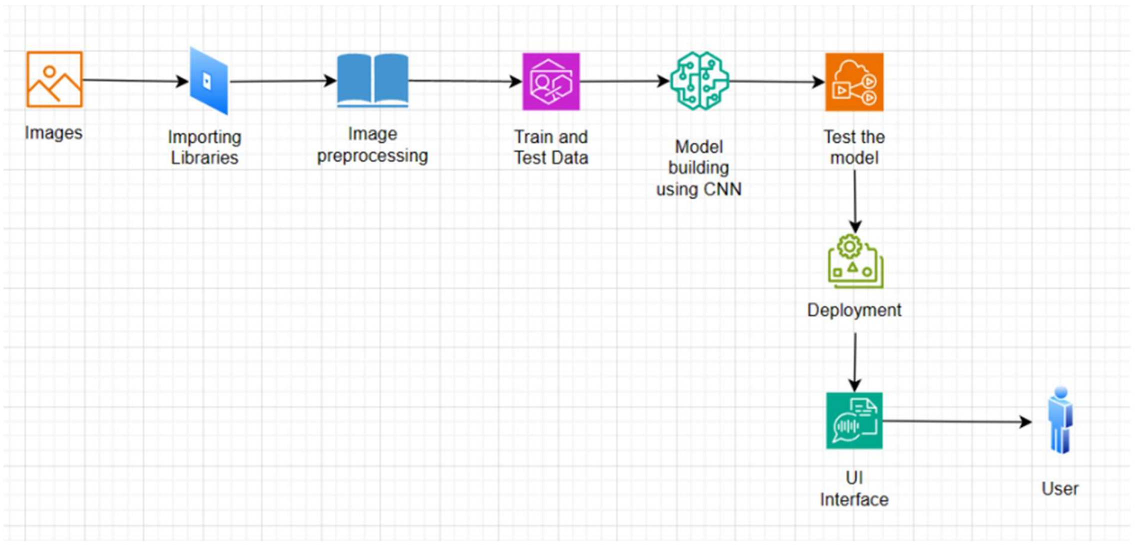
Vitamin Detection using Deep Learning

Introduction:

Vitamins are essential for maintaining overall health and are found in various foods in different amounts. Traditional methods of analyzing vitamins in food are time-consuming. Still, recent advancements in deep learning and computer vision, using the basics of trained neural networks (CNNs), offer a faster and more accurate approach. These networks identify and quantify vitamins in food, providing valuable information for nutritional analysis, meal planning, and ensuring adequate vitamin intake. This application of technology has the potential to revolutionize how we assess and manage nutrition.

Technical Architecture:



Prerequisites:

To complete this project, you need specific software, concepts, and packages:

Software:

- **Anaconda Navigator:** A free and open-source distribution of Python and R for data science and machine learning. It supports Windows, Linux, and macOS. Anaconda includes tools like JupyterLab, Jupyter Notebook, QtConsole, Spyder, Glueviz, Orange, Rstudio, and Visual Studio Code. For this project, we'll use Jupyter Notebook and VS Code.

- **Conda:** An open-source, cross-platform package management system.
- **Video Tutorial:** Refer to this video (Link: Click here to) to install Anaconda Navigator and understand how to use Jupyter Notebook and Spyder.

Machine Learning Packages:

- **Numpy:** An open-source numerical Python library with multidimensional array and matrix data structures for mathematical operations.
- **Scikit-learn:** A free machine learning library for Python, featuring algorithms like support vector machines, random forests, and k-neighbors. It supports Python numerical and scientific libraries like NumPy and SciPy.
- **Flask:** A web framework used for building web applications.
- **Python Packages:** Open Anaconda Prompt as administrator and install the required packages:
  - pip install numpy
  - pip install pandas
  - pip install scikit-learn
  - pip install tensorflow==2.12.0
  - pip install keras==2.12.0
  - pip install Flask
- **Deep Learning Concepts:**
  - **CNN (Convolutional Neural Network):** A class of deep neural networks commonly used for analyzing visual imagery.
  - **Flask Basics:** Flask is a popular Python web framework for developing web applications.
- If you use PyCharm IDE, you can install packages through the command prompt using the same syntax mentioned above.

Project Objectives:

By the project's conclusion, you will:

- Acquire foundational knowledge and skills in Convolutional Neural Networks.
- Develop a comprehensive understanding of image data.
- Learn various data preprocessing techniques for cleaning and preparing data.
- Understand the process of building a web application using the Flask framework.

Project Flow:

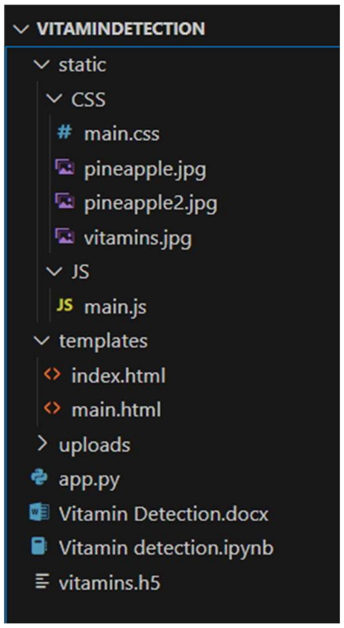
- The user engages with the User Interface (UI) to select an image.
- The chosen image undergoes analysis by the integrated model within the Flask application.
- CNN models examine the image, and the resulting prediction is displayed on the Flask UI.

Completing all listed activities & tasks is essential to achieve these objectives, project flow:

- Data Collection
    - Collect data.
    - Establish Train and Test folders.
- Data Preprocessing
    - Utilize the ImageDataGenerator library.
    - Configure the ImageDataGenerator class.
    - Apply ImageDataGenerator functionality to both the Train and Test datasets.
- Model Building
    - Import necessary model-building libraries.
    - Initialize the CNN model.
    - Add a fully connected layer.
    - Configure the learning process.
    - Train and test the basic CNN model.
    - Save the trained model.
- Application Building
    - Develop an HTML file.
    - Construct Python code for the application.

Project Structure:

Create a Project folder that contains files as shown below.



```
VITAMINDETECTION
  static
    CSS
      main.css
      pineapple.jpg
      pineapple2.jpg
      vitamins.jpg
    JS
      main.js
  templates
    index.html
    main.html
  uploads
  app.py
  Vitamin Detection.docx
  Vitamin detection.ipynb
  vitamins.h5
```

- The Data folder encompasses images designated for training and testing our model.
- For our Flask Application, HTML pages are required and should be stored in the "**templates**" folder. Additionally, a Python script named "**app.py**" is needed for server-side scripting.
- The saved model crucial for this content is "**Vitamins.h5**".
- Within the "**templates**" folder, you will find pages named **index.html**, **main.html**.

<u>Milestone 1:</u> Define Problem / Problem Understanding

Activity 1: Specify the business problem.

Refer to the Project Description.

Activity 2: Business Requirements

a. Accurate Prediction:

Ensure precise classification of food images to maintain model accuracy.

b. Real-time Data Acquisition:

Efficiently acquire real-time data from various sources for up-to-date information.

c. User-friendly Interface:

Develop a straightforward and easy-to-navigate interface for presenting predictor results.

d. Report Generation:

Generate clear and concise reports outlining predicted vitamins with key findings and insights.

Activity 3: Literature Survey

The literature survey on vitamin detection focused on exploring existing studies related to predicting vitamin content in food. The survey aimed to:

- Current Systems:
    - Assess methodologies for vitamin detection in food images.
    - Evaluate the accuracy levels achieved by various existing systems.

- Methods and Techniques:
    - Investigate methods and techniques used in previous projects.
    - Identify successful approaches and shortcomings for potential project insights.

- Data and Findings:
    - Examine relevant data and findings from past studies.
    - Extract insights to contribute to the development of the current vitamin detection project.

- Strengths and Weaknesses:
    - Evaluate existing systems for accuracy, real-time capabilities, and user interface design.
    - Understand limitations to guide improvements in the proposed system.

- Gaps in Knowledge:
    - Identify gaps in current literature.
    - Determine how the project can address these gaps and contribute to existing knowledge.

Activity 4: Social or Business Impact

- **Social Impact:**

  Accurate prediction of vitamin deficiencies enables early detection, promoting improved health outcomes. Individuals can make informed decisions about their health, leading to a sense of control and well-being.

- **Business Impact:**

  Vitamin prediction systems enhance preventive care, personalized medicine, and fitness optimization. Healthcare providers can tailor interventions, researchers can discover trends, and the wellness industry can offer personalized plans for better outcomes.

Milestone 2: Data Collection & Image Preprocessing:

For data collection, gather images of various foods categorized into subdirectories based on their respective names according to the project structure. Create folders for different types of Vitamin Foods to be recognized. This project includes images of Vitamin A, Vitamin B, Vitamin C, Vitamin D, and Vitamin E, each saved in their respective subdirectories.

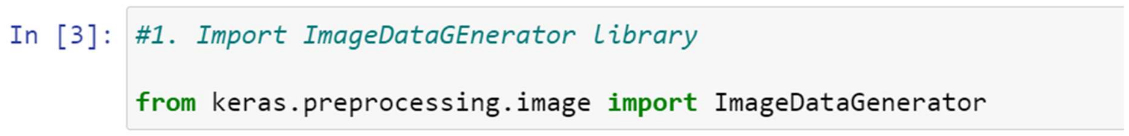
Download the dataset -
https://www.kaggle.com/code/allulucky27/vitamin-detectiondeep-learning

In image preprocessing, enhance image data by suppressing distortions and highlighting features essential for further processing. Apply geometric transformations like rotation, scaling, and translation to improve the dataset.

Activity 1: Import the ImageDataGenerator library.

Image data augmentation is a technique to artificially increase the size of a training dataset by generating modified versions of images in the dataset.

The Keras deep learning neural network library offers the ability to train models using image data augmentation through the ImageDataGenerator class.
Import the ImageDataGenerator class from the TensorFlow Keras module.



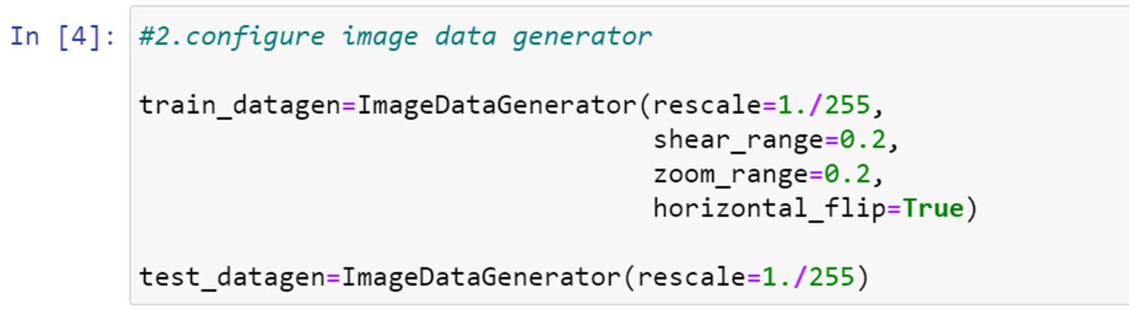
```
In [3]: #1. Import ImageDataGEnerator library

        from keras.preprocessing.image import ImageDataGenerator
```

Activity 2: Configure ImageDataGenerator class

ImageDataGenerator class is instantiated and the configuration for the types of data augmentation. There are five main types of data augmentation techniques for image data; specifically:

- Image shifts via the width_shift_range and height_shift_range arguments.
- 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.



```
In [4]: #2.configure image data generator

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

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

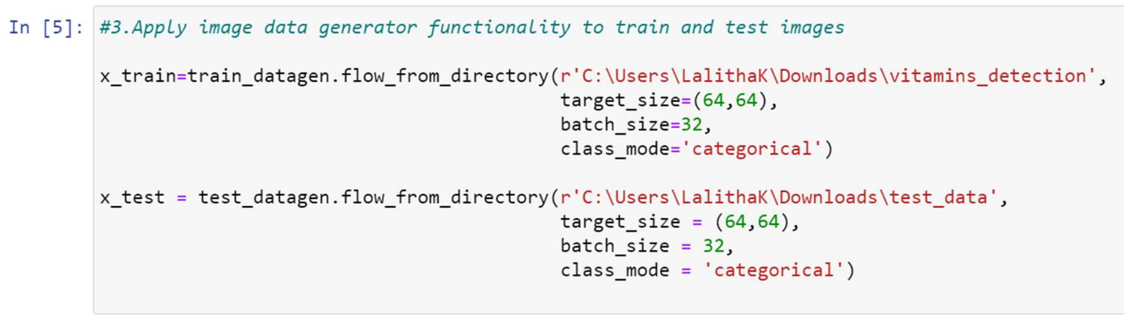
Activity 3: Apply ImageDataGenerator to Train and Test Sets

Apply ImageDataGenerator functionality to the training and test sets using the following code. For the training set, use the flow_from_directory function.

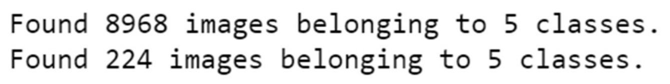
This function returns batches of images from the subdirectories **Vitamin A, Vitamin B, Vitamin C, Vitamin D, and Vitamin E, along with labels 0 to 4** {Vitamin A: 0, Vitamin B: 1, Vitamin C: 2, Vitamin D: 3, Vitamin E: 4}.

Parameters:

- **directory:** The location of the data. If labels are "inferred," it should contain subdirectories, each holding images for a class. Otherwise, the directory structure is ignored.
- **batch_size:** The size of the data batches, set to 32.
- **target_size:** The size to resize images after reading them from disk.
- **class_mode:**
    - **'int':** Labels are encoded as integers (e.g., for sparse_categorical_crossentropy loss).
    - **'categorical':** Labels are encoded as a categorical vector (e.g., for categorical_crossentropy loss).
    - **'binary':** Labels (only 2) are encoded as float32 scalars with values 0 or 1 (e.g., for binary_crossentropy).
    - **'None':** No labels.



```
In [5]: #3.Apply image data generator functionality to train and test images

        x_train=train_datagen.flow_from_directory(r'C:\Users\LalithaK\Downloads\vitamins_detection',
                                                  target_size=(64,64),
                                                  batch_size=32,
                                                  class_mode='categorical')

        x_test = test_datagen.flow_from_directory(r'C:\Users\LalithaK\Downloads\test_data',
                                                  target_size = (64,64),
                                                  batch_size = 32,
                                                  class_mode = 'categorical')
```



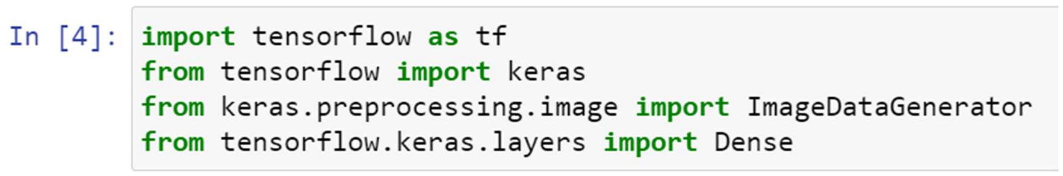
```
Found 8968 images belonging to 5 classes.
Found 224 images belonging to 5 classes.
```

Milestone 3: Model Building

Now it's time to build our Basic Convolutional Neural Networking,

Activity 1: Importing the Model Building Libraries

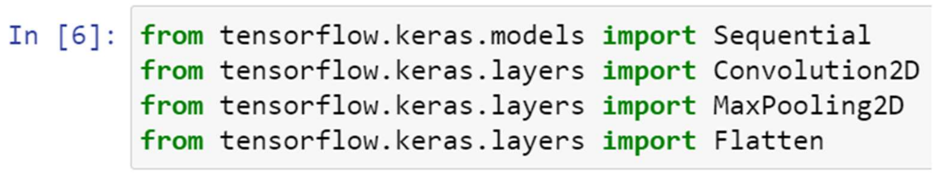
Importing the necessary libraries.



```
In [4]: import tensorflow as tf
        from tensorflow import keras
        from keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.layers import Dense
```

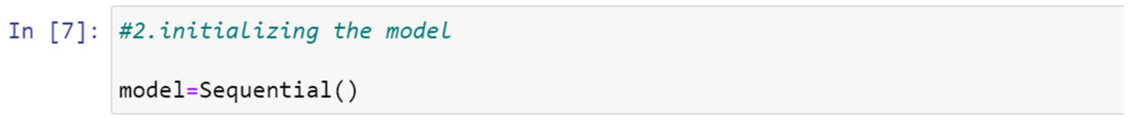
Activity 2: Importing the Basic CNN model

For a basic Convolutional Neural Network (CNN) model, you typically initialize the model architecture and weights from scratch.

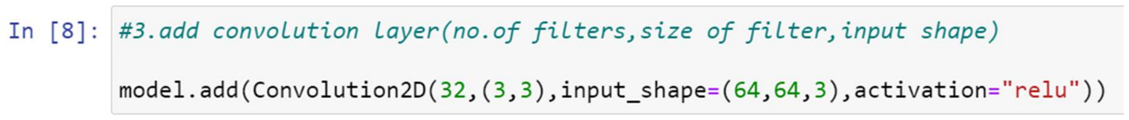


```
In [6]: from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Convolution2D
        from tensorflow.keras.layers import MaxPooling2D
        from tensorflow.keras.layers import Flatten
```

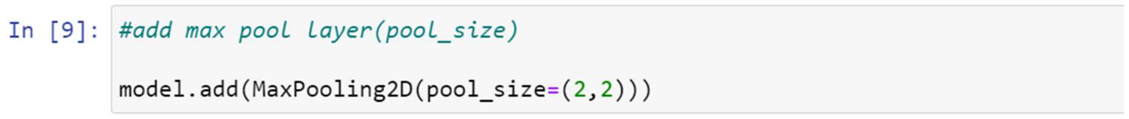
Activity 3: Initializing the model:



```
In [7]: #2.initializing the model

        model=Sequential()
```



```
In [8]: #3.add convolution layer(no.of filters,size of filter,input shape)

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



```
In [9]: #add max pool layer(pool_size)

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

In Keras, the Sequential model is a linear stack of layers, where you can add one layer at a time. It's a straightforward way to build neural networks layer by layer.

Convolutional Layer (Convolution2D):

Role: This layer applies convolutional operations to the input data, helping the network learn spatial hierarchies of features.

Parameters:

32: Number of filters (output channels).

(3, 3): Size of the convolutional kernel (filter).

input_shape=(64, 64, 3): Shape of the input data, representing an image with dimensions 64x64 pixels and 3 color channels (RGB).

activation="relu": Rectified Linear Unit (ReLU) activation function is used to introduce non-linearity.

Max Pooling Layer (MaxPooling2D):

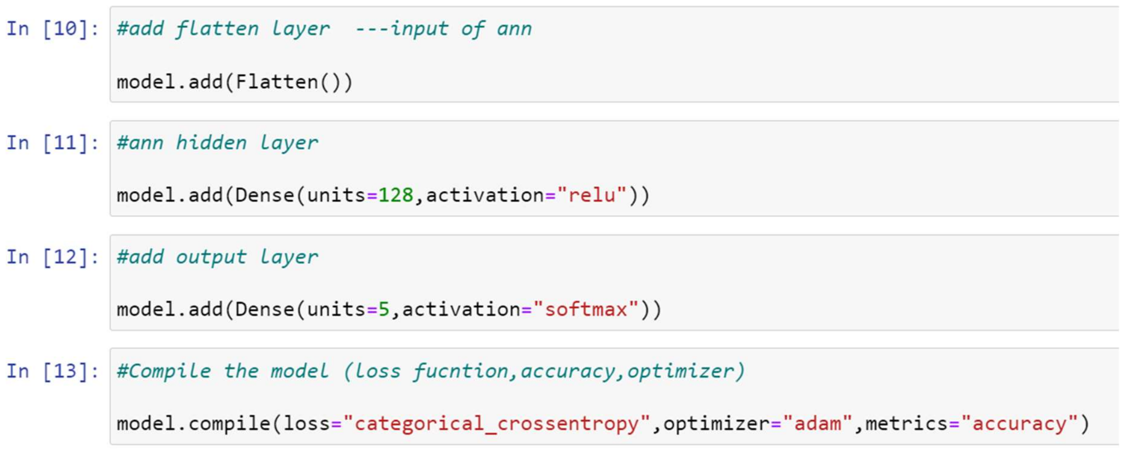
Role: This layer downsamples the spatial dimensions of the input, reducing computational complexity and preserving the most important features.

Parameters:

pool_size=(2, 2): The size of the pooling window, which defines the factor by which the spatial dimensions are reduced.

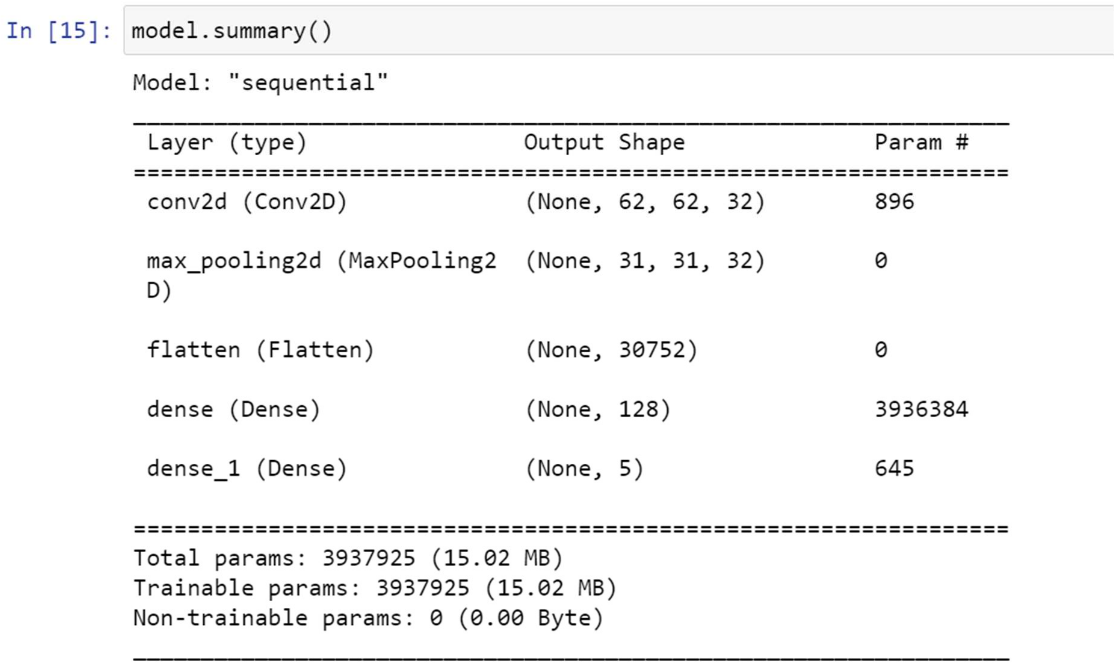
Activity 4: Adding Fully connected Layers

- The model architecture includes a flattening layer, followed by two fully connected (dense) layers.
- The network is designed for a multi-class classification task with five output classes.
- The ReLU activation is used in the first dense layer, and softmax activation is used in the output layer.
- The model is compiled with categorical crossentropy loss, Adam optimizer, and accuracy as the evaluation metric.



```
In [10]: #add flatten layer  ---input of ann

         model.add(Flatten())

In [11]: #ann hidden layer

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

In [12]: #add output layer

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

In [13]: #Compile the model (loss fucntion,accuracy,optimizer)

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

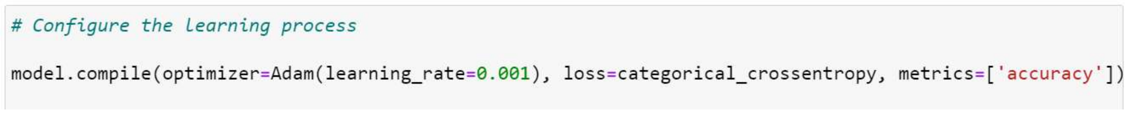
Understanding the model is a very important phase to properly use it for training and prediction purposes. Keras provides a simple method, and summary to get the full information about the model and its layers.



```
In [15]: model.summary()

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

 max_pooling2d (MaxPooling2  (None, 31, 31, 32)        0
 D)

 flatten (Flatten)           (None, 30752)             0

 dense (Dense)               (None, 128)               3936384

 dense_1 (Dense)             (None, 5)                 645

=================================================================
Total params: 3937925 (15.02 MB)
Trainable params: 3937925 (15.02 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Activity 5: Configure The Learning Process

The compilation is the final step in creating a model, requiring a loss function (used to find errors) and an optimizer (here, we use Adam) for weight optimization. Metrics, similar to the loss function but not used in training, are chosen to evaluate model performance**.**

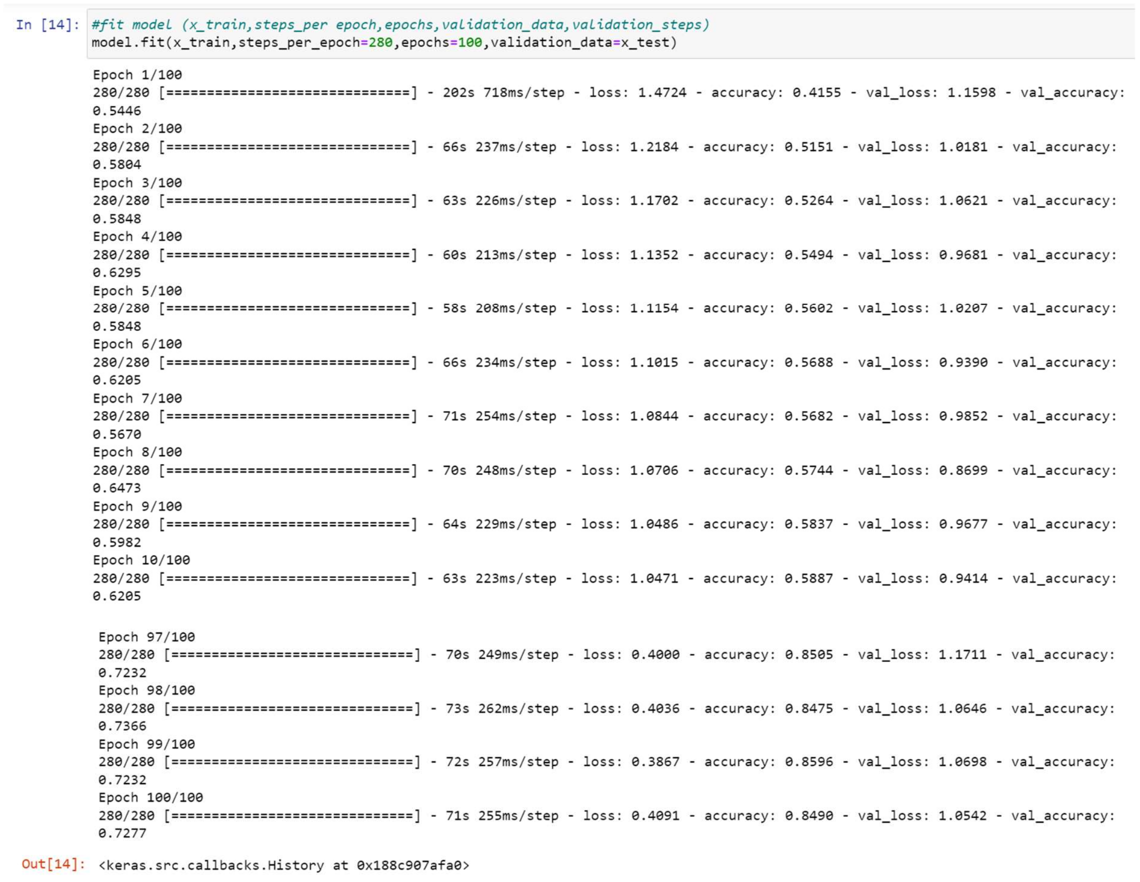


```
# Configure the learning process

model.compile(optimizer=Adam(learning_rate=0.001), loss=categorical_crossentropy, metrics=['accuracy'])
```

Activity 6: Train The model

The model is fitted using the fit function with the following parameters:

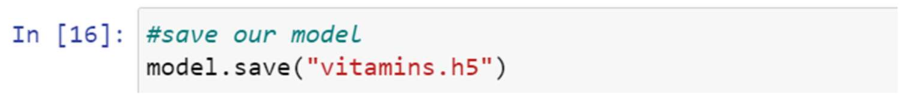
- **x_train:** Training data.
- **steps_per_epoch:** Set to 280, calculated as the total number of training samples divided by the batch size.
- **epochs:** Set to 100, indicating the number of training epochs.
- **validation_data:** Set to x_test for validation.



```
In [14]: #fit model (x_train,steps_per epoch,epochs,validation_data,validation_steps)
         model.fit(x_train,steps_per_epoch=280,epochs=100,validation_data=x_test)

         Epoch 1/100
         280/280 [==============================] - 202s 718ms/step - loss: 1.4724 - accuracy: 0.4155 - val_loss: 1.1598 - val_accuracy:
         0.5446
         Epoch 2/100
         280/280 [==============================] - 66s 237ms/step - loss: 1.2184 - accuracy: 0.5151 - val_loss: 1.0181 - val_accuracy:
         0.5804
         Epoch 3/100
         280/280 [==============================] - 63s 226ms/step - loss: 1.1702 - accuracy: 0.5264 - val_loss: 1.0621 - val_accuracy:
         0.5848
         Epoch 4/100
         280/280 [==============================] - 60s 213ms/step - loss: 1.1352 - accuracy: 0.5494 - val_loss: 0.9681 - val_accuracy:
         0.6295
         Epoch 5/100
         280/280 [==============================] - 58s 208ms/step - loss: 1.1154 - accuracy: 0.5602 - val_loss: 1.0207 - val_accuracy:
         0.5848
         Epoch 6/100
         280/280 [==============================] - 66s 234ms/step - loss: 1.1015 - accuracy: 0.5688 - val_loss: 0.9390 - val_accuracy:
         0.6205
         Epoch 7/100
         280/280 [==============================] - 71s 254ms/step - loss: 1.0844 - accuracy: 0.5682 - val_loss: 0.9852 - val_accuracy:
         0.5670
         Epoch 8/100
         280/280 [==============================] - 70s 248ms/step - loss: 1.0706 - accuracy: 0.5744 - val_loss: 0.8699 - val_accuracy:
         0.6473
         Epoch 9/100
         280/280 [==============================] - 64s 229ms/step - loss: 1.0486 - accuracy: 0.5837 - val_loss: 0.9677 - val_accuracy:
         0.5982
         Epoch 10/100
         280/280 [==============================] - 63s 223ms/step - loss: 1.0471 - accuracy: 0.5887 - val_loss: 0.9414 - val_accuracy:
         0.6205

          Epoch 97/100
          280/280 [==============================] - 70s 249ms/step - loss: 0.4000 - accuracy: 0.8505 - val_loss: 1.1711 - val_accuracy:
          0.7232
          Epoch 98/100
          280/280 [==============================] - 73s 262ms/step - loss: 0.4036 - accuracy: 0.8475 - val_loss: 1.0646 - val_accuracy:
          0.7366
          Epoch 99/100
          280/280 [==============================] - 72s 257ms/step - loss: 0.3867 - accuracy: 0.8596 - val_loss: 1.0698 - val_accuracy:
          0.7232
          Epoch 100/100
          280/280 [==============================] - 71s 255ms/step - loss: 0.4091 - accuracy: 0.8490 - val_loss: 1.0542 - val_accuracy:
          0.7277

Out[14]: <keras.src.callbacks.History at 0x188c907afa0>
```

Activity 7: Save the Model

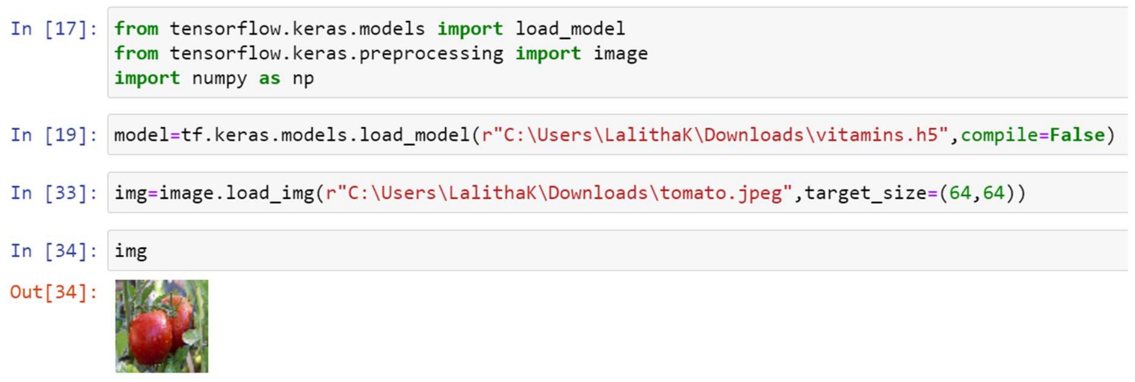
The model is saved with a .h5 extension. An H5 file, in the Hierarchical Data Format (HDF), stores multidimensional arrays of scientific data.



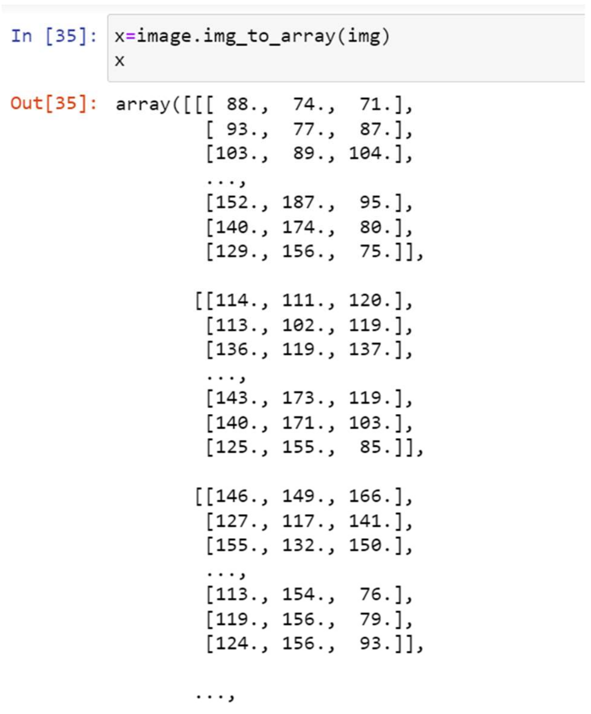
```
In [16]: #save our model
         model.save("vitamins.h5")
```

Activity 8: Test The model

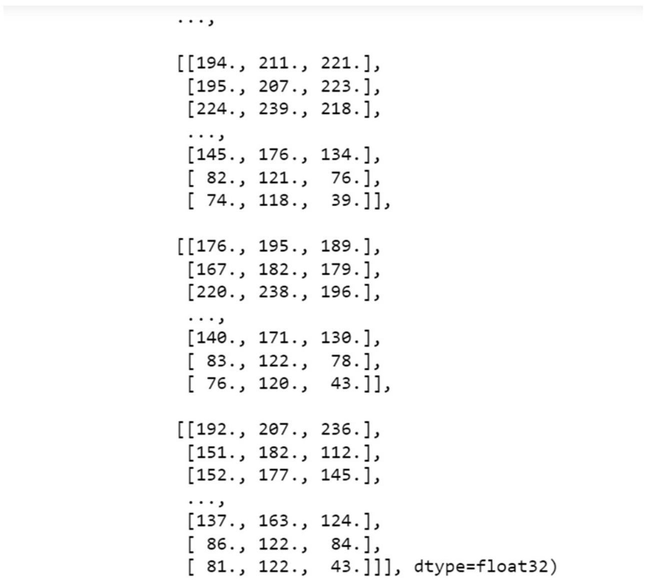
Evaluation is a process during the development of the model to check whether the model is the best fit for the given problem and corresponding data.



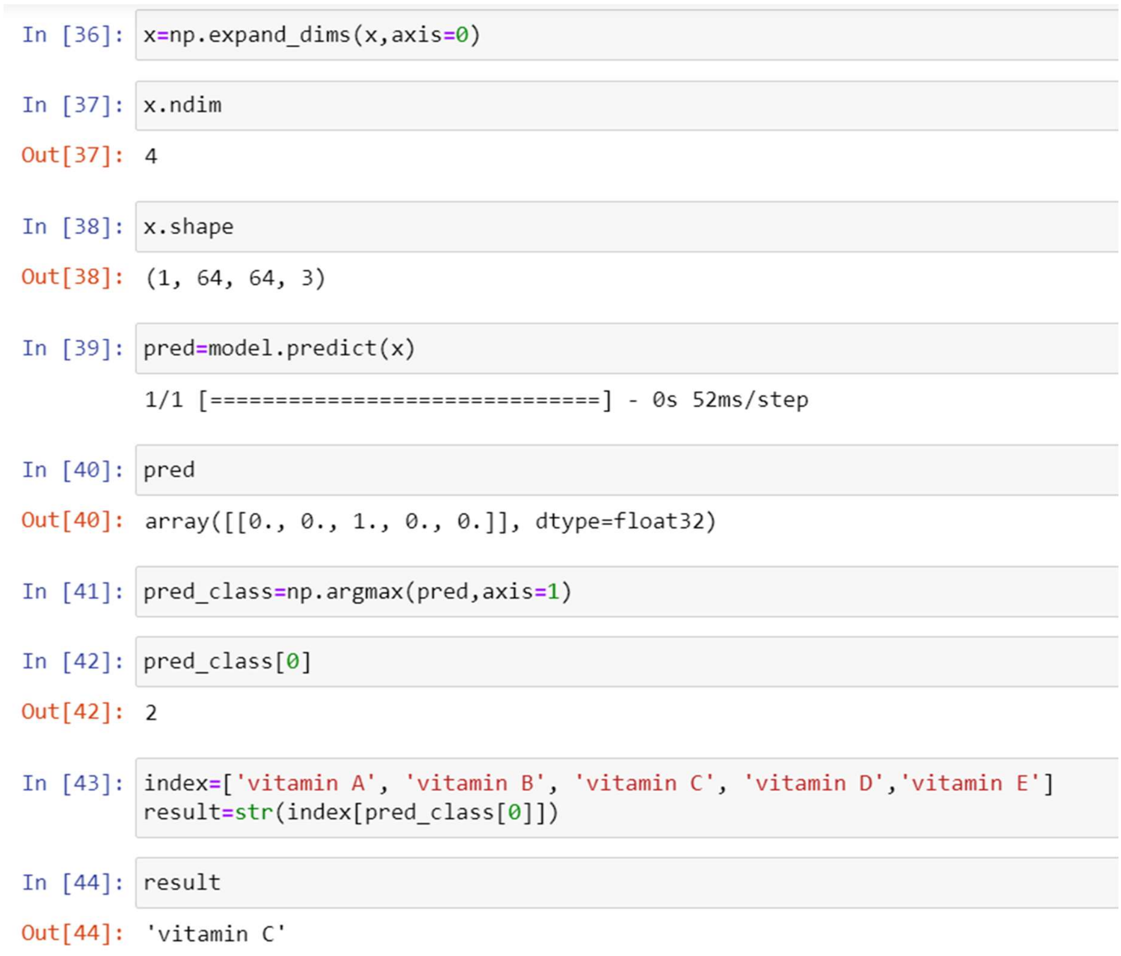
```
In [17]: from tensorflow.keras.models import load_model
         from tensorflow.keras.preprocessing import image
         import numpy as np

In [19]: model=tf.keras.models.load_model(r"C:\Users\LalithaK\Downloads\vitamins.h5",compile=False)

In [33]: img=image.load_img(r"C:\Users\LalithaK\Downloads\tomato.jpeg",target_size=(64,64))

In [34]: img

Out[34]:
```



```
In [35]: x=image.img_to_array(img)
         x

Out[35]: array([[[ 88.,  74.,  71.],
                 [ 93.,  77.,  87.],
                 [103.,  89., 104.],
                 ...,
                 [152., 187.,  95.],
                 [140., 174.,  80.],
                 [129., 156.,  75.]],

                [[114., 111., 120.],
                 [113., 102., 119.],
                 [136., 119., 137.],
                 ...,
                 [143., 173., 119.],
                 [140., 171., 103.],
                 [125., 155.,  85.]],

                [[146., 149., 166.],
                 [127., 117., 141.],
                 [155., 132., 150.],
                 ...,
                 [113., 154.,  76.],
                 [119., 156.,  79.],
                 [124., 156.,  93.]],

                ...,
```



```
         ...,

        [[194., 211., 221.],
         [195., 207., 223.],
         [224., 239., 218.],
         ...,
         [145., 176., 134.],
         [ 82., 121.,  76.],
         [ 74., 118.,  39.]],

        [[176., 195., 189.],
         [167., 182., 179.],
         [220., 238., 196.],
         ...,
         [140., 171., 130.],
         [ 83., 122.,  78.],
         [ 76., 120.,  43.]],

        [[192., 207., 236.],
         [151., 182., 112.],
         [152., 177., 145.],
         ...,
         [137., 163., 124.],
         [ 86., 122.,  84.],
         [ 81., 122.,  43.]]], dtype=float32)
```



```
In [36]: x=np.expand_dims(x,axis=0)

In [37]: x.ndim

Out[37]: 4

In [38]: x.shape

Out[38]: (1, 64, 64, 3)

In [39]: pred=model.predict(x)

         1/1 [==============================] - 0s 52ms/step

In [40]: pred

Out[40]: array([[0., 0., 1., 0., 0.]], dtype=float32)

In [41]: pred_class=np.argmax(pred,axis=1)

In [42]: pred_class[0]

Out[42]: 2

In [43]: index=['vitamin A', 'vitamin B', 'vitamin C', 'vitamin D','vitamin E']
         result=str(index[pred_class[0]])

In [44]: result

Out[44]: 'vitamin C'
```

Milestone 4: Application Building

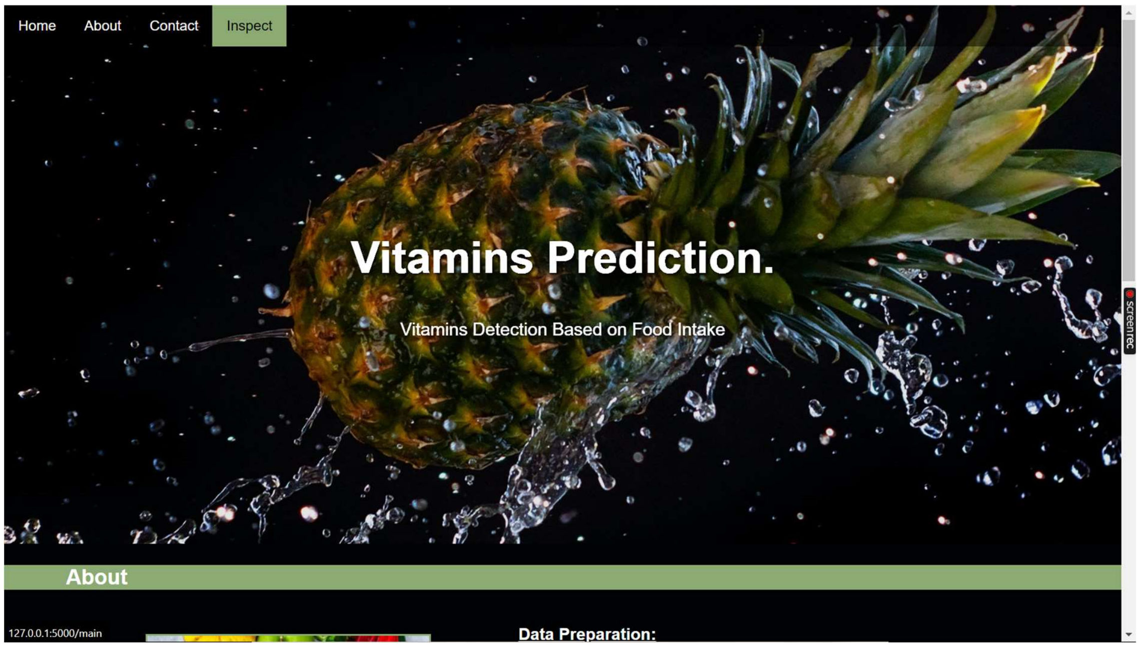
After training the model, a Flask application is built for local use. The app takes input parameters from an HTML page, uses them to predict Colon Diseases types, and displays the results. The "Inspect" button opens a page for image selection and prediction.

Activity 1: Create HTML Pages

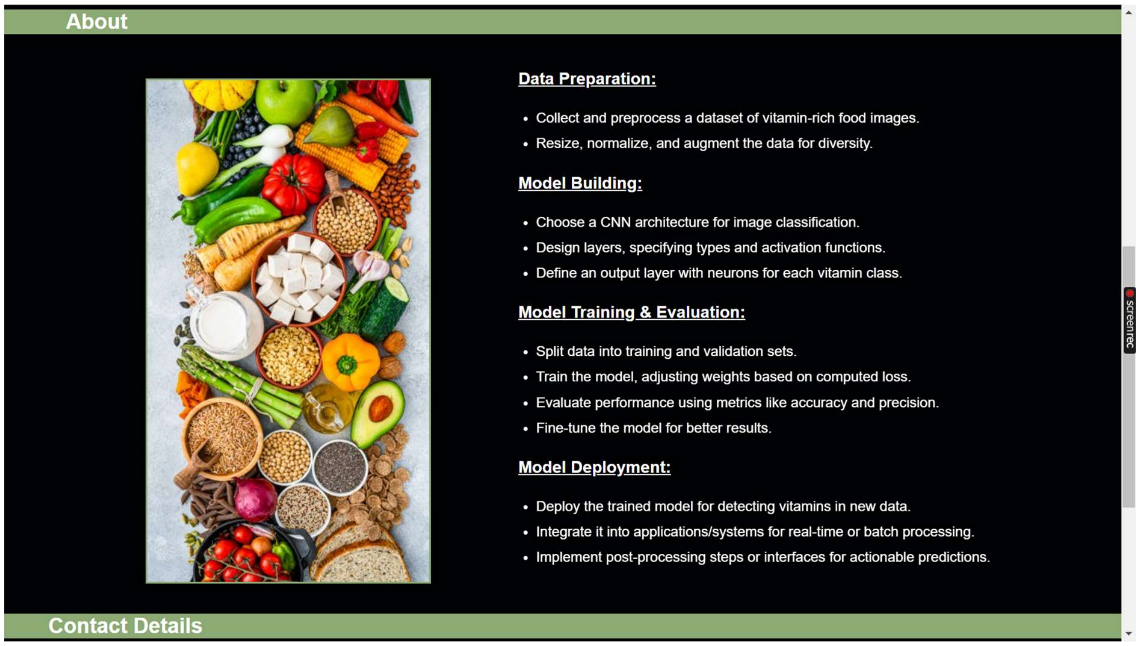
- HTML is utilized to design the front end of the web page.
- Three HTML pages are created: index.html, main.html.
- index.html serves as the homepage, introduction to the project, and issues an emergency alert.
- JavaScript (main.js) and CSS (main.css) are employed to enhance functionality and page aesthetics.

index.html

- **Home Page**



- **About section**



About

Data Preparation:

- Collect and preprocess a dataset of vitamin-rich food images.
- Resize, normalize, and augment the data for diversity.

Model Building:

- Choose a CNN architecture for image classification.
- Design layers, specifying types and activation functions.
- Define an output layer with neurons for each vitamin class.

Model Training & Evaluation:

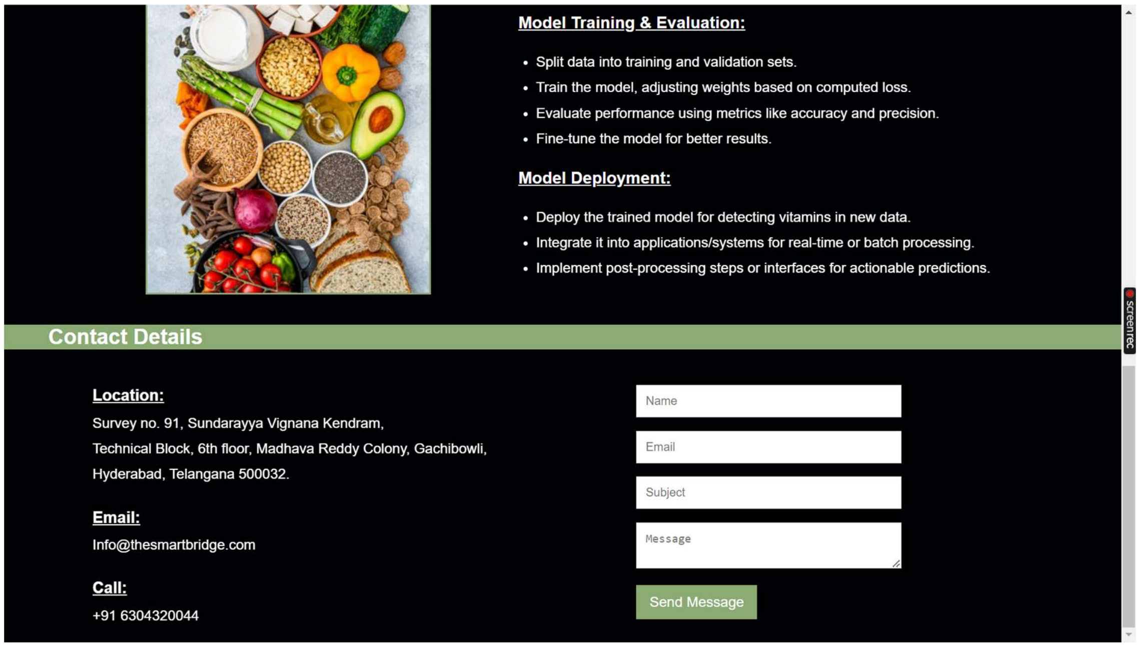
- Split data into training and validation sets.
- Train the model, adjusting weights based on computed loss.
- Evaluate performance using metrics like accuracy and precision.
- Fine-tune the model for better results.

Model Deployment:

- Deploy the trained model for detecting vitamins in new data.
- Integrate it into applications/systems for real-time or batch processing.
- Implement post-processing steps or interfaces for actionable predictions.

Contact Details

- **Contact Details**



Model Training & Evaluation:

- Split data into training and validation sets.
- Train the model, adjusting weights based on computed loss.
- Evaluate performance using metrics like accuracy and precision.
- Fine-tune the model for better results.

Model Deployment:

- Deploy the trained model for detecting vitamins in new data.
- Integrate it into applications/systems for real-time or batch processing.
- Implement post-processing steps or interfaces for actionable predictions.

Contact Details

Location:
Survey no. 91, Sundarayya Vignana Kendram,
Technical Block, 6th floor, Madhava Reddy Colony, Gachibowli,
Hyderabad, Telangana 500032.

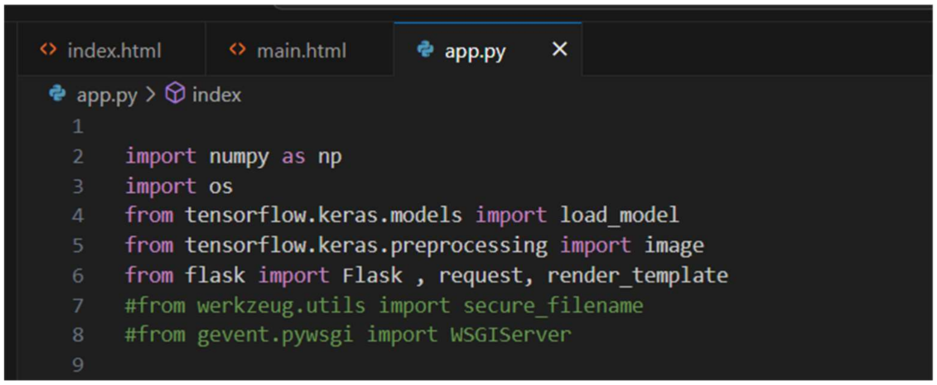
Email:
Info@thesmartbridge.com

Call:
+91 6304320044

Activity 2: Build Python code

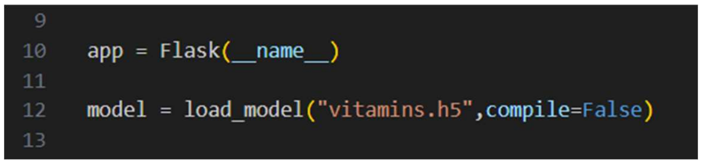
Task 1: Importing Libraries

The initial step involves importing necessary libraries. Flask is mandatory, creating a Flask object for the WSGI application. The Flask constructor uses the name of the current module as an argument. Additionally, the Pickle library is imported to load the model file.



```python

import numpy as np
import os
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask , request, render_template
#from werkzeug.utils import secure_filename
#from gevent.pywsgi import WSGIServer

```

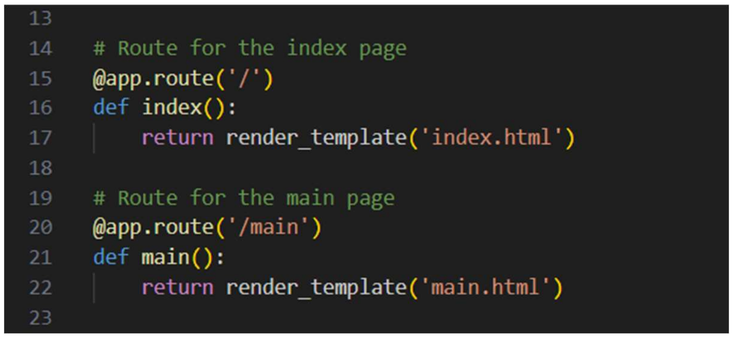
Task 2: Creating our flask application and loading our model by using load_model method



```python

app = Flask(__name__)

model = load_model("vitamins.h5",compile=False)

```

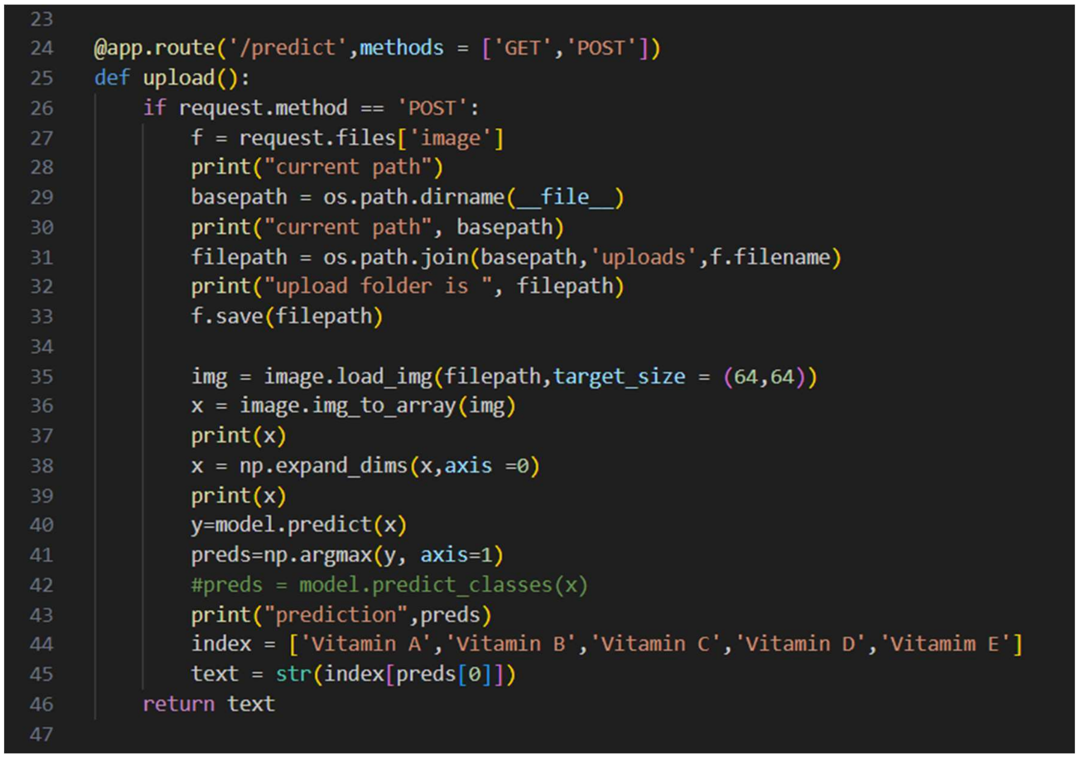
Task 3: Routing to the HTML Page

Using the constructor, routing to the created HTML pages is established. The '/' URL is linked to the index.html function, rendering the HTML page when the home page is opened. Image selection from the HTML page can be accessed via either the POST or GET method.



```python

# Route for the index page
@app.route('/')
def index():
    return render_template('index.html')

# Route for the main page
@app.route('/main')
def main():
    return render_template('main.html')

```

Showcasing prediction on UI:

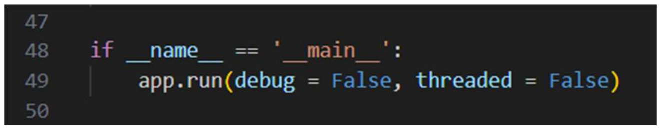


```python

@app.route('/predict',methods = ['GET','POST'])
def upload():
    if request.method == 'POST':
        f = request.files['image']
        print("current path")
        basepath = os.path.dirname(__file__)
        print("current path", basepath)
        filepath = os.path.join(basepath,'uploads',f.filename)
        print("upload folder is ", filepath)
        f.save(filepath)

        img = image.load_img(filepath,target_size = (64,64))
        x = image.img_to_array(img)
        print(x)
        x = np.expand_dims(x,axis =0)
        print(x)
        y=model.predict(x)
        preds=np.argmax(y, axis=1)
        #preds = model.predict_classes(x)
        print("prediction",preds)
        index = ['Vitamin A','Vitamin B','Vitamin C','Vitamin D','Vitamim E']
        text = str(index[preds[0]])
    return text

```

A function is defined to request a browsed file from the HTML page using the POST method. The file is saved to the "uploads" folder in the same directory via the OS library. Using the Keras library's Load Image class, the saved picture is retrieved from the declared path. Image processing techniques are applied, and the preprocessed image is sent to the model for predicting the class. The numerical class value (e.g., 0, 1, 2, 3, 4) in the 0th index of the variable "preds" is returned. This numerical value is assigned to the "index" variable, providing the name of the class. The predicted class name is rendered to the "prediction" variable used in the HTML page.

Finally, Run the application

This is used to run the application in a local host.

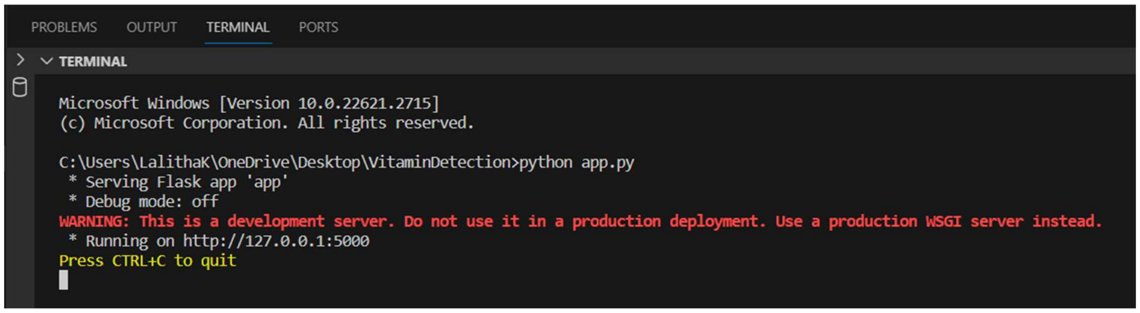


```python

if __name__ == '__main__':
    app.run(debug = False, threaded = False)

```

Activity 3: Run the application

- Open Anaconda Prompt.
- Navigate to the folder containing app.py.
- Type "python app.py" to run the app.
- Access the local host at http://127.0.0.1:5000/.
- Copy and open the URL in a browser.
- Enter values, click predict, and view the result on the web page.

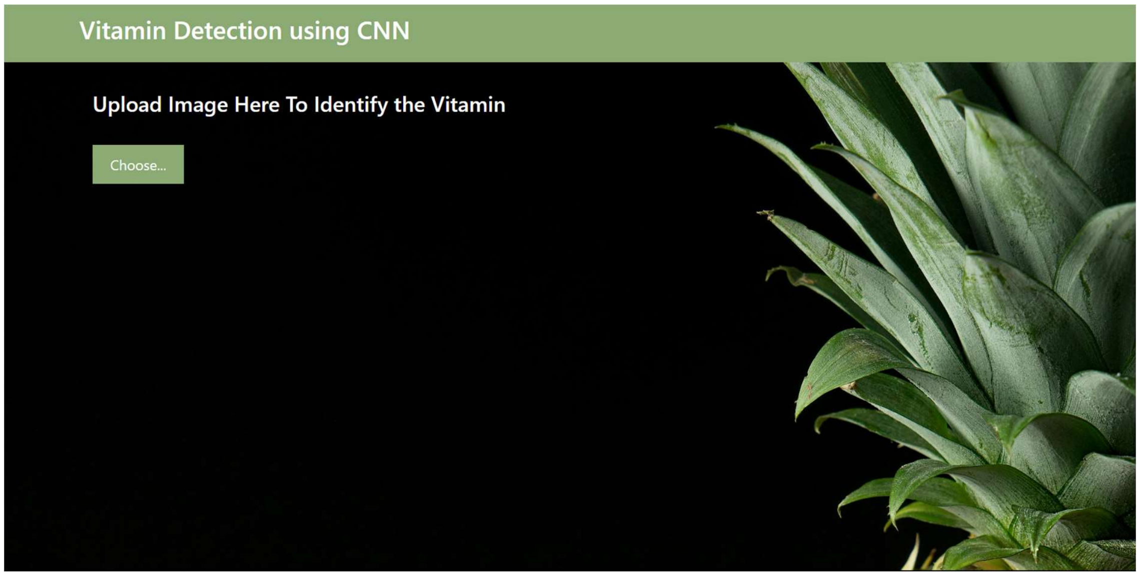
Then it will run on localhost: 5000

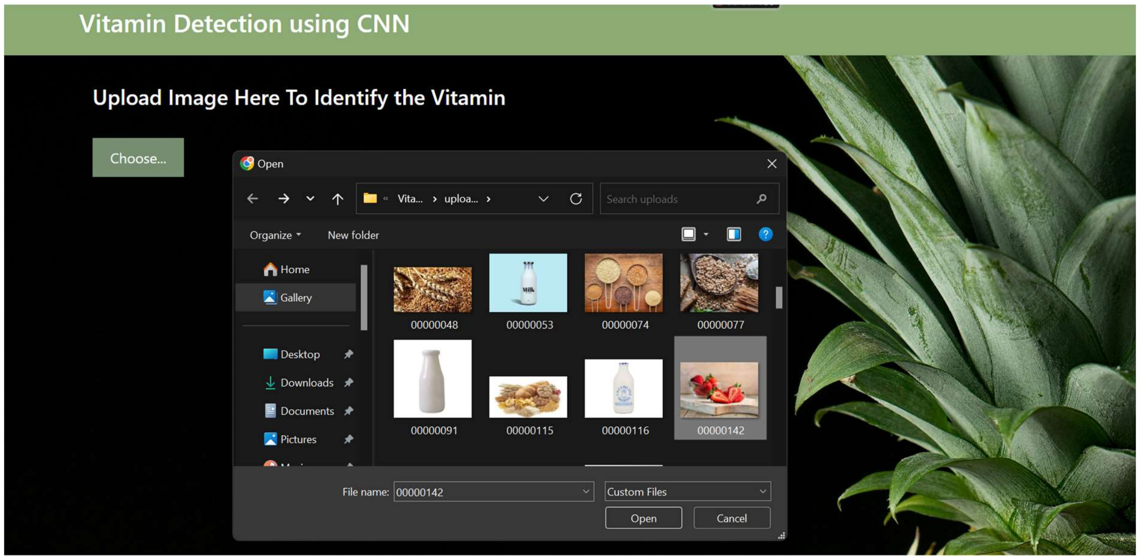


```
Microsoft Windows [Version 10.0.22621.2715]
(c) Microsoft Corporation. All rights reserved.

C:\Users\LalithaK\OneDrive\Desktop\VitaminDetection>python app.py
 * Serving Flask app 'app'
 * Debug mode: off
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
Press CTRL+C to quit
```

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

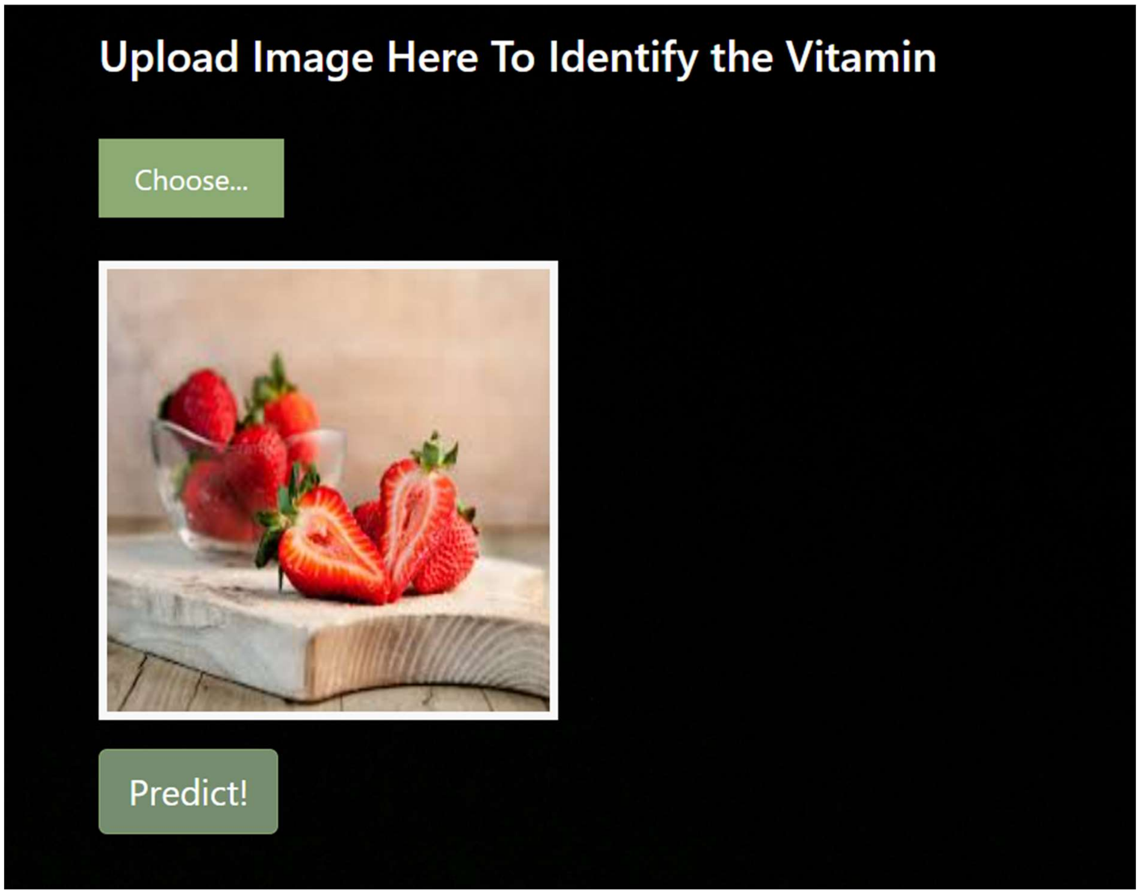
FINAL OUTPUT:


Vitamin Detection using CNN
Upload Image Here To Identify the Vitamin
Choose...

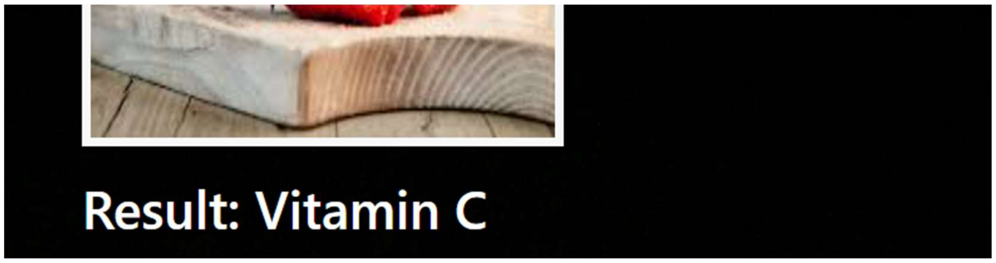


Vitamin Detection using CNN

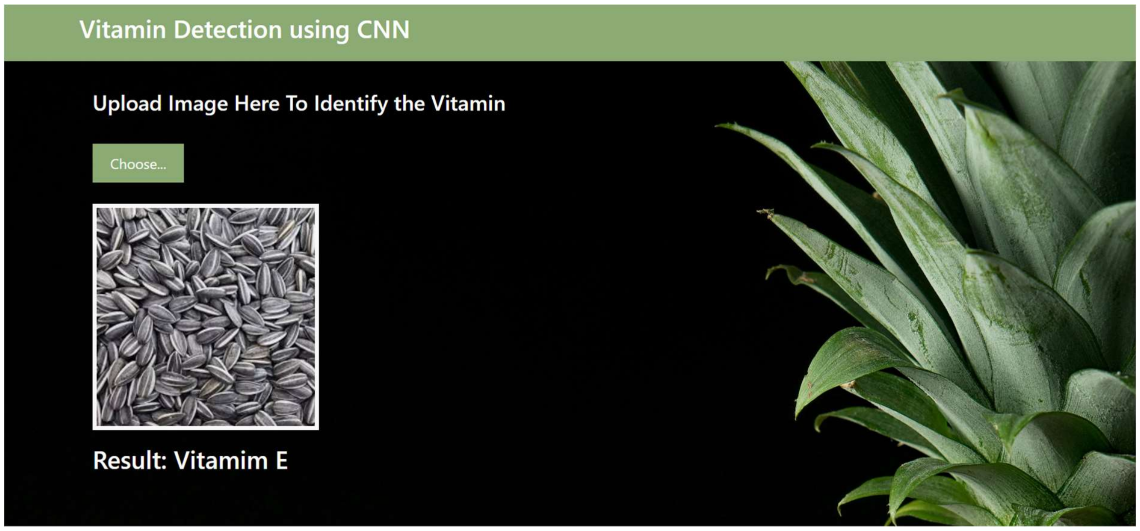
Upload Image Here To Identify the Vitamin



Upload Image Here To Identify the Vitamin



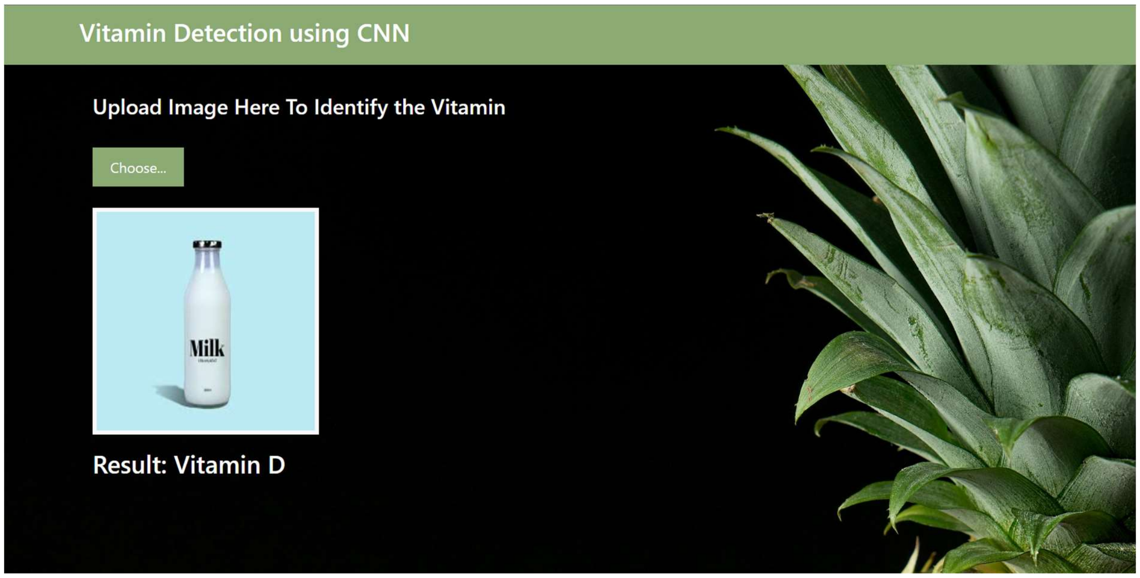
Result: Vitamin C



Vitamin Detection using CNN

Upload Image Here To Identify the Vitamin

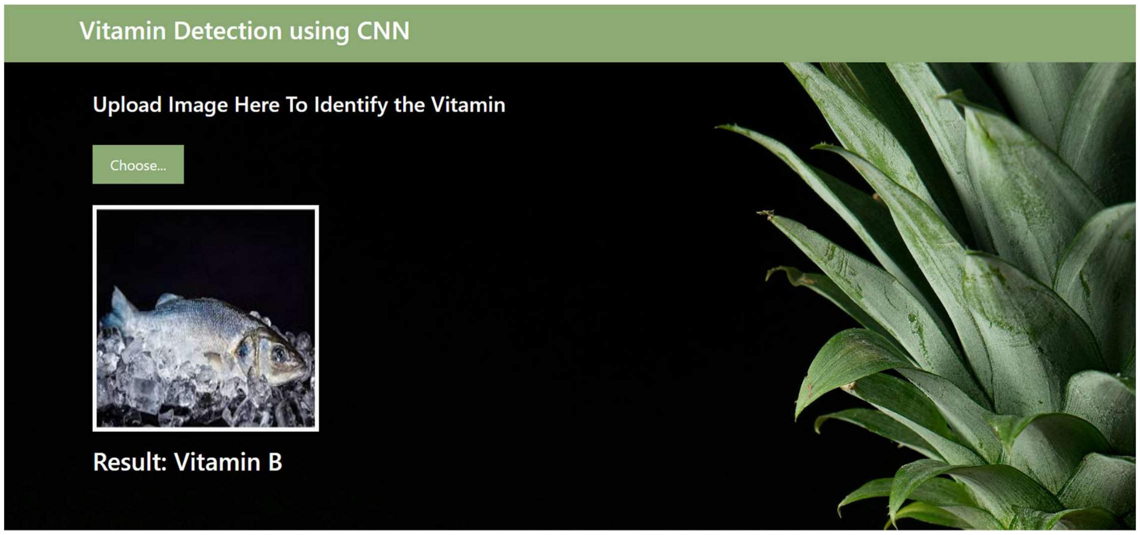
Result: Vitamim E



Vitamin Detection using CNN

Upload Image Here To Identify the Vitamin

Result: Vitamin D



Vitamin Detection using CNN

Upload Image Here To Identify the Vitamin

Result: Vitamin B