Image Captioning using Attention Model

Installing "ipython-autotime" to keep track of time for each cell

In [1]:

```
pip install ipython-autotime
```

```
Collecting ipython-autotime
  Downloading https://files.pythonhosted.org/packages/e6/f9/0626bbdb322e3a
078d968e87e3b01341e7890544de891d0cb613641220e6/ipython-autotime-0.1.tar.bz
2
Building wheels for collected packages: ipython-autotime
  Building wheel for ipython-autotime (setup.py) ... done
  Created wheel for ipython-autotime: filename=ipython_autotime-0.1-cp36-n
one-any.whl size=1832 sha256=48ca11749c446d8c2b7e5ffe718abee7549ec93c5c2b5
07cd92c2435bcc24cd5
  Stored in directory: /root/.cache/pip/wheels/d2/df/81/2db1e54bc91002cec4
0334629bc39cfa86dff540b304ebcd6e
Successfully built ipython-autotime
Installing collected packages: ipython-autotime
Successfully installed ipython-autotime-0.1
```

Importing Libraries

In [2]:

```
%%time
import pandas as pd
import numpy as np
import os
from os import path
import re
from tqdm import tqdm
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
import cv2
from google.colab.patches import cv2_imshow
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.applications.inception_v3 import InceptionV3,preprocess_input
from keras.models import *
from keras.layers import *
from keras.layers.merge import add
from keras.utils import to_categorical
import pickle
from pickle import dump, load
import tensorflow as tf
from keras import optimizers
import random
import keras.backend as K
import matplotlib.pyplot as plt
from keras import regularizers, initializers, constraints
from keras.layers import Layer
from keras.utils import plot_model
```

```
Using TensorFlow backend.

CPU times: user 1.62 s, sys: 221 ms, total: 1.84 s
Wall time: 2.52 s
```

Getting the file Path of Annotation file and Image Directory

In [3]:

```
%%time
annot_file_path = '/content/drive/My Drive/Colab Notebooks/flickr8k/Flickr8k.token.txt'
image_dir_path = '/content/drive/My Drive/Colab Notebooks/flickr8k/Images/'
```

```
CPU times: user 4 µs, sys: 1 µs, total: 5 µs
Wall time: 7.87 µs
```

Loading the annot file as csv with '\t' (tab) as seperator, and manually assigning Column names as 'filename' and 'caption'

In [4]:

```
%%time
data = pd.read_csv(annot_file_path,sep='\t',names=['filename','caption'])
```

```
CPU times: user 58.6 ms, sys: 29.2 ms, total: 87.7 ms
Wall time: 1.51 s
```

Removing ' jpg#number' from the values so we get same values for images

In [5]:

```
%%time
data['filename'] = data['filename'].str.replace('#\d','')
```

```
CPU times: user 38 ms, sys: 3.87 ms, total: 41.9 ms
Wall time: 43.3 ms
```

In [6]:

```
%%time
data.head()
```

```
CPU times: user 335 µs, sys: 65 µs, total: 400 µs
Wall time: 430 µs
```

Out[6]:

	filename	caption
0	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	A girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a woo...

In [7]:

```
%%time
print('Total Number of Unique Images in the Annotation Files : ',data.filename.nunique
())
```

```
Total Number of Unique Images in the Annotation Files :  8092
CPU times: user 7.41 ms, sys: 15 µs, total: 7.43 ms
Wall time: 8.36 ms
```

Checking the number of files in the Image dir, to make sure that we are not having mismatch of data.

In [8]:

```
%%time
annotation_images_name = data.filename.unique()
image_dir_images_name = os.listdir(image_dir_path)
print(set(annotation_images_name) ^ set(image_dir_images_name) )
```

```
{'2258277193_586949ec62.jpg.1'}
CPU times: user 14.8 ms, sys: 3.17 ms, total: 18 ms
Wall time: 59.2 s
```

In [9]:

```
%%time
print('Total Number of Images in the Image Directory : ',len(image_dir_images_name))
```

```
Total Number of Images in the Image Directory :  8091
CPU times: user 146 µs, sys: 0 ns, total: 146 µs
Wall time: 101 µs
```

We found a unique file name which has '.' before the number, this could have created issue in training. When i checked manually such filename doest exist in the image directory. We can remove it from our data.

In [10]:

```
%%time
data.drop(data[data['filename'] == '2258277193_586949ec62.jpg.1'].index, inplace = True
)
```

```
CPU times: user 8.57 ms, sys: 0 ns, total: 8.57 ms
Wall time: 10.9 ms
```

In [11]:

```
%%time
print('Now let\'s check The unique number of images : ',len(data.filename.unique()))
```

```
Now let's check The unique number of images :  8091
CPU times: user 4.6 ms, sys: 526 µs, total: 5.12 ms
Wall time: 5.03 ms
```

Working on Text Data

Preparation for caption text cleaning

In [12]:

```
%%time
# https://stackoverflow.com/a/47091490/4084039

def decontracted(phrase):
  # specific
  phrase = re.sub(r"won't", "will not", phrase)
  phrase = re.sub(r"can\'t", "can not", phrase)
  # general
  phrase = re.sub(r"n\'t", " not", phrase)
  phrase = re.sub(r"\'re", " are", phrase)
  phrase = re.sub(r"\'s", " is", phrase)
  phrase = re.sub(r"\'d", " would", phrase)
  phrase = re.sub(r"\'ll", " will", phrase)
  phrase = re.sub(r"\'t", " not", phrase)
  phrase = re.sub(r"\'ve", " have", phrase)
  phrase = re.sub(r"\'m", " am", phrase)
  phrase = re.sub(r"nan" , '' , phrase)
  return phrase
# remove all the unnecessay characters from the text
def removeSpecialChars(phrase):
  phrase = phrase.replace('\\r', ' ')
  phrase = phrase.replace('\\"', ' ')
  phrase = phrase.replace('\\n', ' ')
  phrase = re.sub('[^A-Za-z0-9]+', ' ', phrase)
  return phrase
```

```
CPU times: user 3 µs, sys: 0 ns, total: 3 µs
Wall time: 5.96 µs
```

Cleaning the caption and storing them in a list

In [13]:

```
%%time
# Combining all the above stundents
caption_list = []
# tqdm is for printing the status bar
for sentance in tqdm(data['caption'].values):
  sent = decontracted(sentance)
  sent = sent.replace('\\r', ' ')
  sent = sent.replace('\\"', ' ')
  sent = sent.replace('\\n', ' ')
  sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
  caption_list.append(sent.lower().strip())
```

```
100%|██████████| 40455/40455 [00:00<00:00, 69900.26it/s]

CPU times: user 580 ms, sys: 2.71 ms, total: 583 ms
Wall time: 582 ms
```

In [14]:

```
%%time
caption_list[:5]
```

```
CPU times: user 3 µs, sys: 1 µs, total: 4 µs
Wall time: 6.68 µs
```

Out[14]:

```
['a child in a pink dress is climbing up a set of stairs in an entry way',
 'a girl going into a wooden building',
 'a little girl climbing into a wooden playhouse',
 'a little girl climbing the stairs to her playhouse',
 'a little girl in a pink dress going into a wooden cabin']
```

Re-Assigning the list back to the Data Frame

In [15]:

```
%%time
data['caption'] = caption_list
```

```
CPU times: user 7.63 ms, sys: 927 µs, total: 8.56 ms
Wall time: 9.8 ms
```

In [16]:

```
%%time
data.head()
```

```
CPU times: user 562 µs, sys: 92 µs, total: 654 µs
Wall time: 755 µs
```

Out[16]:

	filename	caption
0	1000268201_693b08cb0e.jpg	a child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	a girl going into a wooden building
2	1000268201_693b08cb0e.jpg	a little girl climbing into a wooden playhouse
3	1000268201_693b08cb0e.jpg	a little girl climbing the stairs to her playh...
4	1000268201_693b08cb0e.jpg	a little girl in a pink dress going into a woo...

Now to convert words to Tokens we can use a powerful function in Python which can take care of multiple things ar once, like CountVectorizer,

- analyzer='word' , makes sure that single word is taken into consideration for tokenization
- min_df=10, makes sure that only that word is taken into consideration whose occurance is more than 10 times

In [17]:

```
%%time
vectorizer = CountVectorizer(analyzer='word',min_df=10)
```

```
CPU times: user 28 µs, sys: 0 ns, total: 28 µs
Wall time: 32.7 µs
```

In [18]:

```
%%time
vectorizer.fit(caption_list)
```

```
CPU times: user 384 ms, sys: 7.93 ms, total: 392 ms
Wall time: 403 ms
```

Out[18]:

```
CountVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.int64'>, encoding='utf-8', input='cont
ent',
                lowercase=True, max_df=1.0, max_features=None, min_df=10,
                ngram_range=(1, 1), preprocessor=None, stop_words=None,
                strip_accents=None, token_pattern='(?u)\\b\\w\\w+\\b',
                tokenizer=None, vocabulary=None)
```

In [19]:

```
%%time
print('Total Number of unique Words : ',len(vectorizer.get_feature_names()))
```

```
Total Number of unique Words :  1955
CPU times: user 2.01 ms, sys: 0 ns, total: 2.01 ms
Wall time: 2.18 ms
```

Get the feature names from the vectorizer as a word and store it in a index mapping dict, will be used later in this module.

In [20]:

```
%%time
word_index_Mapping = dict()
index = 0
for word in vectorizer.get_feature_names():
  index += 1
  word_index_Mapping[word] = index

# the word 'a' is missed out as a stop word, hence including manually
word_index_Mapping['a'] = index+1
index += 1
word_index_Mapping['startSeq'] = index+1
index += 1
word_index_Mapping['endSeq'] = index+1
index += 1
word_index_Mapping['<pad>'] = 0
print('Last Index : ',index)
print('Length of dictionary', len(word_index_Mapping))
```

```
Last Index :  1958
Length of dictionary 1959
CPU times: user 3.15 ms, sys: 62 µs, total: 3.21 ms
Wall time: 3.64 ms
```

Reverse the word index into index word map, we would need this too, thanks to powerful tools in python this is just 1 line code. We need this for future references.

In [0]:

```
index_word_Mapping = dict(map(reversed, word_index_Mapping.items()))

vocab_size = len(word_index_Mapping) + 1
```

Now, lets check if the words are mapped properly or not, first take any random word pass it as key into word_index_mappin and get its corresponding index value, then pass it as key to index_word_mapping, if both the word, you passed as key and printed are same then its perfect.

In [22]:

```
%%time
print(index_word_Mapping[word_index_Mapping['girl']])
print(index_word_Mapping[word_index_Mapping['the']])
print(index_word_Mapping[word_index_Mapping['ball']])
print(index_word_Mapping[word_index_Mapping['cat']])
print(index_word_Mapping[word_index_Mapping['play']])
```

```
girl
the
ball
cat
play
CPU times: user 0 ns, sys: 620 µs, total: 620 µs
Wall time: 478 µs
```

We have around 8091 images in all we will take 7091 images as training data and remianing will be used as test.

We achieve this by taking first 7091 as training and then take a set difference to find out the rest

In [23]:

```
%%time
train_list = data.filename.unique()[1:7092]
test_list = set(data.filename.unique()) ^ set(train_list)
print('Total Number of images in Train : ',len(train_list))
print('Total Number of images in Test : ',len(test_list))
```

```
Total Number of images in Train :  7091
Total Number of images in Test :  1000
CPU times: user 13.5 ms, sys: 0 ns, total: 13.5 ms
Wall time: 16.1 ms
```

Now filtering out the data from the dataframe using the list we have with us

In [24]:

```
%%time
train_df = data[data['filename'].isin(train_list)]
test_df = data[data['filename'].isin(test_list)]

print(train_df.shape)
print(test_df.shape)
```

```
(35455, 2)
(5000, 2)
CPU times: user 12.6 ms, sys: 21 µs, total: 12.6 ms
Wall time: 25.4 ms
```

Now we have to append a \ and \ among each captions start and end respectively. So to make our model understand where the caption starts and where it ends.

In [25]:

```
%%time
train_df['caption'] = 'startSeq ' + train_df['caption'].astype(str) + ' endSeq'
```

```
CPU times: user 19.8 ms, sys: 4.89 ms, total: 24.7 ms
Wall time: 39 ms

/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: SettingWit
hCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-doc
s/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  """Entry point for launching an IPython kernel.
```

In [26]:

```
%%time
train_df
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.63 µs
```

Out[26]:

	filename	caption
5	1001773457_577c3a7d70.jpg	startSeq a black dog and a spotted dog are fig...
6	1001773457_577c3a7d70.jpg	startSeq a black dog and a tri colored dog pla...
7	1001773457_577c3a7d70.jpg	startSeq a black dog and a white dog with brow...
8	1001773457_577c3a7d70.jpg	startSeq two dogs of different breeds looking ...
9	1001773457_577c3a7d70.jpg	startSeq two dogs on pavement moving toward ea...
...	...	...
35460	378453580_21d688748e.jpg	startSeq a dog is jumping over a log in a wood...
35461	378453580_21d688748e.jpg	startSeq a dog with a stick in his mouth jumps...
35462	378453580_21d688748e.jpg	startSeq dog carries stick and jumps over a lo...
35463	378453580_21d688748e.jpg	startSeq the dog carries a stick and jumps ove...
35464	378453580_21d688748e.jpg	startSeq the dog jumps over the log with a sti...

35455 rows × 2 columns

We will get the length of the max length caption in our data of captions so that we can padd the other ones with the max length

In [27]:

```
%%time
max_caption_length = max(len(caption.split()) for caption in train_df.caption.to_list
())
print('Longets Captiopn Length : ',max_caption_length)
```

```
Longets Captiopn Length :  39
CPU times: user 32.6 ms, sys: 0 ns, total: 32.6 ms
Wall time: 36 ms
```

Let's vectorize our captions first. We will only embedd the words present in our word_index_Mapping as there are 400,000 words in glove, and we only have 1955 words.

Storing the embeddings of the unique words into a dict for reference.

In [28]:

```
%%time
# Load Glove vectors
glove_dir = '/content/drive/My Drive/Colab Notebooks/flickr8k/'
embeddings_index = {} # empty dictionary
f = open(os.path.join(glove_dir, 'glove.6B.300d.txt'), encoding="utf-8")

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    if word in word_index_Mapping.keys():
      embeddings_index[word] = coefs
f.close()
```

```
CPU times: user 32 s, sys: 715 ms, total: 32.7 s
Wall time: 35.6 s
```

In [29]:

```
%%time
len(embeddings_index)
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 5.72 µs
```

Out[29]:

```
1952
```

Creating a matrix so that it can be used for training our model as weights.

In [0]:

```
embedding_dim = 300

# Get 300-dim dense vector for each of the 10000 words in out vocabulary
embedding_matrix = np.zeros((vocab_size, embedding_dim))

for word, i in word_index_Mapping.items():
    #if i < max_words:
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        # Words not found in the embedding index will be all zeros
        embedding_matrix[i] = embedding_vector
```

In [31]:

```
len(embedding_matrix)
```

Out[31]:

```
1960
```

Creating caption Sequence :

In [0]:

```
# w2v encode sequence
def caption_encode(caption, max_caption_length):
    encoding = list()
    for word in caption.split():
        if word in word_index_Mapping.keys():
          index = word_index_Mapping[word]
        else:
          index=0
        encoding.append(index)

    for i in range(len(encoding),max_caption_length):
      encoding.append(0)

    return np.array(encoding)

# decode a one hot encoded string
def caption_decode(encoded_caption):
  caption = ""
  for index in encoded_caption:
    if index != 0:
      caption += index_word_Mapping[index] + " "
  return caption.strip()
```

In [33]:

```
print(train_df.caption.to_list()[0])
encoded = caption_encode(train_df.caption.to_list()[0],max_caption_length)
print(caption_decode(encoded))
```

```
startSeq a black dog and a spotted dog are fighting endSeq
startSeq a black dog and a spotted dog are fighting endSeq
```

Working on Image Data

Checking the load function of an image using cv2 lib

Also as cv2.imshow() method is not allowed in google colab, as it crashes the jupyter session, hence we use google's own cv2_imshow()

In [34]:

```
%%time
imageFile = os.path.join(image_dir_path,data.filename.unique()[random.randint(1,1000)])
img = cv2.imread(imageFile)
cv2_imshow(img)
```

```
CPU times: user 73.8 ms, sys: 7 ms, total: 80.8 ms
Wall time: 495 ms
```

Now we have to extract the featuer vector for every image, and we are going to do it with InceptionV3 model.

Also by removing the last layer form the model we will the feature vector from the images.

In [35]:

```
%%time
model = InceptionV3(weights='imagenet')
incp_model_feature = Model(model.input, model.layers[-2].output)
```

```
Downloading data from https://github.com/fchollet/deep-learning-models/rel
eases/download/v0.5/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96116736/96112376 [==============================] - 1s 0us/step
CPU times: user 5.05 s, sys: 595 ms, total: 5.64 s
Wall time: 6.13 s
```

Storing the feature vector to a file so that we need not to calculate it each time.

In [36]:

```
%%time
imageFeatureFileLocation='/content/drive/My Drive/Colab Notebooks/flickr8k/'
fileName = 'image_feature_train.pkl'

if os.path.isfile(os.path.join(imageFeatureFileLocation,fileName)):
  photo_embeddings_train = load(open(os.path.join(imageFeatureFileLocation,fileName),
"rb"))
  print('Train File Loaded : ')
else:
  print('File is not Created.. Please wait.')
  photo_embeddings_train = {};
  for imageName in tqdm(train_df.filename.unique()):
    imageFile = os.path.join(image_dir_path,imageName)
    img = cv2.imread(imageFile)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(299,299))
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    vector = incp_model_feature.predict(img)
    vector = np.reshape(vector, vector.shape[1])
    photo_embeddings_train[imageName] = vector

    #save the file for later use
    with open(os.path.join(imageFeatureFileLocation,fileName), "wb") as pickel_file:
      pickle.dump(encoding_train, pickel_file)
```

```
Train File Loaded :
CPU times: user 64.3 ms, sys: 41.2 ms, total: 105 ms
Wall time: 6.58 s
```

Same thing is done for test as well.

In [37]:

```
%%time
imageFeatureFileLocation='/content/drive/My Drive/Colab Notebooks/flickr8k/'
fileName = 'image_feature_test.pkl'

if os.path.isfile(os.path.join(imageFeatureFileLocation,fileName)):
  photo_embeddings_test = load(open(os.path.join(imageFeatureFileLocation,fileName), "r
b"))
  print('Test File Loaded : ')
else:
  print('File is not Created.. Please wait.')
  photo_embeddings_test = {};
  for imageName in tqdm(test_df.filename.unique()):
    imageFile = os.path.join(image_dir_path,imageName)
    img = cv2.imread(imageFile)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(299,299))
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)
    vector = incp_model_feature.predict(img)
    vector = np.reshape(vector, vector.shape[1])
    photo_embeddings_test[imageName] = vector

    #save the file for later use
    with open(os.path.join(imageFeatureFileLocation,fileName), "wb") as pickel_file:
      pickle.dump(photo_embeddings_test, pickel_file)
```

```
Test File Loaded :
CPU times: user 11.1 ms, sys: 1.83 ms, total: 12.9 ms
Wall time: 900 ms
```

In [38]:

```
%%time
# prepare data for the LSTM
def get_photo_caption_pair(filename):
  input_1 = list()
  input_2 = list()
  output_2 = list()

  photo_feature = photo_embeddings_train[filename]

  for index, rows in train_df[train_df.filename==filename].iterrows():
    seq = [word_index_Mapping[word] for word in rows['caption'].split(' ') if word in w
ord_index_Mapping]

    for i in range(1, len(seq)):
      in_sequence = seq[:i]
      out_sequence = seq[i]

      in_seq = pad_sequences([in_sequence], maxlen=max_caption_length)[0]
      out_seq = to_categorical([out_sequence], num_classes=vocab_size)[0]

      input_1.append(photo_feature)
      input_2.append(in_seq)
      output_2.append(out_seq)
  return np.array(input_1), np.array(input_2), np.array(output_2)
```

```
CPU times: user 4 µs, sys: 0 ns, total: 4 µs
Wall time: 7.15 µs
```

In [0]:

```
def train_generator(number_of_images_per_batch):
  n=0
  while 1:
    for filename in train_df.filename.unique():
      n+=1
      X1, X2, y = get_photo_caption_pair(filename)
      # yield the batch data
      if n==number_of_images_per_batch:
        yield [[X1, X2], y]
        X1, X2, y = list(), list(), list()
        n=0
```

Implementing Attention Layer with Encoder Decoder Model :

In [0]:

```
class Custom_Attention(Layer):

    def __init__(self, step_dim,
                 W_regularizer=None, b_regularizer=None,
                 W_constraint=None, b_constraint=None,
                 bias=True, **kwargs):
        """
        Keras Layer that implements an Attention mechanism for temporal data.
        Supports Masking.
        Follows the work of Raffel et al. [https://arxiv.org/abs/1512.08756]
        # Input shape
            3D tensor with shape: `(samples, steps, features)`.
        # Output shape
            2D tensor with shape: `(samples, features)`.
        :param kwargs:
        Just put it on top of an RNN Layer (GRU/LSTM/SimpleRNN) with return_sequences=T
rue.
        The dimensions are inferred based on the output shape of the RNN.
        Example:
            model.add(LSTM(64, return_sequences=True))
            model.add(Attention())
        """
        self.supports_masking = True
        #self.init = initializations.get('glorot_uniform')
        self.init = initializers.get('glorot_uniform')

        self.W_regularizer = regularizers.get(W_regularizer)
        self.b_regularizer = regularizers.get(b_regularizer)

        self.W_constraint = constraints.get(W_constraint)
        self.b_constraint = constraints.get(b_constraint)

        self.bias = bias
        self.step_dim = step_dim
        self.features_dim = 0
        super(Custom_Attention, self).__init__(**kwargs)

    def build(self, input_shape):
        assert len(input_shape) == 3

        self.W = self.add_weight(shape=(input_shape[-1],),
                                 initializer=self.init,
                                 name='Attention_W',
                                 regularizer=self.W_regularizer,
                                 constraint=self.W_constraint)
        self.features_dim = input_shape[-1]

        if self.bias:
            self.b = self.add_weight(shape=(input_shape[1],),
                                     initializer='zero',
                                     name='Attention_b',
                                     regularizer=self.b_regularizer,
                                     constraint=self.b_constraint)
        else:
            self.b = None

        self.built = True

    def compute_mask(self, input, input_mask=None):
```

```
        # do not pass the mask to the next layers
        return None

    def call(self, x, mask=None):
        # eij = K.dot(x, self.W) TF backend doesn't support it

        # features_dim = self.W.shape[0]
        # step_dim = x._keras_shape[1]

        features_dim = self.features_dim
        step_dim = self.step_dim

        eij = K.reshape(K.dot(K.reshape(x, (-1, features_dim)), K.reshape(self.W, (feat
ures_dim, 1))), (-1, step_dim))

        if self.bias:
            eij += self.b

        eij = K.tanh(eij)

        a = K.exp(eij)

        # apply mask after the exp. will be re-normalized next
        if mask is not None:
            # Cast the mask to floatX to avoid float64 upcasting in theano
            a *= K.cast(mask, K.floatx())

        # in some cases especially in the early stages of training the sum may be almos
t zero
        a /= K.cast(K.sum(a, axis=1, keepdims=True) + K.epsilon(), K.floatx())

        a = K.expand_dims(a)
        weighted_input = x * a
    #print weigthted_input.shape
        return K.sum(weighted_input, axis=1)

    def compute_output_shape(self, input_shape):
        #return input_shape[0], input_shape[-1]
        return input_shape[0],  self.features_dim
```

In [41]:

```
encoder_input_image = Input(shape=(2048,), name='image_feature')
x = Dropout(0.5)(encoder_input_image)
encoder_output_image = Dense(128)(x)

encoder_image = Model(encoder_input_image, encoder_output_image, name='encoder_image')
encoder_image.summary()
```

```
Model: "encoder_image"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
image_feature (InputLayer)   (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               262272
=================================================================
Total params: 262,272
Trainable params: 262,272
Non-trainable params: 0
_________________________________________________________________
```

In [42]:

```
encoder_input_text = Input(shape=(max_caption_length,))
x = Embedding(vocab_size, embedding_dim, mask_zero=True)(encoder_input_text) #vocab_siz
e = vocab_size-1 beacuse as mask_zero = False here
x = Dropout(0.5)(x)
x = Bidirectional(GRU(128, activation='relu', return_sequences=True))(x)
x = TimeDistributed(Dense(128))(x)
encoder_output_text = Custom_Attention(max_caption_length)(x)
#encoder_output_text = Dense(128)(x)

encoder_text = Model(encoder_input_text, encoder_output_text, name='encoder_text')
encoder_text.summary()
```

```
Model: "encoder_text"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 39)                0
_________________________________________________________________
embedding_1 (Embedding)      (None, 39, 300)           588000
_________________________________________________________________
dropout_2 (Dropout)          (None, 39, 300)           0
_________________________________________________________________
bidirectional_1 (Bidirection (None, 39, 256)           329472
_________________________________________________________________
time_distributed_1 (TimeDist (None, 39, 128)           32896
_________________________________________________________________
custom__attention_1 (Custom_ (None, 128)               167
=================================================================
Total params: 950,535
Trainable params: 950,535
Non-trainable params: 0
_________________________________________________________________
```

In [43]:

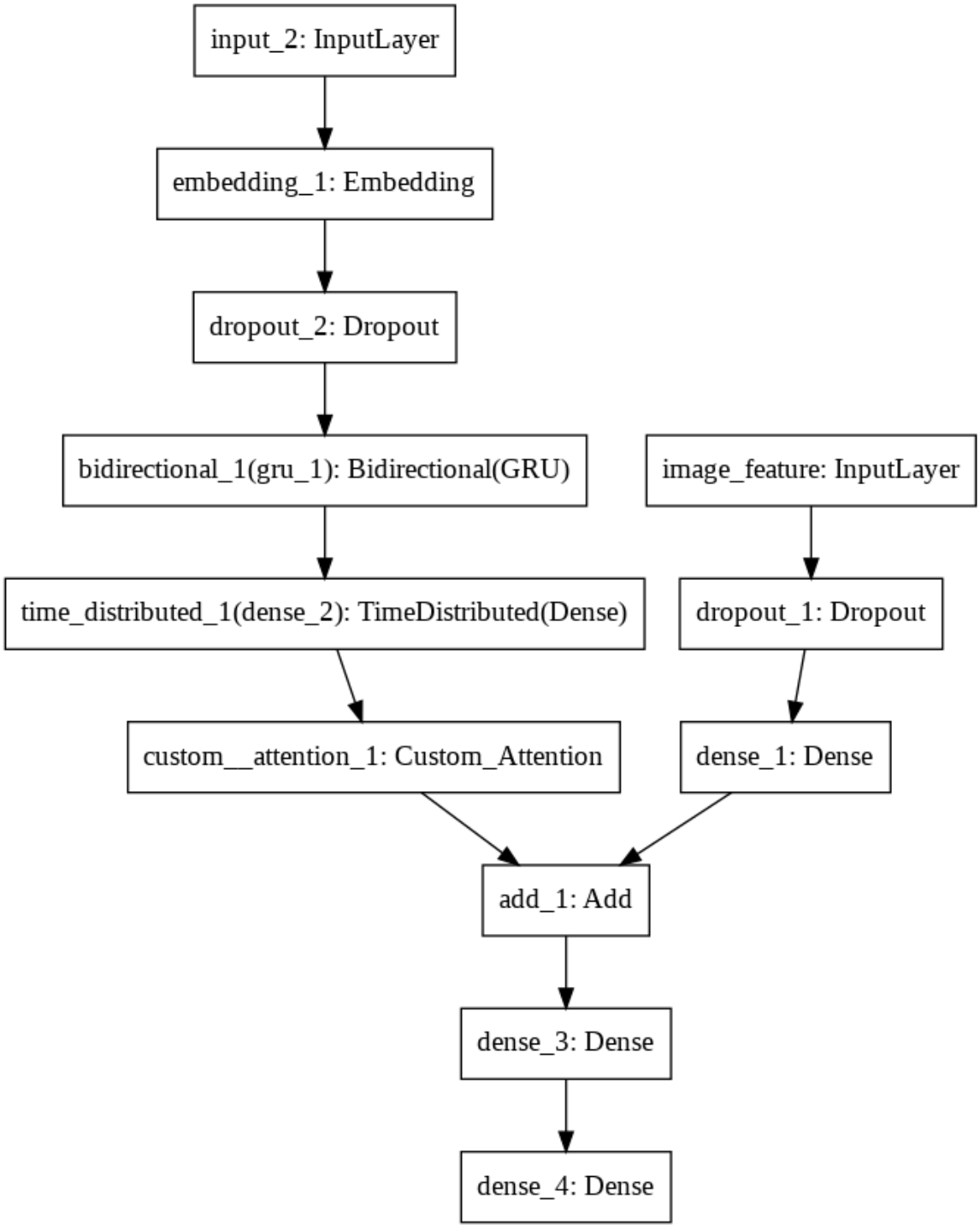
```
decoder_input = add([encoder_output_image, encoder_output_text])
x = Dense(256, activation='relu')(decoder_input)
decoder_output = Dense(vocab_size, activation='softmax')(x)
model = Model(inputs=[encoder_input_image, encoder_input_text], outputs=decoder_output,
name='Decoder_Model')
model.summary()
```

```
Model: "Decoder_Model"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected
to
==========================================================================================
input_2 (InputLayer)            (None, 39)           0
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, 39, 300)      588000      input_2
[0][0]
__________________________________________________________________________________________
dropout_2 (Dropout)             (None, 39, 300)      0           embedding
_1[0][0]
__________________________________________________________________________________________
image_feature (InputLayer)      (None, 2048)         0
__________________________________________________________________________________________
bidirectional_1 (Bidirectional) (None, 39, 256)      329472      dropout_2
[0][0]
__________________________________________________________________________________________
dropout_1 (Dropout)             (None, 2048)         0           image_fea
ture[0][0]
__________________________________________________________________________________________
time_distributed_1 (TimeDistrib (None, 39, 128)      32896       bidirecti
onal_1[0][0]
__________________________________________________________________________________________
dense_1 (Dense)                 (None, 128)          262272      dropout_1
[0][0]
__________________________________________________________________________________________
custom__attention_1 (Custom_Att (None, 128)          167         time_dist
ributed_1[0][0]
__________________________________________________________________________________________
add_1 (Add)                     (None, 128)          0           dense_1
[0][0]
                                                                 custom__a
ttention_1[0][0]
__________________________________________________________________________________________
dense_3 (Dense)                 (None, 256)          33024       add_1[0]
[0]
__________________________________________________________________________________________
dense_4 (Dense)                 (None, 1960)         503720      dense_3
[0][0]
==========================================================================================
Total params: 1,749,551
Trainable params: 1,749,551
Non-trainable params: 0
__________________________________________________________________________________________
```

In [44]:

```
plot_model(model, to_file='model.png')
```

Out[44]:



In [45]:

```
%%time
model.layers[1].set_weights([embedding_matrix])
model.layers[1].trainable = False
```

```
CPU times: user 1.24 ms, sys: 4.08 ms, total: 5.32 ms
Wall time: 9.52 ms
```

In [46]:

```
%%time
optimizer = optimizers.Adam(learning_rate=0.001)
model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['categoric
al_accuracy'])
```

```
CPU times: user 35.4 ms, sys: 2.97 ms, total: 38.4 ms
Wall time: 41.3 ms
```

In [47]:

```
%%time
model_loss_path = '/content/drive/My Drive/Colab Notebooks/flickr8k/attention_model_los
s_list.data'
def load_loss():
  list_of_losses = []
  if not os.path.exists(model_loss_path):
    return list_of_losses
  else:
    with open(model_loss_path, "r") as f:
      for line in f:
        list_of_losses.append(float(line.strip()))
      f.close()
    return list_of_losses
```

```
CPU times: user 6 µs, sys: 0 ns, total: 6 µs
Wall time: 10.3 µs
```

In [48]:

```
%%time
model_loss_path = '/content/drive/My Drive/Colab Notebooks/flickr8k/attention_model_los
s_list.data'
def get_last_loss_value():
  list_of_losses = load_loss()
  if len(list_of_losses) == 0:
    return float('inf')
  else:
    return list_of_losses[-1]
```

```
CPU times: user 5 µs, sys: 1 µs, total: 6 µs
Wall time: 9.3 µs
```

In [49]:

```
%%time
model_loss_path = '/content/drive/My Drive/Colab Notebooks/flickr8k/attention_model_los
s_list.data'
def save_loss(loss):
  list_of_losses = load_loss()
  list_of_losses.append(round(loss,4))
  with open(model_loss_path, "w") as f:
    for s in list_of_losses:
        f.write(str(s) +"\n")
    f.close()
  print('Loss Value Saved in file.')
```

```
CPU times: user 6 µs, sys: 1 µs, total: 7 µs
Wall time: 10.5 µs
```

In [50]:

```
%%time
model_weights_save_path = '/content/drive/My Drive/Colab Notebooks/flickr8k/attention_model.h5'
if os.path.exists(model_weights_save_path):
  print('Weights Found and Loaded')
  model.load_weights(model_weights_save_path)
else:
  print('No Weights found, Train from scratch')
```

```
Weights Found and Loaded
CPU times: user 23.4 ms, sys: 8.95 ms, total: 32.3 ms
Wall time: 1.5 s
```

In [0]:

```
epochs = 10
steps = 1000
num_of_images_per_batch = 3
```

In [0]:

```
%%time
for i in range(epochs):
  print('-'*30,'Batch ',i+1,'-'*30)
  generator = train_generator(num_of_images_per_batch)
  hist = model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
  current_loss = hist.history['loss'][0]
  previous_loss_values = get_last_loss_value()
  if current_loss < previous_loss_values:
    print('Loss reduced from ',previous_loss_values,' to ',round(current_loss,4))
    save_loss(current_loss)
    print('Saving weights in file.')
    model.save(model_weights_save_path)
  else :
    print('Previous Loss : ',previous_loss_values,' and Current Loss : ',round(current_
loss,4))
    print('Not Saving weights.')
```

```
------------------------------ Batch  1 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 214s 214ms/step - loss: 2.689
1 - categorical_accuracy: 0.3755
Loss reduced from  2.7324  to  2.7209
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  2 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 211s 211ms/step - loss: 2.656
0 - categorical_accuracy: 0.3777
Loss reduced from  2.7209  to  2.6886
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  3 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.648
2 - categorical_accuracy: 0.3801
Loss reduced from  2.6886  to  2.6815
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  4 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.609
4 - categorical_accuracy: 0.3841
Loss reduced from  2.6815  to  2.643
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  5 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 216s 216ms/step - loss: 2.584
9 - categorical_accuracy: 0.3876
Loss reduced from  2.643  to  2.6189
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  6 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 214s 214ms/step - loss: 2.584
0 - categorical_accuracy: 0.3881
Loss reduced from  2.6189  to  2.6178
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  7 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.574
3 - categorical_accuracy: 0.3887
Loss reduced from  2.6178  to  2.6094
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  8 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 211s 211ms/step - loss: 2.550
6 - categorical_accuracy: 0.3911
Loss reduced from  2.6094  to  2.5874
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  9 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 211s 211ms/step - loss: 2.529
5 - categorical_accuracy: 0.3925
Loss reduced from  2.5874  to  2.5659
```

```
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  10 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 211s 211ms/step - loss: 2.531
4 - categorical_accuracy: 0.3936
Previous Loss :  2.5659  and Current Loss :  2.5667369550906343
Not Saving weights.
CPU times: user 55min 13s, sys: 7min 35s, total: 1h 2min 48s
Wall time: 35min 40s
```

In [0]:

```
epochs = 10
steps = 1000
num_of_images_per_batch = 3
model.optimizer = optimizers.Adam(learning_rate=0.0001)
```

In [0]:

```
%%time
for i in range(epochs):
  print('-'*30,'Batch ',i+1,'-'*30)
  generator = train_generator(num_of_images_per_batch)
  hist = model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
  current_loss = hist.history['loss'][0]
  previous_loss_values = get_last_loss_value()
  if current_loss < previous_loss_values:
    print('Loss reduced from ',previous_loss_values,' to ',round(current_loss,4))
    save_loss(current_loss)
    print('Saving weights in file.')
    model.save(model_weights_save_path)
  else :
    print('Previous Loss : ',previous_loss_values,' and Current Loss : ',round(current_
loss,4))
    print('Not Saving weights.')
```

```
------------------------------ Batch  1 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 212s 212ms/step - loss: 2.541
9 - categorical_accuracy: 0.3944
Previous Loss :  2.5659  and Current Loss :  2.579
Not Saving weights.
------------------------------ Batch  2 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 211s 211ms/step - loss: 2.493
3 - categorical_accuracy: 0.3988
Loss reduced from  2.5659  to  2.5297
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  3 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 214s 214ms/step - loss: 2.494
7 - categorical_accuracy: 0.4009
Previous Loss :  2.5297  and Current Loss :  2.5322
Not Saving weights.
------------------------------ Batch  4 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 214s 214ms/step - loss: 2.488
3 - categorical_accuracy: 0.3981
Loss reduced from  2.5297  to  2.5254
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  5 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 214s 214ms/step - loss: 2.461
3 - categorical_accuracy: 0.4035
Loss reduced from  2.5254  to  2.4991
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  6 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 212s 212ms/step - loss: 2.462
8 - categorical_accuracy: 0.4066
Previous Loss :  2.4991  and Current Loss :  2.5011
Not Saving weights.
------------------------------ Batch  7 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 212s 212ms/step - loss: 2.450
3 - categorical_accuracy: 0.4060
Loss reduced from  2.4991  to  2.4885
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  8 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 216s 216ms/step - loss: 2.466
4 - categorical_accuracy: 0.4015
Previous Loss :  2.4885  and Current Loss :  2.5048
Not Saving weights.
------------------------------ Batch  9 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 217s 217ms/step - loss: 2.445
7 - categorical_accuracy: 0.4043
Loss reduced from  2.4885  to  2.4848
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  10 ------------------------------
Epoch 1/1
```

```
1000/1000 [==============================] - 221s 221ms/step - loss: 2.429
8 - categorical_accuracy: 0.4081
Loss reduced from  2.4848  to  2.4688
Loss Value Saved in file.
Saving weights in file.
CPU times: user 55min 49s, sys: 7min 40s, total: 1h 3min 29s
Wall time: 35min 55s
```

In [0]:

```
epochs = 20
steps = 1000
num_of_images_per_batch = 3
model.optimizer = optimizers.Adam(learning_rate=0.0001)
```

In [0]:

```
%%time
for i in range(epochs):
  print('-'*30,'Batch ',i+1,'-'*30)
  generator = train_generator(num_of_images_per_batch)
  hist = model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
  current_loss = hist.history['loss'][0]
  previous_loss_values = get_last_loss_value()
  if current_loss < previous_loss_values:
    print('Loss reduced from ',previous_loss_values,' to ',round(current_loss,4))
    save_loss(current_loss)
    print('Saving weights in file.')
    model.save(model_weights_save_path)
  else :
    print('Previous Loss : ',previous_loss_values,' and Current Loss : ',round(current_
loss,4))
    print('Not Saving weights.')
```

```
------------------------------ Batch  1 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 222s 222ms/step - loss: 2.419
2 - categorical_accuracy: 0.4152
Loss reduced from  2.4688  to  2.4579
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  2 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 218s 218ms/step - loss: 2.330
0 - categorical_accuracy: 0.4262
Loss reduced from  2.4579  to  2.3692
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  3 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 216s 216ms/step - loss: 2.298
6 - categorical_accuracy: 0.4307
Loss reduced from  2.3692  to  2.3378
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  4 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 218s 218ms/step - loss: 2.261
3 - categorical_accuracy: 0.4347
Loss reduced from  2.3378  to  2.3024
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  5 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 218s 218ms/step - loss: 2.241
5 - categorical_accuracy: 0.4405
Loss reduced from  2.3024  to  2.284
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  6 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 216s 216ms/step - loss: 2.223
1 - categorical_accuracy: 0.4421
Loss reduced from  2.284  to  2.265
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  7 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 212s 212ms/step - loss: 2.201
7 - categorical_accuracy: 0.4447
Loss reduced from  2.265  to  2.244
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  8 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 211s 211ms/step - loss: 2.188
9 - categorical_accuracy: 0.4476
Loss reduced from  2.244  to  2.2324
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  9 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 212s 212ms/step - loss: 2.169
0 - categorical_accuracy: 0.4501
Loss reduced from  2.2324  to  2.2126
```

```
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  10 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.163
7 - categorical_accuracy: 0.4507
Loss reduced from  2.2126  to  2.2063
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  11 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.155
2 - categorical_accuracy: 0.4505
Loss reduced from  2.2063  to  2.1985
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  12 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.139
7 - categorical_accuracy: 0.4527
Loss reduced from  2.1985  to  2.1831
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  13 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 2.133
7 - categorical_accuracy: 0.4535
Loss reduced from  2.1831  to  2.1783
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  14 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.119
2 - categorical_accuracy: 0.4569
Loss reduced from  2.1783  to  2.1639
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  15 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.115
9 - categorical_accuracy: 0.4555
Loss reduced from  2.1639  to  2.1602
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  16 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 2.101
1 - categorical_accuracy: 0.4576
Loss reduced from  2.1602  to  2.1459
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  17 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.104
2 - categorical_accuracy: 0.4586
Previous Loss :  2.1459  and Current Loss :  2.1477
Not Saving weights.
----------------------------- Batch  18 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 207s 207ms/step - loss: 2.093
7 - categorical_accuracy: 0.4604
```

```
Loss reduced from  2.1459  to  2.1387
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  19 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.087
0 - categorical_accuracy: 0.4598
Loss reduced from  2.1387  to  2.1322
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  20 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 207s 207ms/step - loss: 2.080
8 - categorical_accuracy: 0.4612
Loss reduced from  2.1322  to  2.1258
Loss Value Saved in file.
Saving weights in file.
CPU times: user 1h 50min 43s, sys: 15min 3s, total: 2h 5min 46s
Wall time: 1h 11min 6s
```

In [0]:

```
%%time
for i in range(epochs):
  print('-'*30,'Batch ',i+1,'-'*30)
  generator = train_generator(num_of_images_per_batch)
  hist = model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
  current_loss = hist.history['loss'][0]
  previous_loss_values = get_last_loss_value()
  if current_loss < previous_loss_values:
    print('Loss reduced from ',previous_loss_values,' to ',round(current_loss,4))
    save_loss(current_loss)
    print('Saving weights in file.')
    model.save(model_weights_save_path)
  else :
    print('Previous Loss : ',previous_loss_values,' and Current Loss : ',round(current_
loss,4))
    print('Not Saving weights.')
```

```
------------------------------ Batch  1 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.073
7 - categorical_accuracy: 0.4634
Loss reduced from  2.1258  to  2.1181
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  2 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.059
1 - categorical_accuracy: 0.4637
Loss reduced from  2.1181  to  2.1041
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  3 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 210s 210ms/step - loss: 2.059
3 - categorical_accuracy: 0.4640
Previous Loss :  2.1041  and Current Loss :  2.1054
Not Saving weights.
------------------------------ Batch  4 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.053
3 - categorical_accuracy: 0.4674
Loss reduced from  2.1041  to  2.0991
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  5 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.051
0 - categorical_accuracy: 0.4653
Loss reduced from  2.0991  to  2.0961
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  6 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 2.041
6 - categorical_accuracy: 0.4671
Loss reduced from  2.0961  to  2.0868
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  7 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.037
0 - categorical_accuracy: 0.4689
Loss reduced from  2.0868  to  2.0818
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  8 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.025
1 - categorical_accuracy: 0.4677
Loss reduced from  2.0818  to  2.0699
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  9 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 2.021
9 - categorical_accuracy: 0.4698
Loss reduced from  2.0699  to  2.068
Loss Value Saved in file.
```

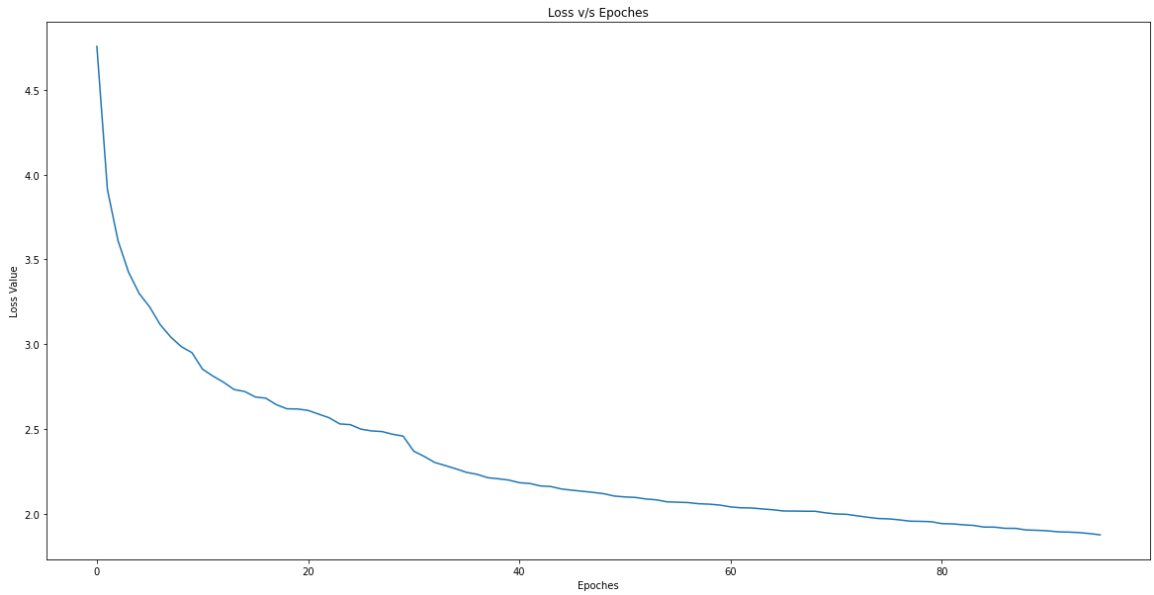
```
Saving weights in file.
----------------------------- Batch  10 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 2.019
9 - categorical_accuracy: 0.4715
Loss reduced from  2.068  to  2.0657
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  11 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 2.012
8 - categorical_accuracy: 0.4705
Loss reduced from  2.0657  to  2.0589
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  12 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 212s 212ms/step - loss: 2.010
8 - categorical_accuracy: 0.4727
Loss reduced from  2.0589  to  2.0564
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  13 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 2.013
0 - categorical_accuracy: 0.4712
Previous Loss :  2.0564  and Current Loss :  2.0578
Not Saving weights.
----------------------------- Batch  14 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 2.006
6 - categorical_accuracy: 0.4738
Loss reduced from  2.0564  to  2.0509
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  15 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 1.995
8 - categorical_accuracy: 0.4746
Loss reduced from  2.0509  to  2.0403
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  16 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 208s 208ms/step - loss: 1.989
7 - categorical_accuracy: 0.4772
Loss reduced from  2.0403  to  2.0349
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  17 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 209s 209ms/step - loss: 1.989
3 - categorical_accuracy: 0.4759
Loss reduced from  2.0349  to  2.0339
Loss Value Saved in file.
Saving weights in file.
----------------------------- Batch  18 -----------------------------
Epoch 1/1
1000/1000 [==============================] - 214s 214ms/step - loss: 1.982
8 - categorical_accuracy: 0.4759
Loss reduced from  2.0339  to  2.0283
```

```
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  19 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 218s 218ms/step - loss: 1.978
6 - categorical_accuracy: 0.4757
Loss reduced from  2.0283  to  2.0234
Loss Value Saved in file.
Saving weights in file.
------------------------------ Batch  20 ------------------------------
Epoch 1/1
1000/1000 [==============================] - 215s 215ms/step - loss: 1.971
5 - categorical_accuracy: 0.4788
Loss reduced from  2.0234  to  2.0162
Loss Value Saved in file.
Saving weights in file.
CPU times: user 1h 49min 43s, sys: 14min 57s, total: 2h 4min 40s
Wall time: 1h 10min 30s
```

Plot Loss Values for our Model :

In [51]:

```
loss_values = load_loss()
plt.figure(figsize=(20,10))
plt.plot(loss_values)
plt.xlabel('Epoches')
plt.ylabel('Loss Value')
plt.title('Loss v/s Epoches')
plt.show()
```



Creating a function to predict caption from test images :

In [52]:

```
%%time
# prepare data for the LSTM
def get_caption_for_photo(filename):
  imageFile = os.path.join(image_dir_path,filename)
  img = cv2.imread(imageFile)
  img = cv2.resize(img,(299,299))
  cv2_imshow(img)
  photo_feature = np.reshape(photo_embeddings_test[filename],(1,2048))
  in_text = 'startSeq'
  for i in range(1, max_caption_length):
    seq = [word_index_Mapping[w] for w in in_text.split() if w in word_index_Mapping]
    in_seq = pad_sequences([seq], maxlen=max_caption_length)
    inputs = [photo_feature,in_seq]
    yhat = model.predict(x=inputs, verbose=0)
    yhat = np.argmax(yhat)
    word = index_word_Mapping[yhat]
    in_text += ' ' + word
    if word == 'endSeq':
      break

  final = in_text.split()
  final = final[1:-1]
  final = ' '.join(final)
  #https://datascience.stackexchange.com/questions/34039/regex-to-remove-repeating-word
s-in-a-sentence
  return re.sub(r'\b(\w+)( \1\b)+', r'\1', final)
```

```
CPU times: user 5 µs, sys: 1e+03 ns, total: 6 µs
Wall time: 8.82 µs
```

Let's See some relevant Image Captioning :

In [0]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a boy is running on a game

In [0]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a person is kayaking over a kayak

In [0]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a brown greyhound is jumping over hurdles

In [0]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a dog is running through the snow

In [0]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a kayaker is splashed across a kayak

In [0]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a black and white dog is running through a grassy field

In [80]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a girl in a bathing suit is jumping into the water

In [83]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a brown and black dog is running through the snow

In [72]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a person in a snow covered mountain

Let's See some Irrelavant Captioning :

In [73]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a little girl is jumping into a pool of water

In [74]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

two children in the snow dogs and a bright red ball

In [75]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a man in a red jacket and white shorts is standing on a concrete wall

In [77]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a group of people are smoking a plate

In [81]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a man in a red shirt is sitting on a railing in a city street

In [86]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a man in a blue shirt is sitting on a stone wall

In [87]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a young boy in a bathing suit is sitting on a yellow rope

In [88]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a man is doing a trick on a concrete

In [89]:

```
caption = get_caption_for_photo(test_df.filename.unique()[random.randint(1,1000)])
print('\n\n',caption)
```

a little boy is laying on a pole

References :

https://www.tensorflow.org/tutorials/text/image_captioning (https://www.tensorflow.org/tutorials/text/image_captioning)

https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8 (https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8)

https://github.com/hlamba28/Automatic-Image-Captioning (https://github.com/hlamba28/Automatic-Image-Captioning)

https://www.kaggle.com/shadabhussain/flickr8k (https://www.kaggle.com/shadabhussain/flickr8k)

https://machinelearningmastery.com/encoder-decoder-attention-sequence-to-sequence-prediction-keras/ (https://machinelearningmastery.com/encoder-decoder-attention-sequence-to-sequence-prediction-keras/)

https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/ (https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/)

https://www.tensorflow.org/guide/keras/functional (https://www.tensorflow.org/guide/keras/functional)

https://www.kaggle.com/suicaokhoailang/10-fold-lstm-with-attention-0-991-lb (https://www.kaggle.com/suicaokhoailang/10-fold-lstm-with-attention-0-991-lb)