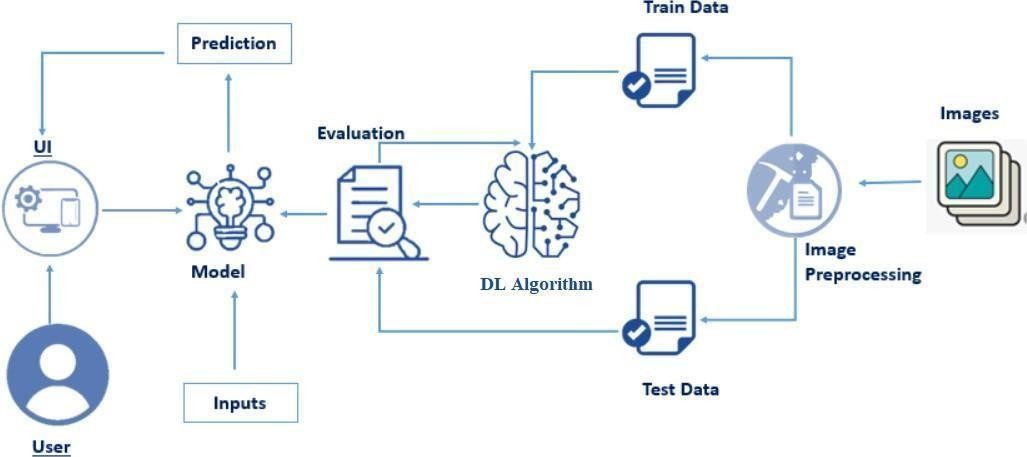
# Eye Disease Detection Using Deep Learning

### Project Description:

In this project we are classifying various types of Eye Diseases that people get due to various reasons like age, diabetes, etc. These diseases are majorly classified into 4 categories namely Normal, cataract, Diabetic Retinopathy & Glaucoma. Deep-learning (DL) methods in artificial intelligence (AI) play a dominant role as high-performance classifiers in the detection of the Eye Diseases using images.

Transfer learning has become one of the most common techniques that has achieved better performance in many areas, especially in image analysis and classification. We used Transfer Learning techniques like Inception V3, VGG19, Xception V3 that are more widely used as a transfer learning method in image analysis and they are highly effective.

**Technical Architecture:**



**Prerequisites:**

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

* **Anaconda navigator and PyCharm / Spyder:**
  + Refer 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” and click enter.
* Type “pip install keras” and click enter.
* Type “pip install Flask” and click enter.

## Prior Knowledge:

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

#### Deep Learning Concepts

* **CNN:** <https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add>

#### VGG19: [VGG-19 convolutional neural network - MATLAB vgg19 - MathWorks India](https://in.mathworks.com/help/deeplearning/ref/vgg19.html)

#### ResNet-50V2: <https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33>

* **Inception-V3:** <https://iq.opengenus.org/inception-v3-model-architecture/>

#### Xception: <https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/>

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

Link: [**https://www.youtube.com/watch?v=lj4I\_CvBnt0**](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 Eye 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 analyzed by the model which is integrated with flask application.
* The VGG19 Model analyzes the image, then the prediction is showcased on the Flask UI.

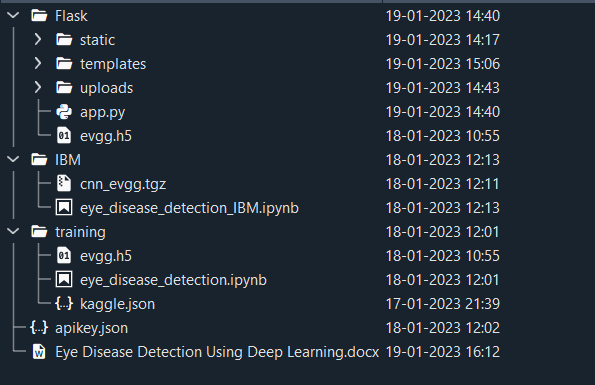
To accomplish this, we have to complete all the activities and tasks listed below

* Data Collection.
  + Create a Train and Test path.
* Data Pre-processing.
  + Import the required library
  + Configure ImageDataGenerator class
  + Apply ImageDataGenerator 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 which contains files as shown below



* Flask folder consists of static, templates and app.py.
* IBM folder consists of trained model notebook.
* Training file consist of eye\_disease\_detection.ipynb , model training.

## Milestone 1: Data Collection

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

**Activity 1: Download the dataset**

Collect images of Eye Diseases then organize into subdirectories based on their respective names as shown in the project structure. Create folders of types of Eye Diseases that need

to be recognized.

In this project, we have collected images of 4 types of Eye Diseases images like Normal, cataract, Diabetic Retinopathy & Glaucoma and they are saved in the respective sub directories with their respective names.

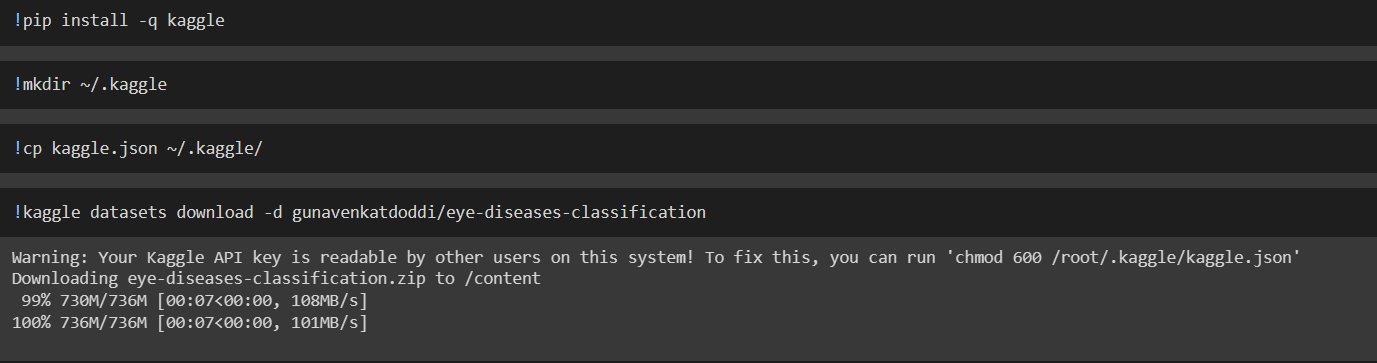
You can download the dataset used in this project using the below link

Dataset:- <https://www.kaggle.com/datasets/gunavenkatdoddi/eye-diseases-classification>

**Note: For better accuracy train on more images**

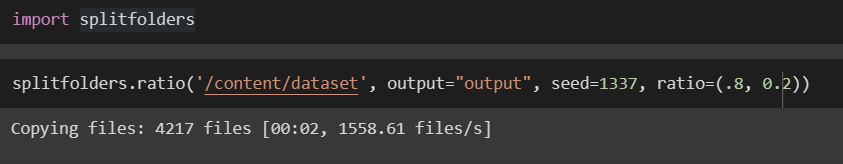
We are going to build our training model on Google colab.

We will be connecting Kaggle with Google Colab because the dataset is too big to import using the following code:



## Activity 2: Create training and testing dataset

To build a DL model we have to split training and testing data into two separate folders. But in this project dataset folder training and testing folders are not present. So, in this case we have to separate the data into train & test folders.



Four different transfer learning models are used in our project and the best model (VGG19) is selected. The image input size of VGG19 model is 224, 224.



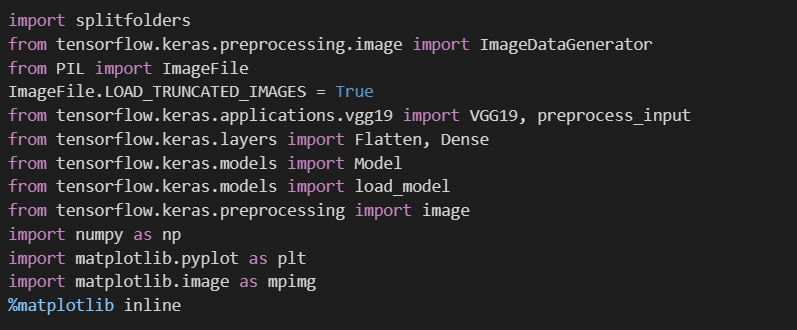
### Milestone 2: Image Preprocessing

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.

Link : <https://thesmartbridge.com/documents/spsaimldocs/CNNprep.pdf>

**Activity 1: Importing the libraries**

Import the necessary libraries as shown in the image

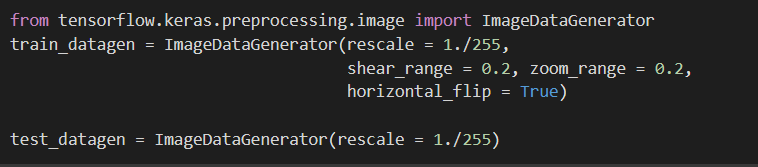
.

#### 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.

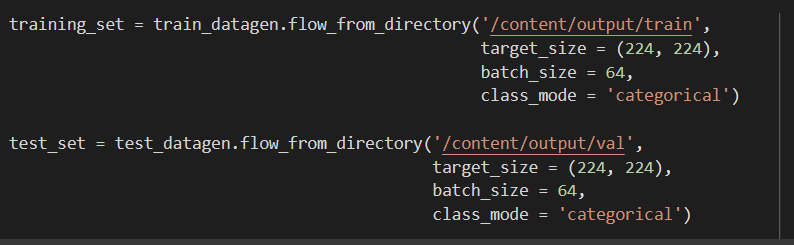


#### Activity 3: Apply ImageDataGenerator functionality to Train set and Test set

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: 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).



Total the dataset is having 3372 train images, 845 test images divided under 4 classes.

## Milestone 3: Model Building

Now it's time to build our model. Let’s use the pre-trained model which is VGG19, one of the convolution

neural net (CNN) architecture which is considered as a very good model for Image classification.

Deep understanding on the VGG19 model – Link is referred to in the prior knowledge section. Kindly refer to it before starting the model building part.

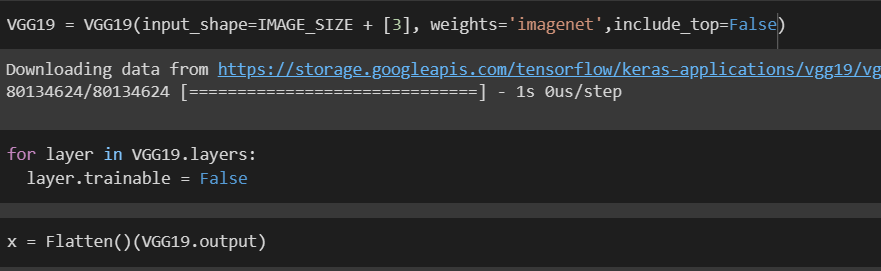
#### Activity 1: Pre-trained CNN model as a Feature Extractor

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

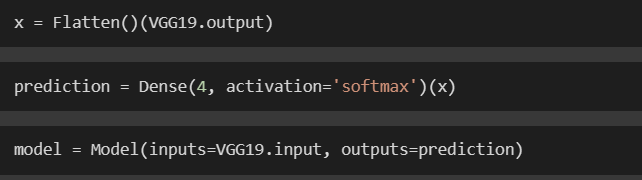
Here, we have considered images of dimension (224,244,3).

Also, we have assigned include\_top = False because we are using convolution layer for features extraction and wants to train 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.



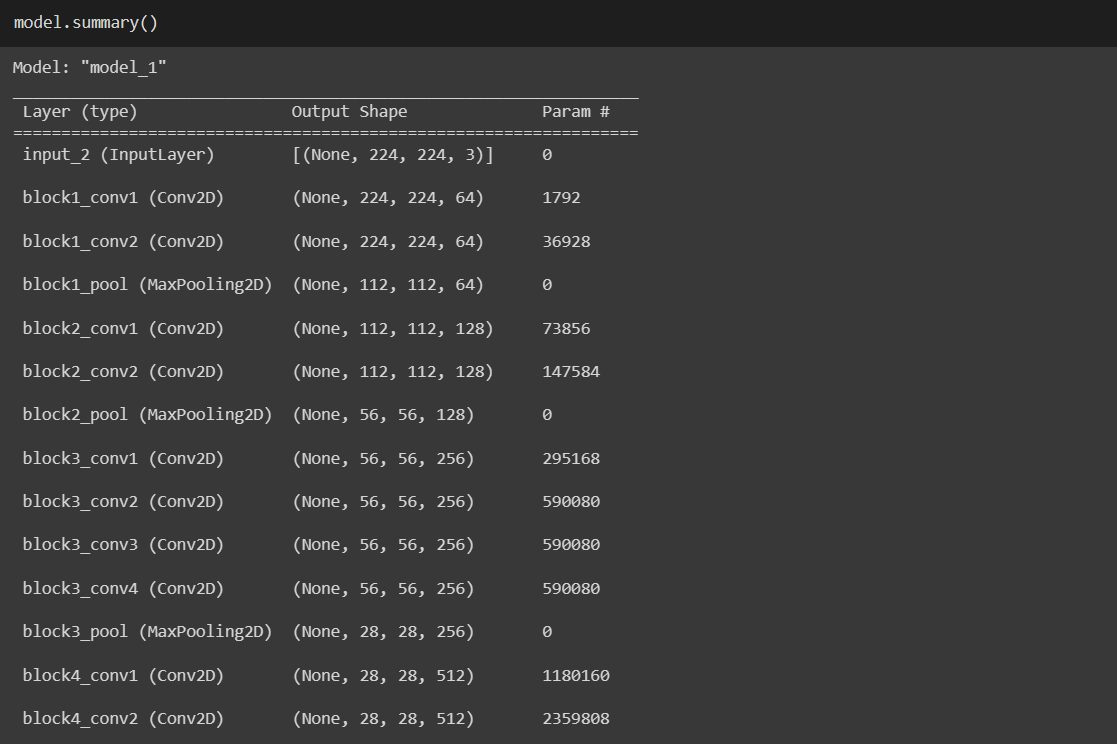
#### Activity 2: Adding Dense Layers

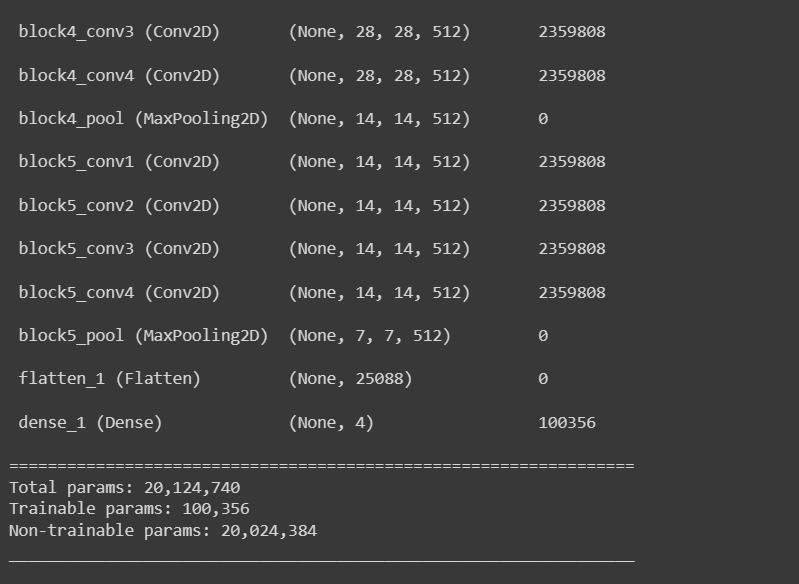


A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. Let us create a model object named model with inputs as VGG19.input and output as dense layer.

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.

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





#### Activity 3: 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



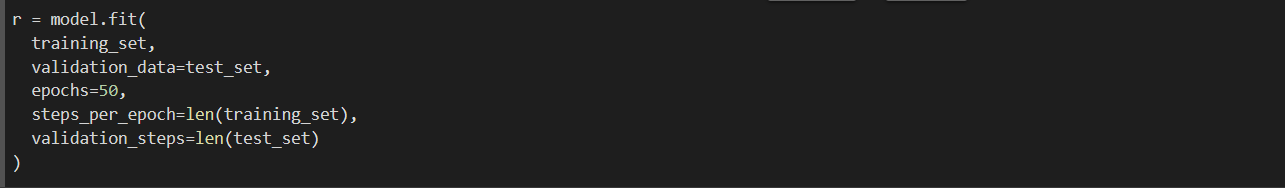
#### Activity 4: Train the model

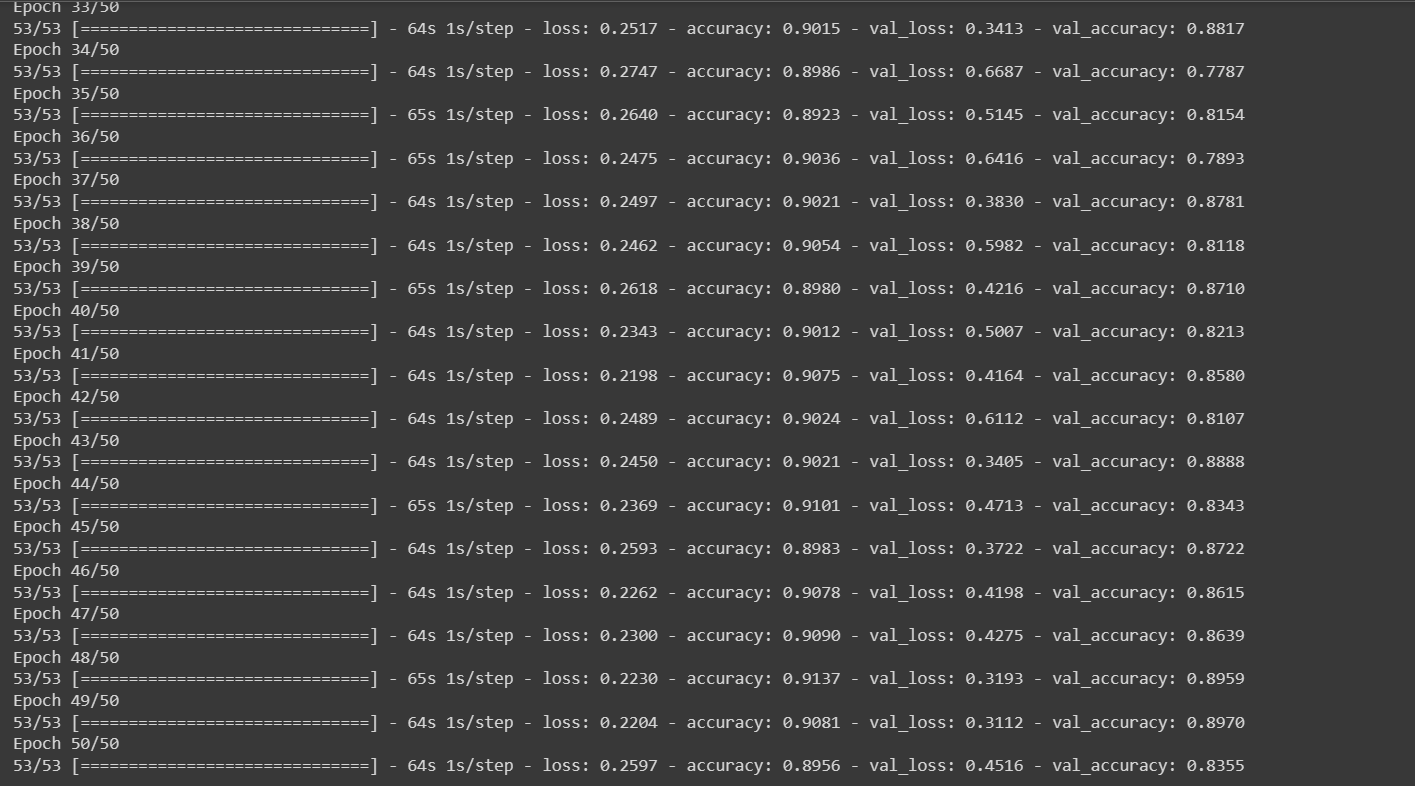
Now, let us train our model with our image dataset. The model is trained for 50 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch and probably there is further scope 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.





From the above run time, we can observe that at 50th epoch the model is giving the best accuracy.

**Milestone 4: 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.

### Milestone 5: Application Building

In this section, we will be building a web application that is integrated to 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

#### Activity1: Building Html Pages:

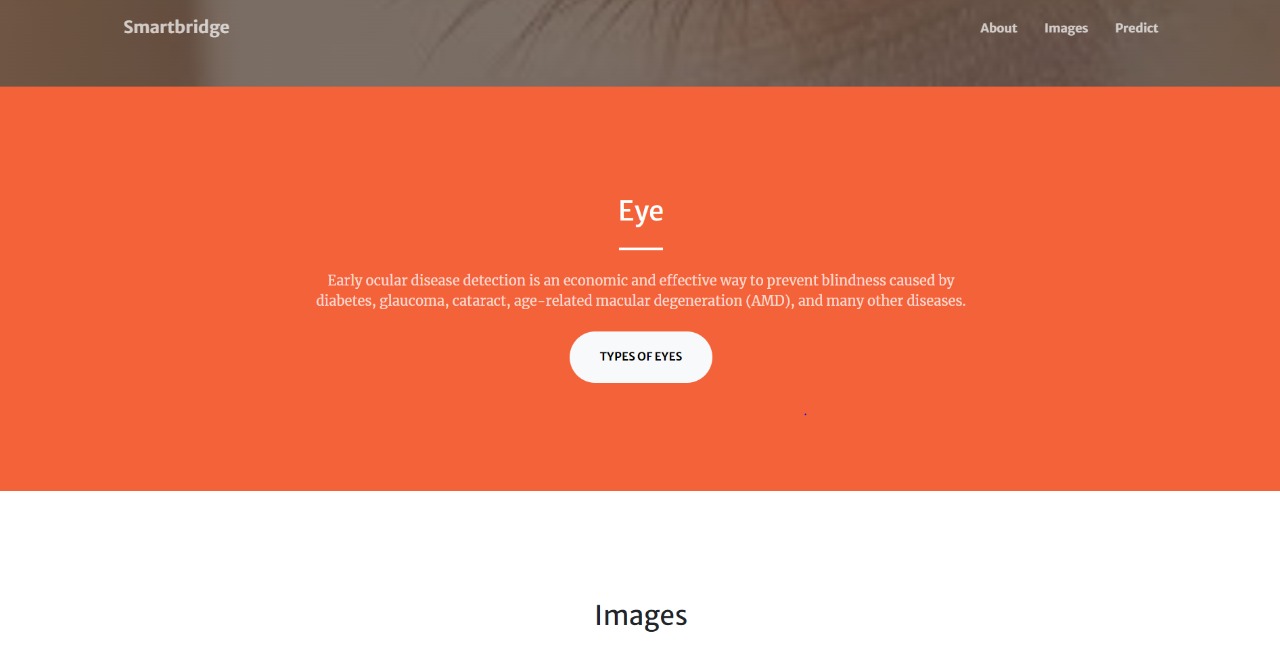
For this project create one HTML file namely

* index.html

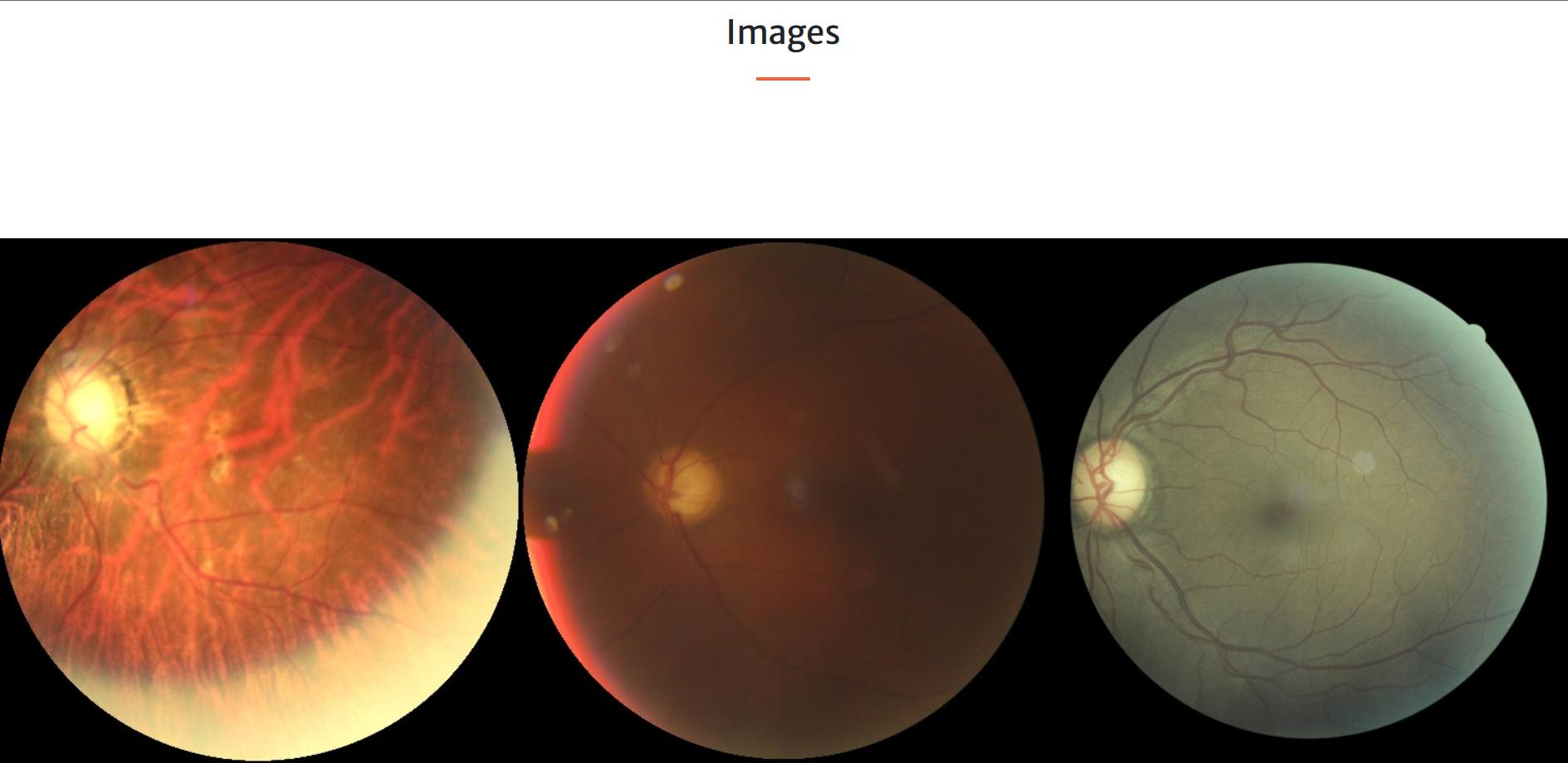
Let’s see how our index.html page looks like:



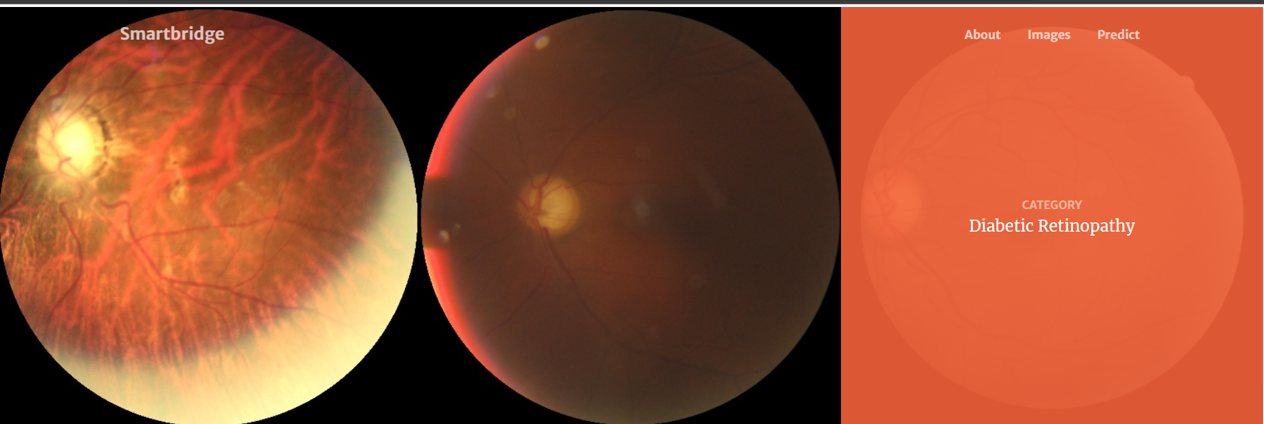
When you click on the FEW DETAILS or About button, you will be redirecting to the following page



When you click on the TYPES OF EYES or Images button, it will redirect you to the below page

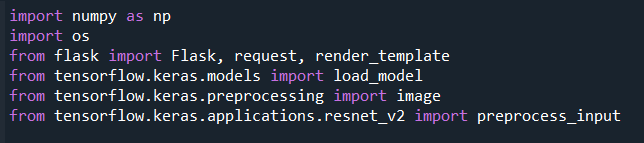


When you scroll down or hover over the images you will be able to see names of various Eye Diseases

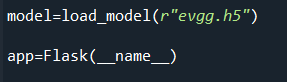


#### Activity 2: Build Python code:

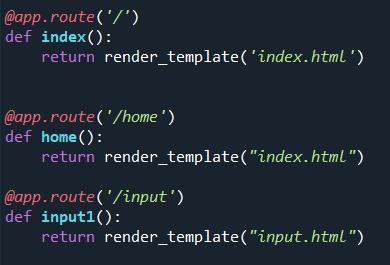
Import the libraries



Loading the saved model and initializing the flask app

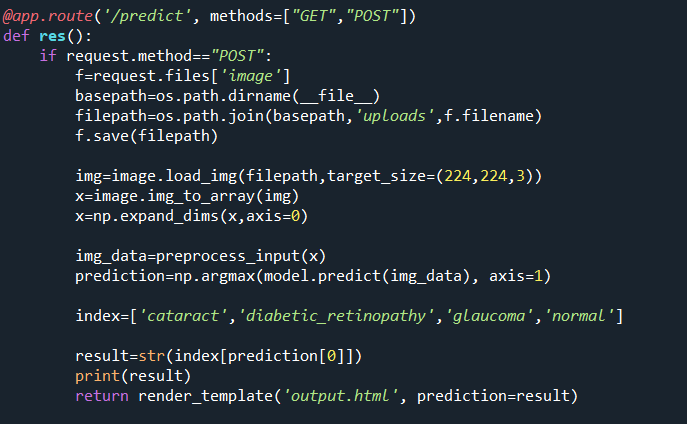


Render HTML pages:



Once we uploaded the file into the app, then verifying the file uploaded properly or not. Here we will be using declared constructor to route to the HTML page which we have created earlier.

In the above example, ‘/’ URL is bound with index.html function. Hence, when the home page of the web server is opened in browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.



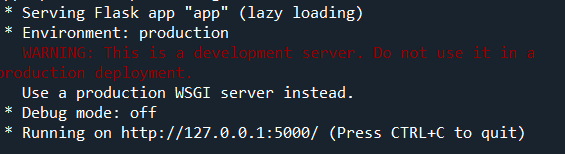
Here we are routing our app to predict function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will rendered to the text that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

#### Main Function:



**Activity 3: Run the application**

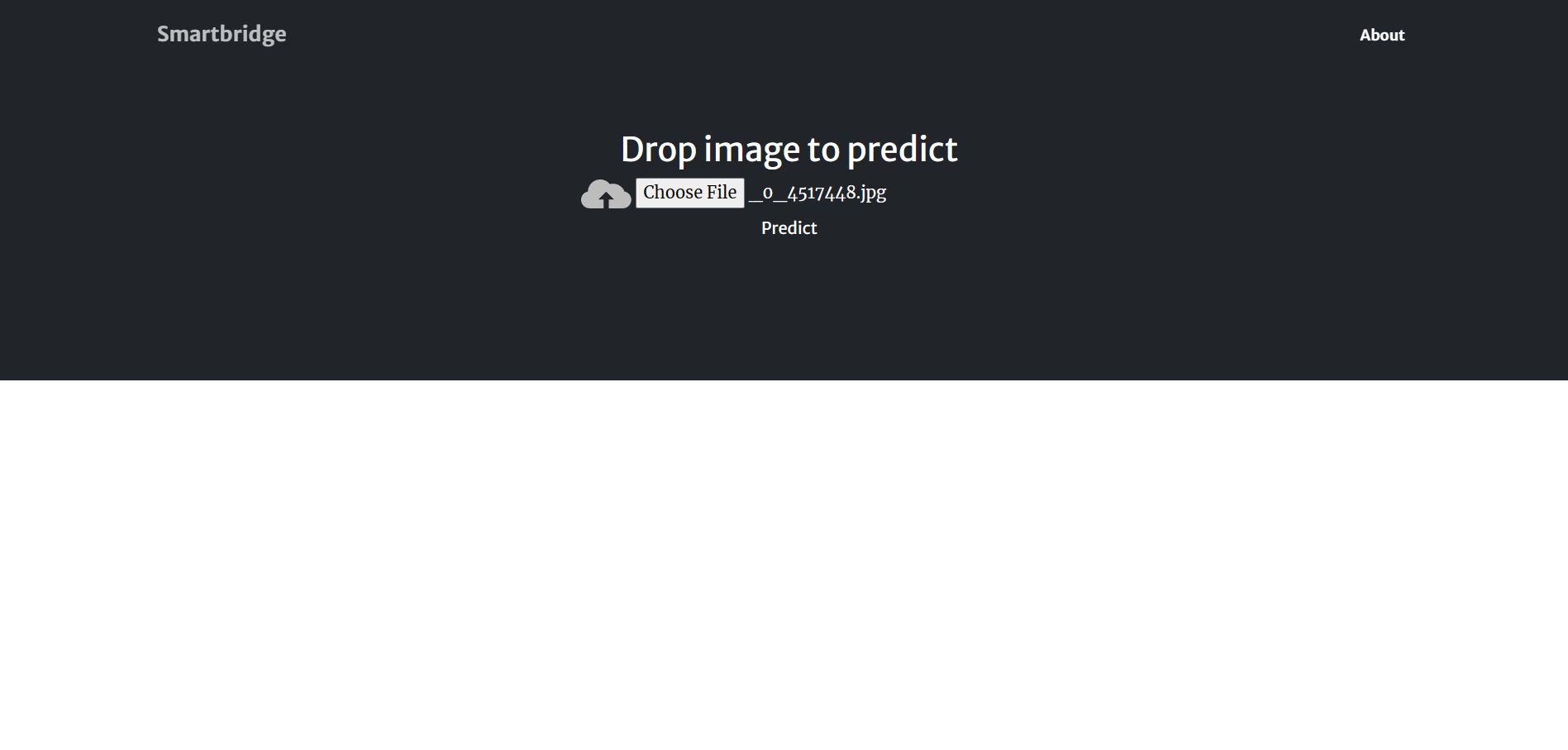
* Open Spyder
* Navigate to the folder where your Python script is.
* Now click on the green play button above.
* Click on the predict button from the top right corner, enter the inputs, click on the Classify button, and see the result/prediction on the web.



The home page looks like this. When you click on the Predict button, you’ll be redirected to the predict section

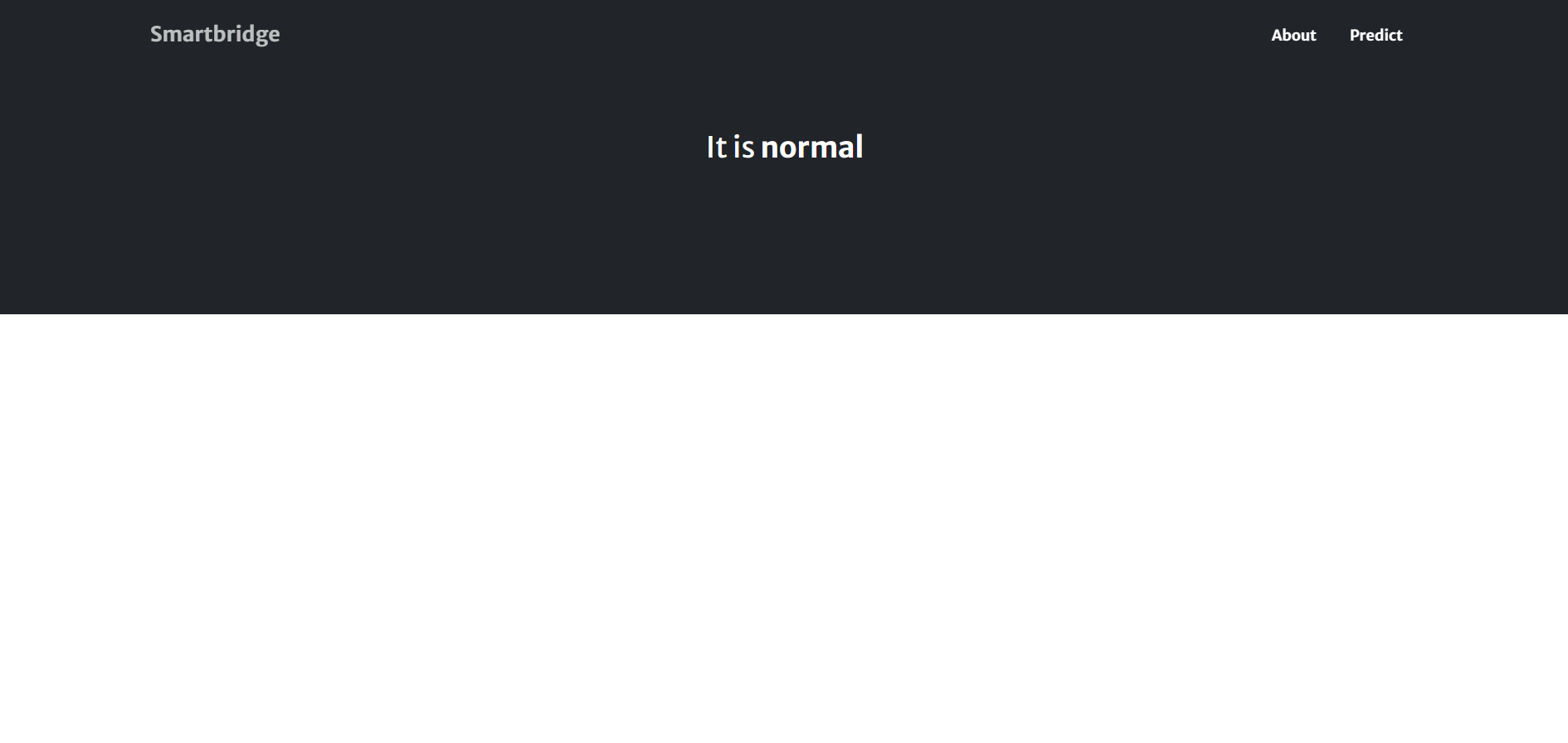


Input 1:

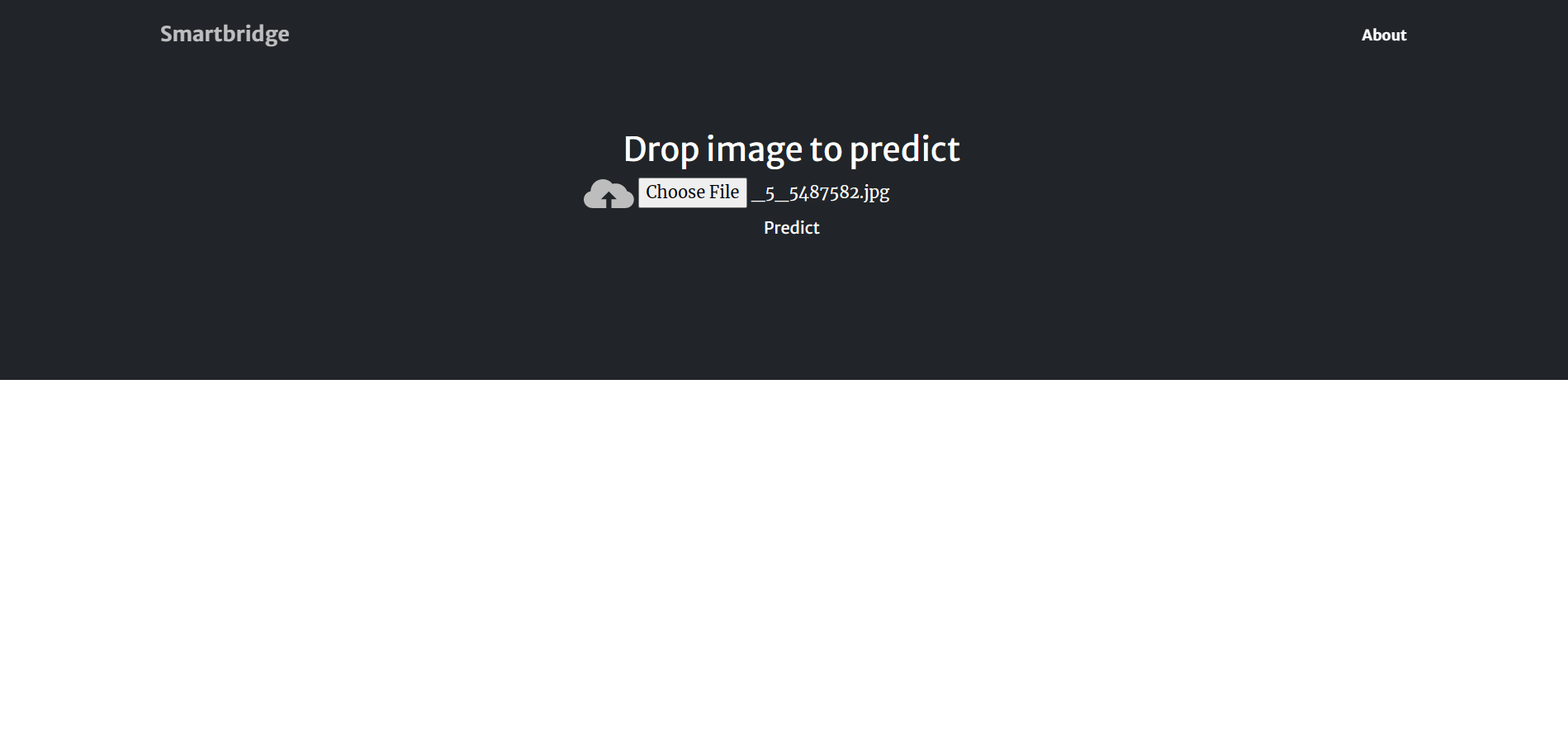


Once you upload the image and click on Predict button, the output will be displayed in the below page

Output 1:



Input 2:



Output2:

