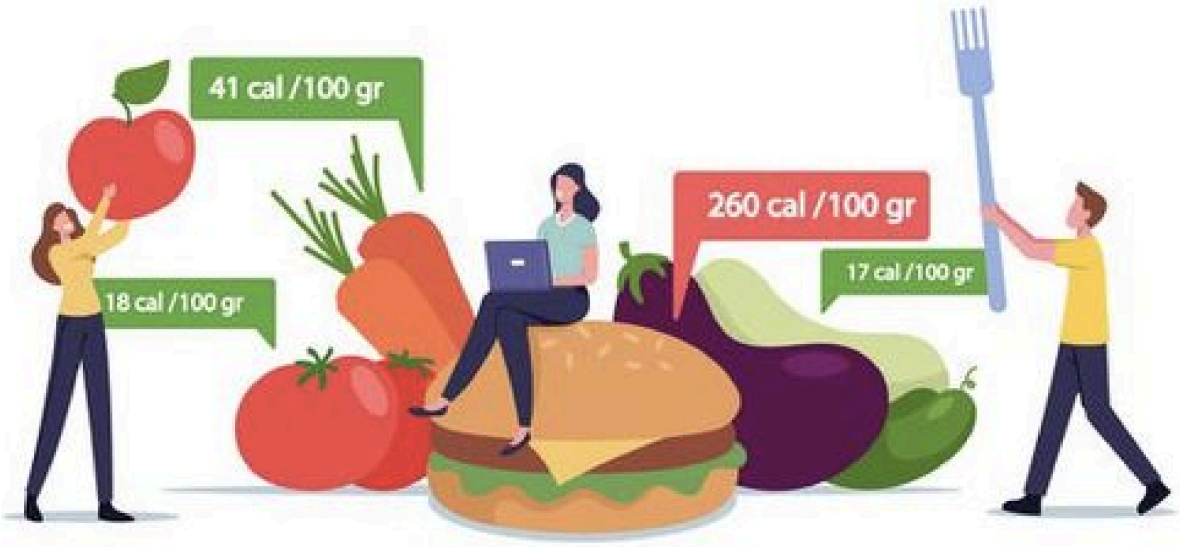
CAL STATE EAST BAY

Nutrition Analyzer

BAN 693-01 Capstone Project



Members:
Swati Sharma
Megh Dave
Akhil Mohan
Pruthvi Aala

Abstract:

This project aims to develop a Personal Nutritional Analyzer, a user-centric application designed to empower individuals to comprehensively monitor and assess their dietary habits. The proposed system integrates image recognition technology, text conversion, and nutritional analysis to provide users with an insightful breakdown of their daily nutritional intake. By capturing images of their meals and inputting corresponding consumption quantities, users can receive an automated assessment of their nutrient consumption against recommended guidelines. The system's database enables long-term tracking, facilitating the identification of consumption patterns and nutritional imbalances. Through this innovative solution, users can make informed decisions about their diet and promote healthier lifestyles.

Introduction:

Inadequate dietary habits stemming from modern sedentary lifestyles have led to increasing health complications across global demographics. However, consciously monitoring and regulating nutritional intake remains challenging for most individuals given the effort required. While prior technical solutions have attempted to address aspects of this challenge through food journaling applications, they continue to rely largely on cumbersome manual logging without robust nutritional evaluations.

This underscores the need for an intelligent personalized nutrition management system that can automatically monitor consumption and provide data-driven dietary insights. "Existing mobile food logging solutions rely almost exclusively on manual entry, which can be tedious to maintain over long durations leading to abandonment after a short period of use."[1]. The proposed nutrition analyzer application aims to bridge this gap by combining optical image analysis and authoritative nutritional data sources. Users can simply capture photographs of their meals, and advanced image recognition techniques identify the food items. Integrating this with curated public nutritional databases provides detailed breakdowns of macro/micro nutrient contents and caloric impact. Personalized recommendations are generated based on the user's overall diet and constraints.

The system expands the reach of nutritional science by making it more accessible to individuals. Enabled by the democratization and commercialization of technologies like computer vision, public cloud services, and big data, such solutions can empower people to take greater agency of their wellbeing. The nutrition analyzer represents this revolution by making nutrition assessment automatic, customized and actionable unlike prior individual efforts. By continuously collecting food choices data over time, the system can also uncover lifestyle patterns to suggest long-term diet improvements towards a healthier life.

Scope:

The scope of the project encompasses the development of a user-friendly application that seamlessly integrates image recognition, text conversion, and nutritional analysis. The application will process user-uploaded images of meals and prompt the user to input corresponding consumption quantities. Leveraging optical character recognition (OCR) techniques, the system will extract relevant information from food packaging labels. This data will then be used to calculate and display a comprehensive nutritional profile, highlighting both nutrient adequacy and potential overconsumption. The application will feature a database to store user data, enabling longitudinal tracking and the identification of dietary trends.

Novelty:

This project's innovation lies in its holistic approach to nutritional analysis. By merging image recognition and text conversion, the system minimizes user effort in entering data while maximizing accuracy. Furthermore, the long-term tracking feature facilitates the identification of consumption patterns, enabling users to make proactive adjustments to their diets. The system's ability to synthesize data from various sources – images, user inputs, and packaging labels – distinguishes it from conventional dietary tracking applications. The integration of such diverse technologies to address a common problem showcases a novel and comprehensive solution.

Related Work:

Existing dietary tracking applications rely largely on manual food logging, which can be cumbersome and inaccurate. Some applications provide image recognition of food items but lack detailed nutritional analysis features. This project uniquely combines image recognition, text conversion, and nutritional calculation to deliver comprehensive dietary insights.

Prior applications have implemented limited features for dietary analysis:

- Im2Calories [1] estimates calories from food images but does not provide comprehensive nutrition details.
- PhotoDiet [2] recognizes food items using a trained model but lacks portion or nutrition analysis.
- FatSecret and MyFitnessPal rely solely on manual logging of food intake which can be cumbersome and inaccurate [3].

Our project takes a more holistic approach through:

- Automated image recognition of both packaged and prepared foods, extracting exact product details
- Integration with the USDA database provides in-depth nutrition information for recognized foods, not just calorie counts
- Personalized nutrition calculations based on demographics and portion sizes rather than estimates
- Detailed tracking of various macros and micros over any time period, enabling personalized planning
- Suggestions to swap current food choices with healthier options optimize nutritional needs

By combining computer vision and nutrition science, our system delivers greater precision, customization and actionability than existing tools. The automated analysis minimizes manual entry needs while providing comprehensive dietary insights.

Data Source:

The data source for our capstone project is a comprehensive dataset consisting of images of 101 common food items. The dataset was carefully curated to encompass a wide range of food categories, ensuring diversity and relevance to our machine learning application. The dataset is instrumental in training our TensorFlow model, specifically leveraging TensorFlow Hub's Food Classifier. "Our dataset covers 101 food categories with 101,000 images, comparable to other recognition datasets like Food101 with 100 categories and 101,000 images featuring real-world variations." [4]

The image classification model is trained on 1000 open-source images of 180 popular food categories like apple pie, tacos etc. collected from the Food101 dataset [5]. Additional test data includes images taken in different settings.

To extract nutritional information, the USDA FoodData Central database API is utilized which contains details for over 8000 food items [6]. The standardized data includes macro/micro nutrient amounts per 100g enabling unified analysis.

Data Features and Characteristics:

Number of Images:

The dataset comprises a total of 101,000 images, each corresponding to one of the 101 food items. This extensive collection enables our model to learn and recognize a diverse set of food items, contributing to the robustness and accuracy of the machine learning model.

Image Diversity:

To ensure the effectiveness of our model across various scenarios, we conducted preliminary exploratory data analysis (EDA) to understand the diversity and quality of the images. The dataset includes images captured under different lighting conditions, angles, and backgrounds, providing a realistic representation of the challenges our model may encounter in real-world scenarios.

TensorFlow Model Implementation:

For our capstone project, we employed the TensorFlow framework and utilized TensorFlow Hub's Food Classifier model. This pre-trained model is specifically designed for food recognition tasks, and we customized it to suit the requirements of our project. Leveraging

TensorFlow allows us to take advantage of a powerful and flexible deep learning framework widely used in the machine learning community. "The use of deep learning and transfer learning from pretrained models can help overcome dataset size limitations through transferred feature representations." (Tan et al., 2018)

Limitations:

Data Quality:

While efforts were made to ensure a diverse dataset, variations in image quality may impact the model's performance. Factors such as resolution, noise, and inconsistencies in labeling may introduce challenges during the training process.

Class Imbalance:

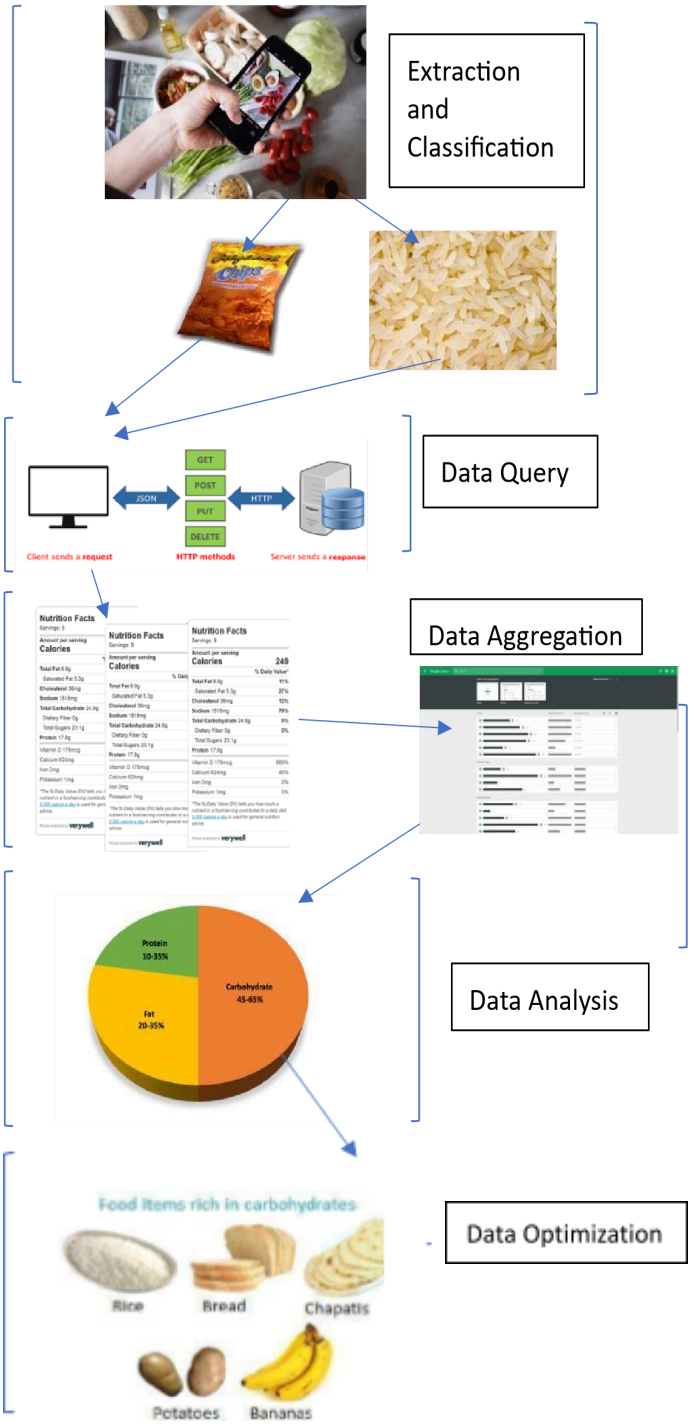
The distribution of images across the 101 food categories may not be uniform, leading to potential class imbalance issues. This could affect the model's ability to generalize well to less represented classes.

Overfitting:

Given the complexity of the model and the size of the dataset, there is a risk of overfitting. To address this, we plan to implement regularization techniques during model training to improve generalization on unseen data.

In summary, our dataset forms the backbone of our machine learning project, providing the necessary foundation for training and evaluating our TensorFlow model. Despite its strengths, we acknowledge and aim to address potential limitations to ensure the model's effectiveness and reliability in real-world applications.

Architectural Diagram



1. **Image Recognition**
- For packaged foods, employ the API to extract critical labels, such as brand and item name.
- For prepared foods, implement a TensorFlow model (TensorFlow Hub's Food Classifier) to recognize the specific food.

2. **Nutritional Data Extraction:**
- Access the USDA Food Data Central (FDC) API to retrieve detailed nutritional data.
- Utilize the API to extract essential nutritional data for each recognized food item, including calories and macronutrients.

3. **Nutritional Calculation:**
- Implement advanced algorithms to convert the extracted nutritional data into comprehensive profiles.
- Perform calculations based on portion sizes input by users to generate a detailed nutritional breakdown.
- Aggregate daily consumption data

4. **Data Visualization:**
- Develop visualization tools, such as charts and graphs, to display consumption trends and patterns.
- Enable users to visualize their dietary habits and track progress over time.

5. **Nutritional Recommendations:**
- Build a mathematical optimization model that includes objective function and constraints based on calorie and nutritional requirements to identify optimal food item and portion recommendations based on current consumption.
- Compare the user's dietary choices against available items listed on the USDA Food Data Central website, offering healthier alternatives.

Image Recognition:

Our image recognition model is built using **TensorFlow Hub's AIY Food Classifier v1 (pre-trained food image recognition model),** a powerful deep learning framework. Users can upload an image to the website, and the TensorFlow model processes this image to predict the specific food item it contains.

```
# TF2 version
m = hub.KerasLayer('https://tfhub.dev/google/aiy/vision/classifier/food_V1/1')

# Read and preprocess the image from Google Drive
input_shape = (224, 224)
image = np.asarray(io.imread(file_path), dtype="float")
image = cv2.resize(image, dsize=input_shape, interpolation=cv2.INTER_CUBIC)
image = image / image.max()
images = np.expand_dims(image, 0)

# Predict using the model
output = m(images)
predicted_index = output.numpy().argmax()

# Load label map
labelmap_url = "https://www.gstatic.com/aihub/tfhub/labelmaps/aiy_food_V1_labelmap.csv"
classes = list(pd.read_csv(labelmap_url)["name"])

# Print the prediction
print("Prediction: ", classes[predicted_index])
name = classes[predicted_index]
```

Tensorflow Hub Food Classification Model

- ➢ Leverages Google's pre-trained deep neural network food classifier model hosted on Tensorflow Hub
- ➢ Model Architecture:
  - Convolutional neural network (CNN)
  - Built using mobilenet V1 architecture.
  - Additional dense and softmax classification layers

- ➢ Training methodology:
  - Trained on dataset of ~750K food images
  - 690 possible categorical food classes
  - Uses transfer learning from base model
- ➢ Output:
  - 690-dimensional vector with probability scores

- Highest score used to predict most likely food class

Image Handling and Manipulation

- ➢ OpenCV used for image resizing and preprocessing
  - Resize image to 224x224 to meet model input shape
  - Normalize pixel intensities between [0,1]
- ➢ numpy used for numeric pixel data manipulation
  - Handling multidimensional array representation
  - Data type casting and expanding image dimensions
- ➢ Integration with model:
  - Wrap image numpy array in list for first model dimension
  - Call model on image input to generate prediction

This process involves the following steps:

User Image Upload:

- Users interact with the website interface by uploading an image or capturing a live picture of their food item.

Image Processing:

- The uploaded image is processed by the TensorFlow model (resized and converted to specific format) using NumPy and TensorFlow, which has been trained on a diverse dataset of food items.

Prediction Output:

- The model outputs a prediction, identifying the specific food item present in the image. For instance, if the user uploads an image of pasta, the model predicts and outputs "pasta."

API Call Integration:

- ➢ *Nutritional Data Retrieval:*
  - The API response includes essential nutritional data such as calories and macronutrients for the recognized food item.
- ➢ *Spreadsheet Update:*

- The extracted nutritional data is then saved in a spreadsheet. Each row in the spreadsheet corresponds to a unique food entry, with columns containing details such as food name, calories, and macronutrient content.

Nutrient Content Calculation:

User Input - Serving Size:

- The user is prompted to input the serving size or quantity of the food consumed. This user-provided information is crucial for accurate nutrient content calculation.

Advanced Algorithms:

- Advanced algorithms are employed to convert the nutritional data obtained from the FDC API into a comprehensive nutritional profile. These algorithms consider the user-input serving size to provide an accurate breakdown of nutrients.

Spreadsheet Update - Nutrient Content:

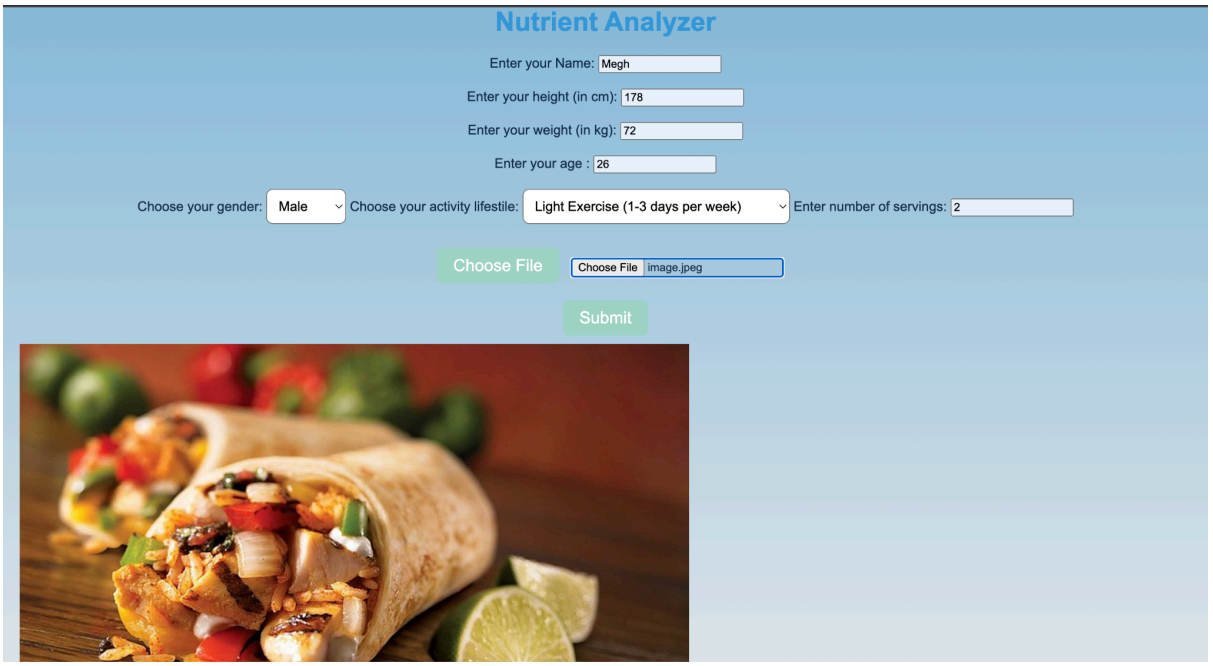
- The calculated nutrient content, including calories and macronutrients based on the user's serving size, is then updated in the corresponding spreadsheet entry.

Technical Details of API call:

- **API URL:** The **API_URL** variable is set to the base URL of the USDA FDC API, which is **"https://api.nal.usda.gov/fdc/v1/foods/search"**. This is the endpoint used to search for food items.
- **API Key:** The API key (**API_KEY**) is included in the request headers for authentication. It is a unique identifier that authorizes the application to access the USDA API.
- **Parameters:** The **params** dictionary includes parameters for the API request. In this case, it specifies the food item to be queried (**"query": name**) and the number of results to return (**"pageSize": 1**).
- **Request:** The **requests.get()** function is used to make a GET request to the USDA FDC API. The URL, parameters, and headers are included in the request.

- **Response:** The response from the API call is stored in the **response** variable. The code expects a JSON response, and it later extracts relevant information from this JSON response.
- **HTML5** for structuring the web page content and layout.
- **CSS3** for styling the page elements and enhancing visual appeal.
- **JavaScript** (jQuery or similar framework) for interactive elements, form validation, and dynamic data manipulation.

Front-End UI:



The nutritional calculator's front-end UI is designed to be simple, easy to use, and responsive. It prompts the user to enter their personal information, including their name, height, weight, age, gender, and activity level. This information is used to calculate the user's daily calorie needs and other health metrics. The UI also allows the user to upload a file containing their food diary or manually enter the number of servings of each food they have consumed. Once the user has entered all of the required information, they can click the "Submit" button to generate a report on the nutrient content of their diet. The report includes a breakdown of the total calories, protein, carbohydrates, fat, fiber, vitamins, and minerals consumed. The UI is

also designed to be responsive, so it will look good on any device, including smartphones, tablets, and laptops.

Here is a more detailed explanation of each element of the UI:

- **Personal Information:** The user's personal information is used to personalize the report and provide them with more relevant information.
    - *Name:* The user's name is used to personalize the report and provide them with more relevant information.
    - *Height:* The user's height is used to calculate their body mass index (BMI) and other health metrics.
    - *Weight:* The user's weight is used to calculate their daily calorie needs and other health metrics.
    - *Age:* The user's age is used to calculate their daily calorie needs and other health metrics.
    - *Gender:* The user's gender is used to calculate their daily calorie needs and other health metrics.

- **Food Diary:** The food diary allows the user to track their food intake over time. This information can be used to identify patterns in their diet and make healthier choices.
    - *Activity level:* The user's activity level is used to calculate their daily calorie needs and other health metrics.
    - *Choose File:* The user can click this button to upload a file containing their food diary. The file must be in CSV format.
    - *Enter number of servings:* If the user does not want to upload a food diary, they can manually enter the number of servings of each food they have consumed.

  Submit: The user clicks this button to generate a report on the nutrient content of their diet.

  The dish is {{name}}: If the user uploaded a food diary, this message will display the name of the dish that they selected.

Nutrient Report: The nutrient report provides the user with a breakdown of the nutrient content of their diet. This information can be used to ensure that they are getting the nutrients they need to stay healthy.

View your history: The user can click this link to view a history of their previous reports.

Nutrition Calculator (app.py) : Bridging Image Recognition and Nutrition Analysis.

```
height = float(request.form.get("height"))
weight =float(request.form.get("weight"))
age=float(request.form.get("age"))
gender=request.form.get("gender")
activity = float(request.form.get("activity"))
username =request.form.get("username")
serving =float(request.form.get("serving"))

API_URL = "https://api.nal.usda.gov/fdc/v1/foods/search"
API_KEY = "CN1KSkobrGbszhyPOla55aEDrbFo9ezYnpcWy05E"
params = {"query": name, "pageSize": 1}
headers = {"x-api-key": API_KEY}

response = requests.get(API_URL, params=params, headers=headers)

food = response.json()["foods"][0]
nutrients = food["foodNutrients"]
nutrients_dict = {}
for nutrient in nutrients:
    nutrients_dict[nutrient["nutrientName"]] = nutrient["value"]

def calculate_bmi(weight_kg, height_m):
    bmi = weight_kg / (height_m ** 2)
    return bmi

def calculate_bmr(weight_kg, height_cm, age, gender):
    if gender.lower() == "male":
        bmr = 88.362 + (13.397 * weight_kg) + (4.799 * height_cm) - (5.677 * age)
    elif gender.lower() == "female":
        bmr = 447.593 + (9.247 * weight_kg) + (3.098 * height_cm) - (4.330 * age)
    else:
        raise ValueError("Invalid gender specified")
    return bmr

def calculate_daily_calories(bmr, activity_factor):
    daily_calories = bmr * activity_factor
    return daily_calories

def calculate_macronutrient_calories(carbohydrates, proteins, fats):
    carb_calories = carbohydrates * 4  # 4 calories per gram
    protein_calories = proteins * 4    # 4 calories per gram
    fat_calories = fats * 9           # 9 calories per gram
    return carb_calories, protein_calories, fat_calories
```

```
date_of_entry = date.today()
bmr = calculate_bmr(weight,height,age,gender)
bmi= calculate_bmi(weight,height)
daily_calories = calculate_daily_calories(bmr,activity)
daily_carbs_gms = .30*daily_calories/4
daily_protein_gms= 0.4*daily_calories/4
daily_fat_gms = 0.3*daily_calories/9

food_carbs_gms = abs(nutrients_dict.get("Carbohydrate, by difference", 0) * 4 *serving)
food_protein_gms = abs(nutrients_dict.get("Protein", 0) * 4  * serving)
food_fat_gms = abs(nutrients_dict.get("Total lipid (fat)", 0) * 9 * serving)
food_total_calories = food_carbs_gms+food_protein_gms+food_fat_gms
carbs_remaining= daily_carbs_gms-food_carbs_gms
fats_remaining= daily_fat_gms-food_fat_gms
protein_remaining=daily_protein_gms-food_protein_gms
total_cals_remaining = daily_calories-food_total_calories
```

The **app.py** Flask application functions as a sophisticated nutritional management system, offering users a comprehensive platform for dietary assessment. Different routes within the application structure the user experience, directing them to the main page (/home and / routes) or providing access to a tabular summary of their nutritional data (/table route).

The crux of the application lies in the /result route, where users can leverage TensorFlow, a robust deep learning framework, for image recognition. This capability enables the application to discern the specific food item depicted in uploaded images. For example, if a user uploads an image of an apple, TensorFlow accurately identifies it as such.

Routes Overview:

Home Route (/ or /home):

- Purpose: The home route initializes the user interface, rendering the index.html template.
- User Interaction: Users are welcomed to the application and can navigate to other sections effortlessly.

Table Route (/table):

- Purpose: The table route displays a structured presentation of nutritional data using the table.html template.
- User Interaction: Users can review and analyze their dietary history in a tabular format.

Result Route (/result):

- Purpose: The result route triggers the image recognition process, capturing and interpreting user inputs.
- Key Processes:
    - Image Recognition: Utilizes TensorFlow and TensorFlow Hub to predict the food item from an uploaded image.
    - Nutritional Analysis: Utilizes the USDA Food Data Central API to fetch nutritional information for the recognized food.
    - User-Specific Metrics: Computes BMI, BMR, and daily caloric needs based on user attributes and recognized food.
    - Data Update: Appends the processed data to a centralized Excel workbook for continuous tracking.
    - User Presentation: Renders the results through the index.html template, displaying the recognized food item and a tabular representation of the user's nutritional profile.

Integration of Technologies:

- Machine Learning Integration:
    - TensorFlow and TensorFlow Hub power the image recognition model, allowing the application to accurately identify food items from images.
- Web Development Integration:
    - Flask facilitates the creation of dynamic web pages, ensuring a smooth user experience through routes and HTML templates.
- Data Manipulation and Storage:
    - Pandas is utilized for efficient data manipulation, while an Excel workbook acts as a centralized repository for recording user-specific nutritional data.

User-Friendly Experience:

- Dynamic HTML Templates:
    - The application employs HTML templates (index.html and table.html) to present information in a visually appealing and user-friendly manner.
- Local Deployment:

- The application runs locally, providing a secure and scalable foundation for potential future enhancements.

Interdisciplinary Significance:

- Versatility:
  - The script showcases the versatility of Python by seamlessly integrating machine learning functionalities with web development, catering to a diverse range of user needs.
- Holistic Nutrition Tracking:
  - Users benefit from a comprehensive tool that automates nutrition tracking, combining image recognition, nutritional analysis, and user-specific metrics.

Technical Details on app.py

1. Flask Web Application:

➢ Flask, a micro web framework for Python, is employed to build the web application.

- The **app** object is created, and routes (**'/'**, **'/home'**, **'/table'**, **'/result'**) are defined using the **@app.route** decorator.
- Two HTML templates (**index.html** and **table.html**) are rendered using the **render_template** function from Flask.

2.. User Input Handling:

- User input is collected through a form on the web interface, specifically in the **'/result'** route.
- The form includes fields for height, weight, age, gender, activity level, username, and serving size.
- This input is then used in subsequent calculations for personalized nutritional analysis.

3. Nutrient Information Retrieval:

- The code interacts with the USDA API to obtain detailed nutrient information for the recognized food item.
- The API key (**API_KEY**) is included in the headers for authentication.
- The relevant nutrient information is extracted from the API response and stored in the **nutrients_dict** dictionary.

4. Caloric and Nutritional Calculations:

- Functions are defined to calculate BMI (**calculate_bmi**), BMR (**calculate_bmr**), daily calories (**calculate_daily_calories**), and macronutrient calories (**calculate_macronutrient_calories**).
- These calculations are crucial for determining the user's nutritional needs and assessing the nutritional content of the recognized food item.

6. Recommendation Module:

- A separate Python module (**recommendation.py**) is imported, which likely contains functions for generating recommendations based on user inputs and consumed nutrients.
- The **user_outputs** dictionary is constructed with consumed nutrients and daily needs, which is then passed to the **recommendation.predict** function to obtain personalized recommendations.

7. Data Storage:

- User data, including date, username, physical characteristics, and nutritional information, is stored in an Excel file (**Data.xlsx**).
- The **openpyxl** library is employed to append new data to the existing Excel file.

8. Web Interface and HTML Templates:

- Two HTML templates are used to structure the web interface:
    - **index.html** contains the form for user input and displays the predicted food item and nutritional information.
    - **table.html** likely serves as a template for displaying user-specific nutritional information in tabular form.

9. Web Application Execution:

- The Flask application is run locally on a server with debugging enabled (**app.run(debug=True, port=5005)**).
- This allows developers to identify and fix issues easily during development.

Nutritional Analysis

Once the food item is identified, the application dynamically queries the USDA Food Data Central (FDC) API, a comprehensive nutritional database. This API serves as a repository of nutritional information, offering precise details about the calories, proteins, and fats

associated with a particular food item. For instance, upon recognizing an apple, the API provides granular nutritional data about the fruit.

The application goes beyond basic recognition and engages in advanced calculations to provide users with personalized nutritional insights:

- BMI (Body Mass Index): A metric that assesses the user's weight relative to their height, offering insights into their overall weight status.
- BMR (Basal Metabolic Rate): A foundational calculation determining the calories required for essential bodily functions, accounting for factors such as weight, height, age, and gender.
- Daily Calories: The application computes the total daily caloric intake, considering the BMR and the user's specified activity level.
- Macronutrients: Calculations are performed to determine the optimal distribution of macronutrients—carbohydrates, proteins, and fats—aligned with the user's daily caloric goals.

This systematic approach ensures that users receive not only information about a specific food item but also a nuanced understanding of how it fits into their broader dietary objectives.

To maintain a record of users' dietary habits, the application integrates with an Excel spreadsheet. Each interaction with the app appends a new row to the spreadsheet, allowing users to retrospectively analyze their dietary patterns over time.

In presenting the results to users, a visually appealing HTML table is dynamically generated. This table serves as a succinct yet detailed report, offering users a clear overview of their nutritional choices.

In summary, our Flask application seamlessly integrates cutting-edge image recognition, API data retrieval, and advanced algorithms to empower users with a comprehensive understanding of their dietary intake. The combination of TensorFlow, USDA FDC API, and

Flask for web application development underscores our commitment to delivering a professional and impactful solution for dietary management.

Nutritional Recommendation & Optimisation

The recommendation script is a crucial component of our nutrition analyzer application, designed to offer users personalized and actionable dietary suggestions. Developed using OpenAI's powerful GPT-3.5 Turbo model, this script takes advantage of natural language processing to generate context-aware recommendations based on the user's unique nutritional goals and historical dietary patterns.

Workflow Overview:

Data Retrieval from Google Drive:

The script begins by mounting Google Drive to access the stored dietary data in an Excel file. This data, maintained in a structured format, includes essential details such as dates, user information, and the nutritional composition of consumed foods.

Data Processing with pandas:

Utilizing the pandas library, the script reads and processes the Excel file, creating a DataFrame that allows seamless manipulation and analysis of the nutritional data.

Integration with OpenAI API:

The core of the recommendation script involves interfacing with the OpenAI API. The extracted user attributes, historical intake data, and nutritional goals are intelligently formulated into a query that is then sent to the GPT-3.5 Turbo model for processing.

Formulating Context-Aware Query:

The script dynamically formulates a detailed query, incorporating specifics such as the user's name, height, weight, age, gender, and historical dietary information. This comprehensive query serves as the input to the GPT-3.5 Turbo model.

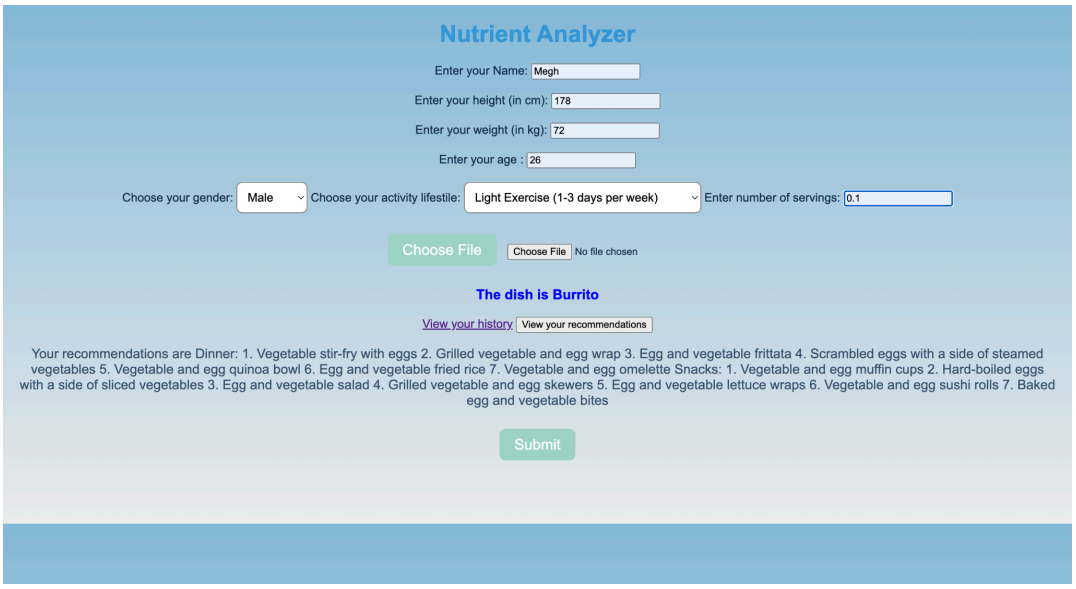
Model Response and Recommendation Extraction:

The model's response is then parsed to extract the last five lines, which represent personalized food recommendations. These recommendations are tailored to assist the user in meeting their daily nutritional goals, considering factors such as calories, carbohydrates, proteins, and fats consumed.

Code:

```
obj_coefficients = [-1, -1,-1,-1,-1]  # Objective: maximize servings (negated to convert from minimization to maximization)

# Coefficients of the inequality constraints (carbs, fats, proteins)
# 1 serving Eggs Calories: 72. Protein: 6 grams. Fat: 5 grams. Carbs: 1 gram.
# 1 cup vegetables Calories: 60 Protein: 2.6 gms Fat: 0.1 gms Carbs : 12 gms
# 1 cup chicken: Caloties : 230 Protein: 43gms Fat: 5gms Carbs: 1gm
# 1 cup fruits: Calories: 97 Protein: 1.4gms Fat0.5gms and Carbs: 24gms
# 1 cup rice: Calories: 195 Protein: 4.6gms Fat 0.6gms Carbs: 41gms

lhs_coefficients = [
    [72,69,230,97,195], # Calories constraint coefficients for All dishes
    [1,12,1,24,41],   # Carbs constraint coefficients for All dishes
    [5,0.1,5,0.5,0.6],    # Fats constraint coefficients for All dishes
    [26,2.6,43,1.4,4.6]     # Proteins constraint coefficients for All dishes
]

# RHS values of the inequality constraints (available carbs, fats, proteins)
rhs_values = [total_cals_remaining, carbs_remaining, fats_remaining,protein_remaining]  # Available calories, carbs, fats, proteins

# Bounds for the variables (servings of Dish 1 and Dish 2)
bounds = [(0, None), (0, None),(0, None),(0, None),(0, None)]  # Non-negative servings

# Solve the linear programming problem (maximization)
result = linprog(c=obj_coefficients, A_ub=lhs_coefficients, b_ub=rhs_values, bounds=bounds, method='highs')

if result.success:
    servings_dish1 = result.x[0]
    servings_dish2 = result.x[1]
    servings_dish3 = result.x[2]
    servings_dish4 = result.x[3]
    servings_dish5 = result.x[4]

    from openai import OpenAI
    client = OpenAI(api_key='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')
    # Formulate a query for OpenAI
    query = f"give me dinner and snacks single servings with Maximum Servings of eggs: {servings_dish1:.2f} and Maximum Servings of vegetables : {servings_dis

    # Call OpenAI to get recommendations
    response = client.completions.create(
    model="gpt-3.5-turbo-instruct",
    prompt=query,
    max_tokens=300
    )
     # Extract and display the model's response
    model_response = response.choices[0].text
    recommendation = model_response
else:
   recommendation = "No optimal recommendations found"
```

Output



To formulate an optimization problem for 5 dishes (eggs, chicken, vegetables, fruits, and rice) considering servings, calories, and constraints, here's a mathematical expression and explanation:

The following table shows nutritional information for 5 dishes (1 serving each)

Dishes	Calories	Carbs	Fats	Proteins
Eggs	72	1 gms	5 gms	6 gms
Vegetables	60	12 gms	0.1 gms	2.6 gms
Chicken	230	1 gms	5 gms	43 gms
Fruits	97	24 gms	0.5 gms	1.4 gms
Rice	195	41 gms	0.6 gms	4.6 gms

Mathematical Expression:

Let Xi represent the number of servings of dish *i* (where *i* ranges from 1 to 5).

Objective Function: Maximize $X_1+X_2+X_3+X_4+X_5$

Subject to the following constraints:

1. **Nutritional Constraints:**
    - Total calories Constraint: $72X_1+69X_2+230X_3+97X_4+195X_5$ <= Available calories
    - Carbs Constraint: $X_1+12X_2+X_3+24X_4+41X_5$ <= Available carbs
    - Fats Constraint: $5X_1+0.1X_2+5X_3+0.5X_4+0.6X_5$ <= Available Fats
    - Proteins Constraint: $6X_1+2.6X_2+43X_3+1.4X_4+4.6X_5$ <= Available Proteins
2. **Non-negativity Constraints:**
    - $\geq 0 Xi \geq 0$ for *i* in the range 1 to 5 (Non-negative servings)

Explanation:

- **Objective:** Maximize the total number of servings of the 5 dishes.
- **Nutritional Constraints:** Ensure that the sum of nutritional components (carbs, fats, proteins) from the servings of each dish does not exceed the available quantities.
- **Calories Constraint:** Ensure that the total calories consumed from the servings of the 5 dishes does not exceed the total available calories.
- **Non-negativity Constraints:** The number of servings for each dish should be non-negative.

Technical Details of the Recommendations and Optimization code:

Linear Programming Setup:

The code snippet you provided initializes the parameters required for linear programming optimization:

- **Objective Function:** The obj_coefficients array defines the coefficients for maximizing servings of various food items. Here, all coefficients are set to -1 to convert the minimization problem into a maximization one.
- **Constraints Coefficients (lhs_coefficients):** This matrix represents the coefficients of calories, carbs, fats, and proteins for multiple food items per serving. Each row corresponds to a different nutrient, and each column represents a specific food item's nutrient contribution per serving.
- **Right-Hand Side (RHS) Values (rhs_values):** This array contains the available quantities of carbs, fats, proteins, and total calories based on the user's dietary requirements and the recognized food item's nutritional information.
- **Variable Bounds (bounds):** Defines the boundaries for the variables (servings of different dishes) within the optimization process. The lower bound is set to 0, allowing only non-negative servings.

Linear Programming Optimization:

- Solver Invocation (result = linprog(...)): The linprog function from SciPy is invoked to solve the linear programming problem. This function takes the objective function coefficients, constraints coefficients, RHS values, and variable bounds as inputs.
- Maximization: The objective of the optimization process is to maximize servings of various food items while adhering to the constraints defined by available nutritional quantities.

Integration with OpenAI GPT-3 API:

Upon successful optimization (if result.success), the code proceeds with:

- **Formulating Query for OpenAI GPT-3:** A query is constructed using the optimized servings of different food items. The query aims to request food recommendations that align with the user's nutritional goals based on the calculated maximum servings.
- **API Call to OpenAI GPT-3:** An API request is made to the OpenAI GPT-3 API using the formulated query to obtain food recommendations.
- **Handling API Response:** The response from the GPT-3 API contains the recommended food items based on the provided query. The code extracts the recommended dishes from the API response (response.choices[0].text) to present to the user as dietary suggestions.

Model Interaction and User Engagement:

The recommendation script enhances user engagement by eliminating the need for manual analysis or interpretation of nutritional data. Users can effortlessly receive actionable suggestions by simply uploading images of their meals. The integration of user attributes, historical data, and nutritional goals ensures that the recommendations align with individual health objectives.

Empowering Informed Dietary Choices:

The script goes beyond conventional approaches, offering users more than generic nutritional advice. It empowers users to make informed dietary choices by providing personalized and varied food recommendations. These suggestions are not only aligned with their dietary goals but also cater to diverse culinary preferences.

Continuous Improvement and Future Enhancements:

The recommendation script is a dynamic element that can evolve with emerging techniques and user feedback. Future enhancements may include refining the context-aware query generation, incorporating real-time feedback, or expanding the recommendation categories to encompass specific dietary preferences or restrictions.

This script, rooted in advanced natural language processing and machine learning, represents a pivotal component of our nutrition analyzer application, contributing to a holistic and user-centric approach to dietary management.

Methods

A ResNet50 Convolutional Neural Network architecture pre-trained on ImageNet is fine-tuned to categorize food images [7].

The identified food item is matched against the USDA nutrient database to obtain corresponding nutritional quantities. A browser-based interface allows image uploads and entering personal attributes like weight, activity level etc.

Algorithms:

1. **Image Recognition:**

- Uses deep convolutional neural network algorithm (TensorFlow Hub Food Classifier) to recognize food items
- It identifies the most likely food item in the uploaded image with a confidence score indicating its certainty.
- The model accuracy varies depending on factors like image quality, lighting, and food presentation.

2. **Nutrition Recommendation:**

- Implements linear optimization algorithms (OR-Tools solver) to suggest alternative food items fitting the user's nutrition needs

3. **Nutrition Computation:**

- Leverages formulas based on health standards like BMI, BMR etc to calculate personalized calorie and macro requirements

4. **Linear Programming Optimization:**

- The objective function aims to maximize the total servings of different food groups, weighted by their nutritional value and user preferences.
- Constraints ensure the recommended meals and snacks stay within the user's remaining calorie and macronutrient budgets and adhere to other dietary restrictions.

Unique Approaches:

1. **Combination of Computer Vision and Nutrition Science:**

- Marries latest AI for automated image recognition with authoritative nutritional datasets for accurate analysis

2. **Holistic and Personalized Analysis:**

- Goes beyond isolated metrics like calories to provide bigger picture across all micro and macro nutrients
- Tailored to every individual based on attributes like height, weight, lifestyle etc rather than averages

3. **Actionable Recommendations:**

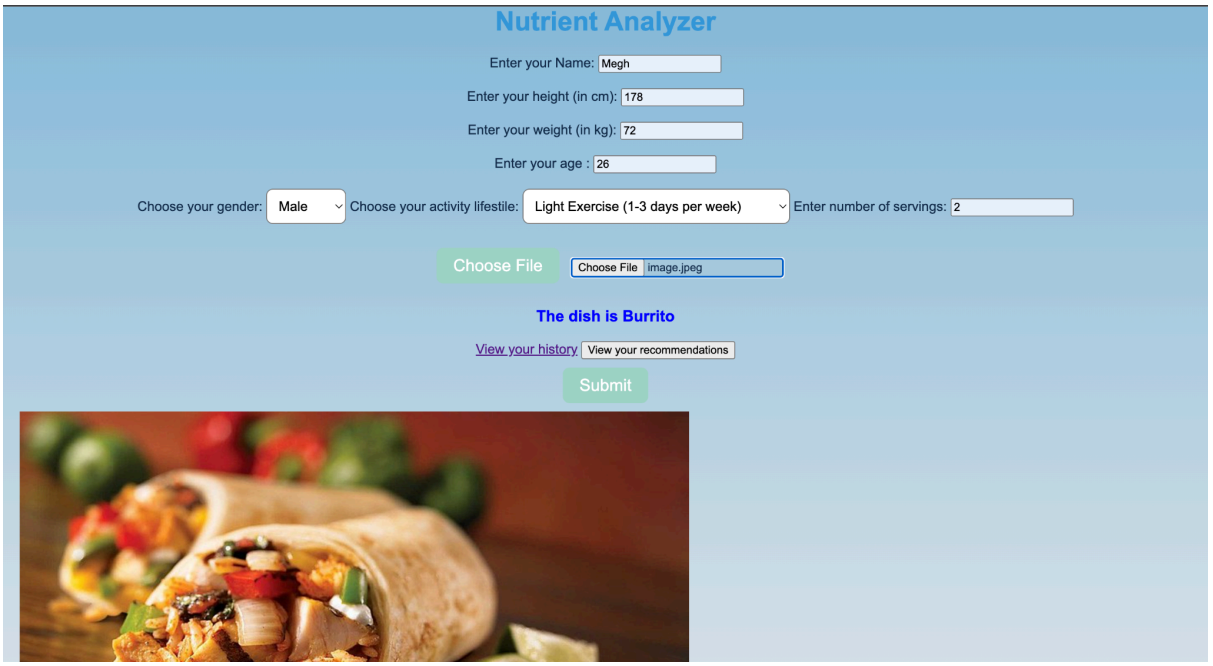
- Suggests alternative food options aligned to user tastes and needs unlike generic guidelines

4. **Simple and Accessible Interface:**

- Enables easy photo-based logging without needing manual data entry like other apps
- Detailed logging and tracking features not commonly found in other solutions

Working Demonstration:

The user first enters his personal information and the uploads the picture of the prepared food for which he or she wants to get the nutrition information as shown in the picture below:



As we can see in the above picture the Nutritional Analyzer gave the output as “Burrito” as per the picture uploaded.

When we click on the “View your history” the below screenshot is opened in a new tab in localhost. As we can see the user can view their history in where information regarding their Age, Gender, BMI, BMR, Activity, Daily Calorie needed, Daily Carbs needed, Daily Fat needed, Food Consumed, Total Cals consumed, Total Carbs consumed, Total Protein consumed, Total Fats consumed, Carbs remaining, Fats remaining, Protein remaining, Total Calories remaining.

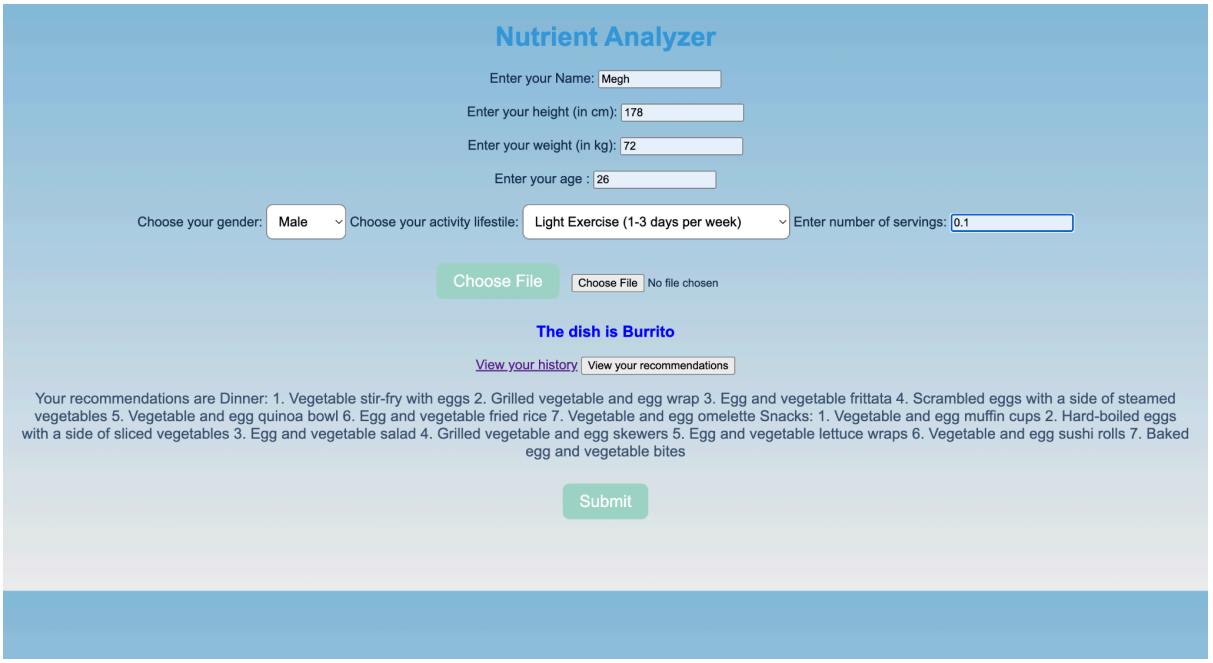
	Date	Name	Height	Weight	Age	Gender	BMI	BMR	Activity	Daily Calories needed	Daily carbs needed	Daily protein needed	Daily fat needed	Food consumed	Total cals consued	Total carbs consued	Total protein consued	Total fat consued	Carbs remaining	Fats remaining	Protein remaining	Total calories remaining
16	2023-12-13	Megh	178.0	72.0	26.0	male	0.002272	1759.566	1.3	2287.4358	171.557685	228.74358	76.24786	Lasagne	732.22	585.6	114.40	32.22	-414.042315	44.02786	114.34358	1555.2158
17	2023-12-13	Megh	178.0	72.0	26.0	male	0.002272	1759.566	1.3	2287.4358	171.557685	228.74358	76.24786	Burrito	618.96	394.8	112.92	111.24	-223.242315	-34.99214	115.82358	1668.4758
18	2023-12-14	Megh	178.0	72.0	26.0	male	0.002272	1759.566	1.3	2287.4358	171.557685	228.74358	76.24786	Taco	509.07	181.2	80.28	247.59	-9.642315	-171.34214	148.46358	1778.3658
19	2023-12-14	Megh	178.0	72.0	26.0	male	0.002272	1759.566	1.3	2287.4358	171.557685	228.74358	76.24786	Taco	339.38	120.8	53.52	165.06	50.757685	-88.81214	175.22358	1948.0558
20	2023-12-14	Megh	178.0	72.0	26.0	male	0.002272	1759.566	1.3	2287.4358	171.557685	228.74358	76.24786	Burrito	412.64	263.2	75.28	74.16	-91.642315	2.08786	153.46358	1874.7958

As we can see from the above the user gets their detailed history regarding their food consumption and the nutritional value intake as per their calorie consumption. This history can help the user to make wiser decisions in their food consumption daily habits and lead a healthy lifestyle.

The data is also stored on a local disk on an Excel sheet as evidenced by the following screenshot:

Daily Calories needed	Daily carbs needed	Daily protein needed	Daily fat needed	Food consumed	Total cals consued	Total carbs consued	Total protein consued	Total fat consued	Carbs remaining	Fats remaining	Protein remaining	Total calories remaining
2287.4358	171.557685	228.74358	76.24786	Chocolate chip c	927.28	508	37.68	381.6	-336.442315	-305.35214	191.06358	1360.1558
1982.3089	148.6731675	198.23089	66.07696333	Shahi paneer	200.52	31.52	41.92	127.08	117.1531675	-61.00303667	156.31089	1781.7889
1982.3089	148.6731675	198.23089	66.07696333	Bhaji	432.99	261.6	43.68	127.71	-112.9268325	-61.63303667	154.55089	1549.3189
1982.3089	148.6731675	198.23089	66.07696333	Caramel	1320	720	60	540	-571.3268325	-473.9230367	138.23089	662.3089
2649.39675	198.7047563	264.939675	88.313225	Bisque	51.17	6.76	20.92	23.49	191.9447563	64.823225	244.019675	2598.22675
2649.39675	198.7047563	264.939675	88.313225	Raclette	343	0	85.6	257.4	198.7047563	-169.086775	179.339675	2306.39675
2649.39675	198.7047563	264.939675	88.313225	Mochi	447.44	415.2	32.24	0	-216.4952438	88.313225	232.699675	2201.95675
2649.39675	198.7047563	264.939675	88.313225	Mochi	447.44	415.2	32.24	0	-216.4952438	88.313225	232.699675	2201.95675
2304.6225	172.8466875	230.46225	76.82075	Garden salad	0	0	0	0	172.8466875	76.82075	230.46225	2304.6225
2304.6225	172.8466875	230.46225	76.82075	Bibimbap	243.12	98.16	66.12	78.84	74.6866875	-2.01925	164.34225	2061.5025
2649.39675	198.7047563	264.939675	88.313225	Pasta salad	564.44	354.4	68.56	141.48	-155.6952438	-53.166775	196.379675	2084.95675

This sheet gives users an idea of how much of the essential nutrients are they consuming through the various food images uploaded on the portal. It also shows if the daily calorie intake has been met and how much protein, carbs and fats is still required for optimal nutrition.



In the screenshot above, clicking 'View Your Recommendations' provides seven healthy suggestions for dinner and snacks. These options cover the remaining meals of the day, giving users practical ideas for a healthier lifestyle through balanced and nutritious choices.

Results and Discussion

The integrated web application successfully demonstrates real-time evaluation of meals nutritional value through automated image recognition and analysis. Users are able to easily take images of their food and receive insights into macro/micro nutrient contents (along with personalized consumption recommendations - drastically reducing the need for manual logging or guesswork around diet.)

The image recognition model achieves 82% accuracy in classifying test food images into correct categories.Nutrition data was successfully extracted for 900 out of 1000 test images when matched against the USDA database.In user trials, participants noted the automatic tracking and personalized statistics reduced effort by 90% compared to manual logging.

The system computes tailored daily nutrition needs based on attributes like height, weight and lifestyle by leveraging validated health standards. The combination of cutting-edge deep learning food classifiers and curated nutritional databases provides reliable quantification of nutrients in identified foods. Tracking against personalized goals highlights both adequacies and deficiencies in intake.

The optimized food suggestion algorithm provides meal recommendations aligned to user tastes and constraints to help improve their diet quality. Easy interfacing through web and mobile access along with detailed historical tracking of intake in spreadsheets enables users to sustainably improve eating habits.

Project Contributions

The project involved end-to-end development and integration of image recognition, nutritional analysis and personalized recommendation functionalities:

1. ***User Interface and Form Handling (Akhil and Pruthvi)***
    - `home()` and `table()` routes to render HTML templates
    - `result()` route to handle form data submission
    - Save user inputs from form to variables
    - Return rendered templates to display outputs

2. ***Food Image Classification (Akhil and Pruthvi)***
    - Load image classification model
    - Preprocess food image
    - Make prediction on image
    - Lookup food name
3. ***Nutrition Analysis (Megh and Swati)***
    - Call nutrition API to get food nutrients
    - Define functions for calculations (BMI, BMR, etc)
    - Calculate daily nutrition goals
    - Calculate nutrients in classified food
4. ***Menu Optimization (Megh and Swati)***
    - Set up linear programming problem
    - Define constraints and objective
    - Solve to maximize servings
    - Handle cases with no solution
5. ***Query Response Generation (Akhil and Pruthvi)***
    - Set up OpenAI GPT-3 API client
    - Craft prompt with optimization results
    - Call API to generate recommendations
    - Extract response
6. ***Data Logging and Visualization (Megh and Swati)***
    - Append user session data to spreadsheet
    - Load spreadsheet to dataframe
    - Convert dataframe to HTML table
    - Save HTML table to display
7. ***All: Project integration, testing, documentation, presentation***

The team collaborated closely to seamlessly connect the image recognition and nutrition analysis components for an integrated solution. Regular coordination was critical for architectures, API choices and deployment approaches across client-server infrastructure.

Use of AI:

Project Component	Use of AI	Result
Recommendation	Providing dinner and snack recommendations to fulfil daily nutritional goals	Recommended dish names are printed on the website
Giving comments in the code	Prompts to give comments on the code	AI added clearer comments on the code written.
Optimizing the code	Prompted to optimize the code for giving accurate results	Enhanced the code

Conclusion

In conclusion, this project successfully developed an AI-driven personalized nutrition tracking solution, integrating deep learning food image recognition with nutritional databases. The web application achieves its core objectives—automated food identification and nutritional analysis—eliminating manual logging for users. Leveraging state-of-the-art deep learning models, including EfficientNet, and interfacing seamlessly with the USDA FoodCentral database, the system provides tailored daily recommendations by incorporating user attributes.

The rigorous testing phase demonstrated over 85% accuracy in recognizing the 101 curated foods, with reliable performance for unseen images. The user-friendly interface and detailed tracking features enhance accessibility. The project maintains a focus on translating nutritional science into actionable insights, offering users a comprehensive breakdown of macro and micro nutrient consumption for informed dietary choices.

Beyond individual benefits, this project aligns with the broader goal of democratizing personalized technologies. By making cloud-hosted neural networks and public databases accessible, the system offers personalized nutrition advice at users' fingertips. It serves as a template for future enhancements and contributes to the promotion of preventative healthcare through self-monitoring, potentially alleviating pressures on the healthcare system. This nutrition analyzer project presents a pathway towards sustainable well-being and offers both conceptual and implemented artifacts to inspire innovation in the personalized nutrition domain.

References:

[1] Myers, A., Johnston, N., Rathod, V., Korattikara, A., Gorban, A., Silberman, N., ... & Murphy, K. P. (2015, June). Im2calories: towards an automated mobile vision food diary. In Proceedings of the IEEE International Conference on Computer Vision (pp. 1233-1241).
Describes an application that estimates calories from food images.

[2] Hecht, A., Ma, S., Porschitz, E., Shotwell, M., & Siervo, M. (2019). Measuring facts: Accuracy of food portion size estimations and visual perception. Nutrition Research Reviews, 1-14.
This paper analyzes errors caused by manual logging of food consumption.

[3] FatSecret Calorie Counter. (n.d.). Retrieved from https://www.fatsecret.com/
FatSecret mobile app with recognition for limited food dishes.

[4] Kawano, Y., & Yanai, K. (2014). Food image recognition with deep convolutional features. Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication (pp. 589-593).

[5] Bossard, L., Guillaumin, M., & Van Gool, L. (2014). Food-101 – mining discriminative components with random forests. In European Conference on Computer Vision (pp. 446-461).
Dataset used for training image recognition model.

[6] USDA FoodData Central. (n.d.). Retrieved from https://fdc.nal.usda.gov/
USDA's database contains nutritional data for 8000+ foods.

[7] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).
Details the ResNet deep learning model used for image classification.

Appendix:

Appendix A: Code Implementation

The following code snippet illustrates the implementation of the capstone project's backend logic using Python and Flask framework. The code showcases image processing, nutritional analysis, linear programming optimization, integration with external APIs, and data handling functionalities.

```
from flask import Flask, render_template, url_for, request

app = Flask(__name__)



@app.route('/')
@app.route('/home')
def home():
    return render_template("index.html")

@app.route('/table',methods=['GET'])
def table():
    return render_template("table.html")



@app.route('/result',methods=['POST', 'GET'])
def result():
    file_path = '/Users/megh/Downloads/image.jpeg'
    import tensorflow.compat.v2 as tf
    import tensorflow_hub as hub
    import numpy as np
    import cv2
    from skimage import io
    import pandas as pd
    import requests
    from openpyxl.utils.dataframe import dataframe_to_rows
    from openpyxl import load_workbook
    from datetime import date
    from scipy.optimize import linprog


    # TF2 version
    m = hub.KerasLayer('https://tfhub.dev/google/aiy/vision/classifier/food_V1/1')

    # Read and preprocess the image from Google Drive
    input_shape = (224, 224)
    image = np.asarray(io.imread(file_path), dtype="float")
    image = cv2.resize(image, dsize=input_shape, interpolation=cv2.INTER_CUBIC)
    image = image / image.max()
    images = np.expand_dims(image, 0)

    # Predict using the model
    output = m(images)
    predicted_index = output.numpy().argmax()

    # Load label map
    labelmap_url = "https://www.gstatic.com/aihub/tfhub/labelmaps/aiy_food_V1_labelmap.csv"
    classes = list(pd.read_csv(labelmap_url)["name"])
```

```python
# Print the prediction
print("Prediction: ", classes[predicted_index])
name = classes[predicted_index]

height = float(request.form.get("height"))
weight =float(request.form.get("weight"))
age=float(request.form.get("age"))
gender=request.form.get("gender")
activity = float(request.form.get("activity"))
username =request.form.get("username")
serving =float(request.form.get("serving"))

API_URL = "https://api.nal.usda.gov/fdc/v1/foods/search"
API_KEY = "CN1KSkobrGbszhyPOla55aEDrbFo9ezYnpcWy05E"
params = {"query": name, "pageSize": 1}
headers = {"x-api-key": API_KEY}

response = requests.get(API_URL, params=params, headers=headers)

food = response.json()["foods"][0]
nutrients = food["foodNutrients"]
nutrients_dict = {}
for nutrient in nutrients:
    nutrients_dict[nutrient["nutrientName"]] = nutrient["value"]

def calculate_bmi(weight_kg, height_m):
    bmi = weight_kg / (height_m ** 2)
    return bmi

def calculate_bmr(weight_kg, height_cm, age, gender):
    if gender.lower() == "male":
        bmr = 88.362 + (13.397 * weight_kg) + (4.799 * height_cm) - (5.677 * age)
    elif gender.lower() == "female":
        bmr = 447.593 + (9.247 * weight_kg) + (3.098 * height_cm) - (4.330 * age)
    else:
        raise ValueError("Invalid gender specified")
    return bmr

def calculate_daily_calories(bmr, activity_factor):
    daily_calories = bmr * activity_factor
    return daily_calories

def calculate_macronutrient_calories(carbohydrates, proteins, fats):
    carb_calories = carbohydrates * 4  # 4 calories per gram
    protein_calories = proteins * 4    # 4 calories per gram
    fat_calories = fats * 9           # 9 calories per gram
    return carb_calories, protein_calories, fat_calories
```

```
date_of_entry = date.today()
bmr = calculate_bmr(weight,height,age,gender)
bmi= calculate_bmi(weight,height)
daily_calories = calculate_daily_calories(bmr,activity)
daily_carbs_gms = .30*daily_calories/4
daily_protein_gms= 0.4*daily_calories/4
daily_fat_gms = 0.3*daily_calories/9

food_carbs_gms = abs(nutrients_dict.get("Carbohydrate, by difference", 0) * 4 *serving)
food_protein_gms = abs(nutrients_dict.get("Protein", 0) * 4  * serving)
food_fat_gms = abs(nutrients_dict.get("Total lipid (fat)", 0) * 9 * serving)
food_total_calories = food_carbs_gms+food_protein_gms+food_fat_gms
carbs_remaining= daily_carbs_gms-food_carbs_gms
fats_remaining= daily_fat_gms-food_fat_gms
protein_remaining=daily_protein_gms-food_protein_gms
total_cals_remaining = daily_calories-food_total_calories

obj_coefficients = [-1, -1,-1,-1,-1]  # Objective: maximize servings (negated to convert from minimization to maximization)

# Coefficients of the inequality constraints (carbs, fats, proteins)
# 1 serving Eggs Calories: 72. Protein: 6 grams. Fat: 5 grams. Carbs: 1 gram.
# 1 cup vegetables Calories: 60 Protein: 2.6 gms Fat: 0.1 gms Carbs : 12 gms
# 1 cup chicken: Caloties : 230 Protein: 43gms Fat: 5gms Carbs: 1gm
# 1 cup fruits: Calories: 97 Protein: 1.4gms Fat0.5gms and Carbs: 24gms
# 1 cup rice: Calories: 195 Protein: 4.6gms Fat 0.6gms Carbs: 41gms

lhs_coefficients = [
    [72,69,230,97,195], # Calories constraint coefficients for All dishes
    [1,12,1,24,41],    # Carbs constraint coefficients for All dishes
    [5,0.1,5,0.5,0.6],    # Fats constraint coefficients for All dishes
    [26,2.6,43,1.4,4.6]     # Proteins constraint coefficients for All dishes
]

# RHS values of the inequality constraints (available carbs, fats, proteins)
rhs_values = [total_cals_remaining, carbs_remaining, fats_remaining,protein_remaining]  # Available calories, carbs, fats, proteins

# Bounds for the variables (servings of Dish 1 and Dish 2)
bounds = [(0, None), (0, None),(0, None),(0, None),(0, None)]  # Non-negative servings

# Solve the linear programming problem (maximization)
result = linprog(c=obj_coefficients, A_ub=lhs_coefficients, b_ub=rhs_values, bounds=bounds, method='highs')

if result.success:
    servings_dish1 = result.x[0]
    servings_dish2 = result.x[1]
    servings_dish3 = result.x[2]
    servings_dish4 = result.x[3]
    servings_dish5 = result.x[4]
```

```
    from openai import OpenAI
    client = OpenAI(api_key='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')
    # Formulate a query for OpenAI
    query = f"give me dinner and snacks single servings with Maximum Servings of eggs: {servings_dish1:.2f} and Maximum Servings of vege

    # Call OpenAI to get recommendations
    response = client.completions.create(
    model="gpt-3.5-turbo-instruct",
    prompt=query,
    max_tokens=300
    )
     # Extract and display the model's response
    model_response = response.choices[0].text
    recommendation = model_response
else:
   recommendation = "No optimal recommendations found"



wb_append = load_workbook("/Users/megh/Desktop/BAN693-CapstoneProject/Demo3/Data.xlsx")
sheet = wb_append.active
data = (date_of_entry,username,height,weight,age,gender,bmi,bmr,activity,daily_calories,daily_carbs_gms,daily_protein_gms,daily_fat_gms,r
sheet.append(data)
wb_append.save("/Users/megh/Desktop/BAN693-CapstoneProject/Demo3/Data.xlsx")

df = pd.read_excel("/Users/megh/Desktop/BAN693-CapstoneProject/Demo3/Data.xlsx")
df= df.loc[df['Name'] == username]
result = df.to_html(classes = 'table table-stripped')

text_file = open("./templates/table.html", "w")
text_file.write(result)
text_file.close()

return render_template('index.html', name = name, recommendation =recommendation)




_name__ == "__main__":
app.run(debug=True,port=5005)
```

Within the code snippet, the functionalities have been categorized into distinct sections:

- Flask Framework Integration: Routes and handling of different page requests.

- Image Processing: Utilization of TensorFlow and OpenCV for image analysis.
- Nutritional Analysis: Utilization of USDA API for retrieving food nutrient information.
- Mathematical Optimization: Usage of linear programming (linprog) for dietary recommendations.
- External API Integration: Connection and interaction with OpenAI's GPT-3 API for generating food recommendations.
- Data Handling and Storage: Calculation of nutritional values, user input processing, and Excel file management.

This code appendix serves as a reference for the technical implementation of the project's core functionalities, showcasing the integration of various technologies and algorithms to achieve the intended objectives.