End-To-End walk through and explanation of jsonToCsv.py

Produced by: Alan Feria & Dr. Alexander White '22

alan214@ucsb.edu

Table of Contents

About This File

This file was created for the benefit of all teachers and students wanting to use the work located on (link to SCI github)

SECTION 1

What is COCO

1.1 A Brief Introduction to the Coco Format

The Common Object in Context (COCO) is one of the most popular large-scale labeled image datasets available for public use. COCO is an image dataset that has been developed by Microsoft and is used for object detection (draw boxes around certain objects in an image), segmentation (label every pixel in an image as some object or background), keypoint detection (place points on human joints), and captioning (produce sentences to describe an image). When you hear about a "custom" COCO dataset or COCO formatted dataset, it just means that you will be working with a dataset that follows the same label scheme as COCO. It's useful to follow their example because a lot of tools will let you automatically load your dataset if it's in that format. For example, you can see what a COCO formatted data set for object detection format looks like by downloading here or on our GitHub page

SECTION 2

Restructuring a COCO dataset

If we want to work with this dataset, we need to restructure the data (images) and sort the images by class. To achieve this, I have created two executables that accomplish exactly that task. The first executable is jsonToCsv.py; within this file, we use modules JSON, CSV, os, pandas, & the search() class from the re module to convert the JSON data structure to a .csv & .txt data structure. Please see section 3 for a breakdown of each class and module used. It is important to have a txt file with the names of the files so that we can move and rename our images using trainMove.sh

In this section, I will give an end-to-end logical breakdown of what is happening within jsonToCsv.py executable.

2.1 Importing Classes and Modules

As in any Python executable, we start off by importing the required modules and libraries with their proper abbreviations and selected classes.

2.2 Setting Working Directory (workDir)

Since we want to generalize this .py file, it is best practice to ask the user to input the working/-parent directory where the JSON file is located. This helps avoid possible human error when altering the code. This will be line 18 (workDir).

2.3 Validating Working Directory (workDir)

Validating workDir is a two-step process. The first process is having the user itself check their spelling and file path. If the user says it's incorrect, they are prompted to re-enter the file path. If the user states that what they have entered is correct, it passes the second stage test. This test is done by os.path.exits(workDir). If this returns true, then the file path exists and is set as the working file. The user is asked to start over and enter a new file path if this returns false. This process will repeat until both conditions are satisfied.

2.4 Creating Folder to Store Files

As a best practice, we will create a folder called "/CSVfiles" that will store all of our CSV files created. This will be done using os.makedirs().

2.5 Converting to CSV

We will be creating three separate CSV files for the categories of interest (cats). In order to create these, we will iterate over cats and open each category as f and load in the JSON object (f) using json.load(). We will then slice the first category of interest. Once done, we create a new CSV file within the CSVfiles folder names channel_islands_camera_traps_(cats).csv

Once we create the **empty** .csv file, we will want to the specified data onto it. We will start by specifying the field names (column names) using the .writeheader() class. Once the field names have been specified we can fill each column with its respective data. To accomplish this, we will use the .writerow() class over each column present.

2.6 Slicing and Selecting Columns by Class

Now that we have created CSV's for the columns of interest, we can we read them in as pandas data frames using the read_csv() class. Now that we have read the CSV files, we want to rename them to annotation_id and image_id to id within the annotations CSV. Now that we have renamed the columns, we can inner join the images and annotations on 'id' and create a parent table to slice columns from. Now it is good practice to do a shape check of the parent table and see if any unexpected columns or rows have been dropped, but as expected, the table retained the correct shape attributes in this case.

Once we create the parent table, we must use .loc[ ] to slice each category individually; once done, we use .to_csv() to export these new data frames as CSV files and save them to the /fileByCat folder. Lastly, we will want to zip these files and save them to your local storage on your computer (only necessary if running this .py on Google Colab)

3

SECTION

Understanding Our Work

In summary, jsonToCsv.py is an executable python script that takes in the relative path of the channel_islands_camera_traps_.json and converts it the to three separate CSV files which contain the categories of interest, that being annotations, images, and categories. These three CSV files will also be saved into a subdirectory folder called CSVfiles placed within the parent directory (input by the user).

3.1 channel_islands_camera_traps_annotation.csv

The annotations .csv file will store attributes of the image such as ID, image_id, category_id, sequence_level, & bbox.

- ID is a unique identifier throughout the entire dataset and is also a superkey.
- image_id is a foreign key within annotations and the primary key to images.id.
- category_id is a category given by the classification method used in a numerical format.
- sequence_id is unknown.
- Bounding boxes (bbox) are an image annotation method to train AI-based machine learning models.

3.2 channel_islands_camera_traps_images.csv

The image .csv file will store metadata about each image, such as locations, temperature, seq_id, and more.

- ID is a unique identifier throughout the entire dataset and is also a superkey.
- file_name is equivalent to the path from the parent directory to the file location. The last subdirectory is the actual image name, while the rest are subdirectories.
- sed_id is shared by n number of amount of images where n is equal to seq_num_frame
- frame_num is the Frame number in a specific sequence_id
- original_relative_path is the original path from where the file was collected. This could have been a directory on someone's local computer or hard drive.
- location is the first subdirectory that is needed to locate the image.
- temperature is the temperature at the time that the image was taken.
- width and height are the sizes of the image.

3.3 channel_islands_camera_traps_catagories.csv

The categories .csv file will store a list of all the possible categories present in the data package.

- Unique identifier for each class present throughout the dataset. Different from image.id and annotation.id.
- name is a comprehensible name for each respective id that is readable by humans.

Example work can be found here

4

SECTION

Data Scientist Tool Box - Python

4.1 json Module

In Python, the json module provides an API similar to converting in-memory Python objects to a serialized representation known as JavaScript Object Notation (JSON) and vice-versa.

4.1.1 json.load()

By default takes json.load() as a file object and returns the JSON object. A JSON object contains data in the form of key/value pair. The keys are strings, and the values are the JSON types. Keys and values are separated by a colon. Each entry (key/value pair) is separated by a comma.

4.2 csv Module

CSV (Comma Separated Values) is a simple file format used to store tabular data, such as a spreadsheet. A CSV file stores tabular data (numbers & text) in plain text. Each line of the file is an entry (a record), and each entry consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format, but we can have custom separators such as a tab or '=.' However, there are more purposed for the CSV than just importing and reading tabular data.

4.2.1 csv.DictWriter()

The csv.DictWriter() class operates like a regular writer but maps Python dictionaries into CSV rows. The fieldnames parameter is a sequence of keys that identify the order in which values in

the dictionary passed to the writerow method are written to the CSV file.

4.3 Pandas

Pandas is a Python package that provides fast, flexible, and expressive data structures designed to make working with “relational” or “labeled” data easy and intuitive. It aims to be the fundamental high-level building block for doing practical, real-world data analysis.

4.3.1 .to_csv()

By default, the .to_csv() method exports DataFrame to a CSV file with row index as the first column and comma as the delimiter. The delimiter can be specified to any custom value.

4.3.2 .read_csv()

To access data from the CSV file, we require a function read_csv() that retrieves data in the form of the Dataframe. When using read_csv() you can specify what delimiter should be used to read the CSV. In our case, we will be reading envVars.csv with ’=’ as the delimiter.

4.3.3 .merge()

The merge() method updates the content of two DataFrame by merging them together using the specified method(s). see more here

4.3.4 .loc[]

Pandas DataFrame.loc attribute accesses a group of rows and columns by label(s) or a boolean array in the given DataFrame.

4.4 os Module

The OS module in Python provides functions for interacting with the operating system. This module provides a portable way of using operating system-dependent functionality.

4.4.1 os.makedirs()

By default os.makedirs() method in Python is used to create a directory recursively. That means while making a leaf directory, if any intermediate-level directory is missing, os.makedirs() method will create them all.

4.4.2 os.path.exists()

os.path.exists() method in Python is used to check whether the specified path exists or not. This method can also be used to check whether or not the given path refers to an open file descriptor.

4.5 re Module

This module provides regular expression matching operations similar to those found in Perl. A regular expression is a special sequence of characters that helps you match or find other strings or sets of strings using a specialized syntax held in a pattern.

4.5.1 re.search()

re.search() function will search the regular expression pattern and return the first occurrence. re.search() function takes the “pattern” and “text” to scan from our main string. In our case, we will be searching for possible paths to the envVars.csv file.

5

SECTION

jsonToCsv.py

5.1 Source Code

```
# -*- coding: utf-8 -*-
"""file select

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1zZoEsv0vdQ_-hSVV-6yFqKB-oYbonEic

"""

import json, csv, os
import pandas as pd

cats = ["images", "annotations", "categories"] #catagories of intrest

os.makedirs('/content/CSVfiles', exist_ok=True)

for name in cats:
  with open("/content/channel_islands_camera_traps.json", "r") as f:
      #insert path to channel_islands_camera_traps.json file
    data = json.load(f)
```

```
    names = data[name]

  with open("/content/CSVfiles/channel_islands_camera_traps_" + name +
      ".csv", "w") as p:
    fieldnames = names[0].keys()
    writer = csv.DictWriter(p,fieldnames=fieldnames)
    writer.writeheader()
    for i in names:
      writer.writerow(i)

annotations =
    pd.read_csv("/content/CSVfiles/channel_islands_camera_traps_annotations.csv")
images =
    pd.read_csv("/content/CSVfiles/channel_islands_camera_traps_images.csv")
catagories =
    pd.read_csv('/content/CSVfiles/channel_islands_camera_traps_categories.csv')

images.columns

images.shape

annotations.columns

annotations.shape

annotations.rename(columns={"id":"annotation_id",
                            "image_id":"id"}, inplace = True)
parentTable = pd.merge(images, annotations,
        how='inner', on='id')

"""sanity check """

parentTable.columns

parentTable.columns

os.makedirs('/content/fileByCat', exist_ok=True)

empty = parentTable.loc[parentTable['category_id'] == 0].file_name
empty.to_csv("/content/fileByCat/emptyPath.txt", header=False, index =
    False)


human = parentTable.loc[parentTable['category_id'] == 1].file_name
```

```
human.to_csv("/content/fileByCat/humanPath.txt", header=False, index =
    False)

fox = parentTable.loc[parentTable['category_id'] == 2].file_name
fox.to_csv("/content/fileByCat/foxPath.txt", header=False, index = False)


skunk = parentTable.loc[parentTable['category_id'] == 3].file_name
skunk.to_csv("/content/fileByCat/skunkPath.txt", header=False, index =
    False)


rodent = parentTable.loc[parentTable['category_id'] == 4].file_name
rodent.to_csv("/content/fileByCat/rodentPath.txt", header=False, index =
    False)


bird = parentTable.loc[parentTable['category_id'] == 5].file_name
bird.to_csv("/content/fileByCat/birdPath.txt", header=False, index = False)

other = parentTable.loc[parentTable['category_id'] == 6].file_name
other.to_csv("/content/fileByCat/otherPath.txt", header=False, index =
    False)

!zip -r /content/fileByCat.zip /content/fileByCat

#Manually download fileByCat.zip from within /content section

!zip -r /content/CSVfiles.zip /content/CSVfiles

#Manually download CSVfiles.zip from within /content section
```