IMAGE CAPTION GENERATION AND TRANSLATION

Group Number: **G13**

SUBMITTED BY:
Guneet Kohli (1805172)
Jashanpreet Kaur (1805188)
Kartika (1805192)

INTRODUCTION

In Image Caption Generation concepts of Computer Vision and Natural Language processing are applied to recognize the context of image and describe the context of Image in some Natural Language like English and using Machine Translation and Natural Language Processing the caption can be described in domain language.

- Caption generation is the challenging artificial intelligence problem of generating a human-readable textual description given a photograph.
- It requires both image understanding from the domain of computer vision and a language model from the field of natural language processing.
- It is important to consider and test multiple ways to frame a given predictive modeling problem and there are indeed many ways to frame the problem of generating captions for photographs.

Why do we need to caption images?

USE CASES OF IMAGE CAPTION GENERATION

- A visually impaired person taking a picture from his phone and then the caption generator will turn the caption to speech for him to understand.
- Advertising industry tries to generate captions automatically without the need to make them separately during production and sales.
- Doctors can use this technology to find tumors or some defects in the images or used by people for understanding geospatial images where they can find out more details about the terrain.

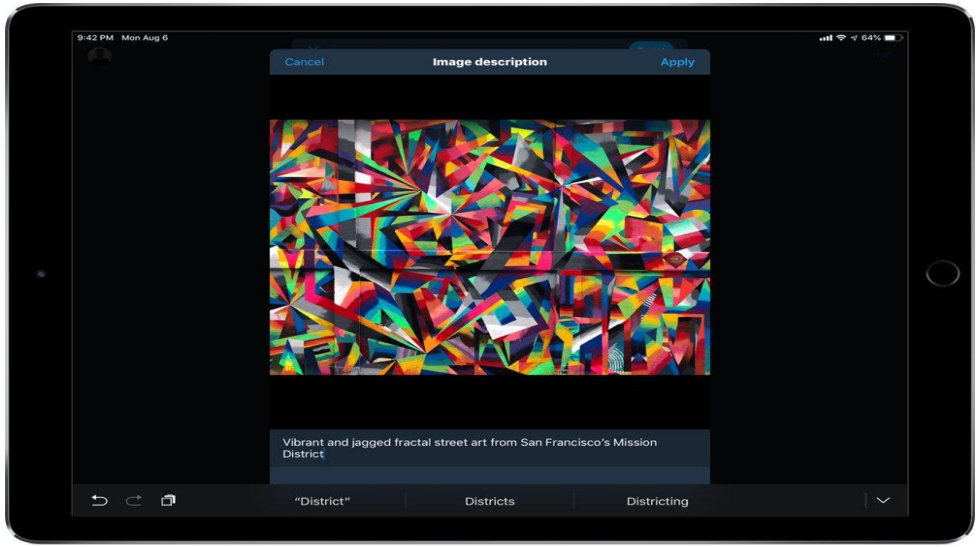
- By helping visually impaired people better understand the content of images on the web. For example an visually impaired person can take a picture from his phone and then the caption generator will turn the caption to speech for him/her to understand better.

- Doctors can use this technology to find tumors or some defects in the images. Since captions can be generated from the scanned images of human parts, it will be feasible to understand the problem.

- It can be used by people for understanding geospatial images where they can find out more details about the terrain.

- It can help make web search more powerful and interactive,since after generation of captions we can search effectively using images as well.
- It can be used for automatic image indexing. Image indexing is important for Content-Based Image Retrieval (CBIR) and therefore, it can be applied to many areas, including biomedicine, commerce, the military, education, digital libraries, and web searching.

- Social media platforms such as Facebook and Twitter can directly generate descriptions from images. The descriptions can include where we are (e.g., beach, cafe), what we wear and importantly what we are doing there.



Objectives

Objectives

- To clean and process the Flickr_8k dataset for feature extraction and training purpose.

- To analyse a given image using VGG-16

Objectives

- To implement CNN, LSTM and/or other deep learning models to model the extracted features for caption generation from images.

- To train the model to maximize the likelihood of the target sentences for a given image.

APPROACH AND TECHNIQUES USED IN OUR MODEL

BOTTOM-UP APPROACH

- Bottom Up approach generates items observed in an image, and then attempts to combine the items identified into a caption.

TOP-DOWN APPROACH

- Top down approach attempts to generate a semantic representation of an image that is then decoded into a caption using various architectures, such as recurrent neural networks.
- The top down approach follows in the footsteps of recent advances in statistical machine translation, and the state-of-the-art models mostly adopt the top-down approach.

CNNs

- CNNs(Convolutional Neural networks) can produce a rich representation of the input image by embedding it to a fixed-length vector, such that this representation can be used for a variety of vision tasks

CNNs

- CNN will be used as an image “encoder”, by first pre-training it for an image classification task and using last hidden layer as an input to the RNN decoder that generates sentences.This model can be referred to as Neural Image Caption.

- The image features can also be extracted from Xception or VGG-16 or Inception V3 which are CNN models trained on imagenet dataset and then we can feed these features into LSTM model which will be responsible for generating the image captions.

DATASET USED AND PROCESSING OF TRAINING DATA

- For training,the FLICKR_8K dataset will be used. It is a labeled dataset consisting of 8000 photos with 5 captions for each photo written by different people for each image. It includes images obtained from the Flickr website.

METHODOLOGY

1. Cleaning the captions

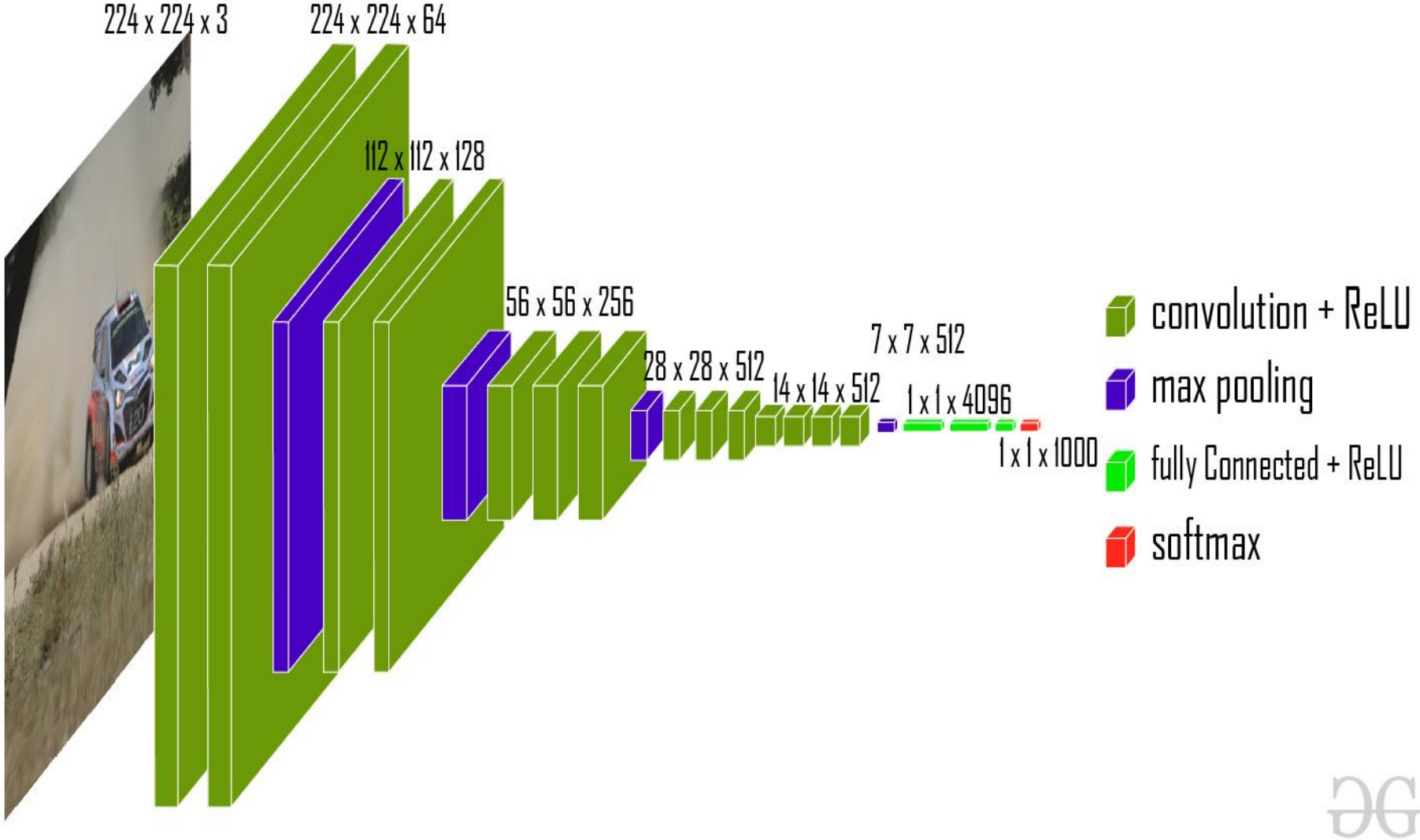
This is the first step of data pre-processing. The captions contain regular expressions, numbers and other stop words which need to be cleaned before they are fed to the model for further training. The cleaning part involves removing punctuations, single character and numerical values. After cleaning we try to figure out the top 50 and least 50 words in our dataset.

2. Adding start and end sequence to the captions

Start and end sequence need to be added to the captions because the captions vary in length for each image and the model has to understand the start and the end.

3. Extracting features from images

- After dealing with the captions images will be processed.For this we can make use of the pretrained VGG-16 weights.
- VGG16 will be used for extracting the features from the images. In order to do that we need to get rid of the last output layer from the model. The model then generates 4096 features from taking images of size (224,224,3).


224 x 224 x 3
224 x 224 x 64
112 x 112 x 128
56 x 56 x 256
28 x 28 x 512
14 x 14 x 512
7 x 7 x 512
1 x 1 x 4096
1 x 1 x 1000
convolution + ReLU
max pooling
fully Connected + ReLU
softmax

4. Viewing similar images

When the VGG-16 model finishes extracting features from all the images from the dataset, similar images from the clusters are displayed together to see if the VGG-16 model has extracted the features correctly and we are able to see them together.

5. Merging the caption with the respective images

- The next step involves merging the captions with the respective images so that they can be used for training.Only the first caption of each image will be taken from the dataset as it becomes complicated to train with all 5 of them.
- Then we have to tokenize all the captions before feeding it to the model

6. Splitting the data for training and testing

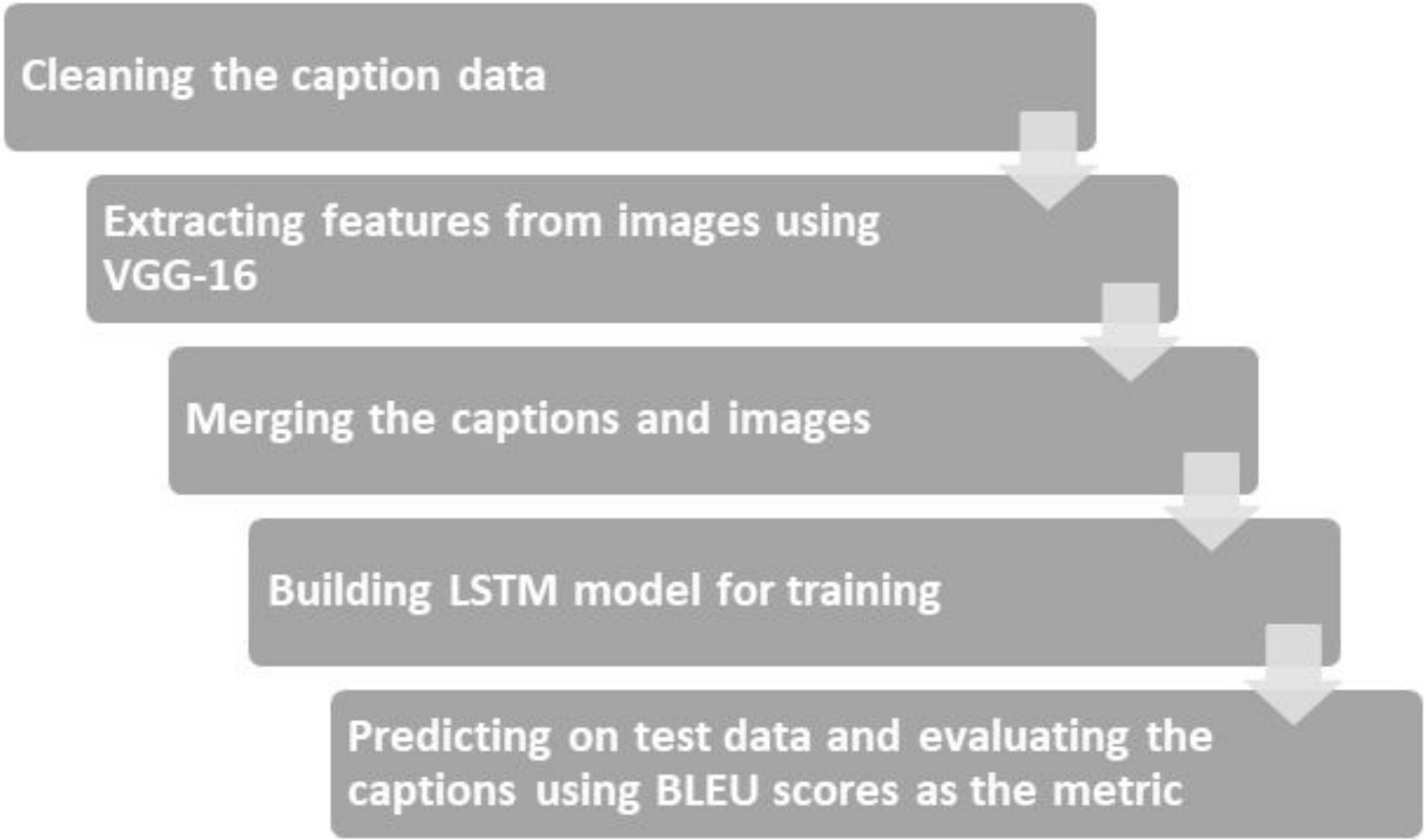
The tokenized captions along with the image data are split into training, test and validation sets as required and are then pre-processed as required for the input for the model.

7. Building the LSTM model

- LSTM model will be used because it takes into consideration the state of the previous cell's output and the present cell's input for the current output. This is useful while generating the captions for the images.

8. Predicting on the test dataset and evaluating using BLEU scores

These generated captions are compared to the actual captions from the dataset and evaluated using BLEU (Bilingual Evaluation Understudy)scores as the evaluation metrics. A score closer to 1 indicates that the predicted and actual captions are very similar.


Cleaning the caption data
Extracting features from images using VGG-16
Merging the captions and images
Building LSTM model for training
Predicting on test data and evaluating the captions using BLEU scores as the metric

MODEL EXPLAINED

Image Caption Generator Model

This is a CNN-RNN model.

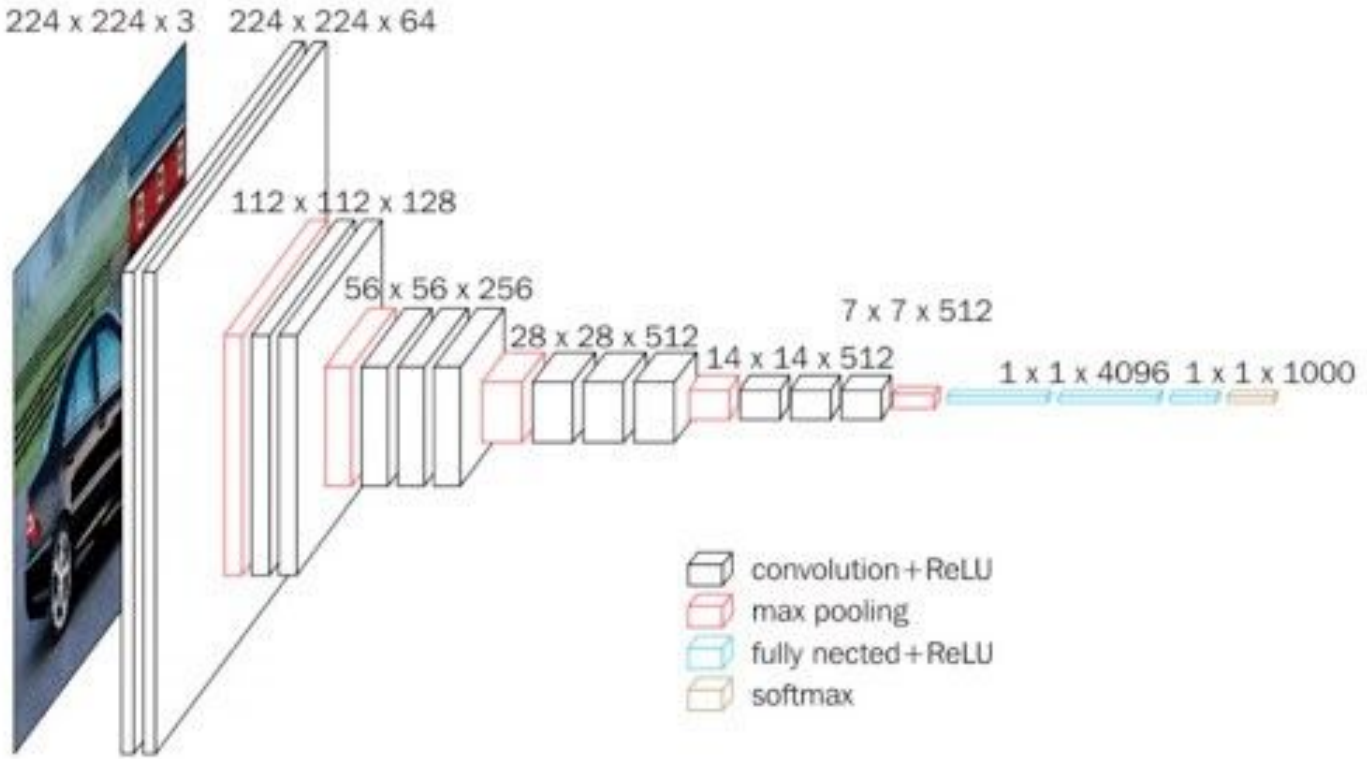
- CNN is used for extracting features from the image. We will use the pre-trained model VGG16.
- LSTM will use the information from CNN to help generate a description of the image.

Architecture

Image Features Detection :

- For image Detecting, we are using a pre-trained model which is VGG16. VGG16 is already installed in the Keras library.

VGG16(Pre-trained model)

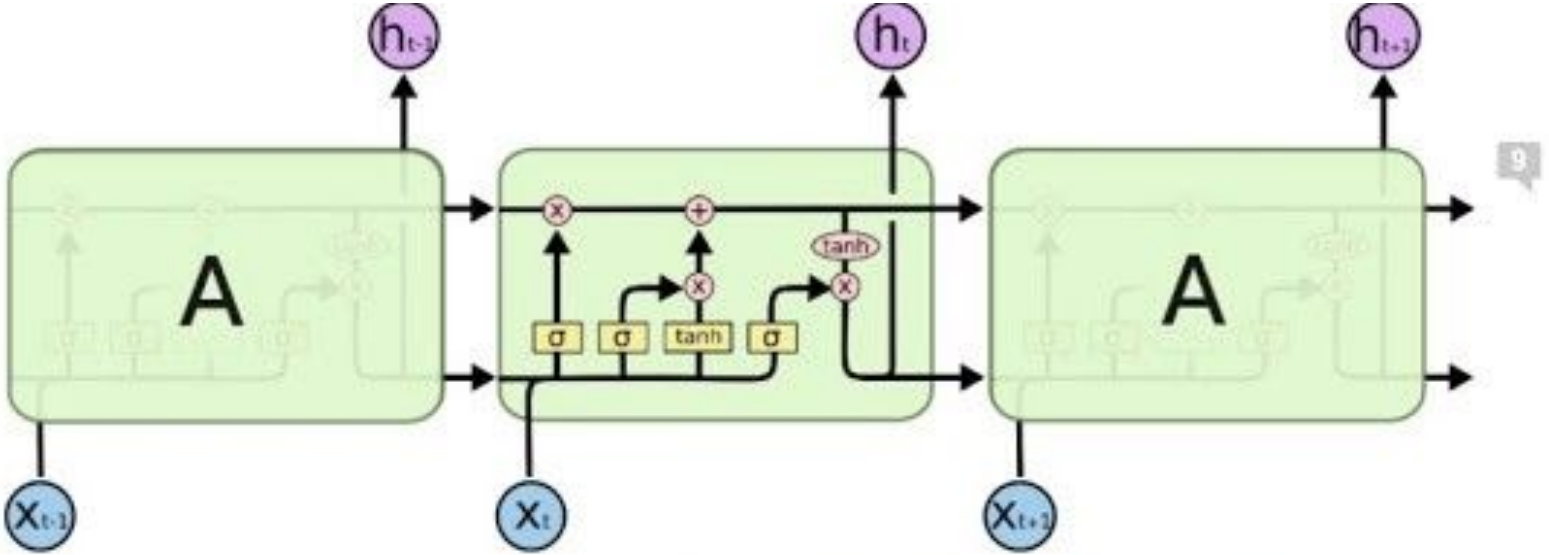


Architecture

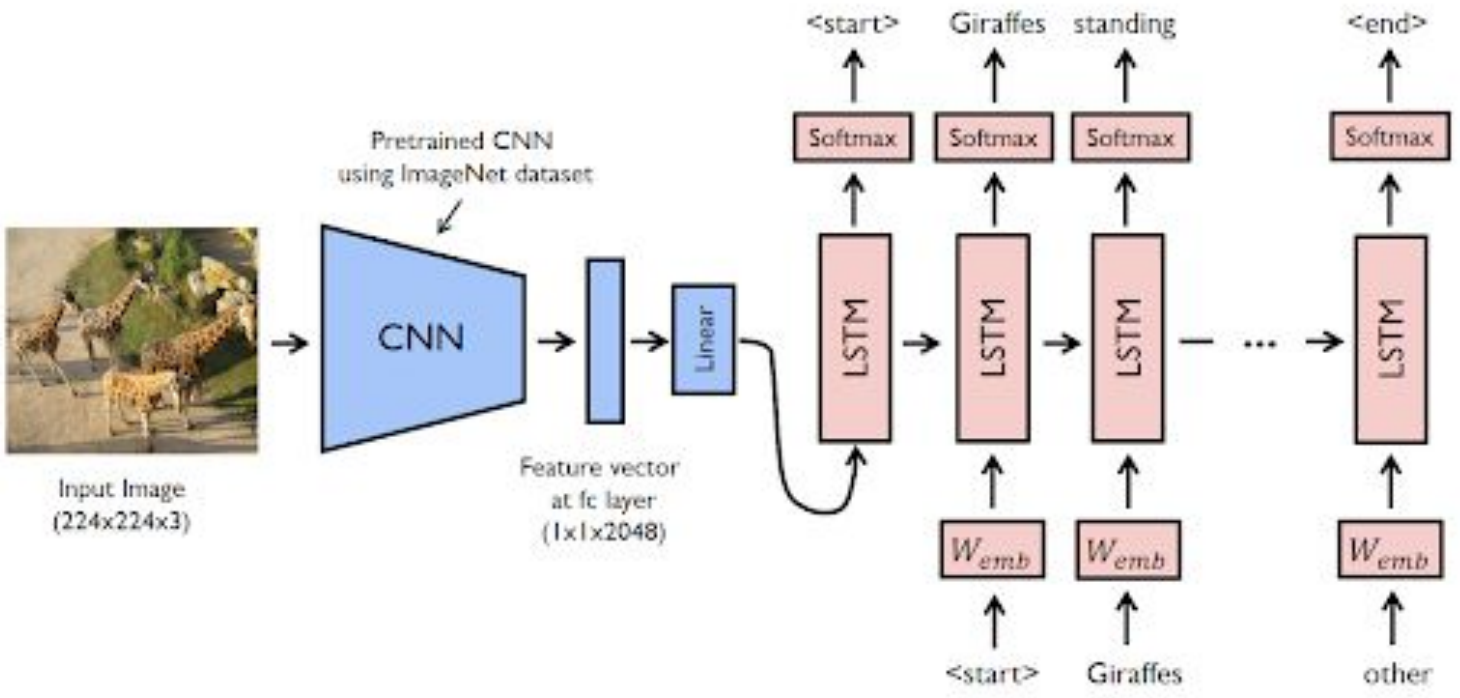
Text Generation using LSTM:

Long Short Term Memory networks — usually just called “LSTMs” — are a special kind of RNN, capable of learning long-term dependencies. They work tremendously well on a large variety of problems and are now widely used.

LSTM



COMBINED MODEL ARCHITECTURE



CODING

Files | Running | Clusters

Select items to perform actions on them.

Upload | New | ↻

0 / Image-Caption-Generator-master

	Name	Last Modified	File size
..		seconds ago	
__results___files		a month ago	
Flickr_Data		a month ago	
Images		a month ago	
logs		22 days ago	
saved_models		2 years ago	
Image Captioning 8k.ipynb		13 hours ago	9.43 MB
Untitled.ipynb		22 days ago	5.49 kB
captions.txt		a year ago	3.32 MB
Image Captioning 8k - Jupyter Notebook.pdf		14 hours ago	3.22 MB
LICENSE		2 years ago	1.07 kB
model.json		2 years ago	4.9 kB
model_weights.h5		2 years ago	30 MB
README.md		2 years ago	6.04 kB
train_encoded_images.p		2 years ago	63.7 MB

jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

Upload New

0 / Image-Caption-Generator-master / Flickr_Data / Flickr_Data

	Name	Last Modified	File size
..		seconds ago	
flickr8ktextfiles		a month ago	
Flickr_TextData		a month ago	
Images		a month ago	
vgg16_weights_tf_dim_ordering_tf_kernels.h5		24 days ago	553 MB

Image Caption Generator

Image caption generator is a model which generates caption based on the features present in the input image.

Introduction

The basic working of the project is that the features are extracted from the images using pre-trained VGG16 model and then fed to the LSTM model along with the captions to train. The trained model is then capable of generating captions for any images that are fed to it.

Dataset

The dataset used here is the **FLICKR 8K** which consists of around 8091 images along with 5 captions for each images. If you have a powerful system with more than 16 GB RAM and a graphic card with more than 4 GB of memory, you can try to take **[FLICKR 30K] (http://web.engr.illinois.edu/~bplumme2/Flickr30kEntities/)** which has around 30,000 images with captions.

Dependencies

- Keras
- Tensorflow GPU
- Pre-trained VGG-16 weights
- NLTK
- Matplotlib

Steps to follow:

Importing the required libraries

In [2]:
```
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.backend.tensorflow_backend import set_session
import keras
import sys, time, os, warnings
import numpy as np
import pandas as pd
from collections import Counter
warnings.filterwarnings("ignore")
print("python {}".format(sys.version))
print("keras version {}".format(keras.__version__)); del keras
print("tensorflow version {}".format(tf.__version__))
```

```
python 3.6.9 (default, Jan 26 2021, 15:33:00)
[GCC 8.4.0]
keras version 2.1.3
tensorflow version 1.3.0
```

Configuring the GPU memory to be used for training purposes

In [3]:
```
config = tf.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.8#Using 95% of the available memory of the GPU
config.gpu_options.visible_device_list = "0"
set_session(tf.Session(config=config))

def set_seed(sd=144):
    from numpy.random import seed
    from tensorflow import set_random_seed
    import random as rn
    ## numpy random seed
    seed(sd)
    ## core python's random number
    rn.seed(sd)
    ## tensor flow's random number
    set_random_seed(sd)
```

Importing the image dataset and its respective captions

Import caption data

Load the text data and save it into a pandas dataframe.

filename : jpg file name

index : unique ID for each caption for the same image

caption : string of caption, all in lower case

In [4]:
```
## The location of the Flickr8K_ images
dir_Flickr_jpg = "Flickr_Data/Flickr_Data/Images"
## The location of the caption file
dir_Flickr_text = "Flickr_Data/Flickr_Data/Flickr_TextData/Flickr8k.token.txt"

jpgs = os.listdir(dir_Flickr_jpg)
print("The number of jpg flies in Flicker8k: {}".format(len(jpgs)))
```

```
The number of jpg flies in Flicker8k: 8091
```

In [5]:
```
## read in the Flickr caption data
file = open(dir_Flickr_text,'r')
text = file.read()
file.close()


datatxt = []
for line in text.split('\n'):
    col = line.split('\t')
    if len(col) == 1:
        continue
    w = col[0].split("#")
    datatxt.append(w + [col[1].lower()])

df_txt = pd.DataFrame(datatxt,columns=["filename","index","caption"])


uni_filenames = np.unique(df_txt.filename.values)
print("The number of unique file names : {}".format(len(uni_filenames)))
print("The distribution of the number of captions for each image:")
Counter(Counter(df_txt.filename.values).values())
```

```
The number of unique file names : 8092
The distribution of the number of captions for each image:
```

Out[5]:
```
Counter({5: 8092})
```

In [6]:
```
#Finding the captions for each image.
file = open(dir_Flickr_text,'r', encoding='utf8')
text = file.read()
file.close()


datatxt = []
for line in text.split('\n'):
    col = line.split('\t')
    if len(col) == 1:
        continue
    w = col[0].split("#") # Splitting the caption dataset at the required position
    datatxt.append(w + [col[1].lower()])
print(datatxt)
df_txt = pd.DataFrame(datatxt,columns=["filename","index","caption"])


uni_filenames = np.unique(df_txt.filename.values)
print("The number of unique file names : {}".format(len(uni_filenames)))
print("The distribution of the number of captions for each image:")
Counter(Counter(df_txt.filename.values).values())
print(df_txt[:5])
```

```
[['1000268201_693b08cb0e.jpg', '0', 'a child in a pink dress is climbing up a set of stairs in an entry way .'],
['1000268201_693b08cb0e.jpg', '1', 'a girl going into a wooden building .'], ['1000268201_693b08cb0e.jpg', '2',
'a little girl climbing into a wooden playhouse .'], ['1000268201_693b08cb0e.jpg', '3', 'a little girl climbing
the stairs to her playhouse .'], ['1000268201_693b08cb0e.jpg', '4', 'a little girl in a pink dress going into a
wooden cabin .'], ['1001773457_577c3a7d70.jpg', '0', 'a black dog and a spotted dog are fighting'], ['1001773457
_577c3a7d70.jpg', '1', 'a black dog and a tri-colored dog playing with each other on the road .'], ['1001773457_
577c3a7d70.jpg', '2', 'a black dog and a white dog with brown spots are staring at each other in the street .'],
['1001773457_577c3a7d70.jpg', '3', 'two dogs of different breeds looking at each other on the road .'], ['100177
3457_577c3a7d70.jpg', '4', 'two dogs on pavement moving toward each other .'], ['1002674143_1b742ab4b8.jpg',
'0', 'a little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .'], ['10026741
43_1b742ab4b8.jpg', '1', 'a little girl is sitting in front of a large painted rainbow .'], ['1002674143_1b742ab
4b8.jpg', '2', 'a small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on i
t .'], ['1002674143_1b742ab4b8.jpg', '3', 'there is a girl with pigtails sitting in front of a rainbow painting
.'], ['1002674143_1b742ab4b8.jpg', '4', 'young girl with pigtails painting outside in the grass .'], ['100316336
6_44323f5815.jpg', '0', 'a man lays on a bench while his dog sits by him .'], ['1003163366_44323f5815.jpg', '1',
'a man lays on the bench to which a white dog is also tied .'], ['1003163366_44323f5815.jpg', '2', 'a man sleepi
ng on a bench outside with a white and black dog sitting next to him .'], ['1003163366_44323f5815.jpg', '3', 'a
shirtless man lies on a park bench with his dog .'], ['1003163366_44323f5815.jpg', '4', 'man laying on bench hol
ding leash of dog sitting on ground'], ['1007129816_e794419615.jpg', '0', 'a man in an orange hat starring at so
mething .'], ['1007129816_e794419615.jpg', '1', 'a man wears an orange hat and glasses .'], ['1007129816_e794419
```

In [7]:
```
from keras.preprocessing.image import load_img, img_to_array
from IPython.display import display
from PIL import Image

npic = 5 # Displaying 5 images from the dataset
npix = 224
target_size = (npix,npix,3)

count = 1
fig = plt.figure(figsize=(10,20))
for jpgfnm in uni_filenames[-5:]:
    filename = dir_Flickr_jpg + '/' + jpgfnm
    captions = list(df_txt["caption"].loc[df_txt["filename"]==jpgfnm].values)
    image_load = load_img(filename, target_size=target_size)

    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ax = fig.add_subplot(npic,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,len(captions))
    for i, caption in enumerate(captions):
        ax.text(0,i,caption,fontsize=20)
    count += 1
plt.show()
```

man on a bicycle riding on only one wheel .
asian man in orange hat is popping a wheelie on his bike .
a man on a bicycle is on only the back wheel .
a man is doing a wheelie on a mountain bike .
a man does a wheelie on his bicycle on the sidewalk .

five people are sitting together in the snow .
five children getting ready to sled .
a group of people sit in the snow overlooking a mountain scene .
a group of people sit atop a snowy mountain .
a group is sitting around a snowy crevasse .

a white crane stands tall as it looks out upon the ocean .
a water bird standing at the ocean 's edge .
a tall bird is standing on the sand beside the ocean .
a large bird stands in the water on the beach .
a grey bird stands majestically on a beach while waves roll in .

woman writing on a pad in room with gold , decorated walls .
the walls are covered in gold and patterns .
a woman standing near a decorated wall writes .
a woman behind a scrolled wall is writing
a person stands near golden walls .

a rock climber practices on a rock climbing wall .
a rock climber in a red shirt .
a person in a red shirt climbing up a rock face covered in assist handles .
a man is rock climbing high in the air .
a man in a pink shirt climbs a rock face

Cleaning captions for further analysis

Text preparation

We create a new dataframe dfword to visualize distribution of the words. It contains each word and its frequency in the entire tokens in decreasing order.

In [8]:
```
# Defining a function to calculate the top 3 words in all the captions available for the images
def df_word(df_txt):
    vocabulary = []
    for txt in df_txt.caption.values:
        vocabulary.extend(txt.split())
    print('Vocabulary Size: %d' % len(set(vocabulary)))
    ct = Counter(vocabulary)
    dfword = pd.DataFrame({"word":list(ct.keys()),"count":list(ct.values())})
    dfword = dfword.sort_values("count",ascending=False)
    dfword = dfword.reset_index()[["word","count"]]
    return(dfword)
dfword = df_word(df_txt)
dfword.head(3)
```

```
Vocabulary Size: 8918
```

Out[8]:

	word	count
0	a	62989
1	.	36581
2	in	18975

Cleaning the captions for further processing

The caption dataset contains punctuations, singular words and numerical values that need to be cleaned before it is fed to the model because uncleaned dataset will not create good captionsfor the images

In order to clean the caption, we will create three functions that:

1. remove punctuation
2. remove single character
3. remove numeric characters

In [9]:
```
import string
text_original = "I ate 1000 apples and a banana. I have python v2.7.It's 2:30 pm. Could you buy me iphone7?"

print(text_original)
print("\nRemove punctuations..")
def remove_punctuation(text_original):
    text_no_punctuation = text_original.translate(str.maketrans('','',string.punctuation))
    return(text_no_punctuation)
text_no_punctuation = remove_punctuation(text_original)
print(text_no_punctuation)


print("\nRemove a single character word..")
def remove_single_character(text):
    text_len_more_than1 = ""
    for word in text.split():
        if len(word) > 1:
            text_len_more_than1 += " " + word
    return(text_len_more_than1)
text_len_more_than1 = remove_single_character(text_no_punctuation)
print(text_len_more_than1)

print("\nRemove words with numeric values..")
def remove_numeric(text,printTF=False):
    text_no_numeric = ""
    for word in text.split():
        isalpha = word.isalpha()
        if printTF:
            print("    {:10} : {:}".format(word,isalpha))
        if isalpha:
            text_no_numeric += " " + word
    return(text_no_numeric)
text_no_numeric = remove_numeric(text_len_more_than1,printTF=True)
print(text_no_numeric)
```

```
I ate 1000 apples and a banana. I have python v2.7. It's 2:30 pm. Could you buy me iphone7?

Remove punctuations..
I ate 1000 apples and a banana I have python v27 Its 230 pm Could you buy me iphone7

Remove a single character word..
 ate 1000 apples and banana have python v27 Its 230 pm Could you buy me iphone7

Remove words with numeric values..
    ate        : True
    1000       : False
    apples     : True
    and        : True
    banana     : True
    have       : True
    python     : True
    v27        : False
    Its        : True
    230        : False
    pm         : True
    Could      : True
    you        : True
    buy        : True
    me         : True
    iphone7    : False
 ate apples and banana have python Its pm Could you buy me
```

In [10]:
```
def text_clean(text_original):
    text = remove_punctuation(text_original)
    text = remove_single_character(text)
    text = remove_numeric(text)
    return(text)


for i, caption in enumerate(df_txt.caption.values):
    newcaption = text_clean(caption)
    df_txt["caption"].iloc[i] = newcaption
```

Plotting the top 50 words that appear in the cleaned dataset

The most and least frequently appearing words

The most common words are articles such as "a", or "the", or punctuations.

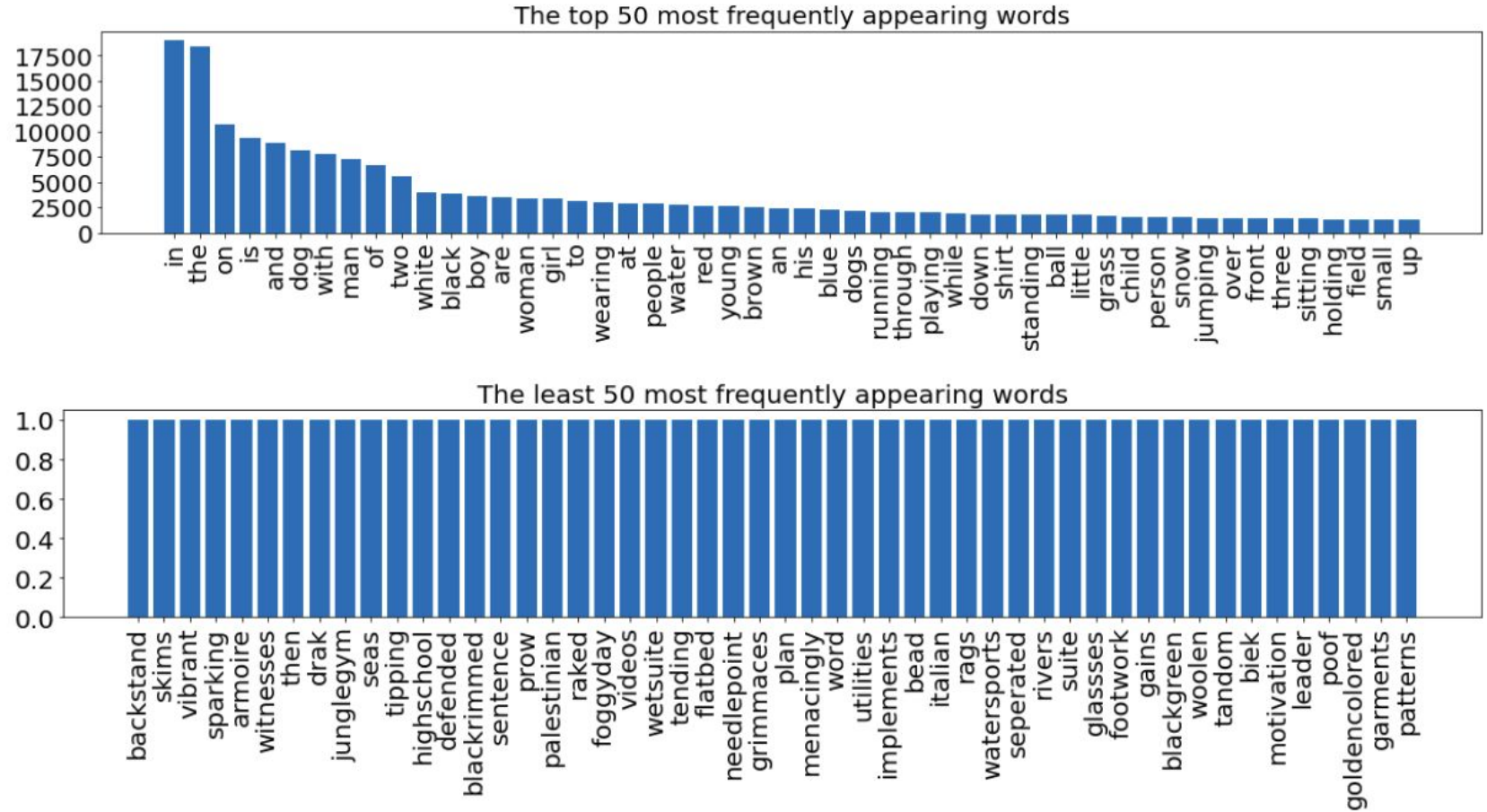
These words do not have much infomation about the data.

In [11]:
```
topn = 50

def plthist(dfsub, title="The top 50 most frequently appearing words"):
    plt.figure(figsize=(20,3))
    plt.bar(dfsub.index,dfsub["count"])
    plt.yticks(fontsize=20)
    plt.xticks(dfsub.index,dfsub["word"],rotation=90,fontsize=20)
    plt.title(title,fontsize=20)
    plt.show()
dfword = df_word(df_txt)
plthist(dfword.iloc[:topn,:],
        title="The top 50 most frequently appearing words")
plthist(dfword.iloc[-topn:,:],
        title="The least 50 most frequently appearing words")
```

```
Vocabulary Size: 8763
```

```
Vocabulary Size: 8763
```



Adding start and end sequence tokens for each captions

Start and end sequence has to be added to the tokens so that it is easier to identify the captions for the images as each of them are of different length

In [12]:
```
from copy import copy
def add_start_end_seq_token(captions):
    caps = []
    for txt in captions:
        txt = 'startseq ' + txt + ' endseq'
        caps.append(txt)
    return(caps)
df_txt0 = copy(df_txt)
df_txt0["caption"] = add_start_end_seq_token(df_txt["caption"])
df_txt0.head(5)
del df_txt
```

In [13]:
```
df_txt0[:5]
```

Out[13]:

	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	startseq child in pink dress is climbing up s...
1	1000268201_693b08cb0e.jpg	1	startseq girl going into wooden building endseq
2	1000268201_693b08cb0e.jpg	2	startseq little girl climbing into wooden pla...
3	1000268201_693b08cb0e.jpg	3	startseq little girl climbing the stairs to h...
4	1000268201_693b08cb0e.jpg	4	startseq little girl in pink dress going into...

Loading VGG16 model and weights to extract features from the images

The pre-trained weights for the VGG-16 model can be downloaded from here.

Image preparation

Create features for each image using VGG16's pre-trained networks Read in the pre-trained network. This network takes input of size (224,224,3). The output layer contains 1,000 nodes.

In [14]:
```
from keras.applications import VGG16
modelvgg = VGG16(include_top=True,weights=None)
## load the locally saved weights
modelvgg.load_weights("Flickr_Data/Flickr_Data/vgg16_weights_tf_dim_ordering_tf_kernels.h5")
modelvgg.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312
predictions (Dense)	(None, 1000)	4097000

Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0

Deleting the last layer of the model

The last layer of the VGG-16 is excluded here because we are are just using it for extracting the features rather than using for object classification.

In [15]:
```
from keras import models
modelvgg.layers.pop()
modelvgg = models.Model(inputs=modelvgg.inputs, outputs=modelvgg.layers[-1].output)
## show the deep learning model
modelvgg.summary()
```

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1180160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2359808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2359808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2359808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2359808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102764544
fc2 (Dense)	(None, 4096)	16781312

Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0

Feature extraction

Here the features are extracted from all the images in the dataset. VGG-16 model gives out 4096 features from the input image of size 224 * 224

In [16]:
```
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input
from collections import OrderedDict

images = OrderedDict()
npix = 224 #image size is fixed at 224 because VGG16 model has been pre-trained to take that size.
target_size = (npix,npix,3)
data = np.zeros((len(jpgs),npix,npix,3))
for i,name in enumerate(jpgs):
    # load an image from file
    filename = dir_Flickr_jpg + '/' + name
    image = load_img(filename, target_size=target_size)
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    nimage = preprocess_input(image)

    y_pred = modelvgg.predict(nimage.reshape( (1,) + nimage.shape[:3]))
    images[name] = y_pred.flatten()
```

Plotting similar images from the dataset

For this we have to first create a cluster and find which images belong together. Hence PCA is used to reduce the dimensions of the features which we got from VGG-16 festure extraction from **4096** to **2**

First the clusters are plotted and few examples are taken from the bunch for displaying

In [17]:
```
from sklearn.decomposition import PCA

encoder = np.array(list(images.values()))
#print(encoder)
pca = PCA(n_components=2)
#print(pca)
y_pca = pca.fit_transform(encoder)
```

Do photo features make sense?

we manually selected similar images that are creating clusters. Namely We created red, green, magenta, blue, yellow and purple clusters. From each cluster, We plotted the original images. As you see, images from the same clusters tend to be very similar:

red: many people are in one image green: dogs on green, yard or on bed magenta: dogs in snow or with water splash blue: guys doing sports with helmets yellow: not sure what these are?? We see many pictures are densely clustered around this area and Photo features seem to make sense!

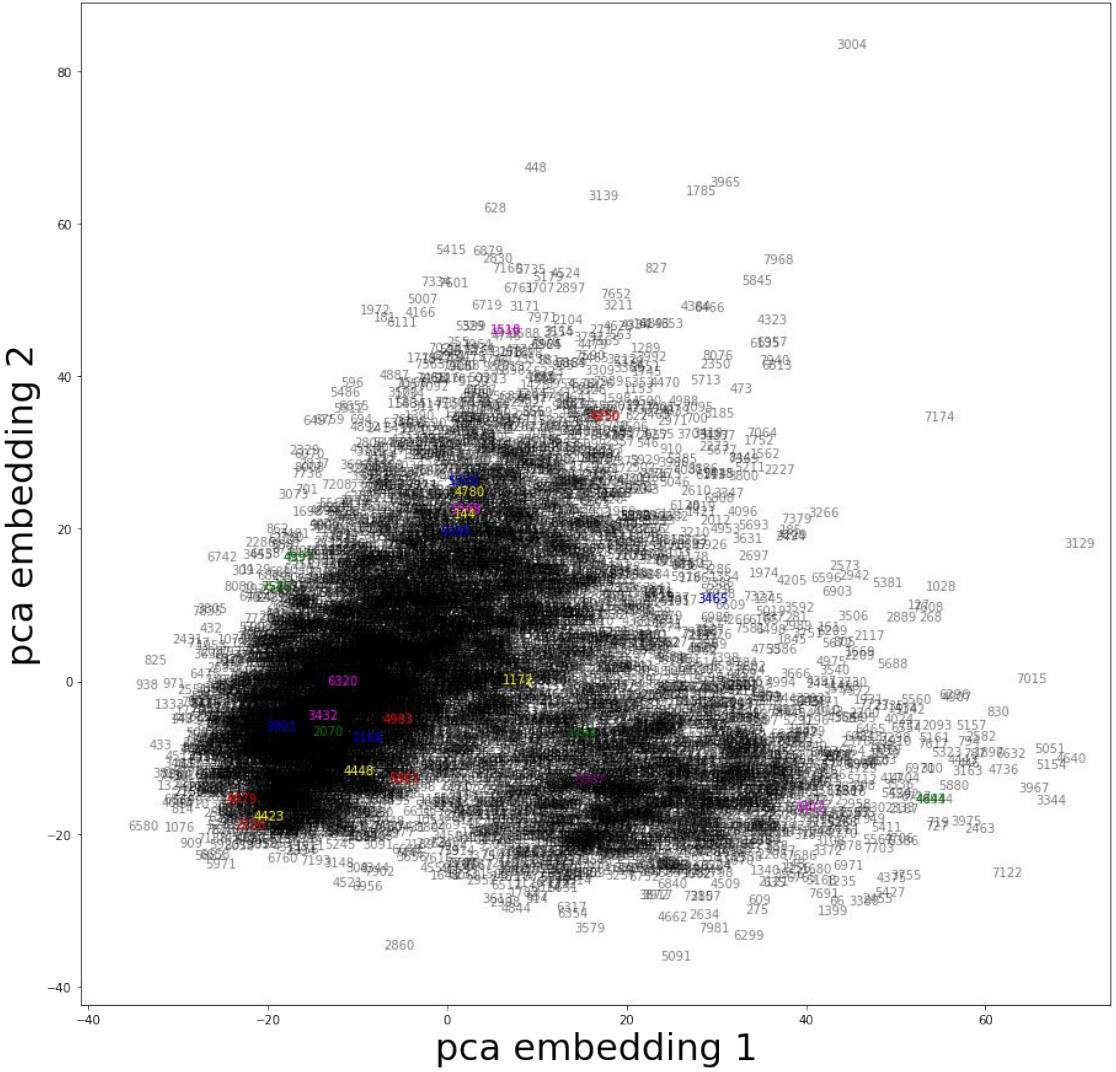
In [18]:
```
## some selected pictures that are creating clusters
#these are just to display the related images from the dataset
picked_pic = OrderedDict()
picked_pic["red"]     = [2720,4250,4983,5862,4079]
picked_pic["green"]   = [2070,3784,7545,4644, 4997]
picked_pic["magenta"] = [6320,3432,1348,7472, 1518]
picked_pic["blue"]    = [3901,2168,3465,5285,5328]
picked_pic["yellow"]  = [144,1172,4423,4780,4448]
picked_pic["purple"]  = [5087]

fig, ax = plt.subplots(figsize=(15,15))
ax.scatter(y_pca[:,0],y_pca[:,1],c="white")

for irow in range(y_pca.shape[0]):
    ax.annotate(irow,y_pca[irow,:],color="black",alpha=0.5) #annotate() is used to place text at the location of th
for color, irows in picked_pic.items():
    for irow in irows:
        ax.annotate(irow,y_pca[irow,:],color=color)
ax.set_xlabel("pca embedding 1",fontsize=30)
ax.set_ylabel("pca embedding 2",fontsize=30)
plt.show()


## plot of images
fig = plt.figure(figsize=(16,20))
count = 1
for color, irows in picked_pic.items():
    for ivec in irows:
        name = jpgs[ivec]
        filename = dir_Flickr_jpg + '/' + name
        image = load_img(filename, target_size=target_size)

        ax = fig.add_subplot(len(picked_pic),5,count,
                             xticks=[],yticks=[])
        count += 1
        plt.imshow(image)
        plt.title("{} ({})".format(ivec,color))
plt.show()
```


pca embedding 2
pca embedding 1
80
60
40
20
0
−20
−40
−40
−20
0
20
40
60

1502 (red)

5298 (red)

2070 (red)

7545 (red)

1965 (red)

746 (green)

7627 (green)

6640 (green)

2733 (green)

4997 (green)

5314 (magenta)

5879 (magenta)

310 (magenta)

5303 (magenta)

3784 (magenta)

4644 (blue)

4209 (blue)

7326 (blue)

7290 (blue)

4394 (blue)

5895 (yellow)

9 (yellow)

27 (yellow)

62 (yellow)

123 (yellow)

5087 (purple)

Merging the images and the captions for training

In [19]:
```
dimages, keepindex = [],[]
# Creating a datframe where only first caption is taken for processing
df_txt0 = df_txt0.loc[df_txt0["index"].values == "0",: ]
for i, fnm in enumerate(df_txt0.filename):
    if fnm in images.keys():
        dimages.append(images[fnm])
        keepindex.append(i)

#fnames are the names of the image files
fnames = df_txt0["filename"].iloc[keepindex].values
#dcaptions are the captions of the images
dcaptions = df_txt0["caption"].iloc[keepindex].values
#dimages are the actual features of the images
dimages = np.array(dimages)
```

In [20]:
```
df_txt0[:5]
```

Out[20]:

	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	startseq child in pink dress is climbing up s...
5	1001773457_577c3a7d70.jpg	0	startseq black dog and spotted dog are fighti...
10	1002674143_1b742ab4b8.jpg	0	startseq little girl covered in paint sits in...
15	1003163366_44323f5815.jpg	0	startseq man lays on bench while his dog sits...
20	1007129816_e794419615.jpg	0	startseq man in an orange hat starring at som...

Tokenizing the captions for further processing

As the model can't take texts as an input, they need to converted into vectors.

In [21]:
```
from keras.preprocessing.text import Tokenizer
## the maximum number of words in dictionary
nb_words = 6000
tokenizer = Tokenizer(nb_words=nb_words)
tokenizer.fit_on_texts(dcaptions)
vocab_size = len(tokenizer.word_index) + 1
print("vocabulary size : {}".format(vocab_size))
dtexts = tokenizer.texts_to_sequences(dcaptions)
print(dtexts[:5])
```

```
vocabulary size : 4476
[[1, 38, 3, 66, 144, 7, 124, 52, 406, 9, 367, 3, 24, 2351, 522, 2], [1, 12, 8, 5, 752, 8, 17, 368, 2], [1, 48, 15,
170, 3, 584, 101, 3, 41, 9, 551, 1198, 11, 55, 213, 3, 1076, 2], [1, 10, 621, 6, 150, 27, 23, 8, 101, 46, 112, 2],
[1, 10, 3, 24, 82, 96, 1199, 19, 162, 2]]
```

Splitting the training and test data

In [22]:
```
prop_test, prop_val = 0.2, 0.2

N = len(dtexts)
Ntest, Nval = int(N*prop_test), int(N*prop_val)

def split_test_val_train(dtexts,Ntest,Nval):
    return(dtexts[:Ntest],
           dtexts[Ntest:Ntest+Nval],
           dtexts[Ntest+Nval:])

dt_test,  dt_val, dt_train   = split_test_val_train(dtexts,Ntest,Nval)
di_test,  di_val, di_train   = split_test_val_train(dimages,Ntest,Nval)
fnm_test,fnm_val, fnm_train  = split_test_val_train(fnames,Ntest,Nval)
```

Finding the max length of the caption

In [23]:
```
maxlen = np.max([len(text) for text in dtexts])
print(maxlen)
```

```
30
```

Processing the captions and images as per the required shape by the model

In [24]:
```
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

def preprocessing(dtexts,dimages):
    N = len(dtexts)
    print("# captions/images = {}".format(N))

    assert(N==len(dimages)) # using assert to make sure that length of images and captions are always similar
    Xtext, Ximage, ytext = [],[],[]
    for text,image in zip(dtexts,dimages):
        # zip() is used to create a tuple of iteratable items
        for i in range(1,len(text)):
            in_text, out_text = text[:i], text[i]
            in_text = pad_sequences([in_text],maxlen=maxlen).flatten()# using pad sequence to make the length of al
            out_text = to_categorical(out_text,num_classes = vocab_size) # using to_categorical to


            Xtext.append(in_text)
            Ximage.append(image)
            ytext.append(out_text)

    Xtext  = np.array(Xtext)
    Ximage = np.array(Ximage)
    ytext  = np.array(ytext)
    print(" {} {} {}".format(Xtext.shape,Ximage.shape,ytext.shape))
    return(Xtext,Ximage,ytext)


Xtext_train, Ximage_train, ytext_train = preprocessing(dt_train,di_train)
Xtext_val,   Ximage_val,   ytext_val   = preprocessing(dt_val,di_val)
# pre-processing is not necessary for testing data
#Xtext_test,  Ximage_test,  ytext_test  = preprocessing(dt_test,di_test)
```

```
# captions/images = 4855
 (49631, 30) (49631, 4096) (49631, 4476)
# captions/images = 1618
 (16353, 30) (16353, 4096) (16353, 4476)
```

Building the LSTM model

This model takes two inputs:

1. 4096-dimensional image features from pre-trained VGG model
2. tokenized captions up to tth word.

The single output is:

1. tokenized t+1th word of caption

Prediction

Given the caption prediction up to the tth word, the model can predict the t+1st word in the caption, and then the input caption can be augmented with the predicted word to contain the caption up to the t+1th word. The augmented caption up to the t+1st word can, in turn, be used as input to predict the t+2nd word in caption. The process is repeated until the "endseq" is predicted.

In [28]:
```
from keras import layers
from keras.layers import Input, Flatten, Dropout, Activation
from keras.layers.advanced_activations import LeakyReLU, PReLU
print(vocab_size)
## image feature

dim_embedding = 64

input_image = layers.Input(shape=(Ximage_train.shape[1],))
fimage = layers.Dense(256,activation='relu',name="ImageFeature")(input_image)
## sequence model
input_txt = layers.Input(shape=(maxlen,))
ftxt = layers.Embedding(vocab_size,dim_embedding, mask_zero=True)(input_txt)
ftxt = layers.LSTM(256,name="CaptionFeature",return_sequences=True)(ftxt)
#,return_sequences=True
#,activation='relu'
se2 = Dropout(0.04)(ftxt)
ftxt = layers.LSTM(256,name="CaptionFeature2")(se2)
## combined model for decoder
decoder = layers.add([ftxt,fimage])
decoder = layers.Dense(256,activation='relu')(decoder)
output = layers.Dense(vocab_size,activation='softmax')(decoder)
model = models.Model(inputs=[input_image, input_txt],outputs=output)

model.compile(loss='categorical_crossentropy', optimizer='adam')

print(model.summary())
```

4476

Layer (type)	Output Shape	Param #	Connected to
input_3 (InputLayer)	(None, 30)	0	
embedding_1 (Embedding)	(None, 30, 64)	286464	input_3[0][0]
CaptionFeature (LSTM)	(None, 30, 256)	328704	embedding_1[0][0]
dropout_1 (Dropout)	(None, 30, 256)	0	CaptionFeature[0][0]
input_2 (InputLayer)	(None, 4096)	0	
CaptionFeature2 (LSTM)	(None, 256)	525312	dropout_1[0][0]
ImageFeature (Dense)	(None, 256)	1048832	input_2[0][0]
add_1 (Add)	(None, 256)	0	CaptionFeature2[0][0] ImageFeature[0][0]
dense_1 (Dense)	(None, 256)	65792	add_1[0][0]
dense_2 (Dense)	(None, 4476)	1150332	dense_1[0][0]

Total params: 3,405,436
Trainable params: 3,405,436
Non-trainable params: 0

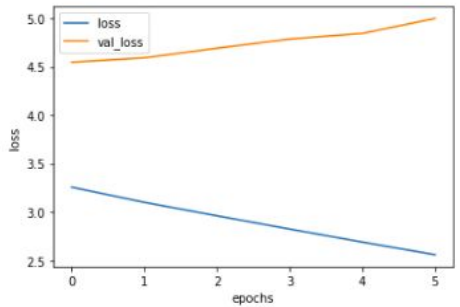
None

Training the LSTM model

In [30]:
```
# fit model
from time import time
from keras.callbacks import TensorBoard
tensorboard = TensorBoard(log_dir="logs/{}".format(time()))
#start = time.time()
hist = model.fit([Ximage_train, Xtext_train], ytext_train,
                 epochs=6, verbose=2,
                 batch_size=32,
                 validation_data=([Ximage_val, Xtext_val], ytext_val),callbacks=[tensorboard])
#end = time.time()
#print("TIME TOOK {:3.2f}MIN".format((end - start )/60))
```

```
Train on 49631 samples, validate on 16353 samples
Epoch 1/6
 - 359s - loss: 3.2557 - val_loss: 4.5460
Epoch 2/6
 - 358s - loss: 3.0992 - val_loss: 4.5912
Epoch 3/6
 - 357s - loss: 2.9603 - val_loss: 4.6902
Epoch 4/6
 - 358s - loss: 2.8232 - val_loss: 4.7849
Epoch 5/6
 - 357s - loss: 2.6889 - val_loss: 4.8433
Epoch 6/6
 - 356s - loss: 2.5588 - val_loss: 4.9975
```

In [31]:
```
for label in ["loss","val_loss"]:
    plt.plot(hist.history[label],label=label)
plt.legend()
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```



Generating captions on a small set of images

After the model finishes training we can test out its performance on the some of the test images to figure out if the generated captions are good enough. If the generated captions are good enough we can generate the captions for the whole test dataset.

In [32]:
```
index_word = dict([(index,word) for word, index in tokenizer.word_index.items()])
def predict_caption(image):
    '''
    image.shape = (1,4462)
    '''

    in_text = 'startseq'

    for iword in range(maxlen):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence],maxlen)
        yhat = model.predict([image,sequence],verbose=0)
        yhat = np.argmax(yhat)
        newword = index_word[yhat]
        in_text += " " + newword
        if newword == "endseq":
            break
    return(in_text)



npic = 5
npix = 224
target_size = (npix,npix,3)

count = 1
fig = plt.figure(figsize=(10,20))
for jpgfnm, image_feature in zip(fnm_test[8:13],di_test[8:13]):
    ## images
    filename = dir_Flickr_jpg + '/' + jpgfnm
    image_load = load_img(filename, target_size=target_size)
    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
    ax.imshow(image_load)
    count += 1

    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    ax = fig.add_subplot(npic,2,count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0,1)
    ax.set_ylim(0,1)
    ax.text(0,0.5,caption,fontsize=20)
    count += 1

plt.show()
```

startseq man in blue shirt is sitting on the street endseq

startseq black and white dog is running through the grass endseq

startseq two children are sitting on the street endseq

startseq black dog is running in the grass endseq

startseq man in white shirt and blue shirt is standing in the camera endseq

Evaluating the model performance

After the model is trained we have to test the models prediction capabilities on test dataset. Traditional accuracy metric can't be used on predictions. For text evaluations we have a metric called as BLEU Score. BLEU stands for Bilingual Evaluation Understudy, it is a score for comparing a candidate text to one or more reference text.

BLEU is a well-acknowledged metric to measure the similarly of one hypothesis sentence to multiple reference sentences. Given a single hypothesis sentence and multiple reference sentences, it returns value between 0 and 1. The metric close to 1 means that the two are very similar. The metric was introduced in 2002 BLEU: a Method for Automatic Evaluation of Machine Translation. Although there are many problems in this metric, for example grammatical correctness are not taken into account, BLEU is very well accepted partly because it is easy to calculate.

In [103]:
```
from nltk.translate.bleu_score import sentence_bleu
reference = [['this', 'is', 'a', 'test'], ['this', 'a','is' 'test']]
candidate = ['this', 'is', 'a', 'test']
score = sentence_bleu(reference, candidate)
print(score)
```

```
1.0
```

In [73]:
```
from nltk.translate.bleu_score import sentence_bleu
reference = [['the', 'quick', 'brown', 'fox','jumped', 'over', 'the', 'lazy', 'dog']]
candidate = ['the', 'quick', 'brown', 'fox', 'jumped', 'over', 'the', 'lazy', 'dog']
score = sentence_bleu(reference, candidate)
print(score)
```

```
1.0
```

Generating captions for the whole test data and finding BLEU score

In [94]:
```
index_word = dict([(index,word) for word, index in tokenizer.word_index.items()])


nkeep = 5
pred_good, pred_bad, bleus = [], [], []
count = 0
for jpgfnm, image_feature, tokenized_text in zip(fnm_test,di_test,dt_test):
    count += 1
    if count % 200 == 0:
        print("{:4.2f}% is done..".format(100*count/float(len(fnm_test))))

    caption_true = [ index_word[i] for i in tokenized_text ]
    caption_true = caption_true[1:-1] ## remove startreg, and endreg
    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    caption = caption.split()
    caption = caption[1:-1]## remove startreg, and endreg

    bleu = sentence_bleu([caption_true],caption)
    bleus.append(bleu)
    if bleu > 0.7 and len(pred_good) < nkeep:
        pred_good.append((bleu,jpgfnm,caption_true,caption))
    elif bleu < 0.3 and len(pred_bad) < nkeep:
        pred_bad.append((bleu,jpgfnm,caption_true,caption))
```

```
12.36% is done..
24.72% is done..
37.08% is done..
49.44% is done..
61.80% is done..
74.17% is done..
86.53% is done..
98.89% is done..
```

Good and bad captions examples from the model

We can check out some of the generated caption's quality. Some times due to the complex nature of the images the generated captions are not acceptable. Below you would find some examples

In [105]:
```
def plot_images(pred_bad):
    def create_str(caption_true):
        strue = ""
        for s in caption_true:
            strue += " " + s
        return(strue)
    npix = 224
    target_size = (npix,npix,3)
    count = 1
    fig = plt.figure(figsize=(10,20))
    npic = len(pred_bad)
    for pb in pred_bad:
        bleu,jpgfnm,caption_true,caption = pb
        ## images
        filename = dir_Flickr_jpg + '/' + jpgfnm
        image_load = load_img(filename, target_size=target_size)
        ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])
        ax.imshow(image_load)
        count += 1

        caption_true = create_str(caption_true)
        caption = create_str(caption)

        ax = fig.add_subplot(npic,2,count)
        plt.axis('off')
        ax.plot()
        ax.set_xlim(0,1)
        ax.set_ylim(0,1)
        ax.text(0,0.7,"true:" + caption_true,fontsize=20)
        ax.text(0,0.4,"pred:" + caption,fontsize=20)
        ax.text(0,0.1,"BLEU: {}".format(bleu),fontsize=20)
        count += 1
    plt.show()

print("Bad Caption")
plot_images(pred_bad)
print("Good Caption")
plot_images(pred_good)
```

true: little girl covered in paint sits in front of painted rainbow with her hands in bowl

pred: two children are sitting on the street

BLEU: 0

true: collage of one person climbing cliff

pred: couple at the background

BLEU: 0

true: couple and an infant being held by the male sitting next to pond with near by stroller

pred: man is sitting on the street

BLEU: 0.121482336484

true: black dog carries green toy in his mouth as he walks through the grass

pred: black dog is shaking the water

BLEU: 0.148231563964

true: black dog and brown dog are jumping up to catch red toy

pred: brown dog is running in the grass

BLEU: 0.228683500859

true: black dog and spotted dog are fighting

pred: black and white dog is running through the grass

BLEU: 0.759835685652

true: black dog leaps over log

pred: black and white dog is running over the grass

BLEU: 0.759835685652

true: man and baby are in yellow kayak on water

pred: man in blue shirt is sitting in the water

BLEU: 0.759835685652

true: white and black dog and brown dog in sandy terrain

pred: black and white dog is running along the beach

BLEU: 0.730633242659

true: child with helmet on his head rides bike

pred: man in red outfit riding his bike on the road

BLEU: 0.740082804492

FUTURE SCOPE

FUTURE SCOPE

This project is having great future scope . The captions generated can be translated into domain language so that people using regional language can also interpret them

We can use other criteria for evaluating the model other than BLEU Score to improve the model.

Larger Datasets could be used to train the model for efficiency like MSCOCO, flickr_30k. This model could be converted to a web interface.

FUTURE SCOPE

Image caption can be applied to image retrieval, video caption, and video movement and the variety of image caption systems available today, experimental results show that this task still has better performance systems and improvement.

CHALLENGES

CHALLENGES

It mainly faces the following three challenges:

- How to generate complete natural language sentences like a human being
- How to make the generated sentence grammatically correct
- How to make the caption semantics as clear as possible and consistent with the given image content.

POSSIBLE IMPROVEMENTS

POSSIBLE IMPROVEMENTS

- An image is often rich in content. The model should be able to generate description sentences corresponding to multiple main objects for images with multiple target objects, instead of just describing a single target object.
- For corpus description languages of different languages, a general image description system capable of handling multiple languages could be developed by machine translation.

POSSIBLE IMPROVEMENTS

- Evaluating the result of natural language generation systems is a difficult problem. The best way to evaluate the quality of automatically generated texts is subjective assessment by linguists, which is hard to achieve. In order to improve system performance, the evaluation indicators should be optimized to make them more in line with human experts' assessments.
- A very real problem is the speed of training, testing, and generating sentences for the model should be optimized to improve performance as the model takes a long time for training, it took almost 3 hrs for flick_8k dataset.

REFERENCES

[1] Sharma.G, Kalena.P, et.al, “Visual Image Caption Generator Using Deep Learning” 2019 ,
2nd International Conference on Advances in Science & Technology (ICAST-2019) K. J. Somaiya Institute of Engineering & Information Technology, University of Mumbai, Maharashtra, India
[2] Hossain.Z, Sohel.F, et.al, “A Comprehensive Survey of Deep Learning for Image Captioning” 2018 ,ACM Computing Surveys, Vol. 0, No. 0, Article 0. Acceptance Date: October 2018
[3] Kunjukuttan.A,et.al The IIT Bombay English-Hindi Parallel Corpus” 2017

[4] Vinyals.O,Toshev.A, et.all, “Show and Tell: A Neural Image Caption Generator” 2015 ,”http://www.mosco.org”
[5] Chen.J, Dong.W , et.al “Image Caption Generator Based On Deep Neural Networks” 2014 ,International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 9 (2018) pp. 7239-7242 © Research India Publications.
http://www.ripublication.com
[6] Josan.G, Lehal.G ,” Direct Approach for Machine Translation from Punjabi to Hindi” 2012,CSI Journal of Computing,Vol 1

THANK YOU!