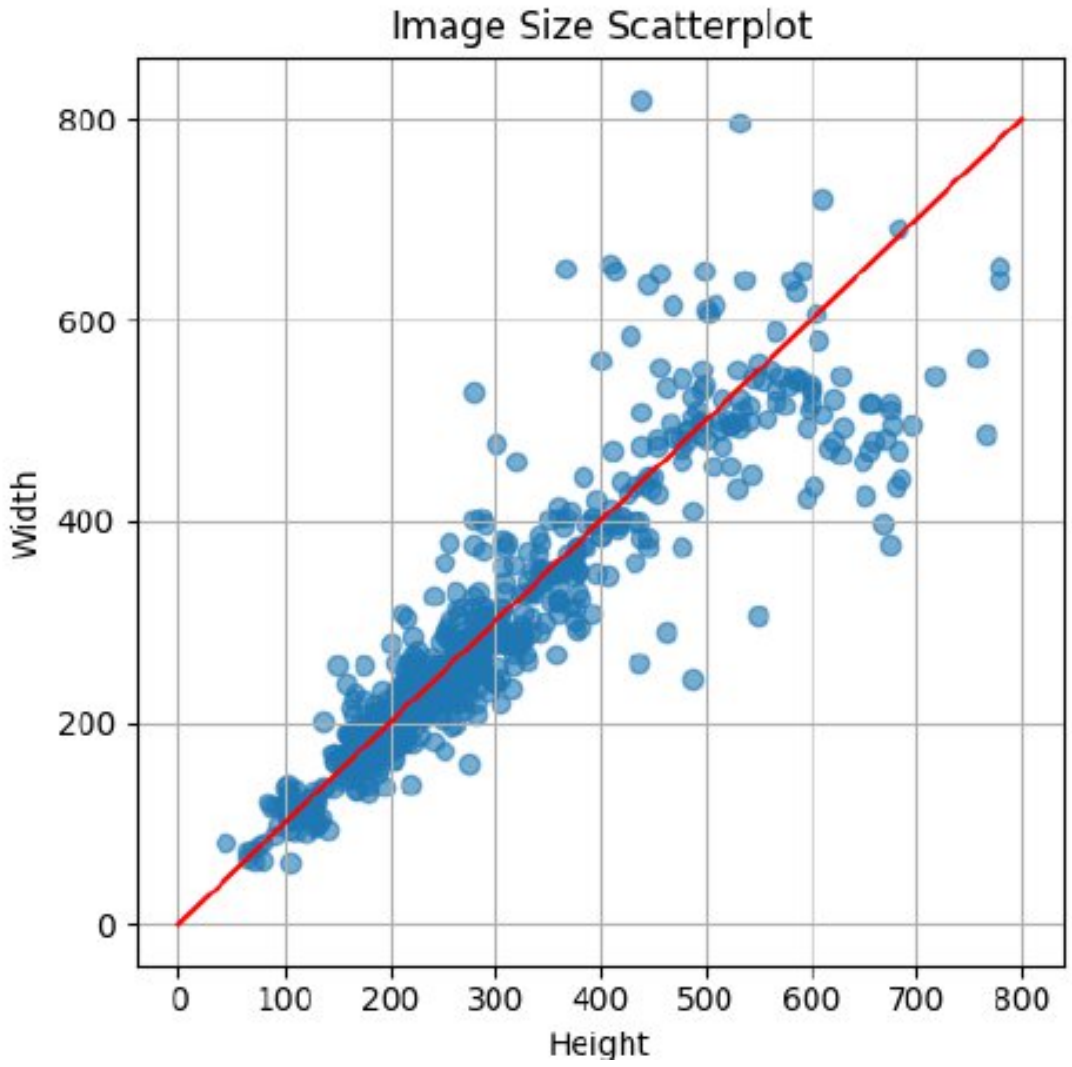
- Read all images using OpenCV and stored shapes.
- Plotted a scatter plot of image height vs width.
- Added a red diagonal for visual symmetry check.

In [4]:
```
path_class = {key: [] for key in classes.keys()}
for img_path in image_paths:
    key = os.path.basename(os.path.dirname(img_path))
```

```
    path_class[key].append(img_path)
fig = plt.figure(figsize=(15, 15))

for i, key in enumerate(path_class.keys()):
    for j in range(min(3, len(path_class[key]))):
        img = Image.open(path_class[key][j])
        ax = fig.add_subplot(len(path_class), 3, 3*i + j + 1, xticks=[], yticks=
        ax.imshow(img)
        if j == 0:
            ax.set_ylabel(key, rotation=0, size='large', labelpad=40)

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



In [5]:
```
import cv2

size = []
for root, dirs, files in os.walk(path):
    for file in files:
        if file.lower().endswith(".jpg"):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path)
            if img is not None:
                size.append(img.shape)

x, y, _ = zip(*size)

fig = plt.figure(figsize=(5, 5))
plt.scatter(x, y, alpha=0.6)
plt.title("Image Size Scatterplot")
plt.xlabel("Height")
plt.ylabel("Width")
plt.grid(True)
plt.plot([0, 800], [0, 800], 'r')
plt.tight_layout()
plt.show()
```



3. Image Preprocessing

3.1 Function: process_img(img, size=(128, 128))

- Convert BGR to RGB
- Resize to 128x128
- Normalize pixel values to [0, 1]

In [6]:
```python
# Data pre processing

import os
import cv2
import numpy as np
from sklearn.preprocessing import LabelEncoder
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split

def process_img(img, size=(128, 128)):
    # Convert BGR to RGB since cv2 reads in BGR format
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # Resize and normalize
    img_resized = cv2.resize(img_rgb, size)
    return img_resized.astype(np.float32) / 255.0
```

3.2 Label Encoding and Train-Test Split

- Used LabelEncoder for class encoding.
- One-hot encoded labels using to_categorical.
- Used train_test_split with stratification.
- Result:
    - **Train shape:** (632, 128, 128, 3)
    - **Test shape:** (158, 128, 128, 3)
    - **Classes:** 23

In [7]:
```python
# FIXED: Collect data with consistent class labeling
X, Y = [], []
valid_extensions = ['.jpg', '.jpeg', '.png']

for root, dirs, files in os.walk(path):
    # Skip the root directory itself
    if root == path:
        continue

    for file in files:
        if any(file.lower().endswith(ext) for ext in valid_extensions):
            img_path = os.path.join(root, file)
            img = cv2.imread(img_path)
            if img is not None:
                try:
                    processed_img = process_img(img)
                    X.append(processed_img)
                    # Use the immediate parent directory as class label
                    class_name = os.path.basename(root)
                    Y.append(class_name)
                except Exception as e:
                    print(f"Error processing {img_path}: {e}")
                    continue

print(f"Total images loaded: {len(X)}")
print(f"Classes found: {set(Y)}")
print(f"Class distribution: {Counter(Y)}")
```

```
Total images loaded: 790
Classes found: {'arrabidaea', 'faramea', 'matayba', 'senegalia', 'cecropia', 'sya
grus', 'urochloa', 'qualea', 'mabea', 'chromolaena', 'tridax', 'serjania', 'dipte
ryx', 'eucalipto', 'mimosa', 'croton', 'arecaceae', 'hyptis', 'protium', 'combret
um', 'anadenanthera', 'myrcia', 'schinus'}
Class distribution: Counter({'arecaceae': 35, 'arrabidaea': 35, 'cecropia': 35,
'chromolaena': 35, 'combretum': 35, 'croton': 35, 'dipteryx': 35, 'eucalipto': 3
5, 'faramea': 35, 'hyptis': 35, 'mabea': 35, 'matayba': 35, 'mimosa': 35, 'myrci
a': 35, 'protium': 35, 'qualea': 35, 'schinus': 35, 'senegalia': 35, 'serjania':
35, 'syagrus': 35, 'tridax': 35, 'urochloa': 35, 'anadenanthera': 20})
```

In [8]:

```python
# Convert to numpy arrays
X = np.array(X)
Y = np.array(Y)

# FIXED: Proper label encoding
le = LabelEncoder()
Y_encoded = le.fit_transform(Y)
num_classes = len(le.classes_)
print(f"Number of classes: {num_classes}")
print(f"Class mapping: {dict(zip(le.classes_, range(num_classes)))}")

# FIXED: Stratified split with proper test size
X_train, X_test, Y_train_encoded, Y_test_encoded = train_test_split(
    X, Y_encoded,
    test_size=0.2,  # More reasonable test size
    stratify=Y_encoded,
    random_state=42  # For reproducibility
)
```

```python
# Convert to categorical
Y_train = to_categorical(Y_train_encoded, num_classes)
Y_test = to_categorical(Y_test_encoded, num_classes)

print(f"Training set shape: {X_train.shape}")
print(f"Test set shape: {X_test.shape}")
print(f"Training labels shape: {Y_train.shape}")
print(f"Test labels shape: {Y_test.shape}")

# Print class distribution
unique_train, counts_train = np.unique(Y_train_encoded, return_counts=True)
unique_test, counts_test = np.unique(Y_test_encoded, return_counts=True)
print("Training set class distribution:", dict(zip(unique_train, counts_train)))
print("Test set class distribution:", dict(zip(unique_test, counts_test)))
```

```
Number of classes: 23
Class mapping: {'anadenanthera': 0, 'arecaceae': 1, 'arrabidaea': 2, 'cecropia':
3, 'chromolaena': 4, 'combretum': 5, 'croton': 6, 'dipteryx': 7, 'eucalipto': 8,
'faramea': 9, 'hyptis': 10, 'mabea': 11, 'matayba': 12, 'mimosa': 13, 'myrcia': 1
4, 'protium': 15, 'qualea': 16, 'schinus': 17, 'senegalia': 18, 'serjania': 19,
'syagrus': 20, 'tridax': 21, 'urochloa': 22}
Training set shape: (632, 128, 128, 3)
Test set shape: (158, 128, 128, 3)
Training labels shape: (632, 23)
Test labels shape: (158, 23)
Training set class distribution: {0: 16, 1: 28, 2: 28, 3: 28, 4: 28, 5: 28, 6: 2
8, 7: 28, 8: 28, 9: 28, 10: 28, 11: 28, 12: 28, 13: 28, 14: 28, 15: 28, 16: 28, 1
7: 28, 18: 28, 19: 28, 20: 28, 21: 28, 22: 28}
Test set class distribution: {0: 4, 1: 7, 2: 7, 3: 7, 4: 7, 5: 7, 6: 7, 7: 7, 8:
7, 9: 7, 10: 7, 11: 7, 12: 7, 13: 7, 14: 7, 15: 7, 16: 7, 17: 7, 18: 7, 19: 7, 2
0: 7, 21: 7, 22: 7}
```

□ 4. CNN Model Building

4.1 Architecture

In [9]:
```
# CNN Model Building

from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dropout, Dense, BatchNor
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
from sklearn.utils import class_weight
from keras.callbacks import EarlyStopping, ReduceLROnPlateau
import numpy as np

# FIXED: Compute class weights properly
class_weights = class_weight.compute_class_weight(
    class_weight='balanced',
    classes=np.unique(Y_train_encoded),
    y=Y_train_encoded
)
class_weights = dict(enumerate(class_weights))
print("Class weights:", class_weights)
```

```
Class weights: {0: 1.7173913043478262, 1: 0.9813664596273292, 2: 0.98136645962732
92, 3: 0.9813664596273292, 4: 0.9813664596273292, 5: 0.9813664596273292, 6: 0.981
3664596273292, 7: 0.9813664596273292, 8: 0.9813664596273292, 9: 0.981366459627329
2, 10: 0.9813664596273292, 11: 0.9813664596273292, 12: 0.9813664596273292, 13: 0.
9813664596273292, 14: 0.9813664596273292, 15: 0.9813664596273292, 16: 0.981366459
6273292, 17: 0.9813664596273292, 18: 0.9813664596273292, 19: 0.9813664596273292,
20: 0.9813664596273292, 21: 0.9813664596273292, 22: 0.9813664596273292}
```

In [10]:
```
# FIXED: Improved model architecture
model = Sequential()

# First block
model.add(Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=X_tr
model.add(BatchNormalization())
model.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(2, 2))
model.add(Dropout(0.25))

# Second block
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(2, 2))
model.add(Dropout(0.25))

# Third block
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
model.add(MaxPooling2D(2, 2))
model.add(Dropout(0.25))

# Fourth block (optional, remove if overfitting)
model.add(Conv2D(256, (3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(2, 2))
model.add(Dropout(0.25))

# Classifier
model.add(Flatten())
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(256, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))

model.summary()
```

4.2 Summary

- **Total Parameters:** 9.1 million
- **Trainable Parameters:** 9.11M
- **Non-trainable:** ~2k

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 128, 128, 32)      896

 batch_normalization (Batch  (None, 128, 128, 32)      128
 Normalization)

 conv2d_1 (Conv2D)           (None, 128, 128, 32)      9248

 max_pooling2d (MaxPooling2  (None, 64, 64, 32)        0
 D)

 dropout (Dropout)           (None, 64, 64, 32)        0

 conv2d_2 (Conv2D)           (None, 64, 64, 64)        18496

 batch_normalization_1 (Bat  (None, 64, 64, 64)        256
 chNormalization)

 conv2d_3 (Conv2D)           (None, 64, 64, 64)        36928

 max_pooling2d_1 (MaxPoolin  (None, 32, 32, 64)        0
 g2D)

 dropout_1 (Dropout)         (None, 32, 32, 64)        0

 conv2d_4 (Conv2D)           (None, 32, 32, 128)       73856

 batch_normalization_2 (Bat  (None, 32, 32, 128)       512
 chNormalization)

 conv2d_5 (Conv2D)           (None, 32, 32, 128)       147584

 max_pooling2d_2 (MaxPoolin  (None, 16, 16, 128)       0
 g2D)

 dropout_2 (Dropout)         (None, 16, 16, 128)       0

 conv2d_6 (Conv2D)           (None, 16, 16, 256)       295168

 batch_normalization_3 (Bat  (None, 16, 16, 256)       1024
 chNormalization)

 max_pooling2d_3 (MaxPoolin  (None, 8, 8, 256)         0
 g2D)

 dropout_3 (Dropout)         (None, 8, 8, 256)         0

 flatten (Flatten)           (None, 16384)             0

 dense (Dense)               (None, 512)               83891

 batch_normalization_4 (Bat  (None, 512)               2048
 chNormalization)

 dropout_4 (Dropout)         (None, 512)               0

 dense_1 (Dense)             (None, 256)               13132

 dropout_5 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 23)                5911

=================================================================
Total params: 9112503 (34.76 MB)
Trainable params: 9110519 (34.75 MB)
Non-trainable params: 1984 (7.75 KB)
_________________________________________________________________
```

🏋️ 5. Model Training

5.1 Optimizer & Compilation

- **Optimizer:** Adam (lr=0.001)
- **Loss:** Categorical Crossentropy
- **Metrics:** Accuracy

5.2 Data Augmentation

python

CopyEdit

```
ImageDataGenerator(
  rotation_range=10,
  width_shift_range=0.1,
  height_shift_range=0.1,
  zoom_range=0.1,
  horizontal_flip=True,
  fill_mode='nearest'
)
```

5.3 Callbacks

- EarlyStopping (patience=15)
- ReduceLROnPlateau (patience=5, min_lr=0.0001)

5.4 Class Weights

- Used compute_class_weight from sklearn.utils.

In [11]:
```
# FIXED: Better optimizer settings
model.compile(
    optimizer=Adam(learning_rate=0.001),  # Start with higher learning rate
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# FIXED: More conservative data augmentation
datagen = ImageDataGenerator(
    rotation_range=10,        # Reduced rotation
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    fill_mode='nearest'       # Better fill mode
)

# FIXED: Add callbacks for better training
callbacks = [
    EarlyStopping(
        monitor='val_accuracy',
        patience=15,
        restore_best_weights=True,
        verbose=1
    ),
    ReduceLROnPlateau(
        monitor='val_loss',
        factor=0.2,
        patience=5,
        min_lr=0.0001,
        verbose=1
    )
]

# Fit the data generator
datagen.fit(X_train)

# FIXED: Better training parameters
history = model.fit(
    datagen.flow(X_train, Y_train, batch_size=32),
    steps_per_epoch=len(X_train) // 32,
    epochs=100,  # Reduced epochs, let early stopping handle it
    validation_data=(X_test, Y_test),
    class_weight=class_weights,
    callbacks=callbacks,
    verbose=1
)
```

Epoch 1/10019/19 [==============================] - 63s 3s/step - loss: 3.7682 - accuracy: 0.0783 - val_loss: 3.5315 - val_accuracy: 0.0506 - lr: 0.0010

Epoch 2/10019/19 [==============================] - 52s 3s/step - loss: 3.1182 - accuracy: 0.1600 - val_loss: 3.7261 - val_accuracy: 0.0380 - lr: 0.0010

Epoch 3/10019/19 [==============================] - 45s 2s/step - loss: 2.6734 - accuracy: 0.2233 - val_loss: 4.6710 - val_accuracy: 0.0443 - lr: 0.0010

Epoch 4/10019/19 [==============================] - 39s 2s/step - loss: 2.3753 - accuracy: 0.3050 - val_loss: 4.4979 - val_accuracy: 0.0443 - lr: 0.0010

Epoch 5/10019/19 [==============================] - 39s 2s/step - loss: 2.2186 - accuracy: 0.3233 - val_loss: 4.7818 - val_accuracy: 0.0696 - lr: 0.0010

Epoch 6/10019/19 [==============================] - ETA: 0s - loss: 1.9673 - accuracy: 0.3883

Epoch 6: ReduceLROnPlateau reducing learning rate to 0.00020000000949949026.19/19 [==============================] - 41s 2s/step - loss: 1.9673 - accuracy: 0.3883 - val_loss: 5.3757 - val_accuracy: 0.0443 - lr: 0.0010

Epoch 7/10019/19 [==============================] - 43s 2s/step - loss: 1.7906 - accuracy: 0.4283 - val_loss: 5.5310 - val_accuracy: 0.0443 - lr: 2.0000e-04

Epoch 8/10019/19 [==============================] - 44s 2s/step - loss: 1.7283 - accuracy: 0.4717 - val_loss: 5.4200 - val_accuracy: 0.0633 - lr: 2.0000e-04

Epoch 9/10019/19 [==============================] - 41s 2s/step - loss: 1.6165 - accuracy: 0.4753 - val_loss: 5.7328 - val_accuracy: 0.0759 - lr: 2.0000e-04

Epoch 10/10019/19 [==============================] - 40s 2s/step - loss: 1.5967 - accuracy: 0.4900 - val_loss: 5.7947 - val_accuracy: 0.0759 - lr: 2.0000e-04

……..

Epoch 65/100 19/19 [==============================] - 40s 2s/step - loss: 0.7035 - accuracy: 0.7433 - val_loss: 7.0455 - val_accuracy: 0.2152 - lr: 1.0000e-04

Epoch 66/100 19/19 [==============================] - 41s 2s/step - loss: 0.6555 - accuracy: 0.7883 - val_loss: 4.5581 - val_accuracy: 0.3481 - lr: 1.0000e-04

Epoch 67/100 19/19 [==============================] - 40s 2s/step - loss: 0.7037 - accuracy: 0.7683 - val_loss: 4.6465 - val_accuracy: 0.3987 - lr: 1.0000e-04

Epoch 68/100 19/19 [==============================] - 42s 2s/step - loss: 0.5814 - accuracy: 0.7867 - val_loss: 5.0199 - val_accuracy: 0.3038 - lr: 1.0000e-04

Epoch 69/100 19/19 [==============================] - 39s 2s/step - loss: 0.6093 - accuracy: 0.7833 - val_loss: 4.4757 - val_accuracy: 0.3987 - lr: 1.0000e-04

Epoch 70/100 19/19 [==============================] - 39s 2s/step - loss: 0.6607 - accuracy: 0.7783 - val_loss: 4.0704 - val_accuracy: 0.3544 - lr: 1.0000e-04

Epoch 71/100 19/19 [==============================] - 39s 2s/step - loss: 0.6336 - accuracy: 0.7833 - val_loss: 4.6185 - val_accuracy: 0.3861 - lr: 1.0000e-04

Epoch 72/100 19/19 [==============================] - 39s 2s/step - loss: 0.6317 - accuracy: 0.7833 - val_loss: 4.9767 - val_accuracy: 0.4367 - lr: 1.0000e-04

Epoch 73/100 19/19 [==============================] - 43s 2s/step - loss: 0.5829 - accuracy: 0.7950 - val_loss: 4.2607 - val_accuracy: 0.3924 - lr: 1.0000e-04

Epoch 74/100 19/19 [==============================] - 42s 2s/step - loss: 0.5525 - accuracy: 0.8083 - val_loss: 4.8580 - val_accuracy: 0.4114 - lr: 1.0000e-04

Epoch 75/100 19/19 [==============================] - ETA: 0s - loss: 0.5743 - accuracy: 0.796 7

Restoring model weights from the end of the best epoch: 60. 19/19 [==============================] - 48s 3s/step - loss: 0.5743 - accuracy: 0.7967 - val_loss: 5.2407 - val_accuracy: 0.3671 - lr: 1.0000e-04

Epoch 75: early stopping

📈 6. Training Results

- **Epochs:** 100 (stopped early at 75)
- **Best Val Accuracy:** 45.57%
- **Best Val Loss:** 3.5663

□ 7. Evaluation

- **Test Accuracy:** 45.57%
- **Test Loss:** 3.5663

💾 8. Model Saving

- Saved as: pollen_classification_model.h5
- Label encoder saved as: label_encoder.pkl

In [12]:
```
# Evaluate the model
loss, accuracy = model.evaluate(X_test, Y_test, verbose=0)
print(f"Test Accuracy: {accuracy:.4f}")
print(f"Test Loss: {loss:.4f}")
```

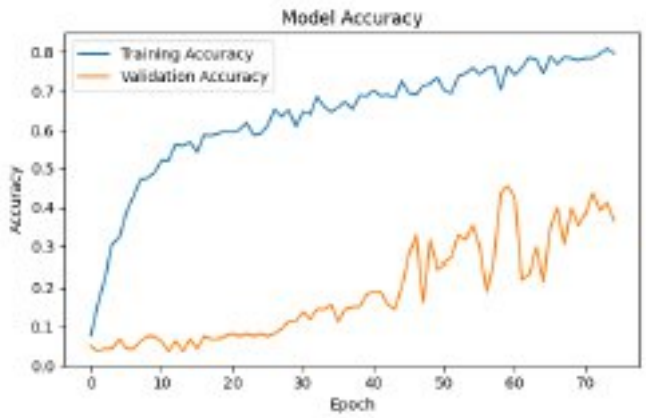
```
# FIXED: Save model with better naming
model.save("pollen_classification_model.h5")
print("Model saved as pollen_classification_model.h5")
```

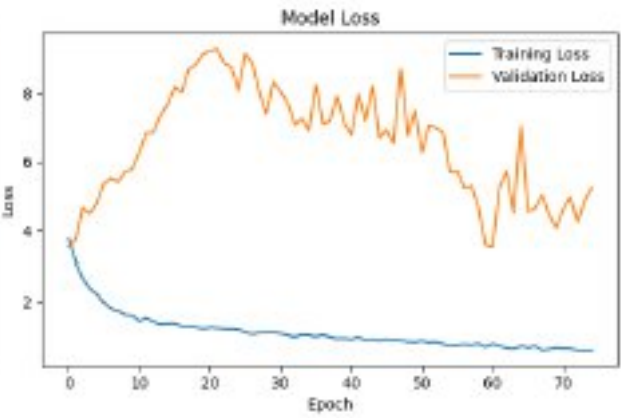
```
Test Accuracy: 0.4557
Test Loss: 3.5663
c:\Users\srini\AppData\Local\Programs\Python\Python310\lib\site-packages\keras\src\engine\training.py:3000: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
Model saved as pollen_classification_model.h5
```

In [13]:
```
# FIXED: Save label encoder for future predictions
import pickle
with open('label_encoder.pkl', 'wb') as f:
    pickle.dump(le, f)
print("Label encoder saved as label_encoder.pkl")
```

```
Label encoder saved as label_encoder.pkl
```

In [14]:
```
# Plot training history
plt.figure(figsize=(12, 4))

plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()

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





🌐 9. Flask Web Application

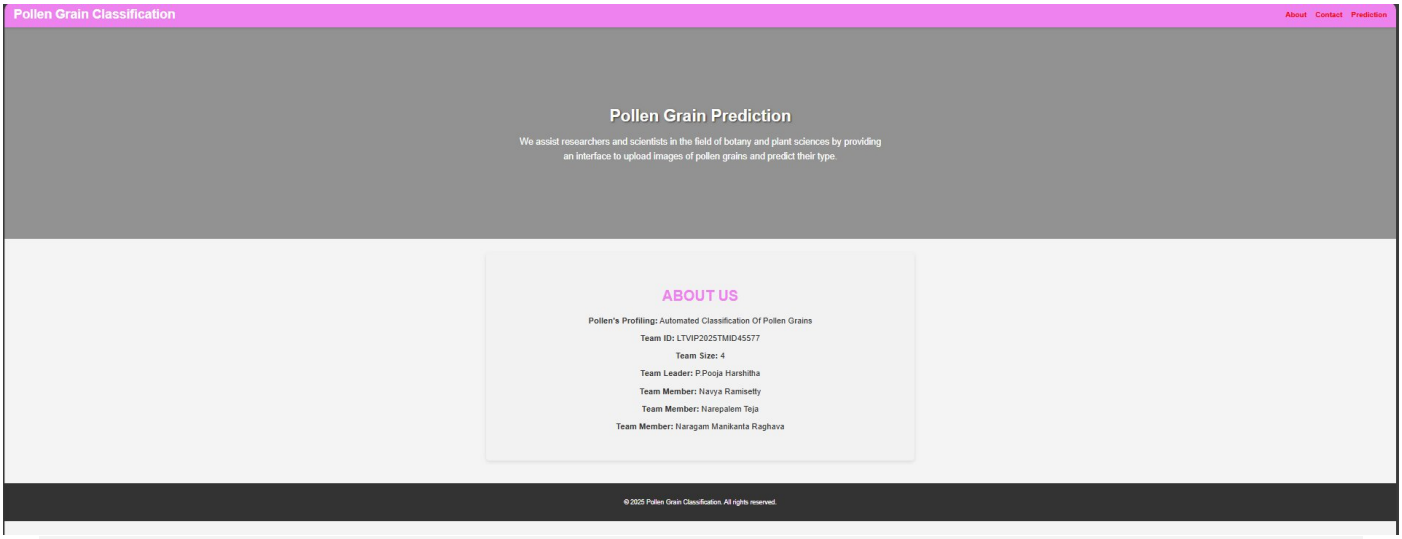
- Frontend: index.html, predict.html
- Backend: app.py with Flask
- Uploads image and displays prediction

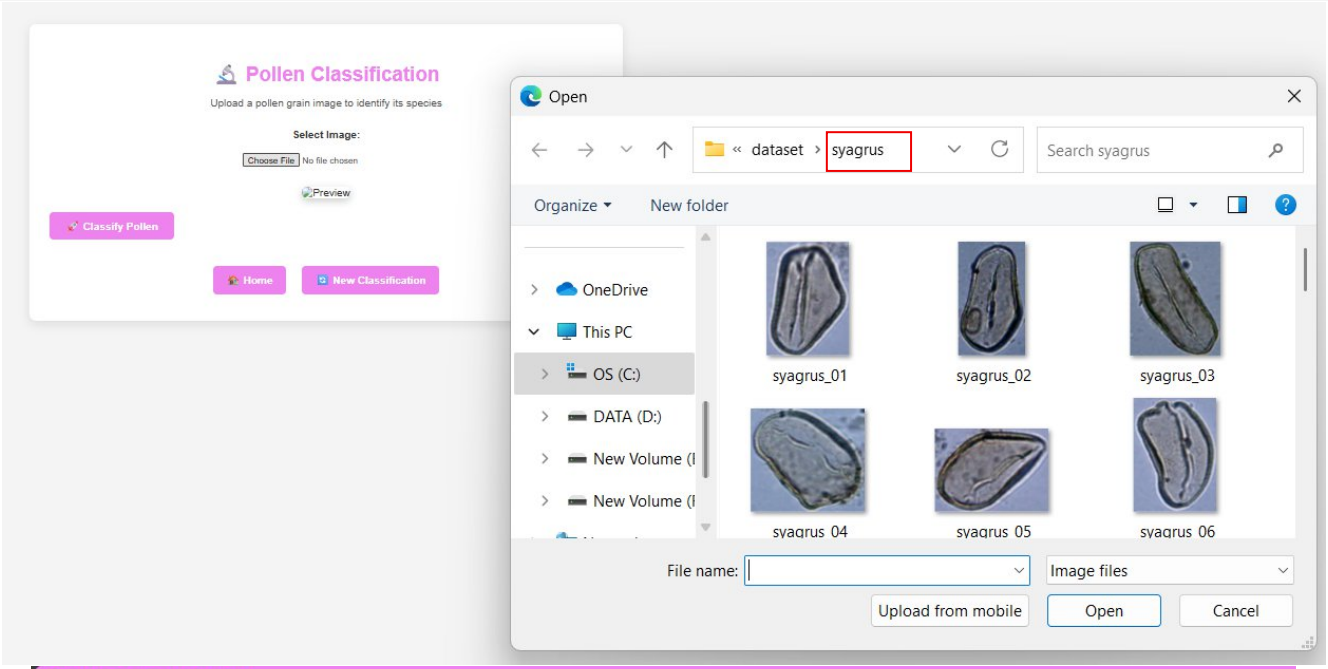
app.py:

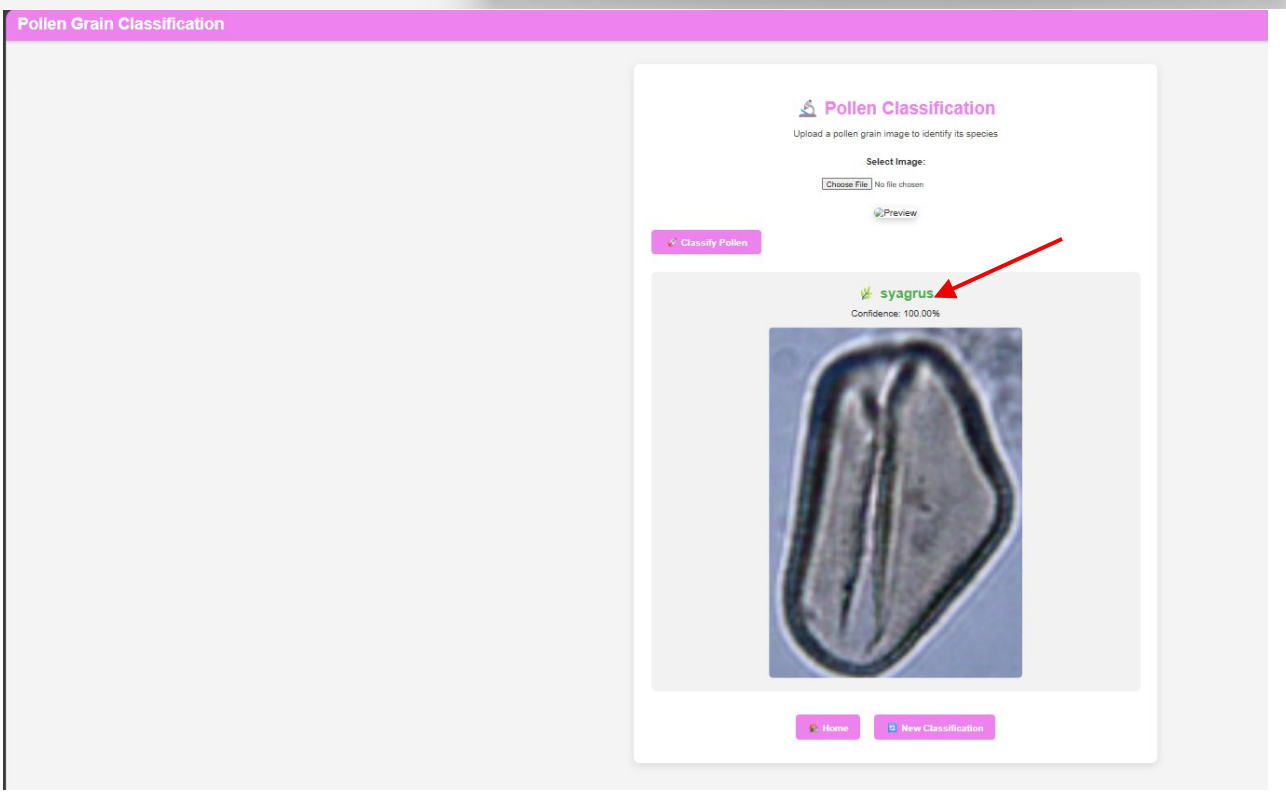
```
import os
import numpy as np
from flask import Flask, request, render_template
from keras.preprocessing import image
from keras.models import load_model

# Initialize Flask app
app = Flask(__name__)

# Load your trained Keras model
model = load_model("model.h5", compile=False)

# List of class labels (ensure it matches your model's output order)
labels = [
    'anadenanthera', 'arecaceae', 'arrabidaea', 'cecropia', 'chromolaena',
    'combretum', 'croton', 'dipteryx', 'eucalipto', 'faramea', 'hyptis',
    'mabea', 'matayba', 'mimosa', 'myrcia', 'protium', 'qualea', 'schinus',
    'senegalia', 'serjania', 'syagrus', 'tridax', 'urochloa'
]

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

# Prediction route
@app.route('/predict', methods=['GET', 'POST'])
def predict():
    if request.method == 'POST':
        file = request.files['image']
        if not file:
            return render_template('predict.html', prediction=None, image_path=None, confidence=None)
```

```
        # Save uploaded file to static/uploads directory
        basepath = os.path.dirname(__file__)
        upload_folder = os.path.join(basepath, 'static', 'uploads')
        os.makedirs(upload_folder, exist_ok=True)
        upload_path = os.path.join(upload_folder, file.filename)
        file.save(upload_path)

        # Preprocess the image
        img = image.load_img(upload_path, target_size=(128, 128))
        x = image.img_to_array(img)
        x = np.expand_dims(x, axis=0)

        # Predict
        probs = model.predict(x)[0]
        pred = np.argmax(probs)
        predicted_label = labels[pred]
        confidence = float(probs[pred] * 100)

        return render_template('predict.html',
                               prediction=predicted_label,
                               image_path=file.filename,
                               confidence=confidence)

    return render_template('predict.html', prediction=None, image_path=None, confidence=None)

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

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

🌐 **10. Flask Web Application execution**


Pollen Grain Classification
About
Contact
Prediction
Pollen Grain Prediction
We assist researchers and scientists in the field of botany and plant sciences by providing an interface to upload images of pollen grains and predict their type.
ABOUT US
Pollen's Profiling: Automated Classification Of Pollen Grains
Team ID: LTVIP2025TMID45577
Team Size: 4
Team Leader: P.Pooja Harshitha
Team Member: Navya Ramiselly
Team Member: Narepalem Teja
Team Member: Naragam Manikanta Raghava
© 2025 Pollen Grain Classification. All rights reserved.


Pollen Classification
Upload a pollen grain image to identify its species
Select Image:
Choose File
No file chosen
Preview
Classify Pollen
Home
New Classification
Open
dataset
syagrus
Search syagrus
Organize
New folder
OneDrive
This PC
OS (C:)
DATA (D:)
New Volume (I
New Volume (I
syagrus_01
syagrus_02
syagrus_03
syagrus_04
syagrus_05
syagrus_06
File name:
Image files
Upload from mobile
Open
Cancel


Pollen Grain Classification
Pollen Classification
Upload a pollen grain image to identify its species
Select Image:
Choose File
No file chosen
Preview
Classify Pollen
syagrus
Confidence: 100.00%
Home
New Classification