Fruit & vegetables image Détection and Classification with YoloV3

The purpose of this model is to detect and classify images of fruits and vegetables according to their variety : apple, banana, avocado, etc.

This model is based on the OpenImage Dataset.

This dataset contains ~9 million varied images with rich annotations. The images are very diverse and often contain complex scenes with several objects (8.4 per image on average). It contains image-level labels annotations, object bounding boxes, object segmentations, visual relationships, localized narratives, and more

We focused on 18 categories of fruits and vegetables images for our Detection model. Those images contain multiple objects and are able to detect if one of those 18 type of fruits or vegetables are represented in the image.

We created this model using Darknet with YoloV3.

Import libraries

In [ ]:
```
import os
import pandas as pd
import numpy as np
```

Preprocessing

1.1 Download classes names from VGG16-Fruits360

In [ ]:
```
fileyoloclasses = open("Documents\\Projet\\yolo18classes.txt","r+")
content_list = fileyoloclasses.readlines()
print(content_list)
type(content_list)
```

```
['Tomato Cucumber Pomegranate Potato Banana Grapefruit Strawberry Mango Watermelon Cantaloupe Orange Peach Pear Grape Lemon Apple Bell_pepper Common_fig']
```

Out[ ]:
```
list
```

1.2 Download custom OpenImage Dataset using OIDv4_ToolKit-master

- **Env Config**
  - Opencv
  - Cuda
  - Darknet
- **Download Toolkit**
  - conda activate env_name
  - git clone https://github.com/EscVM/OIDv4_ToolKit.git
  - cd "OIDv4_ToolKit-master" folder
  - pip install -r requirements.txt
- **Download Dataset**:
  - In Linux/Unix:

    * value=`cat ../yolo18classes.txt`
    * TRAIN: python main.py downloader --classes $value --type_csv train --multiclasses 1
    * VALIDATION: python main.py downloader --classes $value --type_csv validation --multiclasses 1
    * TEST: python main.py downloader --classes $value --type_csv test --multiclasses 1
  - In Windows:

    * TRAIN: python main.py downloader --classes Tomato Cucumber Pomegranate Potato Banana Grapefruit Strawberry Mango Watermelon Cantaloupe Orange Peach Pear Grape Lemon Apple Bell_pepper Common_fig --type_csv train --multiclasses 1

    * VALIDATION: python main.py downloader --classes Tomato Cucumber Pomegranate Potato Banana Grapefruit Strawberry Mango Watermelon Cantaloupe Orange Peach Pear Grape Lemon Apple Bell_pepper Common_fig --type_csv validation --multiclasses 1

    * TEST: python main.py downloader --classes Tomato Cucumber Pomegranate Potato Banana Grapefruit Strawberry Mango Watermelon Cantaloupe Orange Peach Pear Grape Lemon Apple Bell_pepper Common_fig --type_csv test --multiclasses 1

1.3 Annotation - Convert to Yolo Format

* Create a .txt with same file name as image file: (watermelon01.jpg vs watermelon01.txt)

* Structure of this .txt file will be:

    * class name
    * center x : x coordinate of the center of the bounding box
    * center y : y coordinate of the center of the bounding box
    * width of the bounding box
    * height of the bounding box

In [ ]:
```
base_dir = 'Documents\\Projet\\OIDv4_ToolKit\\OID'
fruit_labels = ['Tomato', 'Cucumber', 'Pomegranate', 'Potato', 'Banana', 'Grapefruit', 'Strawberry', 'Mango', 'Watermelon', 'Cantaloupe', 'Orange', 'Peach', 'Pear', 'Grape', 'Lemon', 'Apple', 'Bell pepper', 'Common fig']
print(len(fruit_labels))
```

```
18
```

In [ ]:
```
#Create a function to read Annotation csv files from the directory and save them into txt format (Yolo format)
def convert_to_yolo():
    full_path_to_csv = 'OIDv4_ToolKit\\OID\\csv_folder'

    full_path_to_images = \
        'OIDv4_ToolKit\\OID\\Dataset\\test\\18fruits'

    #List of classes
    labels = fruit_labels

    classes = pd.read_csv(full_path_to_csv + '\\' + 'class-descriptions-boxable.csv',
                          usecols=[0, 1], header=None)


    encrypted_strings = []

    # Get encrypted string for every class
    # Go through all labels
    for v in labels:
        # Get Pandas sub-dataFrame that has only one row
        # By using 'loc' method we locate the needed row, that satisfies condition 'classes[1] == v', that can be found from the 1st column element (equal to v)
        sub_classes = classes.loc[classes[1] == v]
        # Print(sub_classes)  # 570  /m/0k4j  Car

        # Get element from the first row and first column
        e = sub_classes.iloc[0][0]
        # Print(e)  # /m/0k4j

        # Append found encrypted string into the list
        encrypted_strings.append(e)


    # Read csv file with annotations
    # Load only needed columns into Pandas dataFrame

    annotations = pd.read_csv(full_path_to_csv + '\\' + 'test-annotations-bbox.csv',
                              usecols=['ImageID',
                                       'LabelName',
                                       'XMin',
                                       'XMax',
                                       'YMin',
                                       'YMax'])

    # Check point
    # Show first 5 rows from the dataFrame
    # Print(annotations.head())

    # Get Pandas dataFrame that has only needed rows
    # By using 'loc' method we locate needed rows, that only have needed 'encrypted_strings'
    # By using copy() we create a separate new dataFrame ; in this way, the initial dataFrame will not be modified
    sub_ann = annotations.loc[annotations['LabelName'].isin(encrypted_strings)].copy()


    # Add new empty columns to dataFrame to save numbers for YOLO format
    sub_ann['classNumber'] = ''
    sub_ann['center x'] = ''
    sub_ann['center y'] = ''
    sub_ann['width'] = ''
    sub_ann['height'] = ''

    # Go through all encrypted classes strings, and convert them to numbers, according to the order they are in the list
    for i in range(len(encrypted_strings)):
        # Write numbers into the appropriate column
        sub_ann.loc[sub_ann['LabelName'] == encrypted_strings[i], 'classNumber'] = i

    # Calcule bounding box's center in x and y for all rows, using XMax and XMin
    # Save results in the appropriate columns
    sub_ann['center x'] = (sub_ann['XMax'] + sub_ann['XMin']) / 2
    sub_ann['center y'] = (sub_ann['YMax'] + sub_ann['YMin']) / 2

    # Calculate bounding box's width and height for all rows
    # Save results in the appropriate columns
    sub_ann['width'] = sub_ann['XMax'] - sub_ann['XMin']
    sub_ann['height'] = sub_ann['YMax'] - sub_ann['YMin']

    # Get Pandas dataFrame that has only needed columns
    # By using 'loc' method we locate here all rows, but only specified columns
    # By using copy() we create separate dataFrame, not just a reference to the previous one
    # In this way, the initial dataFrame will not be modified
    r = sub_ann.loc[:, ['ImageID',
                        'classNumber',
                        'center x',
                        'center y',
                        'width',
                        'height']].copy()

    # Change the current directory to the one with the images
    os.chdir(full_path_to_images)

    # Check point
    # Get the current directory
    # Print(os.getcwd())

    # Use os.walk for going through all directories
    # and files in them from the current directory
    # Fullstop in os.walk('.') means it is the current directory
    for current_dir, dirs, files in os.walk('.'):
        # Go through all files
        for f in files:
            # Make sure the filenames end with '.jpg'
            if f.endswith('.jpg'):
                # Slice only name of the file without extension
                image_name = f[:-4]
                # Get Pandas dataFrame that has only needed rows
                # By using 'loc' method we locate the needed rows, that satisfies condition 'classes['ImageID'] == image_name', that can be found in the 1st column element (equal to image_name)
                sub_r = r.loc[r['ImageID'] == image_name]

                # Get resulted Pandas dataFrame that has only needed columns
                # By using 'loc' method we locate here all rows, but only specified columns
                # By using copy() we create a separate new dataFrame ; in this way, the initial dataFrame will not be modified
                resulted_frame = sub_r.loc[:, ['classNumber',
                                               'center x',
                                               'center y',
                                               'width',
                                               'height']].copy()

                # Prepare path where to save txt file
                path_to_save = full_path_to_images + '\\' + image_name + '.txt'

                # Save resulted Pandas dataFrame into txt file
                resulted_frame.to_csv(path_to_save, header=False, index=False, sep=' ')
```

1.4 Train Test Split (85% vs 15 %)

* Create train.txt & test.txt in image folder
* Create classes.names (same content as classes.txt)
* Create fruits_data.data (This is for adding into darknet/cfg folder for training later)

In [ ]:
```
split_train_test()
```

1.5 Preparing for Darknet Training

* Download Darknet project and make / cmake project
* Go into dakrnetfolder, open git bash

* Create two files names yolov3_fruits18_train.cfg & yolov3_fruits18_test.cfg in cfg folder
* Move fruits_data.data into darknet.cfg folder

Downoald weights for convolutional layers and train model

* Run the command ./darknet detector train cfg/fruits_data.data cfg/yolov3_fruits18_train.cfg weights/darknet53.conv.74 -don't_show
* Weights will be stored in backup folder
* Run the command ./darknet detector test cfg/fruits_data.data cfg/yolov3_fruits18_test.cfg backup/yolov3_fruits18_train_final.weights -dont_show
* ./darknet detect backup/yolov3_fruits18_train_final.weights data/FruitsTestimage/18fruits/0a891eacc732d25.jpg