Ввод [1]:

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import backend as K
from tensorflow.keras.layers import Dense, Activation,Dropout,Conv2D, MaxPooling2D,
from tensorflow.keras.optimizers import Adam, Adamax
from tensorflow.keras.metrics import categorical_crossentropy
from tensorflow.keras import regularizers
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model, load_model, Sequential
import numpy as np
import pandas as pd
import shutil
import time
import cv2 as cv2
from tqdm import tqdm
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
from matplotlib.pyplot import imshow
import seaborn as sns
sns.set_style('darkgrid')
from PIL import Image
from sklearn.metrics import confusion_matrix, classification_report
from IPython.core.display import display, HTML
# stop annoying tensorflow warning messages
import logging
logging.getLogger("tensorflow").setLevel(logging.ERROR)
print ('modules loaded')
```

```
modules loaded
```

Ввод [4]:

```
def predictor(sdir, csv_path,  model_path, averaged=True, verbose=True):
    # read in the csv file
    class_df=pd.read_csv(csv_path)
    class_count=len(class_df['class'].unique())
    img_height=int(class_df['height'].iloc[0])
    img_width =int(class_df['width'].iloc[0])
    img_size=(img_width, img_height)
    scale=class_df['scale by'].iloc[0]
    image_list=[]
    # determine value to scale image pixels by
    try:
        s=int(scale)
        s2=1
        s1=0
    except:
        split=scale.split('-')
        s1=float(split[1])
        s2=float(split[0].split('*')[1])
    path_list=[]
    paths=os.listdir(sdir)
    for f in paths:
        path_list.append(os.path.join(sdir,f))
    if verbose:
        print (' Model is being loaded- this will take about 10 seconds')
    model=load_model(model_path)
    image_count=len(path_list)
    image_list=[]
    file_list=[]
    good_image_count=0
    for i in range (image_count):
        try:
            img=cv2.imread(path_list[i])
            img=cv2.resize(img, img_size)
            img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
            good_image_count +=1
            img=img*s2 - s1
            image_list.append(img)
            file_name=os.path.split(path_list[i])[1]
            file_list.append(file_name)
        except:
            if verbose:
                print ( path_list[i], ' is an invalid image file')
    if good_image_count==1: # if only a single image need to expand dimensions
        averaged=True
    image_array=np.array(image_list)
    # make predictions on images, sum the probabilities of each class then find cla
    # highest probability
    preds=model.predict(image_array)
    if averaged:
        psum=[]
        for i in range (class_count): # create all 0 values list
            psum.append(0)
        for p in preds: # iterate over all predictions
            for i in range (class_count):
                psum[i]=psum[i] + p[i]  # sum the probabilities
        index=np.argmax(psum) # find the class index with the highest probability s
        klass=class_df['class'].iloc[index] # get the class name that corresponds t
        prob=psum[index]/good_image_count  # get the probability average
        # to show the correct image run predict again and select first image that h
```

```
        for img in image_array:  #iterate through the images
            test_img=np.expand_dims(img, axis=0) # since it is a single image expan
            test_index=np.argmax(model.predict(test_img)) # for this image find the
            if test_index== index: # see if this image has the same index as was se
                if verbose: # show image and print result if verbose=1
                    plt.axis('off')
                    plt.imshow(img) # show the image
                    print (f'predicted species is {klass} with a probability of {pr
                break # found an image that represents the predicted class
        return klass, prob, img, None
    else: # create individual predictions for each image
        pred_class=[]
        prob_list=[]
        for i, p in enumerate(preds):
            index=np.argmax(p) # find the class index with the highest probability
            klass=class_df['class'].iloc[index] # get the class name that correspon
            image_file= file_list[i]
            pred_class.append(klass)
            prob_list.append(p[index])
        Fseries=pd.Series(file_list, name='image file')
        Lseries=pd.Series(pred_class, name= 'species')
        Pseries=pd.Series(prob_list, name='probability')
        df=pd.concat([Fseries, Lseries, Pseries], axis=1)
        if verbose:
            length= len(df)
            print (df.head(length))
        return None, None, None, df

def print_in_color(txt_msg,fore_tupple,back_tupple,):
    #prints the text_msg in the foreground color specified by fore_tupple with the
    #text_msg is the text, fore_tupple is foregroud color tupple (r,g,b), back_tupp
    rf,gf,bf=fore_tupple
    rb,gb,bb=back_tupple
    msg='{0}' + txt_msg
    mat='\33[38;2;' + str(rf) +';' + str(gf) + ';' + str(bf) + ';48;2;' + str(rb) +
    print(msg .format(mat), flush=True)
    print('\33[0m', flush=True) # returns default print color to back to black
    return
```

Ввод [2]:

```
csv_file='class_dict.csv'
model_file='EfficientNetB3-fruits-100.0.h5'
store_path='storage'
```

Ввод [25]:

```
csv_path=csv_file # path to class_dict.csv
model_path=model_file # path to the trained model
klass, prob, img, df =predictor(store_path, csv_path,  model_path, averaged=True, v
msg=f' image is of  {klass}  with a probability of {prob * 100: 6.2f} %'
plt.axis('off')
plt.imshow(img)
print_in_color(msg, (0,255,255), (65,85,55))
```

```
 image is of  orange  with a probability of  85.04 %
```

Ввод [ ]:

```

```