Recipe Recognition With Deep Learning

Introduction:

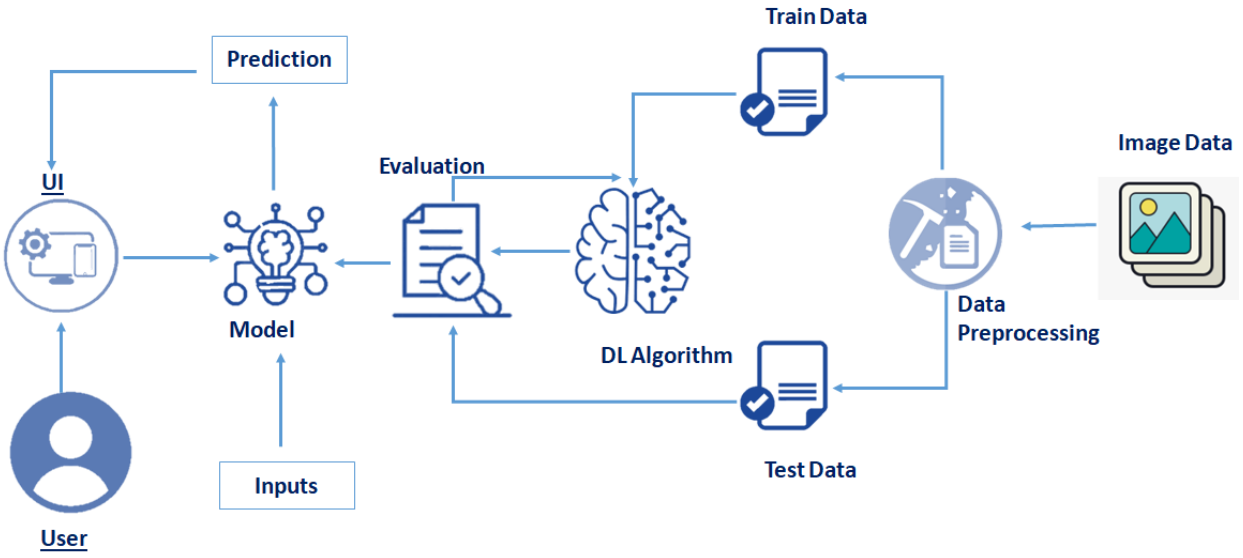
Food is an essential component of our individual and social life. Eating habits have a direct impact on our health and wellbeing, while ingredients, flavor and cooking recipes shape specific cuisines that are part of our personal and collective cultural identities. But there are also interesting applications of automatic food recognition to self-service restaurants and dining halls. For instance, accurate detection and segmentation of the different food items in a food tray can be used for monitoring food intake and nutritional information, and automatic billing to avoid the cashier bottleneck in self-service restaurants. This work deals with the problem of automated recognition of a photographed cooking dish and the subsequent output of the appropriate recipe

Solution:

In this project we focus on applications of automatic food recognition and identify the recipe in a food by using convolutional neural networks. And this model will classify images into food categories and to output a matching recipe.

Technical Architecture:

.



Pre requisites:

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

- **Anaconda navigator:**
    - Refer to the link below to download anaconda navigator
    - **Link : https://www.youtube.com/watch?v=5mDYijMfSzs**
- **Python packages:**
    - open anaconda prompt as administrator
    - Type “pip install tensorflow” (make sure you are working on python 64 bit)
    - Type “pip install flask”.
- **Deep Learning Concepts**

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

    - **Flask Basics** : https://www.youtube.com/watch?v=lj4I_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.
- Knowhow to pre-process/clean the data using different data preprocessing techniques.
- know how to build a web application using Flask framework.

Project Flow:

- User interacts with the UI (User Interface) to upload the image as input
- Uploaded image is analyzed by the model which is integrated
- Once model analyses the uploaded image, the prediction food recipe is showcased on the UI

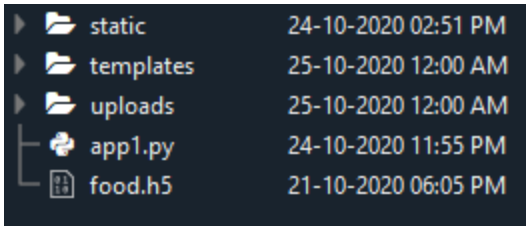
To accomplish this, we have to complete all the activities and tasks listed below

- Data Collection.
    - Collect the dataset or Create the dataset
- Data Preprocessing.
    - Import the ImageDataGenerator library
    - Configure ImageDataGenerator class
    - Apply ImageDataGenerator 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 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



- We are building a Flask Application which 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 food.h5
- The static folder will contain js and css files.
- Whenever we upload an image to predict, that images is saved in uploads folder.

Activities:

Milestone 1: Data Collection

ML depends heavily on data, without data, it is impossible for a machine to learn. It is the most crucial aspect that makes algorithm training possible. In Machine Learning projects, we need a training data set. It is the actual data set used to train the model for performing various actions.

Activity 1 : **Download The dataset**

You can collect datasets from different open sources like kaggle.com, data.gov, UCI machine learning repository etc.

The dataset used for this project was obtained from Kaggle. Please refer to the link given below to download the data set and to know about the dataset

Link: DATASET LINK

The dataset contains three classes:

1. 1.French Fries
2. 2.Pizza
3. 3.Samosa

Milestone 2: Image Preprocessing

- 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 keras

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np#used for numerical analysis
```

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.
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.

```
#setting parameter for Image Data agumentation to the traing data
train_datagen=ImageDataGenerator(rescale=1./255,shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
```

validation_split: Optional float between 0 and 1, fraction of data to reserve for validation.

Activity 3 :Apply ImageDataGenerator functionality to Trainset and Testset

Let us apply ImageDataGenerator functionality to Trainset and Testset by using the following code
For Trainingset using flow_from_directory function.

This function will return batches of images from the subdirectories Infected and uninfected, together with labels 0 and 1 (0 corresponding to Infected and 1 corresponding to uninfected).
Arguments:

- directory: Directory where the data is located. If labels is "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 to 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).

```
#performing data agumentation to train data
x_train=train_datagen.flow_from_directory(directory=r'E:\FOOD Classification\Food-Classification-from-I
                                          ,target_size=(64,64),batch_size=32,class_mode='categorical')
```

There are 2400 images belonging to 3 classes (French fries, Pizza and Samosa)

For Testset

```
#Image Data agumentation to the testing data
test_datagen=ImageDataGenerator(rescale=1./255)
```

```
#performing data agumentation to test data
x_test=test_datagen.flow_from_directory(directory=r'E:\FOOD Classification\Food-Classification-from-Ima
                                        ,target_size=(64,64),batch_size=32,class_mode='categorical')
```

There are 600 images belonging to 3 classes (French fries, Pizza and Samosa)

Milestone 3: Model Building

- Link: https://thesmartbridge.com/documents/spsaimldocs/CNNflow.pdf

Activity 1 : Importing the Model Building Libraries

Importing the necessary libraries

```
import numpy as np#used for numerical analysis
import tensorflow #open source used for both ML and DL for computation
from tensorflow.keras.models import Sequential #it is a plain stack of layers
from tensorflow.keras import layers #A layer consists of a tensor-in tensor-out computation function
#Dense layer is the regular deeply connected neural network layer
from tensorflow.keras.layers import Dense,Flatten
#Faltten-used fot flattening the input or change the dimension
from tensorflow.keras.layers import Conv2D,MaxPooling2D #Convolutional layer
#MaxPooling2D-for downsampling the image
```

Activity 2 : Initializing the model

Sequential model is a linear stack of layers.You can create a Sequential model by passing a list of layer instances to the constructor: from keras. models import Sequential from keras as follows.

```
# create model
model=Sequential()
```

Activity 3 : Adding CNN Layers

- For information regarding CNN Layers reer to the link
  Link: https://victorzhou.com/blog/intro-to-cnns-part-1/

- We are adding a convolution layer with activation function as “relu” and
  with a small filter size (3,3) and number of filters (32) followed by a max pooling layer.
- Maxpool layer is used to downsample the input.
- Dropout layer is used to deactivate the neurons randomly.Dropout(0.2) indicates that 20 % of the neurons are deactivated.
- Flatten layer flattens the input. Does not affect the batch size.

```
# adding model layer
model.add(Conv2D(32,(3,3),input_shape=(64,64,3),activation='relu'))#convolutional layer
model.add(MaxPooling2D(pool_size=(2,2))) #MaxPooling2D-for downsampling the input

model.add(Conv2D(32,(3,3),activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.2))#droping input randomly for preventing from overfiting


model.add(Flatten())#flatten the dimension of the image
model.add(Dense(32))#deeply connected neural network layers.
```

Activity 5: Adding Dense Layers

Dense layer is deeply connected neural network layer. It is most common and frequently used layer.

```
model.add(Dense(3,activation='softmax'))#output layer with 3 neurons
```

Understanding the model is 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.

```
model.summary()#summary of our model
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 32)                200736
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 99
=================================================================
Total params: 210,979
Trainable params: 210,979
Non-trainable params: 0
_________________________________________________________________
```

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 training phase. Loss function is used to find error or deviation in the learning process. Keras requires loss function during model compilation process.
- Optimization is an important process which optimize the input weights by comparing the prediction and the loss function. Here we are using adam optimizer
- Metrics is used to evaluate the performance of your model. It is similar to loss function, but not used in training process

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

Activity 7: Train The model

Now, let us train our model with our image dataset.
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 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.

```
# Fit the model
model.fit_generator(generator=x_train,steps_per_epoch = len(x_train),
                    epochs=80, validation_data=x_test,validation_steps = len(x_test))
```

```
WARNING:tensorflow:From <ipython-input-10-15660ad16113>:2: Model.fit_generator (from tensorflow.python.keras.engine.training) is deprecated and will be removed in a future version.
Instructions for updating:
Please use Model.fit, which supports generators.
Epoch 1/80
75/75 [==============================] - 106s 1s/step - loss: 1.0982 - accuracy: 0.4125 - val_loss: 0.9337 - val_accuracy: 0.5517
Epoch 2/80
75/75 [==============================] - 123s 2s/step - loss: 0.9269 - accuracy: 0.5529 - val_loss: 0.8210 - val_accuracy: 0.6000
Epoch 3/80
75/75 [==============================] - 242s 3s/step - loss: 0.8474 - accuracy: 0.6112 - val_loss: 0.9252 - val_accuracy: 0.5400
Epoch 4/80
75/75 [==============================] - 174s 2s/step - loss: 0.8182 - accuracy: 0.6396 - val_loss: 1.0349 - val_accuracy: 0.5567
Epoch 5/80
75/75 [==============================] - 60s 794ms/step - loss: 0.7738 - accuracy: 0.6542 - val_loss: 0.7489 - val_accuracy: 0.6783
Epoch 6/80
75/75 [==============================] - 55s 730ms/step - loss: 0.7185 - accuracy: 0.6796 - val_loss: 0.7720 - val_accuracy: 0.6817
Epoch 7/80
75/75 [==============================] - 55s 727ms/step - loss: 0.6998 - accuracy: 0.6963 - val_loss: 0.9861 - val_accuracy: 0.5950
Epoch 8/80
75/75 [==============================] - 46s 616ms/step - loss: 0.6810 - accuracy: 0.7000 - val_loss: 0.7379 - val_accuracy: 0.6933
Epoch 9/80
75/75 [==============================] - 48s 643ms/step - loss: 0.6899 - accuracy: 0.7000 - val_loss: 0.6407 - val_accuracy: 0.7300
Epoch 10/80
75/75 [==============================] - 51s 679ms/step - loss: 0.6424 - accuracy: 0.7400 - val_loss: 0.6546 - val_accuracy: 0.7300
```

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.

```
# Save the model
model.save('food.h5')
```

Activity 9: Test The model

Evaluation is a process during development of the model to check whether the model is best fit for the given problem and corresponding data.
Load the saved model using load_model

```
from tensorflow.keras.models import load_model
from keras.preprocessing import image
model = load_model("food.h5") #loading the model for testing
```

Taking an image as input and checking the results

```
img = image.load_img(r"E:\FOOD Classification\Food-Classification-from-Images-Using-Convolutional-Neural
x = image.img_to_array(img)#image to array
x = np.expand_dims(x,axis = 0)#changing the shape
pred = model.predict_classes(x)#predicting the classes
pred
```

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

```
index=['french_fries', 'pizza', 'samosa']
result=str(index[pred[0]])
result
```

```
'samosa'
```

Index of zero is made print here. So, French fries will be printed.

Milestone 4: Application Building

Activity **1 : Create HTML Pages**

- We use HTML to create the front end part of the web page.
- Here, we created 4 html pages- about.html, base.html, index6.html, info.html.
- about.html displays home page.
- Info.html displays all important details to be known about Malaria.
- base.html and index6.html accepts input from the user and predicts the values.
  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 :** https://www.w3schools.com/css/
  - **https://www.w3schools.com/js/DEFAULT.asp**

Activity **2 :** Build python code

- Let us build flask file'app.py' which is a web framework written in python for server-side scripting. Let's see step by step procedure for building the backend application.
- App starts running when "__name__" constructor is called in main.
- render_template is used to return html file.
- "GET" method is used to take input from the user.
- "POST" method is used to display the output to the user.

- Importing Libraries

```
import os
import numpy as np #used for numerical analysis
from flask import Flask,request,render_template#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.
#render_template- used for rendering the html pages
from tensorflow.keras.models import load_model#to load our trained model
from tensorflow.keras.preprocessing import image
```

- Routing to the html Page

```
app=Flask(__name__)#our flask app
model=load_model('food.h5')#loading the model

@app.route("/") #default route
def upload_file():
    return render_template("RR.html")#rendering html page

@app.route("/about") #route about page
def upoad_file1():
    return render_template("RR.html")#rendering html page


@app.route("/upload") # route for info page
def upload_file2():
    return render_template("RRP.html")#rendering html page
```

- Showcasing prediction on UI
  When the image is uploaded, it predicts whether the uploaded the image is French fries or pizza or samosa. If the image predicts value as 1, then it is displayed as "Pizza". If the image predicts value as 0, then it is displayed as "French Fries". If the image predicts value as 2, then it is displayed as "Samosa".

```python
@app.route("/predict",methods=["GET","POST"]) #route for our prediction
def upload():
    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 array
        x=np.expand_dims(x,axis=0)#changing the dimensions of the image

        pred = model.predict_classes(x)  # predicting classes
        print(pred)  # printing the prediction
        index = ['French Fries', 'Pizza', 'Samosa']
        result = str(index[pred[0]])
        if (result=="French Fries"):
            return render_template("0.html",showcase =  str(result))
        elif (result=="Pizza"):
            return render_template("1.html",showcase =  str(result))
        else:
            return render_template("2.html",showcase =  str(result))
        #return result#resturing the result
    else:
        return None

#port = int(os.getenv("PORT"))
if __name__=="__main__":
    app.run(debug=False)#running our app
    #app.run(host='0.0.0.0', port=8000,debug=False)
```

- Run The app in local browser
  - Open anaconda prompt from the start menu
  - Navigate to the folder where your python script is.
  - Now type “python app.py” command
  - Navigate to the localhost where you can view your web page

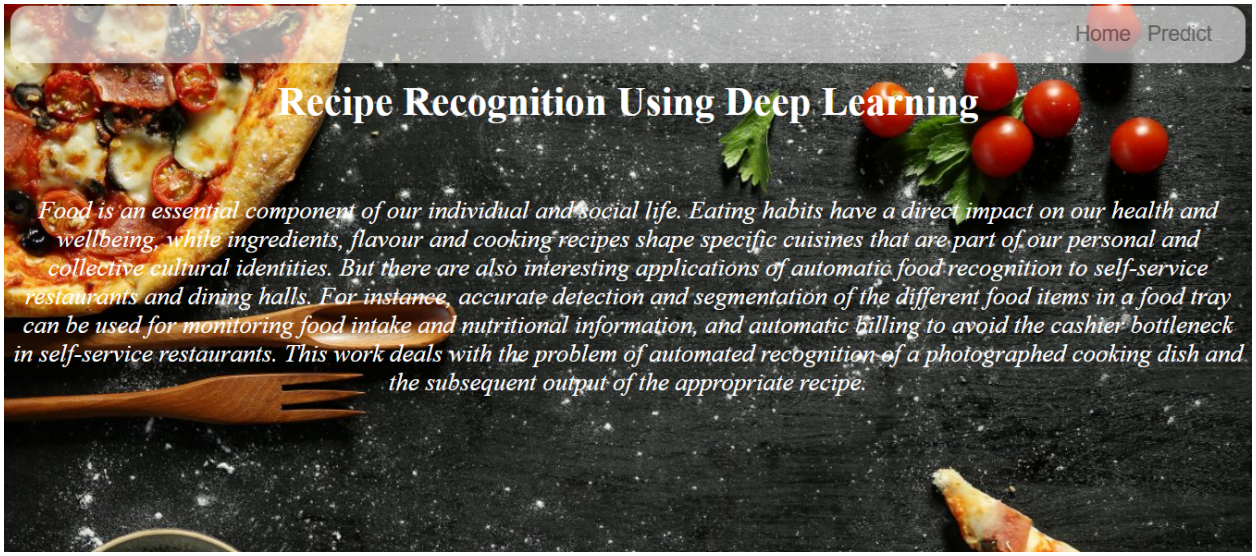
```
(base) E:\FOOD Classification>python app1.py
```

Then it will run on localhost:5000

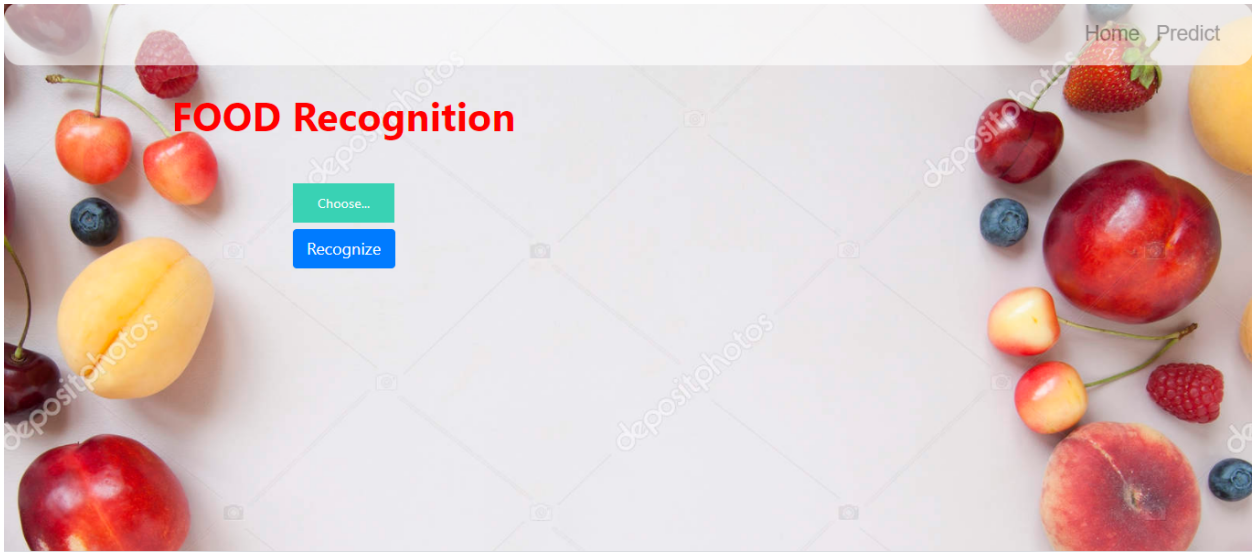
```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

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

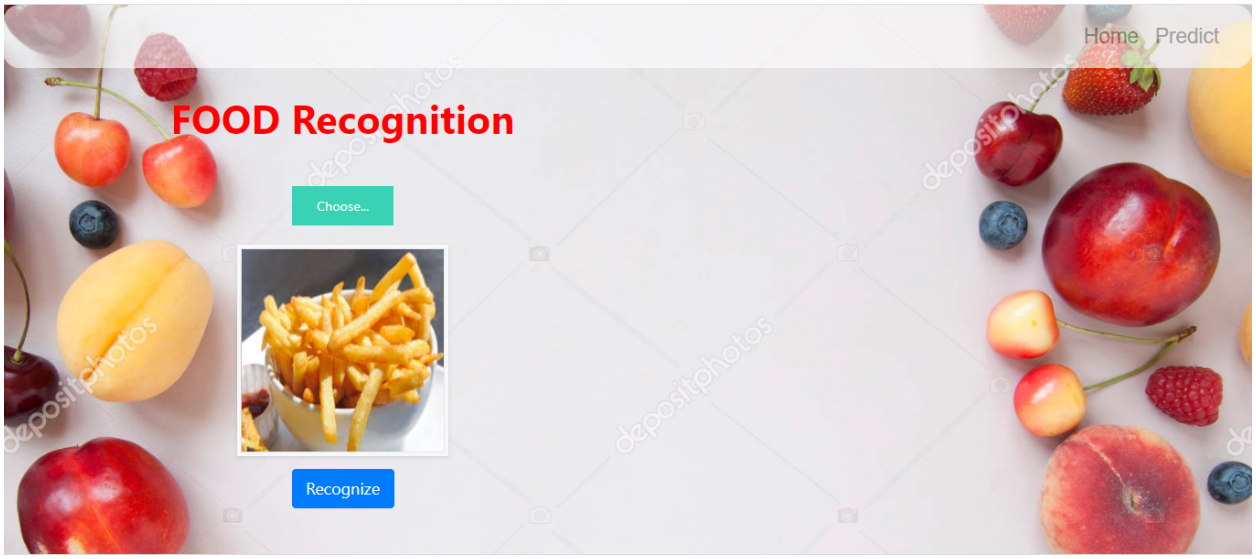
Let’s see how our about.html page looks like:



When “predict” button is clicked it will redirect to prediction page

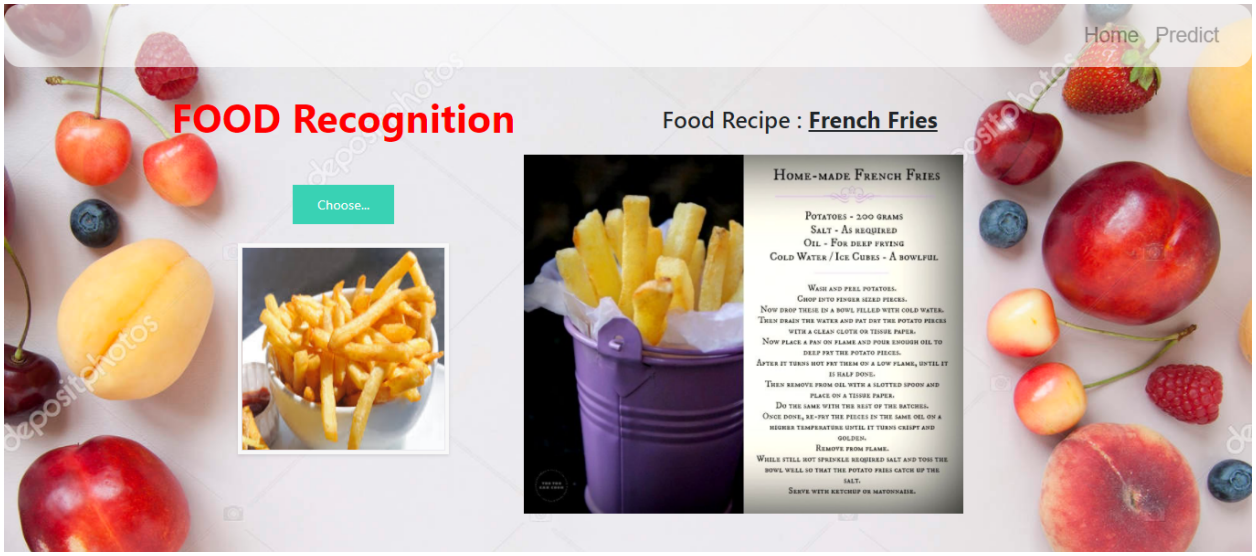


In order to get the output one need to upload the image and click on recognize.

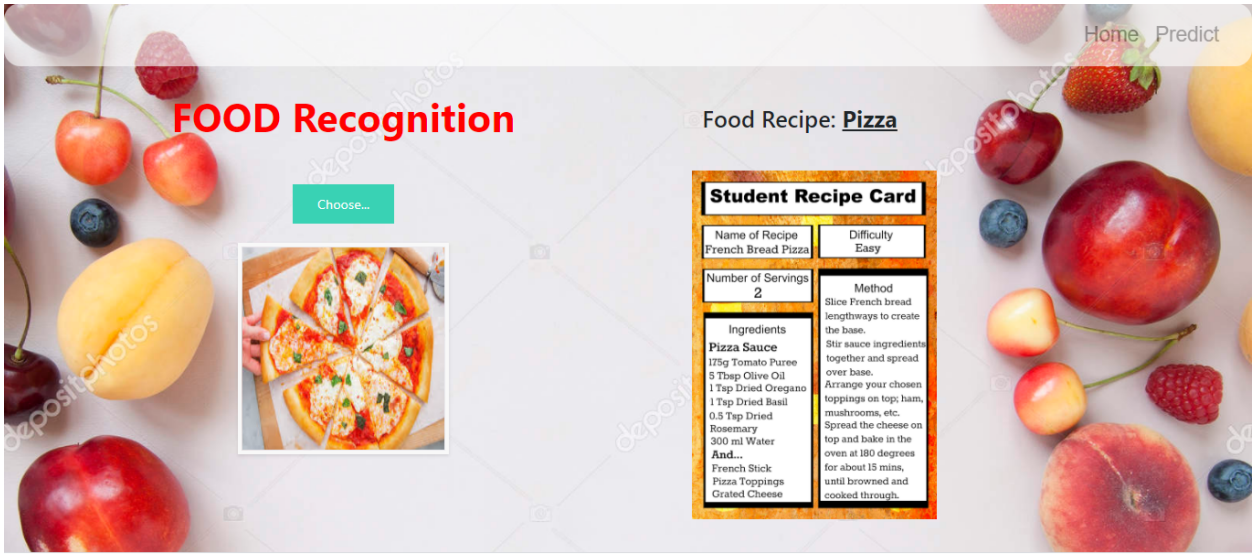


- Upload the image and click on Predict button to view the result on “RRP.html” page on localhost.

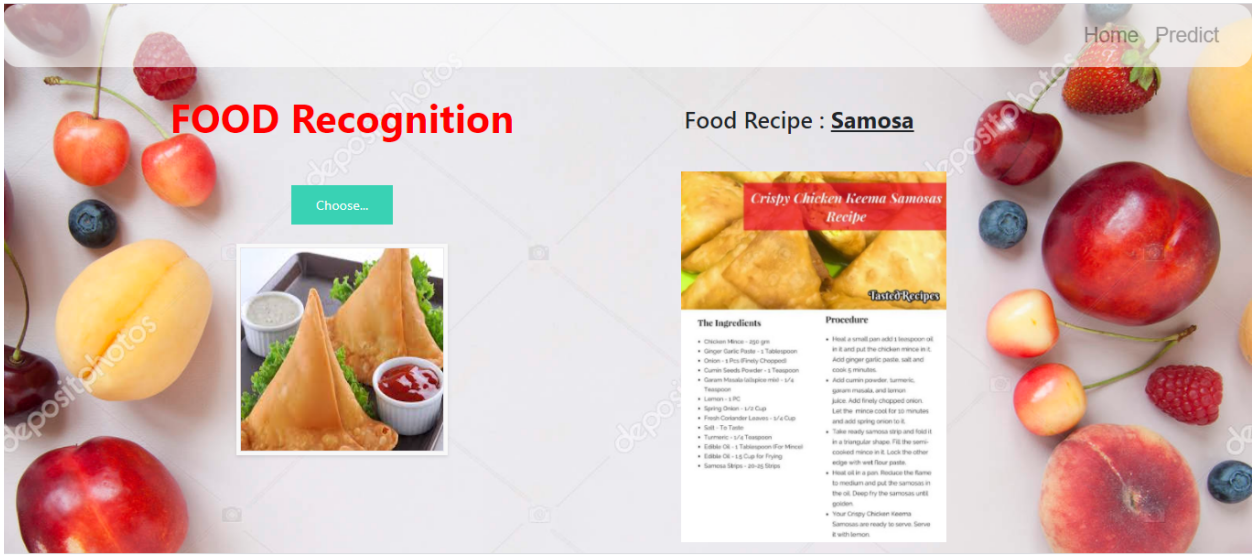
Prediction-1:



sIf the uploaded image is French Fries the matching recipe for French Fries will be show cased.



If the uploaded image is Pizza the matching recipe for Pizza will be show cased.



If the uploaded image is Samosa the matching recipe for Samosa will be show cased.