CS5542-Big data Analytics and Applications

LAB ASSIGNMENT-4 REPORT

Name: Lakshmi Korrapati

ID: 14

Objective:

To implement a generate caption for an image in bottom-up model.

Introduction:

Caption generator is the difficult computerized reasoning issue utilizing NLP system and PC vision. It requires the two pictures understanding and language appear from the field of Natural language preparing. Without a doubt, a delineation must catch the articles contained in an image, yet it moreover should express how these things relate to each other, similarly as their qualities and the activities they are related with.

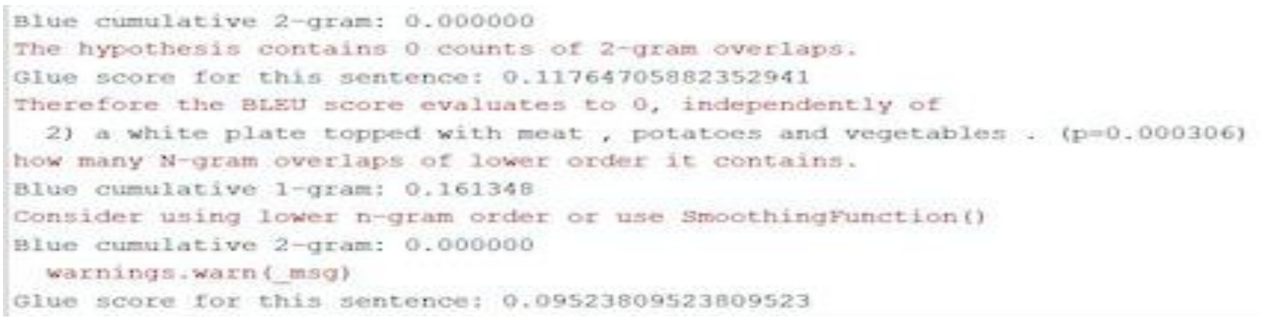
Technologies:

- Pycharm – A python IDE

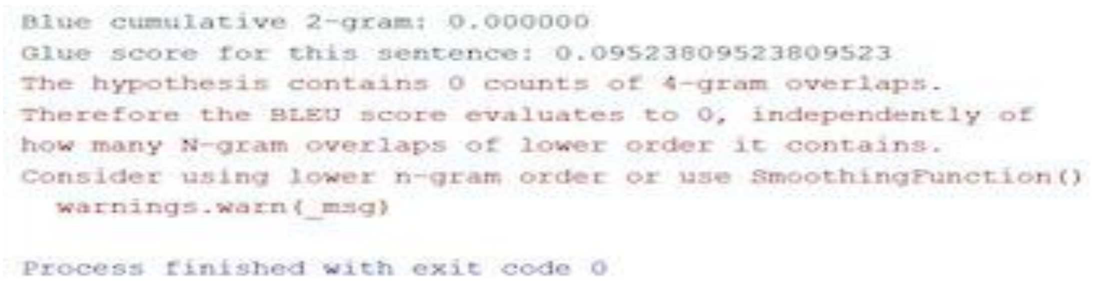
Libraries:

- Tensorflow r1.0
- NLTK
- pandas
- MSCOCO images and captions.
- InceptionV4
- PIL

Result:

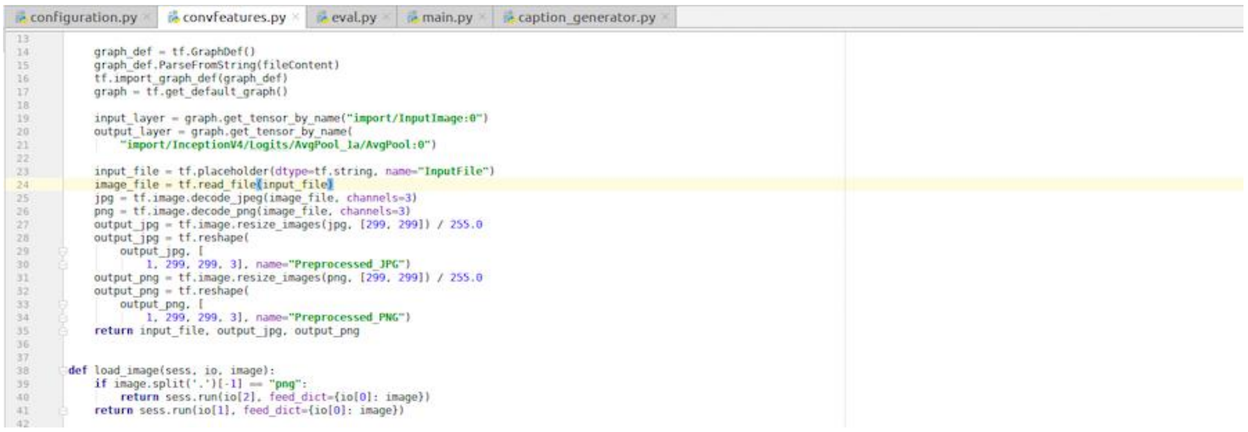


```
Blue cumulative 2-gram: 0.000000
The hypothesis contains 0 counts of 2-gram overlaps.
Glue score for this sentence: 0.11764705882352941
Therefore the BLEU score evaluates to 0, independently of
  2) a white plate topped with meat , potatoes and vegetables . (p=0.000306)
how many N-gram overlaps of lower order it contains.
Blue cumulative 1-gram: 0.161348
Consider using lower n-gram order or use SmoothingFunction()
Blue cumulative 2-gram: 0.000000
  warnings.warn(_msg)
Glue score for this sentence: 0.09523809523809523
```

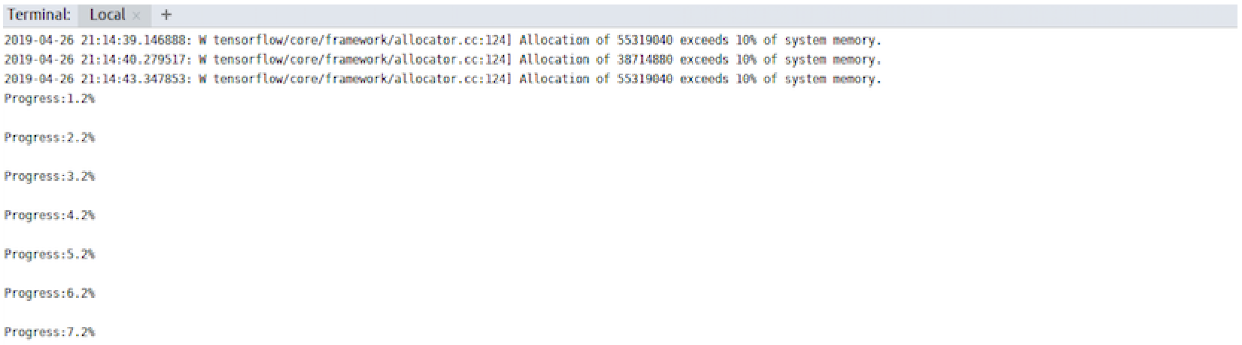


```
Blue cumulative 2-gram: 0.000000
Glue score for this sentence: 0.09523809523809523
The hypothesis contains 0 counts of 4-gram overlaps.
Therefore the BLEU score evaluates to 0, independently of
how many N-gram overlaps of lower order it contains.
Consider using lower n-gram order or use SmoothingFunction()
  warnings.warn(_msg)

Process finished with exit code 0
```

MSCOCO dataset for the generator of highlights document:

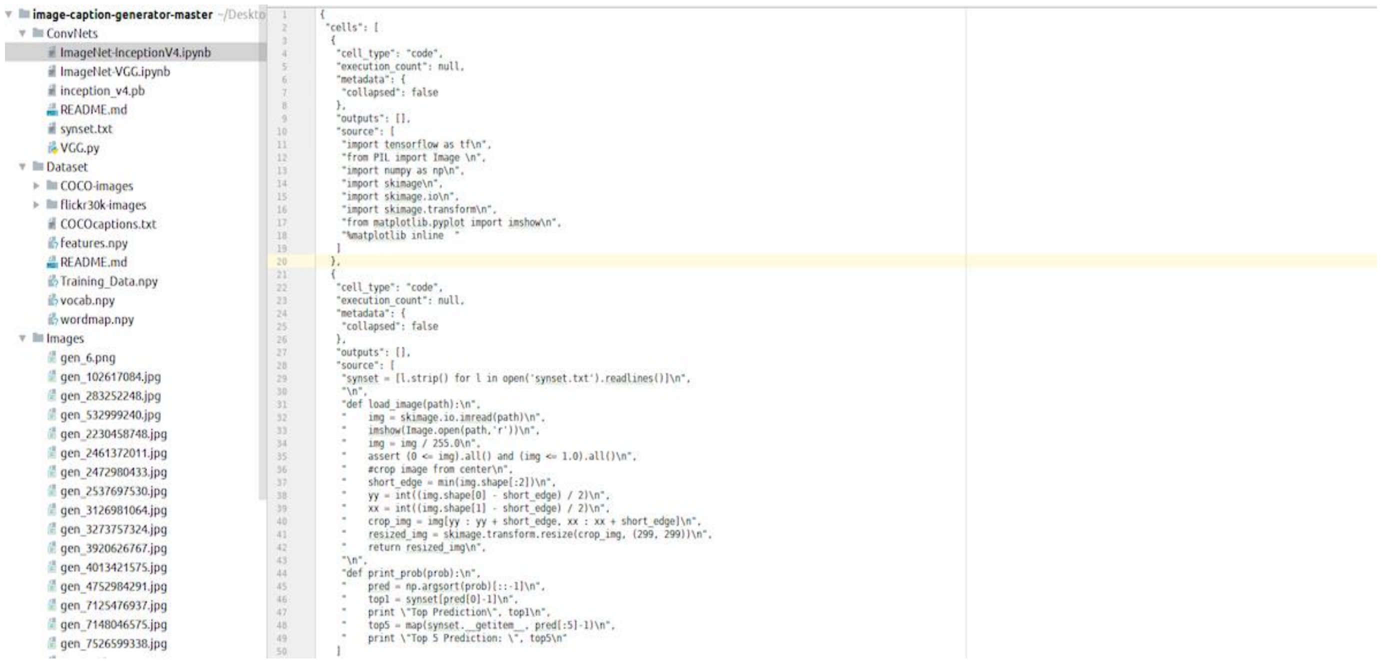


```python

    graph_def = tf.GraphDef()
    graph_def.ParseFromString(fileContent)
    tf.import_graph_def(graph_def)
    graph = tf.get_default_graph()

    input_layer = graph.get_tensor_by_name("import/InputImage:0")
    output_layer = graph.get_tensor_by_name(
        "import/InceptionV4/Logits/AvgPool_1a/AvgPool:0")

    input_file = tf.placeholder(dtype=tf.string, name="InputFile")
    image_file = tf.read_file(input_file)
    jpg = tf.image.decode_jpeg(image_file, channels=3)
    png = tf.image.decode_png(image_file, channels=3)
    output_jpg = tf.image.resize_images(jpg, [299, 299]) / 255.0
    output_jpg = tf.reshape(
        output_jpg, [
            1, 299, 299, 3], name="Preprocessed_JPG")
    output_png = tf.image.resize_images(png, [299, 299]) / 255.0
    output_png = tf.reshape(
        output_png, [
            1, 299, 299, 3], name="Preprocessed_PNG")
    return input_file, output_jpg, output_png


def load_image(sess, io, image):
    if image.split('.')[-1] == "png":
        return sess.run(io[2], feed_dict={io[0]: image})
    return sess.run(io[1], feed_dict={io[0]: image})

```



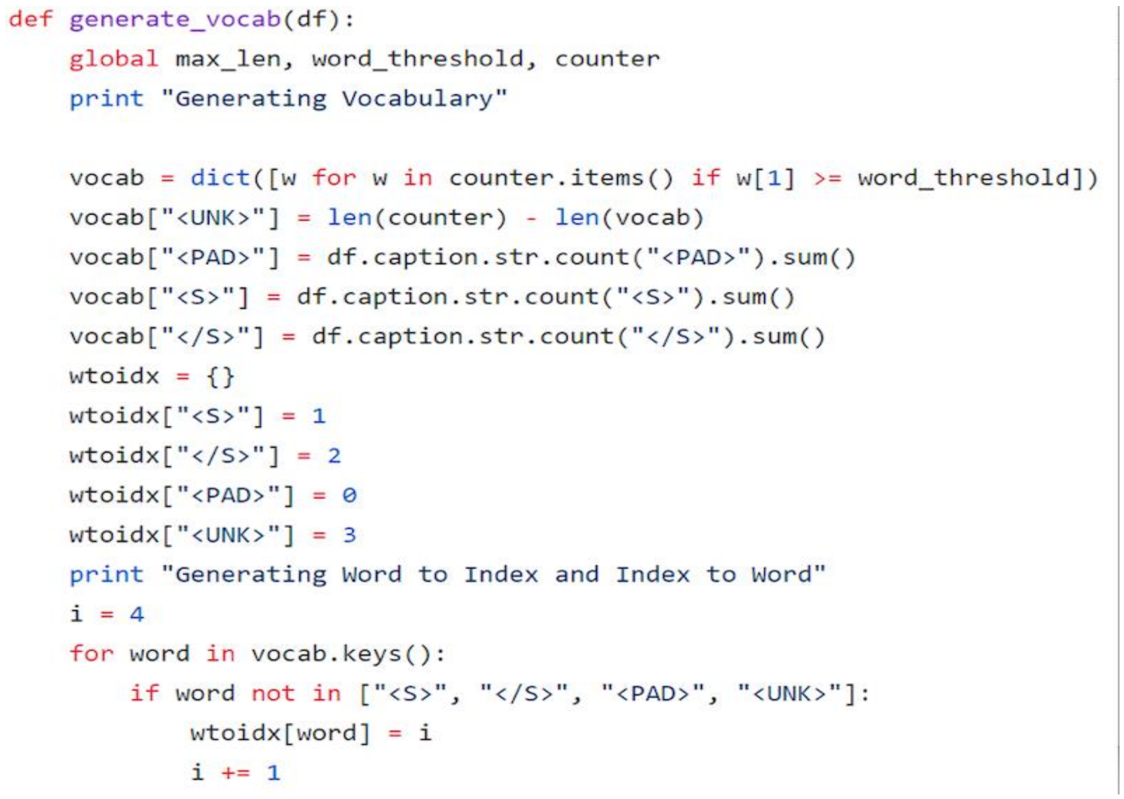
```
2019-04-26 21:14:39.146888: W tensorflow/core/framework/allocator.cc:124] Allocation of 55319040 exceeds 10% of system memory.
2019-04-26 21:14:40.279517: W tensorflow/core/framework/allocator.cc:124] Allocation of 38714880 exceeds 10% of system memory.
2019-04-26 21:14:43.347853: W tensorflow/core/framework/allocator.cc:124] Allocation of 55319040 exceeds 10% of system memory.
Progress:1.2%

Progress:2.2%

Progress:3.2%

Progress:4.2%

Progress:5.2%

Progress:6.2%

Progress:7.2%
```

pretrained inception_v4 model:



```
{
 "cells": [
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
   "source": [
    "import tensorflow as tf\n",
    "from PIL import Image \n",
    "import numpy as np\n",
    "import skimage\n",
    "import skimage.io\n",
    "import skimage.transform\n",
    "from matplotlib.pyplot import imshow\n",
    "%matplotlib inline  "
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "collapsed": false
   },
   "outputs": [],
   "source": [
    "synset = [l.strip() for l in open('synset.txt').readlines()]\n",
    "\n",
    "def load_image(path):\n",
    "    img = skimage.io.imread(path)\n",
    "    imshow(Image.open(path,'r'))\n",
    "    img = img / 255.0\n",
    "    assert (0 <= img).all() and (img <= 1.0).all()\n",
    "    #crop image from center\n",
    "    short_edge = min(img.shape[:2])\n",
    "    yy = int((img.shape[0] - short_edge) / 2)\n",
    "    xx = int((img.shape[1] - short_edge) / 2)\n",
    "    crop_img = img[yy : yy + short_edge, xx : xx + short_edge]\n",
    "    resized_img = skimage.transform.resize(crop_img, (299, 299))\n",
    "    return resized_img\n",
    "\n",
    "def print_prob(prob):\n",
    "    pred = np.argsort(prob)[::-1]\n",
    "    top1 = synset[pred[0]-1]\n",
    "    print \"Top Prediction\", top1\n",
    "    top5 = map(synset.__getitem__, pred[:5]-1)\n",
    "    print \"Top 5 Prediction: \", top5\n"
   ]
```

Features.npy: **Vocabulary file:**



```
def generate_vocab(df):
    global max_len, word_threshold, counter
    print "Generating Vocabulary"

    vocab = dict([w for w in counter.items() if w[1] >= word_threshold])
    vocab["<UNK>"] = len(counter) - len(vocab)
    vocab["<PAD>"] = df.caption.str.count("<PAD>").sum()
    vocab["<S>"] = df.caption.str.count("<S>").sum()
    vocab["</S>"] = df.caption.str.count("</S>").sum()
    wtoidx = {}
    wtoidx["<S>"] = 1
    wtoidx["</S>"] = 2
    wtoidx["<PAD>"] = 0
    wtoidx["<UNK>"] = 3
    print "Generating Word to Index and Index to Word"
    i = 4
    for word in vocab.keys():
        if word not in ["<S>", "</S>", "<PAD>", "<UNK>"]:
            wtoidx[word] = i
            i += 1
```

BLEU: Model Generating Captions

```
for fil in required_files:
    if not os.path.isfile('Dataset/' + fil + ".npy"):
        generate = True
        print "Required Files not present. Regenerating Data."
        break
if not generate:
    print "Dataset Present; Skipping Generation."
    return get_data(required_files)
global max_len, word_threshold, counter
max_len = ml
word_threshold = wt
print "Loading Caption Data", cap_path
if data_is_coco:
    # Prepare COCO captions in Flickr format
    cap_path = prepare_coco_captions(cap_path)
    # Load the COCO captions data
    with open(cap_path, 'r') as f:
        data = f.readlines()
    filenames = [caps.split('\t')[0].split('#')[0] for caps in data]
    captions = [caps.split('\t')[1] for caps in data]
    df = preprocess_coco_captions(filenames, captions)
else:
    with open(cap_path, 'r') as f:
        data = f.readlines()
    filenames = [caps.split('\t')[0].split('#')[0] for caps in data]
    captions = [caps.replace('\n', '').split('\t')[1] for caps in data]
    df = preprocess_flickr_captions(filenames, captions)
```