Image Captioning Bot

SECOND PROGRESS REPORT
OF PROJECT-6th Sem. (CSR-355)

BACHELOR OF ENGINEERING
Computer Science

SUBMITTED BY

AMAN BAHUGUNA **DEEPAK YADAV**

GYAN RANJAN KUMAR

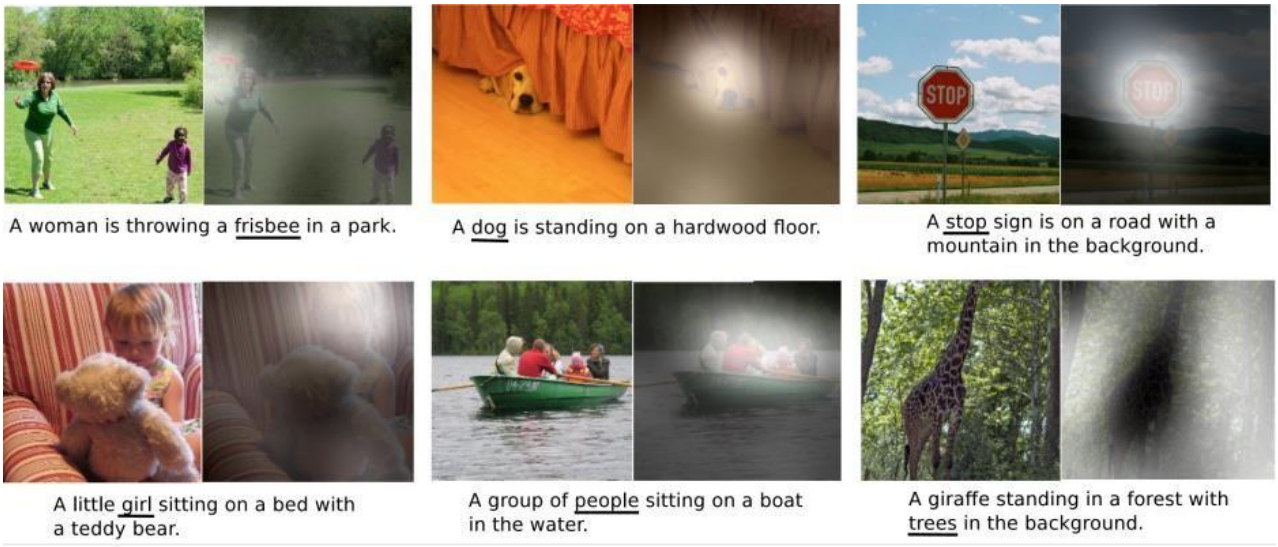
Feb 2021

SUPERVISOR
Er. GAGANJOT KAUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, CHANDIGARH UNIVERSITY, GHARUAN (P.B.) – 140413

1. Introduction

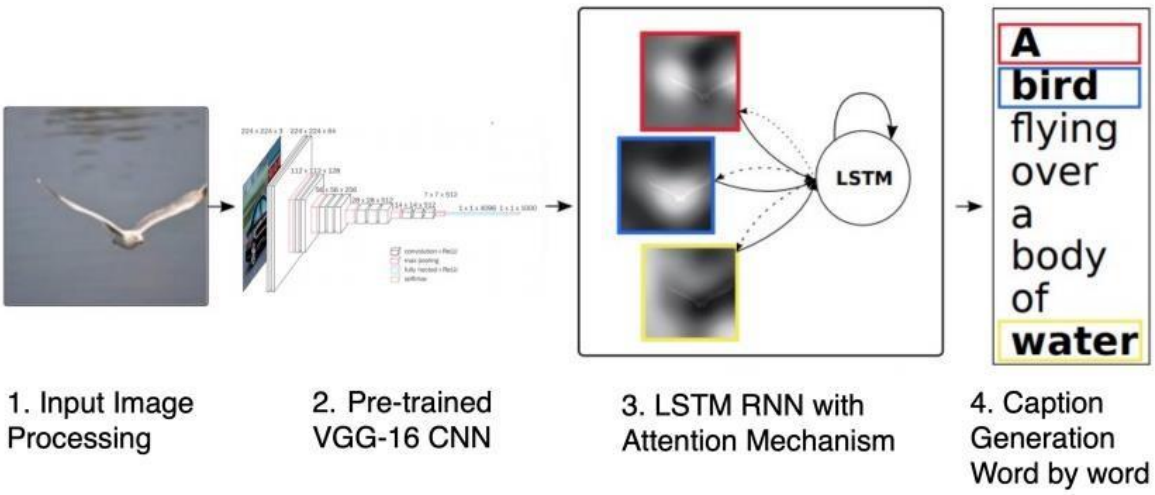
Training computers to be able to automatically generate descriptive captions for images is currently a very hot topic in Computer Vision and Machine Learning. This task is a combination of image scene understanding, feature extraction, and translation of visual representations into natural languages. This project shows some great promises such as building assistive technologies for visually impaired people and help automating caption tasks on the internet. There are a series of relevant research papers attempting to accomplish this task in last decades, but they face various problems such as grammar problems, cognitive absurdity and content irrelevance [2]. However, with the unparalleled advancement in Neural Networks, some groups started exploring Convolutional Neural Network and recurrent neural network to accomplish this task and observed very promising results [1]. The most recent and most popular ones include Show and Tell: A Neural Image Caption Generator [3] and **Show, attend and tell: Neural image caption generator with visual attention [4].** While both papers propose to use a combination of a deep Convolutional Neural Network and a Recurrent Neural Network to achieve this task, the second paper is built upon the first one by adding attention mechanism. As shown in Figure 1, this learnable attention layer allows the network to focus on a specific region of the image for each generated word.



Keywords- Convolutional Neural Network, Recurrent Neural Network, BLEU score

2. Model Architecture

We have written data preprocessing scripts to process raw input data (both images and captions) into proper format; A pre-trained Convolutional Neural Network architecture as an encoder to extract and encode image features into a higher dimensional vector space; An LSTM-based Recurrent Neural Network as a decoder to convert encoded features to natural language descriptions; Attention mechanism which allows the decoder to see features from a specifically



highlighted region of the input image to improve the overall performance; Beam Search to figure out a caption with the highest likelihood. Each individual component of our generator pipeline will be discussed in detail below.

2.1 Data Sources

We have identified the three most commonly used image caption training datasets in Computer Vision research domain - COCO dataset, Flickr8k [5] and Flickr30k. These datasets contain 123,000, 31,000 and 8,000 caption annotated images respectively and each image is labeled with 5 different descriptions. Currently, Flickr8k dataset which contains the least number of images is used as our primary data source over the other two due to our limited storage and computational power. In addition, we also used sanitized Flickr8k data split open-sourced by Andrej Karpathy as a part of our input dataset. This data split has converted original Flickr8k text data to lowercase, discarded non-alphanumerical characters and also split data into train, validation, and test subsets

There are many open-source datasets available for this problem, like Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing180k images), etc.

This dataset contains 8000 images each with 5 captions (as we have already seen in the Introduction section that an image can have multiple captions, all being relevant simultaneously).

These images are bifurcated as follows:

- Training Set — 6000 images
- Dev Set — 1000 images
- Test Set — 1000 images

Understanding the data

If you have downloaded the data from the Kaggle that I have provided, then, along with images, you will also get some text files related to the images. One of the files is "Flickr8k.token.txt" which contains the name of each image along with its 5 captions.

To read the 5 caption for a Sigle Image We have to create the Dictinary to map each Image with List of Caption It has...

In [12]:
```
discription = {}

for x in caption:
    first,second = x.split("\t")
    img_name = first.split(".")[0]
    # if image id is already exist or not
    if discription.get(img_name) is None:
        discription[img_name] = []
    discription[img_name].append(second)
```

In [13]:
```
## We can read this file as follows :
discription['1000268201_693b08cb0e']
```

Out[13]:
```
['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 .']
```

2.2 Data Cleaning

When we deal with text, we generally perform some basic cleaning like lower-casing all the words (otherwise "hello" and "Hello" will be regarded as two separate words), removing special tokens (like '%', '$', '#', etc.), eliminating words which contain numbers (like 'hey199', etc.).
The below code does these basic cleaning steps:

Data Cleaning

To clean the sentences We usualy perform some NLTK library function like - Stopeword removel, Stemming etc. same as we perform in Sentiment analysis. But here we can not perform these operation here. because we have to manage grammer meaning of the sentences for Eg. if we perform stemming on example "Running" it will make Run which will come out different meaning for the caption.

Here When we deal with text, we generally perform some basic cleaning like lowercasing all the words (otherwise "hello" and "Hello" will be regarded as two separate words), removing special tokens (like '%', '$', '#', etc.), eliminating words which contain numbers (like 'hey199', etc.).

In [18]:
```
""" 1. Lower each word
    2. remove puntuations
    3. remove words less than length 1 """

def clean_text(sample):
    sample = sample.lower()

    sample = re.sub("[^a-z]+"," ",sample)

    sample = sample.split()

    sample = [s for s in sample if len(s)>1]

    sample = " ".join(sample)

    return sample
```

In [21]:
```
## Check Function for sample Eg.
clean_text("A cat is sitting over the house #64")
```

Out[21]:
```
'cat is sitting over the house'
```

In [22]:
```
## Clean All the Caption
```

In [23]:
```
#  modify all the captions i.e - cleaned captions

for key, desc_list in descriptions.items():
    for i in range(len(desc_list)):
        desc_list[i] = clean_text(desc_list[i])
```

In [24]:
```
descriptions['1000268201_693b08cb0e'] # Check for an image
```

Out[24]:
```
['child in pink dress is climbing up set of stairs in an entry way',
 'girl going into wooden building',
 'little girl climbing into wooden playhouse',
 'little girl climbing the stairs to her playhouse',
 'little girl in pink dress going into wooden cabin']
```

2.3 Vocabulary Operation:

Create a vocabulary of all the unique words present across all the 8000*5 (i.e. 40000) image captions (**corpus**) in the data set : This means we have 8763 unique words across all the 40000 image captions. We write all these captions along with their image names in a new file namely, "*descriptions.txt"* and save it on the disk.

However, if we think about it, many of these words will occur very few times, say 1, 2 or 3 times. Since we are creating a predictive model, we would not like to have all the words present in our vocabulary but the words which are more likely to occur or which are common. This helps the model become more **robust to outliers** and make less mistakes.

Create a vocabulary

Create a vocabulary of all the unique words present across all the 8000*5 (i.e. 40000) image captions (corpus) in the data set However, if we think about it, many of these words will occur very few times, say 1, 2 or 3 times. Since we are creating a predictive model, we would not like to have all the words present in our vocabulary but the words which are more likely to occur or which are common. This helps the model become more robust to outliers and make less mistakes.

In [27]:
```
# finding the unique vocabulary

vocabulary = set()

for key in descriptions.keys():
    [vocabulary.update(i.split()) for i in descriptions[key]]

print('Vocabulary Size: %d' % len(vocabulary))
```

```
Vocabulary Size: 8424
```

In [28]:
```
#  All words in description dictionary
all_vocab =  []

for key in descriptions.keys():
    [all_vocab.append(i) for des in descriptions[key] for i in des.split()]

print('Vocabulary Size: %d' % len(all_vocab))
print(all_vocab[:15])
```

```
Vocabulary Size: 373837
['child', 'in', 'pink', 'dress', 'is', 'climbing', 'up', 'set', 'of', 'stairs', 'in', 'an', 'entry', 'way', 'girl']
```

In [29]:
```
# Hence we consider only those words which occur at least 10 times in the entire corpus. The code for this is below
# Filter Words from the Vocab according to Certain treshhold frequency
```

Hence we consider only those words which **occur at least 10 times** in the entire corpus. The code for this is below:

3 Loading the training set

The text file “Flickr_8k.trainImages.txt” contains the names of the images that belong to the training set. So we load these names into a list “train”. Thus we have separated the 6000 training images in the list named “train”. Now, we load the descriptions of these images from “descriptions.txt” (saved on the hard disk) in the Python dictionary “train_descriptions”. However, when we load them, we will add two tokens in every caption as follows (significance explained later):

‘**startseq**’ -> This is a start sequence token which will be added at the start of every caption.
‘**endseq**’ -> This is an end sequence token which will be added at the end of every caption.

Prepare Train/Test Data

Read Files

In [34]:
```
# TrainImagesFile
f = open("D:\\Desktop\\Flickr_Data\\Flickr_TextData\\Flickr_8k.trainImages.txt")
train = f.read()
f.close()
```

In [35]:
```
train  = [e.split(".")[0] for e in train.split("\n")[:-1]]
```

In [36]:
```
# TestImagesFile
f = open("D:\\Desktop\\Flickr_Data\\Flickr_TextData\\Flickr_8k.testImages.txt")
test = f.read()
f.close()
```

In [37]:
```
test  = [e.split(".")[0] for e in test.split("\n")[:-1]]
```

In [38]:
```
train[:5]
```

Out[38]:
```
['2513260012_03d33305cf',
 '2903617548_d3e38d7f88',
 '3338291921_fe7ae0c8f8',
 '488416045_1c6d903fe0',
 '2644326817_8f45080b87']
```

In [39]:
```
# create train_descriptions dictionary, which will be similar to earlier one, but having only train samples
# add startseq + endseq

train_descriptions = {}

for t in train:
    train_descriptions[t] = []
    for cap in descriptions[t]:
        cap_to_append = "startseq " + cap + " endseq"
        train_descriptions[t].append(cap_to_append)
```

In [40]:
```
train_descriptions['1000268201_693b08cb0e']
```

Out[40]:
```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']
```

4 Image Preprocessing

Transfer Learning

- Image --> Feature
- Text -->Feture

Step - 1 Image Feature Extraction

Images are nothing but input (X) to our model. As you may already know that any input to a model must be given in the form of a vector. We need to convert every image into a fixed sized vector which can then be fed as input to the neural network. For this purpose, we opt for transfer learning by using the ResNet50 model (Convolutional Neural Network). ResNet50 is not the first model coming from the ResNet family. The original model was called the Residual net or ResNet and was another milestone in the CV domain back in 2015. The main motivation behind this model was to avoid poor accuracy as the model went on to become deeper. Additionally, if you are familiar with Gradient Descent, you would have come across the Vanishing Gradient issue – the ResNet model aimed to tackle this issue as well. ResNet34(ResNet50 also follows a similar technique with just more layers)

The code for this is as follows:

In [41]:
```
IMG_PATH = "D:\\Desktop\\Flickr_Data\\Images"
```

In [42]:
```
model = ResNet50(weights="imagenet", input_shape=(224,224,3))
model.summary()
```

```
Model: "resnet50"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 224, 224, 3) 0
__________________________________________________________________________________________________
conv1_pad (ZeroPadding2D)       (None, 230, 230, 3)  0           input_1[0][0]
__________________________________________________________________________________________________
conv1_conv (Conv2D)             (None, 112, 112, 64) 9472        conv1_pad[0][0]
__________________________________________________________________________________________________
conv1_bn (BatchNormalization)   (None, 112, 112, 64) 256         conv1_conv[0][0]
__________________________________________________________________________________________________
conv1_relu (Activation)         (None, 112, 112, 64) 0           conv1_bn[0][0]
__________________________________________________________________________________________________
pool1_pad (ZeroPadding2D)       (None, 114, 114, 64) 0           conv1_relu[0][0]
__________________________________________________________________________________________________
pool1_pool (MaxPooling2D)       (None, 56, 56, 64)   0           pool1_pad[0][0]
__________________________________________________________________________________________________
conv2_block1_1_conv (Conv2D)    (None, 56, 56, 64)   4160        pool1_pool[0][0]
__________________________________________________________________________________________________
conv2_block1_1_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_1_conv[0][0]
__________________________________________________________________________________________________
conv2_block1_1_relu (Activation (None, 56, 56, 64)   0           conv2_block1_1_bn[0][0]
__________________________________________________________________________________________________
conv2_block1_2_conv (Conv2D)    (None, 56, 56, 64)   36928       conv2_block1_1_relu[0][0]
__________________________________________________________________________________________________
conv2_block1_2_bn (BatchNormali (None, 56, 56, 64)   256         conv2_block1_2_conv[0][0]
```

4.1 Data Preprocessing For Image Encoding

We must note that captions are something that we want to predict. So during the training period, captions will be the target variables (Y) that the model is learning to predict.
But the prediction of the entire caption, given the image does not happen at once. We will predict the caption **word by word**. Thus, we need to encode each word into a fixed sized vector. However, this part will be seen later when we look at the model design, but for now we will create two Python Dictionaries namely "wordtoix" (pronounced — word to index) and "ixtoword" (pronounced — index to word).

Data Preproccesing for Images/Image Encoding :

It is basically when we feed the image into the madel it gives an enconding as output each corresponding image. Preprocessing --> paass through ResNet Model ---> Output(encoding)

In [46]:
```
def encode_image(img):
    img = preprocess_image(img)
    feature_vector = model_new.predict(img)
    feature_vector = feature_vector.reshape(feature_vector.shape[1],)
    return feature_vector
```

In [47]:
```
encode_image("D:\\Desktop\\Flickr_Data\\Images\\1000268201_693b08cb0e.jpg")
```

Out[47]:
```
array([0.06535921, 0.16782534, 0.3251761 , ..., 0.05107138, 0.32821208,
       1.0043364 ], dtype=float32)
```

In [48]:
```
start = time()

from tqdm.notebook import tqdm

encoding_train = {}

for ix, img_id in tqdm(enumerate(train),total=len(train)):

    img = IMG_PATH+"/"+img_id+".jpg".format(train[ix])
    # encoding_train[img[len(images):]] = encode_image(img)
    encoding_train[img_id] = encode_image(img)
    if ix%100==0:
        print("Encoding image- "+ str(ix))

print("Time taken in seconds =", time()-start)
```

Data pre-processing for Captions

In [54]:
```
"""
word_to_idx is mapping between each unique word in all_vocab to int value
and idx_to_word is vice-versa
"""


ix = 1
word_to_idx = {}
idx_to_word = {}

for e in all_vocab:
    word_to_idx[e] = ix
    idx_to_word[ix] = e
    ix +=1
```

In [55]:
```
word_to_idx["dog"]
```

Out[55]:
```
6
```

In [56]:
```
idx_to_word[6]
```

Out[56]:
```
'dog'
```

In [57]:
```
print(len(idx_to_word))
```

```
1845
```

4.2 Data Preparation using Generator Function

This is one of the most important steps in this case study. Here we will understand how to prepare the data in a manner which will be convenient to be given as input to the deep learning model.Hereafter, I will try to explain the remaining steps by taking a sample example as follows:

Data Loader (Generator)

In [61]:
```
def data_generator(train_descriptions, encoding_train, word_to_idx, max_len, num_photos_per_batch):

    X1, X2, y = [], [], []

    n=0

    while True:

        for key, desc_list in train_descriptions.items():
            n +=1

            photo = encoding_train[key]

            for desc in desc_list:

                seq = [ word_to_idx[word] for word in desc.split() if word in word_to_idx]

                for i in range(1,len(seq)):

                    in_seq = seq[0:i]
                    out_seq = seq[i]

                    in_seq = pad_sequences([in_seq], maxlen=max_len, value=0, padding='post')[0]

                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                    X1.append(photo)
                    X2.append(in_seq)
                    y.append(out_seq)

            if n==num_photos_per_batch:
                yield [np.array(X1), np.array(X2)], np.array(y)
                X1, X2, y = [], [], []
                n=0
```

4.3 Word Embeddings

As already stated above, we will map the every word (index) to a 200-long vector and for this purpose, we will use a pre-trained GLOVE Model:

Word Embedding

In [62]:
```
!wget http://nlp.stanford.edu/data/wordvecs/glove.6B.zip
!unzip glove.6B.zip
```

```
'wget' is not recognized as an internal or external command,
operable program or batch file.
'unzip' is not recognized as an internal or external command,
operable program or batch file.
```

In [63]:
```
f = open("D:\\Desktop\\datasets\\glove.6B.50d.txt", encoding = 'utf8')
```

In [64]:
```
embedding_index = {}

for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype="float")

    embedding_index[word] = coefs

f.close()
```

In [65]:
```
f.close()
```

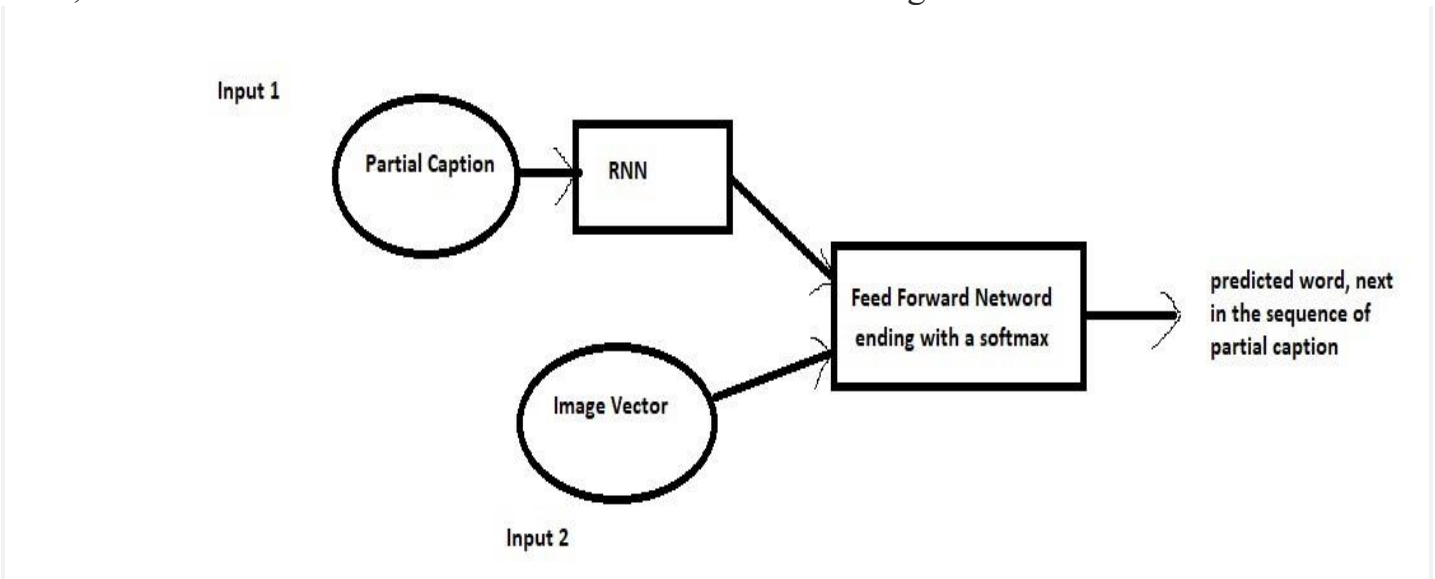
In [66]:
```
embedding_index['apple']
```

Out[66]:
```
array([ 0.52042 , -0.8314  ,  0.49961 ,  1.2893  ,  0.1151  ,  0.057521,
       -1.3753  , -0.97313 ,  0.18346 ,  0.47672 , -0.15112 ,  0.35532 ,
        0.25912 , -0.77857 ,  0.52181 ,  0.47695 , -1.4251  ,  0.858   ,
        0.59821 , -1.0903  ,  0.33574 , -0.60891 ,  0.41742 ,  0.21569 ,
       -0.07417 , -0.5822  , -0.4502  ,  0.17253 ,  0.16448 , -0.38413 ,
        2.3283  , -0.66682 , -0.58181 ,  0.74389 ,  0.095015, -0.47865 ,
       -0.84591 ,  0.38704 ,  0.23693 , -1.5523  ,  0.64802 , -0.16521 ,
```

5 Model Architecture

Since the input consists of two parts, an image vector and a partial caption, we cannot use the Sequential API provided by the Keras library. For this reason, we use the Functional API which allows us to create Merge Models.

First, let's look at the brief architecture which contains the high level sub-modules:



Model Architecture

In [70]:
```
# image feature extractor model

input_img_fea = Input(shape=(2048,))
inp_img1 = Dropout(0.3)(input_img_fea)
inp_img2 = Dense(256, activation='relu')(inp_img1)
```

In [71]:
```
# partial caption sequence model

input_cap = Input(shape=(max_len,))
inp_cap1 = Embedding(input_dim=vocab_size, output_dim=50, mask_zero=True)(input_cap)
inp_cap2 = Dropout(0.3)(inp_cap1)
inp_cap3 = LSTM(256)(inp_cap2)
```

In [72]:
```
decoder1 = add([inp_img2 , inp_cap3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

# Merge 2 networks
model = Model(inputs=[input_img_fea, input_cap], outputs=outputs)
```

In [73]:
```
model.summary() # Basicallly Show How my Lyers are connected
```

```
Model: "model_1"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 35)]         0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 35, 50)       92400       input_3[0][0]
```

6 Training of Model

Training of Model

In [76]:
```
epochs = 15
number_pics_per_batch = 3
steps = len(train_descriptions)//number_pics_per_batch
```

In [77]:
```
for i in range(epochs):
    generator = data_generator(train_descriptions, encoding_train, word_to_idx, max_len, number_pics_per_batch)
    model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save('./model_weights/model_' + str(i) + '.h5')
```

```
C:\Users\IRON MAN\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py:1844: UserWarning: `Model.fit_generator` is depr
ecated and will be removed in a future version. Please use `Model.fit`, which supports generators.
  warnings.warn('`Model.fit_generator` is deprecated and '
2000/2000 [==============================] - 773s 383ms/step - loss: 4.7617
2000/2000 [==============================] - 747s 374ms/step - loss: 3.6049
2000/2000 [==============================] - 851s 425ms/step - loss: 3.3489
2000/2000 [==============================] - 861s 431ms/step - loss: 3.1931
2000/2000 [==============================] - 757s 378ms/step - loss: 3.0793
2000/2000 [==============================] - 787s 393ms/step - loss: 2.9972
2000/2000 [==============================] - 784s 392ms/step - loss: 2.9316
2000/2000 [==============================] - 698s 349ms/step - loss: 2.8778
2000/2000 [==============================] - 752s 376ms/step - loss: 2.8315
2000/2000 [==============================] - 725s 362ms/step - loss: 2.7912
2000/2000 [==============================] - 691s 346ms/step - loss: 2.7565
2000/2000 [==============================] - 708s 354ms/step - loss: 2.7254
2000/2000 [==============================] - 769s 384ms/step - loss: 2.7007
2000/2000 [==============================] - 780s 390ms/step - loss: 2.6771
2000/2000 [==============================] - 763s 382ms/step - loss: 2.6551
```

7 Predictions

In [78]:
```
def predict_caption(photo):
    in_text = "startseq"

    for i in range(max_len):
        sequence = [word_to_idx[w] for w in in_text.split() if w in word_to_idx]
        sequence = pad_sequences([sequence], maxlen=max_len, padding='post')

        ypred =  model.predict([photo,sequence])
        ypred = ypred.argmax()
        word = idx_to_word[ypred]
        in_text+= ' ' +word

        if word =='endseq':
            break


    final_caption =  in_text.split()
    final_caption = final_caption[1:-1]
    final_caption = ' '.join(final_caption)

    return final_caption
```

In [80]:
```
for i in range(20):
    rn =  np.random.randint(0, 1000)

    img_name = list(encoding_test.keys())[rn]
    photo = encoding_test[img_name].reshape((1,2048))

    i = plt.imread(IMG_PATH +"/" +img_name+".jpg")
    plt.imshow(i)
    plt.axis("off")
    plt.show()
```

8 RESULTS /OUTPUTS

The output of the model is caption to the image genrater text. I'm atteching my whole code with dataset feel free to review it.

GitHub: CODE

man in blue shirt and blue shirt is hitting tennis ball

two children playing soccer on field

crowd of people are waiting in line in front of crowd of people

two dogs are walking through the woods

black and white dog is running on the grass

two girls in orange shirts are standing on the pavement

man in blue shirt is riding on horse

man in red shirt and black shorts is standing on beach

References

[1] Alex Graves. Generating sequences with recurrent neural networks. CoRR, abs/1308.0850, 2013

[2] Polina Kuznetsova, Vicente Ordonez, Alexander C. Berg, Tamara L. Berg, and Yejin Choi. Collective generation of natural image descriptions. pages 359–368, 2012.

[3] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. CoRR, abs/1411.4555, 2014.

[4] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron C. Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. CoRR, abs/1502.03044, 2015

[5] Micah Hodosh, Peter Young, and Julia Hockenmaier. Framing image description as a ranking task: Data, models and evaluation metrics. J. Artif. Int. Res., 47(1):853–899, May 2013.

[6] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. pages 730–734, Nov 2015