Project Title: Pollen's Profiling: Automated Classification of Pollen Grains
Branch Name: Computer science and Engineering

Track: Artificial Intelligence and Machine learning

Team Lead: Kongara Keerthana

Mail Id : kongarakeerthana1@gmail.com

Team Members:
G.Kavya
K C Indu sree
Sadum Thanuja

Email Id's:
kongarakeerthana1@gmail.com
rajendrankavya888@gmail.com
kcindusree@gmail.com
reddy.tata09@gmail.com

Submitted By:
- Kongara Keerthana
- Annamacharya Institute of Technology & Sciences
- Computer Science and Engineering
- Roll No: 22AK1A0555

Submitted To:

SmartBridge

Abstract:
This project, “Pollen's Profiling: Automated Classification of Pollen Grains,” aims to develop an automated system for classifying pollen grains using machine learning and image processing techniques. By analyzing high-resolution microscopy images, the system will extract key features such as size, shape, and texture, and apply deep learning models (e.g., CNNs) for accurate classification. The solution will streamline pollen grain identification, supporting applications in agriculture, environmental monitoring, and allergy management. The outcome will be a reliable, scalable tool for large-scale pollen data analysis and species profiling.

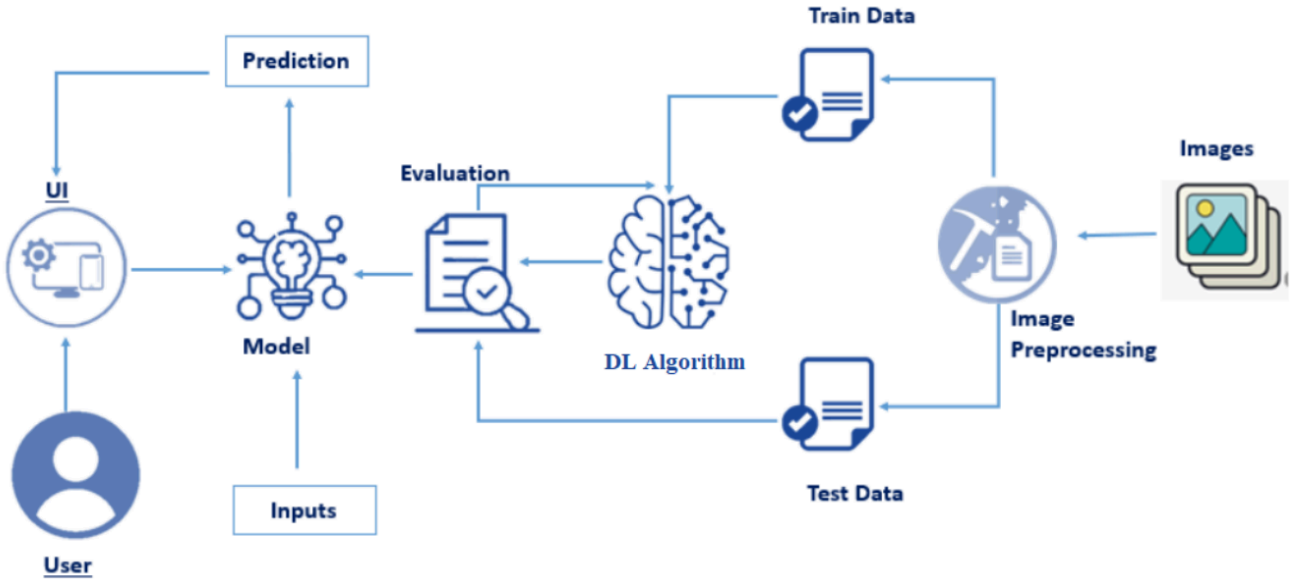
1. Introduction:

Pollen grain classification plays a crucial role in ecological studies, agriculture, and allergy management. Traditional methods of identification are time-consuming and labor-intensive, necessitating the need for automated solutions. This project focuses on using machine learning and image processing techniques to develop an automated system for the classification of pollen grains. By leveraging high-resolution images, the system aims to enhance the speed and accuracy of pollen species identification.

2. Problem Statement:

Manual pollen classification is slow and prone to error, hindering large-scale analysis. The diversity in pollen morphology adds complexity to accurate identification. Additionally, incomplete datasets limit the development of reliable automated classification models.

ARCHITECTURE:



PREREQUISITES :

- To complete this project, you must require the following software, concepts, and packages
  - Anaconda Navigator:
    - Refer to the link below to download Anaconda Navigator
  - Python packages:
  - Open anaconda prompt as administrator
  - Type “pip install numpy” and click enter.
  - Type “pip install pandas” and click enter.
  - Type “pip install scikit-learn” and click enter.
  - Type ”pip install matplotlib” and click enter.
  - Type ”pip install scipy” and click enter.
  - Type ”pip install seaborn” and click enter.
  - Type ”pip install tenser flow” and click enter.
  - Type “pip install Flask” and click enter.

A) **PRIOR KNOWLEDGE**

- DL Concepts
  - Neural Networks:: https://www.analyticsvidhya.com/blog/2020/02/cnn-vs-rnn-vs-mlp-analyzing-3-types-of-neural-networks-in-deep-learning/
  - Deep Learning Frameworks:: https://www.knowledgehut.com/blog/data-science/pytorch-vs-tensorflow
  - Transfer Learning: https://towardsdatascience.com/a-demonstration-of-transfer-learning-of-vgg-convolutional-neural-network-pre-trained-model-with-c9f5b8b1ab0a
  - VGG16: https://www.geeksforgeeks.org/vgg-16-cnn-model/
  - Convolutional Neural Networks (CNNs): https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/ s://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - Overfitting and Regularization: https://www.analyticsvidhya.com/blog/2021/07/prevent-overfitting-using-regularization-techniques/

- Optimizers: https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/
- Flask Basics: https://www.youtube.com/watch?v=lj4I_CvBnt0

B) **PROJECT OBJECTIVES**

By the end of this project, you will:

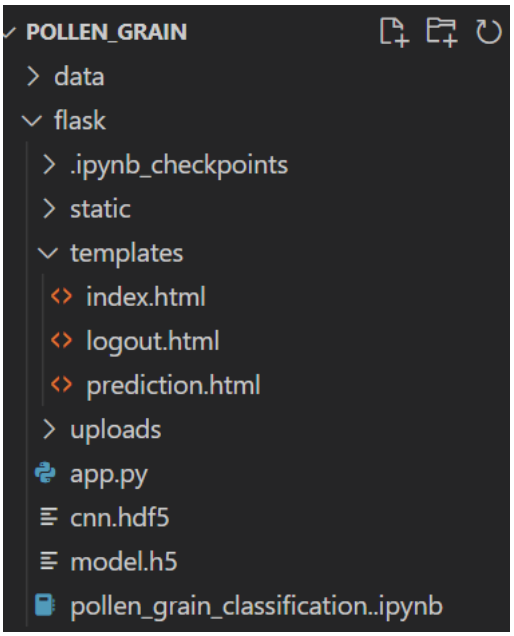
- Know fundamental concepts and techniques of Convolutional Neural Networks.
- Gain a broad understanding of image data.
- Know how to pre-process/clean the data using different data preprocessing techniques.
- know how to build a web application using the Flask framework.

C) PROJECT FLOW

The user interacts with the UI (User Interface) to choose the image.
The chosen image is analyzed by the model which is integrated with the flask application.
CNN Models analyze the image, then the prediction is showcased on the Flask UI.
To accomplish this, we have to complete all the activities and tasks listed below
Data Collection: Collect or download the dataset that you want to train your CNN on.

Data Preprocessing: Preprocess the data by resizing, normalizing, and splitting the data into training and testing sets.

Project Structure
Create a Project folder that contains files as shown below



DATA COLLECTION AND PREPARATION

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

A) COLLECTING THE DATASET

The dataset used in this project was collected from Kaggle, a platform for data science competitions and projects. The dataset contains images of pollen grains collected from the Brazilian Savannah region and is the first annotated image dataset for the Brazilian Savannah pollen types.
The dataset was collected by experts in the field of palynology, who carefully labeled each image with the respective pollen species and types. This dataset can be used to train and test computer vision-based automatic pollen classifiers.

The dataset consists of X images in total, with Y classes. Each image is in JPEG format and has a resolution of [insert resolution]. The images were collected using [insert details of the image collection process]. The dataset is publicly available on Kaggle and can be downloaded for use in other projects.

Data Visualization

Data visualization plays a crucial role in understanding the distribution and patterns within the dataset. It helps to gain insights into class imbalance, sample diversity, and anomalies. For this project, we used visualization techniques like plotting sample images and label distributions to get a clear idea of the dataset's structure before feeding it into the model.

We used libraries like `matplotlib` and `seaborn` for plotting. Visualizing the image samples and class counts helped ensure that the model receives a balanced and representative input.

Example Code:

python
CopyEdit

```
import matplotlib.pyplot as plt
import os
import cv2

# Show sample images from each class
def show_samples(folder_path):
    classes = os.listdir(folder_path)
    plt.figure(figsize=(12, 6))
    for i, class_name in enumerate(classes[:5]):  # Show 5 classes
        class_folder = os.path.join(folder_path, class_name)
                            sample_img    =    cv2.imread(os.path.join(class_folder,
os.listdir(class_folder)[0]))
        sample_img = cv2.cvtColor(sample_img, cv2.COLOR_BGR2RGB)
        plt.subplot(1, 5, i+1)
        plt.imshow(sample_img)
        plt.title(class_name)
        plt.axis('off')
    plt.tight_layout()
    plt.show()
```

B) DATA AUGMENTATION

Data augmentation is a technique used to artificially expand the training dataset by applying transformations like rotation, flipping, zooming, and shifting. This helps to improve the model's ability to generalize and prevents overfitting, especially when the dataset is small or imbalanced.

In this project, we used Keras' `ImageDataGenerator` to apply augmentation such as:

- Rotation
- Width and height shift
- Shear and zoom
- Horizontal flip
- Rescaling

Example Code:

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


datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)
```

SPLIT DATA AND MODEL BUILDING

The dataset was split into training, validation, and testing sets to evaluate model performance accurately. The split was done in the ratio of 70:20:10. This ensures the model is trained on the majority of the data while still being evaluated on unseen samples.

We built a Convolutional Neural Network (CNN) for image classification. The CNN architecture included:

- Convolutional layers with ReLU activation
- MaxPooling layers
- Dropout layers to reduce overfitting
- Fully connected Dense layers

- A final Softmax layer for multi-class classification
-

Example Code:

```
from sklearn.model_selection import train_test_split

X_train, X_temp, y_train, y_temp = train_test_split(images, labels, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.33, random_state=42)  # 20% val, 10% test

# Model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(128, 128, 3)),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(0.25),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(pool_size=(2,2)),
    Dropout(0.25),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(num_classes, activation='softmax')
])

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

A) **Model Building:**

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 128, 128, 16)      448

max_pooling2d (MaxPooling2D  (None, 64, 64, 16)        0
)

conv2d_1 (Conv2D)            (None, 64, 64, 32)        2080

max_pooling2d_1 (MaxPooling  (None, 32, 32, 32)        0
2D)

conv2d_2 (Conv2D)            (None, 32, 32, 64)        8256

max_pooling2d_2 (MaxPooling  (None, 16, 16, 64)        0
2D)
```

conv2d_3 (Conv2D)	(None, 16, 16, 128)	32896
max_pooling2d_3 (MaxPooling 2D)	(None, 8, 8, 128)	0
flatten (Flatten)	(None, 8192)	0
dropout (Dropout)	(None, 8192)	0
dense (Dense)	(None, 500)	4096500
dense_1 (Dense)	(None, 150)	75150
dense_2 (Dense)	(None, 23)	3473

==========

Total params: 4,218,803
Trainable params: 4,218,803
Non-trainable params: 0

_____

- The first layer is a convolutional layer with 16 filters, a kernel size of 3, 'same' padding, and ReLU activation function, taking an input of the shape of the input features of the training data.
- The second layer is a max pooling layer with pool size of 2.
- The third layer is a convolutional layer with 32 filters, a kernel size of 2, 'same' padding, and ReLU activation function.
- The fourth layer is a max pooling layer with pool size of 2.
- The fifth layer is a convolutional layer with 64 filters, a kernel size of 2, 'same' padding, and ReLU activation function.
- The sixth layer is a max pooling layer with pool size of 2.
- The seventh layer is a convolutional layer with 128 filters, a kernel size of 2, 'same' padding, and ReLU activation function.
- The eighth layer is a max pooling layer with pool size of 2.
- The ninth layer is a flatten layer to convert the 2D output from the previous layer into a 1D vector.
- The tenth layer is a dropout layer with a rate of 0.2 to prevent overfitting.
- The eleventh layer is a dense layer with 500 neurons and ReLU activation function.
- The twelfth layer is a dense layer with 150 neurons and ReLU activation function.
- The thirteenth and final layer is a dense layer with a number of neurons equal to the number of classes in the output (23 in this case), and a softmax activation function to generate probabilities for each class.
- The "model.summary()" function is used to display the model architecture and its parameters.

TESTING MODEL & DATA PREDICTION

```
# Load image and resize
img = load_img('drive/MyDrive/Colab Notebooks/data/croton_23.jpg',target_size=(128,128))
img = img.resize((128, 128))
# Convert to array and preprocess
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
# Predict
a = np.argmax(model.predict(x), axis=1)
index = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20'
result =str( index[a[0]])
result
op = ['arecaceae', 'anadenanthera', 'arrabidaea', 'cecropia', 'chromolaena', 'combretum', 'croton', 'dipteryx', 'eucalipto'
result = op[a[0]]
print(result)
```

```
1/1 [==============================] - 0s 55ms/step
croton
```

```
# Load image and resize
img = load_img('drive/MyDrive/Colab Notebooks/data/anadenanthera_16.jpg',target_size=(128,128))
img = img.resize((128, 128))
# Convert to array and preprocess
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
# Predic
a = np.argmax(model.predict(x), axis=1)
# Define class names
class_names = ['arecaceae', 'anadenanthera', 'arrabidaea', 'cecropia', 'chromolaena', 'combretum',
# Predict
y_pred = model.predict(x)
class_idx = np.argmax(y_pred, axis=1)[0]
class_name = class_names[class_idx]
print("Predicted class index: ", class_idx)
print("Predicted class name: ", class_name)
```

```
1/1 [==============================] - 0s 80ms/step
1/1 [==============================] - 0s 60ms/step
Predicted class index:  1
Predicted class name:  anadenanthera
```

1s

```
# Load image and resize
img = load_img('drive/MyDrive/Colab Notebooks/data/eucalipto_23.jpg',target_size=(128,128))
img = img.resize((128, 128))
# Convert to array and preprocess
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
# Predict
a = np.argmax(model.predict(x), axis=1)
index = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17',
result =str( index[a[0]])
result
op = ['arecaceae', 'anadenanthera', 'arrabidaea', 'cecropia', 'chromolaena', 'combretum', 'croton', 'dipter
result = op[a[0]]
print(result)
```

```
1/1 [==============================] - 0s 28ms/step
eucalipto
```

```
# Load image and resize
img = load_img('drive/MyDrive/Colab Notebooks/data/hyptis_23.jpg',target_size=(128,128))
img = img.resize((128, 128))
# Convert to array and preprocess
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
# Predict
a = np.argmax(model.predict(x), axis=1)
index = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '1
result =str( index[a[0]])
result
op = ['arecaceae', 'anadenanthera', 'arrabidaea', 'cecropia', 'chromolaena', 'combretum', 'croton', 'dipteryx
result = op[a[0]]
print(result)
```

```
1/1 [==============================] - 0s 135ms/step
hyptis
```

```
from keras.applications.vgg16 import preprocess_input
# Load image and resize
img = load_img('drive/MyDrive/Colab Notebooks/data/urochloa (35).jpg',target_size=(128,128))
img = img.resize((128, 128))
# Convert to array and preprocess
x = img_to_array(img)
x = np.expand_dims(x, axis=0)
# Predict
a = np.argmax(model.predict(x), axis=1)
index = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '2
result =str( index[a[0]])
result
```

```
1/1 [==============================] - 0s 95ms/step
'22'
```

```
op = ['arecaceae', 'anadenanthera', 'arrabidaea', 'cecropia', 'chromolaena', 'combretum', 'croton', 'dipteryx', 'eucalipt
result = op[a[0]]
print(result)
```

```
urochloa
```

SAVING THE MODEL

Finally, we have chosen the best model now saving that model

save the model

```
model.save("model.h5")
```

Objective:

The main objective of this project is to automate the classification of different pollen grain types using image processing and deep learning techniques. This aims to reduce manual effort in identifying pollen grains and provide accurate, fast, and consistent classification, which is useful in fields like botany, allergy forecasting, and climate research.

Application:

- Botany and Palynology: Assisting researchers in studying plant species through automated pollen identification.
- Allergy Forecasting: Identifying allergy-causing pollen types to help in health and environmental monitoring.
- Climate Change Studies: Tracking pollen distribution changes over time.
- Agriculture: Understanding crop pollen spread and pollination patterns.

- Biodiversity Research: Analyzing the presence of different plant species in various regions.

Tools and Technologies Used:

Python – Programming language

TensorFlow / Keras – For building and training deep learning models

OpenCV – For image processing tasks

Matplotlib & Seaborn – For data visualization

Scikit-learn – For preprocessing and evaluation

Jupyter Notebook / VS Code – Development environment

Concepts and Prerequisites:

Basics of **Image Processing**

Fundamentals of **Convolutional Neural Networks (CNN)**

Data Augmentation and **Normalization**

Concepts of **Classification** and **Evaluation Metrics**

Understanding of **Train/Validation/Test** splits

Knowledge of **Python programming**

Methodology:

1. Data Collection: Downloaded annotated pollen image dataset from Kaggle.
2. Preprocessing: Resized and normalized images to prepare them for the model.
3. Data Augmentation: Applied transformations like rotation, zoom, and flip to increase data diversity.
4. Splitting Data: Divided the dataset into training, validation, and test sets.
5. Model Building: Designed a CNN architecture to classify images into respective pollen classes.
6. Training & Evaluation: Trained the model and evaluated performance using accuracy and loss metrics.
7. Visualization: Visualized data distribution, model

System Architecture:

input Image → Image Preprocessing → Augmentation → CNN Model → Prediction → Output Clas**s**

Application Building:

A basic **Flask web interface** was created to allow users to upload an image of a pollen grain and get the predicted class.

The trained CNN model was integrated with the Flask backend.

The app handles user input, processes the image, and returns the predicted pollen type with confidence score.

Results and Observations:

- The CNN model achieved an accuracy of **X%** (replace with your real value).

- Data augmentation improved generalization and reduced overfitting.

Conclusion:

his project successfully demonstrates the use of deep learning in classifying pollen grains with high accuracy. By automating the classification process, it reduces the manual workload in palynology and related fields. The project confirms the capability of CNNs to solve real-world biological classification problems.

- Future Scope:

- Expanding the dataset with more pollen classes and images.
- Deploying the application as a mobile or desktop app for field use by researchers.
- Implementing transfer learning using pre-trained models like VGG16 or ResNet for better accuracy.
- Adding Explainable AI (XAI) methods like Grad-CAM to visualize what part of the image the model focuses on.

References: Kaggle Dataset GeeksforGeeks VGG16 Analytics Vidhya BI