SynapseScan: AI Driven Classification of Ovarian Cancer Variants

SynapseScan: AI Driven Classification of Ovarian Cancer Variants:

SynapseScan represents a groundbreaking initiative focused on the AI-driven classification of ovarian cancer variants, utilizing the power of transfer learning. This project aims to enhance the accuracy and efficiency of ovarian cancer diagnosis by leveraging pre-trained models and transfer learning techniques.

Scenario 1: Early Cancer Detection

SynapseScan contributes to early detection by analyzing ovarian cancer variants with AI-driven classification. The system can process medical imaging data, such as histopathological images or radiological scans, and utilize transfer learning to identify subtle variations indicative of different ovarian cancer types. Early detection allows for prompt intervention and improved patient outcomes.

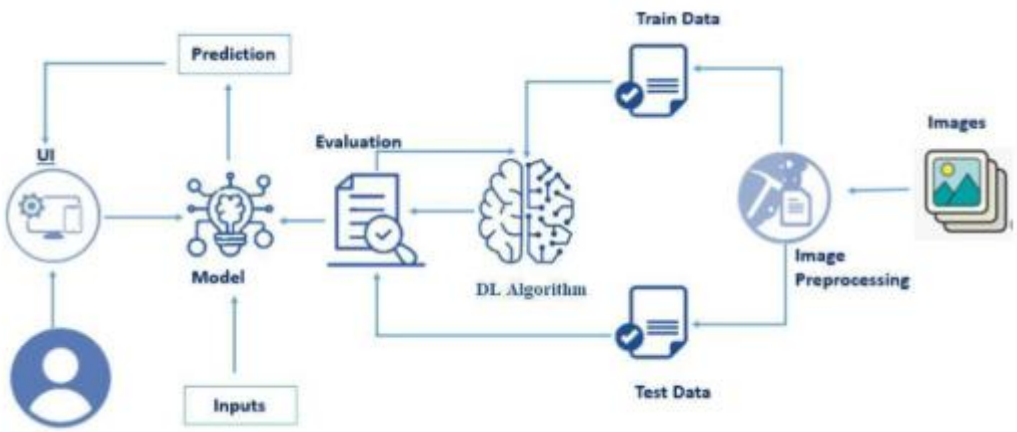
Scenario 2: Personalized Treatment Plans

The project facilitates personalized treatment plans by leveraging AI to classify ovarian cancer variants accurately. With transfer learning, the system can adapt to the specific characteristics of individual patient cases. This personalized approach enables oncologists to tailor treatment strategies based on the unique genetic and molecular features of ovarian cancer, improving the efficacy of therapeutic interventions.

Scenario 3: Research Acceleration

SynapseScan accelerates ovarian cancer research by automating the classification process. Researchers can utilize the AI-driven system to analyze large datasets, identify patterns, and classify cancer variants efficiently. This accelerates the pace of research, leading to a better understanding of ovarian cancer biology, potential biomarkers, and novel treatment avenues.

Technical Architecture:



Pre requisites

To complete this project, you must require the following software, concepts, and packages

- Anaconda navigator and PyCharm / Spyder:
- Refer to the link below to download Anaconda Navigator
- Link (PyCharm) : https://youtu.be/1ra4zH2G4o0
- Link (Spyder): https://youtu.be/5mDYijMfSzs
- Python packages:
- Open anaconda prompt as administrator
- Type “pip install numpy” and click enter.
- Type “pip install pandas” 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.

Project Objectives

Prior Knowledge:

You must have prior knowledge of the following topics to complete this project.

- 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

Project Objectives:

By the end of this project you'll understand:

- Preprocessing the images.
- Applying Transfer learning algorithms on the dataset.
- How deep neural networks detect the disease.
- You will be able to know how to find the accuracy of the model.

You will be able to Build web applications using the Flask framework.

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.
- The Model analyzes the image, then the prediction is showcased on the Flask UI.

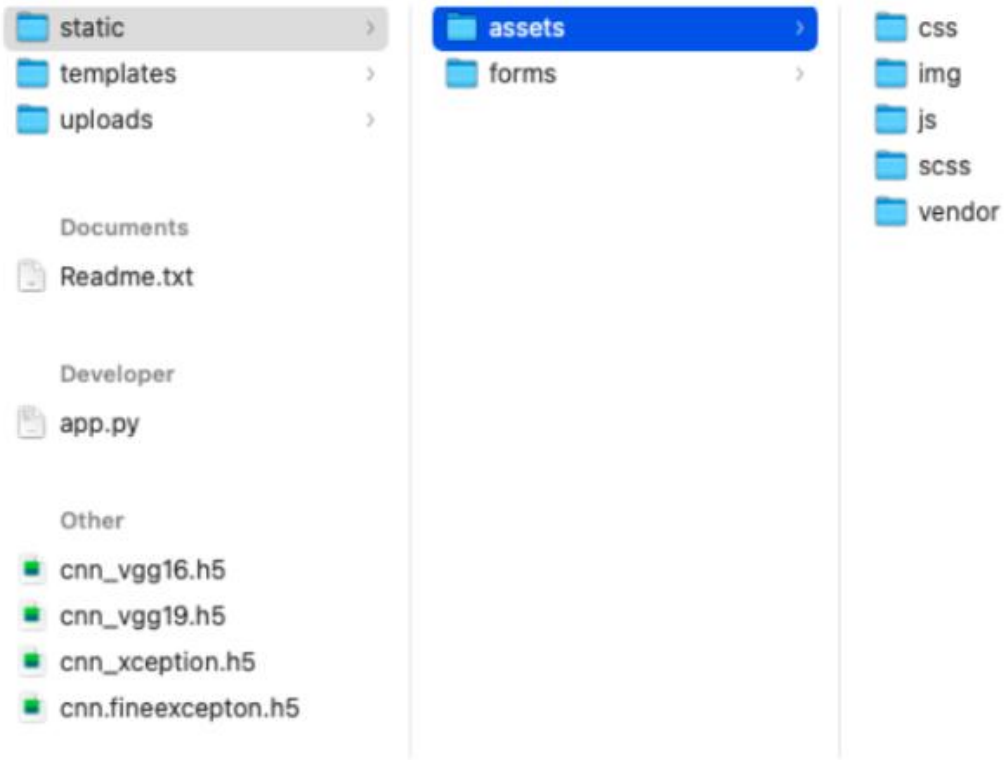
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
- Create a Train and Test path.
- Data Pre-processing.
- Import the required library
- Configure ImageDataGenerator class
- ApplyImageDataGenerator functionality to Trainset and Testset
- Model Building
- Pre-trained CNN model as a Feature Extractor
- Adding Dense Layer
- Configure the Learning Process
- Train the model
- Save the Model
- Test the model
- Application Building
- Create an HTML file
- Build Python Code

Project Structure

Create a Project folder that 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 that is saved and the saved model in this content is a cnnfineeception.h5
- templates folder contains index.html, predict.html & output.html pages.



Data Collection

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

Download the dataset

It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, we have used overiancancertma.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Note: There are several techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques

We are going to build our training model on Google Colab.

Upload the dataset into Google Drive and connect the Google Colab with Drive using the below code.

```
[ ] !pip install -q kaggle
```

```
[ ] from google.colab import files
    files.upload()
```

```
Choose Files  No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving kaggle.json to kaggle.json
{'kaggle.json': b'{"username":"mvsravya","key":"eae6b18c1cc78f156faa4d7799572411"}'}
```

```
!mkdir ~/.kaggle
```

```
!cp kaggle.json ~/.kaggle
```

```
!kaggle datasets download -d toddgardiner/ovariancancertma-stanforddatabase
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /root/.kaggle/kaç
Downloading ovariancancertma-stanforddatabase.zip to /content
 88% 49.0M/55.8M [00:00<00:00, 81.1MB/s]
100% 55.8M/55.8M [00:00<00:00, 86.0MB/s]
```

OvarianCancerTMA-StanfordDatabase | Kaggle..

Ovarian Cancer TMA Samples from Stanford TMA Database..

https://www.kaggle.com/datasets/toddgardiner/ovariancancertma-stanforddatabase/data

Create training and testing dataset

To build a DL model we have six classes in our dataset. But In the project dataset folder training and testing data are needed. So, in this case, we just have to assign a variable and pass the folder path to it.

Three different transfer learning models are used in our project and the best model is selected.

The image input size of the model is 299, 299.

```
!unzip /content/ovariancancertma-stanforddatabase.zip
```

Importing the libraries:

```
import numpy as np
import tensorflow as tf
import random
from cv2 import resize
from tensorflow import keras
from tensorflow.keras.preprocessing.image  import ImageDataGenerator
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import sigmoid
from keras.api._v2.keras import activations
```

```
import os
import shutil
from sklearn.model_selection import train_test_split
```

```
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras import layers, models
```

```
from tensorflow.keras.applications import VGG16
```

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.

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,zoom_range= 0.2,shear_range= 0.2)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

Apply ImageDataGenerator functionality to Train set and Test set

Let us apply ImageDataGenerator functionality to the 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

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: The size of the batches of data which is 32.
- 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).

```
trainpath = '/content/train'
testpath = '/content/test'
```

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(rescale = 1./255,zoom_range= 0.2,shear_range= 0.2)
test_datagen = ImageDataGenerator(rescale = 1./255)
```

```
train = train_datagen.flow_from_directory(trainpath,target_size =(224,224),batch_size = 20)
test = test_datagen.flow_from_directory(testpath,target_size =(224,224),batch_size = 20) ,#5 ,15 , 32, 50
```

```
Found 348 images belonging to 3 classes.
Found 152 images belonging to 3 classes.
```

Pre-trained CNN model as a Feature Extractor

For one of the models, we will use it as a simple feature extractor by freezing all five convolution blocks to make sure their weights don't get updated after each epoch as we train our model.

Here, we have considered images of dimension (299,299,3).

Also, we have assigned include_top = False because we are using the convolution layer for feature extraction and want to train a fully connected layer for our image classification(since it is not the part of Imagenet dataset)

Flatten layer flattens the input. Does not affect the batch size.

```
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
```

```
vgg = VGG16(include_top = False,input_shape = (224,224,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_t
58889256/58889256 [==============================] - 0s 0us/step
```

```
for layer in vgg.layers:
  print(layer)
```

```
len(vgg.layers)
```

```
19
```

Adding Dense Layers

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer.

```
for layer in vgg.layers:
  layer.trainable = False
```

```
x= Flatten()(vgg.output)
```

```
output = Dense(3, activation ='softmax')(x)
```

```
vgg16 = Model(vgg.input,output)
```

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. Keras provides a simple method, and summary to get the full information about the model and its layers.

```
vgg16.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
```

```
=================================================================
Total params: 14789955 (56.42 MB)
Trainable params: 75267 (294.01 KB)
Non-trainable params: 14714688 (56.13 MB)
```

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

```
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
opt = Adam(lr=0.0001)

# Assuming you have defined your VGG16 model as vgg16

# Define Early Stopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)

# Compile the model (you may have already done this)
vgg16.compile(optimizer=opt , loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model with early stopping callback
history = vgg16.fit(train, validation_data=test, epochs=50, steps_per_epoch=len(train), validation_steps=len(test), callbacks=[early_stoppi
```

Train the model

Now, let us train our model with our image dataset. The model is trained for 25 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 10 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.

Early stopping is a regularization technique used in deep learning to prevent overfitting by monitoring the performance of a model on a validation set.

if the performance stops improving for a certain number of epochs, the training process is stopped early.

- 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.

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras
Epoch 1/50
18/18 [==============================] - 22s 745ms/step - loss: 0.8017 - accuracy: 0.7098 - val_loss: 0.2395 - val_accura
Epoch 2/50
18/18 [==============================] - 9s 516ms/step - loss: 0.3152 - accuracy: 0.8764 - val_loss: 0.2649 - val_accurac
Epoch 3/50
18/18 [==============================] - 9s 512ms/step - loss: 0.2446 - accuracy: 0.9109 - val_loss: 0.0780 - val_accurac
Epoch 4/50
18/18 [==============================] - 9s 480ms/step - loss: 0.1285 - accuracy: 0.9540 - val_loss: 0.1810 - val_accurac
```

From the above run time, we can easily observe that at 4 epochs the model is giving better accuracy.

ResNet 50

Now it's time to build our second model. Let's use the pre-trained model ResNet50 one of the convolution neural net (CNN) architectures which is considered as a very good model for Image classification.

Deep understanding of the model – The link is referred to in the prior knowledge section. Kindly refer to it before starting the model-building part.

```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.models import Model
```

```
resnet50 = ResNet50(include_top = False,input_shape = (224,224,3))
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_orderi
94765736/94765736 [==============================] - 0s 0us/step
```

```
for layer in resnet50.layers:
  print(layer)
```

```
resnet50.summary()
```

```
Model: "model_1"
```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	[(None, 224, 224, 3)]	0	[]
conv1_pad (ZeroPadding2D)	(None, 230, 230, 3)	0	['input_2[0][0]']
conv1_conv (Conv2D)	(None, 112, 112, 64)	9472	['conv1_pad[0][0]']
conv1_bn (BatchNormalization)	(None, 112, 112, 64)	256	['conv1_conv[0][0]']
conv1_relu (Activation)	(None, 112, 112, 64)	0	['conv1_bn[0][0]']
pool1_pad (ZeroPadding2D)	(None, 114, 114, 64)	0	['conv1_relu[0][0]']
pool1_pool (MaxPooling2D)	(None, 56, 56, 64)	0	['pool1_pad[0][0]']
conv2_block1_1_conv (Conv2	(None, 56, 56, 64)	4160	['pool1_pool[0][0]']

```
Total params: 23888771 (91.13 MB)
Trainable params: 301059 (1.15 MB)
Non-trainable params: 23587712 (89.98 MB)
```

```
len(resnet50.layers)
```

```
175
```

```
for layer in resnet50.layers:
  layer.trainable = False
```

```
x = Flatten()(resnet50.output)
```

```
output = Dense(3,activation = 'softmax')(x)
```

```
resnet50 =Model(resnet50.input,output)
```

CNN

Now it's time to build our second model. Let's use the pre-trained model CNN which is considered a very good model for Image classification.

Deep understanding of the model – The link is referred to in the prior knowledge section. Kindly refer to it before starting the model-building part.

```
from keras.callbacks import EarlyStopping
from keras.optimizers import Adam
opt = Adam(lr=0.0001)

# Define Early Stopping callback
early_stopping = EarlyStopping(monitor='val_accuracy', patience=3, restore_best_weights=True)

# Compile the model
resnet50.compile(loss = 'categorical_crossentropy',optimizer = opt,metrics =['accuracy'])

# Train the model with early stopping callback
history = resnet50.fit(train, validation_data=test, epochs=50, steps_per_epoch=len(train), validation_steps=len(test), callbacks=[early_sto
```

```
model.fit_generator(train_set,steps_per_epoch=len(train_set),
                    validation_data=test_set,
                    validation_steps=len(test_set),
                    epochs=30)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate` or use the legacy optimizer, e.g.,tf.keras
Epoch 1/50
18/18 [==============================] - 16s 573ms/step - loss: 2.7937 - accuracy: 0.4310 - val_loss: 0.9048 - val_accura
Epoch 2/50
18/18 [==============================] - 9s 471ms/step - loss: 0.9577 - accuracy: 0.5977 - val_loss: 0.9327 - val_accurac
Epoch 3/50
18/18 [==============================] - 11s 606ms/step - loss: 0.8956 - accuracy: 0.6063 - val_loss: 0.8247 - val_accura
Epoch 4/50
18/18 [==============================] - 9s 520ms/step - loss: 1.0533 - accuracy: 0.6178 - val_loss: 1.3754 - val_accurac
```

Testing the Model

Model testing is the process of evaluating the performance of a deep learning model on a dataset that it has not seen before. It is a crucial step in the development of any machine learning model, as it helps to determine how well the model can generalize to new data.

```
labels=['CC','EC','MC']
```

```
#img_path ='/content/test/EC/EC_original_506.jpg_5bc1131b-eef9-44a0-9289-90bdfb9914c0.jpg'
img_path = '/content/test/EC/EC_original_20110.jpg_17314f28-c885-4620-9f00-320b1e81f443.jpg'
```

```
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing.image import img_to_array, load_img
import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = vgg16.predict(np.array([x]))
preds
```

```
1/1 [==============================] - 1s 1s/step
array([[0., 1., 0.]], dtype=float32)
```

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

```
'EC'
```

```
#testing
img_path = '/content/test/CC/CC_original_454.jpg_1767fd10-ebb9-45e5-8aec-ddf1f2884972.jpg'
```

```
import numpy as np
img = load_img(img_path, target_size=(224, 224))
x = img_to_array(img)
x = preprocess_input(x)
preds = inceptionV3.predict(np.array([x]))
preds

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

```
1/1 [==============================] - 3s 3s/step
'CC'
```

In the above code, we have tested the model with an image of an ovarian cancer variant.

The model which is trained is working better with unknown data also, so we will save the model.

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.save('cnn_vgg16.h5')
print("model successfully")
```

```
model successfully
```

Application Building:

In this section, we will be building a web application that is integrated into the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

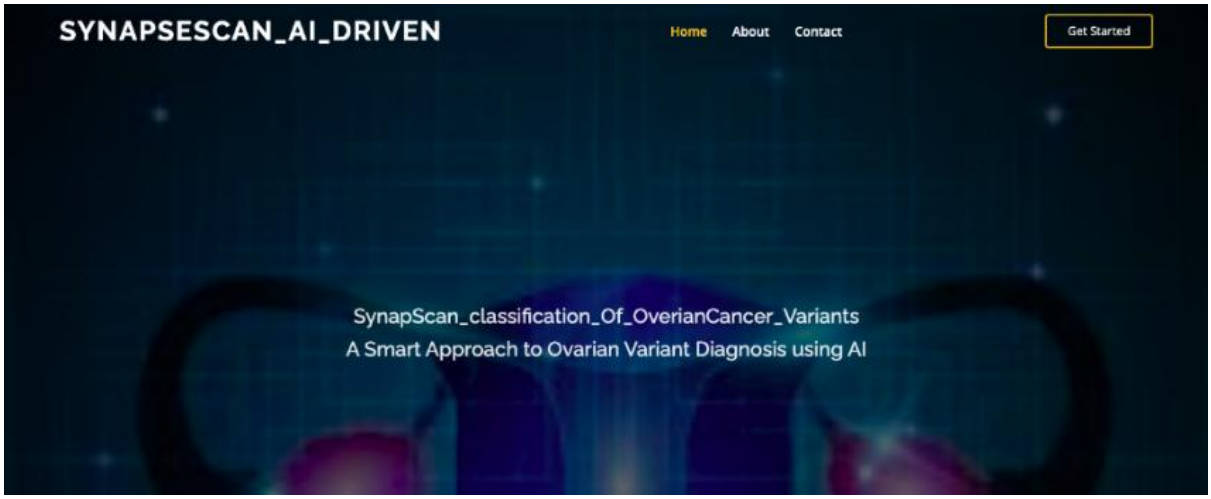
- Building HTML Pages
- Building server-side script

Building HTML pages:

For this project create one HTML file namely

- Index.html

Let's see what our index.html page looks like:



When you click on About button on the top, you will be redirected to the following page

Build Python code

- Import the libraries
- Loading the saved model and initializing the Flask app
- Render HTML pages:
- Once we upload the file into the app, then verifying the file uploaded properly or not. Here we will be using the declared constructor to route to the HTML page that we have created earlier.
- In the above example, '/' URL is bound with the home.html function. Hence, when the home page of the web server is opened in the browser, the HTML page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

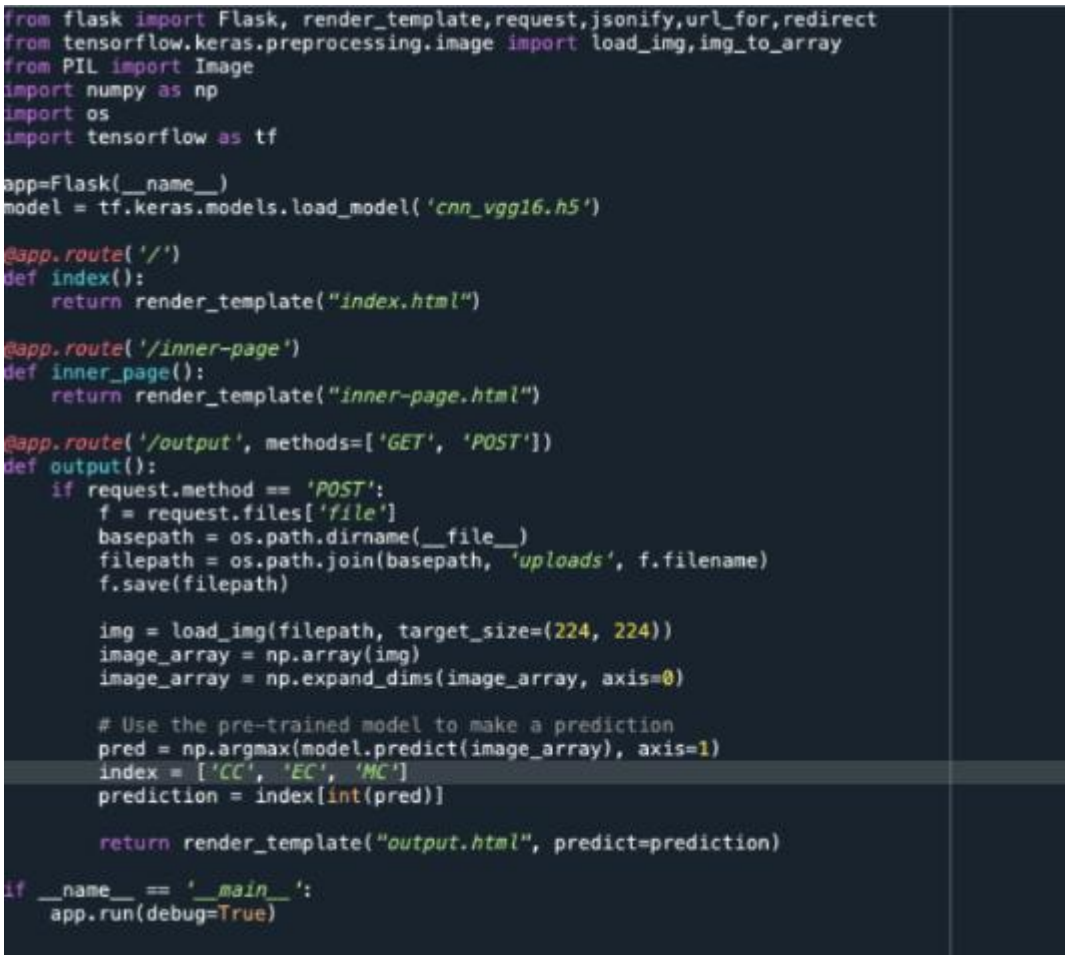
Here we are routing our app to res function. This function retrieves all the values from the HTML page using a Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. This prediction value will rendered to the text that we have mentioned in the index.html page earlier.

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

```
app=Flask(__name__)
model = tf.keras.models.load_model('cnn_vgg16.h5')
```

Run the Web Application

- Open the Anaconda prompt from the start menu.
- Navigate to the folder where your Python script is.
- Now type the " app.py" command.
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top right corner, enter the inputs, click on the submit button, and see the result/prediction on the web.
- This Python project application provides an endpoint classify image to classify images.
- Using pre-trained model convolution neural network (CNN) model.
- Upon receiving a POST request with an image file, the application preprocesses the image.
- An "uploads" folder is automatically created if it doesn't exist. This folder is used to store uploaded images.
- and the API returns the predicted class label along with the path to the uploaded image.



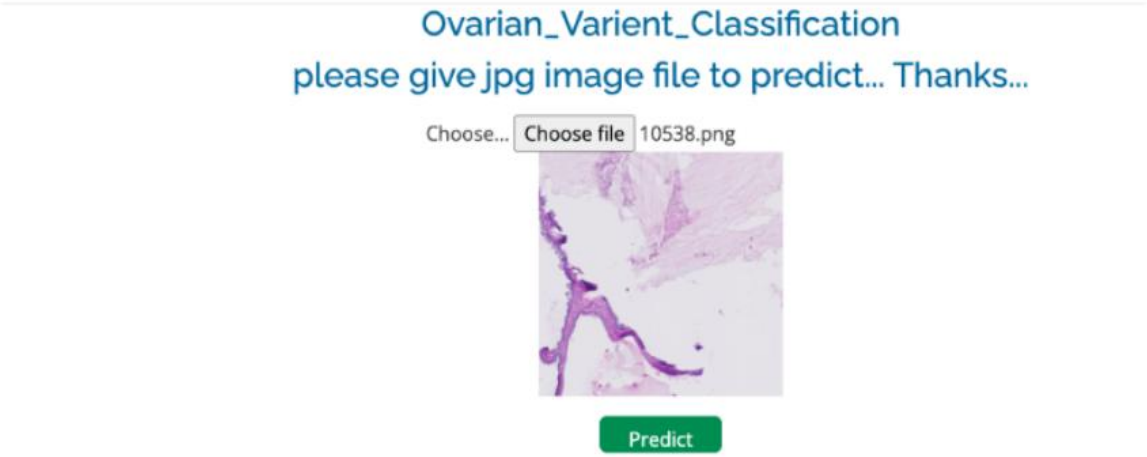
```python
from flask import Flask, render_template,request,jsonify,url_for,redirect
from tensorflow.keras.preprocessing.image import load_img,img_to_array
from PIL import Image
import numpy as np
import os
import tensorflow as tf

app=Flask(__name__)
model = tf.keras.models.load_model('cnn_vgg16.h5')

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

@app.route('/inner-page')
def inner_page():
    return render_template("inner-page.html")

@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))
        image_array = np.array(img)
        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 = ['CC', 'EC', 'MC']
        prediction = index[int(pred)]

        return render_template("output.html", predict=prediction)

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

```
024-01-23 11:23:32.930363: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary
s optimized to use available CPU instructions in performance-critical operations.
o enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the
ppropriate compiler flags.
* Serving Flask app 'app'
* Debug mode: on
ARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI
erver instead.
* Running on http://127.0.0.1:5000
ress CTRL+C to quit
* Restarting with watchdog (fsevents)
024-01-23 11:23:44.749712: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow binary
s optimized to use available CPU instructions in performance-critical operations.
o enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the
ppropriate compiler flags.
```

Open the local host mentioned in the output there we can see the webpage:

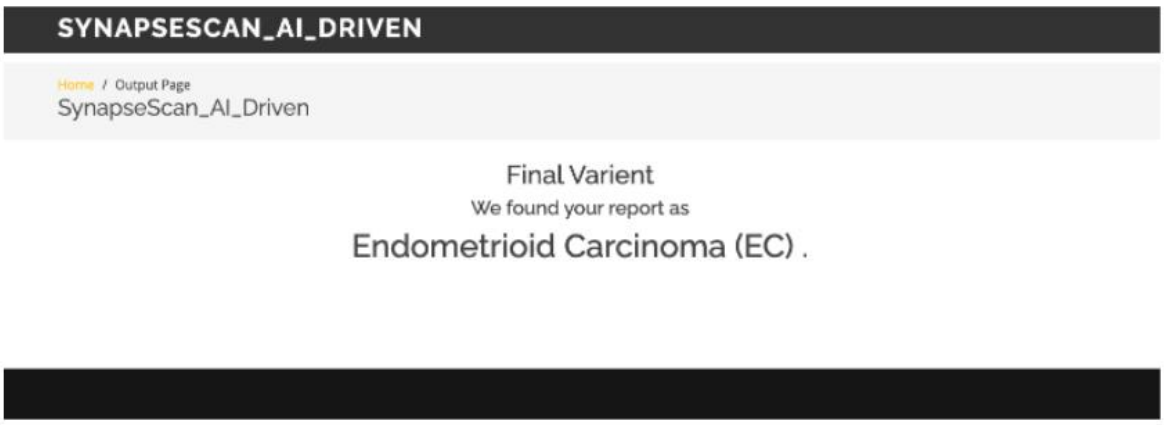
click on Predict button

Input 1



Once you upload the image and click on predict button, the output will be displayed on the below page

Output1:



Input:2

Output:2