AI-Powered Nutrition Analyzer for Fitness Enthusiasts

PROJECT OVERVIEW

❖ Introduction:

The AI-powered Nutrition Analyzer for Fitness Enthusiasts is a comprehensive solution designed to analyse and optimize nutritional intake for individuals pursuing fitness goals. Leveraging artificial intelligence (AI) algorithms, the system processes dietary data, user preferences, and fitness objectives to provide personalized nutritional recommendations and insights. This project aims to empower fitness enthusiasts with actionable information to support their health and fitness journey effectively.

Scenario 1: Personalized Meal Planning

Users can input their dietary preferences, fitness goals, and health restrictions into the AI-powered Nutrition Analyzer. The system then generates personalized meal plans that meet their nutritional needs, ensuring they consume the right balance of macronutrients (carbohydrates, proteins, fats) and micronutrients (vitamins, minerals) to support their fitness goals.

Scenario 2: Nutrient Analysis and Tracking

Fitness enthusiasts can track their daily food intake using the AI-powered Nutrition Analyzer. The system analyses the nutritional content of each food item consumed and provides real-time feedback on nutrient intake. Users can monitor their progress, identify areas for improvement, and make informed adjustments to their diet for optimal performance.

Scenario 3: Recipe Enhancement and Modification

The system can also analyse recipes and suggest modifications to enhance their nutritional value. Users can explore healthier alternatives, substitutions, and portion adjustments to create balanced and nutritious meals that align with their fitness objectives. This feature encourages users to experiment with new recipes while maintaining a focus on health and wellness.

Technical Architecture:
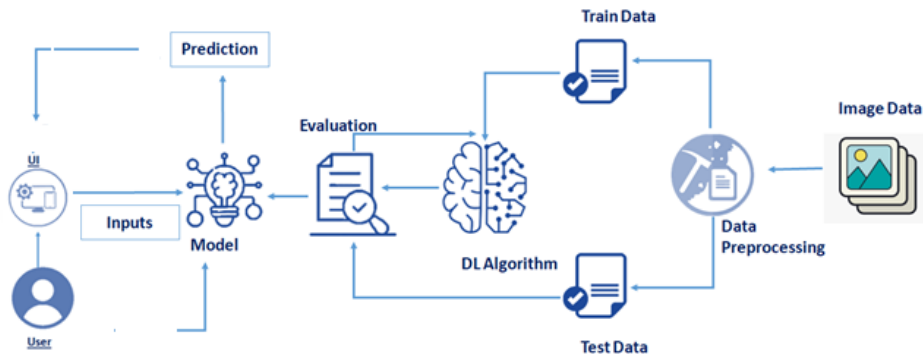
❖ Project Objectives:

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 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
  - Apply ImageDataGenerator functionality to Trainset and Test set
- 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

❖ Prerequisites:

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

Anaconda Navigator
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 great tools like Jupyter Lab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, Visual Studio Code.
For this project, we will be using a Jupyter notebook and Spyder
To install the Anaconda navigator and to know how to use Jupyter Notebook & Spyder using Anaconda watch the video

Flask - Web framework used for building Web applications.

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

Open anaconda prompt as administrator

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

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.

Deep Learning Concepts.

Artificial Neural Networks

Convolution Neural Networks

A convolutional neural network is a class of deep neural networks, most commonly applied to analysing visual imagery

❖ 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

❖ 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: Dataset

Note: For better accuracy train on more images
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.

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

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

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.

```python
# Image Data Agumentation

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

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).

```
### Loading our data and performing data agumentation

#performing data agumentation to train data
x_train = train_datagen.flow_from_directory(
    r'C:\Users\DELL\Desktop\Desk Files\Nutrition Analysis Using Image Classification\DataSet\TRAIN_SET',
    target_size=(64, 64),batch_size=5,color_mode='rgb',class_mode='sparse')
#performing data agumentation to test data
x_test = test_datagen.flow_from_directory(
    r'C:\Users\DELL\Desktop\Desk Files\Nutrition Analysis Using Image Classification\DataSet\TEST_SET',
    target_size=(64, 64),batch_size=5,color_mode='rgb',class_mode='sparse')

Found 2626 images belonging to 5 classes.
Found 1055 images belonging to 5 classes.
```

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

```
print(x_train.class_indices)#checking the number of classes

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

print(x_test.class_indices)#checking the number of classes

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

from collections import Counter as c
c(x_train .labels)

Counter({0: 606, 1: 445, 2: 479, 3: 621, 4: 475})
```

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.

❖ 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.

Importing the Model Building Libraries

Importing the necessary libraries

```
### Importing Neccessary 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,Dropout #Convolutional Layer
#MaxPooling2D-for downsampling the image
from keras.preprocessing.image import ImageDataGenerator
```

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.

```
model=Sequential()
```

Adding CNN Layers:

- For information regarding CNN Layers

- 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.

```
### Creating the model

# Initializing the CNN
classifier = Sequential()

# First convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), input_shape=(64, 64, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Second convolution layer and pooling
classifier.add(Conv2D(32, (3, 3), activation='relu'))

# input_shape is going to be the pooled feature maps from the previous convolution layer
classifier.add(MaxPooling2D(pool_size=(2, 2)))

# Flattening the layers
classifier.add(Flatten())
```

Adding Dense Layers:

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

```
# Adding a fully connected layer
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dense(units=5, activation='softmax')) # softmax for more than 2
```

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.

```
classifier.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, 128)               802944
_____
dense_1 (Dense)              (None, 5)                 645
=================================================================
Total params: 813,733
Trainable params: 813,733
Non-trainable params: 0
_____
```

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

```
### Compiling the model

# Compiling the CNN
# categorical_crossentropy for more than 2
classifier.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

Train 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.

```
## Fitting the model

classifier.fit_generator(
        generator=x_train,steps_per_epoch = len(x_train),
        epochs=20, validation_data=x_test,validation_steps = len(x_test))# No of images in test set
```

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.

```
### Saving our model

# Save the model
classifier.save('nutrition.h5')
```

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

```
### Predicting our results

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

Taking an image as input and checking the results

```
img = image.load_img(r"C:\Users\DELL\Desktop\Desk Files\Nutrition Analysis Using Image Classification\
                Sample_Images\Test_Image5.jpg",
                grayscale=False,target_size= (64,64))#Loading of the image
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=['APPLES', 'BANANA', 'ORANGE','PINEAPPLE','WATERMELON']
result=str(index[pred[0]])
result

'PINEAPPLE'
```

The predicted class index name will be printed here.

❖ 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.

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, 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.

  o Link :CSS , JS

Build 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
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

```python
@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

```python
@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. Soaking a smart use of it will be an efficient way.

```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']
```

How to access and use the API will be shown in the video
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)
```

Signatures:

S. Muskan.

T. Revanth.

G. Bhavani Sankar Sai.