Image Classifier

Installing Tensorflow using pip

Prerequisites:

1. The first step is to verify if Python is installed using the following command:

```
$ python -V
```

At least one of 2 possible versions must be installed: * Python 2.7 * Python 3.3+

1. Now that we know that Python is installed, we proceed to verify if pip or pip3 is installed

```
$ pip -V
$ pip3 -V
```

If it's not installed, proceed to install it using the following command:

```
$ sudo apt-get install python-pip python-dev #For Python 2.7
$ sudo apt-get install python3-pip python3-dev #For Python 3.3+
```

Installing Tensorflow

1. Now we proceed to install tensorflow

~~~
$ pip install tensorflow      # Python 2.7; CPU support (no GPU support)
$ pip3 install tensorflow     # Python 3.3+; CPU support (no GPU support)
$ pip install tensorflow-gpu  # Python 2.7;  GPU support
$ pip3 install tensorflow-gpu # Python 3.3+; GPU support
 ```
2. If Step 1 failed, you need to install the latest version of tensorflow, issuing
one of the following commands:
```bash
$ sudo pip  install --upgrade TF_PYTHON_URL   # Python 2.7
$ sudo pip3 install --upgrade TF_PYTHON_URL   # Python 3.3+
 ```
3. TF_PYTHON_URL identifies the URL of the TensorFlow Python package. This URL
might be found in this [link][tensor url].

### Validate installation
#### Run short program of tensorflow
* Invoke python from console
```bash
$ python
 ```
* Enter following short program.
```py
>>> import tensorflow as tf
~~~

```
>>> hello = tf.constant('Hello, TensorFlow!')
>>> sess = tf.Session()
>>> print(sess.run(hello))
```

- The output should be:

```
Hello, TensorFlow!
```

Compiling the program

Exporting tensorflow variable

1. To correctly run the program, first we must export a variable for tensorflow, we can do this by running the following comand:

```
$ export TF_CPP_MIN_LOG_LEVEL=2
```

Compiling using Makefile

1. There first option is to compile using the Makefile in the directory where the file is.

```
$ make
```

1. To delete the created files use: `bash $ make clean`

Compiling manually

1. The second option is to compile manually using the following command (To run parallelism it must be compiled manually):

```
$ gcc index.c -o $name -lpthread -fopenmp
```

Running the program

Prerequisites

1. We need a directory named "images" inside the current working directory. We can do this by using the following command:

```
$ mkdir images
```

1. Inside the directory we must put some images (ex. car.jpeg). We may download more than 1.

Running the program

1. First we execute the compiled program:

```
$ ./index
```

1. The output should look like this:

```
Created user thread
Hello, welcome to iamyoureyes
 With me you will be able to recognize and know many new things
choose the option you want
 1.- Image recognition
 2.- smt else
 3.- smt else
 0.- exit/quit
 ```

3. We select the option 1 first. This will be the image that we will classify. This
will be the output:
```bash
Created image recognizer thread
please type the name of the image you want to analyze as it is (exaple: car.jpg)
```

1. We must enter the name of the image we wish to classify. Note that the name is case sensitive and it must exist on the images directory. We might get some output of TenserFlow libraries, we can ignore that. Now the output should look like this:

```
rana.jpg
tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui (score = 0.92112)
bullfrog, Rana catesbeiana (score = 0.02468)
tree frog, tree-frog (score = 0.01106)
common newt, Triturus vulgaris (score = 0.00207)
banded gecko (score = 0.00202)
```

Get help with Python file

1. If you wish to know which arguments are being parsed to Python. You can run the following command:

```
$ python classify_image.py -h
```

Analyzing the results

How to analyze

1. As we saw in the 4th point of the previous section, we get some results. After gathering data, the program will look for certain matches within it's training. They look like this:

```
tailed frog, bell toad, ribbed toad, tailed toad, Ascaphus trui (score = 0.92112)
bullfrog, Rana catesbeiana (score = 0.02468)
tree frog, tree-frog (score = 0.01106)
common newt, Triturus vulgaris (score = 0.00207)
banded gecko (score = 0.00202)
```

1. In the brackets, we see a number between 0 and 1. This number represents how likely it is for it to be that category.
    - For example, the tailed frog category has a likeliness of 0.921112, which is very close to 1. That means that that image must belong to that category.

Documentation

C Documentation

- Needed libraries

```
#include <stdio.h>
#include <stdlib.h>
#include <pthread.h>
#include <string.h>
#include <sys/types.h>
#include <dirent.h>
#include <omp.h>
```

- Global variables

```
#define BUFFER_SIZE 255
#define IMAGE 50
int keep_going = 1;
```

- Needed functions for the code to work.

```
void *recognize_image(void * arg);
void * user_interaction(void * arg);
void printDir();
```

- Main function

```
int main(){
    int status;
    void * void_varible;
    pthread_t user_thread; //thread variable
    //we start each thread checking if it was successfully created
    status = pthread_create(&user_thread, NULL, &user_interaction, &void_varible);
    if (status)
      {
        fprintf(stderr, "ERROR: pthread_create %d\n", status);
        exit(EXIT_FAILURE);
      }
    printf("Created user thread \n");
    // Wait for all other threads to finish
    pthread_exit(NULL);
    return 0;
}
```

- Menu for the user to interact with the program.

```
//Manu for the user
void * user_interaction(void * arg){
    int op = 0;
    int status;
    void * void_varible;
    pthread_t image_recognizer;

    printf("Hello, welcome to iamyoureyes\n With me you will be able to recognize
and know many new things\n");

    printf("choose the option you want\n 1.- Image recognition\n 2.- List images
directory\n 0.- exit/quit\n\n");
    scanf("%d", &op); //Waitting for the user
    //Loop the menu until a valid option is used.
    while(keep_going > 0 && keep_going < 10){
      if(op == 1){
        keep_going = 11; //Escape the loop
        status = pthread_create(&image_recognizer, NULL, &recognize_image,
&void_varible); //Call the Python file as a thread.
        if (status)
      {
        fprintf(stderr, "ERROR: pthread_create %d\n", status);
        exit(EXIT_FAILURE);
      }
        printf("Created image recognizer thread \n");
      }else if(op == 2){
        printDir();
        keep_going = 5;
      }else if(op == 0){
        keep_going = 0;
      }else{
         printf("Please chose a valid option \n");
      }
    }
}
```

- Call Python file and function that will be threaded.

```
void *recognize_image(void * arg){
    FILE * file_ptr = NULL;
    char * command1 = "python classify_image.py --image_file images/"; //File name
plus images directory specification.
    char command2[IMAGE];
    printf("please type the name of the image you want to analyze as it is (exaple:
car.jpg)\n" );
    scanf("%s", command2);
    char * command = (char *) malloc(1 + strlen(command1)+ strlen(command2) );
    strcpy(command, command1);
    strcat(command, command2);
    char buffer[BUFFER_SIZE];
    // Open the pipe
    file_ptr = popen(command, "r");
    // Validate that the pipe could be opened
    if (file_ptr != NULL)
    {
        while ( fgets(buffer, BUFFER_SIZE, file_ptr) )
        {
            printf("\t%s", buffer);
        }
        // Close the pipe
        pclose(file_ptr);
    }
    keep_going = 1;
}
```

- Print images directory to check existing files.

```
void printDir()
{
      DIR *dir; //Directory variable
      struct dirent *sd; //Library structure
      dir = opendir("./images"); //Open images directory
      //If no directory, exit
      if(dir == NULL)
      {
          printf("Error! Directory not located.\n");
          exit(1);
      }
      //As long as there is a next file, print it
      #pragma omp parallel private(sd)
      {
        while((sd=readdir(dir)) != NULL)
        {
            #pragma omp task
              rintf(">> %s\n", sd->d_name);
        }
      }
      closedir(dir); //Close directory
}
```

Python Documentation

- Needed modules for the program.

```
#Partial imports from modules
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function
#Needed Python modules
import argparse
import os.path
import re
import sys
import tarfile
#Needed modules for TensorFlow
import numpy as np
```

```
from six.moves import urllib
import tensorflow as tf
```

- Global variables

```
FLAGS = None
# pylint: disable=line-too-long
DATA_URL =
'http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz'
# pylint: enable=line-too-long
```

- Class to look up in the database

```
class NodeLookup(object):
    """Converts integer node ID's to human readable labels."""
    def __init__(self,
                 label_lookup_path=None,
                 uid_lookup_path=None):
      if not label_lookup_path:
        label_lookup_path = os.path.join(
            FLAGS.model_dir, 'imagenet_2012_challenge_label_map_proto.pbtxt')
      if not uid_lookup_path:
        uid_lookup_path = os.path.join(
            FLAGS.model_dir, 'imagenet_synset_to_human_label_map.txt')
      self.node_lookup = self.load(label_lookup_path, uid_lookup_path)
    def load(self, label_lookup_path, uid_lookup_path):
      """Loads a human readable English name for each softmax node.
      Args:
        label_lookup_path: string UID to integer node ID.
        uid_lookup_path: string UID to human-readable string.
      Returns:
        dict from integer node ID to human-readable string.
      """
      if not tf.gfile.Exists(uid_lookup_path):
        tf.logging.fatal('File does not exist %s', uid_lookup_path)
      if not tf.gfile.Exists(label_lookup_path):
        tf.logging.fatal('File does not exist %s', label_lookup_path)
      # Loads mapping from string UID to human-readable string
      proto_as_ascii_lines = tf.gfile.GFile(uid_lookup_path).readlines()
      uid_to_human = {}
      p = re.compile(r'[n\d]*[ \S,]*')
      for line in proto_as_ascii_lines:
        parsed_items = p.findall(line)
        uid = parsed_items[0]
        human_string = parsed_items[2]
        uid_to_human[uid] = human_string
      # Loads mapping from string UID to integer node ID.
      node_id_to_uid = {}
      proto_as_ascii = tf.gfile.GFile(label_lookup_path).readlines()
      for line in proto_as_ascii:
        if line.startswith('  target_class:'):
          target_class = int(line.split(': ')[1])
        if line.startswith('  target_class_string:'):
          target_class_string = line.split(': ')[1]
          node_id_to_uid[target_class] = target_class_string[1:-2]
      # Loads the final mapping of integer node ID to human-readable string
      node_id_to_name = {}
      for key, val in node_id_to_uid.items():
        if val not in uid_to_human:
          tf.logging.fatal('Failed to locate: %s', val)
        name = uid_to_human[val]
        node_id_to_name[key] = name
      return node_id_to_name
    def id_to_string(self, node_id):
      if node_id not in self.node_lookup:
        return ''
      return self.node_lookup[node_id]
```

- Create a Graph of the Brain Network

```
def create_graph():
    """Creates a graph from saved GraphDef file and returns a saver."""
    # Creates graph from saved graph_def.pb.
    with tf.gfile.FastGFile(os.path.join(
        FLAGS.model_dir, 'classify_image_graph_def.pb'), 'rb') as f:
      graph_def = tf.GraphDef()
      graph_def.ParseFromString(f.read())
      _ = tf.import_graph_def(graph_def, name='')
```

- Run the trained interface on the image to start the analysis.

```
def run_inference_on_image(image):
    """Runs inference on an image.
    Args:
      image: Image file name.
    Returns:
      Nothing
    """
    if not tf.gfile.Exists(image):
      tf.logging.fatal('File does not exist %s', image)
    image_data = tf.gfile.FastGFile(image, 'rb').read()

    # Creates graph from saved GraphDef.
    create_graph()
    with tf.Session() as sess:
      # Some useful tensors:
      # 'softmax:0': A tensor containing the normalized prediction across
      #   1000 labels.
      # 'pool_3:0': A tensor containing the next-to-last layer containing 2048
      #   float description of the image.
      # 'DecodeJpeg/contents:0': A tensor containing a string providing JPEG
      #   encoding of the image.
      # Runs the softmax tensor by feeding the image_data as input to the graph.
      softmax_tensor = sess.graph.get_tensor_by_name('softmax:0')
      predictions = sess.run(softmax_tensor,
                             {'DecodeJpeg/contents:0': image_data})
      predictions = np.squeeze(predictions)

      # Creates node ID --> English string lookup.
      node_lookup = NodeLookup()
      top_k = predictions.argsort()[-FLAGS.num_top_predictions:][::-1]
      for node_id in top_k:
        human_string = node_lookup.id_to_string(node_id)
        score = predictions[node_id]
        print('%s (score = %.5f)' % (human_string, score))
```

- Download more data if insufficient for current analysis.

```
def maybe_download_and_extract():
    """Download and extract model tar file."""
    dest_directory = FLAGS.model_dir
    if not os.path.exists(dest_directory):
      os.makedirs(dest_directory)
    filename = DATA_URL.split('/')[-1]
    filepath = os.path.join(dest_directory, filename)
    if not os.path.exists(filepath):
      def _progress(count, block_size, total_size):
        sys.stdout.write('\r>> Downloading %s %.1f%%' % (
            filename, float(count * block_size) / float(total_size) * 100.0))
        sys.stdout.flush()
      filepath, _ = urllib.request.urlretrieve(DATA_URL, filepath, _progress)
      print()
      statinfo = os.stat(filepath)
      print('Successfully downloaded', filename, statinfo.st_size, 'bytes.')
    tarfile.open(filepath, 'r:gz').extractall(dest_directory)
```

- Python main function

```
def main(_):
    maybe_download_and_extract()
    image = (FLAGS.image_file if FLAGS.image_file else
             os.path.join(FLAGS.model_dir, 'cropped_panda.jpg'))
    run_inference_on_image(image)
```

- Call main function if no parsed arguments are introduced (Ex. -h). Here we also define the required and optional arguments.

```
if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    # classify_image_graph_def.pb:
    #   Binary representation of the GraphDef protocol buffer.
    # imagenet_synset_to_human_label_map.txt:
    #   Map from synset ID to a human readable string.
    # imagenet_2012_challenge_label_map_proto.pbtxt:
    #   Text representation of a protocol buffer mapping a label to synset ID.
    parser.add_argument(
        '--model_dir',
        type=str,
        default='/tmp/imagenet',
        help="""\
        Path to classify_image_graph_def.pb,
        imagenet_synset_to_human_label_map.txt, and
        imagenet_2012_challenge_label_map_proto.pbtxt.\
        """
    )
    parser.add_argument(
        '--image_file',
        type=str,
        default='',
        help='Absolute path to image file.'
    )
    parser.add_argument(
        '--num_top_predictions',
        type=int,
        default=5,
        help='Display this many predictions.'
    )
    FLAGS, unparsed = parser.parse_known_args()
    tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```