Lab 2 – Image Caption Generator

Report

By

Akhil Yerrapragada (akhily@kth.se)

Gibson Chikafa (chikafa@kth.se)

Group Name – gibson_akhil

Problem Description:

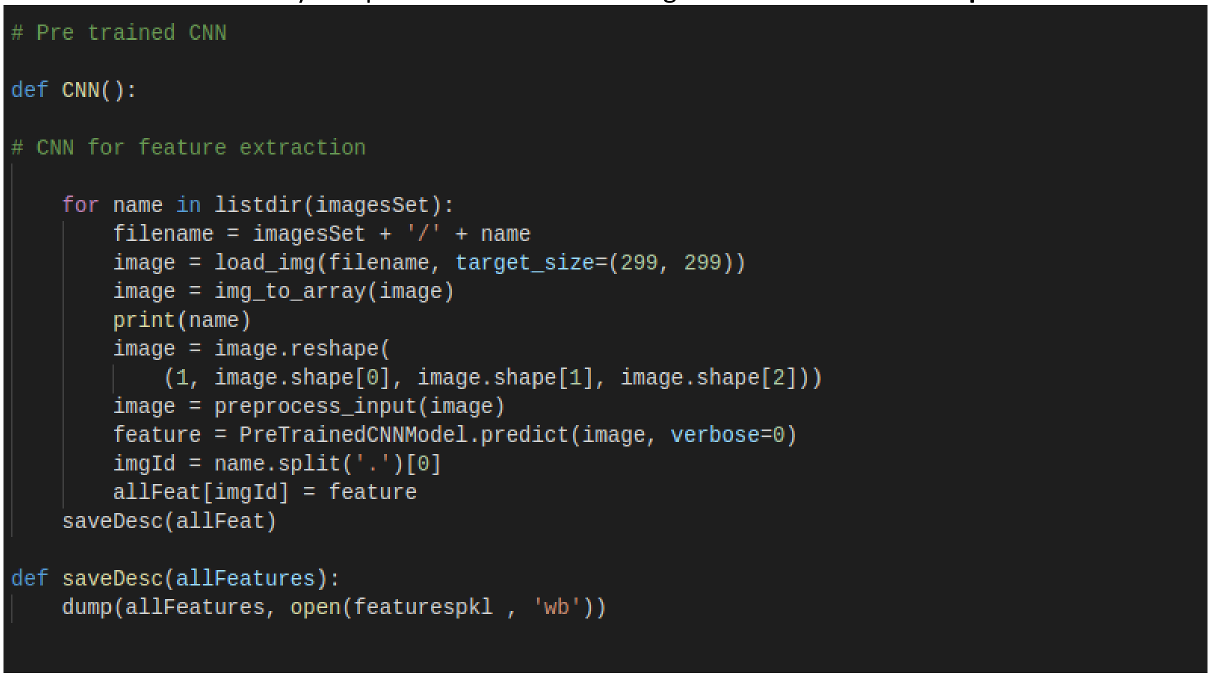
To implement an image caption generator as described in the paper using CNN, RNN and Tensorflow.

Data:

The dataset used for the implementation is Flicker8k_Dataset which contains images with descriptions, training and testing sets which can be used for respective purposes.

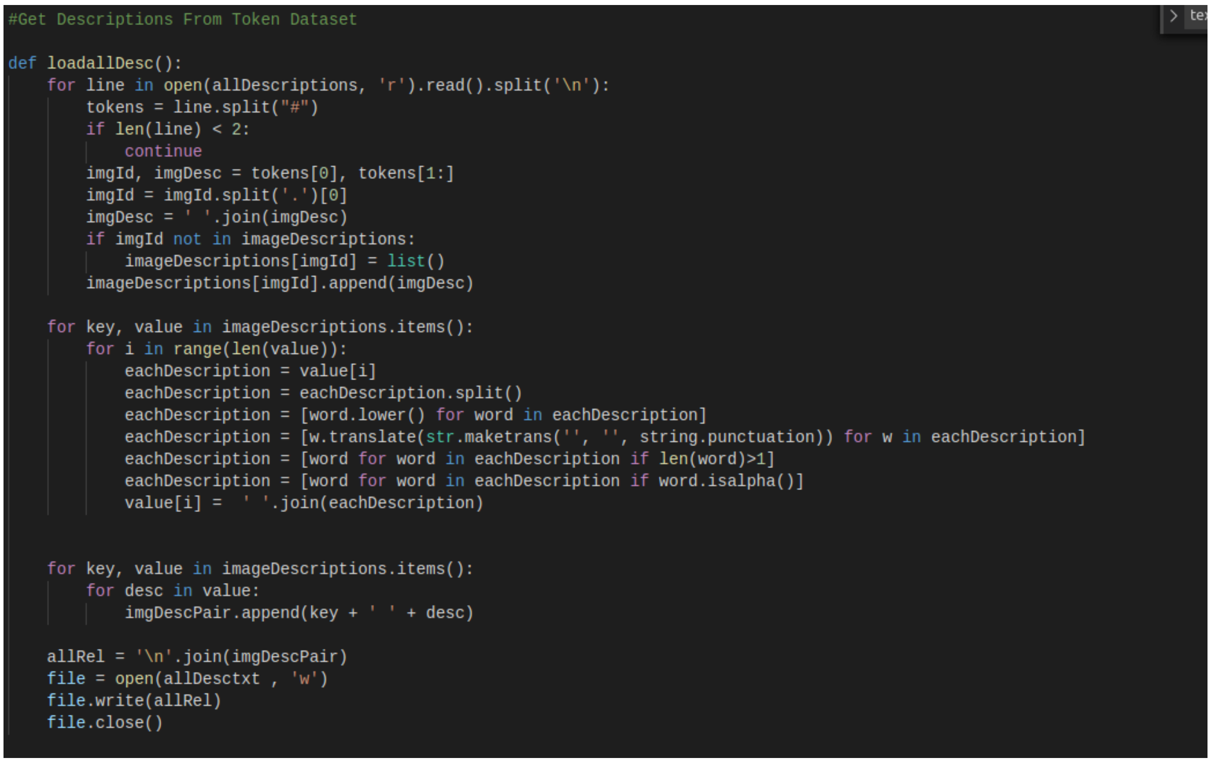
Implementation:

1) A pretrained model, **InceptionV3** provided by keras is used for to extract the features from images.
2) The features extracted by Inception V3 from all the images are saved in **features.pkl** file.



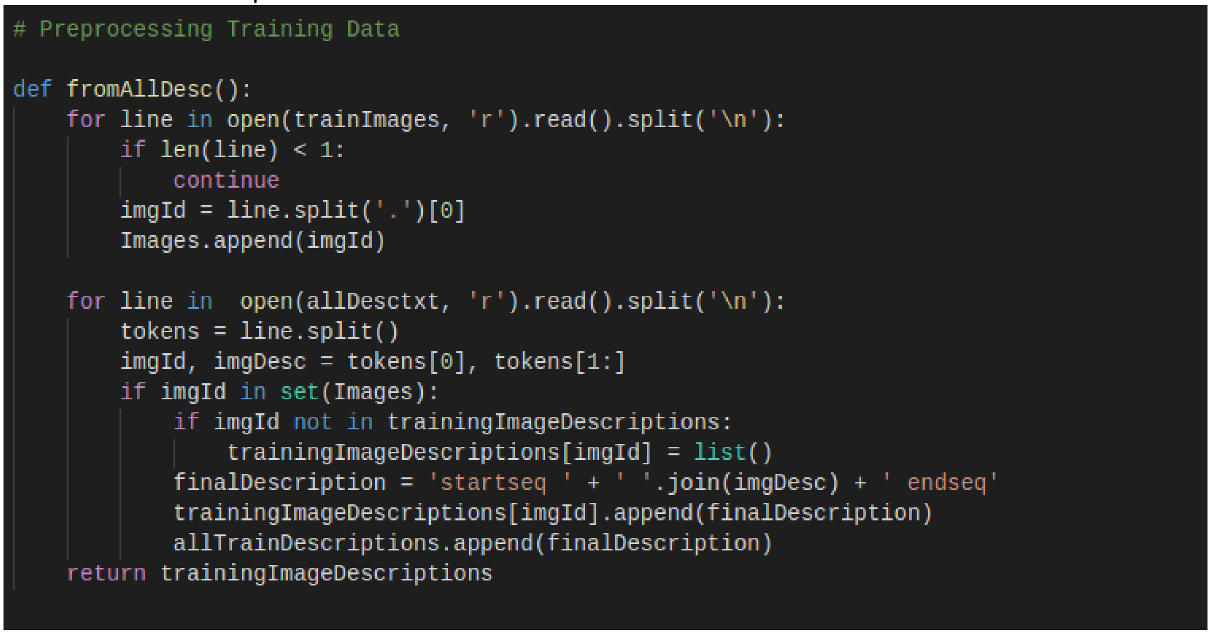
```
# Pre trained CNN

def CNN():

# CNN for feature extraction

    for name in listdir(imagesSet):
        filename = imagesSet + '/' + name
        image = load_img(filename, target_size=(299, 299))
        image = img_to_array(image)
        print(name)
        image = image.reshape(
            (1, image.shape[0], image.shape[1], image.shape[2]))
        image = preprocess_input(image)
        feature = PreTrainedCNNModel.predict(image, verbose=0)
        imgId = name.split('.')[0]
        allFeat[imgId] = feature
    saveDesc(allFeat)

def saveDesc(allFeatures):
    dump(allFeatures, open(featurespkl , 'wb'))
```

3) Now, we take all the descriptions for images from descriptions.txt file and preprocess them before we append all the descriptions for an image. The appended will look as below:



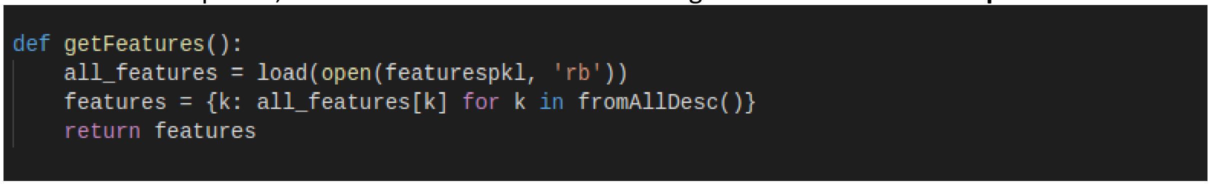
```python
#Get Descriptions From Token Dataset

def loadallDesc():
    for line in open(allDescriptions, 'r').read().split('\n'):
        tokens = line.split("#")
        if len(line) < 2:
            continue
        imgId, imgDesc = tokens[0], tokens[1:]
        imgId = imgId.split('.')[0]
        imgDesc = ' '.join(imgDesc)
        if imgId not in imageDescriptions:
            imageDescriptions[imgId] = list()
        imageDescriptions[imgId].append(imgDesc)

    for key, value in imageDescriptions.items():
        for i in range(len(value)):
            eachDescription = value[i]
            eachDescription = eachDescription.split()
            eachDescription = [word.lower() for word in eachDescription]
            eachDescription = [w.translate(str.maketrans('', '', string.punctuation)) for w in eachDescription]
            eachDescription = [word for word in eachDescription if len(word)>1]
            eachDescription = [word for word in eachDescription if word.isalpha()]
            value[i] =  ' '.join(eachDescription)


    for key, value in imageDescriptions.items():
        for desc in value:
            imgDescPair.append(key + ' ' + desc)

    allRel = '\n'.join(imgDescPair)
    file = open(allDesctxt , 'w')
    file.write(allRel)
    file.close()
```

4) Now that we have all the descriptions, we can begin training the data. For that, we use trainImages.txt file which contains the details of images that can be used for training.
5) Now that we know what images can be used for training, we obtain the descriptions of the training images. For all the descriptions we append a start and end sequence which will be useful later in our implementation.



```python
# Preprocessing Training Data

def fromAllDesc():
    for line in open(trainImages, 'r').read().split('\n'):
        if len(line) < 1:
            continue
        imgId = line.split('.')[0]
        Images.append(imgId)

    for line in  open(allDesctxt, 'r').read().split('\n'):
        tokens = line.split()
        imgId, imgDesc = tokens[0], tokens[1:]
        if imgId in set(Images):
            if imgId not in trainingImageDescriptions:
                trainingImageDescriptions[imgId] = list()
            finalDescription = 'startseq ' + ' '.join(imgDesc) + ' endseq'
            trainingImageDescriptions[imgId].append(finalDescription)
            allTrainDescriptions.append(finalDescription)
    return trainingImageDescriptions
```

6) Similar to descriptions, we obtain the features of the image as well from **features.pkl** file.



```python
def getFeatures():
    all_features = load(open(featurespkl, 'rb'))
    features = {k: all_features[k] for k in fromAllDesc()}
    return features
```

7) Now, we define a model which has two inputs, one for the input image and the other for descriptions. We combine both and obtain the output which is our prediction. The input

layer is of shape = (47, 1000) which is the shape of our input image. For the input 2 we give the length of our longest description.

```
# Input for Image
inputs1 = layers.Input(shape=(47,1000))
fe1 = layers.Dropout(0.5)(inputs1)
fe2 = layers.Dense(256, activation='relu')(fe1)

# Input for text
inputs2 = layers.Input(shape=(getLength(),))
se1 = layers.Embedding(vocabSize, 256, mask_zero=True)(inputs2)
se2 = layers.Dropout(0.5)(se1)
se3 = layers.LSTM(256)(se2)

# Combine here
decoder1 = layers.add([fe2, se3])
decoder2 = layers.Dense(256, activation='relu')(decoder1)
outputs = layers.Dense(vocabSize, activation='softmax')(decoder2)
model = models.Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

8) Now that we have our model defined and training data ready, we can proceed with training.
9) For training the model, we give the training descriptions, features and all the vocabulary available in the training descriptions and the length of longest description.
10) First, we convert all the descriptions that we are having to sequence using **text_to_sequences** provided by tokenizer which is also used to obtain vocabulary of our descriptions. We train the model by giving the features and input sequence as an input and the output sequence. Both will be given as a tuple for the model.

```
def txtToseq(tokenizer, maxLen, value, feat):
    X1, X2, y = list(), list(), list()
    for desc in value:
        seq = tokenizer.texts_to_sequences([desc])[0]
        for i in range(1, len(seq)):
            inSeq, outSeq = seq[:i], seq[i]
            inSeq = pad_sequences([inSeq], maxlen=maxLen)[0]
            outSeq = to_categorical([outSeq], num_classes=vocabSize)[0]

            X1.append(feat)
            X2.append(inSeq)
            y.append(outSeq)
    return array(X1), array(X2), array(y)


def trainingData(descriptions, trainFeat, tokenizer, maxLen):
    while 1:
        for key, value in descriptions.items():
            feat = trainFeat[key][0]
            recvFeatVect, inSeqVect, outSeqVect = txtToseq(tokenizer, maxLen, value, feat)
            yield [recvFeatVect, inSeqVect], outSeqVect
```

Training:

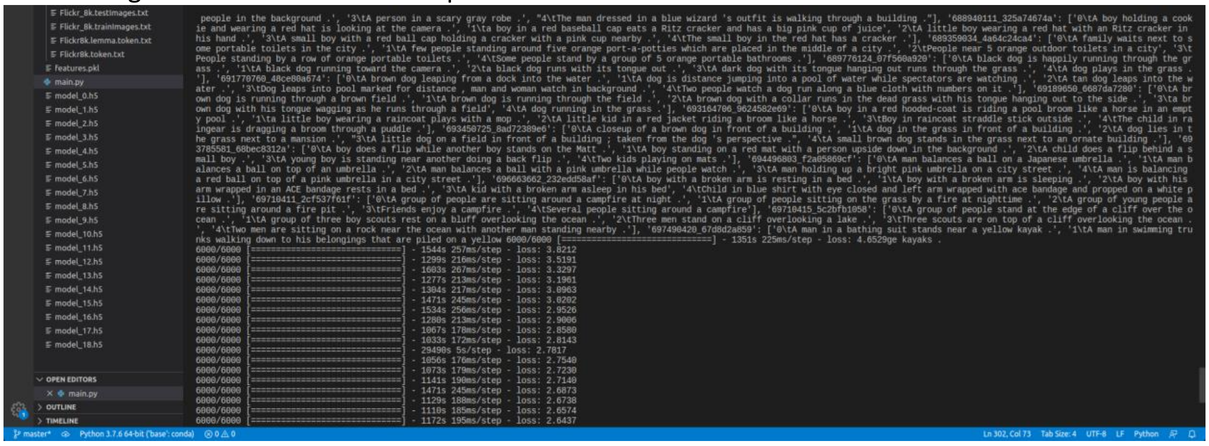
1) First, we extract the features using the **CNN** function which has our pre trained model.

```
# Step 1: Uncomment below to initiate pre training and extract features from all images

"""

CNN()

"""
```

2) Now we load all the training descriptions and their features, available vocabulary.

```
# Step 2: load all training descriptions, training features and available vocabulary

loadallDesc()
print('Loaded training descriptions from all desc ', len(fromAllDesc()))
tokenizer = getVocab()
vocabSize = len(tokenizer.word_index) + 1
print('Total Vocabulary from training descriptions ', vocabSize)
print('Longest sentence for RNN input layer ',  getLength())
trainFeat = getFeatures()
```

3) We train the model using the above information for 18 epochs each for 6000 steps considering that we have 6000 descriptions.



```
 people in the background .', '3\tA person in a scary gray robe .', "4\tThe man dressed in a blue wizard 's outfit is walking through a building ."], '688940111_325a74674a': ['0\tA boy holding a cook
ie and wearing a red hat is looking at the camera .', '1\ta boy in a red baseball cap eats a Ritz cracker and has a big pink cup of juice', '2\tA little boy wearing a red hat with an Ritz cracker in
his hand .', '3\tA small boy with a red ball cap holding a cracker with a pink cup nearby .', '4\tThe small boy in the red hat has a cracker .'], '689359034_4a64c24ca4': ['0\tA family waits next to s
ome portable toilets in the city .', '1\tA few people standing around five orange port-a-potties which are placed in the middle of a city .', '2\tPeople near 5 orange outdoor toilets in a city', '3\t
People standing by a row of orange portable toilets .', '4\tSome people stand by a group of 5 orange portable bathrooms .'], '689776124_07f560a920': ['0\tA black dog is happily running through the gr
ass .', '1\tA black dog running toward the camera .', '2\ta black dog runs with its tongue out .', '3\tA dark dog with its tongue hanging out runs through the grass .', '4\tA dog plays in the grass .
'], '691770760_48ce80a674': ['0\tA brown dog leaping from a dock into the water .', '1\tA dog is distance jumping into a pool of water while spectators are watching .', '2\tA tan dog leaps into the w
ater .', '3\tDog leaps into pool marked for distance , man and woman watch in background .', '4\tTwo people watch a dog run along a blue cloth with numbers on it .'], '69189650_6687da7280': ['0\tA br
own dog is running through a brown field .', '1\tA brown dog is running through the field .', '2\tA brown dog with a collar runs in the dead grass with his tongue hanging out to the side .', '3\ta br
own dog with his tongue wagging as he runs through a field', '4\tA dog running in the grass .'], '693164706_9624582e69': ['0\tA boy in a red hooded-coat is riding a pool broom like a horse in an empt
y pool .', '1\ta little boy wearing a raincoat plays with a mop .', '2\tA little kid in a red jacket riding a broom like a horse .', '3\tBoy in raincoat straddle stick outside .', '4\tThe child in ra
ingear is dragging a broom through a puddle .'], '693450725_8ad72389e6': ['0\tA closeup of a brown dog in front of a building .', '1\tA dog in the grass in front of a building .', '2\tA dog lies in t
he grass next to a mansion .', "3\tA little dog on a field in front of a building ; taken from the dog 's perspective .", '4\tA small brown dog stands in the grass next to an ornate building .'], '69
3785581_68bec8312a': ['0\tA boy does a flip while another boy stands on the Matt .', '1\tA boy standing on a red mat with a person upside down in the background .', '2\tA child does a flip behind a s
mall boy .', '3\tA young boy is standing near another doing a back flip .', '4\tTwo kids playing on mats .'], '694496803_f2a05869cf': ['0\tA man balances a ball on a Japanese umbrella .', '1\tA man b
alances a ball on top of an umbrella .', '2\tA man balances a ball with a pink umbrella while people watch .', '3\tA man holding up a bright pink umbrella on a city street .', '4\tA man is balancing
a red ball on top of a pink umbrella in a city street .'], '696663662_232edd58af': ['0\tA boy with a broken arm is resting in a bed .', '1\tA boy with a broken arm is sleeping .', '2\tA boy with his
arm wrapped in an ACE bandage rests in a bed .', '3\tA kid with a broken arm asleep in his bed', '4\tChild in blue shirt with eye closed and left arm wrapped with ace bandage and propped on a white p
illow .'], '69710411_2cf537f61f': ['0\tA group of people are sitting around a campfire at night .', '1\tA group of people sitting on the grass by a fire at nighttime .', '2\tA group of young people a
re sitting around a fire pit .', '3\tFriends enjoy a campfire .', '4\tSeveral people sitting around a campfire'], '69710415_5c2bfb1058': ['0\tA group of people stand at the edge of a cliff over the o
cean .', '1\tA group of three boy scouts rest on a bluff overlooking the ocean .', '2\tThree men stand on a cliff overlooking a lake .', '3\tThree scouts are on top of a cliff overlooking the ocean .
', '4\tTwo men are sitting on a rock near the ocean with another man standing nearby .'], '697490420_67d8d2a859': ['0\tA man in a bathing suit stands near a yellow kayak .', '1\tA man in swimming tru
nks walking down to his belongings that are piled on a yellow 6000/6000 [==============================] - 1351s 225ms/step - loss: 4.6529ge kayaks .
6000/6000 [==============================] - 1544s 257ms/step - loss: 3.8212
6000/6000 [==============================] - 1299s 216ms/step - loss: 3.5191
6000/6000 [==============================] - 1603s 267ms/step - loss: 3.3297
6000/6000 [==============================] - 1277s 213ms/step - loss: 3.1961
6000/6000 [==============================] - 1304s 217ms/step - loss: 3.0963
6000/6000 [==============================] - 1471s 245ms/step - loss: 3.0202
6000/6000 [==============================] - 1534s 256ms/step - loss: 2.9526
6000/6000 [==============================] - 1280s 213ms/step - loss: 2.9006
6000/6000 [==============================] - 1067s 178ms/step - loss: 2.8580
6000/6000 [==============================] - 1033s 172ms/step - loss: 2.8143
6000/6000 [==============================] - 29490s 5s/step - loss: 2.7817
6000/6000 [==============================] - 1056s 176ms/step - loss: 2.7540
6000/6000 [==============================] - 1073s 179ms/step - loss: 2.7230
6000/6000 [==============================] - 1141s 190ms/step - loss: 2.7140
6000/6000 [==============================] - 1471s 245ms/step - loss: 2.6873
6000/6000 [==============================] - 1129s 188ms/step - loss: 2.6738
6000/6000 [==============================] - 1110s 185ms/step - loss: 2.6574
6000/6000 [==============================] - 1172s 195ms/step - loss: 2.6437
```

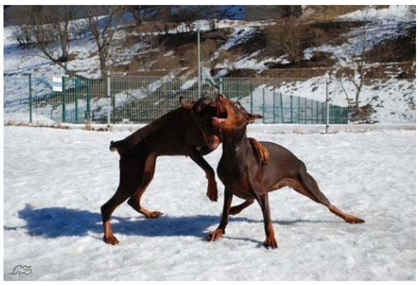
Test:

1) We load one of our trained models using **load_model.**
2) We also load our pre trained CNN for feature extraction, i.e. InceptionV3.
3) We load one of the images available in our test set and give it as an input for our pre trained CNN.
4) The pre trained CNN will get the features of the image which will act as an input to or model.
5) We also give our sequence initiator along with the features and the sequence ender will act as a terminator of the sentence.

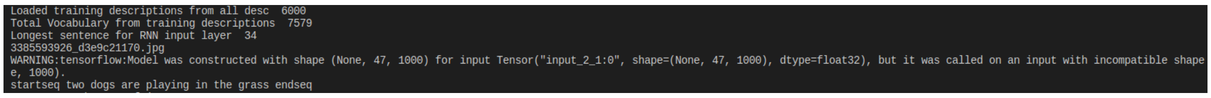
```
# Step 5: Test the trained model

trainedModel = models.load_model(savedModels)

def getTestDesc(model, tokenizer, feat, maxLen):
    in_text = 'startseq'
    for i in range(maxLen):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=maxLen)
        outWord = model.predict([feat,sequence], verbose=0)
        outWord = argmax(outWord)
        for each, index in tokenizer.word_index.items():
            if index == outWord:
                word = each
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text

for line in open(testImages, 'r').read().split('\n'):
    print(line)
    for name in listdir(imagesSet):
        if re.match(name, line):

            PreTrainedCNNModel.layers.pop()
            PreTrainedCNNModel = models.Model(inputs=PreTrainedCNNModel.inputs, outputs=PreTrainedCNNModel.layers[-1].output)
            image = load_img(imagesSet +'/' + line, PreTrainedCNNModel, target_size=(299, 299))
            image = img_to_array(image)
            image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
            image = preprocess_input(image)
            feature = PreTrainedCNNModel.predict(image, verbose=0)

            description = getTestDesc(trainedModel, tokenizer, feature, getLength())
            print(description)
```

6) We iterate until longest possible description available from our training and when we receive sequence terminator or when we complete iterations, we print the predicted output word so far.

Input Image:



Output Description:



```
Loaded training descriptions from all desc  6000
Total Vocabulary from training descriptions  7579
Longest sentence for RNN input layer  34
3385593926_d3e9c21170.jpg
WARNING:tensorflow:Model was constructed with shape (None, 47, 1000) for input Tensor("input_2_1:0", shape=(None, 47, 1000), dtype=float32), but it was called on an input with incompatible shape
e, 1000).
startseq two dogs are playing in the grass endseq
```

References:

Show and Tell: A Neural Image Caption Generator
https://arxiv.org/abs/1411.4555

Geeks for geeks – Image Caption Generator
https://www.geeksforgeeks.org/image-caption-generator-using-deep-learning-on-flickr8k-dataset/