Project Documentation

Vitamin Vision :Unveiling the spectrum of nutrient detection

1. INTRODUCTION

Vitamin Vision: Unveiling the Spectrum of Nutrient Detection" explores how modern technology can be used to identify and analyze essential nutrients in food. This project aims to enhance awareness of nutrient content using innovative detection methods, promoting better health and informed dietary choices.

Team members

Team Leader : Sadhu Anusri Mounika (228X1A05F8)

Team member: Shaik Reshma (228X1A05G1)

Team member : Seelam Pallavi (228X1A05F9)

Team Member : pilli Abhishikth Raj (228X1A05F6)

2. Project Overview

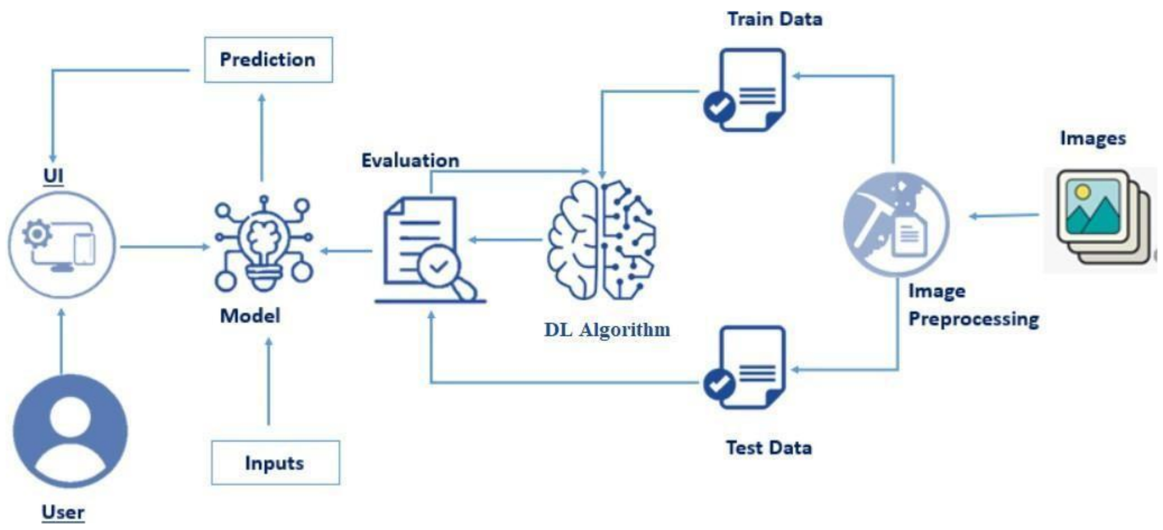
Purpose:

The Vitamin Vision project focuses on detecting and quantifying vitamins present in food, supplements, and biological fluids using machine learning, spectroscopy, and image processing. The main goals are to ensure product quality, verify nutritional claims, support regulatory compliance, and enable research in personalized nutrition and health.

Features:

- Detection and quantification of vitamins A, B, C, D, and E.
- Integration of a CNN (VGG19) model for image-based vitamin classification.
- Flask-based web application for interactive user interface.
- Dataset collection and preprocessing with data augmentation.
- Real-time prediction and visualization of results via Flask UI.

3. Architecture



Frontend:

Developed using HTML, CSS, and JavaScript, rendered through Flask templates. Pages include index.html, predict.html, and output.html. The frontend allows users to upload food images and view vitamin detection results in real time.

Backend:

Built with Flask (Python) for web serving and API handling. Integrates a Convolutional Neural Network (CNN) model based on VGG19 architecture for image classification. Handles model loading, prediction requests, and result rendering.

Database:

The system can integrate MongoDB for storing user uploads, historical predictions, and model data.

4. Setup Instructions

Prerequisites:

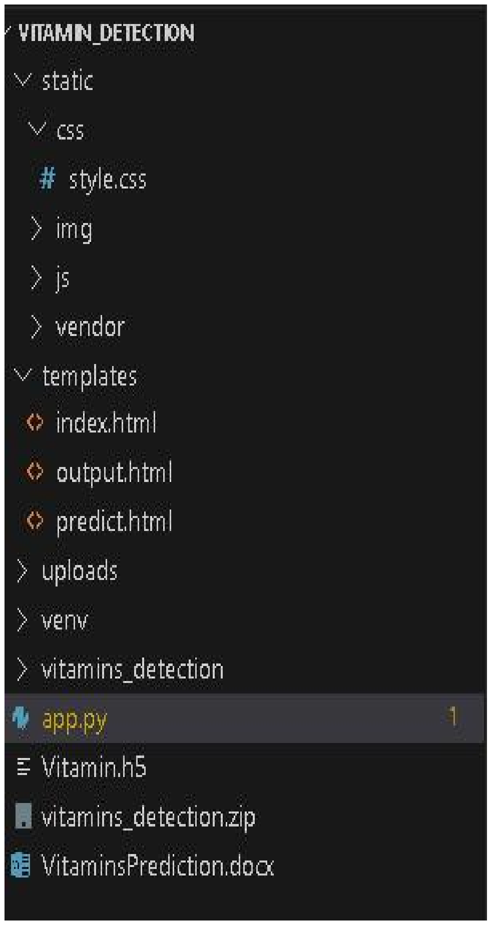
Python 3.x, Anaconda Navigator, Flask, VS Code or Jupyter Notebook, MongoDB (optional).

- Python Packages:
  - numpy
  - pandas
  - scikit-learn
  - tensorflow==2.12.0
  - keras==2.12.0
  - Flask

Installation Steps:

➢ 1. Clone the repository.
2. Create a virtual environment.
3. Install dependencies.
4. Download the dataset from Kaggle.
5. Configure environment variables if necessary.

5. Folder Structure



```
VITAMIN_DETECTION
  static
    css
      style.css
    img
    js
    vendor
  templates
    index.html
    output.html
    predict.html
  uploads
  venv
  vitamins_detection
  app.py                1
  Vitamin.h5
  vitamins_detection.zip
  VitaminsPrediction.docx
```

```
VitaminVision/
|
├── Data/            # Training & testing image datasets
├── templates/       # HTML templates (index.html, predict.html, output.html)
├── static/          # CSS, JS, and assets
├── app.py           # Flask backend
├── model/           # Trained CNN model (Vitamin.h5)
└── requirements.txt
```

Data Collection & Image Preprocessing

In this milestone First, we will collect images of Food then organized into subdirectories based on their respective names as shown in the project structure. Create folders of types of Vitamin Food that need to be recognized. In this project, we have collected images of 5 types of Images like Vitamin A, Vitamin B, Vitamin C, Vitamin D, Vitamin E. they are saved in the respective sub directories with their respective names.

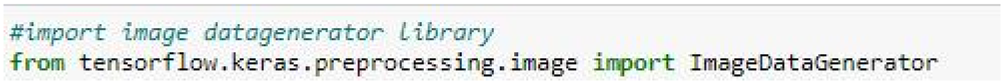
Download the Dataset - [https://www.kaggle.com/code/allulucky27/vitamin-detection deep-learning](https://www.kaggle.com/code/allulucky27/vitamin-detection-deep-learning)

In Image Processing, we will be improving the image data that suppresses unwilling distortions or enhances some image features important for further processing, although perform some geometric transformations of images like rotation, scaling, translation, etc.

Import the Image Data Generator library.

Image data augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of images in the dataset.

The Keras deep learning neural network library provides the capability to fit models using image data augmentation via the ImageDataGenerator class.
Let us import the ImageDataGenerator class from tensorflow Keras.



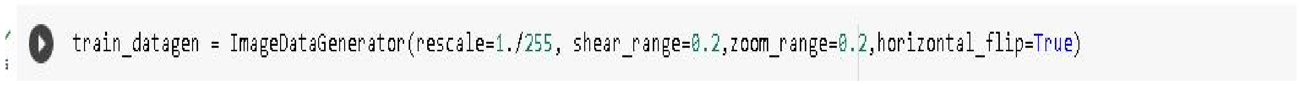
```
#import image datagenerator library
from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

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.



```
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,zoom_range=0.2,horizontal_flip=True)
```

Apply ImageDataGenerator functionality to Trainset and Testset

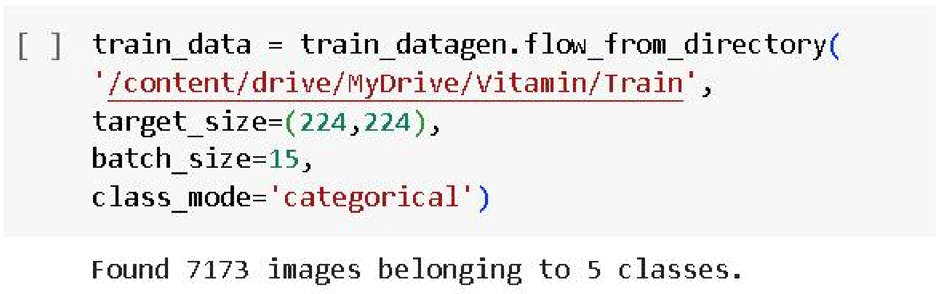
Let us apply ImageDataGenerator functionality to Train set and Test set by using the following code. For Training set using flow_from_directory function.

This function will return batches of images from the subdirectories Vitamin A, Vitamin B, Vitamin C, Vitamin D, Vitamin E together with labels 0 to 4{Vitamin A: 0, Vitamin B: 1, Vitamin C: 2, Vitamin D: 3, Vitamin E: 4 }

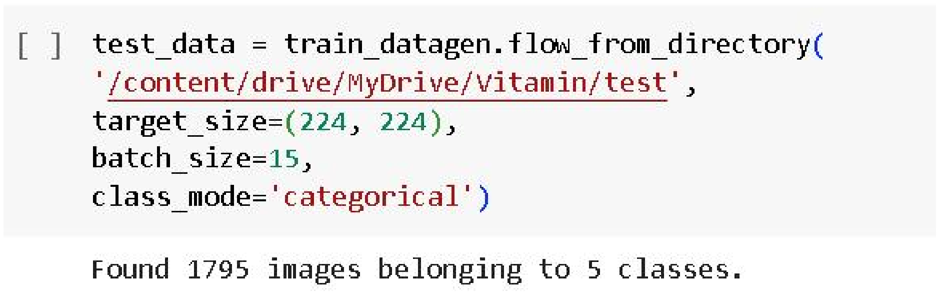
Arguments:

- 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 which is 15.
- 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).



```
[ ] train_data = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Vitamin/Train',
    target_size=(224,224),
    batch_size=15,
    class_mode='categorical')

    Found 7173 images belonging to 5 classes.
```



```
[ ] test_data = train_datagen.flow_from_directory(
    '/content/drive/MyDrive/Vitamin/test',
    target_size=(224, 224),
    batch_size=15,
    class_mode='categorical')

    Found 1795 images belonging to 5 classes.
```

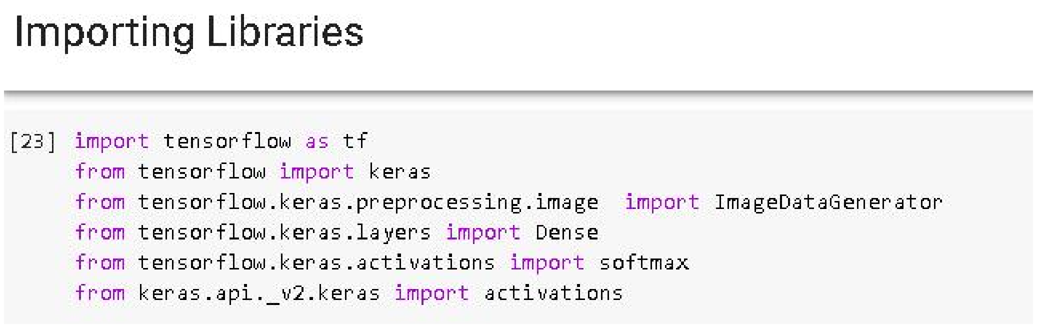
We notice that 7173 images belong to 5 classes for training and 1795 images belong to 5 classes for testing purposes.

Model Building

Now it's time to build our Convolutional Neural Networking using vgg19 which contains an input layer alongwith the convolution, max-pooling, and finally an output layer.

Importing the Model Building Libraries

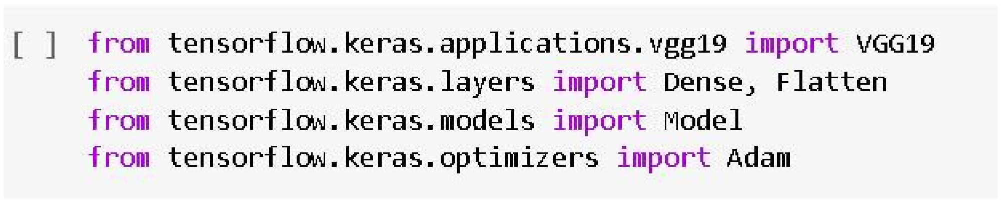
Importing the necessary libraries


Importing Libraries

```
[23] import tensorflow as tf
     from tensorflow import keras
     from tensorflow.keras.preprocessing.image  import ImageDataGenerator
     from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
     from keras.api._v2.keras import activations
```

Importing the VGG19 model

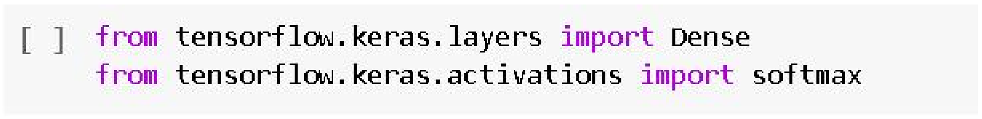
To initialize the VGG19 model, the weights are usually pre-trained on the ImageNet dataset, which is a large-scale dataset of images belonging to 1,000 different categories. These pre-trained weights can be downloaded from the internet, and they can be used as a starting point to fine-tune the model for a specific task, such as object recognition or classification.



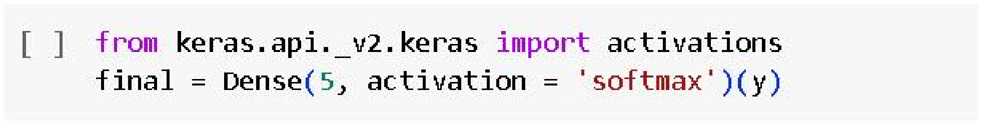
```
[ ]  from tensorflow.keras.applications.vgg19 import VGG19
     from tensorflow.keras.layers import Dense, Flatten
     from tensorflow.keras.models import Model
     from tensorflow.keras.optimizers import Adam
```

Adding Fully connected Layers

- For information regarding CNN Layers refer to the link Link: https://victorzhou.com/blog/intro-to-cnns-part-1/
- As the input image contains three channels, we are specifying the input shape as (128,128,3).
- We are adding a output layer with activation function as "softmax".



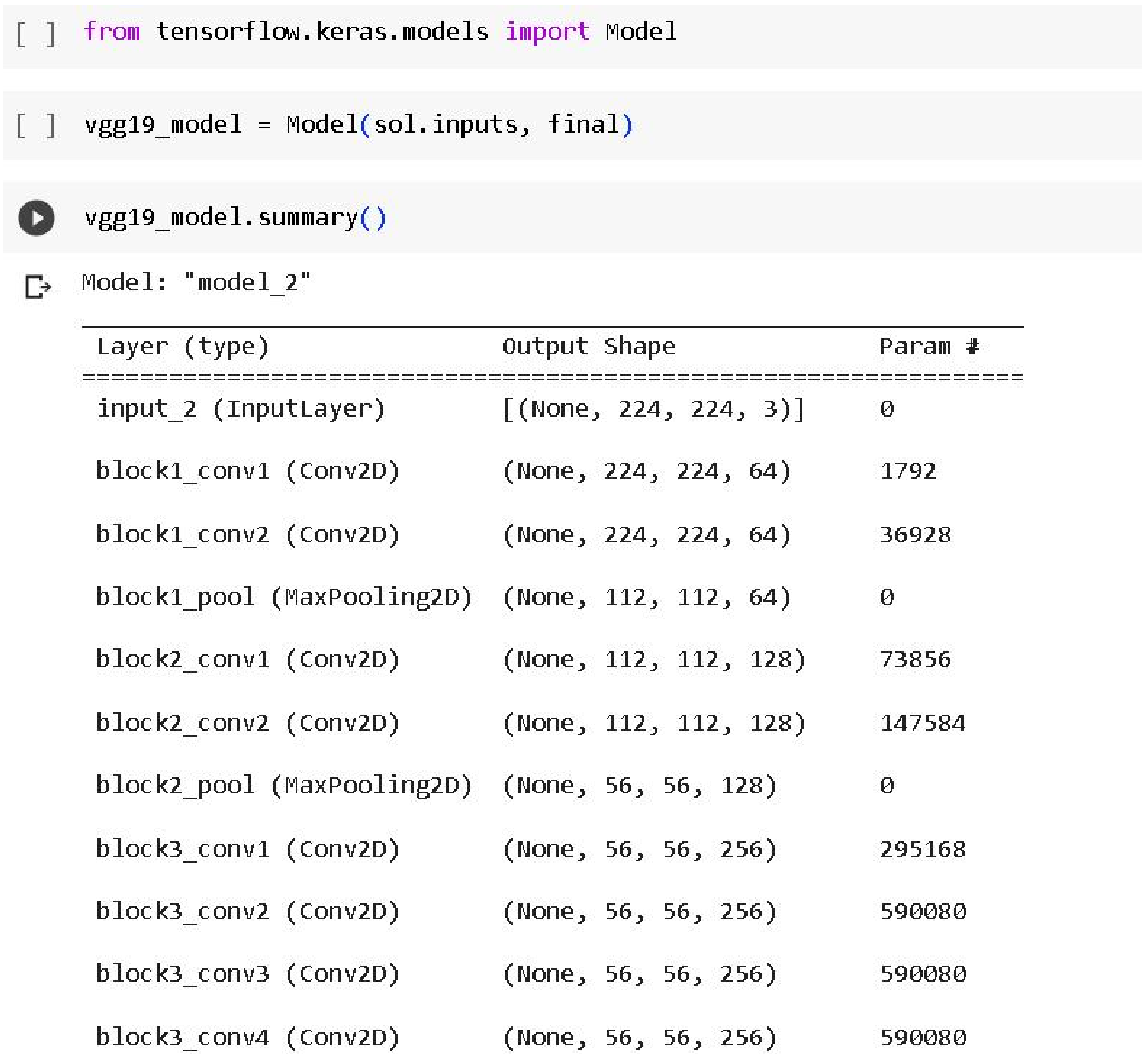
```
[ ]  from tensorflow.keras.layers import Dense
     from tensorflow.keras.activations import softmax
```



```
[ ]  from keras.api._v2.keras import activations
     final = Dense(5, activation = 'softmax')(y)
```

A dense layer is a deeply connected neural network layer. It is the most common and frequently used layer. The number of neurons in the Dense layer is the same as the number of classes in the training set. The neurons in the last Dense layer, use softmax activation to convert their outputs into respective probabilities.

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



```
[ ]  from tensorflow.keras.models import Model
```

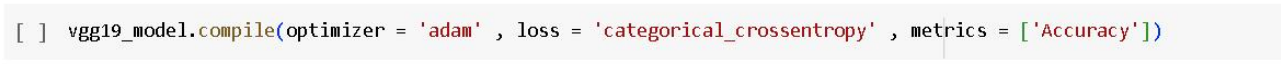
```
[ ]  vgg19_model = Model(sol.inputs, final)
```

```
vgg19_model.summary()
```

```
Model: "model_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_2 (InputLayer)        [(None, 224, 224, 3)]     0

 block1_conv1 (Conv2D)       (None, 224, 224, 64)      1792

 block1_conv2 (Conv2D)       (None, 224, 224, 64)      36928

 block1_pool (MaxPooling2D)  (None, 112, 112, 64)      0

 block2_conv1 (Conv2D)       (None, 112, 112, 128)     73856

 block2_conv2 (Conv2D)       (None, 112, 112, 128)     147584

 block2_pool (MaxPooling2D)  (None, 56, 56, 128)       0

 block3_conv1 (Conv2D)       (None, 56, 56, 256)       295168

 block3_conv2 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv3 (Conv2D)       (None, 56, 56, 256)       590080

 block3_conv4 (Conv2D)       (None, 56, 56, 256)       590080
```

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 inthe learning process. Keras requires a 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.



```
[ ]  vgg19_model.compile(optimizer = 'adam' , loss = 'categorical_crossentropy' , metrics = ['Accuracy'])
```

Train The model

Now, let us train our model with our image dataset. The model is trained for 9 epochs and after every epoch, the current model state is saved if the model has the least loss encountered till that time. We can see that the training loss decreases in almost every epoch till 30 epochs and probably there is further scope to improve the model.

fit_generator functions used to train a deep learning neural network.

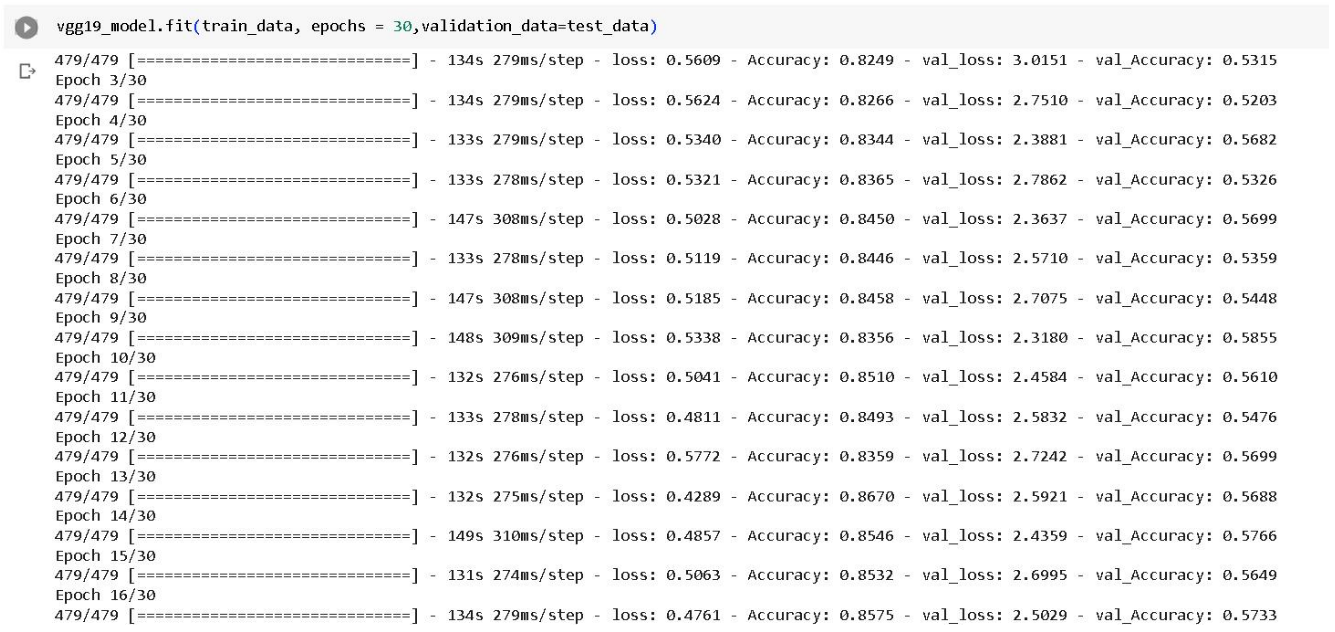
Arguments:

- steps_per_epoch: it specifies the total number of steps taken from the generator as soon as one epoch is finished and the next epoch has started. We can calculate the value of steps_per_epoch as the total number of samples in your dataset divided by the batch size.

- Epochs: an integer and number of epochs we want to train our model for.

- validation_data can be either:
  - an inputs and targets list
  - a generator
  - an inputs, targets, and sample_weights list which can be used to evaluate the loss and metrics for any model after any epoch has ended.
- validation_steps: only if the validation_data is a generator then only this argument can be used. It specifies the total number of steps taken from the generator before it is

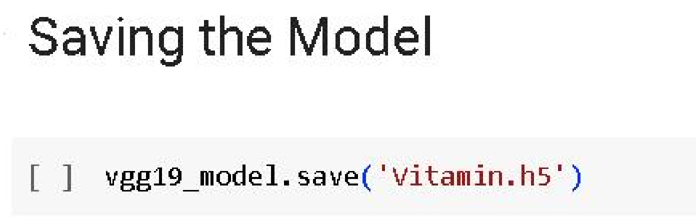
stopped at every epoch and its value is calculated as the total number of validation data points in your dataset divided by the validation batch size.



```
vgg19_model.fit(train_data, epochs = 30,validation_data=test_data)

479/479 [==============================] - 134s 279ms/step - loss: 0.5609 - Accuracy: 0.8249 - val_loss: 3.0151 - val_Accuracy: 0.5315
Epoch 3/30
479/479 [==============================] - 134s 279ms/step - loss: 0.5624 - Accuracy: 0.8266 - val_loss: 2.7510 - val_Accuracy: 0.5203
Epoch 4/30
479/479 [==============================] - 133s 279ms/step - loss: 0.5340 - Accuracy: 0.8344 - val_loss: 2.3881 - val_Accuracy: 0.5682
Epoch 5/30
479/479 [==============================] - 133s 278ms/step - loss: 0.5321 - Accuracy: 0.8365 - val_loss: 2.7862 - val_Accuracy: 0.5326
Epoch 6/30
479/479 [==============================] - 147s 308ms/step - loss: 0.5028 - Accuracy: 0.8450 - val_loss: 2.3637 - val_Accuracy: 0.5699
Epoch 7/30
479/479 [==============================] - 133s 278ms/step - loss: 0.5119 - Accuracy: 0.8446 - val_loss: 2.5710 - val_Accuracy: 0.5359
Epoch 8/30
479/479 [==============================] - 147s 308ms/step - loss: 0.5185 - Accuracy: 0.8458 - val_loss: 2.7075 - val_Accuracy: 0.5448
Epoch 9/30
479/479 [==============================] - 148s 309ms/step - loss: 0.5338 - Accuracy: 0.8356 - val_loss: 2.3180 - val_Accuracy: 0.5855
Epoch 10/30
479/479 [==============================] - 132s 276ms/step - loss: 0.5041 - Accuracy: 0.8510 - val_loss: 2.4584 - val_Accuracy: 0.5610
Epoch 11/30
479/479 [==============================] - 133s 278ms/step - loss: 0.4811 - Accuracy: 0.8493 - val_loss: 2.5832 - val_Accuracy: 0.5476
Epoch 12/30
479/479 [==============================] - 132s 276ms/step - loss: 0.5772 - Accuracy: 0.8359 - val_loss: 2.7242 - val_Accuracy: 0.5699
Epoch 13/30
479/479 [==============================] - 132s 275ms/step - loss: 0.4289 - Accuracy: 0.8670 - val_loss: 2.5921 - val_Accuracy: 0.5688
Epoch 14/30
479/479 [==============================] - 149s 310ms/step - loss: 0.4857 - Accuracy: 0.8546 - val_loss: 2.4359 - val_Accuracy: 0.5766
Epoch 15/30
479/479 [==============================] - 131s 274ms/step - loss: 0.5063 - Accuracy: 0.8532 - val_loss: 2.6995 - val_Accuracy: 0.5649
Epoch 16/30
479/479 [==============================] - 134s 279ms/step - loss: 0.4761 - Accuracy: 0.8575 - val_loss: 2.5029 - val_Accuracy: 0.5733
```

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 the Model

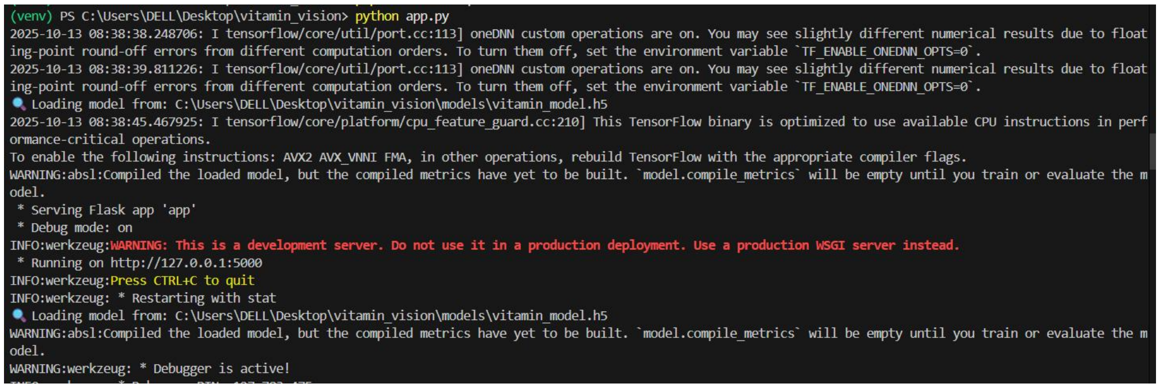
```
[ ] vgg19_model.save('Vitamin.h5')
```

6.Running the Application

Frontend: npm start in the client directory.

Backend: python app.py in the server directory.

The Flask server runs on http://127.0.0.1:5000



```
(venv) PS C:\Users\DELL\Desktop\vitamin_vision> python app.py
2025-10-13 08:38:38.248706: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to float
ing-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
2025-10-13 08:38:39.811226: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical results due to float
ing-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONEDNN_OPTS=0`.
🔍 Loading model from: C:\Users\DELL\Desktop\vitamin_vision\models\vitamin_model.h5
2025-10-13 08:38:45.467925: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in perf
ormance-critical operations.
To enable the following instructions: AVX2 AVX_VNNI FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the m
odel.
 * Serving Flask app 'app'
 * Debug mode: on
INFO:werkzeug:WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug: * Restarting with stat
🔍 Loading model from: C:\Users\DELL\Desktop\vitamin_vision\models\vitamin_model.h5
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty until you train or evaluate the m
odel.
WARNING:werkzeug: * Debugger is active!
```

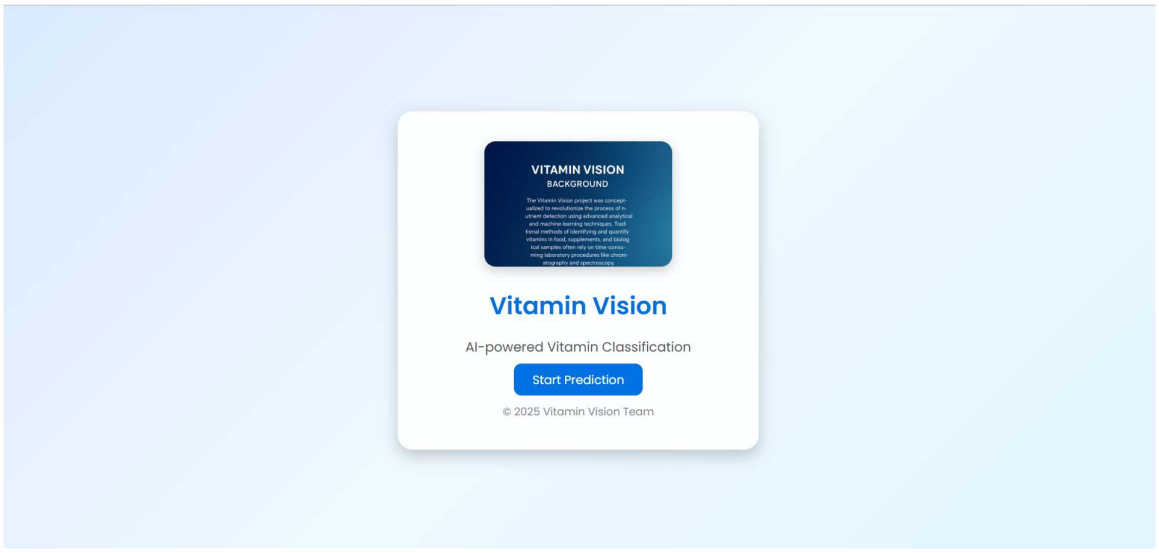
7.API Documentation

Endpoints:

- /predict [POST] - Takes an image and predicts the vitamin type.
- /upload [POST] - Saves uploaded image.
- /results [GET] - Displays prediction results.

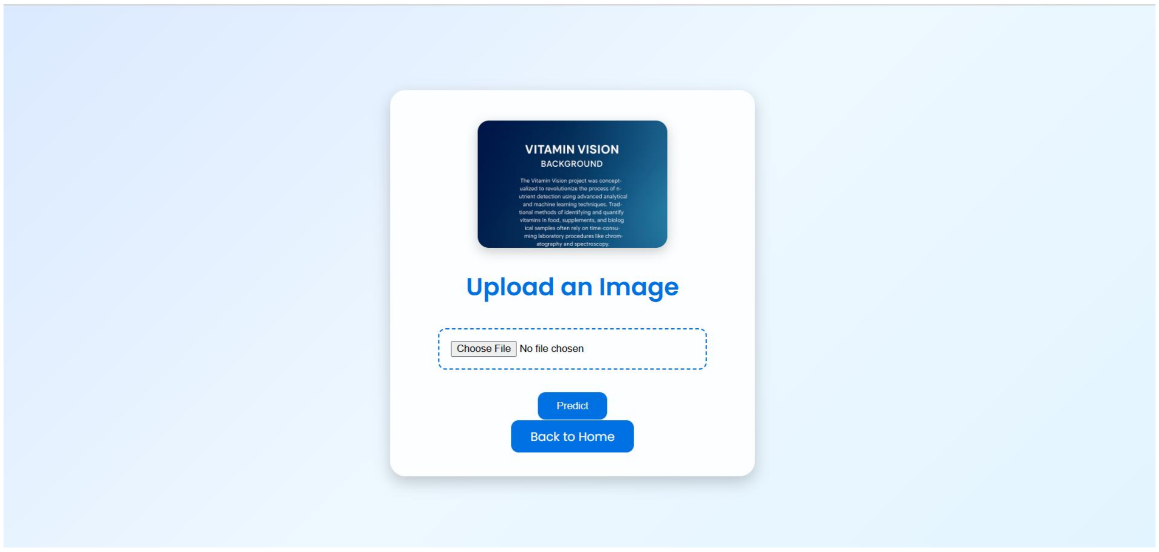
8.Outcomes:

Home page :



The Vitamin Vision project was conceptualized to revolutionize the process of nutrient detection using advanced analytical and machine learning techniques. Traditional methods of identifying and quantifying vitamins in food, supplements, and biological samples often rely on time-consuming laboratory procedures like chromatography and spectroscopy.

Upload page :



VITAMIN VISION
BACKGROUND
The Vitamin Vision project was conceptualized to revolutionize the process of nutrient detection using advanced analytical and machine learning techniques. Traditional methods of identifying and quantify vitamins in food, supplements, and biological samples often rely on time-consuming laboratory procedures like chromatography and spectroscopy.

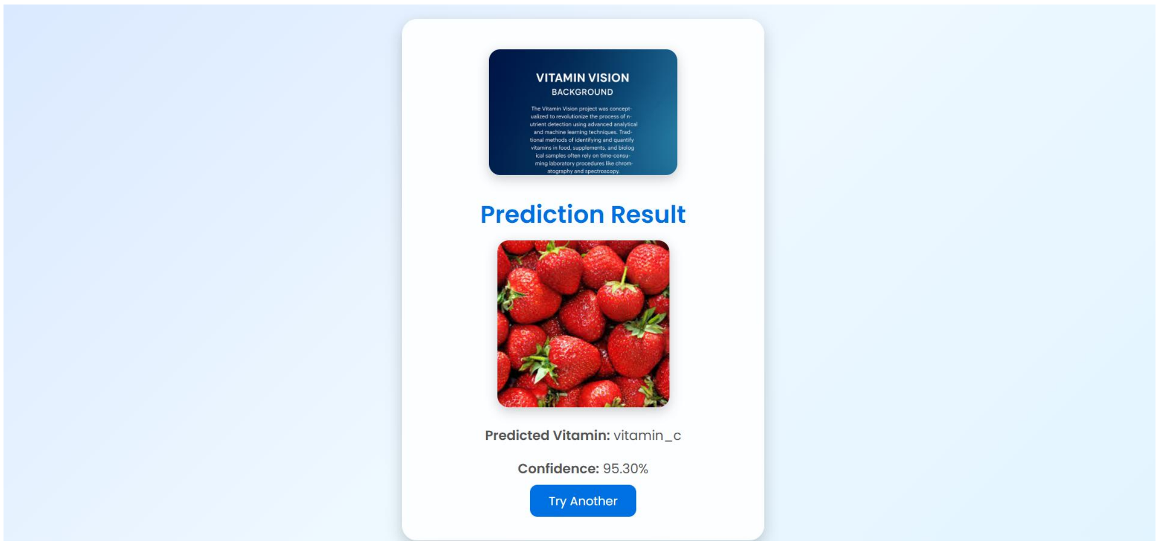
Upload an Image

Choose File No file chosen

Predict

Back to Home

Result page :



VITAMIN VISION
BACKGROUND
The Vitamin Vision project was conceptualized to revolutionize the process of nutrient detection using advanced analytical and machine learning techniques. Traditional methods of identifying and quantify vitamins in food, supplements, and biological samples often rely on time-consuming laboratory procedures like chromatography and spectroscopy.

Prediction Result

Predicted Vitamin: vitamin_c

Confidence: 95.30%

Try Another

9.Reference :

Github link : https://github.com/sadhu-anusrimounika22/vitamin_vision

Drive link :
https://drive.google.com/drive/folders/1Rg35nniN0wZrNtFQbzyORoL4vAhEESiQ

Video demo link: https://drive.google.com/drive/folders/1Tjf-zzCKJePAy32HWxvdBUQ-x3MZirWD

10. Known Issues

• Limited dataset affects accuracy.
• Mixed food images not fully supported.
• Mobile integration pending.

11. Future Enhancements

• Expand detection to include minerals, calories, and macronutrients.
• Integrate mobile app and AR support.
• Implement voice-based interaction and cloud model deployment.
• Improve accuracy using larger datasets and advanced CNN architectures.