**Coal Classification**

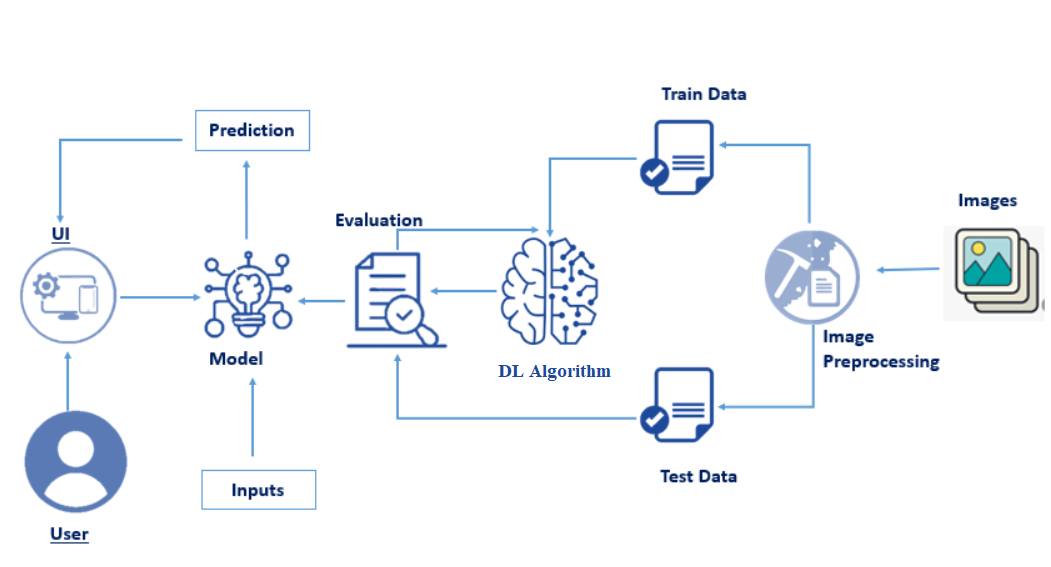
**Introduction:**

Coals occur worldwide and vary significantly in their physical and chemical characteristics for a variety of reasons, both with respect to the organic coal substance and to associated mineral matter that is always present to a varying extent. Coals are an important source of energy, as well as being essential for the production of metallurgical cokes and are also widely used as feedstocks for other industrial process such as the production of gaseous fuels and synthesis gas. However, coals are complex, heterogeneous solids that vary widely in their properties and hence in their suitability for particular applications. Irrespective of the way coal is looked upon, there has always been a need to classify it, particularly by coal geologists, technologists, and scientists. Such a need becomes evident when one realizes that the principle of systematization has always been applied to coals for the purpose of establishing their classification. The purpose of any classification is to group together things that are similar and to distinguish between those that are not. The nature of a classification system will depend upon the particular application for which the system is to be employed. Classifications for coal may be subdivided into those that are scientific/genetic and those that are technical or commercial, for immediate practical or applied use.

**Solution:**

In this project, we will be building a deep learning model that can classify four types of Coal.The most common classification is based on rank, referring to the degree of coalification that has occurred. The rank of a coal is determined primarily by the depth of burial and temperature to which the coal was subjected over time. A web application is integrated with the model, from where the user can upload a coal image like Anthracites, Bituminous, Lignite and Peat, and see the analyzed results on User Inter face.

**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 Spyder

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

Link:  [Click here to](https://www.youtube.com/watch?v=5mDYijMfSzs&feature=emb_logo) 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-neighbors, 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 analyzing visual imagery.

[CNN Basic](https://towardsdatascience.com/basics-of-the-classic-cnn-a3dce1225add)

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

[**Flask Basics**](https://www.youtube.com/watch?v=lj4I_CvBnt0)

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

1. **Download the folder containing all required files & documents to build the project from the given link.**

**Kaggle:** <https://www.kaggle.com/datasets/pattnaiksatyajit/coal-classification>

**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:**

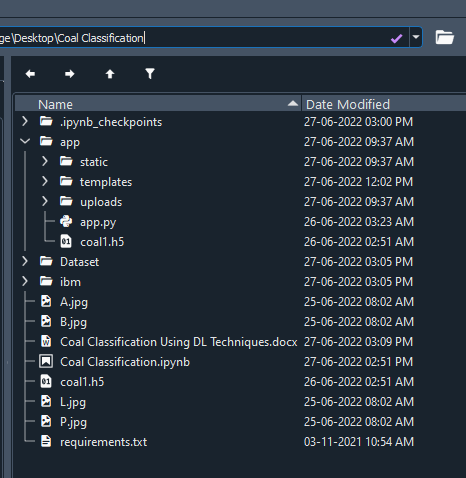
* The user interacts with the UI (User Interface) to choose the image.
* The chosen image analyzed by the model which is integrated with flask application.
* CNN Models analyze the image, then prediction is showcased on the Flask UI.

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

* Data Collection.
  + Create Train and Test Folders.
* Data Preprocessing.
* Import the ImageDataGenerator library
* Configure ImageDataGenerator class
* ApplyImageDataGenerator functionality to Trainset and Testset
* 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 **coal1.h5**
* templates folder contains base.html, index.html pages.

**Milestone 1: Data Collection**

Collect images of coal then organized into subdirectories based on their respective names as shown in the project structure. Create folders of types of coal that need to be recognized.

In this project, we have collected images of 4 types of coal like Anthracite, Bituminous,

Lignite, & Peat and they are saved in the respective sub directories with their respective

names.

You can download the dataset used in this project using the Kaggle link which is posted in the prerequisite section.

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

**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.

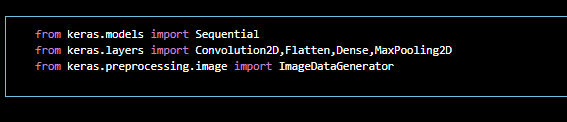
Link : <https://thesmartbridge.com/documents/spsaimldocs/CNNprep.pdf>

**Activity 1: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



**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.

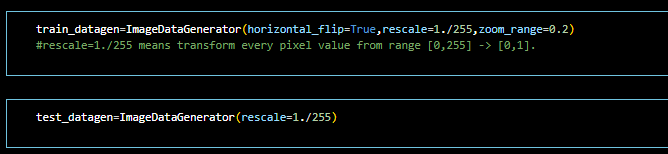
The image flips via the horizontal\_flip and vertical\_flip arguments.

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 Trainset and Testset**

Let us apply ImageDataGenerator functionality to Trainset and Testset by using the following code.For Training set using flow\_from\_directory function.

This function will return batches of images from the subdirectories Anthracite , Bituminous, Lignite, & Peat together with labels 0 to 3 {'Anthracite': 0, 'Bituminous': 1, 'Lignite': 2, 'Peat': 3}

Arguments:

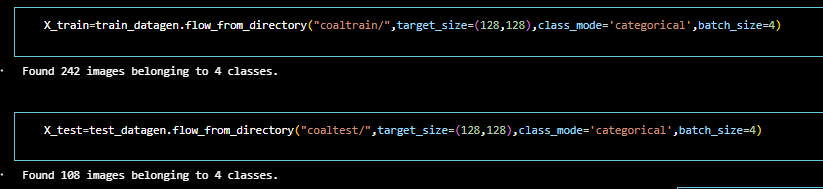
* directory: Directory where the data is located. If labels are "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.
* batch\_size: Size of the batches of data which is 4.
* target\_size: Size to resize images after they are 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).



We notice that 242 images are belonging to 4 classes for training and 108 images belonging to 4 classes for testing purposes.

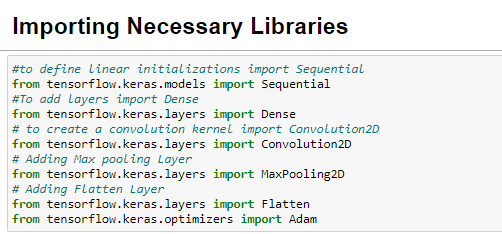
**Milestone 3: Model Building**

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.

* + Link: <https://thesmartbridge.com/documents/spsaimldocs/CNNflow.pdf>

**Activity 1: Importing the Model Building Libraries**

Importing the necessary libraries



**Activity 2: 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.

sequential

**Activity 3: Adding CNN Layers**

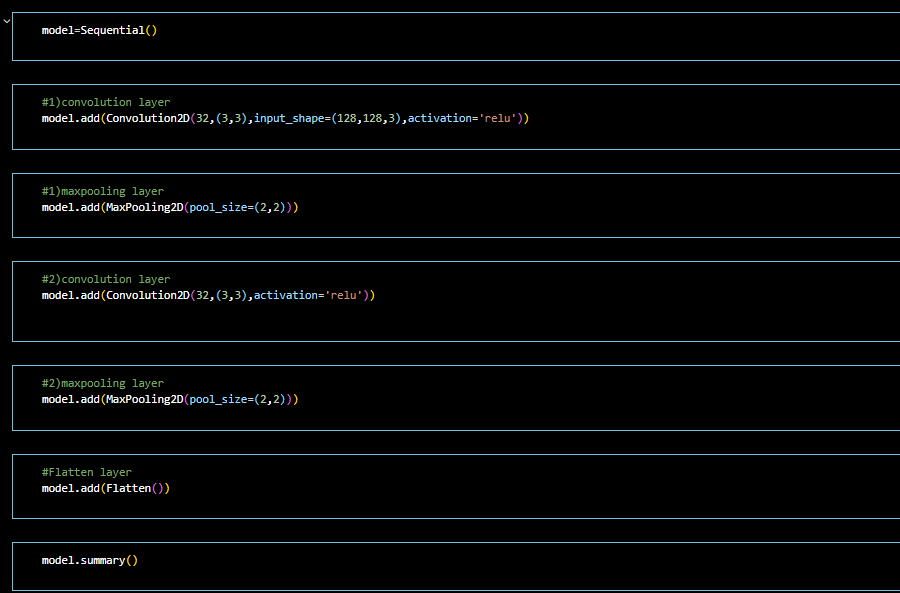
* For information regarding CNN Layers refer to the link

Link: <https://victorzhou.com/blog/intro-to-cnns-part-1/>

* As the input image contains three channels, we are specifying the input shape as (128,128,3).
* We are adding a convolution layer with activation function as “relu” and

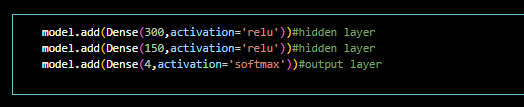
with a small filter size (3,3) and the number of filters (32) followed by a max-pooling layer.

* Max pool layer is used to downsample the input.( Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter)
* Flatten layer flattens the input. Does not affect the batch size.



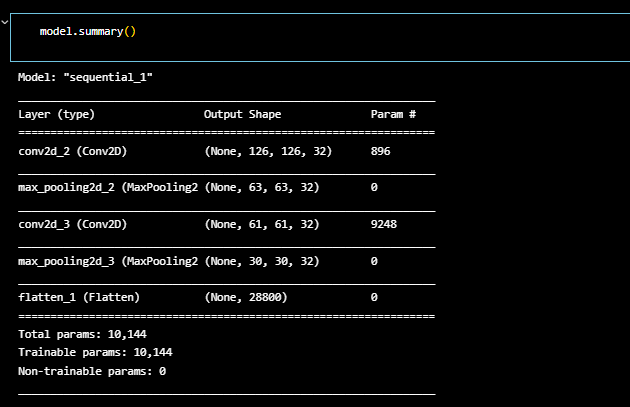
**Activity 5: Adding Dense Layers**

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



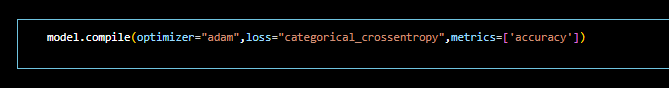
The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

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



**Activity 6: 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 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 7: 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

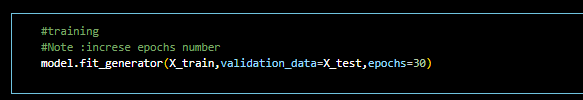
the loss and metrics for any model after any epoch has ended.

* 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

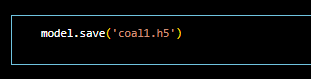
in your dataset divided by the validation batch size.



**Activity 8: Save the Model**

The model is saved with .h5 extension as follows

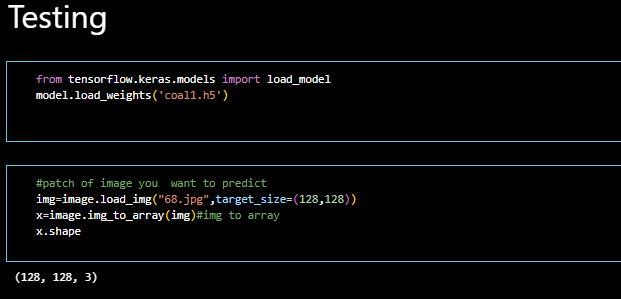
An H5 file is a data file saved in the Hierarchical Data Format (HDF). It contains multidimensional arrays of scientific data.



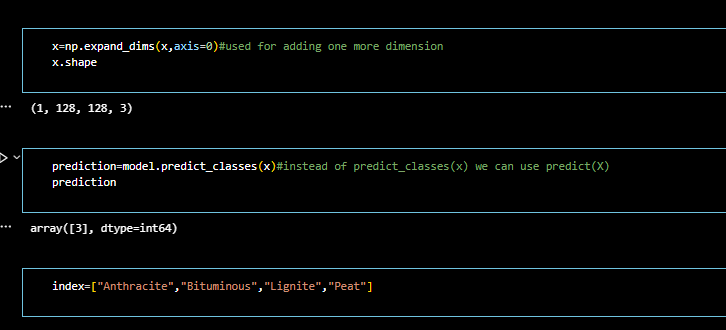
**Activity 9: Test The model**

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.

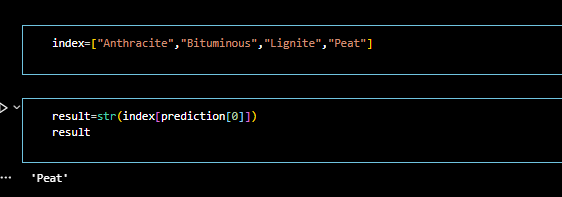
Load the saved model using load\_model



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 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**

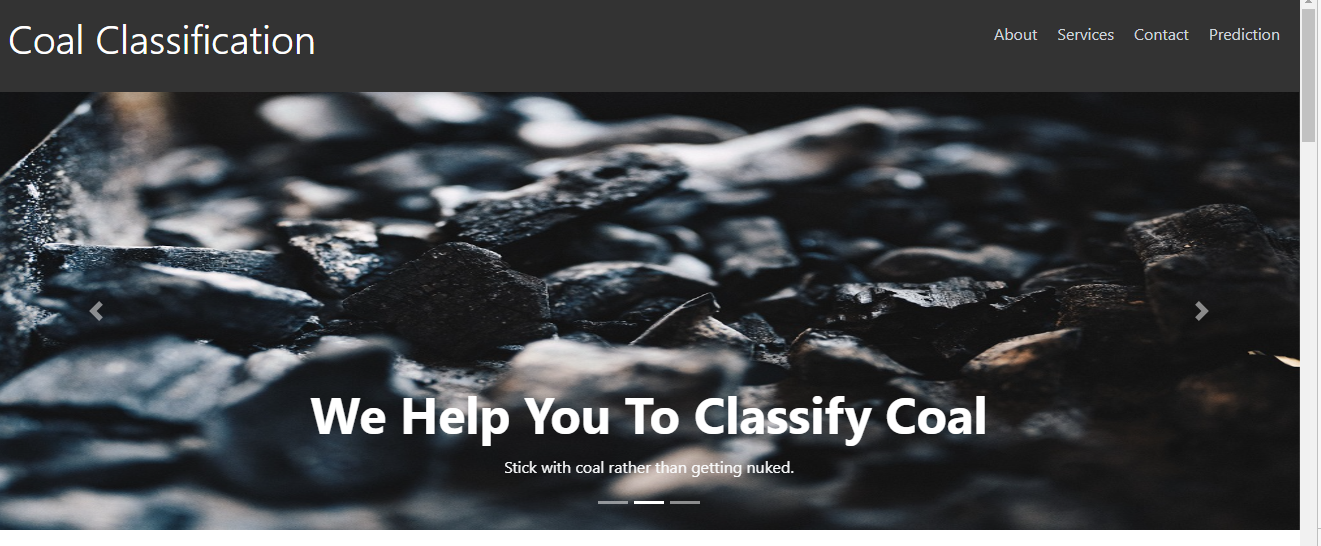
* + We use HTML to create the front end part of the web page.
  + Here, we have created 3 HTML pages- home.html, intro.html, and upload.html
  + home.html displays the home page.
  + Intro.html displays an introduction about the project
  + upload.html gives the emergency alert

For more information regarding HTML

<https://www.w3schools.com/html/>

* + We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.
  + **Link :**[CSS](https://www.w3schools.com/css/) , [JS](https://www.w3schools.com/js/DEFAULT.asp)

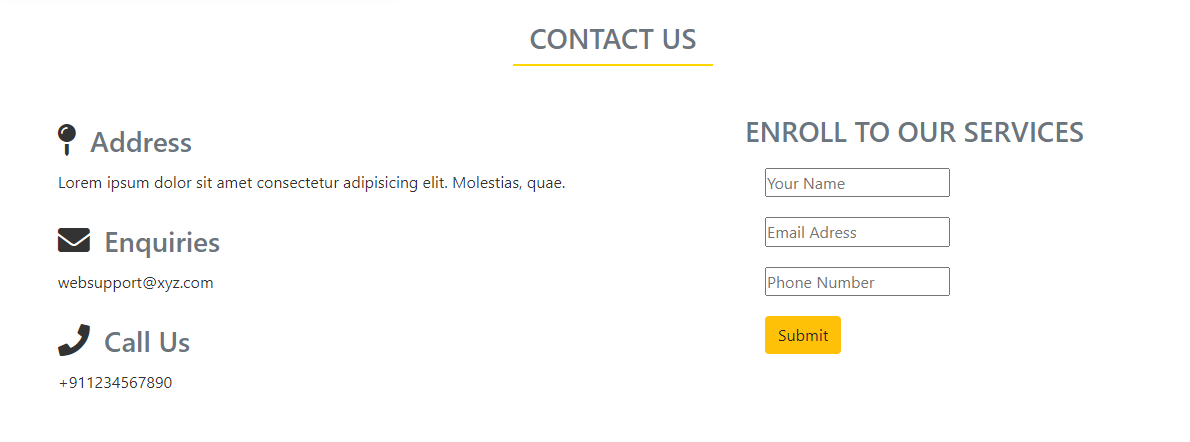
**index.html looks like this**

****

**About Section:-**

****

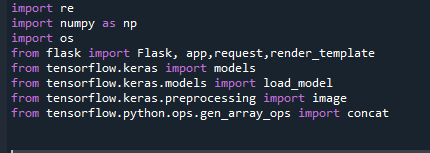
**Contact Us:-**

****

**Activity 2: Build python code**

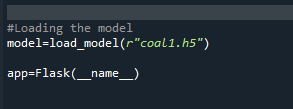
**Task 1: Importing Libraries**

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.

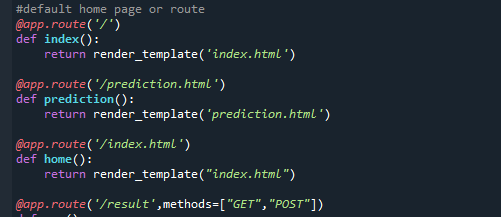
**Task 2: Creating our flask application and loading our model by using load\_model method**



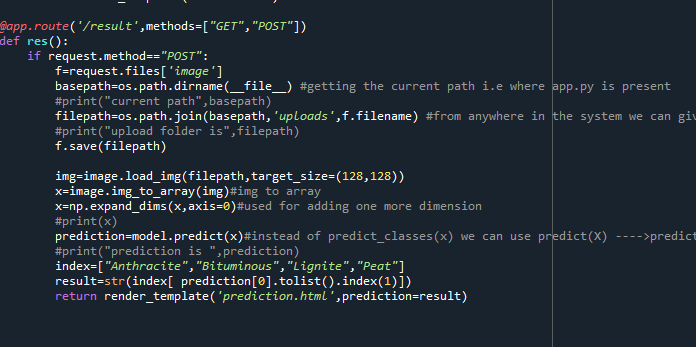
**Task 3: Routing to the html Page**

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.



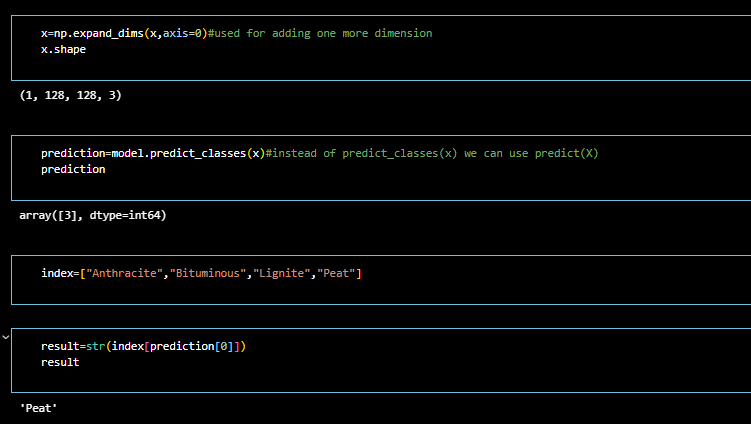
**Showcasing prediction on UI:**

****

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 Keras library we are retrieving the saved picture from the path declared. We are applying some image processing techniques and then sending that preprocessed image to the model for predicting the class. This returns the numerical value of a class (like 0,1 ,2 etc.) 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 predict variable used in the html page.

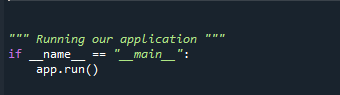
**Predicting the results**

We then proceed to detect all type of Garbage in the input image using model.predict function and the result is stored in the result variable.



**Finally, Run the application**

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

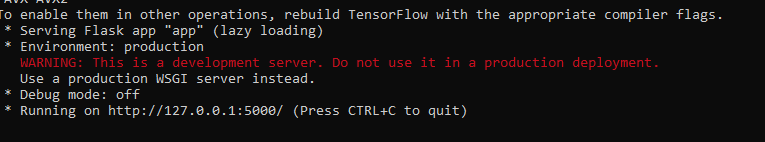


**Activity 3:Run the application**

* 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.



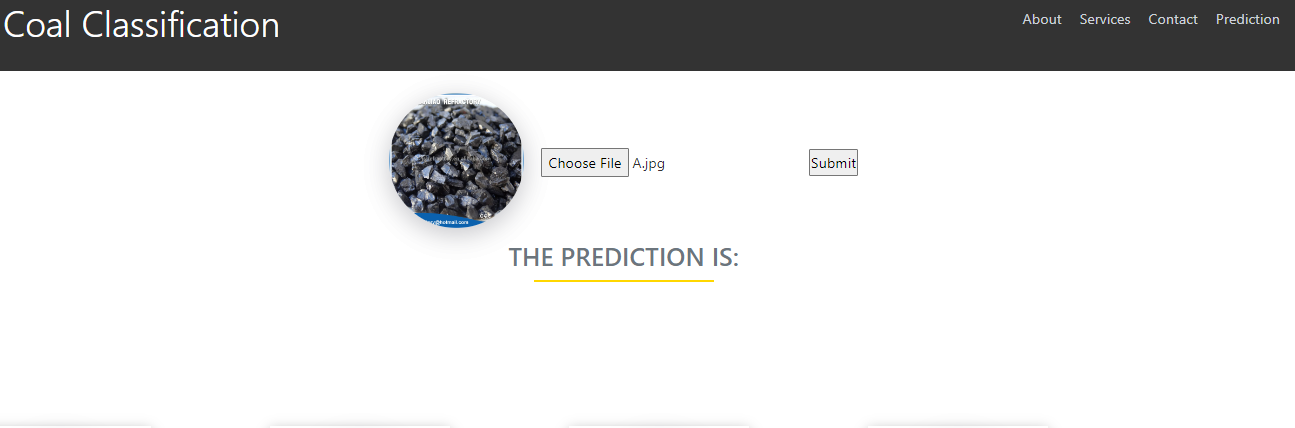
Then it will run on localhost:5000



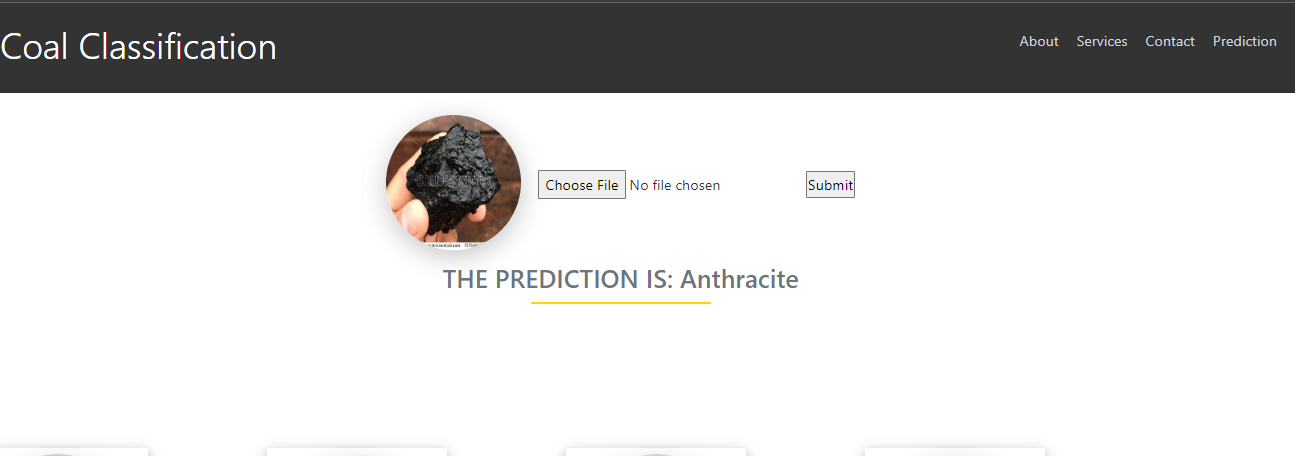
Navigate to the localhost ([http://127.0.0.1:5000/](http://127.0.0.1:8000/))where you can view your web page.

**FINAL OUTPUT:**

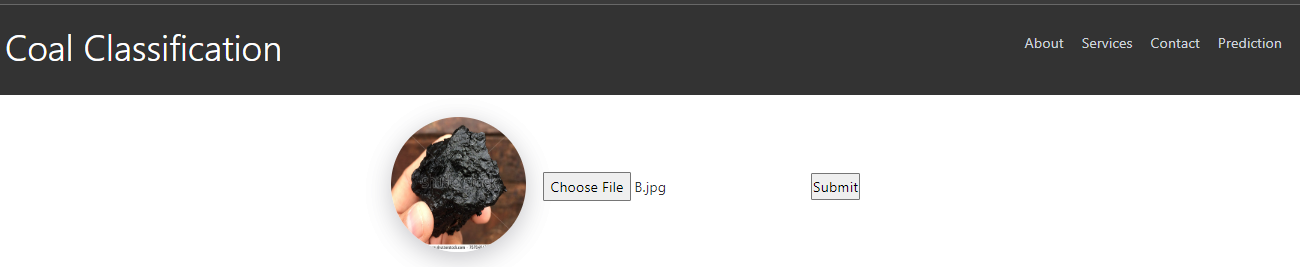
**Input 1:**



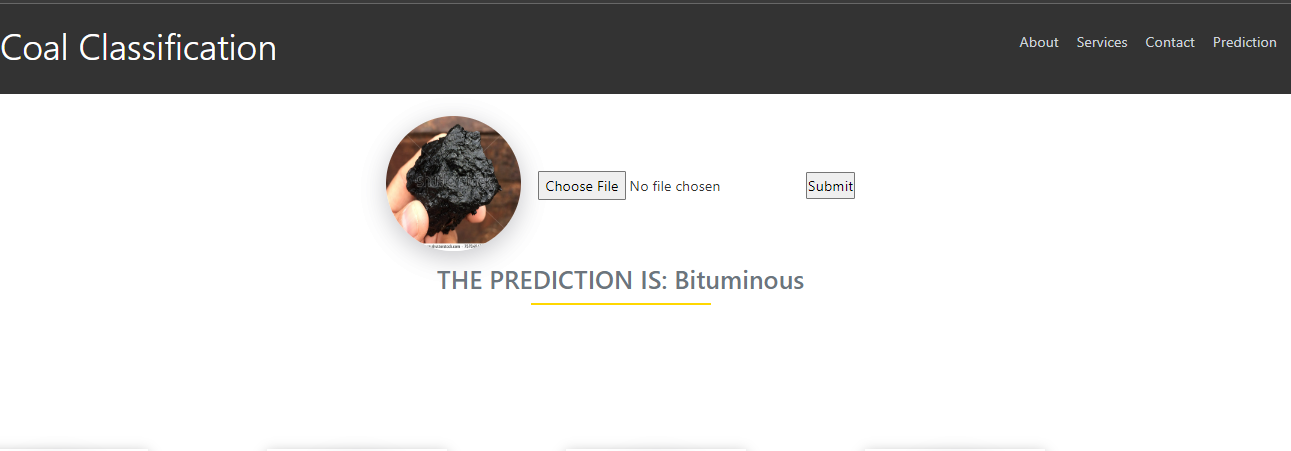
**Output:**

****

**Input 2:**

****

**Output:**

****