from keras.preprocessing.image import ImageDataGenerator from google.colab import drive drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force\_remount=True). Image Data Augmentation

train\_datagen = ImageDataGenerator(rescale=1./255,shear\_range=0.2,zoom\_range=0.2,horizontal\_flip=True) test\_datagen=ImageDataGenerator(rescale=1./255) Loading Data and performing Data Augmentation

x\_train=train\_datagen.flow\_from\_directory(r'/content/drive/MyDrive/ibm-nutritionanalyser/TRAIN\_SET',target\_size=(64,64),batch\_size=32,class\_mode='sparse') x\_test=train\_datagen.flow\_from\_directory(r'/content/drive/MyDrive/ibm-nutritionanalyser/TEST\_SET',target\_size=(64,64),batch\_size=32,class\_mode='sparse') Found 4119 images belonging to 5 classes.

Found 929 images belonging to 5 classes. print(x\_train.class\_indices)

{'APPLES': 0, 'BANANA': 1, 'ORANGE': 2, 'PINEAPPLE': 3, 'WATERMELON': 4} print(x\_test.class\_indices)

{'APPLES': 0, 'BANANA': 1, 'ORANGE': 2, 'PINEAPPLE': 3, 'WATERMELON': 4} from collections import Counter as c

c(x\_train .labels)

Counter({0: 996, 1: 1354, 2: 1019, 3: 275, 4: 475}) Importing necessasry library

import numpy as np#used for numerical analysis

import tensorflow #open source used for both ML and DL for computation from tensorflow.keras.models import Sequential #it is a plain stack of layers

from tensorflow.keras import layers #A layer consists of a tensor-in tensor-out computation function

#Dense layer is the regular deeply connected neural network layer from tensorflow.keras.layers import Dense,Flatten

#Faltten-used fot flattening the input or change the dimension

from tensorflow.keras.layers import Conv2D,MaxPooling2D,Dropout #Convolutional layer

#MaxPooling2D-for downsampling the image

from keras.preprocessing.image import ImageDataGenerator

Initializing The Model model = Sequential() Creating the model # Initializing the CNN

classifier = Sequential()

# First convolution layer and pooling

classifier.add(Conv2D(32, (3, 3), input\_shape=(64, 64, 3), activation='relu')) classifier.add(MaxPooling2D(pool\_size=(2, 2)))

# Second convolution layer and pooling

classifier.add(Conv2D(32, (3, 3), activation='relu'))

# input\_shape is going to be the pooled feature maps from the previous convolution layer classifier.add(MaxPooling2D(pool\_size=(2, 2)))

# Flattening the layers

classifier.add(Flatten())

# Adding a fully connected layer

classifier.add(Dense(units=128, activation='relu'))

classifier.add(Dense(units=5, activation='softmax')) # softmax for more than 2

classifier.summary()

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

================================================================= conv2d (Conv2D) (None, 62, 62, 32) 896

max\_pooling2d (MaxPooling2D (None, 31, 31, 32) 0

)

conv2d\_1 (Conv2D) (None, 29, 29, 32) 9248

max\_pooling2d\_1 (MaxPooling (None, 14, 14, 32) 0

2D)

flatten (Flatten) (None, 6272) 0

dense (Dense) (None, 128) 802944

dense\_1 (Dense) (None, 5) 645

=================================================================

Total params: 813,733

Trainable params: 813,733

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Compiling the model

classifier.compile(optimizer='adam', loss='sparse\_categorical\_crossentropy', metrics=['accuracy']) Fitting the model classifier.fit\_generator(

generator=x\_train,steps\_per\_epoch = len(x\_train),

epochs=10, validation\_data=x\_test,validation\_steps = len(x\_test))

/usr/local/lib/python3.7/dist-packages/ipykernel\_launcher.py:3: UserWarning: `Model.fit\_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators. This is separate from the ipykernel package so we can avoid doing imports until Epoch 1/10

129/129 [==============================] - 1192s 9s/step - loss: 0.6981 - accuracy: 0.7312 - val\_loss: 0.6113 - val\_accuracy: 0.7470

Epoch 2/10

129/129 [==============================] - 39s 300ms/step - loss: 0.4519 - accuracy: 0.8267 - val\_loss: 0.5630 - val\_accuracy: 0.7761

Epoch 3/10

129/129 [==============================] - 37s 286ms/step - loss: 0.3904 - accuracy: 0.8536 - val\_loss: 0.4508 - val\_accuracy: 0.8224

Epoch 4/10

129/129 [==============================] - 37s 286ms/step - loss: 0.3631 - accuracy: 0.8653 - val\_loss: 0.4773 - val\_accuracy: 0.8181

Epoch 5/10

129/129 [==============================] - 37s 289ms/step - loss: 0.3238 - accuracy: 0.8755 - val\_loss: 0.4213 - val\_accuracy: 0.8407

Epoch 6/10

129/129 [==============================] - 38s 294ms/step - loss: 0.3063 - accuracy: 0.8844 - val\_loss: 0.3872 - val\_accuracy: 0.8558

Epoch 7/10

129/129 [==============================] - 39s 304ms/step - loss: 0.2774 - accuracy: 0.8934 - val\_loss: 0.3918 - val\_accuracy: 0.8579

Epoch 8/10

129/129 [==============================] - 37s 286ms/step - loss: 0.2752 - accuracy: 0.8937 - val\_loss: 0.4671 - val\_accuracy: 0.8256

Epoch 9/10

129/129 [==============================] - 37s 288ms/step - loss: 0.2678 - accuracy: 0.8992 - val\_loss: 0.3788 - val\_accuracy: 0.8515

Epoch 10/10

129/129 [==============================] - 37s 287ms/step - loss: 0.2651 - accuracy: 0.8980 - val\_loss: 0.4373 - val\_accuracy: 0.8310 Saving the model

classifier.save('nutrition.h5') Testing the Model

from tensorflow.keras.models import load\_model from tensorflow.keras.preprocessing import image import numpy as np

img = image.load\_img("/content/drive/MyDrive/ibm-nutritionanalyser/TEST\_SET/APPLES/n07740461\_9461.jpg",target\_size= (64,64)) x=image.img\_to\_array(img) x=np.expand\_dims(x,axis=0) pred = classifier.predict(x)

pred

1/1 [==============================] - 0s 99ms/step

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

index=['APPLES', 'BANANA', 'ORANGE','PINEAPPLE','WATERMELON'] index[np.argmax(pred)]

'APPLES' from flask import Flask,render\_template,request

# Flask-It is our framework which we are going to use to run/serve our application. #request-for accessing file which was uploaded by the user on our application. import os

import numpy as np #used for numerical analysis

from tensorflow.keras.models import load\_model #to load our trained model from tensorflow.keras.preprocessing import image import requests

app = Flask(\_\_name\_\_,template\_folder="templates") #initializing a flask app

# Loading the model model=load\_model('nutrition.h5') print("Loaded model from disk")

Loaded model from disk

@ app.route('/')# route to display the home page def home(): return render\_template('home.html')

@ app.route('/image1', methods=['GET', 'POST']) # routes to the index html def image1(): return render\_template("image.html")

@ app.route('/predict' ,methods=['GET','POST']) # route to show the predictions in a Web UI def lanuch(): if request.method=='POST':

f=request.files['file'] # requesting the file

basepath=os.path.dirname('\_\_file\_\_') #storing the file directory

filepath=os.path.join(basepath,"uploads",f.filename) #storing the file in uploads folder f.save(filepath) #saving the file

img=image.load\_img(filepath,target\_size=(64,64)) #load and reshaping the image x=image.img\_to\_array(img) #converting image to an array x=np.expand\_dims(x,axis=0) #changing the dimensions of the image

pred=np.argmax(model.predict(x), axis=1)

print("prediction",pred) #printing the prediction

index=['APPLE','BANANA','ORANGE','PINEAPPLE','WATERMELON']

result=str(index[pred[0]])

print(result) x=result

result=nutrition(result)

print(result)

return render\_template("0.html",showcase=(result),showcase1=(x)) def nutrition(index):

import requests

url = "https://calorieninjas.p.rapidapi.com/v1/nutrition"

querystring = {"query":index}

headers = {

"X-RapidAPI-Key": "85887549f4msh51e7315b280a87ep1f43e0jsn585c940f2ea6",

"X-RapidAPI-Host": "calorieninjas.p.rapidapi.com"

}

response = requests.request("GET", url, headers=headers, params=querystring)

print(response.text) return response.json()['items'] if \_\_name\_\_ == "\_\_main\_\_": # running the app app.run(debug=False)

* Serving Flask app "\_\_main\_\_" (lazy loading)
* Environment: production

WARNING: This is a development server. Do not use it in a production deployment.

Use a production WSGI server instead.

* Debug mode: off

INFO:werkzeug: \* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)