TensorFlow train and test own data images

Train and test our own images and classify them.

1 . go to git huh url: <https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/image_retraining>

And clone git hub repogitory to your local machine:

<https://github.com/tensorflow/tensorflow.git>

2. go to E:\Git1\tensorflow\tensorflow\examples\image\_retraining folder in mahine.

* Create road\_images and bottlenecks folder in image\_retraining
* And create sub folders in road\_images folder
* Sub folder are bridge, junction and straightroad
* Paste the respective training data into those sub folder , should minimum 20 images

3. refer this url: <https://codelabs.developers.google.com/codelabs/tensorflow-for-poets/#4>

4. keep the test data images into image\_retraining folder

5. run this command for traing own images in the folder road\_images

python retrain.py --bottleneck\_dir=bottlenecks --how\_many\_training\_steps=500 --model\_dir=inception --summaries\_dir=training\_summaries/basic --output\_graph=retrained\_graph.pb --output\_labels=retrained\_labels.txt --image\_dir=road\_images

6. run this command for test your data in image\_retraining folder

python label\_image.py --image 19.jpg --graph retrained\_graph.pb --labels retrained\_labels.txt

TensorFlow Image Captioning

https://github.com/tensorflow/models/tree/master/im2txt#architecture

Keras Setup

1. Fallow the below link for Keras setup.

<https://stackoverflow.com/questions/34097988/how-do-i-install-keras-and-theano-in-anaconda-python-on-windows>

2. Install git and set git path to environmental variables:

<https://stackoverflow.com/questions/29957484/how-to-pip-install-from-github-powershell-gives-error-cannot-find-command-gi>

3. Setting the git path

http://www.chambaud.com/2013/07/08/adding-git-to-path-when-using-github-for-windows/

**NLP:**

Code github :

https://github.com/PythonProgramming/NLTK-3----Natural-Language-Processing-with-Python-series

Installing nltk via pip:

1. python -m pip install nltk
2. python -m pip install numpy
3. pip install opencv-python
4. python -m pip install scikit-image

solving errors from this link:

https://stackoverflow.com/questions/26693736/nltk-and-stopwords-fail-lookuperror

1. Method 1 for NLTK installation

] python

>>>> import nltk

>>> nltk.download()

1. Method for NLTK installation

>> pip install nltk

>> python -m nltk.downloader all

This downloads all the nltk data. You can also specify a directory with the option -d. For example

>> python -m nltk.downloader -d /usr/local/share/nltk\_data all

Well, if you are using Python 3, you can replace python in the above command to python3and pip to pip3.

**Project 1: Extracting the actionable sentences from Huge text file**

from nltk.tag import pos\_tag

from nltk.chunk import RegexpParser

import nltk

import random

from nltk.corpus import movie\_reviews

from nltk.tokenize import sent\_tokenize, word\_tokenize

from nltk.corpus import state\_union

from nltk.tokenize import PunktSentenceTokenizer

import shutil

import sys

import io

txt\_opn = open("action.txt",newline='',encoding='utf8')

text1 = txt\_opn.read()

text1 = text1.rstrip()

#print(text1)

words = text1.split()

sent = [w for w in words if not w == 'computer']

sentence = pos\_tag(sent)

sentences1 = sent\_tokenize(text1)

all\_words1 = []

query = [e1 for (e1, rel) in sentence if rel=='VB']

print("actions-start===================================================================================")

print(query)

print("actions-end ====================================================================================")

for w in sentences1:

for j in query:

if j in w:

all\_words1.append(w)

#print("\n")

#print(query)

all\_words1 = set(all\_words1)

#print(all\_words1)

j = 0

for word in all\_words1:

j = j + 1

print(str(j) +". "+ word)

OpenCv:

Installation :

1. pip install opencv-python

links:

1. <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html>
2. <http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html>
3. http://opencv-python-tutroals.readthedocs.io/en/latest/py\_tutorials/py\_feature2d/py\_table\_of\_contents\_feature2d/py\_table\_of\_contents\_feature2d.html

code: finding corner detections

import numpy as np

import cv2

img = cv2.imread('sample2.png')

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

gray = np.float32(gray)

corners = cv2.goodFeaturesToTrack(gray, 100, 0.01, 10)

corners = np.int0(corners)

for corner in corners:

x,y = corner.ravel()

cv2.circle(img,(x,y),3,255,-1)

cv2.imwrite('subpixel5.png',img)

cv2.imshow('Corner',img)

cv2.waitKey(10000)

cv2.destroyAllWindows()

code : findind edge detection:

import cv2

import numpy as np

filename = 'sample2.png'

img = cv2.imread(filename)

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

gray = np.float32(gray)

dst = cv2.cornerHarris(gray,2,3,0.08)

#result is dilated for marking the corners, not important

dst = cv2.dilate(dst,None)

# Threshold for an optimal value, it may vary depending on the image.

img[dst>0.01\*dst.max()]=[0,0,255]

cv2.imshow('dst',img)

if cv2.waitKey(0) & 0xff == 27:

cv2.destroyAllWindows()

imp link:

https://pythonprogramming.net/drawing-writing-python-opencv-tutorial/

**Clustring**

**K Means Clustering:**

setwd("D:/data")

getwd()

data = read.csv("data.csv")

View(data)

library(ggplot2)

mydata <- data

wss <- (nrow(mydata)-1)\*sum(apply(mydata,2,var))

for (i in 2:15) wss[i] <- sum(kmeans(mydata,

centers=i)$withinss)

plot(1:15, wss, type="b", xlab="Number of Clusters",

ylab="Within groups sum of squares",

main="Assessing the Optimal Number of Clusters with the Elbow Method",

pch=20, cex=2)

set.seed(7)

km2 = kmeans(data, 6, nstart=100)

plot(data, col =(km2$cluster +1) , main="K-Means result with 6 clusters", pch=20, cex=2)

**SCClust:**

**Project 1: Find the specific size clusters using the SCClustering**

setwd("D:/data")

getwd()

data = read.csv("Associate\_lat\_lon2.csv")

library(ggplot2)

library(scclust)

my\_data <- data[,c(-1,-2,-3)]

mydata1 <- my\_data

# Construct distance metric

my\_dist <- distances(my\_data,

dist\_variables = c("Lat", "Lon"))

# Make clustering with at least 3 data points in each cluster

my\_clustering <- sc\_clustering(my\_dist, 16)

# Check so clustering satisfies constraints

check\_clustering(my\_clustering, 16)

# > TRUE

# Get statistics about the clustering

get\_clustering\_stats(my\_dist, my\_clustering)

# Make clustering with at least 8 points in total of which at least

# one must be "A", two must be "B" and five can be any type

my\_refined\_clustering <- hierarchical\_clustering(my\_dist,

size\_constraint = 16,

existing\_clustering = my\_clustering)

# Make clustering from scratch with `hierarchical\_clustering`

my\_other\_clustering <- hierarchical\_clustering(my\_dist, 16)

data$group <- my\_other\_clustering

write.csv(data,"Sol\_Associate\_lat\_lon.csv")

#Ploting

plot(my\_other\_clustering)

points(my\_other\_clustering, col = 1:65, pch = 8, cex = 2)

#frequency finding

table(my\_refined\_clustering)

table(my\_other\_clustering)