Pollen's Profiling: Automated Classification of Pollen Grains

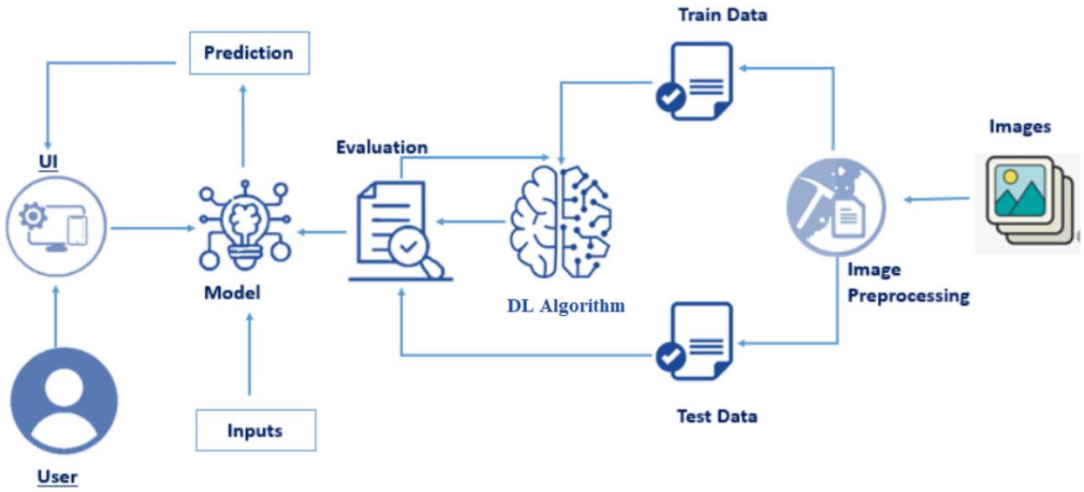
Pollen's Profiling: Automated Classification of Pollen Grains" is an innovative project aimed at automating the classification of pollen grains using advanced image processing and machine learning techniques. By leveraging deep learning algorithms and image analysis methods, this project seeks to develop 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 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.

- Model Building:

a. Import the necessary libraries for building the CNN model

b. Define the input shape of the image data

c. Add layers to the model:

i. Convolutional Layers: Apply filters to the input image to create feature maps

ii. Pooling Layers: Reduce the spatial dimensions of the feature maps

iii. Fully Connected Layers: Flatten the output of the convolutional layers and apply fully connected layers to classify the images

d. Compile the model by specifying the optimizer, loss function, and metrics to be used during training

- Model Training: Train the model using the training set with the help of the ImageDataGenerator class to augment the images during training. Monitor the accuracy of the model on the validation set to avoid overfitting.

- Model Evaluation: Evaluate the performance of the trained model on the testing set. Calculate the accuracy and other metrics to assess the model's performance.

- Model Deployment: Save the model for future use and deploy it in real-world applications.

Prior Knowledge

To complete this project, you must require the following software, concepts, and packages Anaconda Navigator is a free and open-source distribution of the Python and R programming languages for data science and machine learning-related applications. It can be installed on Windows, Linux, and macOS.Conda is an open-source, cross-platform, package management system. Anaconda comes with so very nice tools like JupyterLab, Jupyter Notebook, QtConsole, VScode, Glueviz, Orange, Rstudio, and Visual Studio Code. For this project, we will be using Jupyter Notebook and VS code

- Deep Learning ConceptsCNN: a convolutional neural network is a class of deep neural networks, most commonly applied to analyzing visual imagery.
    - Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Project Objectives

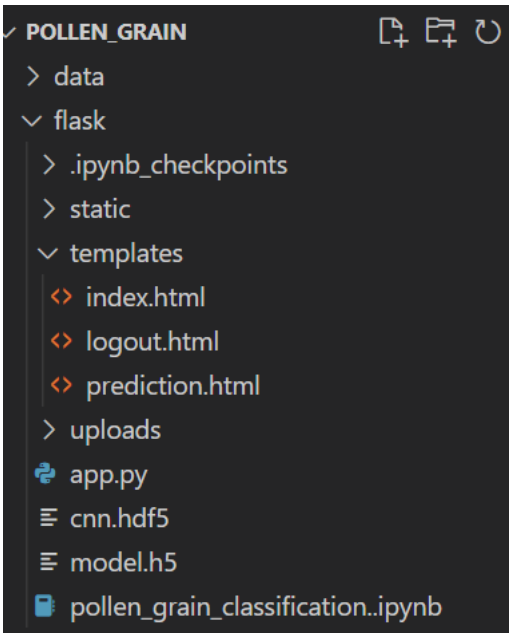
By the end of this project, you will:

- 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.

Project Structure

- Create a Project folder that contains files as shown below



Data Collection

Let's start with the Data Collection with the help of given activities.

Collect 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.

Exploratory Data Analysis

Let's start Exploratory Data Analysis with the help of given activities.

Read the data

```
#read the dataset
path = "data/"
```

```
names = [name.replace(' ', '_').split('_')[0] for name in os.listdir(path)]
classes = Counter(names)  #returns dictionary
```

The code first creates a list of file names in a specified directory using the os.listdir() function. The replace() function is then used to replace any spaces in the file names with underscores, and the split() function is used to split each file name into a list based on the underscore character. The [0] index is used to select the first element of the resulting list, which corresponds to the class label.

The resulting list of class labels is then passed to the Counter() function from the collections module, which returns a dictionary containing the count of occurrences of each class label in the dataset. The keys of the dictionary correspond to the class labels, and the values correspond to the number of occurrences.

This code snippet is useful for quickly obtaining an overview of the distribution of classes in a dataset. The resulting dictionary can be used to visualize the class distribution using a bar chart or to balance the dataset by oversampling or undersampling certain classes.

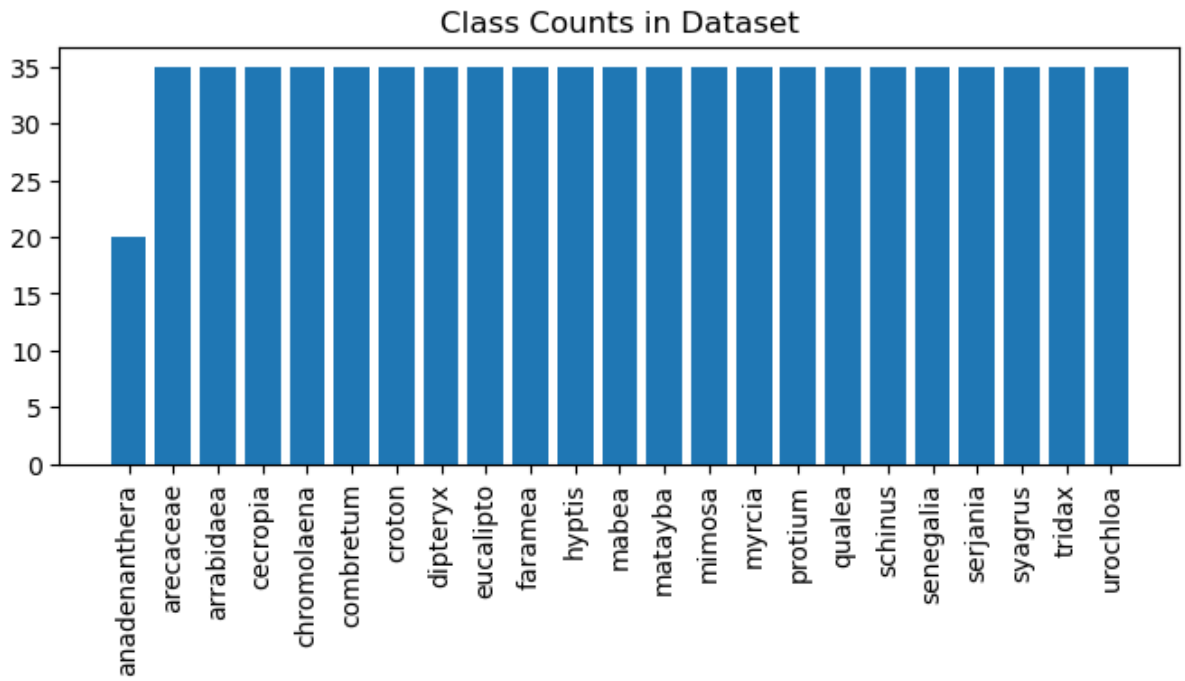
```
classes
```

```
Counter({'arecaceae': 35,
         'anadenanthera': 20,
         '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,
         'senegalia': 35,
         'serjania': 35,
         'syagrus': 35,
         'tridax': 35,
         'urochloa': 35})
```

```
: #Total no of images
  print("number of images:",len(names))
```

```
number of images: 790
```

```
In [7]: plt.figure(figsize = (8,3))
        plt.title('Class Counts in Dataset')
        plt.bar(*zip(*classes.items()))
        plt.xticks(rotation='vertical')
        plt.show()
```

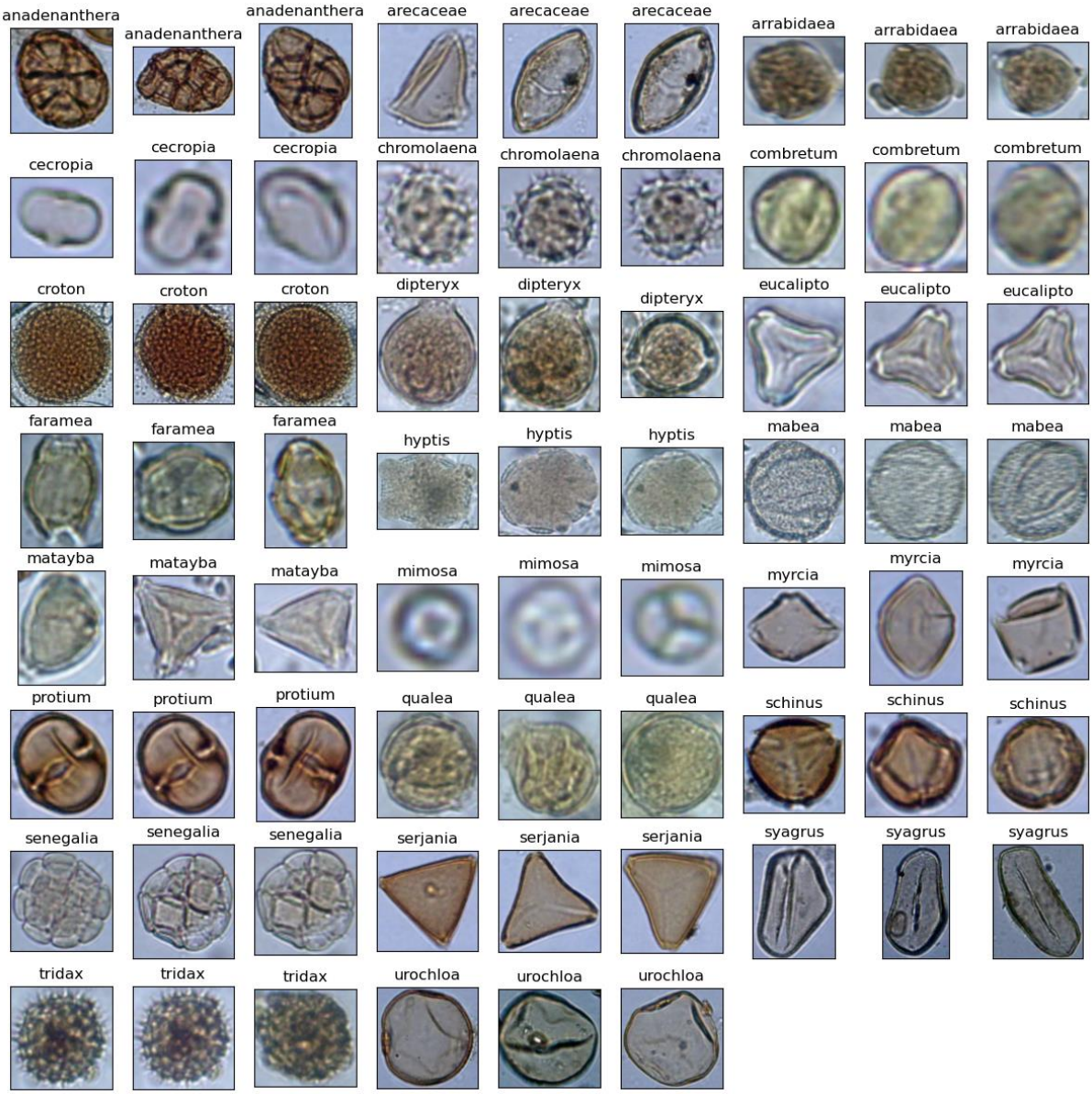


Grouping Image Paths by Class Label in a Dataset

```
path_class  = {key:[] for key in classes.keys()}

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

Visualizing Images by Class Label in a Dataset

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

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

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

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



The code reads the sizes of images in a directory, creates a scatter plot of their dimensions and adds a diagonal line to it

```
size = [cv2.imread(path + name).shape for name in os.listdir(path)]
x, y, _ = zip(*size)

fig = plt.figure(figsize=(5, 5))
# scatter plot
plt.scatter(x,y)
plt.title("Image size scatterplot")

# add diagonal red line
plt.plot([0,800],[0,800], 'r')
```

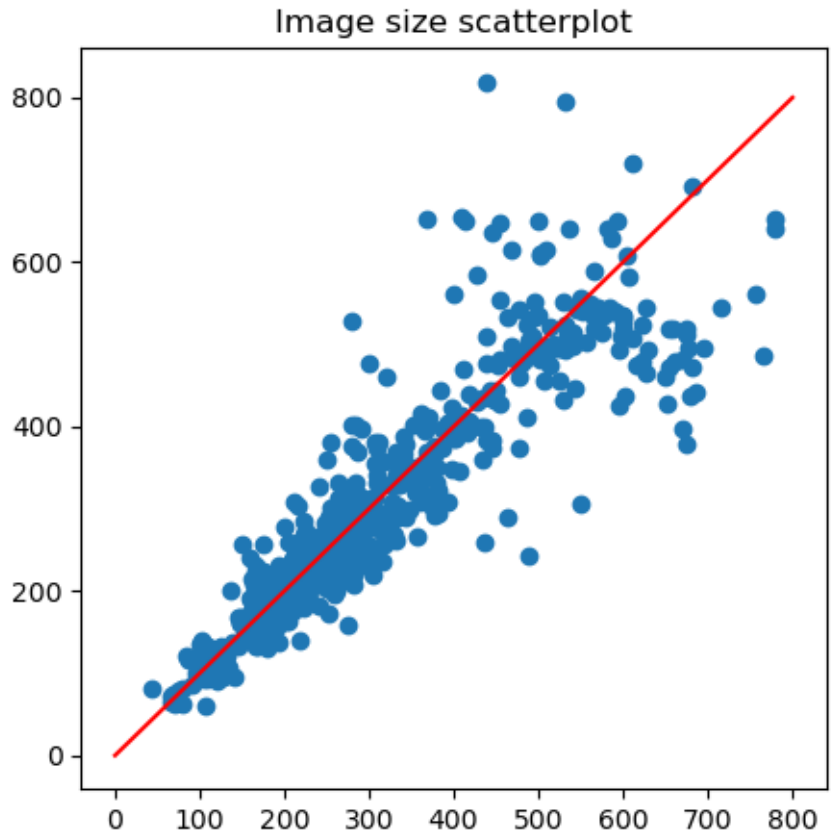


Image Pre-processing

Let's start Image Pre-processing with the help of given activities.

Image Pre-processing

```
def process_img(img, size = (128,128)):
    img = cv2.resize(img, size)  # resize image
    img = img/255                # devide values to 255
    return img
```

The "process_img" function takes an image and a size as input, resizes the image to the specified size using OpenCV's "resize" function, normalizes the pixel values by dividing them by 255, and then returns the processed image. This function can be written in 2 lines as:

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

X = np.array(X)
```

```
from sklearn.preprocessing import LabelEncoder
from keras.utils import np_utils

le = LabelEncoder()
Y_le = le.fit_transform(Y)
Y_cat = np_utils.to_categorical(Y_le, 23)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y_cat, test_size=0.285, stratify=Y_le)
print("Images in each class in Test set: {}".format(np.sum(Y_test, axis =0)))
```

```
Images in each class in Test set: [ 6. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10. 10.
 10. 10. 10. 10. 10.]
```

The code above performs one-hot encoding on the labels using LabelEncoder and np_utils from sci-kit-learn and Keras, respectively. And uses the train_test_split function from sci-kit-learn to split the dataset into training and testing sets.

Model Building

Let's start Model Building with the help given activities.

Training the model

```
input_shape =  X_train[0].shape
output_shape = 23

model = Sequential()
model.add(Conv2D(filters = 16, kernel_size = 3, input_shape = input_shape, activation= 'relu', padding='same'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 32, kernel_size = 2, activation= 'relu', padding='same'))
model.add(MaxPooling2D(pool_size=2))

model.add(Conv2D(filters = 64, kernel_size = 2, activation= 'relu', padding='same'))
model.add(MaxPooling2D(pool_size=2))

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

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

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 (MaxPooling2D)	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 64)	8256
max_pooling2d_2 (MaxPooling2D)	(None, 16, 16, 64)	0

conv2d_3 (Conv2D) (None, 16, 16, 128) 32896

max_pooling2d_3 (MaxPooling (None, 8, 8, 128) 0
2D)

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.

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'])
print('Model is Compiled!')
```

```
Model is Compiled!
```

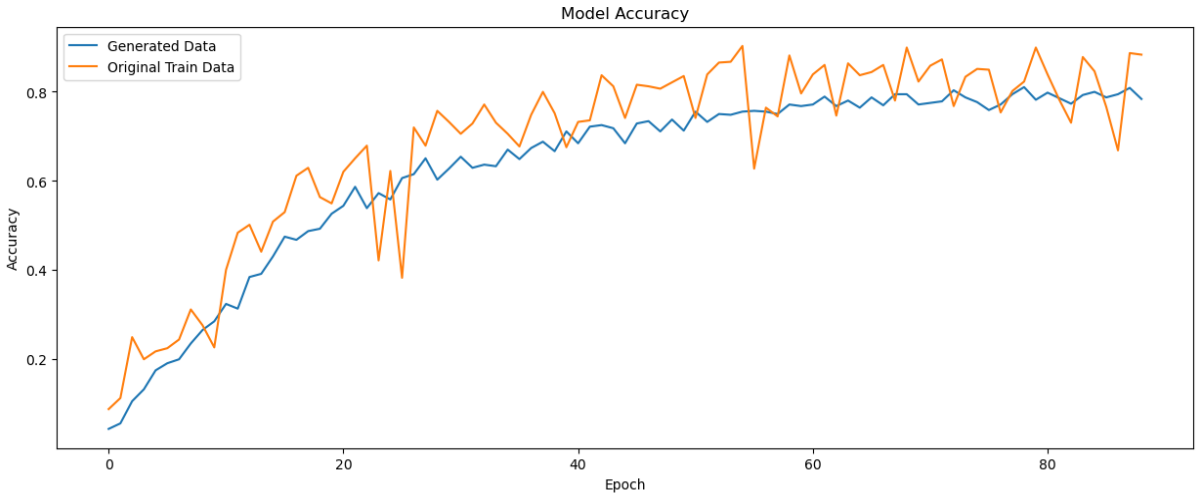
```
datagener = ImageDataGenerator(
                rotation_range= 20,
                width_shift_range = 0.2,
                height_shift_range = 0.2,
                horizontal_flip = True,
                vertical_flip = True,)
# fit data generator
datagener.fit(X_train)
```

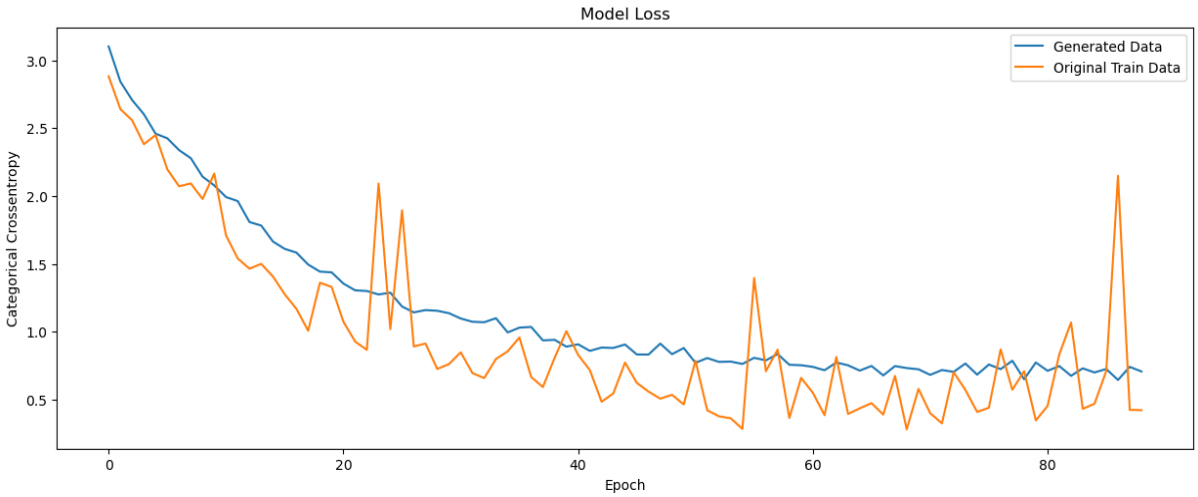
```
batch_size = 4
epochs = 500

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


history = model.fit(
        datagener.flow(X_train, Y_train, batch_size=batch_size),
        batch_size = batch_size,
        steps_per_epoch = len(X_train) // batch_size,
        epochs = epochs,
        validation_data = (X_train, Y_train),
        callbacks = callbecks,
        verbose = 1)
```

```
Epoch 1/500
141/141 [==============================] - 19s 120ms/step - loss: 3.1035 - accuracy: 0.0443 - val_loss: 2.8829 - val_accuracy:
0.0887
Epoch 2/500
141/141 [==============================] - 16s 115ms/step - loss: 2.8440 - accuracy: 0.0567 - val_loss: 2.6414 - val_accuracy:
0.1135
Epoch 3/500
141/141 [==============================] - 16s 115ms/step - loss: 2.7084 - accuracy: 0.1064 - val_loss: 2.5590 - val_accuracy:
0.2500
Epoch 4/500
141/141 [==============================] - 16s 115ms/step - loss: 2.6040 - accuracy: 0.1330 - val_loss: 2.3825 - val_accuracy:
0.2004
Epoch 5/500
141/141 [==============================] - 16s 114ms/step - loss: 2.4599 - accuracy: 0.1755 - val_loss: 2.4496 - val_accuracy:
0.2181
Epoch 6/500
141/141 [==============================] - 16s 113ms/step - loss: 2.4261 - accuracy: 0.1915 - val_loss: 2.1971 - val_accuracy:
0.2252
```

```
model.load_weights(model_path)
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test set accuracy: {}'.format(score[1]))
```

```
Test set accuracy: 0.8495575189590454
```





Save the model

save the model

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

Test the model

```
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
```

```
# 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/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
```

```
# 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/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
```

Application Building

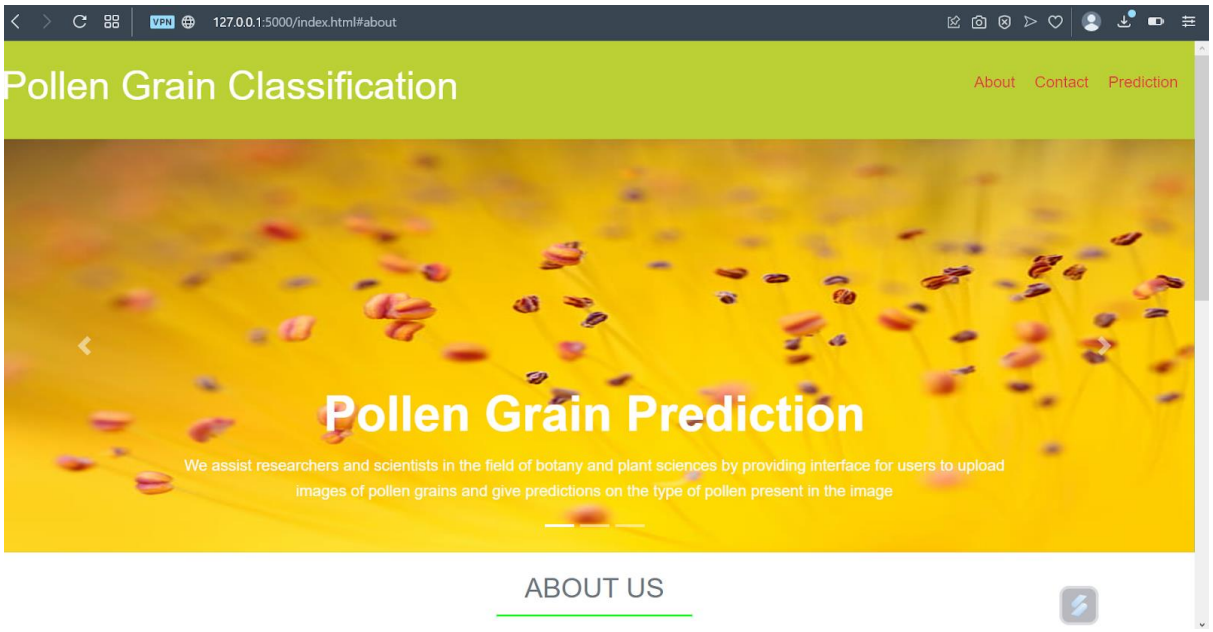
Now that we have trained our model, let us build our flask application which will be running in
our local browser with a user interface.
In the flask application, the input parameters are taken from the HTML page These factors are
then given to the model to know to predict the type of Garbage and showcased on the HTML
page to notify the user. Whenever the user interacts with the UI and selects the “Image” button,

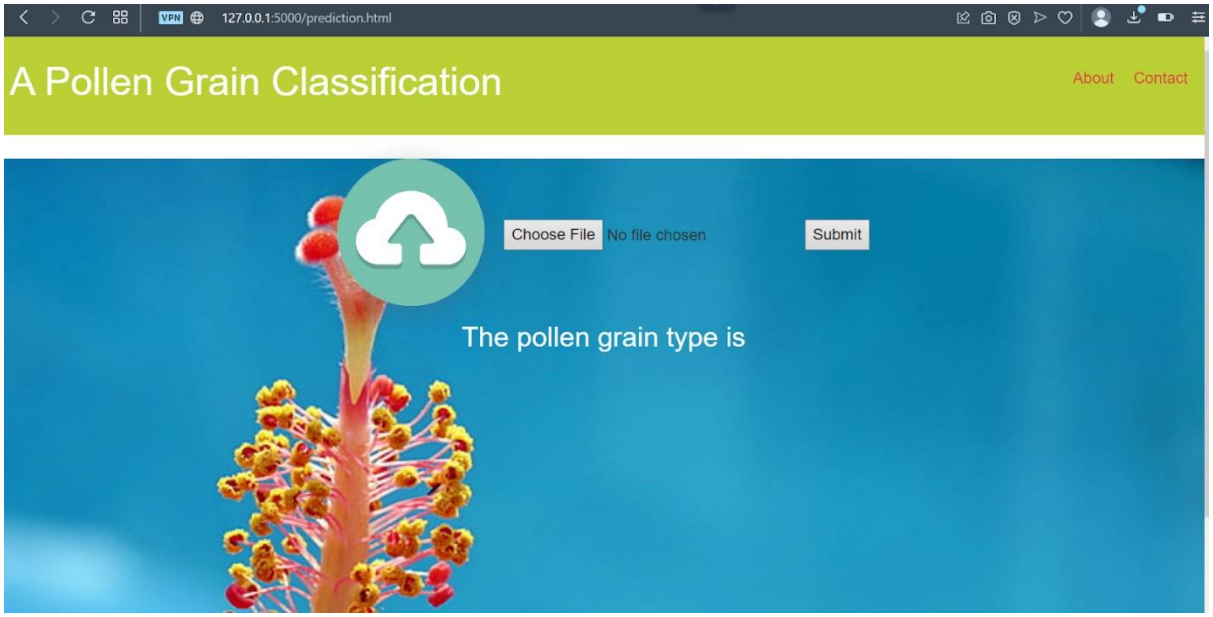
the next page is opened where the user chooses the image and predicts the output.

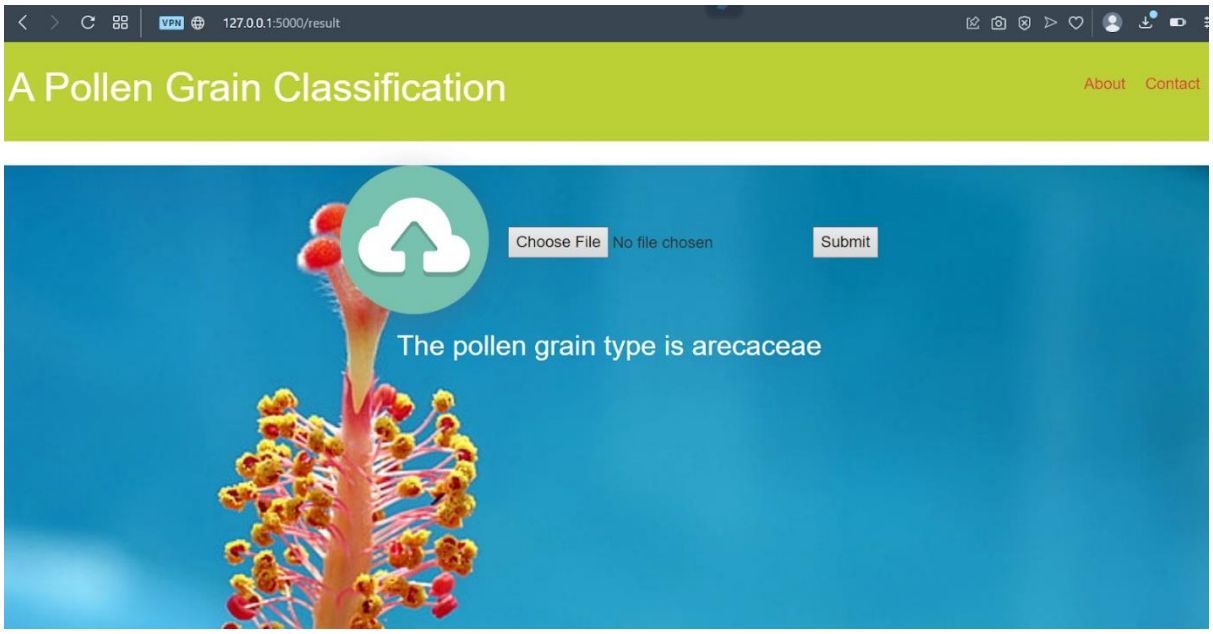
Create HTML Pages

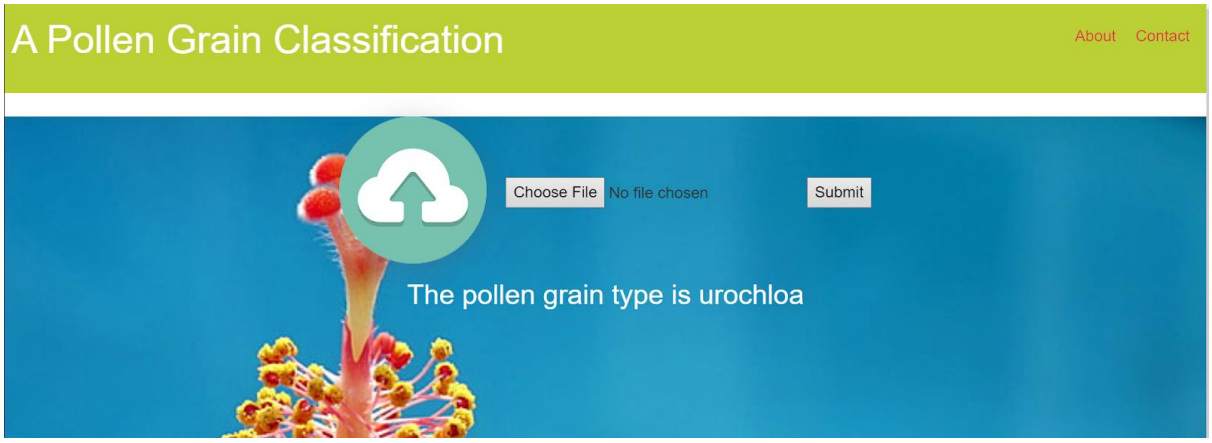
- We use HTML to create the front-end part of the web page.
- Here, we have created 3 HTML pages- index.html, prediction.html, and logout.html
- index.html displays the home page.

- prediction.html displays the prediction page
- logout.html gives the result
- For more information regarding HTML
- https://www.w3schools.com/html/
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.




127.0.0.1:5000/prediction.html
A Pollen Grain Classification
About
Contact
Choose File
No file chosen
Submit
The pollen grain type is


127.0.0.1:5000/result
A Pollen Grain Classification
About
Contact
Choose File
No file chosen
Submit
The pollen grain type is arecaceae



This is a Python script for a Flask web application that loads a pre-trained deep-learning model for image classification and makes predictions on images uploaded by the user. The app has several routes, such as the home page ('/'), the prediction page ('/prediction.html'), and the logout page ('/logout.html'). The main prediction functionality is implemented in the '/result' route, where the uploaded image is loaded, preprocessed, and passed through the model for prediction. The predicted result is then displayed on the prediction page. The app can be run by executing the script, and it will start a local server accessible through a web browser.

```python
import re
import numpy as np
import pandas as pd
import os
import tensorflow as tf
from flask import Flask, app,request,render_template
from keras.models import Model
from keras.preprocessing import image
from tensorflow.python.ops.gen_array_ops import Concat
from keras.models import load_model
#Loading the model
model=load_model(r"model.h5",compile=False)
app=Flask(__name__)

#default home page or route
@app.route('/')
def index():
    return render_template('index.html')

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

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

@app.route('/logout.html')
def logout():
```

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

@app.route('/result',methods=["GET","POST"])
def res():
    if request.method=="POST":
        f=request.files['image']
        basepath=os.path.dirname(__file__) #getting the current path i.e where app.py is present
        #print("current path",basepath)
        filepath=os.path.join(basepath,'uploads',f.filename) #from anywhere in the system we can give image but we wa
        #print("upload folder is",filepath)
        f.save(filepath)
        img = tf.keras.utils.load_img(filepath,target_size=(128,128)) # Reading image
        x = tf.keras.utils.img_to_array(img)
        x = np.expand_dims(x,axis=0) # expanding Dimensions
        pred = np.argmax(model.predict(x)) # Predicting the higher probablity index
        op = ['anadenanthera', 'arecaceae', 'arrabidaea', 'cecropia', 'chromolaena', 'combretum', 'croton', 'dipteryx
        op[pred]
        result = op[pred]
        #result=str(op[ pred[0].tolist().op(1)])
        return render_template('prediction.html',pred=result)

""" Running our application """
if __name__ == "__main__":
    app.run(debug=True)
```

To run this Flask application, simply navigate to the project directory in the terminal and run the command "python app.py". This will start the Flask server, and you can access the web application by visiting the local host address in your web browser. Once you upload an image and submit the form, the application will use the trained model to predict the species of the plant in the image and display the result on the page.