Configurable Convolutional Neural Networks for classifying images using TensorFlow

Yupeng Wang, Ph.D., Data Scientist

Overview

Convolutional Neural Networks (CNNs), a type of deep learning frameworks, are very powerful in image recognition. TensorFlow has become a popular tool for applying deep learning models. Although there are a bunch of tutorials on the TensorFlow website, it is difficult to apply those tutorials to new problems. In real applications, deep learning models should be continuously tuned until an effectiveness criterion is reached. Thus, modifying source code to adjust model parameters is inconvenient. I developed a toolkit which can take in a configuration file and then classify different images accordingly. Wrapper scripts are also provided to enhance convenience.

In this demo, I build a CNN model to classify different animals including cats, dogs and cows. The source code can be downloaded from https://github.com/wyp1125/cCNN-Image-Classifier.

Key techniques: deep learning, convolutional neural network, python, TensorFlow, bash, wrapper, json, numpy, logging, image recognition, OpenCV

Raw data

Raw data are image files in "jpg" format. The image files are assumed to have at least one cat, dog or cow which can be clearly distinguished by eyes.



Images of any size are fine at the beginning and I will have a procedure of preprocessing. For each class of animals, at least several hundreds of files are needed. Here I use the URLs provided by ImageNet (http://www.image-net.org). Each line is in the format:

```
GroupID_ImageID  URL
```

The map between GroupIDs and classes can be found from its website. I made a python script (`batch_image_download.py`) to download all the images for a group ID.

```
import sys
import subprocess
import os
if len(sys.argv)<3:
   print("group_id output_dir start_id")
   quit() sta=0
if sys.argv[3]!='':
   sta=int(sys.argv[3])
if os.path.isdir(sys.argv[2])==False:
   os.mkdir(sys.argv[2])
cmd="grep "+sys.argv[1]+" all.urls"
p=subprocess.Popen(cmd,shell=True,stderr=subprocess.PIPE,stdout=subprocess.PIPE)
out,err=p.communicate()
urls=out.strip().split("\n")
n=0
for line in urls:
   if n>=sta:
      word=line.strip().split("\t")
      cmd1="wget "+word[-1]+" -O "+sys.argv[2]+"/"+str(n)+".jpg"
      p1=subprocess.Popen(cmd1,shell=True,stderr=subprocess.PIPE,stdout=subprocess.PIPE)
      out1,err1=p1.communicate()
   n=n+1
   print(n)
```

Some downloaded image files may be broken, too small, or without 3 channels. Thus, the images should be further filtered by the filter_images.py python script.

```
import sys
import subprocess
import os
import cv2
if len(sys.argv)<3:
   print("input_folder output_folder #num")
   quit()
if not os.path.exists(sys.argv[2]):
   os.makedirs(sys.argv[2])
cmd="ls "+sys.argv[1]
p=subprocess.Popen(cmd,shell=True,stderr=subprocess.PIPE,stdout=subprocess.PIPE)
out,err=p.communicate()
fls=out.rstrip("\r\n").split("\n")
n=0
for fl in fls:
   pth=sys.argv[1]+"/"+fl
   img=cv2.imread(pth)
   try:
      ht,wt,ch=img.shape
      if ht>=200 and wt>=200 and ch==3:
         os.system("cp "+pth+" "+sys.argv[2]+"/"+str(n)+".jpg")
         n+=1
      if n>=int(sys.argv[3]):
         break
   except:
      continue
print n
```

Description of the software

All the parameter settings for building a CNN model should be provided in a ".json" file.

```
{
  "classes":{
     "dogs":"/home/yupeng/bdx/image_recognition/cCNNImageClassifier_dev/training_data/dogs",
     "cats":"/home/yupeng/bdx/image_recognition/cCNNImageClassifier_dev/training_data/cats",
     "cows":"/home/yupeng/bdx/image_recognition/cCNNImageClassifier_dev/training_data/cows"},
  "train":{
     "validation_size":0.2,
     "batch_size":32,
     "img_size":128,
     "channel":3,
     "run_dir":"/home/yupeng/bdx/image_recognition/cCNNImageClassifier_dev/run1",
     "learning_rate":0.0001,
     "num_iterations":3000,
     "optimizer":"Adam",
     "model_name":"cat_doc_cow_classifier"},
  "model":{
     "cv1":{
        "type":"convolutional",
        "input":"x",
        "filter_size":3,
        "num_filters":32,
        "activation":"relu",
        "pooling":"max_pool",
        "win_strd_size":2},
     "cv2":{
        "type":"convolutional",
        "input":"cv1",
        "filter_size":3,
        "num_filters":32,
        "activation":"relu",
        "pooling":"max_pool",
        "win_strd_size":2},
     "cv3":{
        "type":"convolutional",
        "input":"cv2",
        "filter_size":3,
        "num_filters":64,
        "activation":"relu",
        "pooling":"max_pool",
        "win_strd_size":2},
     "flt":{
        "type":"flatten",
        "input":"cv3"},
     "fc1":{
        "type":"fc",
        "input":"flt",
        "num_outputs":128,
        "activation":"relu"},
     "fc2":{
        "type":"fc",
        "input":"fc1"}
  }
}
```

Most parameters in the ".json" file are self-explanatory. The CNN model should be configured under the "model" section. The naming of a layer is flexible, but the layer type must be chosen from "convolutional", "flattern" or "fc" (i.e. fully connected). Layers can be conveniently added or deleted, provided that a sequence of layers is established: for the first layer, the input must be "x", while for subsequent layers, the input is the name of the previous layer.

The next step is to train the specified CNN model using TensorFlow. There are several checkups of the Linux environment prior to running TensorFlow, so I made a wrapper script (runCNN.sh) to invoke the multiple steps for the training at once.

```
#!/bin/bash
usage() {
cat << EOF
usage: $0 options
This script runs a customized CNN model to distinguish images with different classes of objects. A
configuration json file should be supplied.
OPTIONS:
   -h help, Show this message
   -i configuration json file (required)
EOF
}
if [ "$#" -eq "0" ]; then
   usage;
   exit;
fi
while getopts "hi:" OPTION
do
   case $OPTION in
      h) usage ; exit 1 ;;
      i) json=$OPTARG ;;
      ?) usage ; exit ;;
   Esac
done
if [ ! -f $json ]; then
   echo "The configuration json file does not exist!";
   exit;
fi
PYTHON3=/usr/bin/python3
if ! [ -x "$PYTHON3" ]; then
   echo "Python3 could not be found!";
   exit;
fi
tf_activate=/home/yupeng/tensorflow/bin/activate
if ! [ -f "$tf_activate" ]; then
   echo "TensorFlow could not be found!";
   exit;
fi
source ${tf_activate}
echo "Tensorflow is activated!"
${PYTHON3} train_cnn.py $json
```

To run the wrapper, the path to the json file should be provided using the "-i" option. The wrapper primarily calls the train_cnn.py program, which itself will further call dataset.py and layer_function.py. These three python programs are the core programs to build the CNN model. Because these python programs are a little long, I do not show the source code here. Actually, the source code can be found from my Github repo:

train_cnn.py: https://github.com/wyp1125/cCNN-Image-Classifier/blob/master/train_cnn.py
dataset.py: https://github.com/wyp1125/cCNN-Image-Classifier/blob/master/dataset.py
layer_function.py: https://github.com/wyp1125/cCNN-Image-Classifier/blob/master/layer_function.py

When the programs are executed, the screen outputs the shapes of different layers:

```
2018-01-21 08:45:18,053 - INFO - Create neural networks according to configuration json file
2018-01-21 08:45:18,053 - INFO - Layer: cv1
2018-01-21 08:45:18,061 - INFO - Shape: [None, 64, 64, 32]
2018-01-21 08:45:18,061 - INFO - Layer: cv2
2018-01-21 08:45:18,068 - INFO - Shape: [None, 32, 32, 32]
2018-01-21 08:45:18,068 - INFO - Layer: cv3
2018-01-21 08:45:18,075 - INFO - Shape: [None, 16, 16, 64]
2018-01-21 08:45:18,075 - INFO - Layer: flt
2018-01-21 08:45:18,076 - INFO - Shape: [None, 16384]
2018-01-21 08:45:18,076 - INFO - Layer: fc1
2018-01-21 08:45:18,082 - INFO - Shape: [None, 128]
2018-01-21 08:45:18,082 - INFO - Layer: fc2
2018-01-21 08:45:18,087 - INFO - Shape: [None, 3]
```

and the training and validation accuracy at each epoch:

```
2018-01-21 08:45:19,281 - INFO - Training Epoch 1 of 82 - Training Accuracy: 40.6%, Validation Accuracy: 31.2%, Validation Loss: 1.099
2018-01-21 08:45:39,529 - INFO - Training Epoch 2 of 82 - Training Accuracy: 37.5%, Validation Accuracy: 43.8%, Validation Loss: 1.044
2018-01-21 08:45:59,888 - INFO - Training Epoch 3 of 82 - Training Accuracy: 37.5%, Validation Accuracy: 40.6%, Validation Loss: 1.057
2018-01-21 08:46:20,313 - INFO - Training Epoch 4 of 82 - Training Accuracy: 40.6%, Validation Accuracy: 43.8%, Validation Loss: 1.056
2018-01-21 08:46:40,663 - INFO - Training Epoch 5 of 82 - Training Accuracy: 37.5%, Validation Accuracy: 53.1%, Validation Loss: 0.936
2018-01-21 08:47:01,036 - INFO - Training Epoch 6 of 82 - Training Accuracy: 40.6%, Validation Accuracy: 43.8%, Validation Loss: 0.975
2018-01-21 08:47:21,317 - INFO - Training Epoch 7 of 82 - Training Accuracy: 46.9%, Validation Accuracy: 59.4%, Validation Loss: 0.855
2018-01-21 08:47:41,692 - INFO - Training Epoch 8 of 82 - Training Accuracy: 43.8%, Validation Accuracy: 46.9%, Validation Loss: 0.794
2018-01-21 08:48:02,077 - INFO - Training Epoch 9 of 82 - Training Accuracy: 50.0%, Validation Accuracy: 56.2%, Validation Loss: 0.756
```

The model training completed at the 82nd epoch, with training accuracy of 100% and validation accuracy of 56.2%. Note that this validation accuracy is not high enough for real applications, but is adequate to demonstrate that the model is functional correctly because the random chance is only 33.3%. Actually, fine-tuning a CNN model can be a time-consuming task.

After the CNN model is built, I can implement the model to classify new images. Here, I still restrict that the images for prediction have at least one cat, cow or dog. For the prediction stage, I also made a wrapper script predCNN.sh. The wrapper should be fed with the json file ("-i" option), a folder containing new images ("-f" option), and an output file ("-o option).

```
#!/bin/bash
usage() {
cat << EOF
usage: $0 options
This script uses an established CNN model for different classes of objects to distinguish any new images. The configuration json file for the CNN model should be supplied.
OPTIONS:
   -h help, Show this message
   -i configuration json file (required)
   -f folder containing new images for classification (required)
   -o output file (required)
EOF
}
if [[ "$#" -lt 6 ]]; then
   usage;
   exit;
fi
while getopts "hi:f:o:" OPTION
do
   case $OPTION in
      h) usage ; exit 1 ;;
      i) json=$OPTARG ;;
      f) folder=$OPTARG ;;
      o) out_file=$OPTARG ;;
      ?) usage ; exit ;;
   esac
done
if [ ! -f $json ]; then
   echo "The configuration json file does not exist!";
   exit;
fi
if [ ! -d $folder ]; then
   echo "The folder containing new images does not exist!";
   exit;
fi
PYTHON3=/usr/bin/python3
if ! [ -x "$PYTHON3" ]; then
   echo "Python3 could not be found!";
   exit;
fi
tf_activate=/home/yupeng/tensorflow/bin/activate
if ! [ -f "$tf_activate" ]; then
   echo "TensorFlow could not be found!";
   exit;
fi
source ${tf_activate}
echo "Tensorflow is activated!"
${PYTHON3} predict_cnn.py $json $folder $out_file
```

I created a folder containing 500 images (none of the training images should be used), and ran the prediction wrapper script on this folder. A portion of the output file is displayed below:

```
cow.610.jpg     [  4.90840583e-04   9.71148729e-01   2.83603705e-02]    cows
cat.1022.jpg    [  6.42051399e-01   5.53177553e-04   3.57395440e-01]    cats
dog.1186.jpg    [  1.31079897e-01   2.36758133e-04   8.68683398e-01]    dogs
cat.1101.jpg    [  9.87142980e-01   1.78001355e-04   1.26789743e-02]    cats
dog.1191.jpg    [ 0.04813955  0.00427994  0.94758052]   dogs
cat.1007.jpg    [  8.04442689e-02   3.53243413e-05   9.19520378e-01]    dogs
cow.666.jpg     [  1.36678091e-05   9.21759844e-01   7.82265961e-02]    cows
cat.1094.jpg    [  9.98448968e-01   5.91753778e-05   1.49190275e-03]    cats
dog.1020.jpg    [ 0.77361315  0.21800408  0.00838282]   cats
cat.1036.jpg    [ 0.02397376  0.92811418  0.04791206]   cows
dog.1182.jpg    [ 0.58077896  0.41682914  0.0023919 ]   cats
dog.1167.jpg    [  9.29247200e-01   1.02781551e-05   7.07424805e-02]    cats
cow.633.jpg     [  2.54619431e-06   9.99863744e-01   1.33783513e-04]    cows
cow.628.jpg     [ 0.04605798  0.31665161  0.63729036]   dogs
cow.636.jpg     [  2.87734565e-05   9.97675121e-01   2.29612342e-03]    cows
dog.1169.jpg    [  6.45254433e-01   1.78403585e-04   3.54567260e-01]    cats
cat.1077.jpg    [ 0.20285824  0.02803275  0.76910901]   dogs
cat.1119.jpg    [ 0.04277043  0.00384043  0.95338917]   dogs
dog.1105.jpg    [  5.21536767e-01   6.58958525e-05   4.78397280e-01]    cats
```

In the output file, the first column contains the names of the images for prediction; the last column contains the predicted class; while the columns in the middle are the probabilities of each class.