End-To-End walk through and explanation of rpiClassify.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

About this Document

This document serves as a procedural guide for various processes, detailing explicitly how tasks have been accomplished and how to navigate rpiClassify.py. This software and document is written for students and mentors of the UCSB-Smithsonian Scholars Program.

If at any moment you feel that there is any ambiguity, lack of explanation, or uncertainty, please refer to section 7 and section 8 of this document as it provides an in-depth explanation of most functions, modules, and packages that are utilized throughout the python executable.

SECTION 2

A Brief Overview

In this section, I will give a brief walkthrough of what must be done in preparation for the deployment of a model and how to deploy the model; you can see an in-depth explanation of this process in section section 3 and onward.

2.1 Installing tflite-runtime

Once we download the Interpreter class, you can reference the package as you would in any notebook. An example can be seen in the rpiClassify.py executable file.

2.2 Preparing for classification

In order to classify an image, we must do more than give a file path and expect a result. It takes a little more work than that.

- **Setting up Interpreter**: To set up the interpreter we must use Interpreter(<path/to/model.tflite>). This will allow us to open up our pre-trained model and reference it down the line.
- **Allocate Memory**: Memory is allocated to the input and output tensors by calling the allocate_tensor() method on the previously loaded interpreter.
- **Sanity Check**: Now that we have allocated memory and set up the interpreter class, it is a good idea to do a sanity check and see if the dimensions of our input tensors are what we expect them to be.
- **Invoke Interpreter**: After loading the input image, the invoke() method is called to invoke the interpreter over the input tensor and evaluate the output tensor. We Call the invoke()

method from inside a function to avoid runtime errors.

- **Read Class Labels**: Reading the text file 'class_labels.txt' holding the labels of the classes by which the model was trained. The class_labels.txt can be found here and should look exactly like what is shown below.

```
Bird
Empty
Fox
Skunk
```

2.3 Making a Prediction & Saving Results

- **Making a Prediction**: Once all that has been done, we can finally give the model a file path; the model will produce the following once given a correct file path: Image label, File name, Classification confidence, & Classification time stamp.
- **Saving Results**: The attributes mentioned above will be saved to a CSV file following the schema shown below. The CSV will have as many rows as files present when the model makes the classifications. This data package will not have a primary key but instead will have a foreign key (fileName) which is a primary key within MetaData....csv. metaData...csv is a file produced by mass_rename.sh

```
Example: metaData_2022_UCSB02_2B_01.csv
```

3

SECTION

TensorFlow and Tensorflow Lite

3.1 TensorFlow

TensorFlow (TF) is an open-source library for numerical computation and large-scale machine learning. It can be used to train and run deep neural networks for handwritten digit classification, image recognition, word embeddings, recurrent neural networks, and PDE-based simulations. TF provides easy-to-learn, works with Python code, and is supported on Python versions 3.7 through 3.10. TF uses Python to create nodes and tensors, but the actual math operations are performed in C++ binaries, giving it many of its high-performance capabilities. TF models can be deployed on edge computing or mobile devices, and the TensorFlow Lite toolset allows you to make tradeoffs between model size and accuracy. This will be covered and further explained in section 3.1.2.

3.2 TensorFlow Lite

Tensorflow Lite (TF Lite) is an open-source, cross-platform deep-learning framework that converts a pre-trained model to a special format that can be optimized for speed or storage. TF Lite shares many of the same capabilities as TensorFlow (TF). However, TF Lite specializes in optimized on-device machine learning by addressing the following key points:

- Compatibility with various edge devices such as Android, iOS, Linux, and Linux devices (i.e. Raspberry Pi).
- Low Latency.
- Locally process files (doesn't require a connection to the cloud)
- Size & power consumption are reduced to make up for edge device requirements.

Note: The TensorFlow Lite binary is about 1MB when all 125+ supported operators are linked (for 32-bit ARM builds), and less than 300KB when using only the operators needed for supporting the common image classification models InceptionV3 and MobileNet (TensorFlow Website).

4

SECTION

Preparing Tensorflow Lite on RPI

TensorFlow Lite is part of TensorFlow when you install the TensorFlow library, and you will also install TF Lite. However, we don't want the entire TF library; we only need TF Lite. In this tutorial, we need to run a TF Lite model for classifying images and nothing more. Based on this, we do not need to install everything in TF, just the parts relevant to our task. For making predictions using a TF Lite model, the only class needed from TensorFlow is the Interpreter class, accessed by tensorflow.lite.python.interpreter.Interpreter. So, rather than installing everything in TensorFlow, we can install this class. This saves our storage from holding unused files.

4.1 Installing tflite-runtime

To access the interpreter class, we need to find a specific package that contains the interpreter. tflite_runtime contains only the Interpreter class and can be accessed using tflite_runtime.interpreter.Interpreter. In order to download this file, you will need to enter the following command into your activated SkunkEnv Conda Environment. See how to build and activate a Conda environment for our project here (link to building a condo environment for our project.)

```
sudo OPENBLAS\_CORETYPE=ARMV8 python3 -m pip install tflite-runtime
```

With this command line, we are specifying that our operating system uses ARMV8 architecture. And to subsequently download the tflite-runtime package (compatible with ARMV8) via pip.

4.2 Importing Classes & Modules

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

```
from PIL import Image
import numpy as np
import os
import pandas as pd
import datetime
import warnings
import glob
```

4.3 Suppressing Unwanted Warning Messages

Since we are using an older version of python and Pandas we will get error messages encouraging us to use new libraries and functions which have now been removed. One example of this is .append(); this function is no longer used and has been replaced by .concat(). For us it is more convenient to use .append(), so we will suppress the warning message we get from using this function by using the following line of code.

```
warnings.simplefilter(action='ignore', category=FutureWarning)
```

4.4 Locating envVars.csv

In order to avoid as much possible human error as possible, we have automated the program as much as possible; one step that has been taken to do this is by implementing the use of glob.glob() to locate envVars.csv. This CSV file contains file names and other attributes necessary to maintain consistency throughout the project.

```
glob.glob("/media/pi/*/DCIM1/envVars.csv")
```

Once the envVars.csv has been located the program will automatically read in the csv file using .read_csv().

```
pd.read_csv(filename, sep = "=")
```

4.5 def load_label()

In order to correctly classify images, we must use the labels that were used to train the model and that is what load_label() does. This function reads the classes and them in their respective order.

4.6 def set_input_tensor()

Based on the input tensor's index, return this tensor. The input image is then assigned to that tensor.

4.7 def classify_image()

After loading the input image, the invoke() method invokes the interpreter over the input tensor and evaluates the output tensor. We Call the invoke() method from inside a function to avoid the error shown.

```
RuntimeError: reference to internal data in the interpreter in the form of
    a NumPy array or slice.
```

After that, the details of the output tensor (e.g. index) are returned by calling the get_output_details() method. Based on the index of the output tensor, the output of the output tensor is returned using the get_tensor() method. Because the used TensorFlow Lite model is quarantized, the class prediction scores of the output tensor are returned as integers. The prediction scores are dequantized to return the probability of each class. Based on the class probabilities, the index of the predicted label is returned.

4.8 Setting Model & label Path

Since this is a Model that has been pre-trained and saved to our local working directory we must specify. This will be a static path that will never be changed so there is no need to make the user input either of these paths.

```
model_path =
    "/home/pi/Downloads/SCI_skunks-main/tflite_model/model-4.tflite"
label_path =
    "/home/pi/Downloads/SCI_skunks-main/tflite_model/class_labels.txt"
```

4.9 Loading Pre-trained TF Lite Model

We will create an instance of the Interpreter class and load the pre-trained TensorFlow Lite model.

```
Interpreter(model_path)
```

4.10 Allocating Memory

Memory is allocated to the input and output tensors by calling the allocate_tensors() method.

```
interpreter.allocate_tensors()
```

4.11 Sanity Check

Now that we have allocated memory and set up the interpreter class, it is a good idea to do a sanity check and see if the dimensions of our input tensors are what we expect them to be. In our case, we should see a message printed as follows below.

```
Details about the input tensors:
    [{'name': 'input_1', 'index': 0, 'shape': array([ 1, 280, 280, 3],
        dtype=int32), 'shape_signature': array([ -1, 280, 280, 3],
        dtype=int32), 'dtype': <class 'numpy.uint8'>, 'quantization':
        (0.003921568859368563, 0), 'quantization_parameters': {'scales':
        array([0.00392157], dtype=float32), 'zero_points': array([0],
        dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}]
There exist 1 input tensor(s).
   Type: <class 'list'>

Details about the first input tensor:
   {'name': 'input_1', 'index': 0, 'shape': array([ 1, 280, 280, 3],
       dtype=int32), 'shape_signature': array([ -1, 280, 280, 3],
       dtype=int32), 'dtype': <class 'numpy.uint8'>, 'quantization':
       (0.003921568859368563, 0), 'quantization_parameters': {'scales':
       array([0.00392157], dtype=float32), 'zero_points': array([0],
       dtype=int32), 'quantized_dimension': 0}, 'sparsity_parameters': {}}
   Type: <class 'dict'>

Shape of the first input tensor:
    [  1 280 280   3]
   Type: <class 'numpy.ndarray'>

Image Width = 280
Image Height = 280
```

4.12 Creating an Empty Pandas Dataframe

To store the data, we will be using a dictionary which will subsequently be converted to a Pandas Dataframe. The data in a dictionary is stored as a key/value pair. It is separated by a colon, and the key/value pair is separated by a comma. The keys in a dictionary are unique and can be a string, integer, tuple, etc. The values can be a list or a list within a list, numbers, strings, etc.

To create this empty data frame, we first create an empty dictionary with the key values that we know will be used later on. Once this has been created, we can convert this dictionary to a pandas data frame and save it as 'results'. We can name it anything else, but for convenience purposes, I have named it 'results'. Please refer to the code below for a better understanding.

```
dict = {'imageLabel':[],
'imageName':[],
'accuracy':[],
'classificationTimeStamp' : []}

results = pd.DataFrame(dict)
```

5

SECTION

Classifying Images

Now that all the pre-requisite steps have been taken, we can finally start the classification process. In essence what we are doing is going through every image within the SD card and having the model make a prediction based on patterns picked up within the image.

5.1 A Simple For Loop

A for loop is used for sequential traversal i.e. it is used for iterating over an iterable like String, Tuple, List, Set, or Dictionary.

In our case, we are using a for loop that extracts the parent directory (directory of which file belongs to; dirpath), directory/subdirectory (names of directory/subdirectories in a vector format; dirnames), and file names (name of the file; filenames). In this case, we want the filenames since we want that is was is we will be feeding into classify_image(). Once we have the file name, we have our first control statement, which is used to ensure that the file being proccessed is a .jpg file and not anything else.

```
for (dirpath, dirnames, filenames) in os.walk("/media/pi/C060-4E55/DCIM1"):
    for filename in filenames:
        if filename.endswith('.jpg'):
```

If and only if the file is a .jpg then we can move on with the process, which is to concatenate both the parent directory (dirpath) and file name (filename). This will give the absolute/relative path to the file that way; we can open it using image.open() method.

```
os.sep.join([dirpath, filename]))
```

Now that we have gone through the process of checking that we have indeed used a .jpg file, we can open the file, convert it to the RBG color pallet, and resize it to the correct specifications using the code shown below.

```
Image.open(images).convert('RGB')

image.resize((width, height))
```

Finally, we make the classification by feeding in the resized and recolored image into classify_image(interpreter, image). This function will return a label id and a probability for the image that was classified. Now that we have a label, we must match it to the labels in the 'class_labels.txt' file. To do so, we first open the file and save the labels to memory, then we match the labels and set that output as classification_label. Reference the code shown below.

```
#classify the image.
label_id, prob = classify_image(interpreter, image)

# Read class labels.
with open(label_path, 'r') as f:
labels = [line.strip() for i, line in enumerate(f.readlines())]

# Return the classification label of the image.
classification_label = labels[label_id]
```

Lastly, in order **to comply with Camera Trap DP** specification, we must get the **extact** time of when the .jpg file was classified in an ISO 1860 format. To do this, we will use the datetime module, as shown below.

```
datetime.datetime.now()
```

This concludes the classification process, to properly format & save our results, please see the following section, section 4.

SECTION 6

Formatting and Saving Our Results

6.1 Formatting Our Data

To Save our since we are saving the data produced by each to a CSV file, we must use a quick and efficient method. To accomplish this, we will use the empty dictionary created in the steps shown prior.
In our case, we have **keys shown in purple text**, and the **values shown in black text**. This dictionary will be saved as 'current_entry' and will store the key/value pairs of the current entry that is being processed as explained in section 3.1.

```
current_entry = {'observationType': classification_label, "imageName":
    fileName, 'classificationConfidence': (np.round(prob*100, 2), "%."),
    "classificationTimestamp": timeOfClass}
```

Now that we have properly structured the current entry, we can append it to the 'results' data frame using the .append() function as shown below. This function will append 'current_entry' to the data frame 'results' as shown below. This data frame will also contain an index column as specified by the 'ignore_index' attribute.

```
results.append(current_entry, ignore_index = True
```

Once that image has been classified and saved in the data frame we should see the following

message appear on the user's terminal window.

```
"classified image <image_name>"
```

The last couple steps of this process are to export this data frame as a CSV, to do this we will use .to_csv(). This function uses the existing Panda's data frame and exports it to your current working directory as a CSV file under a specified name. The name we choose to give the CSV follows the schema which was created when executing mass_rename.sh.

```
csvName = "modelResults_" + str(fileName)

results.to_csv(csvName, sep=',', index=Fals
```

If all was done correctly you should see the following message appear in your terminal window.

```
'Model results file for this camera has been created and saved in the
    current working directory under <csvName>'
```

7

SECTION

Understanding Our Work

In summary, RpiClassify.py is an executable python script that utilizes a pre-trained Tensorflow Lite model and images from the camera trap sd card to create one file named 'model_results...csv'. This CSV File will contain the following entries.

- **observationType**: Classification of the media file (image) is done by the custom-trained tensor flow model.

  Example: Skunk

- **imageName**: The name of the file which was processed.

  Example: 2022_UCSB02_2B_01_img_00001.jpg

- **classificationConfidence**: classficationConfidence should be the accuracy or confidence that the classificationMethod had when classifying the media file (image).

  Example: 75.5%

- **classificationTimestamp**: classificationTimestamp should be the time when the media file (image) was processed using the previously mentioned classificationMethod.

  > Example: 2022-08-22T10:25:19Z

It is important to double-check the location of this file. If it was done properly, it should be located in the SD card's parent directory if it was sent someone where else, please use to the following command to place it in the correct directory. **Be sure not to forget the period at the end of the mv command.**

On the Raspberry Pi, you can right-click on the directory location and select "copy path". Once done, move over to the terminal, right-click, and select "paste" in place of the <path> shown below.

```
cd <absolute/path/to/sd/card/parentDir>

mv <absolute/path/to/modelResults...csv> .
```

8

SECTION

Data Scientist Tool Box - Unix Shell

8.1 sudo Command

Sudo (superuser do) is a utility for UNIX- and Linux-based systems that efficiently give specific users permission to use specific system commands at the system's root (most powerful) level.

8.2 pip Command

Python pip is the package manager for Python packages. We can use pip to install packages that do not come with Python. The basic syntax of pip commands in the command prompt is as follows.

```
pip -argument
```

9

SECTION

Data Scientist Tool Box - Python

9.1 os Module

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

9.1.1 .walk()

os.walk() generate's the file names in a directory tree by walking the tree either top-down or bottom-up and yields 3-tuples (dirpath, dirnames, filenames)

9.1.2 .sep.join()

os.path.join() method concatenates various path components with exactly one directory separator ('/') following each non-empty part except the last path component. If the last path component to be joined is empty, then a directory separator ('/') is put at the end.

Note: If a path component represents an absolute path, then all previous components joined are discarded and joining continues from the absolute path component.

9.2 Pillow Module

Python Imaging Library (PIL) is an image-processing package for Python. It incorporates lightweight image-processing tools that aid in editing, creating, and saving images.

9.2.1 Image.open().convert()

Opens and identifies the given image file and converts it to a specified color grid.

9.2.2 Image.resize()

.resize() is used to change the size of an image.

9.3 Numpy Module

9.3.1 .argpartition()

Used to perform an indirect partition along the given axis using the algorithm specified by the "kind" keyword. It returns an array of indices of the same shape as that index data along the given axis in partitioned order.

9.3.2 .squeeze()

.squeeze() function is used when we want to remove single-dimensional entries from the shape of an array.

9.4 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.

9.4.1 .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.

9.4.2 .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.

9.4.3 .loc[]

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

9.4.4 .DataFrame()

Utilized to create a dataframe in Pandas.

9.4.5 .append()

.append is utilized to add a row to an existing pandas dataframe.

9.5 datetime Module

Python Datetime module supplies classes to work with date and time.

9.5.1 .datetime.now()

.datetime.now() is a combination of date and time along with the attributes year, month, day, hour, minute, second, microsecond, and tzinfo at the time when it was run.

9.5.2

9.6 warnings Module

We use the warnings module to control and suppress warning messages that may come from outdated libraries and modules.

9.6.1 .simplefilter()

the .simplefilter() class is utilized to suppress unwanted messages. We specifically do this to avoid a warning message that is produced from using .append() rather than .concat()

9.7 glob module

The glob module searches all path names looking for files matching a specified pattern according to the rules dictated by the Unix shell. Results so obtained are returned in no specific order.

9.7.1 .glob()

We can use the function glob.glob() to retrieve paths recursively from inside the directories/files and subdirectories/subfiles that match the specified conditions.

9.8 Interpreter Module

Interpreter interface for running TensorFlow Lite models on Raspberry Pi. This specific module is drawn from tflite_runtime.

9.8.1 .get_input_details()

A list in which each item is a dictionary with details about an input tensor. Each dictionary contains the following fields that describe the tensor:

- **name**: The tensor name.
- **index**: The tensor index in the interpreter.

- **shape**: The shape of the tensor.
- **shape_signature**: Same as shape for models with known/fixed shapes. If any dimension sizes are unknown, they are indicated with -1.
- **dtype**: The NumPy data type
- **quantization**: Deprecated, use quantization_parameters.
- **quantization_parameters**: A dictionary of parameters used to quantize the tensor.
- **quantized_dimension**: Specifies the dimension of per-axis quantization, in the case of multiple scales/zero_points.
- **sparsity_parameters**: A dictionary of parameters used to encode a sparse tensor. This is empty if the tensor is dense

9.8.2 .get_output_details()

A list in which each item is a dictionary with details about an output tensor. The dictionary contains the same fields as described for .get_input_details().

9.8.3 .allocate_tensors()

Memory is allocated to the input and output tensors by calling the .allocate_tensors() method on the interpreter.

10

SECTION

rpiClassify.py

10.1 Source Code

```
#chmod +x <path to rpiClassify.py>
#sudo OPENBLAS_CORETYPE=ARMV8 pip install progressbar2

#pip3 install --extra-index-url https://google-coral.github.io/py-repo
    tflite_runtime

from tflite_runtime.interpreter import Interpreter #sudo
    OPENBLAS_CORETYPE=ARMV8 python3 -m pip install tflite-runtime
from PIL import Image
import numpy as np
import os
import pandas as pd #sudo OPENBLAS_CORETYPE=ARMV8 pip3 install pandas
import datetime
import warnings

import glob

warnings.simplefilter(action='ignore', category=FutureWarning)

listing = glob.glob("/media/pi/*/DCIM1/envVars.csv") #using wild card to
    find the specific file names envVars.csv that is in the DCIM1 folder
for filename in listing:
```

```
  envVars = pd.read_csv(filename, sep = "=") #customo delimeter "=" rather
     than normal ','

fileName = envVars.loc[envVars.Variables=="fileNamesMeta"].Values
    #extracting csv file name that will be convension for the rest of the
    files created

def load_labels(path): # Read the labels from the text file as a Python
    list.
   with open(path, 'r') as f:
    return [line.strip() for i, line in enumerate(f.readlines())]

def set_input_tensor(interpreter, image):
  tensor_index = interpreter.get_input_details()[0]['index']
  input_tensor = interpreter.tensor(tensor_index)()[0]
  input_tensor[:, :] = image

def classify_image(interpreter, image, top_k=1):
  set_input_tensor(interpreter, image)

  interpreter.invoke()
  output_details = interpreter.get_output_details()[0]
  output = np.squeeze(interpreter.get_tensor(output_details['index']))

  scale, zero_point = output_details['quantization']
  output = scale * (output - zero_point)

  ordered = np.argpartition(-output, 1)
  return [(i, output[i]) for i in ordered[:top_k]][0]

model_path =
    "/home/pi/Downloads/SCI_skunks-main/tflite_model/model-4.tflite"
label_path =
    "/home/pi/Downloads/SCI_skunks-main/tflite_model/class_labels.txt"

interpreter = Interpreter(model_path)
#print("Model Loaded Successfully.")

interpreter.allocate_tensors()

print("Details about the input tensors:\n ",
    interpreter.get_input_details())
print("There exist {num} input
    tensor(s).".format(num=len(interpreter.get_input_details())))
print("  Type:", type(interpreter.get_input_details()), end="\n\n")
```

```
print("Details about the first input tensor:\n ",
    interpreter.get_input_details()[0])
print("  Type:", type(interpreter.get_input_details()[0]), end="\n\n")

print("Shape of the first input tensor:\n ",
    interpreter.get_input_details()[0]['shape'])
print("  Type:", type(interpreter.get_input_details()[0]['shape']),
    end="\n\n")

_, height, width, _ = interpreter.get_input_details()[0]['shape']
print("Image Width = {width}\nImage Height = {height}".format(width=width,
    height=height))

dict = {'imageLabel':[],
        'imageName':[],
        'accuracy':[],
        'classificationTimeStamp' : []}
results = pd.DataFrame(dict)

for (dirpath, dirnames, filenames) in os.walk("/media/pi/C060-4E55/DCIM1"):
    for filename in filenames:
        if filename.endswith('.jpg'):
          images = (os.sep.join([dirpath, filename]))
          timeOfClass = datetime.datetime.now()

          # Load an image to be classified.
          image = Image.open(images).convert('RGB')
          #print("Original image size:", image.size)

          image = image.resize((width, height))
          #print("New image size:", image.size, end="\n\n")

          # Classify the image.
          label_id, prob = classify_image(interpreter, image)

          # Read class labels.
          with open(label_path, 'r') as f:
            labels = [line.strip() for i, line in enumerate(f.readlines())]

          # Return the classification label of the image.
          classification_label = labels[label_id]
          #print("Image Label:", classification_label, " ", "\nImage
              Name:", images, "\nAccuracy :", np.round(prob*100, 2), "%.")
```

```
            current_entry = {'observationType': classification_label,
                "imageName": fileName, 'classificationConfidence':
                (np.round(prob*100, 2), "%."), "classificationTimestamp" :
                timeOfClass}

            results = results.append(current_entry, ignore_index = True)

            print("classified image " + images)

csvName = "modelResults_" + str(fileName)

results.to_csv(csvName, sep=',', index=False)

#results.to_csv(csvName, sep=',', index=False)

print("Model results file for this camera has been created and saved in
    the current working directory under'" + csvName + "'")
```