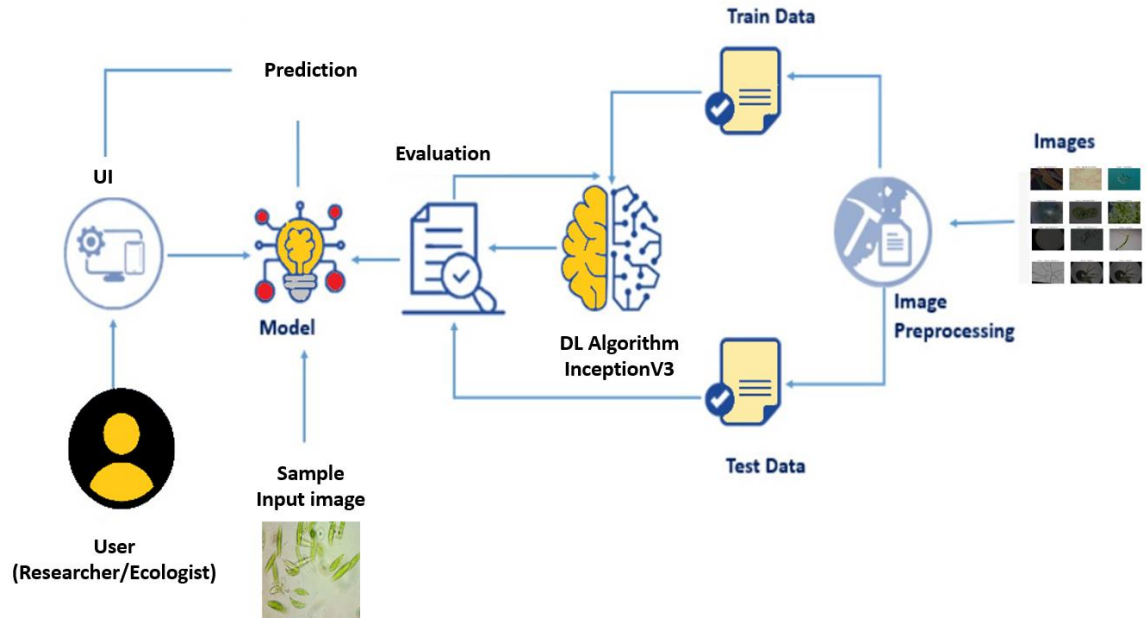
Microbe Mapper: Visual Recognition of Micro-organisms

Introduction:

Microorganisms such as protozoa and bacteria play very important roles in many practical domains, like agriculture, industry and medicine. To explore functions of different categories of microorganisms is a fundamental work in biological studies, which can assist biologists and related scientists to get to know more properties, habits and characteristics of these tiny but obligatos living beings. However, taxonomy of microorganisms (microorganism classification) is traditionally investigated through morphological, chemical or physical analysis, which is time and money consuming. In order to overcome this, since the 1970s CBMIA methods are used to classify microorganisms into different categories using multiple artificial intelligence approaches, such as machine vision, pattern recognition and machine learning algorithms. With the advancement of technology, many new techniques of Deep learning have contributed towards classification of image in a more efficient way such as ResNet, VGG16, Inception V3 etc. Here in the given project, we are using the Inception V3 to classify the microorganisms into their original classes.

Technical Architecture:



Pre-requisites:

To complete this project, you must require the following software's, 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, Spyder, Visual Studio Code. For this project, we will be using Jupyter notebook and Spyder

To install Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Link: Click here to watch the video

1. To build Machine learning models you must require the following packages

- Numpy: o It is an open-source numerical Python library. It contains a multidimensional array and matrix data structures and can be used to perform mathematical operations
- Scikit-learn: o It is a free machine learning library for Python. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy
- Flask: Web framework used for building Web applications 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 tensorflow==2.3.2" and click enter.
    - Type "pip install keras==2.3.1" and click enter. ✔ Type "pip install Flask" and click enter.

Deep Learning Concepts

CNN: a convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery.

CNN Basic

Flask: Flask is a popular Python web framework, meaning it is a third-party Python library used for developing web applications.

Flask Basics

If you are using Pycharm IDE, you can install the packages through the command prompt and follow the same syntax as above.

Project Objectives:

By the end of this project, you will:

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

Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image analysed by the model which is integrated with flask application.
- Inceptionv3 Model analyse the image, then prediction is showcased on the Flask UI.

To accomplish this, we have to complete all the activities and tasks listed below

Data Collection.

- Create Train and Test Folders.

Data Pre-processing.

- Import the ImageDataGenerator library
- Configure ImageDataGenerator class
- Apply ImageDataGenerator functionality to Trainset and Test set

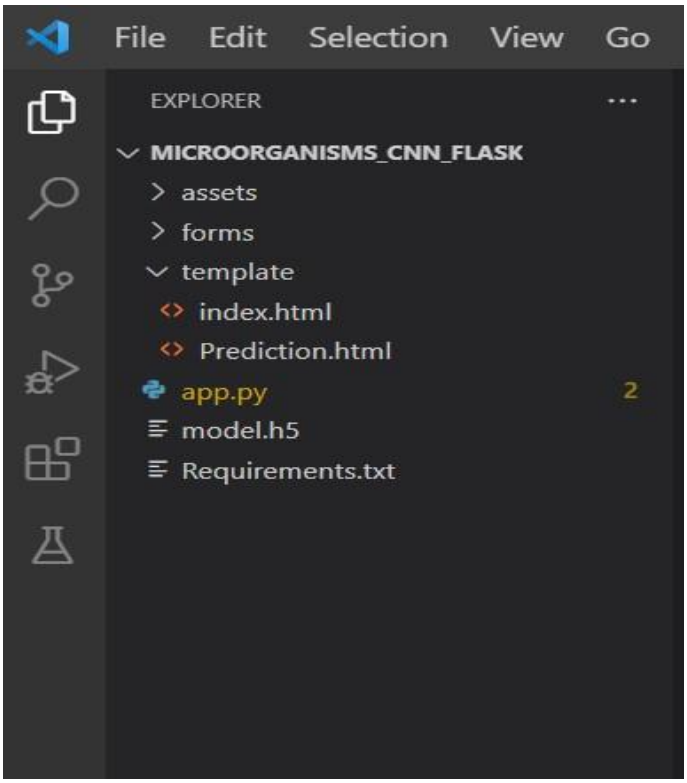
Model Building

- Import the model building Libraries
- Initializing the model
- Adding Input Layer
- Adding Hidden Layer
- Adding Output Layer
- Configure the Learning Process

Training and testing the model

- Save the Model
- Application Building
- Create an HTML file
- Build Python Code

Project Structure:
Create a Project folder which contains files as shown below



- The Dataset folder contains the training and testing images for training our model.
- We are building a Flask Application that needs HTML pages stored in the templates folder and a python script app.py for server-side scripting
- we need the model which is saved and the saved model in this content is a model.h5 templates folder contains base.html, index.html pages.

Milestone 1: Collection of Data

Data Collection Collect images of micro-organisms then organized into subdirectories based on their respective names as shown in the project structure. Create folders of types of microbes that need to be recognized.
The given dataset has 8 different types of folders given for Micro Organism they are following:

- Amoeba
- Euglena
- Hydra
- Paramecium
- Rod bacteria

- Spherical bacteria
- Spiral bacteria
- Yeast

Download the Dataset-
https://www.kaggle.com/datasets/mdwaquarazam/microorganism-image-classification

Milestone 2: Image Pre-processing

In this milestone we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Activity 1: Import the ImageDataGenerator library
Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset. The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class. Let us import the ImageDataGenerator class from TensorFlow Keras

```
# making all necessary imports

import os
import keras
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import plotly.express as px
import matplotlib.pyplot as plt
from keras.models import Sequential, load_model
from keras.layers import GlobalAvgPool2D as GAP, Dense, Dropout
from keras.callbacks import EarlyStopping as ES, ModelCheckpoint as MC
from tensorflow.keras.applications import ResNet50V2, ResNet50, InceptionV3, Xception
from matplotlib import RcParams
import warnings
warnings.filterwarnings('ignore')
```

Activity 2: Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation

```
gen = ImageDataGenerator(rescale=1./255, rotation_range=10, horizontal_flip=True, brightness_range=[0.3,0.8], validation_sp
```

There are five main types of data augmentation techniques for image data; specifically:

Image shifts via the width_shift_range and height_shift_range arguments.
The image flips via the horizontal flip and vertical flip arguments. Image rotations via the rotation range argument
Image brightness via the brightness range argument.
Image zoom via the zoom range argument. An instance of the ImageDataGenerator class can be constructed for train and test.

Activity 3: Apply ImageDataGenerator functionality to Trainset and Test set

Let us apply ImageDataGenerator functionality to Trainset and Test set by using the following code. For Training set using flow_from_directory function.
This function will return batches of images from the different classes as 'Amoeba': 0, 'Euglena': 1, 'Hydra': 2, 'Paramecium': 3, 'Rod_bacteria': 4, 'Spherical_bacteria': 5, 'Spiral_bacteria': 6, 'Yeast': 7

Arguments:

directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored. batch_size: Size of the batches of data which is 64. target_size: Size to resize images after they are read from disk.
class_mode:

- 'int': means that the labels are encoded as integers (e.g., for sparse_categorical_crossentropy loss).
- 'categorical' means that the labels are encoded as a categorical vector (e.g., for categorical_crossentropy loss).
- 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g., for binary_crossentropy).
- None (no labels).

```
train_ds= gen.flow_from_directory(
    '/content/Micro_Organism/',
    batch_size=128,
    shuffle=True,
    class_mode='binary',
    target_size=(256,256),
    subset='training'
)
valid_ds= gen.flow_from_directory(
    '/content/Micro_Organism/',
    batch_size=64,
    shuffle=True,
    class_mode='binary',
    target_size=(256,256),
    subset='validation'
)
```

```
Found 714 images belonging to 8 classes.
Found 75 images belonging to 8 classes.
```

Milestone 3: Model Building

Now it's time to build our Convolutional Neural Networking which contains an input layer along with the convolution, max-pooling, and finally an output layer.

Activity 1: Importing the Model Building Libraries

Importing the necessary libraries

```
# making all necessary imports

import os
import keras
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from keras.preprocessing.image import ImageDataGenerator
import plotly.express as px
import matplotlib.pyplot as plt
from keras.models import Sequential, load_model
from keras.layers import GlobalAvgPool2D as GAP, Dense, Dropout
from keras.callbacks import EarlyStopping as ES, ModelCheckpoint as MC
from tensorflow.keras.applications import ResNet50V2, ResNet50, InceptionV3, Xception
from matplotlib import RcParams
import warnings
warnings.filterwarnings('ignore')
```

Activity 2: Exploratory Data Analysis

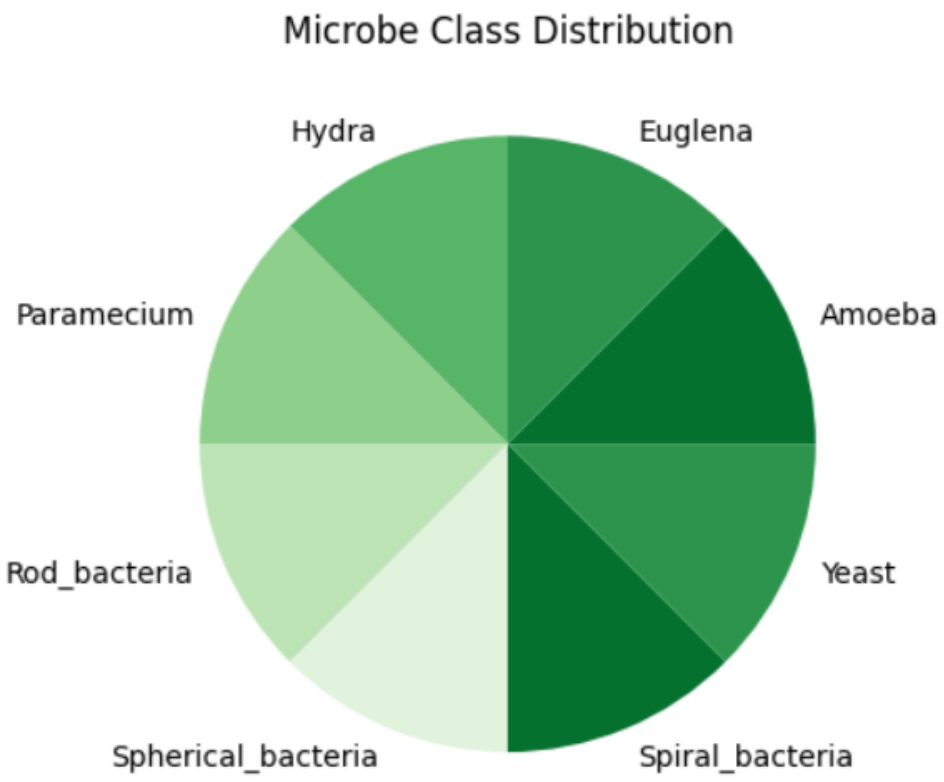
```
class_names = sorted(os.listdir('/content/Micro_Organism/'))
n_classes = len(class_names)
print('class_names: ',class_names)
print('number_of_classes: ',n_classes)
```

```
class_names:  ['Amoeba', 'Euglena', 'Hydra', 'Paramecium', 'Rod_bacteria', 'Spherical_bacteria', 'Spiral_bacteria', 'Yeast']
number_of_classes:  8
```

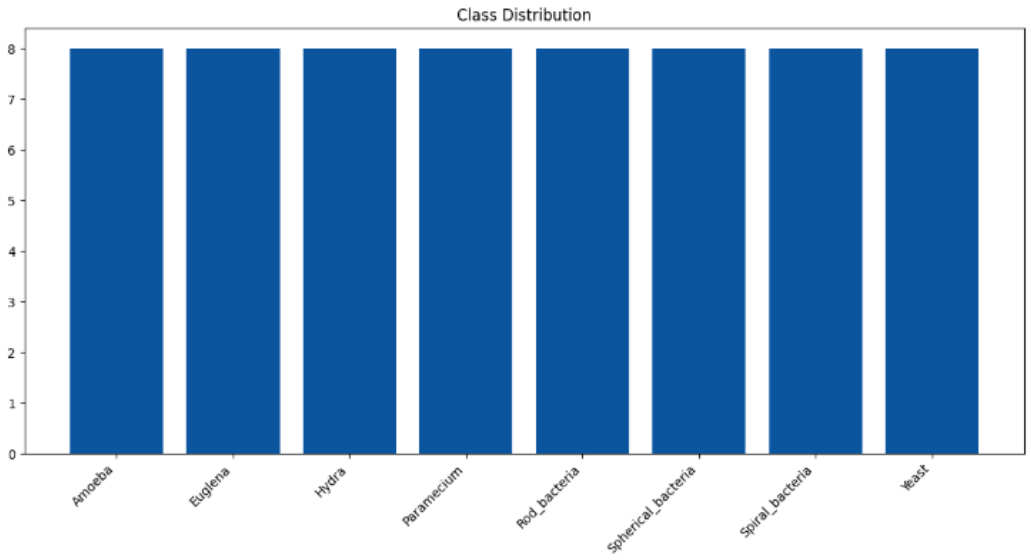
```
# fig = px.pie(names=class_names, title="Class Distribution")
# fig.update_layout({'title':{'x':0.45}})
# fig.show()
```

Activity 3: Visualisation of the data

```
# fig = px.pie(names=class_names, title="Class Distribution")
# fig.update_layout({'title':{'x':0.45}})
# fig.show()

# plt.pie(names=class_names, labels=class_names)
# plt.title("Class Distribution", loc="center", pad=20)
# plt.show()
sns.set_palette('Greens_r')
plt.pie([1] * len(class_names), labels=class_names)
plt.title("Microbe Class Distribution", loc="center")
plt.show()
```

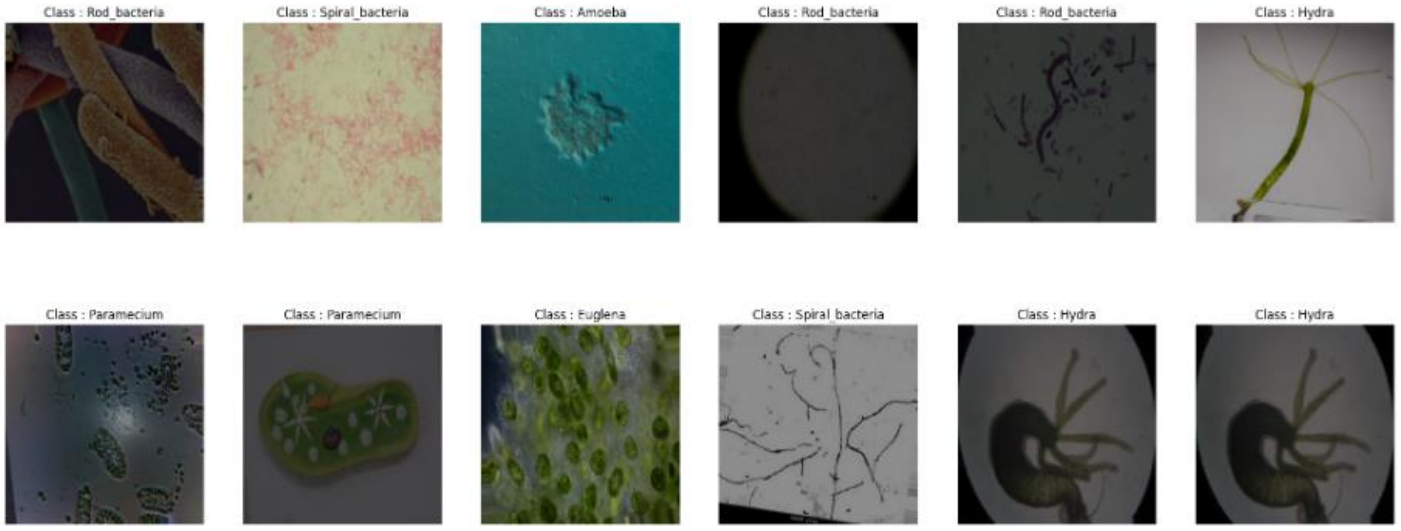


```
sns.set_palette('Blues_r')
plt.figure(figsize=(14, 6))
plt.bar(class_names, n_classes)
plt.title("Class Distribution")
plt.xticks(rotation=45, ha='right')
# Show the plot
plt.show()
```




```
def show_images(data, GRID=[2,6], model=None, size=(25,10)):

    # The plotting configurations
    n_rows, n_cols = GRID
    n_images = n_rows * n_cols
    plt.figure(figsize=size)

    # Data for visualization
    images, labels = next(iter(data)) # This process can take a little time because of the large batch size

    # Iterate through the subplots.
    for i in range(1, n_images+1):

        # Select a random data
        id = np.random.randint(len(images)) # This is a dynamic function because for validation data and training data, the
        image, label = images[id], class_names[int(labels[id])]

        # Plot the sub plot
        plt.subplot(n_rows, n_cols, i)
        plt.imshow(image)
        plt.axis('off')
        # If model is available make predictions.
        if model is not None:
            pred = class_names[np.argmax(model.predict(image[np.newaxis,...]))]
            title = f"Class : {label}\nPred : {pred}"
        else:
            title = f"Class : {label}"

        plt.title(title)
    plt.show()
```

```
In [14]: show_images(data=train_ds)
```



Activity 4: Initializing the model

Keras has 2 ways to define a neural network:

Sequential
Function API
The Sequential class is used to define linear initializations of network layers which then, collectively, constitute a model. In our example below, we will use the Sequential constructor to create a model, which will then have layers added to it using the add() method.

Activity 5: Configure the Learning Process

The compilation is the final step in creating a model. Once the compilation is done, we can move on to the training phase. The loss function is used to find errors or deviations in the learning process. Keras requires a loss function during the model compilation process.
Optimization is an important process that optimizes the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
Metrics are used to evaluate the performance of your model. It is similar to the loss function, but not used in the training process

Activity 6: Train The model

Now, let us train our model with our image dataset. The model is trained for 30 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.
fit_generator functions used to train a deep learning neural network

Arguments:
steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.
Epochs: an integer and number of epochs we want to train our model for.
validation_data can be either:

- an inputs and targets list
- a generator
- an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size

```
# Give you a model, a name
name = "inception-v3"

# Base model
base = InceptionV3(input_shape=(256,256,3), include_top=False)
base.trainable = False

# Model Architecture
model = Sequential([
    base, GAP(),
    Dense(256, kernel_initializer='he_normal', activation='relu'),
    Dropout(0.2),
    Dense(n_classes, activation='softmax')
])

# Callbacks
cbs = [ES(patience=3, restore_best_weights=True), MC(name + ".h5", save_best_only=True)]

# Compile Model
opt = tf.keras.optimizers.Adam(learning_rate=1e-3) # Higher than the default learning rate 1e-3
model.compile(loss='sparse_categorical_crossentropy', optimizer=opt, metrics=['accuracy'])

# Training
history = model.fit(train_ds, validation_data=valid_ds, epochs=50, callbacks=cbs)
```

Analysing the trained model by looking at the accuracy through the epochs:

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/inception_v3_weights_tf_dim_ordering_tf_kernels_notop.h5
87910968/87910968 [==============================] - 1s 0us/step
Epoch 1/50
6/6 [==============================] - 43s 4s/step - loss: 2.3698 - accuracy: 0.2101 - val_loss: 1.6713 - val_accuracy: 0.3867
Epoch 2/50
6/6 [==============================] - 25s 4s/step - loss: 1.6406 - accuracy: 0.4272 - val_loss: 1.3934 - val_accuracy: 0.5067
Epoch 3/50
6/6 [==============================] - 22s 3s/step - loss: 1.2546 - accuracy: 0.5952 - val_loss: 1.2651 - val_accuracy: 0.6267
Epoch 4/50
6/6 [==============================] - 21s 3s/step - loss: 1.0707 - accuracy: 0.6625 - val_loss: 1.0814 - val_accuracy: 0.6133
Epoch 5/50
6/6 [==============================] - 18s 3s/step - loss: 0.9312 - accuracy: 0.6863 - val_loss: 1.1289 - val_accuracy: 0.6000
Epoch 6/50
6/6 [==============================] - 22s 4s/step - loss: 0.8170 - accuracy: 0.7269 - val_loss: 1.0833 - val_accuracy: 0.6400
Epoch 7/50
6/6 [==============================] - 20s 3s/step - loss: 0.7743 - accuracy: 0.7479 - val_loss: 1.0415 - val_accuracy: 0.6533
Epoch 8/50
6/6 [==============================] - 21s 4s/step - loss: 0.6723 - accuracy: 0.7843 - val_loss: 1.0062 - val_accuracy: 0.7067
Epoch 9/50
6/6 [==============================] - 22s 4s/step - loss: 0.6172 - accuracy: 0.7983 - val_loss: 1.0328 - val_accuracy: 0.6800
Epoch 10/50
6/6 [==============================] - 18s 3s/step - loss: 0.5701 - accuracy: 0.8179 - val_loss: 1.0517 - val_accuracy: 0.6400
Epoch 11/50
6/6 [==============================] - 19s 3s/step - loss: 0.4999 - accuracy: 0.8473 - val_loss: 1.0725 - val_accuracy: 0.6667
```

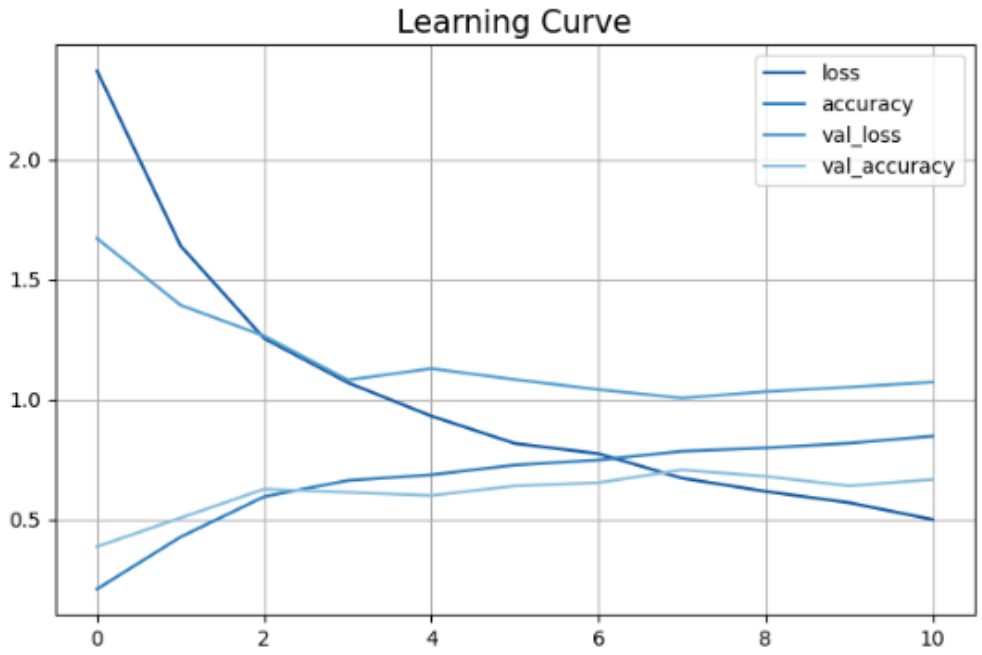
Visualising Model Summary:

```
model.summary()
```

```
Model: "inception_v3"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 input_2 (InputLayer)        [(None, 299, 299, 3)]     0         []

 conv2d_94 (Conv2D)          (None, 149, 149, 32)      864       ['input_2[0][0]']

 batch_normalization_94 (Ba  (None, 149, 149, 32)      96        ['conv2d_94[0][0]']
 tchNormalization)

 activation_94 (Activation)  (None, 149, 149, 32)      0         ['batch_normalization_94[0][0]
                                                                 ']

 conv2d_95 (Conv2D)          (None, 147, 147, 32)      9216      ['activation_94[0][0]']

 batch_normalization_95 (Ba  (None, 147, 147, 32)      96        ['conv2d_95[0][0]']
 tchNormalization)

 avg_pool (GlobalAveragePoo  (None, 2048)              0         ['mixed10[0][0]']
 ling2D)

 predictions (Dense)         (None, 1000)              2049000   ['avg_pool[0][0]']

==================================================================================================
Total params: 23851784 (90.99 MB)
Trainable params: 23817352 (90.86 MB)
Non-trainable params: 34432 (134.50 KB)
__________________________________________________________________________________________________
```

Visualising the loss and accuracy of the trained model.

```
pd.DataFrame(history.history).plot(figsize=(8,5))
plt.title("Learning Curve", fontsize=15)
plt.grid()
plt.show()
```



Activity 7: Save the Model

The model is saved with .h5 extension as follows An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.

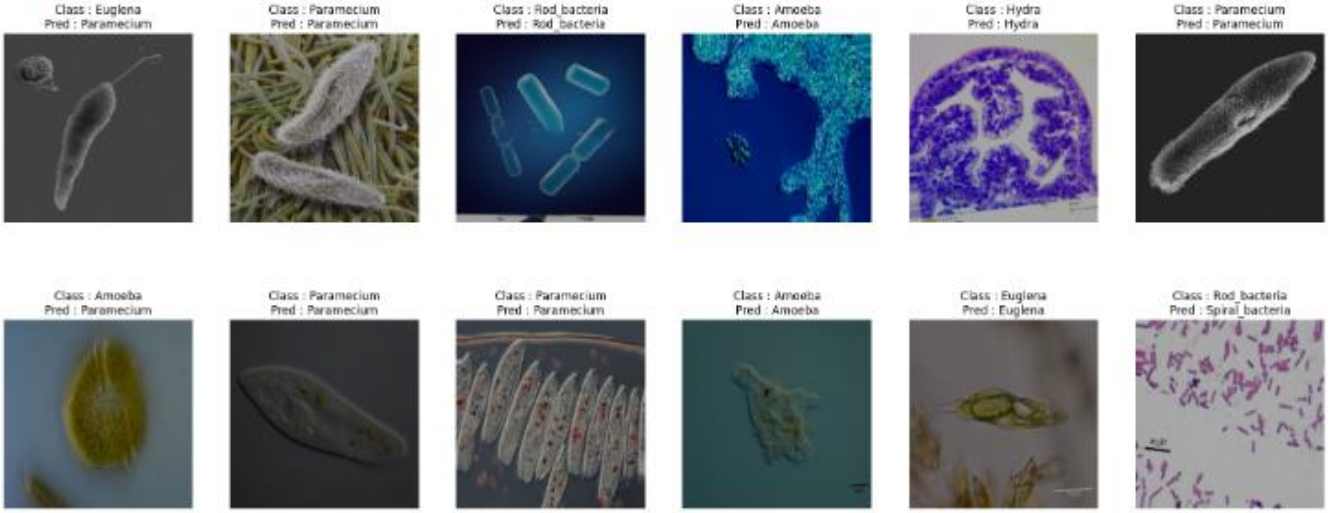
```
model.save('microbeclassification.h5')
```

Activity 8: Test The model

Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data. Load the saved model using load_model

```
show_images(data=valid_ds, model=model)
```

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 28ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
```



Taking an image as input and checking the results. **Displaying 4 Test Results.**
For the sake of testing multiple images, an additional function was written.

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.preprocessing import image

def predict(img_path, model):
    img = image.load_img(img_path, target_size=(256, 256))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x /= 255.0  # Normalize the pixel values

    predictions = model.predict(x)

    # Decode the predictions
    class_names = ['Amoeba', 'Euglena', 'Hydra', 'Paramecium', 'Rod_bacteria', 'Spherical_bacteria', 'Spiral_bacteria', 'Yeast']
    predicted_class = class_names[np.argmax(predictions)]

    # Display the image and predicted class
    plt.imshow(img)
    plt.title(f"Predicted Class: {predicted_class}")
    plt.axis('off')

def show_images_in_subplot(paths, model, cols=3, rows=None):
    if rows is None:
        rows = len(paths) // cols + (len(paths) % cols > 0)  # Calculate the number of rows dynamically

    plt.figure(figsize=(15, 7))

    for i, img_path in enumerate(paths, 1):
        plt.subplot(rows, cols, i)
        predict(img_path, model)

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

paths=['/content/spherical.jpg','/content/amoeba.jpg','/content/paramoecium.jpg','/content/Euglena.jpeg']
show_images_in_subplot(paths, model)
```

Milestone 4: 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.

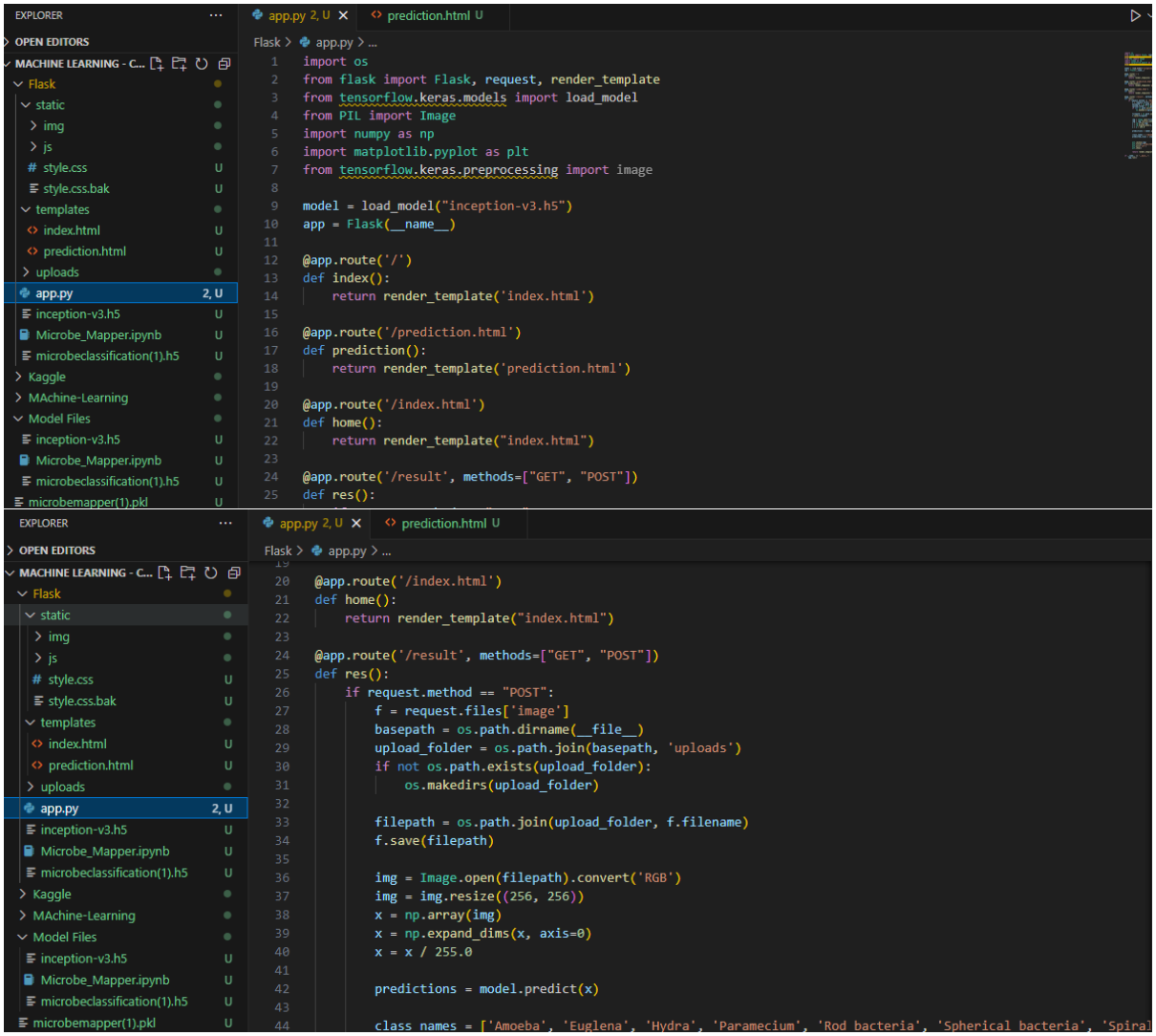
In the flask application, the input parameters are taken from the HTML page These factors are then given to the model to know to predict the type of Garbage and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the "Image" button, the next page is opened where the user chooses the image and predicts the output.

Activity 1: Create HTML Pages

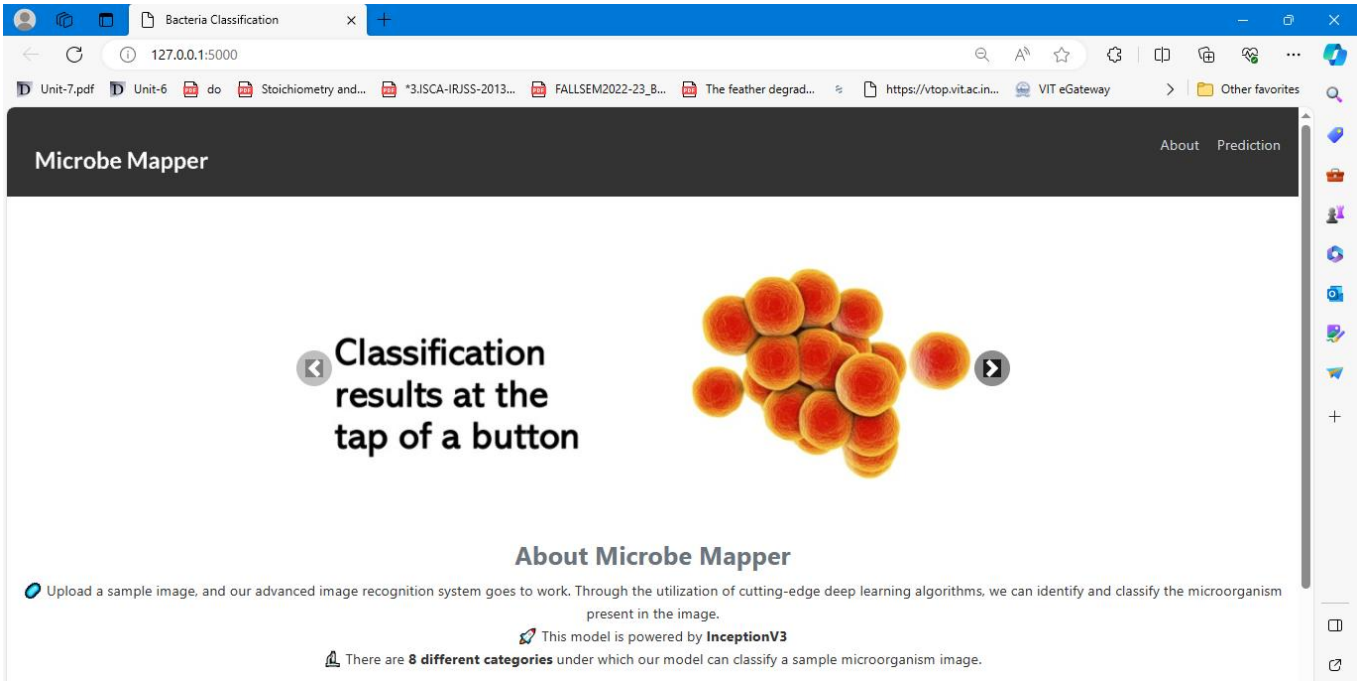
- We use HTML to create the front end part of the web page.
- Here, we have created 2 HTML pages- index.html, prediction.html,
- Intro.html displays an introduction about the project
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

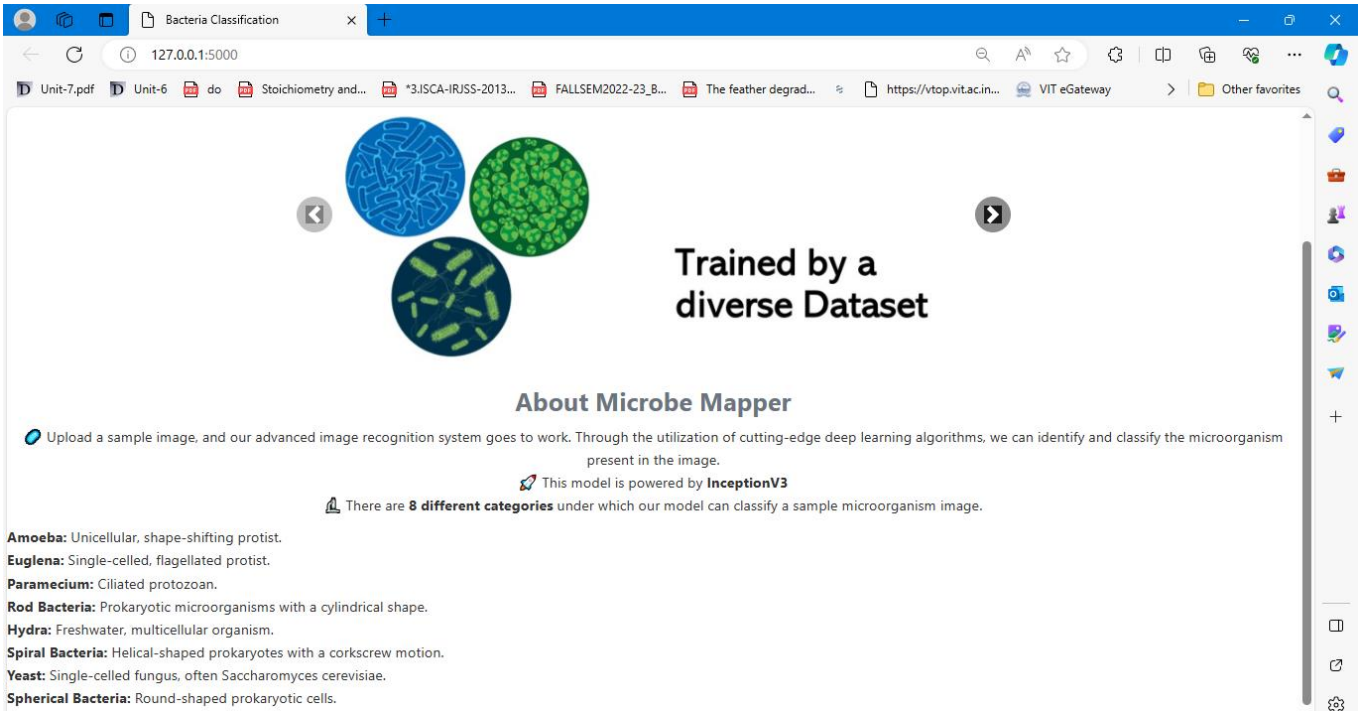
```
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [21/Nov/2023 11:37:39] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2023 11:37:39] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/Nov/2023 11:37:39] "GET /static/img/picture2.png HTTP/1.1" 304 -
127.0.0.1 - - [21/Nov/2023 11:37:39] "GET /static/img/first.png HTTP/1.1" 304 -
127.0.0.1 - - [21/Nov/2023 11:37:39] "GET /static/img/second.png HTTP/1.1" 304 -
127.0.0.1 - - [21/Nov/2023 11:37:39] "GET /static/img/third.png HTTP/1.1" 304 -
127.0.0.1 - - [21/Nov/2023 11:37:40] "GET /favicon.ico HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 11:37:43] "GET /prediction.html HTTP/1.1" 200 -
127.0.0.1 - - [21/Nov/2023 11:37:43] "GET /static/style.css HTTP/1.1" 304 -
127.0.0.1 - - [21/Nov/2023 11:37:43] "GET /static/img/testimonials/stages.png HTTP/1.1" 404 -
127.0.0.1 - - [21/Nov/2023 11:37:43] "GET /static/img/dummy/dr.jpg HTTP/1.1" 304 -
```

Create app.py (Python Flask) file: -

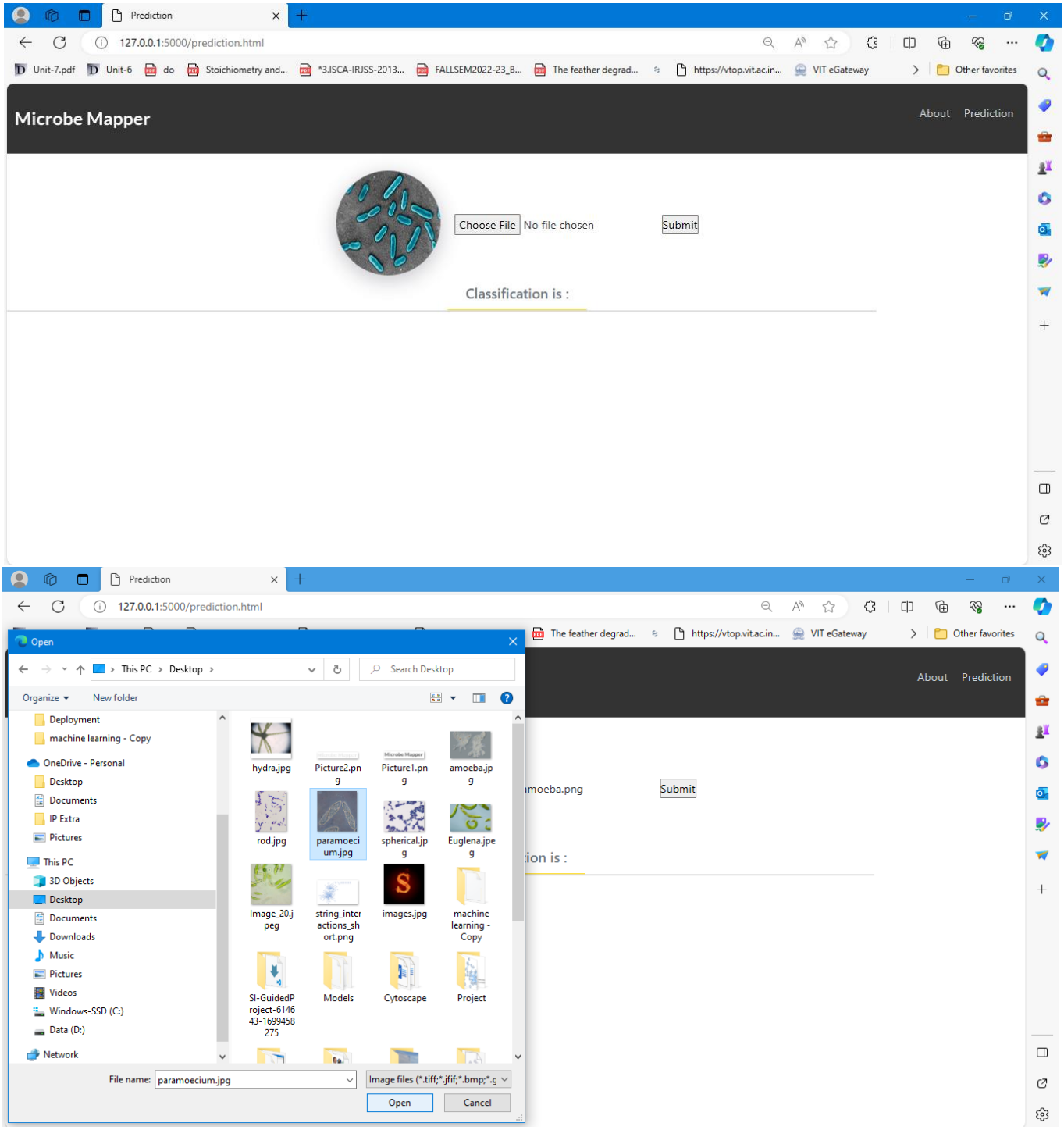


A slider carousel and About Section is displayed on the index.html page:

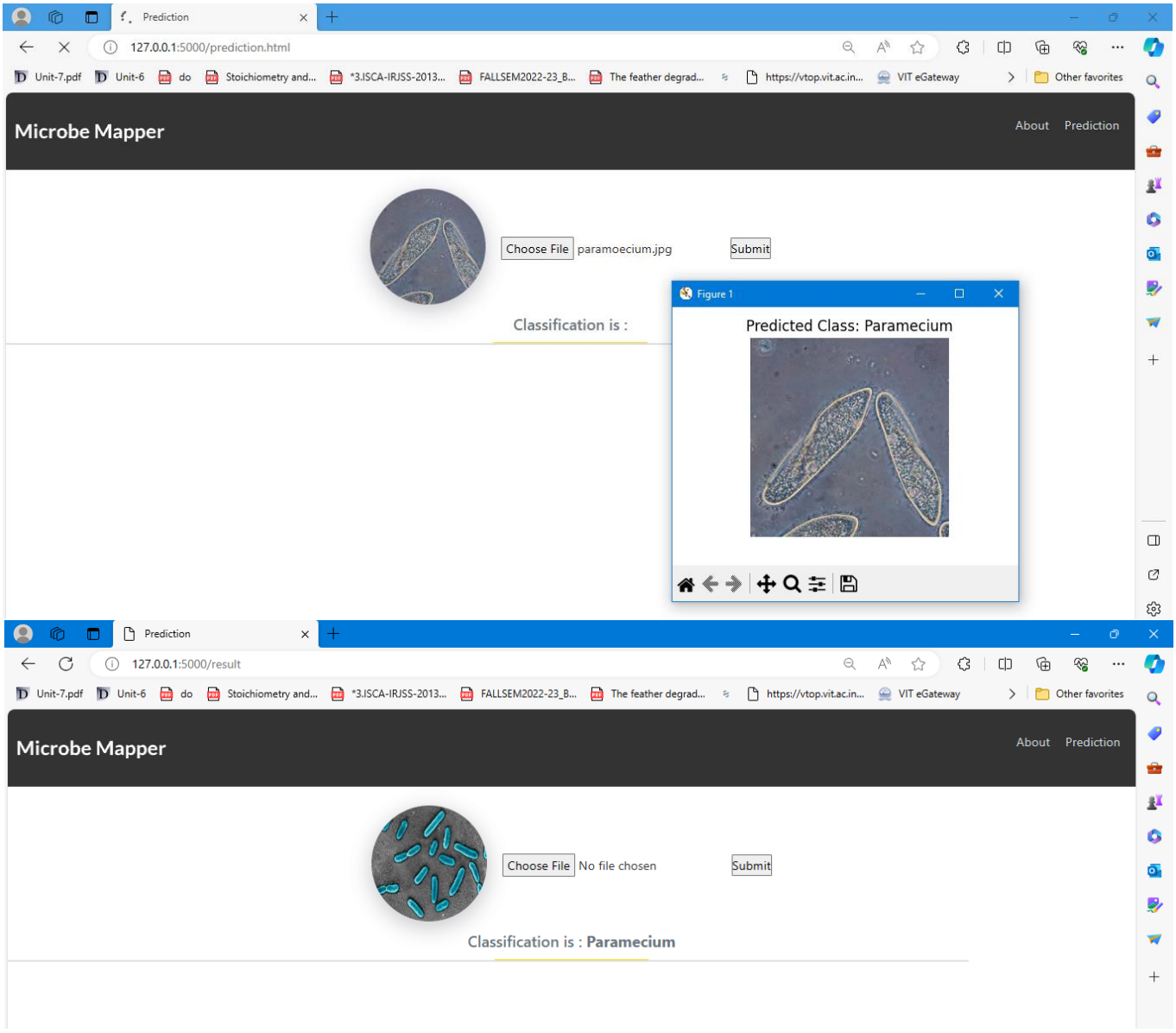




Finally, the prediction page to upload sample image



Final Predicted Output (after I click on the Submit Button) is displayed as follows:



The prediction is correct!!