```
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-nutrition-
analyser/TRAIN_SET',target_size=(64,64),batch_size=32,class_mode='sparse')
x_test=train_datagen.flow_from_directory(r'/content/drive/MyDrive/ibm-nutrition-
analyser/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-nutrition-
analyser/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)
```