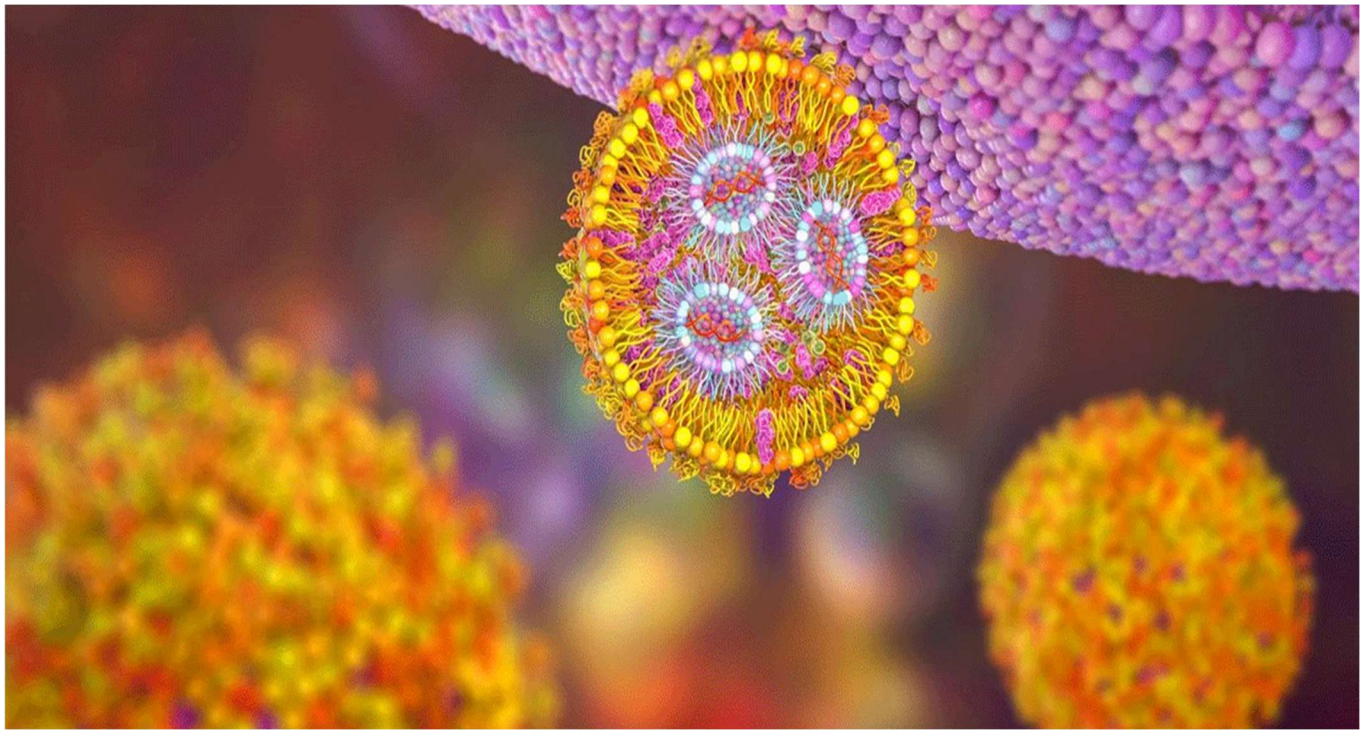
POLLEN GRAIN CLASSIFICATION USING DEEP LEARNING

Pollen's Profiling: Automated Classification of Pollen Grains



Team Id: LTVIP2025TMID45378

Team Size: 3

Team Leader: S Lalitha Nandini

Team member**:** Tammineni Akhil Kumar

Team member: Tamarakolanu Raja Rajeswari

ABOUT THE PROJECT

PROJECT DESCRIPTION

Automated Classification of Pollen Grains Using CNN

Pollen grain classification is a critical task in several scientific domains, including botany, allergy diagnostics, forensic investigations, and climate change research. Traditionally, the classification process is carried out manually by experts using microscopes, which is labor-intensive, time-consuming, and prone to human error. To address these challenges, this project proposes an end-to-end solution that leverages image processing and deep learning, specifically Convolutional Neural Networks (CNNs), to automate the classification of pollen grains. The ultimate goal is to build a scalable and accurate model that can differentiate between various types of pollen grains based on their microscopic images.

The project begins with data collection, where a dataset containing high-resolution images of different pollen grain types is compiled. This can be obtained from public datasets such as the Pollen Dataset through the Web Services like Google and other Browsing platforms. The images are labeled according to the pollen type they represent. The next step involves Exploratory Data Analysis (EDA), which includes visualizing sample images, understanding class distributions, analyzing image dimensions, and identifying any class imbalance or image quality issues. Preprocessing techniques such as resizing, normalization, grayscale conversion, and data augmentation (rotation, zoom, shift) are applied to enhance the quality and variety of input images, ensuring the model generalizes well.

For the image classification task, a CNN-based architecture is employed due to its proven effectiveness in handling visual data. The model typically consists of multiple layers—convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification. After training the CNN on the preprocessed image dataset, the model's performance is evaluated using metrics like accuracy, precision, recall, and F1-score. A confusion matrix is also generated to visualize

classification errors across different pollen classes. Finally, techniques such as hyperparameter tuning, dropout regularization, and fine-tuning of pre-trained models (like VGG16 or ResNet) are used to further optimize performance. The result is a robust pollen classification system that significantly reduces manual effort and improves accuracy and scalability in real-world applications.

BRAIN STROMING AND IDEATION

Key Points:

Background & Context:

Pollen grains are microscopic particles that serve as the male reproductive unit in flowering plants. Studying them has long been essential in a variety of disciplines— botany, environmental science, forensic science, paleoclimatology, and agriculture. Accurate identification and classification of pollen grains can help:

• Understand plant biodiversity and geographical distributions.

• Monitor environmental changes and allergens.

• Study historical climates via palynology (the study of fossilized pollen).

• Enhance crop breeding and disease resistance strategies. However, current methodologies for pollen grain classification are largely manual. Trained experts analyze pollen morphology under microscopes, compare structures with reference charts, and classify accordingly. This manual process is slow, labor-intensive, and prone to human error, especially when dealing with large sample sizes or subtle morphological differences between species

Problem Statement:

Pollen grains, though microscopic, play a vital role in plant reproduction and environmental studies. Their classification is essential for fields such as botany, allergy research, climate change monitoring, and even forensic science. However, manual identification of pollen types under a microscope is tedious, error-prone, and requires deep expertise. This project aims to automate the classification of pollen grains using advanced image processing and machine learning techniques, enabling fast, accurate, and scalable identification across various species.

Key issues include:

• Subjectivity in identification: Different observers might classify the same pollen grain differently based on their judgment.

• Time inefficiency: Experts may take several minutes to analyze a single grain, making large-scale studies difficult.

• Inconsistent results: Fatigue, variations in lighting or magnification, and differences in training can affect the accuracy of results.

Purpose And Impact of the Project:

The project begins with data collection, where a dataset containing high-resolution images of different pollen grain types is compiled. This can be obtained from public datasets such as the Pollen Dataset through the Web Services like Google and other Browsing platforms. The images are labelled according to the pollen type they represent. The next step involves Exploratory Data Analysis (EDA), which includes visualizing sample images, understanding class distributions, analyzing image dimensions, and identifying any class imbalance or image quality issues. Preprocessing techniques such as resizing, normalization, grayscale conversion, and data augmentation (rotation, zoom, shift) are applied to enhance the quality and variety of input images, ensuring the model generalizes well.

For the image classification task, a CNN-based architecture is employed due to its proven effectiveness in handling visual data. The model typically consists of multiple layers—convolutional layers for feature extraction, pooling layers for dimensionality reduction, and fully connected layers for classification. After training the CNN on the preprocessed image dataset, the model's performance is evaluated using metrics like accuracy, precision, recall, and F1-score. A confusion matrix is also generated to visualize classification errors across different pollen classes. Finally, techniques such as hyperparameter tuning, dropout regularization, and fine-tuning of pre-trained models (like VGG16 or ResNet) are used to further optimize performance. The result is a robust pollen classification system that significantly reduces manual effort and improves accuracy and scalability in real-world applications.

Target Users :

1. Botanists and Palynologists (Pollen Scientists).
2. Allergists and Healthcare Professionals.
3. Environmental Scientists and Climate Researchers.
4. Agricultural Scientists and Crop Managers.
5. Forensic Investigators.
6. Archaeologists and Paleobotanists.

Expected outcome:

1. A well-optimized Convolutional Neural Network (CNN) model capable of classifying different types of pollen grains based on microscopic images.

2. The model will achieve high accuracy (typically >85% depending on dataset quality) and demonstrate strong performance on key metrics such as precision, recall, and F1-score.

3. The system will contribute to faster, more accurate, and less labor-intensive pollen grain analysis.

4. It will enhance productivity in botany, allergy studies, environmental science, forensics, and other fields.

5. Long-term, the model may support real-time airborne pollen monitoring, historical vegetation mapping, or allergy forecasting systems.

6. The system will contribute to faster, more accurate, and less labor-intensive pollen grain analysis.

7. It will enhance productivity in botany, allergy studies, environmental science, forensics, and other fields.

8. Long-term, the model may support real-time airborne pollen monitoring, historical vegetation mapping, or allergy forecasting systems.

REQUIREMENT ANALYSIS

Objective:

The purpose of the Requirement Analysis phase is to define and understand both the technical and functional aspects necessary for building the pollen grain classification system. This phase ensures that all the resources, constraints, and expectations are clearly identified before development begins. It bridges the gap between the conceptual understanding from Phase 1 and the actual implementation to follow in Phase 3. Through in-depth analysis, the team aims to avoid scope creep, manage risks, and ensure the solution is technically feasible and aligned with stakeholder needs.

Key Points:

Understanding the System Requirements:

To successfully develop an automated pollen grain classification system using deep learning, it is essential to establish a comprehensive understanding of:

1. Technical Requirements – The software, hardware, tools, and frameworks required for successful model development and deployment.

2. Functional Requirements – The core functionalities the system must support to be valuable and effective to end-users.

3. Non-Functional Requirements – Performance, usability, and scalability expectations.

4. Constraints & Risks – Limitations and potential challenges that must be addressed or planned for during development

Technology Stack :

Python, OpenCV (for image preprocessing and enhancement, Convolutional Neural Networks (CNNS)), TensorFlow or Pytorch (for deep learning model development), Scikit-learn (for classical ML comparisons or model evaluation), Matplotlib/Seaborn (for visualization), Microscopic Pollen Grain Image Dataset (e.g., from Kaggle or academic sources).

Use Cases:

1. Assisting doctors in early, non-invasive diagnosis of liver cirrhosis.
2. Supporting hospital decision systems to prioritize high-risk patients.
3. Empowering telemedicine platforms with smart liver health assessments.

Functional Requirements:

1. Data related features:

These features support how input images and associated metadata are gathered and managed:

a. Labelled Pollen Image Dataset:

- A curated dataset with images of pollen grains where each image is tagged with its respective class label.
- Enables supervised learning.

b. Image Metadata:

- Includes auxiliary details such as:
    - Image dimensions
    - Date captured
    - Microscope magnification
    - Staining method
- Useful for filtering, preprocessing, and EDA.

2. Preprocessing Features:

Prepares data for optimal training and generalization:

a. Image Resizing and Normalization

- Resize images to a fixed input shape.
- Normalize pixel values (e.g., scale 0–255 to 0–1 or mean-subtraction for zero-centering).

b. Data Augmentation Techniques

- Apply transformations to artificially expand dataset:
    - Rotation, flipping
    - Zoom, shift, brightness adjustment
    - Random noise injection

c. Noise Reduction and Background Removal

- Techniques like:
    - Gaussian blur or median filtering for noise

- Thresholding or edge-based segmentation for background removal

- Helps improve model focus on the grain.

3. Model Development Features:

Core components for building and training the classifier:

a. CNN Architecture

- A Convolutional Neural Network designed for image classification:
  - Layers: Conv2D → ReLU → MaxPooling → Dropout → Dense
  - May use pre-trained models (e.g., VGG16, ResNet) for transfer learning.

b. Training Configuration

- Parameters and setup:
  - Loss function (e.g., categorical_crossentropy)
  - Optimizer (e.g., Adam, SGD)
  - Batch size, learning rate, number of epochs
  - Early stopping criteria

c. Model Checkpointing

- Save model weights at intervals or best validation accuracy.
- Allows recovery and continued training from saved checkpoints.

---

4. Evolution and visualization tools:

Tools for model assessment and interpretability:

a. Confusion Matrix – To visualize correct vs. incorrect predictions per class.
b. Accuracy & Loss Curves – To monitor overfitting/underfitting during training.
c. Classification Report – Includes precision, recall, F1-score.
d. Sample Predictions – Visual display of model predictions with actual labels.
e. Grad-CAM or Heatmaps – Highlight areas of image influencing decisions

Constraints and Challenges:

1. Data-Related Limitations

- Limited or Imbalanced Dataset:
  High-quality, labelled pollen datasets may be limited or imbalanced (some pollen types have far fewer images), which can lead to biased predictions and poor performance on underrepresented classes.
- Variability in Image Quality:
  Differences in image resolution, magnification, lighting conditions, and background noise across microscopy images can negatively impact feature extraction by the CNN.

2. Model-Related Limitations

- Overfitting:
  CNNs can easily overfit on small datasets, performing well on training data-poorly on unseen data. This is especially risky if data augmentation or regularization techniques are not applied properly.
- Poor Generalization:
  The model trained on a specific dataset might not generalize well to pollen images from different microscopes, labs, or environments without additional domain adaptation or retraining.
- Difficulty in Distinguishing Similar Pollen Types:
  Some pollen grains have very similar shapes and textures, making it hard for the model to distinguish them without additional features (like 3D structure or chemical properties).

3. Human and Ethical Risks:

- Misclassification Risks:
  Incorrect classifications can lead to wrong conclusions in scientific studies or public health responses (e.g., misidentifying allergenic pollen types).
- Dependence on Automation:
  Overreliance on AI without expert oversight can lead to reduced vigilance or critical thinking, especially in sensitive fields like forensics or medical diagnostics.

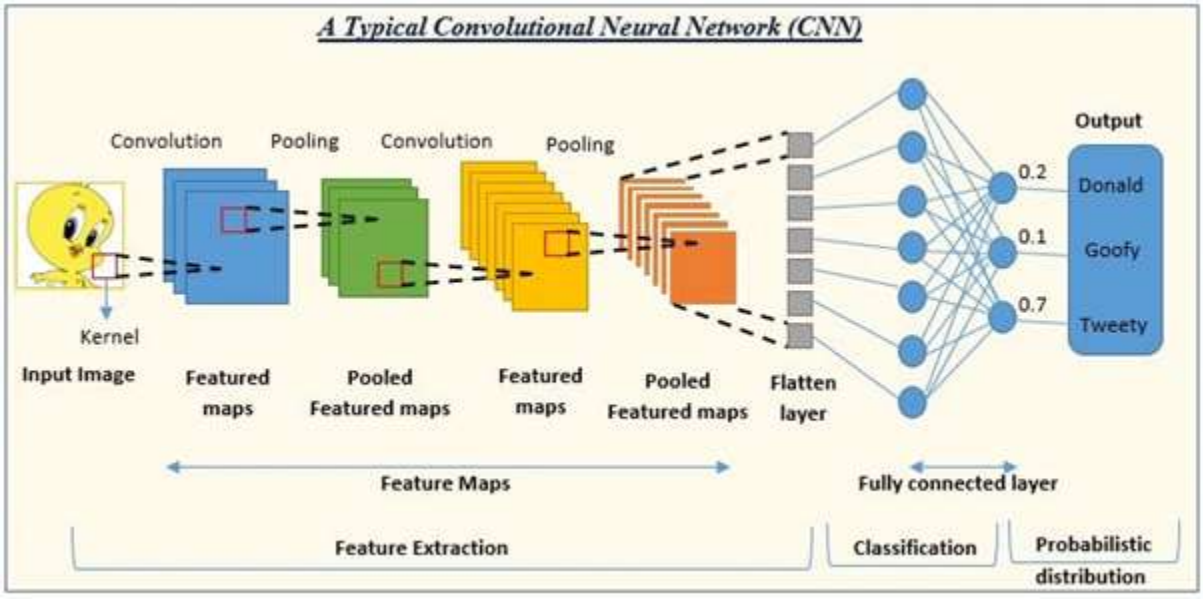
PROJECT DESIGN

Objective:

The objective of the Project Design phase is to structure the system architecture, define how the application components will interact, and design the user experience to ensure seamless usability. It translates abstract requirements into concrete design decisions, diagrams, workflows, and interface mockups. A well-structured design ensures that the system is scalable, maintainable, user-friendly, and technically robust. This phase is a bridge between analysis and implementation, giving the development team a clear map to follow during coding.

Convolutional Neural Networks:

Convolutional Neural Networks (CNNs) are a deep learning architecture designed to automatically and adaptively learn spatial hierarchies of features from input images. They are particularly effective for image recognition, object detection, and other tasks involving visual data. CNNs utilize convolutional layers, pooling layers, and fully connected layers to extract and classify features from input images.



Key Features of CNNs:

1. **Convolutional Layers:**
   - These layers apply filters to input data to extract features like edges, textures, shapes.
   - The filters move (convolve) across the image, capturing local patterns.
2. **ReLU Activation (Rectified Linear Unit):**

- Introduces non-linearity to the network.
- Ensures the network can learn complex patterns by applying f(x) = max(0, x).

3. **Pooling Layers:**
   - Reduce the spatial size (width & height) of the feature maps.
   - Common pooling methods: Max Pooling, Average Pooling.
   - Helps reduce computation and control overfitting.
4. **Flattening Layer:**
   - Converts 2D feature maps into a 1D vector.
   - Prepares the data for fully connected (dense) layers.
5. **Fully Connected Layers (Dense Layers):**
   - Perform high-level reasoning.
   - Typically used at the end for classification tasks.
6. **Parameter Sharing:**
   - Filters are shared across the input, reducing the number of parameters significantly.
7. **Translation Invariance:**
   - CNNs can recognize objects in different positions within the image due to shared weights and local receptive fields.
8. **Hierarchical Feature Learning:**
   - Lower layers detect simple features (edges), while higher layers detect complex patterns (faces, objects).
9. **Backpropagation and Gradient Descent:**
   - Used to train the network by updating weights based on loss/error.

System Architecture:

User → Web Interface → Flask Backend → Trained CNN Model → Output Prediction

1. User:

The end-user interacts with the system via a web application. They can upload an image of a pollen grain and initiate prediction.

2. Web Interface (Frontend):

The frontend is built using HTML5, CSS3, and optionally JavaScript for interactivity. It offers:

- File upload field
- Predict button
- Error/warning messages

- Result display area

3. Flask Backend:

This lightweight server-side application handles requests from the user interface. It processes uploaded files, passes them to the trained model, collects predictions, and sends them back to the frontend. Flask is ideal because of its minimal overhead, support for routing, easy integration with machine learning models, and RESTful architecture.

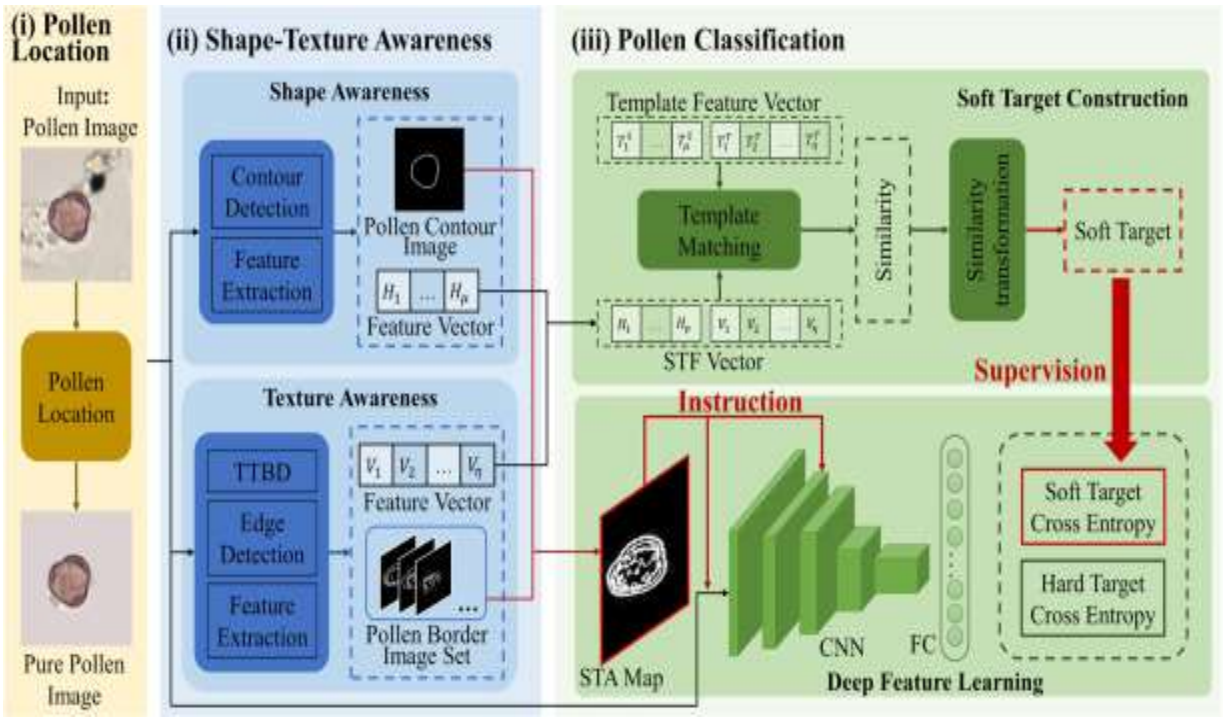
4. Trained CNN Model:

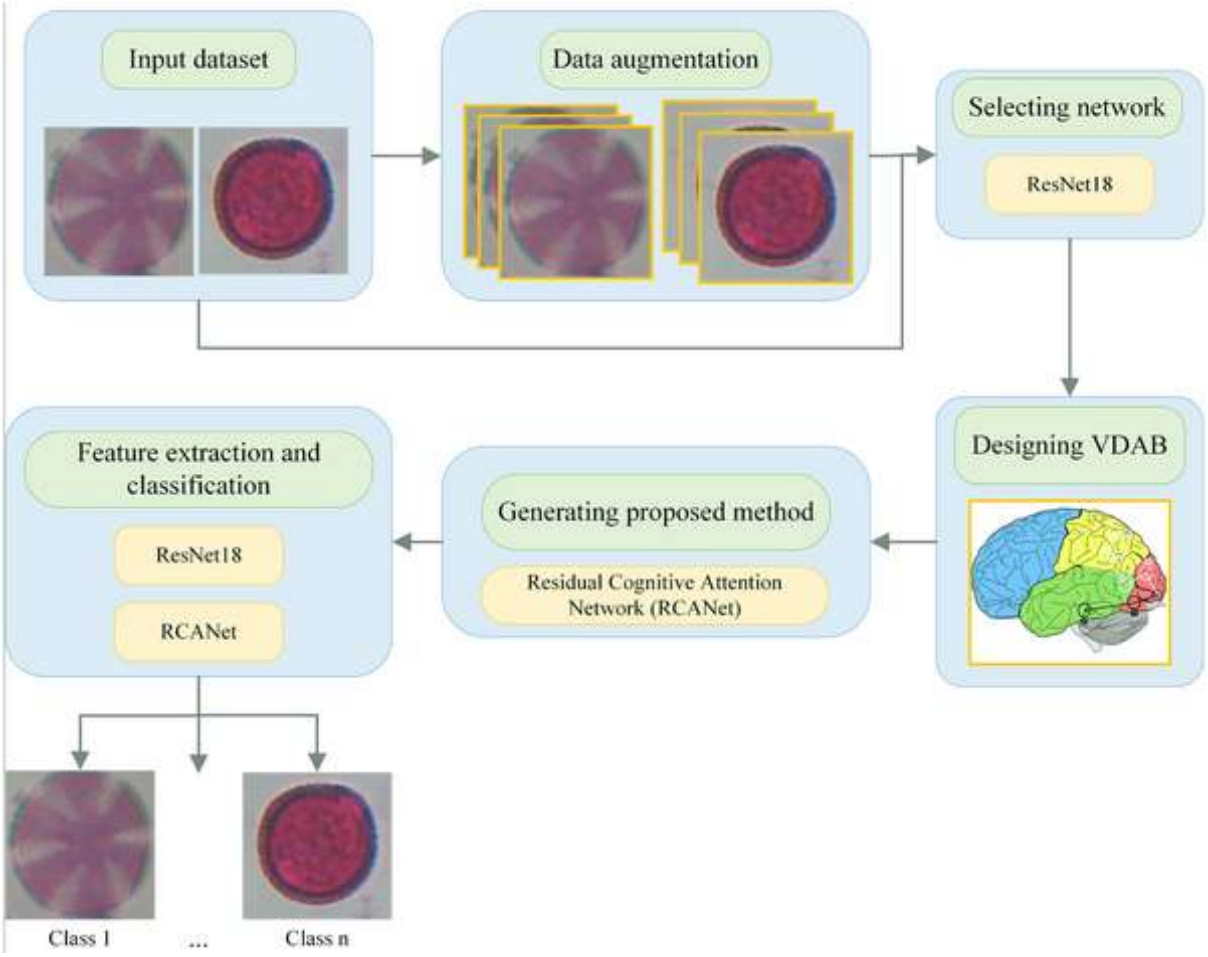
A Convolutional Neural Network (CNN) trained using TensorFlow/Keras is the intelligence behind the system. Once loaded into memory at server start, it takes input images and returns predictions quickly. It may be stored in formats like .h5 or .tflite for deployment.

5. Prediction Output:

The model's output (predicted class/species name and optionally confidence level) is sent to the Flask app, which renders the result in the frontend.

Flow Chart:





Data Flow Design:

The data flow in the application is as follows:

1. Input: User uploads a pollen image through the web interface.
2. Validation: The backend checks the file type and size.
3. Preprocessing: Image is resized, normalized, and reshaped to match model input dimensions.
4. Prediction: The CNN model processes the image and predicts its class.
5. Output: The result is formatted and sent back to the frontend for display.

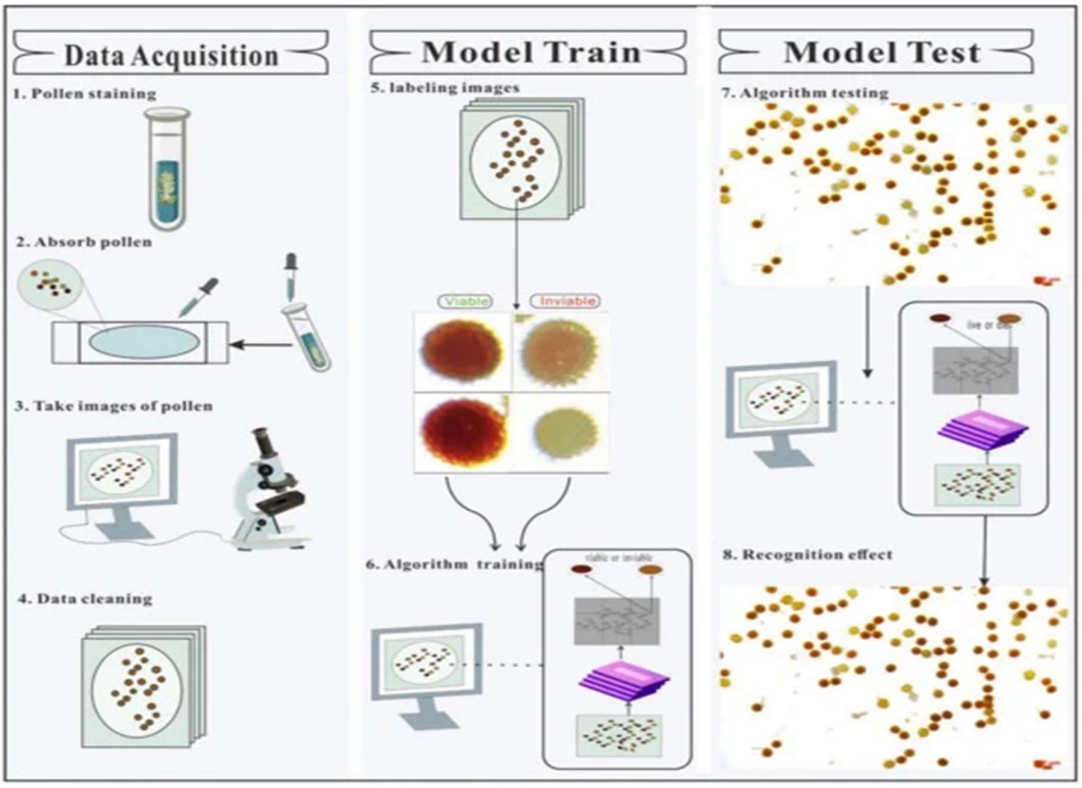
This flow is designed to ensure speed, accuracy, and robustness in handling various types of inputs and usage scenarios.

Work Flow:

1. Data Acquisition:

This stage involves preparing and collecting data (images of pollen) for training the model.

1. **Pollen Staining**:
   Pollen grains are stained using a chemical dye to differentiate viable from non-viable pollen.
2. **Absorb Pollen**:
   A pipette is used to transfer the stained pollen onto a glass slide.
3. **Take Images of Pollen**:
   A microscope is used to capture images of the stained pollen on the slide.
4. **Data Cleaning**:
   Unwanted or poor-quality images are removed to ensure that the dataset is clean and consistent for training.



2. Model Train:

This stage involves labeling and training a CNN model.

1. **Labeling Images**:
   The collected images are annotated or labeled as “Viable” or “Inviable” based on appearance.
2. **Algorithm Training**:
   A Convolutional Neural Network is trained using the labeled images to learn patterns that distinguish between viable and non-viable pollen grains.

3.Model Test:

This stage evaluates the model on new, unseen data.

1. **Algorithm Testing**:
   The trained model is tested with a new set of pollen images to assess its performance.
2. **Recognition Effect**:
   The model successfully classifies the pollen as viable or non-viable and displays the results.

PROJECT PLANNING

Project Title: Pollen Profiling: Automated Classification of Pollen Grains using CNN

1. Importing Necessary Modules:

Module	**Purpose**
Numpy	Image and data array handling.
Pandas	Optional for tabular data handling.
matplotlib.pyplot	Plotting accuracy/loss curves.

Module	Purpose
Seaborn	Confusion matrix heatmap.
Os	File system navigation.
cv2	Image resizing, color conversion (manual preprocessing).
tensorflow.keras	Building and training the CNN model.
sklearn. metrics	Evaluation – confusion matrix and classification report.
sklearn.model_selection	Optional manual train-test splitting.

2. Data Collection and Assigning Labels:

https://www.kaggle.com/code/killa92/pollengrain-classification-98-accuracy-pytorch/input

- **Data Collection**:
  - Collect microscopic images of different pollen grains.
  - Dataset can be obtained from open-source pollen datasets or captured using lab equipment.
- **Labelling**:
  - Assign a label to each pollen image (e.g., "Pine", "Oak", "Birch").
  - Folder structure format:
- **Preprocessing**:
  - Resize all images to the same size (e.g., 128x128 pixels).
  - Normalize pixel values.

3. Splitting the Data into Training and Testing:

Splitting data into **training** and **testing** sets is a key step in building machine

Learning models. Here's a brief explanation:

- **Training Data**: This subset is used to train the model. The model learns patterns, relationships, and structures in the data.

- **Testing Data**: This subset is kept separate and is used to evaluate how well the trained model performs on unseen data. It helps assess the model's **generalization ability**.

Typical Split Ratio

- Common splits are **80/20**, **70/30**, or **75/25** (training/testing), depending on the size of the dataset.

4. Building the CNN model / Model Selection:

1. Conv2D (32, (3,3), activation = 'relu')

> Applies 32 filters of size 3x3 to the input image.

> Extracts low-level features like edges and textures.

> ReLU activation adds non-linearity.

2.MaxPooling2D (2,2)

> Reduces the spatial size (down sampling).

> Keeps dominant features and reduces computation.

3. Conv2D (64, (3,3), activation = 'relu')

> Applies 64 filters of size 3x3.

> Learns more complex features (shapes, patterns).

4. MaxPooling2D(2,2)

> Further reduces dimensionality and keeps important features.

5. Flatten ()

> Converts the 2D feature maps into a 1D vector to feed into dense layers.

6. Dense (128, activation = 'relu')

> Fully connected layer with 128 neurons.

> Learns non-linear combinations of extracted features.

7. Drop (0.5)

> Randomly drops 50% of neurons during training to prevent overfitting.

8. Dense (train_generator.num_classes, activation = 'SoftMax')

> Final output layer with one neuron for each pollen class.

> SoftMax turns outputs into probabilities for multi-class classification.

5. Compile and Train the Model:

1. Optimizer – 'adam'

> Adam (Adaptive Moment Estimation) is a popular and efficient optimizer.

> It adjusts the learning rate dynamically during training.

> Helps in faster and more stable convergence, especially useful for image data.

2.Loss Function – 'categorical_crossentropy'

> This is used for multi-class classification where each image belongs to one of classes.

> It measures how well the predicted class probabilities match the true labels.

3. Metrics – 'accuracy'

> Accuracy is used to track how often the model's predictions are correct.

> This is useful for evaluating the model's performance during training and validation.

6. Evaluate the model:

Purpose of This Step

> To determine how well the trained CNN model generalizes to new, unseen data.

> Helps in identifying overfitting or underfitting:

> High training accuracy but low validation accuracy = Overfitting.

> Low accuracy overall = Underfitting (model too simple or not trained enough).

Expected Outcome

We get a quantitative performance measure of your model, which tells you if it's ready for testing on real-world pollen grain images or needs improvement.

7. Confusion Matrix and Classification Report:

After evaluating the model's overall accuracy, it's important to analyze its detailed behavior — how well it performs per class. This is where the confusion matrix and classification report come in.

Key Components of This Step:

1. **y_pred = model. Predict (Val generator)**

   > The model makes predictions on all validation images.

   > It returns a probability vector for each image (e.g., [0.1, 0.8, 0.1]).

2. **y_pred_classes = np.argmax(y_pred, axis=1)**

   > Converts the predicted probability vectors into actual class labels.

3. **y_true = val_generator.classes**

   > These are the true labels for the validation images.

Classification Report:

print (classification_report(y_true, y_pred_classes, target_names=...))

✧ This generates a detailed report including:

Precision: How many of the predicted positives were correct.

Recall: How many actual positives were correctly predicted.

F1-Score: Harmonic mean of precision and recall.

Support: Number of actual samples.

Confusion Matrix:

conf_matrix = confusion_matrix(y_true, y_pred_classes)

sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='g')

✧ The confusion matrix shows how often each class is correctly predicted vs. misclassified.

Diagonal values: Correct predictions.

Off-diagonal values: Misclassifications.

The heatmap visually highlights strengths and weaknesses.

8. Visualization – Accuracy and Loss Curves:

- **Purpose of This Step**

The goal of this step is to visually monitor how the model is learning during training. By plotting accuracy and loss curves, we can understand:

> How well the model is fitting the training data.

> Whether the model is overfitting, underfitting, or learning effectively.

> When to stop training or tune hyperparameters.

Two Main Plots:

1. Training vs. Validation Accuracy Curve

Shows how the model's classification accuracy improves over each epoch.

Helps determine:

> If the model is learning effectively.

> If the validation accuracy plateaus or drops, indicating overfitting.

2. Training vs. Validation Loss Curve

Displays how the model's error (loss) changes over time.

Helps identify:

> Whether the model is minimizing the error consistently.

> If the validation loss increases while training loss decreases → overfitting.

9: Future Scope:

1. Larger and More Diverse Dataset The model can be improved by using a larger dataset with more pollen grain types from different regions and seasons. This will make the model more general and accurate.

2. Real-Time Classification System This project can be developed into a mobile or web application that allows users to upload images and get instant classification results.

3. Integration with Microscopes The model can be connected to digital microscopes for real-time automatic pollen grain analysis during sample observation.

4. Multi-Feature Analysis In addition to images, features like accurate predictions.

5. size, shape, and texture Transfer Learning and Advanced Models Using more powerful pre-trained models like improve performance and reduce training time.

6. Application in Allergy Forecasting can be added for more ResNet, VGG, or EfficientNet can By identifying pollen types in air samples, the model can help in predicting and preventing pollen allergy outbreaks.

7. Use in Forensic and Climate Studies Pollen analysis can help in crime scene investigation and studying climate change. Your model can be used as a base tool for such research.

10: CONCLUSION:

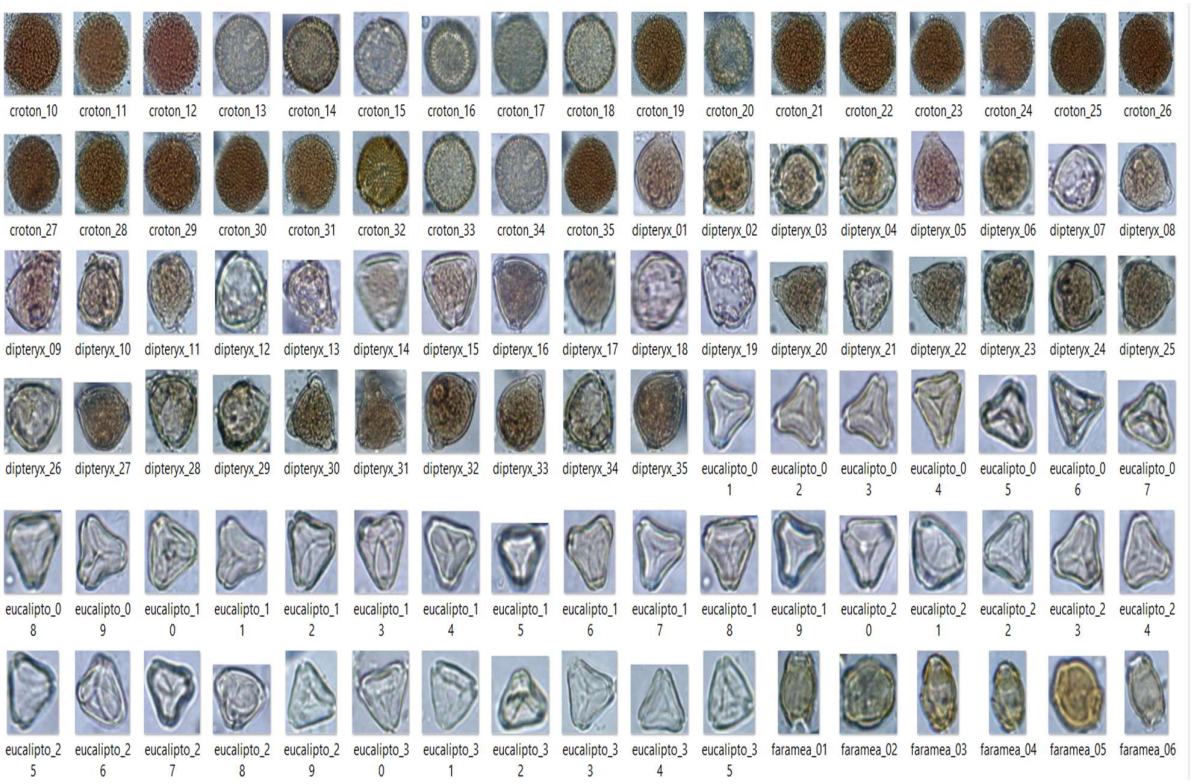
The Pollen Grain Classification project successfully demonstrates how deep learning techniques, especially Convolutional Neural Networks (CNNs), can be used to accurately classify different types of pollen grains from images. This system provides a fast, reliable, and automated method that reduces the need for manual identification, which is usually time-consuming and error-prone. Through this project, we learned the complete process of building an image classification model — from data preprocessing, model building, training, testing, and performance evaluation. The results show that deep learning can significantly improve classification accuracy in biological image analysis. Overall, this project has potential real-world applications in botany, agriculture, environmental studies, and medical research, and can be expanded further with a larger dataset and more advanced models.

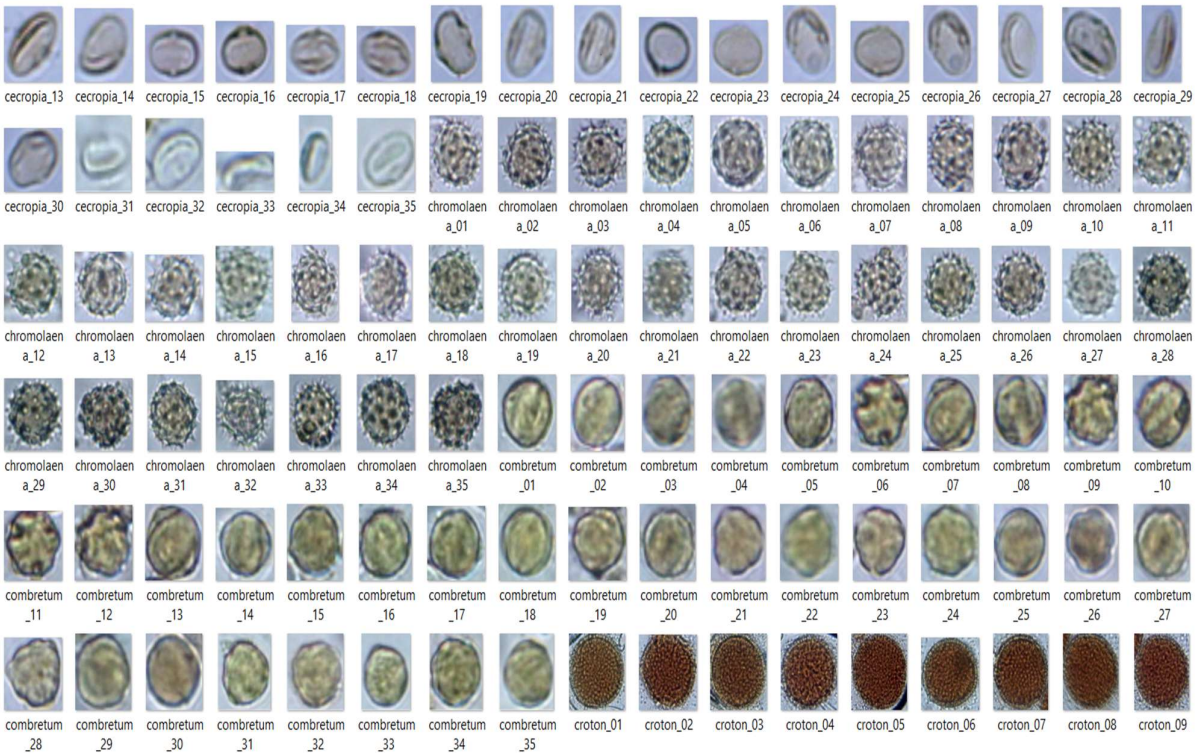
PROGRAM DEVELOPMENT

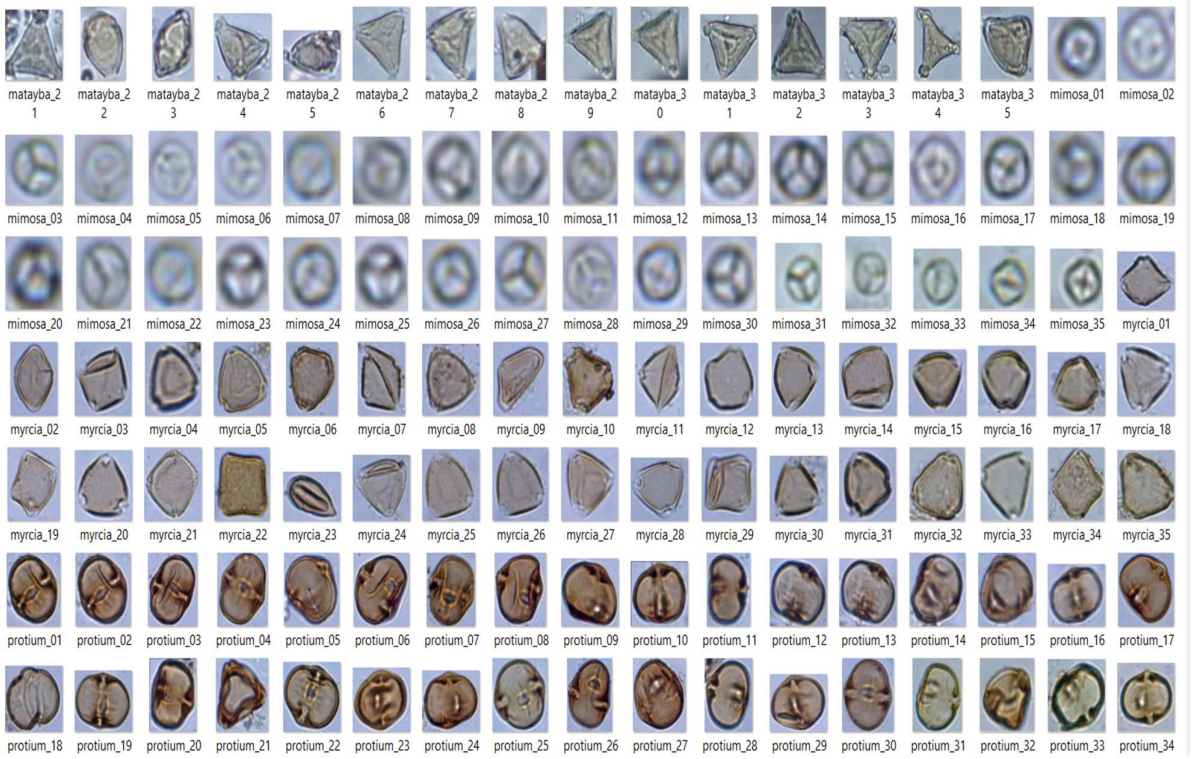
```
[ ]: import zipfile
     import os

     # Path to the dataset
     zip_path = r"C:\IMP\pollen_grains\data\archive.zip"
     extract_path = r"C:\IMP\pollen_grains\data\pollen_data"

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

     print("  Dataset extracted to:", extract_path)
```

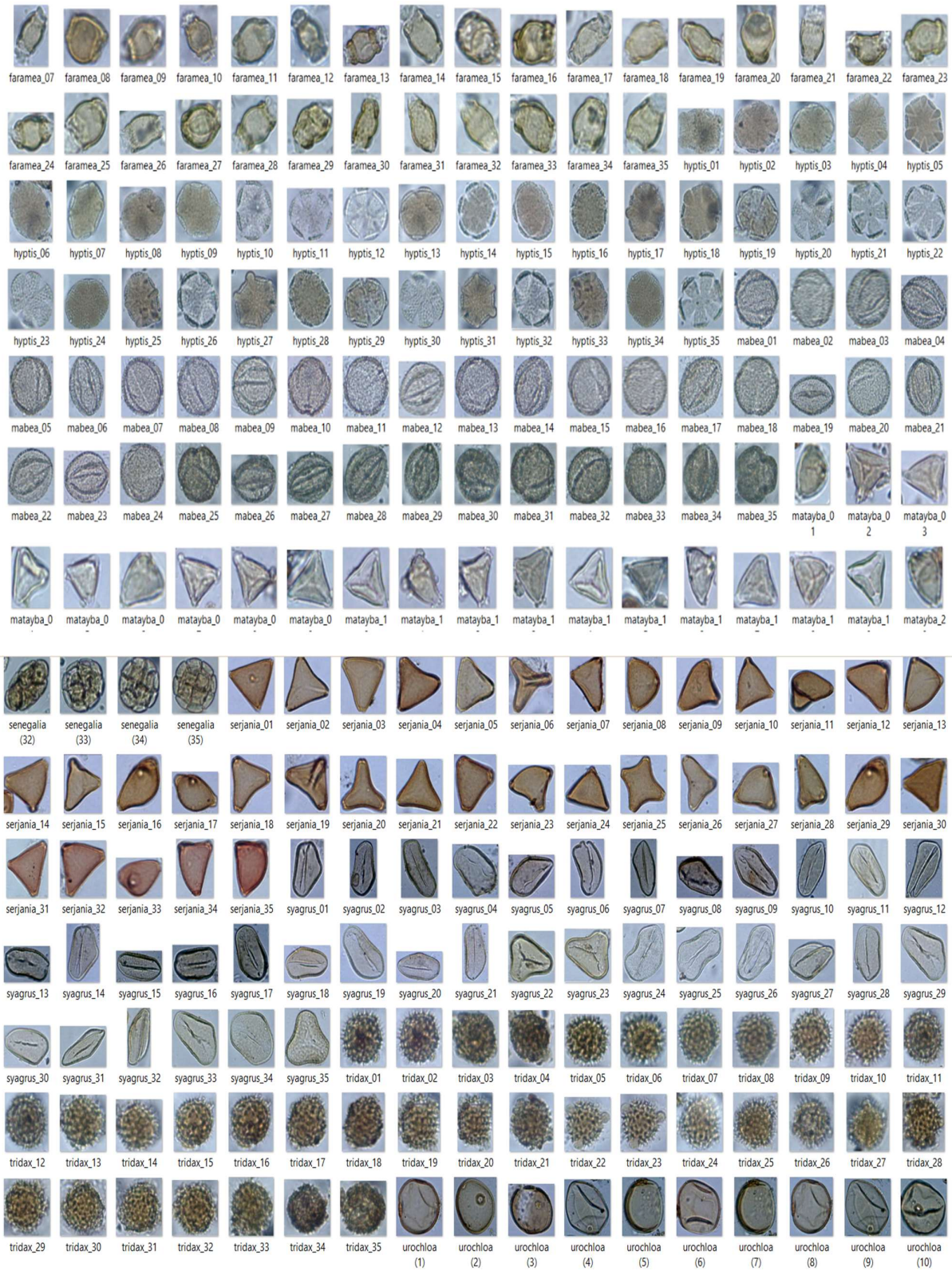
```
[2]: import os
     from collections import Counter
     import matplotlib.pyplot as plt
     from PIL import Image
```

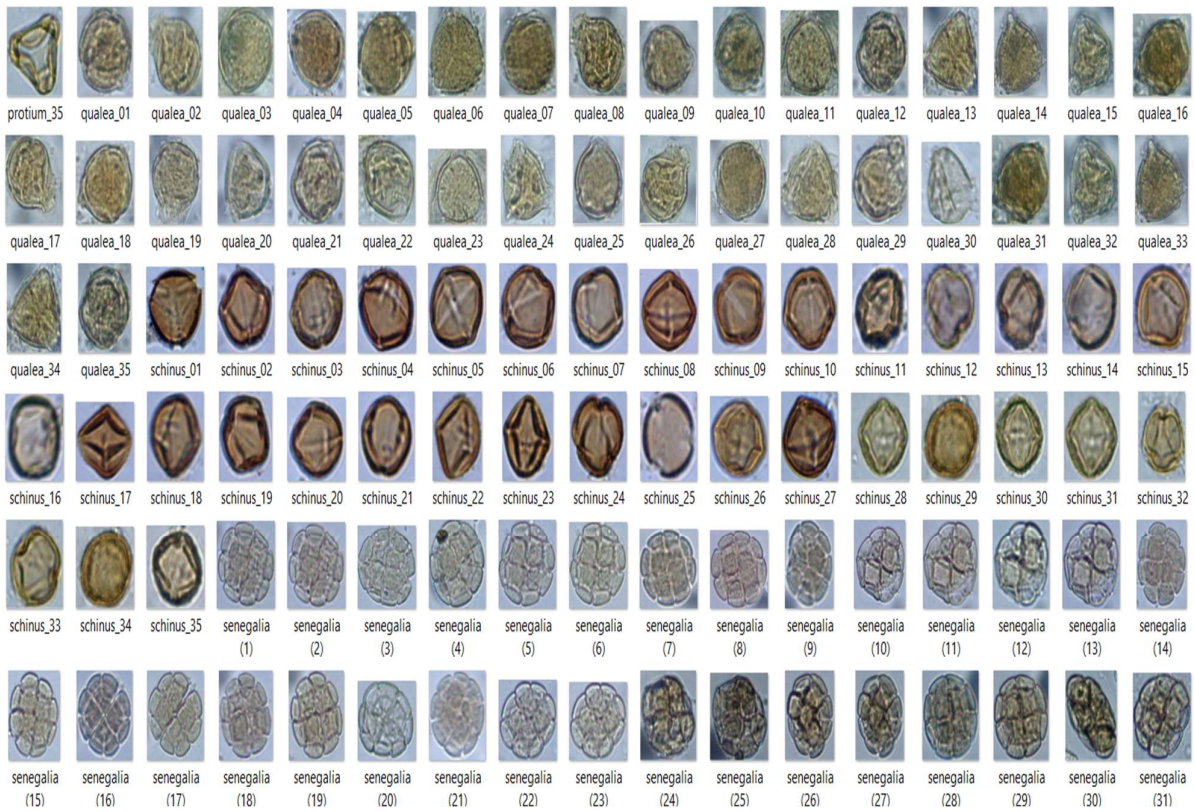
ACQUIRED DATASET IS: https://www.kaggle.com/code/killa92/pollengrain-classification-98-accuracy-pytorch/input

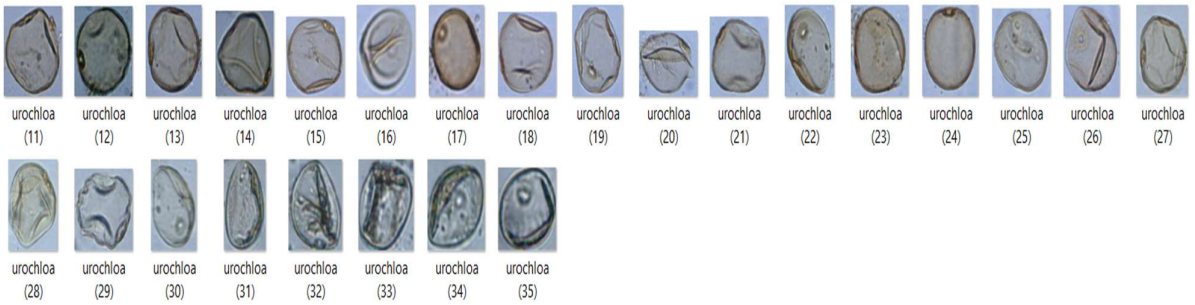



cecropia_13
cecropia_14
cecropia_15
cecropia_16
cecropia_17
cecropia_18
cecropia_19
cecropia_20
cecropia_21
cecropia_22
cecropia_23
cecropia_24
cecropia_25
cecropia_26
cecropia_27
cecropia_28
cecropia_29
cecropia_30
cecropia_31
cecropia_32
cecropia_33
cecropia_34
cecropia_35
chromolaena_01
chromolaena_02
chromolaena_03
chromolaena_04
chromolaena_05
chromolaena_06
chromolaena_07
chromolaena_08
chromolaena_09
chromolaena_10
chromolaena_11
chromolaena_12
chromolaena_13
chromolaena_14
chromolaena_15
chromolaena_16
chromolaena_17
chromolaena_18
chromolaena_19
chromolaena_20
chromolaena_21
chromolaena_22
chromolaena_23
chromolaena_24
chromolaena_25
chromolaena_26
chromolaena_27
chromolaena_28
chromolaena_29
chromolaena_30
chromolaena_31
chromolaena_32
chromolaena_33
chromolaena_34
chromolaena_35
combretum_01
combretum_02
combretum_03
combretum_04
combretum_05
combretum_06
combretum_07
combretum_08
combretum_09
combretum_10
combretum_11
combretum_12
combretum_13
combretum_14
combretum_15
combretum_16
combretum_17
combretum_18
combretum_19
combretum_20
combretum_21
combretum_22
combretum_23
combretum_24
combretum_25
combretum_26
combretum_27
combretum_28
combretum_29
combretum_30
combretum_31
combretum_32
combretum_33
combretum_34
combretum_35
croton_01
croton_02
croton_03
croton_04
croton_05
croton_06
croton_07
croton_08
croton_09


matayba_21
matayba_22
matayba_23
matayba_24
matayba_25
matayba_26
matayba_27
matayba_28
matayba_29
matayba_30
matayba_31
matayba_32
matayba_33
matayba_34
matayba_35
mimosa_01
mimosa_02
mimosa_03
mimosa_04
mimosa_05
mimosa_06
mimosa_07
mimosa_08
mimosa_09
mimosa_10
mimosa_11
mimosa_12
mimosa_13
mimosa_14
mimosa_15
mimosa_16
mimosa_17
mimosa_18
mimosa_19
mimosa_20
mimosa_21
mimosa_22
mimosa_23
mimosa_24
mimosa_25
mimosa_26
mimosa_27
mimosa_28
mimosa_29
mimosa_30
mimosa_31
mimosa_32
mimosa_33
mimosa_34
mimosa_35
myrcia_01
myrcia_02
myrcia_03
myrcia_04
myrcia_05
myrcia_06
myrcia_07
myrcia_08
myrcia_09
myrcia_10
myrcia_11
myrcia_12
myrcia_13
myrcia_14
myrcia_15
myrcia_16
myrcia_17
myrcia_18
myrcia_19
myrcia_20
myrcia_21
myrcia_22
myrcia_23
myrcia_24
myrcia_25
myrcia_26
myrcia_27
myrcia_28
myrcia_29
myrcia_30
myrcia_31
myrcia_32
myrcia_33
myrcia_34
myrcia_35
protium_01
protium_02
protium_03
protium_04
protium_05
protium_06
protium_07
protium_08
protium_09
protium_10
protium_11
protium_12
protium_13
protium_14
protium_15
protium_16
protium_17
protium_18
protium_19
protium_20
protium_21
protium_22
protium_23
protium_24
protium_25
protium_26
protium_27
protium_28
protium_29
protium_30
protium_31
protium_32
protium_33
protium_34

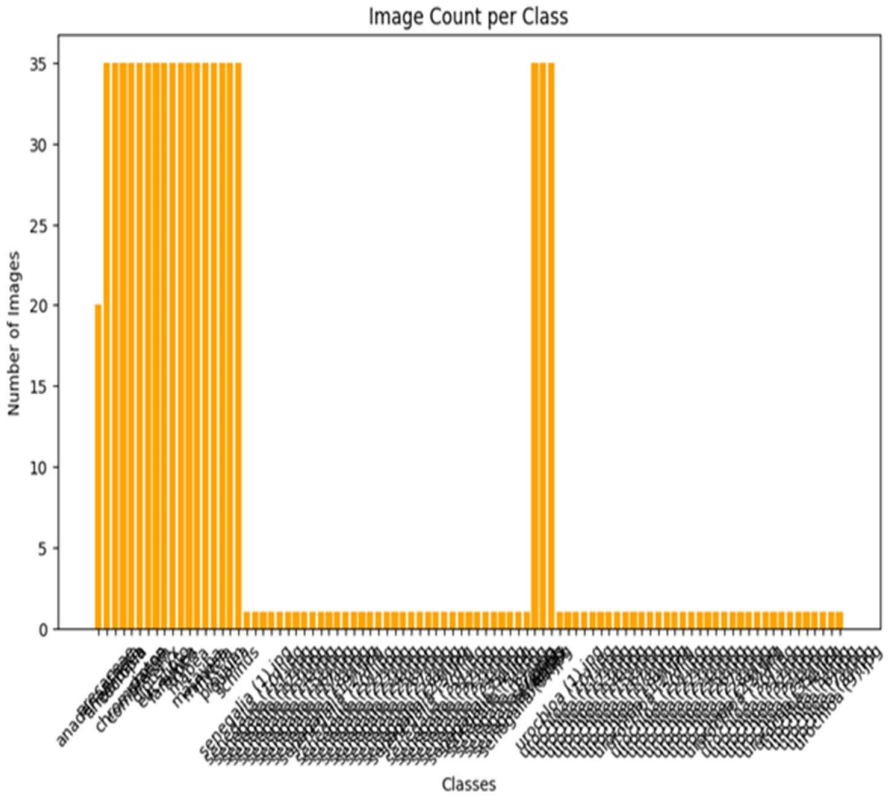

faramea_07 faramea_08 faramea_09 faramea_10 faramea_11 faramea_12 faramea_13 faramea_14 faramea_15 faramea_16 faramea_17 faramea_18 faramea_19 faramea_20 faramea_21 faramea_22 faramea_23
faramea_24 faramea_25 faramea_26 faramea_27 faramea_28 faramea_29 faramea_30 faramea_31 faramea_32 faramea_33 faramea_34 faramea_35 hyptis_01 hyptis_02 hyptis_03 hyptis_04 hyptis_05
hyptis_06 hyptis_07 hyptis_08 hyptis_09 hyptis_10 hyptis_11 hyptis_12 hyptis_13 hyptis_14 hyptis_15 hyptis_16 hyptis_17 hyptis_18 hyptis_19 hyptis_20 hyptis_21 hyptis_22
hyptis_23 hyptis_24 hyptis_25 hyptis_26 hyptis_27 hyptis_28 hyptis_29 hyptis_30 hyptis_31 hyptis_32 hyptis_33 hyptis_34 hyptis_35 mabea_01 mabea_02 mabea_03 mabea_04
mabea_05 mabea_06 mabea_07 mabea_08 mabea_09 mabea_10 mabea_11 mabea_12 mabea_13 mabea_14 mabea_15 mabea_16 mabea_17 mabea_18 mabea_19 mabea_20 mabea_21
mabea_22 mabea_23 mabea_24 mabea_25 mabea_26 mabea_27 mabea_28 mabea_29 mabea_30 mabea_31 mabea_32 mabea_33 mabea_34 mabea_35 matayba_01 matayba_02 matayba_03
matayba_0 matayba_0 matayba_0 matayba_0 matayba_0 matayba_0 matayba_1 matayba_1 matayba_1 matayba_1 matayba_1 matayba_1 matayba_1 matayba_1 matayba_1 matayba_2
senegalia (32) senegalia (33) senegalia (34) senegalia (35) serjania_01 serjania_02 serjania_03 serjania_04 serjania_05 serjania_06 serjania_07 serjania_08 serjania_09 serjania_10 serjania_11 serjania_12 serjania_13
serjania_14 serjania_15 serjania_16 serjania_17 serjania_18 serjania_19 serjania_20 serjania_21 serjania_22 serjania_23 serjania_24 serjania_25 serjania_26 serjania_27 serjania_28 serjania_29 serjania_30
serjania_31 serjania_32 serjania_33 serjania_34 serjania_35 syagrus_01 syagrus_02 syagrus_03 syagrus_04 syagrus_05 syagrus_06 syagrus_07 syagrus_08 syagrus_09 syagrus_10 syagrus_11 syagrus_12
syagrus_13 syagrus_14 syagrus_15 syagrus_16 syagrus_17 syagrus_18 syagrus_19 syagrus_20 syagrus_21 syagrus_22 syagrus_23 syagrus_24 syagrus_25 syagrus_26 syagrus_27 syagrus_28 syagrus_29
syagrus_30 syagrus_31 syagrus_32 syagrus_33 syagrus_34 syagrus_35 tridax_01 tridax_02 tridax_03 tridax_04 tridax_05 tridax_06 tridax_07 tridax_08 tridax_09 tridax_10 tridax_11
tridax_12 tridax_13 tridax_14 tridax_15 tridax_16 tridax_17 tridax_18 tridax_19 tridax_20 tridax_21 tridax_22 tridax_23 tridax_24 tridax_25 tridax_26 tridax_27 tridax_28
tridax_29 tridax_30 tridax_31 tridax_32 tridax_33 tridax_34 tridax_35 urochloa (1) urochloa (2) urochloa (3) urochloa (4) urochloa (5) urochloa (6) urochloa (7) urochloa (8) urochloa (9) urochloa (10)


protium_35
qualea_01
qualea_02
qualea_03
qualea_04
qualea_05
qualea_06
qualea_07
qualea_08
qualea_09
qualea_10
qualea_11
qualea_12
qualea_13
qualea_14
qualea_15
qualea_16
qualea_17
qualea_18
qualea_19
qualea_20
qualea_21
qualea_22
qualea_23
qualea_24
qualea_25
qualea_26
qualea_27
qualea_28
qualea_29
qualea_30
qualea_31
qualea_32
qualea_33
qualea_34
qualea_35
schinus_01
schinus_02
schinus_03
schinus_04
schinus_05
schinus_06
schinus_07
schinus_08
schinus_09
schinus_10
schinus_11
schinus_12
schinus_13
schinus_14
schinus_15
schinus_16
schinus_17
schinus_18
schinus_19
schinus_20
schinus_21
schinus_22
schinus_23
schinus_24
schinus_25
schinus_26
schinus_27
schinus_28
schinus_29
schinus_30
schinus_31
schinus_32
schinus_33
schinus_34
schinus_35
senegalia (1)
senegalia (2)
senegalia (3)
senegalia (4)
senegalia (5)
senegalia (6)
senegalia (7)
senegalia (8)
senegalia (9)
senegalia (10)
senegalia (11)
senegalia (12)
senegalia (13)
senegalia (14)
senegalia (15)
senegalia (16)
senegalia (17)
senegalia (18)
senegalia (19)
senegalia (20)
senegalia (21)
senegalia (22)
senegalia (23)
senegalia (24)
senegalia (25)
senegalia (26)
senegalia (27)
senegalia (28)
senegalia (29)
senegalia (30)
senegalia (31)


urochloa (11)
urochloa (12)
urochloa (13)
urochloa (14)
urochloa (15)
urochloa (16)
urochloa (17)
urochloa (18)
urochloa (19)
urochloa (20)
urochloa (21)
urochloa (22)
urochloa (23)
urochloa (24)
urochloa (25)
urochloa (26)
urochloa (27)
urochloa (28)
urochloa (29)
urochloa (30)
urochloa (31)
urochloa (32)
urochloa (33)
urochloa (34)
urochloa (35)

```
data_path = r"C:\IMP\pollen_grains\data\pollen_data"

# Count files per class
labels = [file.split('_')[0] for file in os.listdir(data_path) if file.
 ↪endswith(('.png', '.jpg', '.jpeg'))]
count = Counter(labels)

plt.figure(figsize=(10,6))
plt.bar(count.keys(), count.values(), color='orange')
plt.xlabel('Classes')
plt.ylabel('Number of Images')
plt.title('Image Count per Class')
plt.xticks(rotation=45)
plt.show()

# Display sample images
plt.figure(figsize=(10,10))
i = 1
for file in os.listdir(data_path)[:9]:
```

```
    img = Image.open(os.path.join(data_path, file))
    plt.subplot(3,3,i)
    plt.imshow(img)
    plt.title(file.split("_")[0])
    plt.axis('off')
    i += 1
plt.show()
```



[6]:
```python
import numpy as np
from tensorflow.keras.utils import img_to_array, load_img
from sklearn.preprocessing import LabelEncoder

X, y = [], []

for file in os.listdir(data_path):
    if file.endswith(('.png', '.jpg', '.jpeg')):
        label = file.split('_')[0]
        img = load_img(os.path.join(data_path, file), target_size=(128, 128))
        img = img_to_array(img) / 255.0
        X.append(img)
```

```
        y.append(label)

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

print(" Data shape:", X.shape)
print(" Labels shape:", y.shape)
```

```
 Data shape: (790, 128, 128, 3)
 Labels shape: (790,)
```

[7]:
```
from collections import Counter
import os

data_path = r"C:\IMP\pollen_grains\data\pollen_data"

labels = [file.split('_')[0] for file in os.listdir(data_path) if file.
 ↪endswith(('.jpg', '.jpeg', '.png'))]
count = Counter(labels)

print(count)
```

```
Counter({'arecaceae': 35, 'arrabidaea': 35, 'cecropia': 35, 'chromolaena': 35,
'combretum': 35, 'croton': 35, 'dipteryx': 35, 'eucalipto': 35, 'faramea': 35,
'hyptis': 35, 'mabea': 35, 'matayba': 35, 'mimosa': 35, 'myrcia': 35, 'protium':
35, 'qualea': 35, 'schinus': 35, 'serjania': 35, 'syagrus': 35, 'tridax': 35,
'anadenanthera': 20, 'senegalia (1).jpg': 1, 'senegalia (10).jpg': 1, 'senegalia
(11).jpg': 1, 'senegalia (12).jpg': 1, 'senegalia (13).jpg': 1, 'senegalia
(14).jpg': 1, 'senegalia (15).jpg': 1, 'senegalia (16).jpg': 1, 'senegalia
(17).jpg': 1, 'senegalia (18).jpg': 1, 'senegalia (19).jpg': 1, 'senegalia
(2).jpg': 1, 'senegalia (20).jpg': 1, 'senegalia (21).jpg': 1, 'senegalia
(22).jpg': 1, 'senegalia (23).jpg': 1, 'senegalia (24).jpg': 1, 'senegalia
(25).jpg': 1, 'senegalia (26).jpg': 1, 'senegalia (27).jpg': 1, 'senegalia
(28).jpg': 1, 'senegalia (29).jpg': 1, 'senegalia (3).jpg': 1, 'senegalia
(30).jpg': 1, 'senegalia (31).jpg': 1, 'senegalia (32).jpg': 1, 'senegalia
(33).jpg': 1, 'senegalia (34).jpg': 1, 'senegalia (35).jpg': 1, 'senegalia
(4).jpg': 1, 'senegalia (5).jpg': 1, 'senegalia (6).jpg': 1, 'senegalia
(7).jpg': 1, 'senegalia (8).jpg': 1, 'senegalia (9).jpg': 1, 'urochloa (1).jpg':
1, 'urochloa (10).jpg': 1, 'urochloa (11).jpg': 1, 'urochloa (12).jpg': 1,
'urochloa (13).jpg': 1, 'urochloa (14).jpg': 1, 'urochloa (15).jpg': 1,
'urochloa (16).jpg': 1, 'urochloa (17).jpg': 1, 'urochloa (18).jpg': 1,
'urochloa (19).jpg': 1, 'urochloa (2).jpg': 1, 'urochloa (20).jpg': 1, 'urochloa
(21).jpg': 1, 'urochloa (22).jpg': 1, 'urochloa (23).jpg': 1, 'urochloa
(24).jpg': 1, 'urochloa (25).jpg': 1, 'urochloa (26).jpg': 1, 'urochloa
(27).jpg': 1, 'urochloa (28).jpg': 1, 'urochloa (29).jpg': 1, 'urochloa
(3).jpg': 1, 'urochloa (30).jpg': 1, 'urochloa (31).jpg': 1, 'urochloa
(32).jpg': 1, 'urochloa (33).jpg': 1, 'urochloa (34).jpg': 1, 'urochloa
```

```
(35).jpg': 1, 'urochloa (4).jpg': 1, 'urochloa (5).jpg': 1, 'urochloa (6).jpg':
1, 'urochloa (7).jpg': 1, 'urochloa (8).jpg': 1, 'urochloa (9).jpg': 1})
```

[8]:
```
# Keep only classes with at least 2 images
valid_labels = [label for label, cnt in count.items() if cnt >= 2]

X, y = [], []
for file in os.listdir(data_path):
    if file.endswith(('.jpg', '.jpeg', '.png')):
        label = file.split('_')[0]
        if label in valid_labels:
            img = load_img(os.path.join(data_path, file), target_size=(128,␣
 ↪128))
            img = img_to_array(img) / 255.0
            X.append(img)
            y.append(label)

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

print("Data Shape:", X.shape)
```

```
Data Shape: (720, 128, 128, 3)
```

[9]:
```
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import LabelEncoder
import pickle

# Label encoding
le = LabelEncoder()
y_encoded = le.fit_transform(y)

# Save label encoder for later use
with open('label_encoder.pkl', 'wb') as f:
    pickle.dump(le, f)

# Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, stratify=y_encoded, random_state=42)

# CNN Model
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 3)),
```

```
    MaxPooling2D(2,2),
    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(len(le.classes_), activation='softmax')
])

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])
model.summary()

# Train model
early_stop = EarlyStopping(monitor='val_loss', patience=3)
model.fit(X_train, y_train, epochs=10, validation_split=0.2,␣
 ↪callbacks=[early_stop])

# Evaluate
loss, accuracy = model.evaluate(X_test, y_test)
print(f" Test Accuracy: {accuracy*100:.2f}%")
```

```
C:\Users\ponna\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\convolutional\base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 126, 126, 32)	896
max_pooling2d (MaxPooling2D)	(None, 63, 63, 32)	0
conv2d_1 (Conv2D)	(None, 61, 61, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 30, 30, 64)	0

```
flatten (Flatten)                    (None, 57600)                                 
↪  0

dense (Dense)                        (None, 128)                                   
↪7,372,928

dropout (Dropout)                    (None, 128)                                   
↪  0

dense_1 (Dense)                      (None, 21)                                    
↪2,709


Total params: 7,395,029 (28.21 MB)

Trainable params: 7,395,029 (28.21 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10
15/15             9s 402ms/step -
accuracy: 0.0407 - loss: 4.2339 - val_accuracy: 0.1379 - val_loss: 3.0272
Epoch 2/10
15/15             5s 336ms/step -
accuracy: 0.1017 - loss: 2.9171 - val_accuracy: 0.1983 - val_loss: 2.8151
Epoch 3/10
15/15             5s 340ms/step -
accuracy: 0.2253 - loss: 2.6266 - val_accuracy: 0.2845 - val_loss: 2.5170
Epoch 4/10
15/15             5s 329ms/step -
accuracy: 0.3234 - loss: 2.2932 - val_accuracy: 0.3448 - val_loss: 2.1328
Epoch 5/10
15/15             5s 326ms/step -
accuracy: 0.4318 - loss: 1.8284 - val_accuracy: 0.2500 - val_loss: 2.1004
Epoch 6/10
15/15             5s 329ms/step -
accuracy: 0.5384 - loss: 1.5096 - val_accuracy: 0.4828 - val_loss: 1.7300
Epoch 7/10
15/15             5s 323ms/step -
accuracy: 0.7003 - loss: 1.0694 - val_accuracy: 0.5172 - val_loss: 1.5130
Epoch 8/10
15/15             5s 327ms/step -
accuracy: 0.7581 - loss: 0.8620 - val_accuracy: 0.6552 - val_loss: 1.2226
Epoch 9/10
15/15             5s 323ms/step -
```

```
accuracy: 0.8547 - loss: 0.6067 - val_accuracy: 0.6207 - val_loss: 1.1769
Epoch 10/10
15/15             5s 326ms/step -
accuracy: 0.8562 - loss: 0.5086 - val_accuracy: 0.5776 - val_loss: 1.2587
5/5             0s 61ms/step -
accuracy: 0.6487 - loss: 1.2471
 Test Accuracy: 63.89%
```

[10]:
```
model.save('pollen_model.h5')
print(" Model saved as pollen_model.h5")
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or
`keras.saving.save_model(model)`. This file format is considered legacy. We
recommend using instead the native Keras format, e.g.
`model.save('my_model.keras')` or `keras.saving.save_model(model,
'my_model.keras')`.

 Model saved as pollen_model.h5
```

[14]:
```
import tensorflow as tf

# Load your original model
model = tf.keras.models.load_model("pollen_model.h5")

# Convert to quantized TFLite model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_model = converter.convert()

# Save the compressed model
with open("pollen_model.h5", "wb") as f:
    f.write(tflite_model)

print(" Saved as: pollen_model.h5")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be
built. `model.compile_metrics` will be empty until you train or evaluate the
model.

INFO:tensorflow:Assets written to:
C:\Users\ponna\AppData\Local\Temp\tmpubrvu146\assets

INFO:tensorflow:Assets written to:
C:\Users\ponna\AppData\Local\Temp\tmpubrvu146\assets

Saved artifact at 'C:\Users\ponna\AppData\Local\Temp\tmpubrvu146'. The following
endpoints are available:

* Endpoint 'serve'
  args_0 (POSITIONAL_ONLY): TensorSpec(shape=(None, 128, 128, 3),
```

```
dtype=tf.float32, name='input_layer')
Output Type:
  TensorSpec(shape=(None, 21), dtype=tf.float32, name=None)
Captures:
  1918130911760: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1918130912720: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1918130913104: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1918130911568: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1918130898896: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1918130909840: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1918130906000: TensorSpec(shape=(), dtype=tf.resource, name=None)
  1918130910992: TensorSpec(shape=(), dtype=tf.resource, name=None)
 Saved as: pollen_model.h5
```

]:

]:
```python
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
from tensorflow.keras.utils import img_to_array, load_img
import numpy as np
import pickle
import os

app = Flask(_name_)
UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Load model and label encoder
model = load_model('pollen_model.h5')
label_encoder = pickle.load(open('label_encoder.pkl', 'rb'))

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

@app.route('/predict', methods=['POST'])
def predict():
    if 'file' not in request.files:
        return 'No file uploaded'
    file = request.files['file']
    if file.filename == '':
        return 'No file selected'

    file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
    file.save(file_path)

    image = load_img(file_path, target_size=(128, 128))
```

```
    image = img_to_array(image) / 255.0
    image = np.expand_dims(image, axis=0)

    prediction = model.predict(image)
    pred_label = label_encoder.inverse_transform([np.argmax(prediction)])[0]

    return f'Prediction: {pred_label}'

if _name_ == '_main_':
    app.run(debug=True)
```

[ ]:
```
#HTML page
```

[ ]:
```
<!DOCTYPE html>
<html>
<head>
    <title>Pollen Grain Classification</title>
    <style>
        body {
            margin: 0;
            padding: 0;
            background-image: url("/static/background.jpg"); /* Place your␣
 ↪image in static/ */
            background-size: cover;
            background-repeat: no-repeat;
            background-position: center;
            font-family: Arial, sans-serif;
            color: white;
            text-align: center;
        }

        .container {
            background-color: rgba(0, 0, 0, 0.6);
            padding: 40px;
            border-radius: 15px;
            width: 400px;
            margin: 100px auto;
        }

        h1 {
            margin-bottom: 30px;
        }

        input[type="file"] {
            padding: 10px;
            margin-bottom: 20px;
        }
```

```
        input[type="submit"] {
            padding: 10px 25px;
            font-size: 16px;
            background-color: #4CAF50;
            border: none;
            color: white;
            cursor: pointer;
            border-radius: 5px;
        }

        input[type="submit"]:hover {
            background-color: #45a049;
        }

        img {
            margin-top: 20px;
            border-radius: 10px;
            box-shadow: 0 0 10px #fff;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Pollen Grain Classification</h1>
        <form action="/predict" method="POST" enctype="multipart/form-data">
            <input type="file" name="file" required><br>
            <input type="submit" value="Predict">
        </form>

        {% if prediction %}
            <h2>Prediction: {{ prediction }}</h2>
            <img src="{{ image_path }}" alt="Uploaded Image" style="width:␣
↪300px;">
        {% endif %}
    </div>
</body>
</html>
```

[ ]:

[ ]:
```
#LOGOUT.HTML
```

[ ]:
```
<!DOCTYPE html>
<html>
<head><title>Logout</title></head>
<body>
```

```
    <h2>You have been logged out.</h2>
</body>
</html>
```

[ ]:

[ ]: PREDICTION.HTML

[ ]:
```
<!DOCTYPE html>
<html>
<head><title>Result</title></head>
<body>
    <h2>Prediction: {{ prediction }}</h2>
    <img src="{{ image_path }}" width="200">
    <br><a href="/">Try Another</a>
</body>
</html>
```

[ ]:

[ ]: APP.PY

[ ]:
```
from flask import Flask, render_template, request
import tensorflow as tf
import numpy as np
from PIL import Image
import os
import pickle

# Create uploads folder if not exists
UPLOAD_FOLDER = 'uploads'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Load model and label encoder
model = tf.keras.models.load_model('pollen_model.h5')
label_encoder = pickle.load(open('label_encoder.pkl', 'rb'))

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

@app.route('/predict', methods=['POST'])
def predict():
```

```
    if 'file' not in request.files:
        return "No file uploaded."

    file = request.files['file']
    if file.filename == '':
        return "No selected file."

    if file:
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)
        file.save(filepath)

        try:
            image = Image.open(filepath).resize((128, 128))
            image = np.array(image) / 255.0
            image = np.expand_dims(image, axis=0)

            prediction = model.predict(image)
            class_index = np.argmax(prediction)
            class_name = label_encoder.inverse_transform([class_index])[0]

            return render_template('index.html', prediction=class_name,␣
 ↪image_path=filepath)

        except Exception as e:
            return f"Error processing image: {e}"

if __name__ == '__main__':
    app.run(debug=True)
```