A Machine Learning Workflow on the Santa Cruz Island Spotted Skunk Project

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 here.

SECTION 1

Background & Purpose

1.1 Background

The island spotted skunk (Spilogale gracilis amphialus), endemic to Santa Cruz Island (SCI) in Southern California, has been the subject of study by student researchers in the University of California Santa Barbara-Smithsonian Scholars Program since 2017. This student-led project utilizes infrared camera traps positioned across the island to capture images of these skunks to monitor the population and determine if the island spotted skunk is endangered. The ecology of the island spotted skunk has been historically understudied, so this project is the first of its kind. Over the course of the project, nearly 1 million individual images have captured life on Santa Cruz Island. To date, data analysis has been impossible because a researcher would need to look through and classify tens or hundreds of thousands of images each year to observe any information about skunk presence or absence. While this task is essentially impossible for a human to perform, a machine learning (ML) model can complete this task efficiently and accurately. It does so by identifying the presence or absence of skunks using statistical and mathematical techniques in a specific and unique order to recognize color patterns in images.

In collaboration with the Smithsonian Institution's Data Science Lab, our research team developed a plan to implement computer vision machine learning techniques for data analysis. Specifically, we are using Efficientnet_lite3, an architecture designed to run on edge devices. This model was trained using labeled images from the Labeled Information Library of Alexandria: Biology and Conservation (LILS BC), which includes over 240,00 images across seven different classes. Integrating machine learning techniques will be a huge milestone, but we will finally be able to find answers to some of the project's core research questions.

1.2 Purpose

This document serves as a procedural guide for various processes, detailing explicitly how tasks have been accomplished and how to navigate the current model and data collection system so future interns and users of this software can understand and navigate it with ease.

One can expect the following points will be covered:

- **COCO Camera Trap Format and Description**: A brief coverage and explanation of a COCO formatted data set and the formatting of our data set.
- **Sorting a COCO Formatted**: The steps we took to sort the LILA BC Data Set using python frameworks.
- **Machine Learning**: A brief explanation to machine learning and the 'efficientnet_lite3' architecture that was used to create our custom trained model.
- **Model Training**: How to trained a model using TensorFlow lite and its dependencies.
- **Model Deployment**: The required steps to deploy a custom trained TensorFlow Lite model on a Raspberry Pi 400.
- **Formatting Data**: The steps taken to transform our raw data to a common format known as Camera Trap Data Package using python frameworks.

SECTION 2

ML Workflow

It is crucial that we fully understand what the Camera Trap Data Package entails and what necessary measures must be taken to follow that format, as we will be using this structure to tell our story and to guide and explain the decisions that were made.

reference (camera trap data package explanation)

2.1 Data Acquisition and Cleaning

Having established a method of communicating data, we must gather and scrape the relevant data necessary for the project. We will be utilizing machine learning to classify and identify the large number of images that we have collected over the years; as we know, machine learning depends on labeled data, but accessing such data in biology and conservation is a challenge. Although many public libraries contain skunk, fox, bird, and empty images, we decided to instead utilize the open labeled dataset from the Labeled Information Library of Alexandria: Biology and Conservation (LILA BC). Our partner at The Nature Conservancy (TNC) created this dataset. The LILA BC is a repository for data sets related to biology and conservation, intended as a resource for both machine learning (ML) researchers and those that want to harness ML for biology and conservation (LILA BC). Within this dataset, over 240,000 label images are collected from 73 individual cameras spread out over seven different classes. Every single image was annotated with bounding boxes and followed a similar structure to that of the COCO Camera Traps format.

2.1.1 COCO Camera Trap Format and Description

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 here.

COCO Formatted - LILA BC Channel Islands Camera Trap Data Set

```
{
"images": [
 {
  "id": "dd8b68e9-360b-429e-a43b-892c2e036455",
  "file_name": "loc-h500ee07133376/000/000.jpg",
  "seq_id": "836f6487-50fd-42f5-8dcc-336fc538b7a8",
  "seq_num_frames": 6,
  "frame_num": 0,
  "original_relative_path":
      "2011_09_Set/Station%201/2011/2011-09-13/IMG_0001.JPG",
  "location": "h500ee07133376",
  "temperature": "21 c",
  "width": 1920,
  "height": 1080
 }
  .
  . Signifying that there are more entries following
  . same shcema but were left out.
  .
],
 "annotations": [
  {
   "id": "16e360cc-4a53-11eb-b9b3-000d3a74c7de",
   "image_id": "dd8b68e9-360b-429e-a43b-892c2e036455",
   "category_id": 0,
   "sequence_level_annotation": false,
   "bbox": [
    0,
    0,
    1919,
    1079
   ]
  }
  .
  . Signifying that there are more entries following
  . same shcema but were left out.
  .                   .
],
"categories": [
 {
  "id": 0,
  "name": "empty"
 },
  .
  . Signifying that there are more entries following
  . same shcema but were left out.
  .
],
"info": {
 "year": 2020,
 "version": 1.0,
 "description": "Camera trap data collected from the Channel Islands,
    California",
 "contributor": "The Nature Conservancy of California"
}
}
```

COCO Formatted - LILA BC Channel Islands Camera Trap Data Set Description

---

```
{
"images": [
 {
  "id": Unique identifier within the image section,
  "file_name": The file name is equal to the path from the parent
     directory to the file location,
  "seq_id": Sequence identifier shared by seq_num_frame amount of images,
  "seq_num_frames": Number of images taken in a sequence,
  "frame_num": Frame number in a specific sequence_id,
  "original_relative_path": Original file path from where the image was
     collected,
  "location": First subdirectory that is needed to locate the image,
  "temperature": Temperature at the time that the image was taken,
  "width": Width of the image,
  "height": Height of the image
 }
  .
  . Signifying that there are more entries following the same
  . schema but were left out.
  .
],
 "annotations": [
  {
   "id": Unique identifier within the annotation section, different from
       images.id,
   "image_id": Foreign key within annotations and primary key to images.id,
   "category_id": Category given by classification method used,
   "sequence_level_annotation": unknown,
   "bbox": [Four integers describing the location of the bounding box.
       These integer values range from a minimum/max height and width equal
       to images.height and images.width respectively
   ]
  }
  .
  . Signifying that there are more entries following
  . the same schema but were left out.
  .
],
"categories": [
 {
  "id": Unique identifier for each class present throughout the dataset.
      Different from image.id and annotation.id,
  "name": Comprehensible name for each respective id
 },
  .
  . Signifying that there are more entries following
  . same schema but were left out.
  .
],
"info": {
 "year": The year when dataset was created,
 "version": Version of the dataset that was downloaded,
 "description": Short description of what the data is representative of,
 "contributor": Contributors of this dataset
}
}
```

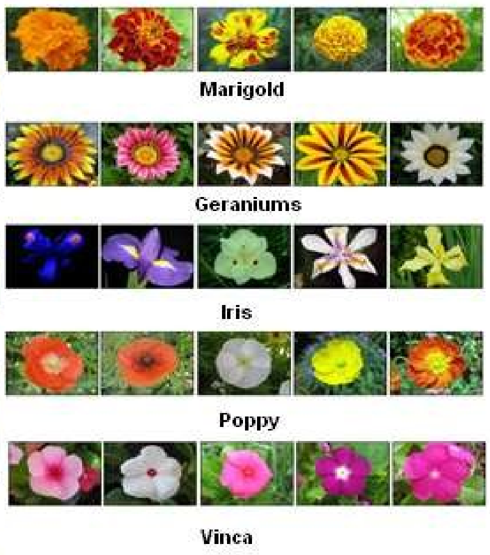
2.1.2 Sorting a COCO Formatted - LILA BC Data Set by Class(link both end-to-end codduments respectively)

Now that we have a general understanding of what a COCO formatted dataset contains, we must convert it to the CSV file from which we can slice columns and values. Using this CSV file, we will be able to separate images into folders sorted by their respective class. To automate this process, we have created a python executable (jsonToCsv.py). jsonToCsv.py utilizes the relative or absolute path of the COCO JSON file to covnert the COCO file to multiple CSV files which contain each images relative path and each images file name. Having the relative path of the files is important as we will use this path to move locate and move the images into folders using trainMove.sh.

massMove.sh is a shell executable which is written using UNIX scripting language, this language is used to complete repetitive task in a fraction of the time that a person would take to complete the same task. Knowing the capabilities of this scripting language we will use the files (foxPath.csv, otherPath.csv, skunkPath.csv etc) containing the files path and file name of each image to move each image into folders separated by class.

The output produced by trainMove.sh will have a similar schema as what is shown below, with (A) being the input format (no specific format) and (B) being the output format of the files (images sorted into folders by category).

(a)



(b)

2.2 Machine Learning on the Edge

2.2.1 A Brief Introduction to Machine Learning and EffecientNet

Neural networks are mathematical models composed of many equations arranged in layers and connected to other nodes in different layers but not within the same layer. The neural network also consists of an input layer, at least one intermediate or hidden layer, and an output layer. Each node has a bias value, a weight value, and an activation function. Fully connected or dense neural networks use information flowing from left to right and are manipulated using linear algebra. Convolutional neural networks use several different types of layers but are still often used in many architectures. For our project, we will be going with a simpler, more lightweight architecture called "effecientnet_lite3" as we will be running this model on edge devices.

EfficientNet is a convolutional neural network architecture and scaling method that uniformly scales all depth/width/resolution dimensions using a compound coefficient. Unlike conventional practice that arbitrarily scales these factors, the EfficientNet scaling method uniformly scales network width, depth, and resolution with a set of fixed scaling coefficients. For example, if we want to use 2^N times more computational resources, then we can simply increase the network depth by α^N, width by β^N, and image size by γ^N, where α, β, γ are constant coefficients determined by a small grid search on the original small model. EfficientNet uses a compound coefficient ϕ to uniformly scales network width, depth, and resolution in a principled way (paperswithcode.com).

2.2.2 Training a Model

Since we will be using the pre-designed EffecientNet Lite3 architecture, we can load the training data using various methods. However, the method that best aligns with the current structure of our training data would be tflite_model_maker.image_classifier.DataLoader.from_folder() as this method assumes that the training images will come from subdirectories sorted by class and that the subdirectory names as the class labels for the images in that specified subdirectory.

In our case, the seven facets available to us are empty, human, fox, skunk, rodent, bird, and other. As mentioned, these seven classes are in their respective subdirectories, which are labeled accordingly. To train the model all, you must run the ML_Model.ipynb notebook with the correctly formatted training data that we organized earlier using trainMove.sh. For an in-depth explanation of trainMove.sh please reference the documentation located here.

2.2.3 Deploying the Model

We must develop more code compatible with an edge device such as the Raspberry Pi (RPI) to use this model. As mentioned, these edge devices (Raspberry Pi) tend to be less GPU and CPU capable, so we must accommodate for these limitations.

As mentioned, we have over twenty cameras strategically placed throughout the island, and each camera produces an average of about 35,000 images. To keep track of the images and **comply**

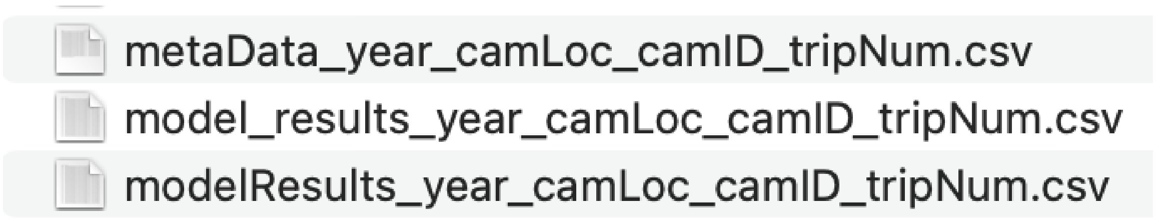
with Camera Trap Data Package specification, we must assign each file a unique file name. To do so, we created mass_rename.sh. What this file does is take in n user-defined number of variables and attach them to the image name and rename every file throughout the SD card numerically starting at one and ending at i where i is the total number of images present throughout SD card. An example of a filename can be seen below.

```
Example: 2022_UCSB02_2B_img_00001.jpg
```

Now that all the images have been given a unique name we can proceed to the next step which is to use our custom-trained image classifying model to identify weather or not there any skunks, fox, etc in any of the images. To classify images, we will use rpiClassify.py. This python executable has been designed using TensorFlow lite to accommodate for the less capable GPU and CPU present on edge devices. **Once all the images have been classified, you will see a message**.

```
"Model Results file for this camera has been created and saved in the current working directory <SD card parent directory>
```

Upon executing rpiClassify.py you should see three files appear within the parent directory of the SD card being used. These files should follow a similar schema as shown below.


metaData_year_camLoc_camID_tripNum.csv
model_results_year_camLoc_camID_tripNum.csv
modelResults_year_camLoc_camID_tripNum.csv

2.2.4 metaData_year_camLoc_camID_tripNum.csv

As stated in the name, metaData_year_camLoc_camID_tripNum.csv is a CSV file containing all the metadata of each image within the SD card's parent directory. To make this possible, we use an open-source tool called Exiftool by Phil Harvey. **Exif Tool** is a widely used meta-data information recorder built on Perl by Phil Harvey. It is a kind of open-source tool that works on various file types. EXIF stands for Exchangeable Image File Format, and it is mainly used for including metadata in various file types like txt, png, jpeg, pdf, HTML, and many more. With the EXIF tool, we can also read, write and manipulate such meta-data information. The Length of metaData_year_camLoc_camID_tripNum.csv should be equal to the total number of images (.jpg files) in the parents directory.

2.2.5 modelResults_year_camLoc_camID_tripNum.csv

As stated in the name, modelResult_year_camLoc_camID_tripNum.csv is a CSV file containing all the classification results of each image within the SD cards parent directory. Each image will have an observationType, classificationConfidence, classificationTimestamp, & mediaID. All of these are discussed in section 2.4 The Length of modelResults_year_camLoc_camID_tripNum.csv should be equal to a total number of images (.jpg files) in the parent's directory.

2.2.6 envVars.csv

envVars.csv is a .csv file with a single column of 4 entries. These four entries are important to have because deployMediaObs.py is dependent on this file. The four entries in this CSV are read_location, mass_rename_direct, regex, and metaData_$regex.csv.

- **read_location**: The file path to the parent directory of the SD card. This file path is utilized within deployMediaObs.py to find every possible path to the other files created, such as read_location/modelResults_year_camLoc_camID_tripNum.csv. This is more convenient as we eliminate as much human error as possible.

  Example: /Volumes/NO NAME/DCIM

- **mass_rename_direct**: When executing mass_rename.sh the user is prompted to input the absolute path to mass_rename.sh as a way of calling within itself in case the user would like to do a complete restart of the function.

  Example: /absolute/path/to/mass_rename.sh

- **regex**: regex is the prefix given to every image and name schema used when naming metaData...csv, modelResults...csv, and every other file produced. Having this prefix saved is convenient as it saves the user from possible errors when naming other files.

  Example: year_camLoc_camID_tripNum

- **metaData_${regex}.csv**: this is the file name of the metaData....csv file plus the regex mentioned above. This is another file name that will also be called when executing deployMediaObs.py and should be saved to do so.

  Example: metaData_year_camLoc_camID_tripNum.csv

2.2.7 Camera Trap Data Package & deployMediaObs.py

In the section 2 (Camera Trap Data Package) we saw what a well-established method for data collection and data sharing looked like. If we compare it to the files produced in section 3.2.4 & section 3.2.5 we can notice that we have all the necessary tools required by Camera Trap Data Package, all that needs to be done is a bit of reordering and renaming of the columns. To complete the necessary task, I have developed a python executable called deployMediaObs.py. THis python executable utilized functions like glob.glob() to search for the parent directory, meta...csv, envvars.csv, and all other necessary files required. We decided to automate this process as the file paths are not going to change and remove as much room for human error as possible (i.e. having them manually input the path). Once having found the correct files the user will be asked to input answers to questions regarding other sections that may change over time or that aren't included in the metaData..csv file(i.e. latitude & longitude).

Once all the fields have been answered correctly, deployMediaObs.py will produce the three following files which comply with deployments, media and observations sector of Camera Trap Data Package specification.

```
deployments_year_camLoc_camID_tripNum.csv

media_year_camLoc_camID_tripNum.csv

observations_year_camLoc_camID_tripNum.csv
```

This concludes the workflow document.