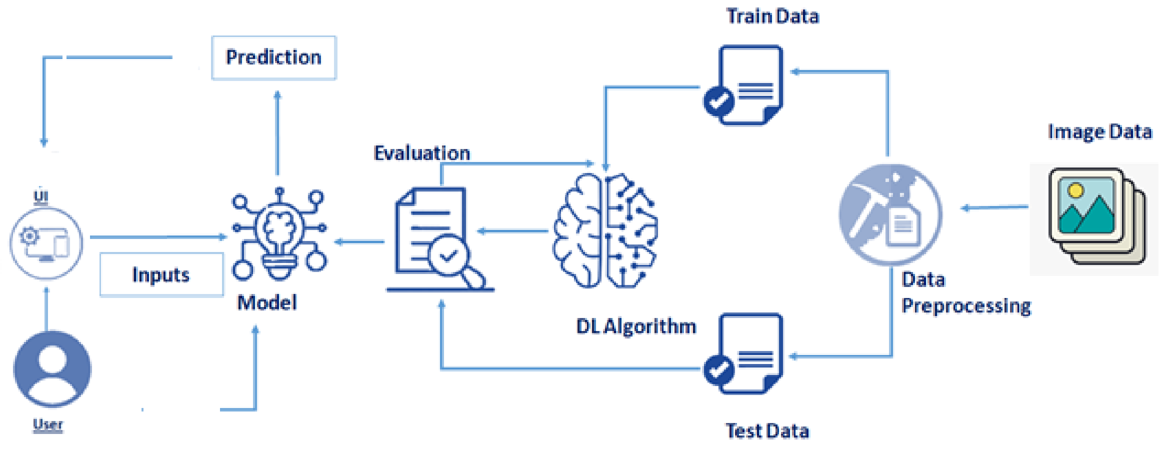
AI-Powered Nutrition Analyzer For Fitness Enthusiasts Using IBM Watson

Project description:

Food is essential for human life and has been the concern of many healthcare conventions. Nowadays new dietary assessment and nutrition analysis tools enable more opportunities to help people understand their daily eating habits, exploring nutrition patterns and maintain a healthy diet. Nutritional analysis is the process of determining the nutritional content of food. It is a vital part of analytical chemistry that provides information about the chemical composition, processing, quality control and contamination of food.

The main aim of the project is to building a model which is used for classifying the fruit depends on the different characteristics like colour, shape, texture etc. Here the user can capture the images of different fruits and then the image will be sent the trained model. The model analyses the image and detect the nutrition based on the fruits like (Sugar, Fibre, Protein, Calories, etc.).

Technical Architecture:



Pre requisites:

In order to develop this project we need to install the following software/packages:

- Anaconda Navigator

For this project, we will be using a Jupyter notebook and Spyder

If you are using anaconda navigator, follow the below steps to download the required packages:

Open anaconda prompt as administrator

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==2.3.0” and click enter.
    - Type “pip install keras==2.4.0” and click enter.
    - Type “pip install Flask” and click enter.

Prior Knowledge:

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

- **ML Concepts**
    - Supervised Learning: https://www.javapoint.com/supervised-machine-learning
    - KNN: htttps://www.javapoint.com/k-nearest-neighbor-algorithm-for-machine-learning
- **Flask Basics:** https://www.youtube.com/watch?v=lj4l-CvBnt0

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) and give the image as input.
- Then the input image is then pass to our flask application,
- And finally with the help of the model which we build we will classify the result and showcase it on the UI.

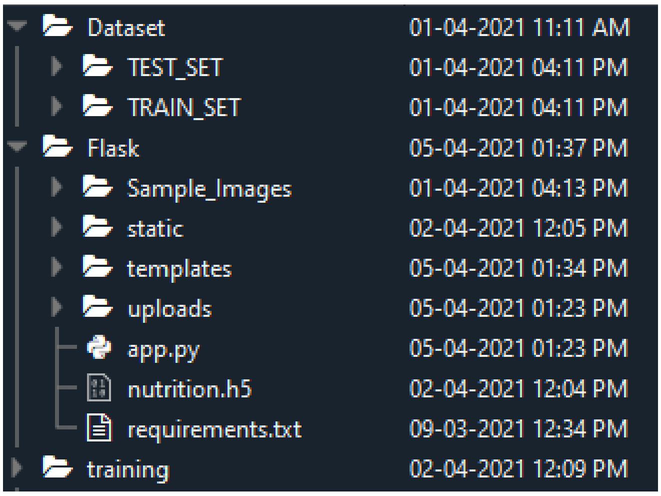
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
    - Collect the dataset or Create the dataset
- 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 that contains files as shown below



- 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 serverside scripting
- we need the model which is saved and the saved model in this content is a nutrition.h5
- templates folder contains home.html, image.html, imageprediction.html pages.
- Statis folder had the css and js files which are necessary for styling the html page and for executing the actions.
- Uploads folder will have the uploaded images(which are already tested).
- Sample_images will have the images which are used to test or upload.
- Training folder contains the trained model file.

Milestone 1:Data Collection

Collect images of different food items organized into subdirectories based on their respective names as shown in the project structure.

Create folders of types of food items that need to be recognized.

In this project, we have collected images of 5 types of food items apples, 'banana', 'orange', 'pineapple' and 'watermelon', they are saved in the respective subdirectories with their respective names.
For more accurate results we can collect images of high resolution and feed the model with more images.
You can download the dataset used in this project using the link below.

Data Set: LINK

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 performing some geometric transformations of images like rotation, scaling, translation, etc.

2.1 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 Keras

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

2.2 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)
test_datagen=ImageDataGenerator(rescale=1./255)
```

2.3 ApplyImageDataGenerator 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'apples', 'banana', 'orange', 'pineapple', 'watermelon' together with labels 0 to 4{'apples': 0, 'banana': 1, 'orange': 2, 'pineapple': 3, 'watermelon': 4}

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. Default: 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).

x_train=train_datagen.flow_from_directory(r"D:\main_project\Nutrition_Image_Analysis\Dataset\TRAIN_SET",target_size=(64,64),batch_size=32,class_mode="categorical")
x_test=train_datagen.flow_from_directory(r"D:\main_project\Nutrition_Image_Analysis\Dataset\TEST_SET",target_size=(64,64),batch_size=32,class_mode="categorical")

We notice that 2626 images are belonging to 5 classes for training and 1055 images belong to 5 classes for testing purposes.

```
x_train.class_indices
```

```
{'APPLES': 0, 'BANANA': 1, 'ORANGE': 2, 'PINEAPPLE': 3, 'WATERMELON': 4}
```

Here we are checking the number of classes in train and test data and counting the number of images in each class of train set data by using the counter function.

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.

3.1Importing The Model Building Libraries

Importing the necessary libraries

```
#importing necessary libraries
import numpy as np
import tensorflow
from tensorflow.keras.models import Sequential
from tensorflow.keras import layers
from tensorflow.keras.layers import Dense,Flatten
from tensorflow.keras.layers import Convolution2D,MaxPooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

3.2Initializing 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.

```
model=Sequential()
```

3.3Adding 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 (64,64,3).

- We are adding a two 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.

3.4Adding Dense Layers

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

```
model=Sequential()
```

```
model.add(Convolution2D(32,(3,3),input_shape=(64,64,3)))
```

```
model.add(MaxPooling2D((2,2)))
```

```
model.add(Flatten())
```

```
model.add(Dense(units=128, kernel_initializer="random_uniform",activation="relu"))
```

```
model.add(Dense(units=5, kernel_initializer="random_uniform",activation="softmax"))
```

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 using it for training and prediction purposes. Keras provides a simple method, a summary to get the full information about the model and its layers.

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 62, 62, 32)        896

 max_pooling2d (MaxPooling2D  (None, 31, 31, 32)       0
 )

 flatten (Flatten)           (None, 30752)             0

 dense (Dense)               (None, 128)               3936384

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3,937,925
Trainable params: 3,937,925
Non-trainable params: 0
```

3.5Configure 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

```
# compile the model
```

```
model.compile(optimizer="adam",loss="categorical_crossentropy",metrics=["accuracy"])
```

3.6Train The Model

Now, let us train our model with our image dataset. The model is trained for 20 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 20 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
  - 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.

```
model.fit_generator(x_train,steps_per_epoch=82, epochs=10, validation_data=x_test,validation_steps=28)
```

3.7Save The Model

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

```
model.save("nutrition.h5")
```

3.8Test 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

```
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
model= load_model("nutrition.h5")
```

Taking an image as input and checking the results

```
from tensorflow.keras.preprocessing import image
path2=r"D:\main_project\Nutrition_Image_Analysis\sample_image\test_image1.jpg"

img=image.load_img(path2,target_size=(64,64))
```

```
x=image.img_to_array(img)
x=np.expand_dims(x,axis=0)
pred=np.argmax(model.predict(x))
pred
```

By using the model we are predicting the output for the given input image

```
index=['APPLES','BANANA','ORANGE','PINEAPPLE','WATERMELON']
result=str(index[pred])
result
```

```
'PINEAPPLE'
```

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 predict the type of food and to know the nutrition content in it. In order to know the nutrition content we will be using an API in this project.

4.1Create HTML Pages

- We use HTML to create the front-end part of the web page.
- Here, we have created 3 HTML pages- home.html, image.html,imageprediction.html, and 0.html.
- home.html displays the home page.
- image.html is used for uploading the image
- imageprediction.html will showcase the output
- 0.html is to showcase the result. It tells the action to be performed on imageprediction.html while showcasing the result.
  For more information regarding HTML go to Link
- We also use JavaScript-main.js and CSS-main.css to enhance our functionality and view of HTML pages.

  - Link :CSS , JS

Home.html looks like this

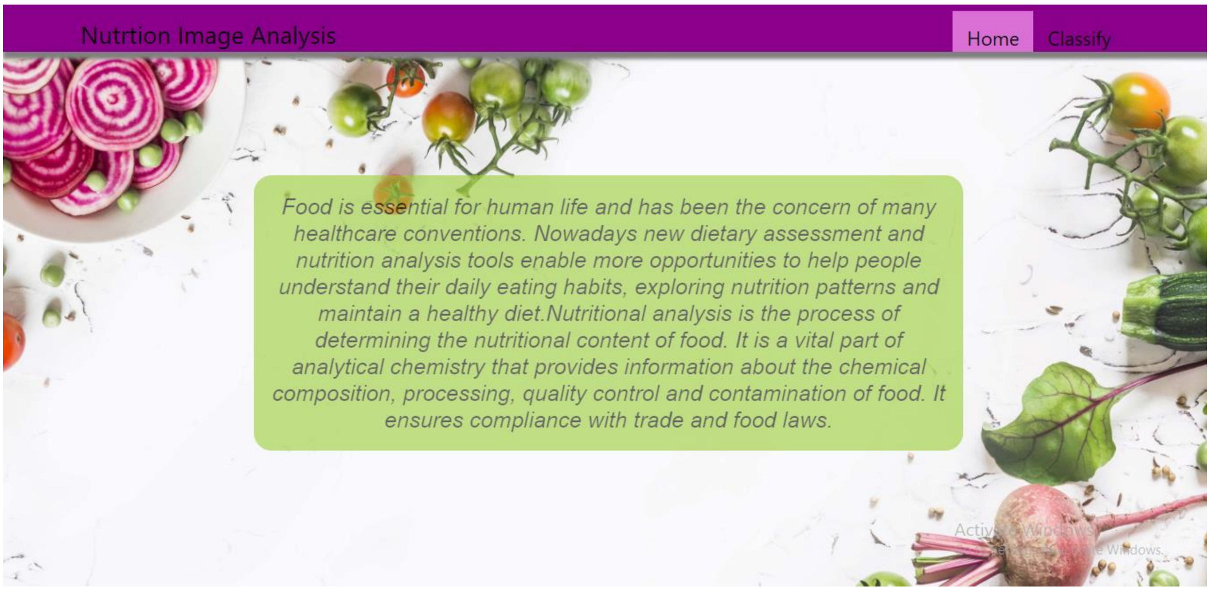
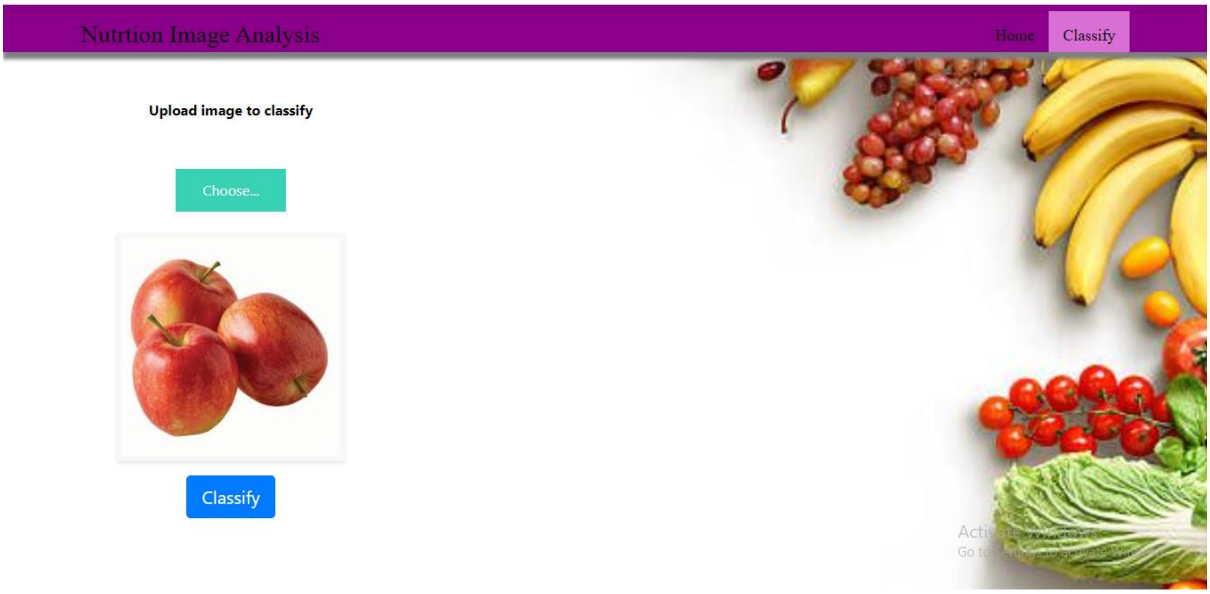
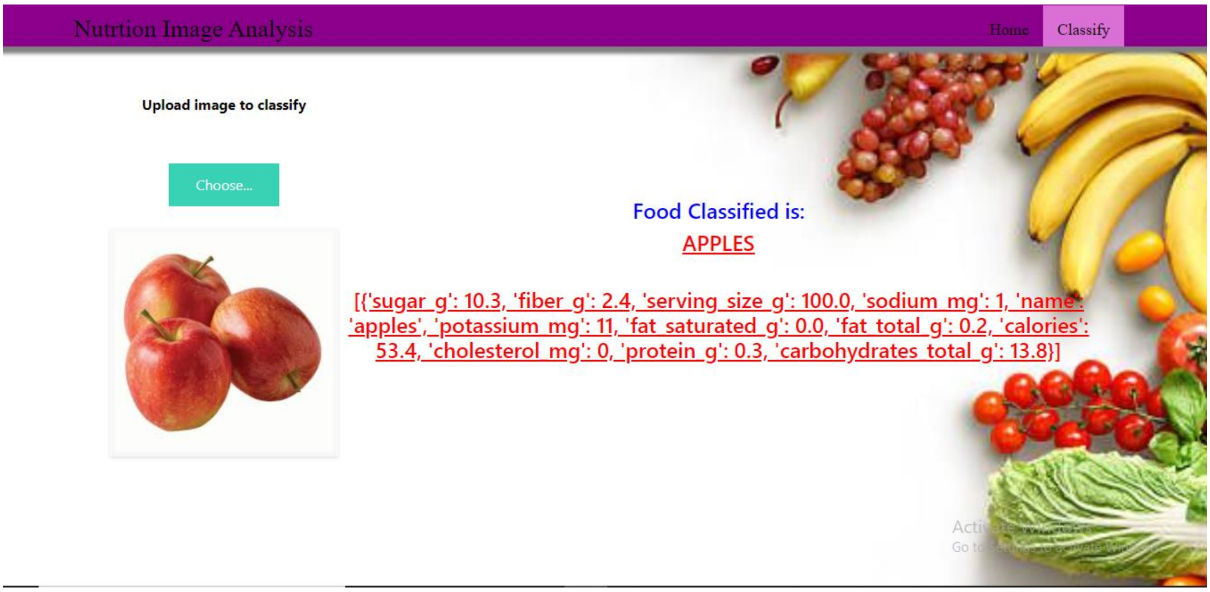


image.html



Imageprediction.html



4.2Build Python Code

Importing Libraries

The first step is usually importing the libraries that will be needed in the program.

```
from flask import Flask,render_template,request
# Flask-It is our framework which we are going to use to run/serve our application.
#request-for accessing file which was uploaded by the user on our application.
import os
import numpy as np #used for numerical analysis
from tensorflow.keras.models import load_model#to load our trained model
from tensorflow.keras.preprocessing import image
import requests
```

Importing the flask module into 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 an argument Pickle library to load the model file.

Creating Our Flask Application And Loading Our Model By Using Load_model Method

Creating our flask application and loading our model by using the load_model method

```
app = Flask(__name__,template_folder="templates") # initializing a flask app
# Loading the model
model=load_model('nutrition.h5')
print("Loaded model from disk")
```

Routing To The Html Page

Here, the declared constructor is used to route to the HTML page created earlier.

In the above example, the '/' URL is bound with the home.html function. Hence, when the home page of the webserver is opened in the browser, the HTML page is rendered. Whenever you enter the values from the HTML page the values can be retrieved using the POST Method. Here, "home.html" is rendered when the home button is clicked on the UI

```
@app.route('/')# route to display the home page
def home():
    return render_template('home.html')#rendering the home page

@app.route('/image1',methods=['GET','POST'])# routes to the index html
def image1():
    return render_template("image.html")
```

When "image is uploaded "on the UI, the launch function is executed

```
@app.route('/predict',methods=['GET', 'POST'])# route to show the predictions in a web UI
def launch():
```

It will take the image request and we will be storing that image in our local system then we will convert the image into our required size and finally, we will be predicting the results with the help of our model which we trained and depending upon the class identified we will showcase the class name and its properties by rendering the respective html pages.

```python
@app.route('/predict',methods=['GET', 'POST'])# route to show the predictions in a web UI
def launch():
    if request.method=='POST':
        f=request.files['file'] #requesting the file
        basepath=os.path.dirname('__file__')#storing the file directory
        filepath=os.path.join(basepath,"uploads",f.filename)#storing the file in uploads folder
        f.save(filepath)#saving the file

        img=image.load_img(filepath,target_size=(64,64)) #load and reshaping the image
        x=image.img_to_array(img)#converting image to an array
        x=np.expand_dims(x,axis=0)#changing the dimensions of the image

        pred=np.argmax(model.predict(x), axis=1)
        print("prediction",pred)#printing the prediction
        index=['APPLES','BANANA','ORANGE','PINEAPPLE','WATERMELON']

        result=str(index[pred[0]])

        x=result
        print(x)
        result=nutrition(result)
        print(result)

        return render_template("0.html",showcase=(result),showcase1=(x))
```

API Integration:

Here we will be using Rapid API

Using RapidAPI, developers can search and test the APIs, subscribe, and connect to the APIs — all with a single account, single API key and single SDK. Engineering teams also use RapidAPI to share internal APIs and microservice documentation.

Reference link: RapidAPI

API used: API
The link above will allow us to test the food item and will result the nutrition content present in the food item.
NOTE: When we keep hitting the API the limit of it might expire. Somaking a smart use of it will be an efficient way.

How to access and use the API will be shown in the video below:
Link

```python
def nutrition(index):


    url = "https://calorieninjas.p.rapidapi.com/v1/nutrition"

    querystring = {"query":index}

    headers = {
        'x-rapidapi-key': "5d797ab107mshe668f26bd044e64p1ffd34jsnf47bfa9a8ee4",
        'x-rapidapi-host': "calorieninjas.p.rapidapi.com"
        }

    response = requests.request("GET", url, headers=headers, params=querystring)

    print(response.text)
    return response.json()['items']
```

Finally, Run the application
This is used to run the application in a localhost. The local host runs on port number 5000.(We can give different port numbers)

```python
if __name__ == "__main__":
   # running the app
    app.run(debug=False)
```

Run The Application

- Open the anaconda prompt from the start menu.
- Navigate to the folder where your app.py resides.
- Now type the "python app.py" command.
- It will show the local host where your app is running on http://127.0.0.1.5000/
- Copy that localhost URL and open that URL in the browser. It does navigate 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.

```
(base) C:\Users\DELL>cd C:\Users\DELL\Desktop\Desk Files\Nutrition Analysis Using Image Classification\Flask

(base) C:\Users\DELL\Desktop\Desk Files\Nutrition Analysis Using Image Classification\Flask>python app.py
```

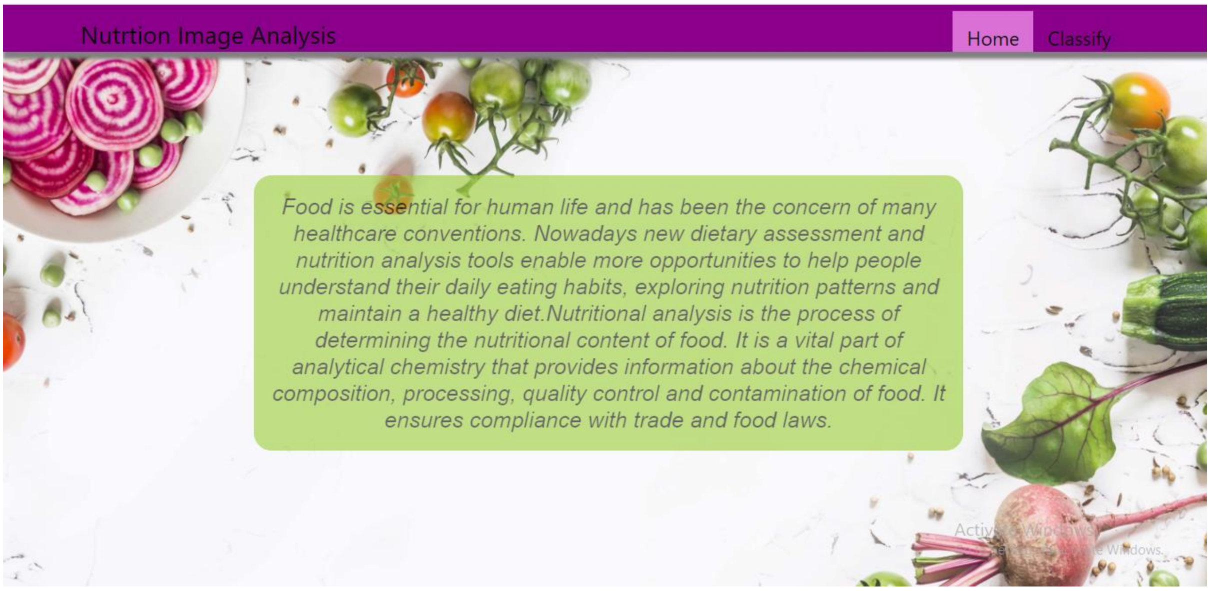
Then it will run on localhost:5000

```
* 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: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

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

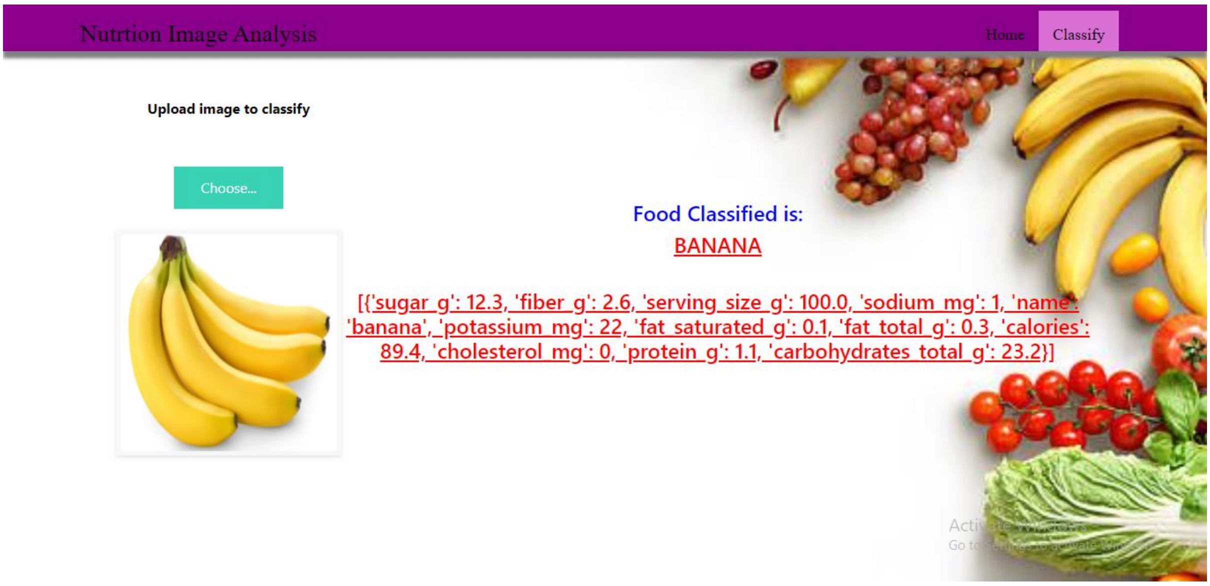
Click on classify button to see the results.

Output screenshots:


Nutrtion Image Analysis
Home
Classify
Food is essential for human life and has been the concern of many healthcare conventions. Nowadays new dietary assessment and nutrition analysis tools enable more opportunities to help people understand their daily eating habits, exploring nutrition patterns and maintain a healthy diet.Nutritional analysis is the process of determining the nutritional content of food. It is a vital part of analytical chemistry that provides information about the chemical composition, processing, quality control and contamination of food. It ensures compliance with trade and food laws.


Nutrtion Image Analysis
Home
Classify
Upload image to classify
Choose...
Food Classified is:
APPLES
[{'sugar_g': 10.3, 'fiber_g': 2.4, 'serving_size_g': 100.0, 'sodium_mg': 1, 'name': 'apples', 'potassium_mg': 11, 'fat_saturated_g': 0.0, 'fat_total_g': 0.2, 'calories': 53.4, 'cholesterol_mg': 0, 'protein_g': 0.3, 'carbohydrates_total_g': 13.8}]


Nutrtion Image Analysis
Home
Classify
Upload image to classify
Choose...
Food Classified is:
PINEAPPLE
[{'sugar_g': 9.9, 'fiber_g': 1.4, 'serving_size_g': 100.0, 'sodium_mg': 0, 'name': 'pineapple', 'potassium_mg': 8, 'fat_saturated_g': 0.0, 'fat_total_g': 0.1, 'calories': 50.8, 'cholesterol_mg': 0, 'protein_g': 0.5, 'carbohydrates_total_g': 13.0}]


Nutrtion Image Analysis
Home
Classify
Upload image to classify
Choose...
Food Classified is:
BANANA
[{'sugar_g': 12.3, 'fiber_g': 2.6, 'serving_size_g': 100.0, 'sodium_mg': 1, 'name': 'banana', 'potassium_mg': 22, 'fat_saturated_g': 0.1, 'fat_total_g': 0.3, 'calories': 89.4, 'cholesterol_mg': 0, 'protein_g': 1.1, 'carbohydrates_total_g': 23.2}]