Curated Colon Disease Classification using Deep learning

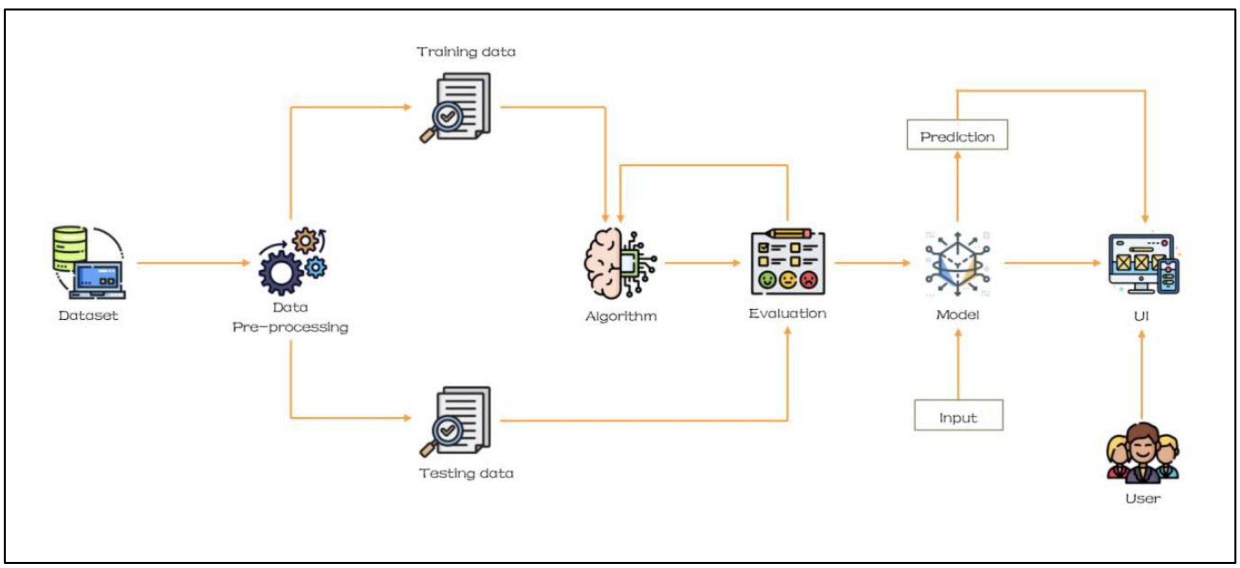
Introduction:

The colon, also known as the large intestine, is an important part of the digestive system. It absorbs water and electrolytes from the digestive tract and eliminates waste products from the body. However, like other organs, it can be affected by various diseases that can disrupt its normal function. The WCE Curated Colon Disease Classification is a classification system that categorizes diseases and abnormalities of the colon based on findings from wireless capsule endoscopy (WCE). WCE is a minimally invasive diagnostic tool that uses a small capsule with a camera to capture images of the inside of the colon as it passes through.

The colon diseases and abnormalities that can be classified using the WCE Curated Colon Disease Classification include polyps, ulcerative colitis & esophagitis, and other pathologies. These conditions can range from benign to malignant, and their management and treatment depend on the specific diagnosis. In the context of colon disease classification, deep learning can be used to analyze images captured by wireless capsule endoscopy (WCE) and classify them into different disease categories. This approach has shown promising results, with some studies reporting high accuracy rates for colon disease diagnosis using deep learning algorithms.

Overall, the WCE Curated Colon Disease Classification is an important tool for clinicians to accurately diagnose and manage colon diseases, leading to improved patient care and outcomes.

Technical Architecture:



Prerequisites:

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,
QtConsole, Spyder, Glueviz, Orange, RStudio, Visual Studio Code. For this project, we will be using Jupyter notebook and VS code.
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:
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:
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==2.12.0” and click enter.
- Type “pip install keras==2.12.0” 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

VGG16: VGG16 is a deep convolutional neural network architecture for image classification, consisting of 16 layers with small convolution filters.

VGG16:-

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 preprocessing techniques.
- know how to build a web application using the Flask framework.

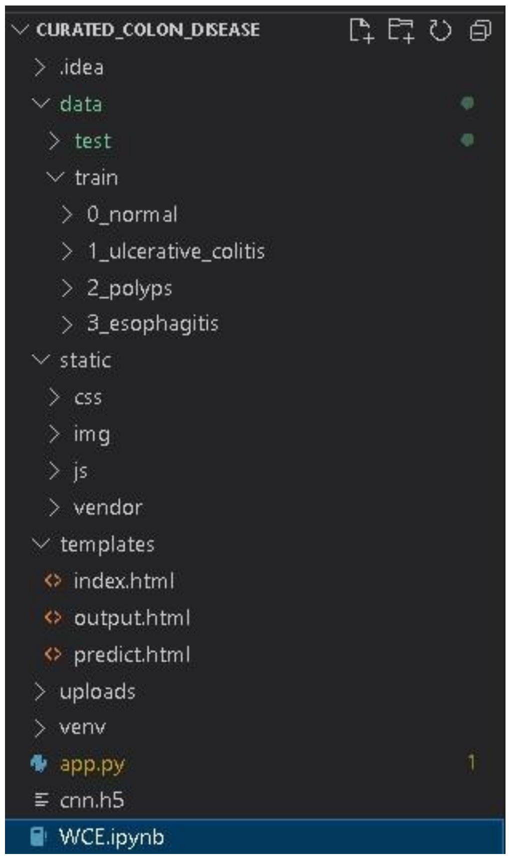
Project Flow:

- The user interacts with the UI (User Interface) to choose the image.
- The chosen image is analysed by the model which is integrated with flask application.
- CNN Models analyse the image, then prediction is showcased on the Flask UI.
- To accomplish this, we must complete all the activities and tasks listed below:
    - Data Collection.
    - Create Train and Test Folders.
    - Data Preprocessing
    - Import the ImageDataGenerator library
    - Configure ImageDataGenerator class
    - ApplyImageDataGenerator functionality to Train dataset and Test dataset
    - Model Building
    - Import the model building Libraries.
    - Importing the VGG16.
    - Initializing the model
    - Adding Fully connected 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 Data 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 **CNN.**
- **h5** templates folder contains index.html, predict.html & output.html pages.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem.

Refer Project Description

Activity 2: Business requirements

Here are some potential business requirements for Colon Disease Prediction:

a. **Accurate Prediction:**

The predictor must be able to accurately classify the images of Colon Diseases. So that there will be no misclassification which decreases the accuracy of the model.

b. **Real-time data acquisition:**

The predictor must be able to acquire real-time data from the various sources. The data acquisition must be seamless and efficient to ensure that the predictor is always up-to-date with the latest information.

c. **User-friendly interface:**

The predictor must have a user-friendly interface that is easy to navigate and understand. The interface should present the results of the predictor in a clear and concise manner.

d. **Report generation:**

Generate a report outlining the predicted Colon Diseases. By analysing data and applying advanced algorithms, the project generates detailed reports that include key findings, disease classifications, and relevant insights. The report should be presented in a clear and concise manner, with appropriate insights to help patients confirm their results.

Activity 3: Literature Survey:

A Studies assess the clinical effectiveness of WCE in various gastrointestinal disorders, such as Crohn's disease, small bowel tumors, obscure gastrointestinal bleeding, and celiac disease. Research focuses on evaluating the diagnostic yield and accuracy of WCE compared to traditional endoscopic procedures. This includes studies that analyze the ability of WCE to detect abnormalities in different parts of the gastrointestinal tract.

Advances in image processing and analysis techniques are discussed in the literature to enhance the quality of images captured by the capsule and improve the accuracy of diagnostic outcomes.

Activity 4: Social or Business Impact.
The Curated Colon Disease Prediction project can have both social and business impacts.

Social Impact:
Accurate and reliable classification of colon diseases can contribute to improved healthcare outcomes. It can help healthcare professionals make more informed decisions about diagnosis, treatment, and management of colon diseases, leading to better patient care.

A curated disease classification project can provide valuable information about colon diseases to the public. This can empower individuals to understand the signs, symptoms, and risk factors associated with different colon diseases, leading to increased awareness and proactive healthcare seeking behaviours.

Business Impact:
Companies in the medical technology and healthcare services sector can benefit from the curated colon disease classification project. They can leverage the project's outcomes to develop innovative medical devices, diagnostic tools, and software applications that aid in accurate disease classification and improve patient care.

Healthcare institutions can utilize the curated disease classification project to enhance their diagnostic capabilities and streamline their treatment processes.
It can lead to more efficient workflows, reduced diagnostic errors, and improved patient outcomes.

Milestone 2: Data Collection & Image Preprocessing:
In this milestone First, we will collect images of Colon Diseases then organized into subdirectories based on their respective names as shown in the project structure.

Create folders of types of Colon Diseases that need to be recognized. In this project, we have collected images of 4 types of Images like:

- Normal,
- Ulcerative-colitis,
- Polyps &
- Esophagitis

they are saved in the respective sub directories with their respective names.

In this project, we have collected images of 4 types of Images like Normal, Ulcerative-colitis, Polyps & Esophagitis they are saved in the respective sub directories with their respective names.

In Image Processing, 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 Kera's 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.

```
#import image datagenerator library
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

Activity 2: **Configure ImageDataGenerator class**

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

```
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
```

Activity 3: Apply ImageDataGenerator functionality to Trainset and Testset

Let us apply ImageDataGenerator functionality to Train set 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 subdirectories normal, ulcerative colitis, polyps, esophagitis, together with labels 0 to 3{normal: 0, ulcerative colitis: 1, polyps: 2, esophagitis: 3 }

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 15.
- 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).

```
[5] train_data = train_datagen.flow_from_directory(
    '/content/train/train',
    target_size=(224,224),
    batch_size=15,
    class_mode='categorical')

    Found 3200 images belonging to 4 classes.
```

```
[7] test_data = train_datagen.flow_from_directory(
    '/content/test/test',
    target_size=(224, 224),
    batch_size=15,
    class_mode='categorical')

    Found 800 images belonging to 4 classes.
```

We notice that 3200 images belong to 4 classes for training and 800 images belong to 4 classes for testing purposes.

Milestone 3: Model Building

Now it's time to build our Convolutional Neural Networking using vgg16 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

Importing Libraries

```
[23] import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.preprocessing.image  import ImageDataGenerator
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
     from keras.api._v2.keras import activations
```

Activity 2: Importing the VGG16 model

To initialize the VGG16 model, the weights are usually pre-trained on the ImageNet dataset, which is a large-scale dataset of images belonging to 1,000 different categories.

These pre-trained weights can be downloaded from the internet, and they can be used as a starting point to fine-tune the model for a specific task, such as object recognition or classification.

Importing VGG16 Architecture

```
[8]  from tensorflow.keras.applications.vgg16 import VGG16
     from tensorflow.keras.layers import Flatten
```

Activity 3: Initializing the model:

- The model will be initialized with the pre-trained weights from the ImageNet dataset, and the last fully connected layer will be excluded from the model architecture.
- The loop that follows freezes the weights of all the layers in the VGG16 model by setting `i.trainable=False` for each layer in the model.
- This is done to prevent the weights from being updated during training, as the model is already pre-trained on a large dataset.
- Finally, a `Flatten()` layer is added to the output of the VGG16 model to convert the output tensor into a 1D tensor.
- The resulting model can be used as a feature extractor for transfer learning or as a starting point for building a new model on top of it.

```
[ ] Image_size=[224,224]

[ ] sol=VGG16(input_shape=Image_size + [3] , weights='imagenet' , include_top = False)

    Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
    58889256/58889256 [==============================] - 0s 0us/step

[ ] for i in sol.layers:
      i.trainable = False

[ ] y=Flatten()(sol.output)
```

Activity 4: Adding Fully connected Layers

As the input image contains three channels, we are specifying the input shape as (128,128,3). We are adding a output layer with activation function as "SoftMax" .

```
[ ]  from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
```

```
[ ]  from keras.api._v2.keras import activations
     final = Dense(4, activation = 'softmax')(y)
```

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use SoftMax activation to convert their outputs into respective probabilities.

Understanding the model is a very important phase to properly use it for training and prediction purposes. Kera's provides a simple method, summary to get the full information about the model and its layers.

```
vgg16_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_pool (MaxPooling2D)  (None, 28, 28, 256)       0

 block4_conv1 (Conv2D)       (None, 28, 28, 512)       1180160

 block4_conv2 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_conv3 (Conv2D)       (None, 28, 28, 512)       2359808

 block4_pool (MaxPooling2D)  (None, 14, 14, 512)       0

 block5_conv1 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv2 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_conv3 (Conv2D)       (None, 14, 14, 512)       2359808

 block5_pool (MaxPooling2D)  (None, 7, 7, 512)         0

 flatten (Flatten)           (None, 25088)             0
```

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.

Kera's 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.

Compiling the Model

```
[ ]  vgg16_model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['Accuracy'])
```

Activity 6: Train The model

Now, let us train our model with our image dataset. The model is trained for 9 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.

Training the Model

```
vgg16_model.fit(train_data, epochs = 9,validation_data=test_data)
```

```
Epoch 1/9
214/214 [==============================] - 112s 467ms/step - loss: 0.2310 - Accuracy: 0.9128 - val_loss: 0.4381 - val_Accuracy: 0.8300
Epoch 2/9
214/214 [==============================] - 100s 468ms/step - loss: 0.1121 - Accuracy: 0.9581 - val_loss: 0.3417 - val_Accuracy: 0.8813
Epoch 3/9
214/214 [==============================] - 99s 462ms/step - loss: 0.0893 - Accuracy: 0.9684 - val_loss: 0.7283 - val_Accuracy: 0.7875
Epoch 4/9
214/214 [==============================] - 100s 465ms/step - loss: 0.0551 - Accuracy: 0.9775 - val_loss: 0.5964 - val_Accuracy: 0.8213
Epoch 5/9
214/214 [==============================] - 99s 465ms/step - loss: 0.0617 - Accuracy: 0.9781 - val_loss: 0.8503 - val_Accuracy: 0.8012
Epoch 6/9
214/214 [==============================] - 98s 459ms/step - loss: 0.0700 - Accuracy: 0.9737 - val_loss: 0.4146 - val_Accuracy: 0.8687
Epoch 7/9
214/214 [==============================] - 98s 458ms/step - loss: 0.0553 - Accuracy: 0.9806 - val_loss: 0.4795 - val_Accuracy: 0.8750
Epoch 8/9
214/214 [==============================] - 99s 464ms/step - loss: 0.0370 - Accuracy: 0.9875 - val_loss: 1.1152 - val_Accuracy: 0.8188
Epoch 9/9
214/214 [==============================] - 98s 457ms/step - loss: 0.0377 - Accuracy: 0.9866 - val_loss: 0.6684 - val_Accuracy: 0.8512
<keras.callbacks.History at 0x7f59100f5450>
```

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.

```
vgg16_model.save('cnn .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.

Testing the Data

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
```

```
labels=['0_normal','1_ulcerative_colitis','2_polyps','3_esophagitis']
```

```
img_path ='/content/test/test/1_ulcerative_colitis (164).jpg'
```

```
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16_model.predict(np.array([x]))
preds
```

```
1/1 [==============================] - 1s 732ms/step
array([[0.000000e+00, 1.000000e+00, 0.000000e+00, 6.255547e-24]],
      dtype=float32)
```

```
labels[np.argmax(preds)]
```

```
'1_ulcerative_colitis'
```

Taking an image as input and checking the results. By using the model we are predicting the output for the given input image. The predicted class index name will be printed here.

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 Colon Diseases and showcased on the HTML page to notify the user. Whenever the user interacts with the UI and selects the “Inspect” button, the next page is opened where the user chooses the image and predicts the output.

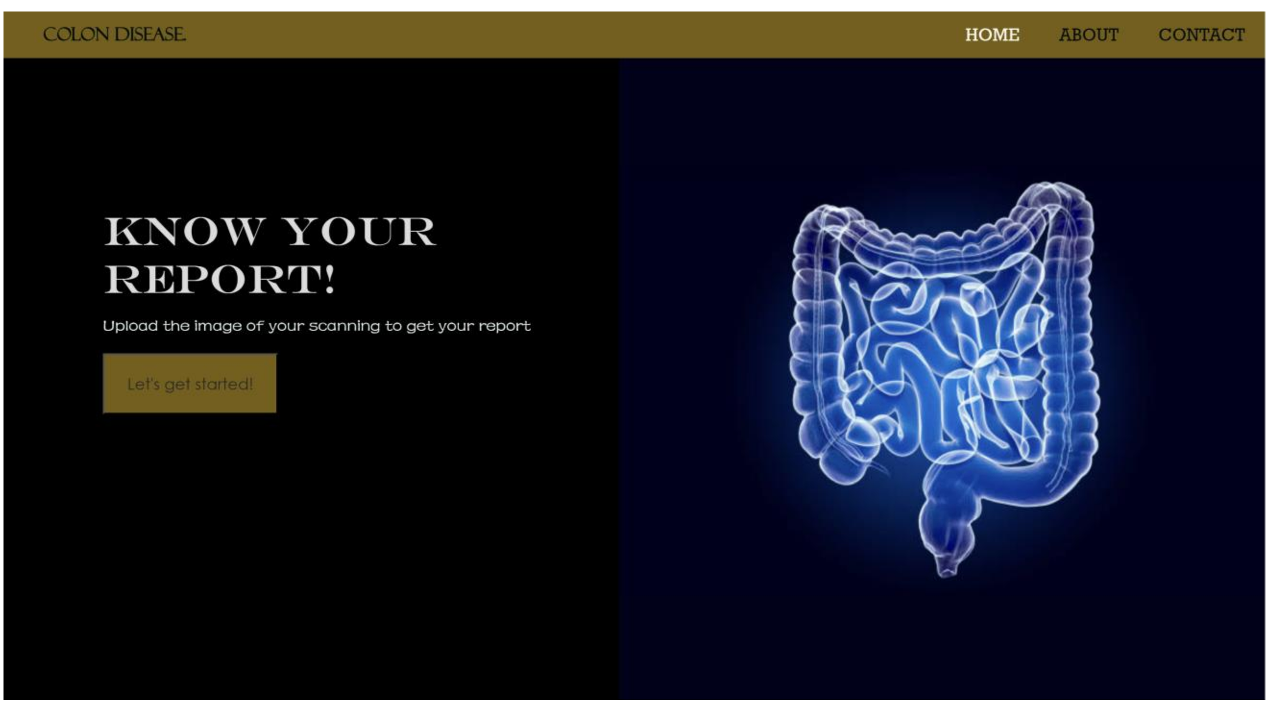
Activity 1: Create HTML Pages

We use HTML to create the front-end part of the web page.

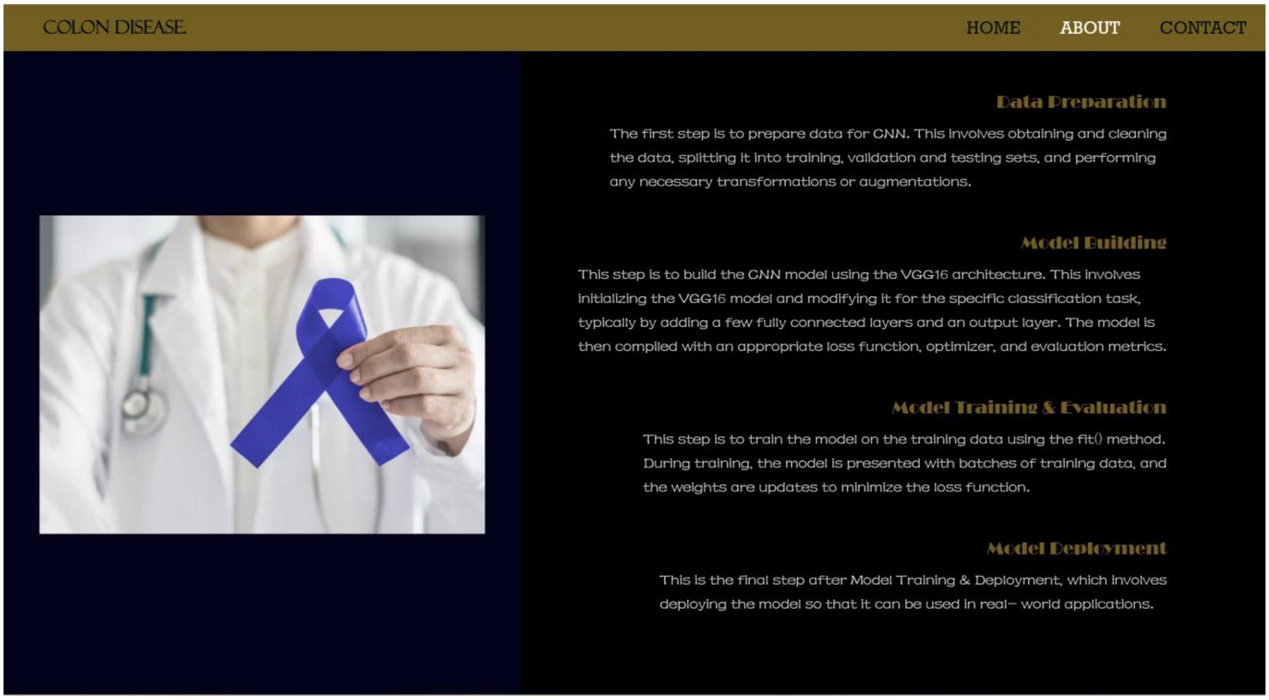
Here, we have created 3 HTML pages- index.html, predict.html, and output.html

- home.html displays the home page.
- index.html displays an introduction about the project
- upload.html gives the emergency alert

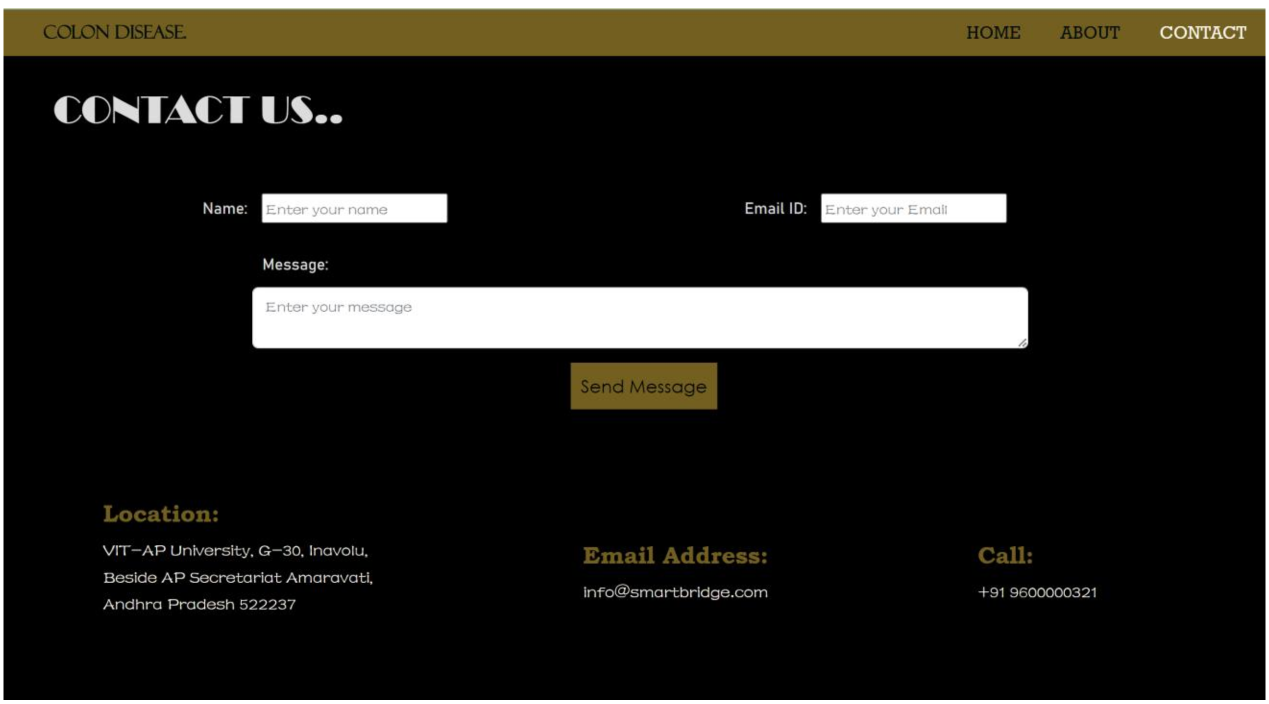
Index page looks like this:



About Section:



Contact Us:



Activity 2: Build python code

Task 1: Importing Libraries

The first step is usually importing the libraries that will be needed in the program. Importing the flask module in the project is mandatory. An object of the Flask class is our WSGI application. Flask constructor takes the name of the current module (name) as argument Pickle library to load the model file.

```
import numpy as np
import os
import tensorflow as tf
from PIL import Image
from flask import Flask, render_template,request,jsonify,url_for,redirect
from tensorflow.keras.preprocessing.image import load_img,img_to_array
```

Task 2: Creating our flask application and loading our model by using load_model method

```
app=Flask(__name__)
model = tf.keras.models.load_model('cnn.h5')
```

Task 3: Routing to the html Page

Here, the declared constructor is used to route to the HTML page created earlier.
In the above example, '/' URL is bound with index.html function. Hence, when the home page of a web server is opened in the browser, the html page will be rendered. Whenever you browse an image from the html page this photo can be accessed through POST or GET Method.

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

@app.route('/predict')
def predict():
    return render_template("predict.html")
```

Showcasing prediction on UI:

```python
@app.route('/output',methods=['GET','POST'])
def output():
    if request.method=='POST':
        f=request.files['file']
        basepath=os.path.dirname(__file__)
        filepath=os.path.join(basepath,'uploads',f.filename)
        f.save(filepath)
        img=load_img(filepath,target_size=(224,224))
        # Resize the image to the required size

        # Convert the image to an array and normalize it
        image_array = np.array(img)
        # Add a batch dimension
        image_array = np.expand_dims(image_array, axis=0)
        # Use the pre-trained model to make a prediction
        pred=np.argmax(model.predict(image_array),axis=1)
        index=['Normal','Uclerative_colitis','Polyps','Esophagitis']
        prediction = index[int(pred)]
        print("prediction")
        return render_template("output.html",predict = prediction)
```

Here we are defining a function which requests the browsed file from the html page using the post method. The requested picture file is then saved to the uploads folder in this same directory using OS library.

Using the load image class from Kera's library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that pre-processed image to the model for predicting the class.

This returns the numerical value of a class (like 0, 1 ,2, 3.) which lies in the 0th index of the variable preds. This numerical value is passed to the index variable declared. This returns the name of the class. This name is rendered to the prediction variable used in the html page.

Finally, Run the application

This is used to run the application in a local host.

```python
if __name__ == '__main__':
    app.run(debug=False,threaded = False)
```

Activity 3: Run the application

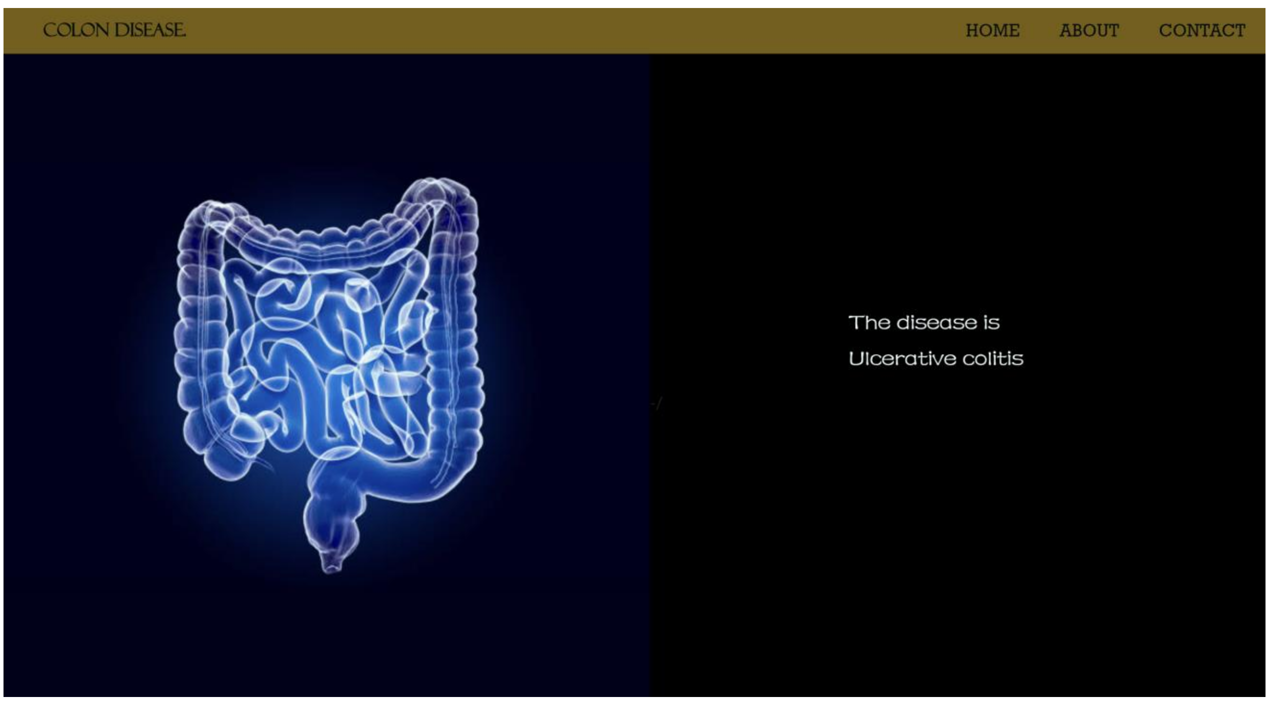
- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type "python app.py" command.
- It will show the local host where your app is running on **http://127.0.0.1.5000/**
- Copy that local host URL and open that URL in the browser. It does navigate me to where you can view your web page.
- Enter the values, click on the predict button and see the result/prediction on the web page.

Then it will run on localhost: 5000

```
PS F:\Otherproj\Curated_Colon_Disease\Curated_Colon_Disease> python -u "f:\Otherproj\Curated_Colon_Disease\Curated_Colon_Disease\app.py"
 * Serving Flask app 'app' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 580-415-876
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

FINAL OUTPUT:

Output 1: (Ulcerative colitis)



Output 2: (Polyps)