“AUTOMATIC IMAGE CAPTIONING USING DEEP LEARNING”

Report submitted in partial fulfilment of the requirement for degree of

Bachelor of Technology
In
Computer Science & Engineering

By
Parul Diwakar
To
Shaily Malik, Assistant Professor, Dept. of CSE

Maharaja Surajmal Institute of Technology
Affiliated to Guru Gobind Singh Indraprastha University
Janakpuri, New Delhi-58

Batch 2018-22

CANDIDATE'S DECLARATION

I, **Parul Diwakar**, Roll No. **07115002718**, B.Tech (Semester 5) of the Maharaja Surajmal Institute of Technology, New Delhi, hereby declare that the Training Report entitled "**AUTOMATIC IMAGE CAPTIONING USING DEEP LEARNING"** , is an original work and the data provided in the study is authentic to the best of my knowledge. This report has not been submitted to any other Institute for the award of any other degree.

Parul Diwakar .
(Roll no. 071150027218)

CERTIFICATE

CODING BLOCKS
Code Your Way To Success

COURSE COMPLETION CERTIFICATE

Date: 6th July, 2020

This is to certify that **Parul Diwakar**, student of **Maharaja Surajmal Institute of Technology** has completed **classroom** course in **Machine Learning** at Coding Blocks. This course was started on **28th Dec, 2019** and lasted for **4 months**.

During this period, the student has covered all the topics, completed all the assignments and created a project as well.

For Coding Blocks Pvt. Ltd.

Priyanshu Agrawal

Business Head

Mail id: hr@codingblocks.com

ACKNOWLEDGEMENT

I would like to acknowledge the contributions of the following people without whose help and guidance this report would not have been completed.

I would like to thank Dr Koyal Datta Gupta (HOD, Dept of CSE at MSIT, New Delhi) to encourage us and providing us with the opportunity to prepare this project. I am immensely obliged to our Project Coordinator, Prof. Shaily Malik for her encouraging guidance and kind supervision in the completion of my project report.

I acknowledge the support of our course mentors at Coding Blocks, **Mr Prateek Narang and Mr Mohit Uniyal,** with respect and gratitude, under whose expertise, guidance, support, encouragement, and enthusiasm has made this report possible. Their feedback vastly improved the quality of this project and provided an enthralling experience. I am indeed proud and fortunate to be supported by them.

Although it is not possible to name everyone individually, I shall ever remain thankful to the faculty members at **Coding Blocks, Pitampura** their persistent support and cooperation extended during this course and being patient during the difficulties caused by this pandemic.

This acknowledgement will remain incomplete if I fail to express our deep sense of gratitude to my parents and God for their consistent blessings and encouragement.

Parul Diwakar .
(Roll no. 071150027)

ABSTRACT

In recent years, with the rapid development of artificial intelligence, image caption has gradually attracted the attention of many researchers in the field of artificial intelligence and has become an interesting task. Automatically generating natural language descriptions according to the content observed in an image, is an important part of scene understanding, which combines the knowledge of computer vision and natural language processing. The application of image captioning is extensive and significant. This project describes the methodology and implementation of Automatic Image Captioning using the concepts of Deep Learning. The implementation is shown in detail with explanation and concluded with stating the improvements that could be made and future scope.

TABLE OF CONTENTS

Chapter 1

Introduction

Objective

Can you caption the image given above?

Some might caption it, "A cat with black and white spots.", some may say, "A cat is walking on grass." and some others might say, "A cat is walking in a forest."

All of these captions are relevant for this image and there may be some others as well. The important point made here is that as human beings, to just have a glance at a picture and describe it in an appropriate language. But can we write a computer program that takes an image as input and produces a relevant caption as output. Just prior to the recent development of Deep Neural Networks this problem was inconceivable even by the most advanced researchers in Computer Vision. But with the advent of Deep Learning this problem can be solved very easily if we have the required dataset. This problem was well researched by **Andrej Karapathy** in his PhD thesis at Stanford, who is also now the **Director of AI at Tesla.**

The purpose of this Report is to explain that how Deep Learning can be used to solve this problem of generating a caption for a given image, hence the name "AUTOMATIC IMAGE CAPTIONING USING DEEP LEARNING".

A similar, more advanced example is state-of-the-art system created by Microsoft called as Caption Bot.

Need

Most importantly we need to understand how important this problem is to real world scenarios. Let us see few applications where a solution to this problem can be very useful.

- Self driving cars — Automatic driving is one of the biggest challenges and if we can properly caption the scene around the car, it can give a boost to the self driving system.

- Aid to the blind — We can create a product for the blind which will guide them travelling on the roads without the support of anyone else. We can do this by first converting the scene into text and then the text to voice. Both are now famous applications of Deep Learning.

- CCTV cameras are everywhere today, but along with viewing the world, if we can also generate relevant captions, then we can raise alarms as soon as there is some malicious activity going on somewhere. This could probably help reduce some crime and/or accidents.

- Automatic Captioning can help, make Google Image Search as good as Google Search, as then every image could be first converted into a caption and then search can be performed based on the caption.

Some of the on going applications of the similar kind are given as follows.

- Google Photos: Classify your photo into Mountains, sea etc.

- Facebook: Using AI to classify, segmenting and finding patterns in pictures.

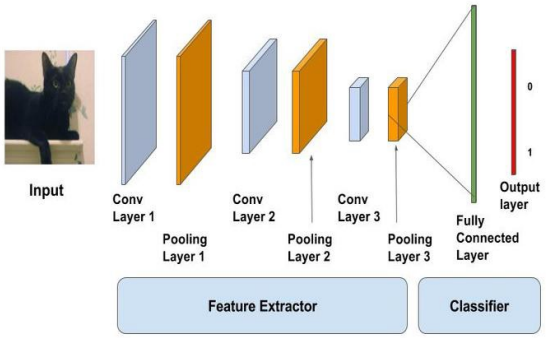
- Fed Ex and other courier services: Are using hand written digit recognition system from may times now to detect pin code correctly.

- Picasa : Using facial Recognition to identify your friends and you in a group picture.

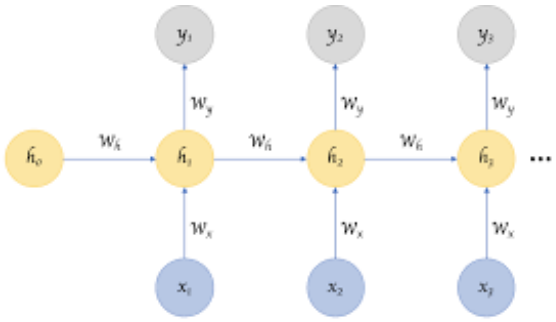
- Tesla/Google Self Drive Cars: All the self drive cars are using image/video processing with neural network to attain their goal.

Methodology

The performed project requires familiarity with basic Deep Learning concepts like Multi-layered Perceptrons, Convolution Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Transfer Learning, Gradient Descent, Back propagation, Overfitting, Probability, Text Processing, Python syntax and data structures, Keras library, etc.



Convolution Neural Networks (CNNs)



Recurrent Neural Networks (RNNs)

There are many open source datasets available for this problem, like Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc.

But for the purpose of this project, I have used the Flickr 8k dataset which can be downloaded on the Kaggle website provided by the University of Illinois at Urbana-Champaign. Also training a model with a large dataset as this one may not be feasible on a system which is not a very high end PC/Laptop and hence batch approach has been adopted and is implemented on Google Colaboratory,

This dataset contains 8000 images each with 5 captions per image. These images are bifurcated as follows:

- Training Set — 6000 images
- Test Set — 2000 images

The training set is fed to the model to train it and finally model is tested on the testing data to generate captions.

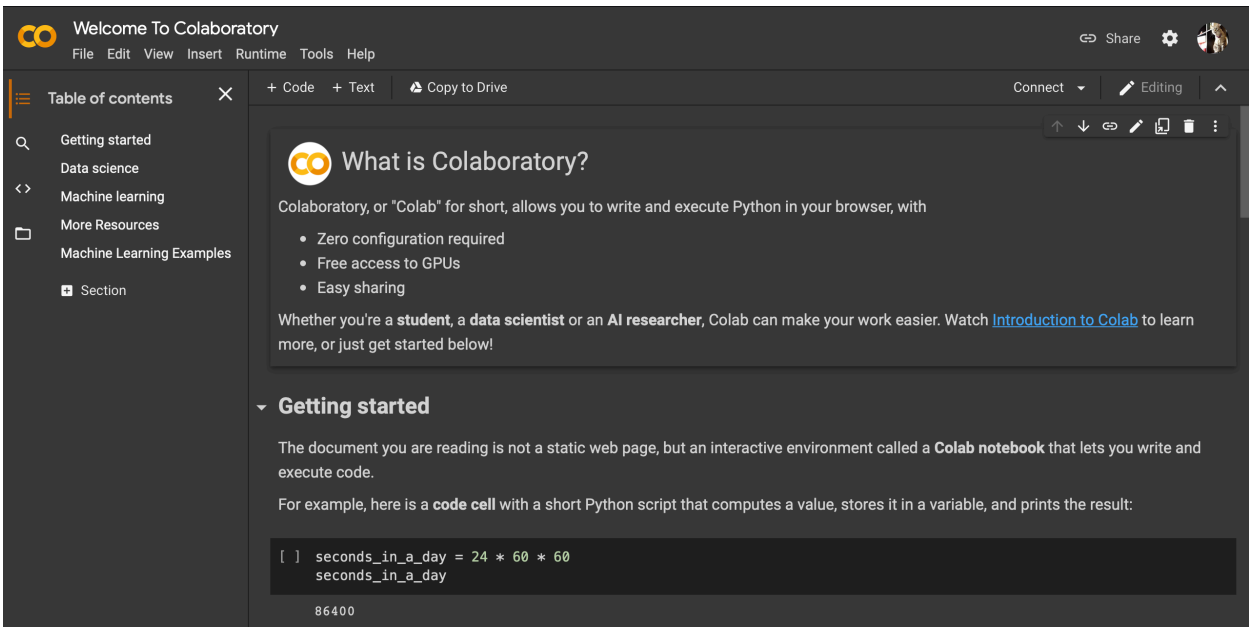
Software Used

Colaboratory is ideal for everything from improving your Python coding skills to working with deep learning libraries, like **PyTorch**, **Keras**, **TensorFlow**, and **OpenCV**. Google has provided a free cloud service based on Jupyter Notebooks that supports free GPU. We can create notebooks in Colaboratory, upload notebooks, store notebooks, share notebooks, mount our Google Drive and use suitable/required data we have stored in our Google Drive, import most of our required directories, upload our personal Jupyter Notebooks, upload notebooks directly from GitHub, upload Kaggle files, download our notebooks.

It supports **Python 2.7** and **3.6**, but not **R** or **Scala** yet. There is a limit to your sessions and size i.e after a certain period of inactivity or restarting the server the files are lost and are needed to be uploaded again.

If we are uploading small files, we can just upload them directly with some simple code. However, if we want to, we can also just go to the left side of the screen and click "**upload files**" to use a local file.

Google Colaboratory is incredibly easy to use on pretty much every level, especially if we are familiar with Jupyter Notebooks. However, for using some large files we can use the link and transfer them to our Colab Notebook or save them in our Google Drive and link it to the Notebook



Google Colaboratory Welcome Page

About Organization

CODING BLOCKS

Address

47, Nishant Kunj, 1st & 2nd Floor, Main Road, opposite Metro Pillar Number 337, Pitam Pura, New Delhi, Delhi 110034

VISION of Coding Blocks

Create More Employable Talent

Coding Blocks was founded in 2014 with a mission to create skilled Software Engineers for our country and the world. We are here to bridge the gap between the quality of skills demanded by industry and the quality of skills imparted by conventional institutes.

Make People Feel Enthusiastic About Coding

Programming is a lot of fun because, unlike other subjects, you get to instantly apply the concepts you are learning. At Coding Blocks, we strive to increase student interest by providing hands on practical training on every concept taught in the classroom.

Unlock New Opportunities

We create confident developers who think beyond industrial jobs and march their ideas into self-created entrepreneurship ventures. Skill and innovative thinking give developers the confidence to transform their ideas into real life products and hopefully go on to build million dollar companies.

Connect Talent with Employers

Along with training students with the latest technologies and programming languages, we also connect them to software companies via our placement assistance program. This program includes practicing a lot of interview problems and mock interviews conducted by company representatives.

Chapter 2
Project Design

This project requires understanding of basic Machine Learning and Deep Learning concepts like Image Processing and manipulation, Multi-layered Perceptrons, Convolution Neural Networks (CNNs), Recurrent Neural Networks (RNNs), Transfer Learning, Gradient Descent, Forward propagation, Back propagation, Overfitting, Applying Probability, Text Processing, Python syntax and data structures, Keras library, etc.

The Project Design followed throughout the process is given below .

1. Data Collection

2. Understanding the Data

3. Data Cleaning

4. Loading the training Set

5. Data Processing - Images (Image —> Vectors)

6. Data Processing - Captions (Words —> Vectors)

7. Word Embeddings

8. Model Architecture

9. Inference

Data Collection

There are many open source datasets available for this problem, like Flickr 8k (containing8k images), Flickr 30k (containing 30k images), MS COCO (containing 180k images), etc.

But for the purpose of this project, I have used the Flickr 8k dataset which can be downloaded on the Kaggle website provided by the University of Illinois at Urbana-Champaign. Also training a model with a large dataset such as this one may not be feasible on a system which is not a very high end PC/Laptop and hence batch approach has been adopted and the code is implemented on Google Colaboratory, a free cloud service provided by Google with free GPU.

The data being huge in size, was uploaded to my Google Drive account and was linked to my Colaboratory Notebook.

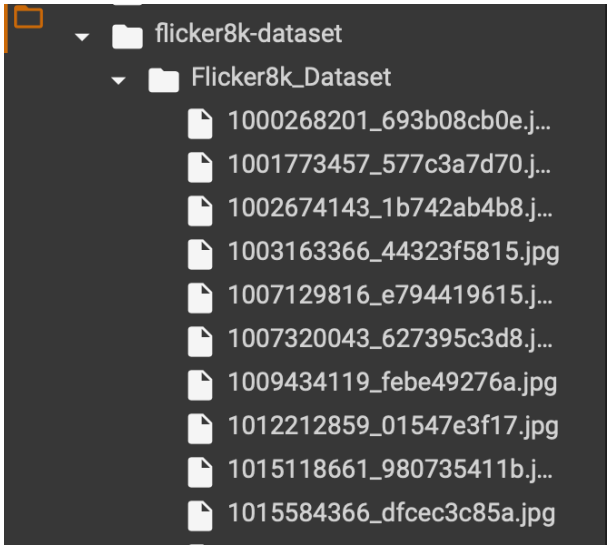
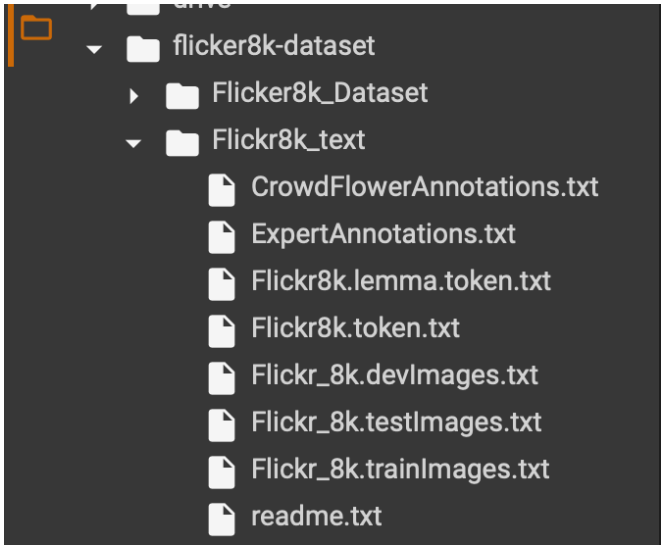


Image Dataset



Text Dataset

Understanding the Data

This dataset contains 8000 images each with all images having 5 captions per image.

These images are bifurcated as follows:

- Training Set — 6000 images
- Test Set — 2000 images

After acquiring the dataset using the link of the Flickr 8k Dataset, from Google Drive, then, along with images, we receive some text files related to the images as well. One of the files is "Flickr8k.token.txt" which contains the name of each image along with its five corresponding captions. Here is a screenshot of how the data is presented in the "Flickr8k.token.txt" file.

```
101654506_8eb26cfb60.jpg#0      A brown and white dog is running through the snow .
101654506_8eb26cfb60.jpg#1      A dog is running in the snow
101654506_8eb26cfb60.jpg#2      A dog running through snow .
101654506_8eb26cfb60.jpg#3      a white and brown dog is running through a snow covered field .
101654506_8eb26cfb60.jpg#4      The white and brown dog is running over the surface of the snow .

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 .
```

Dataset Representation

Thus every line contains the **<image name>#i <caption>**, where 0≤i≤4 i.e. the name of the image, caption number (0 to 4) and the actual caption. Each image is mapped to its five corresponding captions using **Dictionary** data structure. For example

Captions mapped to the image '1119015538_e8e796281e.jpg'

Data Cleaning

When we deal with text that is not preprocessed, for example Movie Reviews Dataset acquired from the resources available on the Internet, we generally perform some basic cleaning like lower-casing all the words (otherwise“Apple” and “apple” will be considered as two separate words), removing special tokens (like ‘%’, ‘$’, ‘#’, etc.), eliminating words which contain numbers (like ‘hey199’, etc.) and white spaces.

Then we code to find out all the 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.

We perform the following operation on the caption:
1. Lower-casing all the words.
2. Removing punctuations.
3. Removing words of length less than 2.
4. Can remove special symbols like @,<,# etc.

descriptions.txt

```
{'1000268201_693b08cb0e': ['child in pink dress is climbing up set of stairs in an entry way',
                           'girl going into wooden building', 'little girl climbing into wooden playho
                           'little girl climbing the stairs to her playhouse',
                           'little girl in pink dress going into wooden cabin'],
 '1001773457_577c3a7d70': ['black dog and spotted dog are fighting',
                           'black dog and tri colored dog playing with each other on the road',
                           'black dog and white dog with brown spots are staring at each other in the
                           'two dogs of different breeds looking at each other on the road',
                           'two dogs on pavement moving toward each other'], '1002674143_1b742ab4b8':
```

descriptions.txt

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.
Hence we consider only those words which **occur at least 10 times** in the entire corpus. We get a **Vocabulary** of 8424 words and **Total Words** 373837

Loading the training Set

The text file “Flickr_8k.trainImages.txt” contains the names of the images that belong to the training set as **train_file_data** and we load these names into a list “train”.

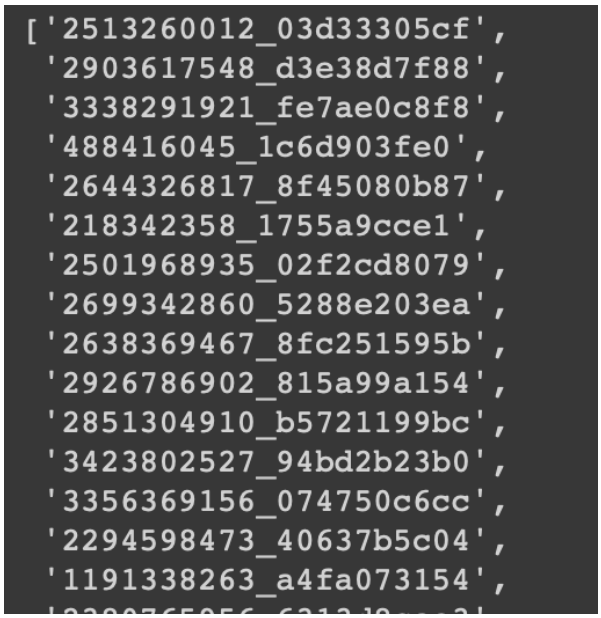
Thus we have separated the 6000 training images in the list named “train”.

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):
‘<s>’ —> This is a start sequence token which will be added at the start of every caption.
‘<e>’ —> This is an end sequence token which will be added at the end of every caption.

Loading the testing Set

The text file “Flickr_8k.testImages.txt” contains the names of the images that belong to the training set as **test_file_data** and we load these names into a list “test”.



Train list with names of all the training set images.

Data Preprocessing - Images

Input to a Machine Learning model is provided in the form of a vector. Vector input format means that we need to represent each object of interest as a list of numbers. Hence all the input dataset is converted into vectors i.e images as well as textual data.

In this case images are treated as nothing but input to our model and fed to the model in the form of vectors.

For making the processing and predicting process easy for the model, 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 InceptionV3 model (Convolutional Neural Network) created by Google Research.

Our purpose ids to get a fixed length informative vector for each image. This process is called automated feature engineering. Automated feature engineering helps by automatically creating many candidate features out of a dataset from which the best features can be selected and use it for training.

Hence, we just remove the last softmax layer from the model and extract a 2048 length vector. We save all the (2048,) vector features in a Python dictionary and save it on the disk using Pickle file, namely "**./encoded_train_images.pkl**" whose keys are image names and values are corresponding 2048 length feature vector.

Similarly we encode all the test images and save them in the file "**encoded_test_images.pkl"**.

```
Encoding image :0
Encoding image :1000
Encoding image :2000
Encoding image :3000
Encoding image :4000
Encoding image :5000
Time taken in sec :177.58313059806824
```

encoding_train

```
Encoding image :0
Encoding image :200
Encoding image :400
Encoding image :600
Encoding image :800
Time taken in sec :29.186405181884766
```

encoding_test

Data Preprocessing - Captions

The caption will be predicted word by word. During training the model, captions will be the target variables (Y) that the model is learning to predict. Since the caption will be predicted word by word, each word is encoded into a fixed sized vector.

All the unique words were mapped to the frequency of their occurrences using the Dictionary data structure named "**sorted_frequency_count**". Total Words list was modified to include words only with frequency greater than the threshold value set in the document using the "**sorted_frequency_count**" dictionary. The threshold was set as 5 . The output was a Total Words of 1845.

```
Total words :
['in', 'the', 'on', 'is', 'and', 'dog', 'with', 'man', 'of', 'two', 'white',
sorted_frequency_count
[('in', 18987), ('the', 18420), ('on', 10746), ('is', 9345), ('and', 8863),
```

Total Words list and Reverse order sorted frequency count list

Two Python Dictionaries namely word_to_idx (word to index) and idx_to_word (index to word) were created.These two Python dictionaries can be used as follows:

word_to_idx['abc'] —> returns index of the word 'abc'
idx_to_word[k] —> returns the word whose index is 'k'

Every unique word in the vocabulary is represented by an integer (index). As concluded already, there are 1846 unique words in the corpus and thus each word will be represented by an integer index between 1 to 1846.
The <start>and <end> tokens are added to the two dictionaries.

```
word_to_idx['dog']
6
idx_to_word[6]
'dog'
```

word_to_idx and idx_to_word dictionaries

Word Embeddings

Word Embeddings are the texts converted into numbers and there may be different numerical representations of the same text.

Machine Learning algorithms and almost all Deep Learning Architectures are incapable of processing strings or plain text in their raw form. They require numbers as inputs to perform any sort of job, be it classification, regression etc.Every word (or index) will be mapped (embedded) to higher dimensional space through one of the word embedding techniques.

A vector representation of a word may be a one-hot encoded vector where 1 stands for the position where the word exists and 0 everywhere else. For example

[‘Word’,’Embeddings’,’are’,’Converted’,’into’,’numbers’]

The vector representation of “numbers” in this format according to the above dictionary is [0,0,0,0,0,1] and of converted is [0,0,0,1,0,0].

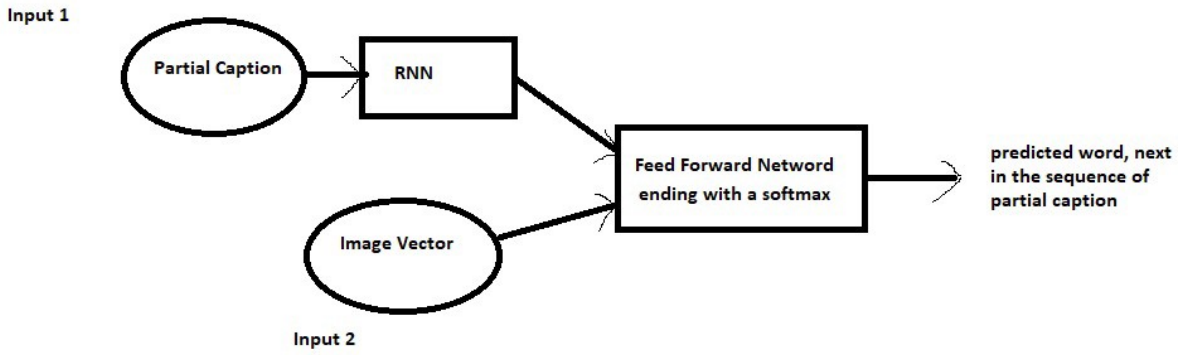
During the model building stage, we will see that each word/index is mapped to a 50-long vector using a pre-trained GLOVE word embedding model.Now each sequence contains 35 indices, where each index is a vector of length 50. Therefore x = 35*50 = 1750. Where 35 is the length of the maximum length of a caption available in the training data.

This will become a huge amount of data considered along with image feature vectors and even if we are able to manage to load this much data into the RAM, it will make the system very slow.

For this reason we use data generators a lot in Deep Learning.As already stated above, we will map the every word (index) to a 50-long vector and for this purpose, we will use a pre-trained GLOVE Model.

Model Architecture

Since the input consists of two parts, an image vector and a partial caption, We use the Functional API which allows us to create Merge Models.



High level architecture

ResNet-50 used for images is a convolutional neural network that is 50 layers deep. You can load a pretrained version of the network trained on more than a million images from the **ImageNet** database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

The **LSTM (Long Short Term Memory)** layer is a specialized Recurrent Neural Network to process the sequence input (partial captions in our case).

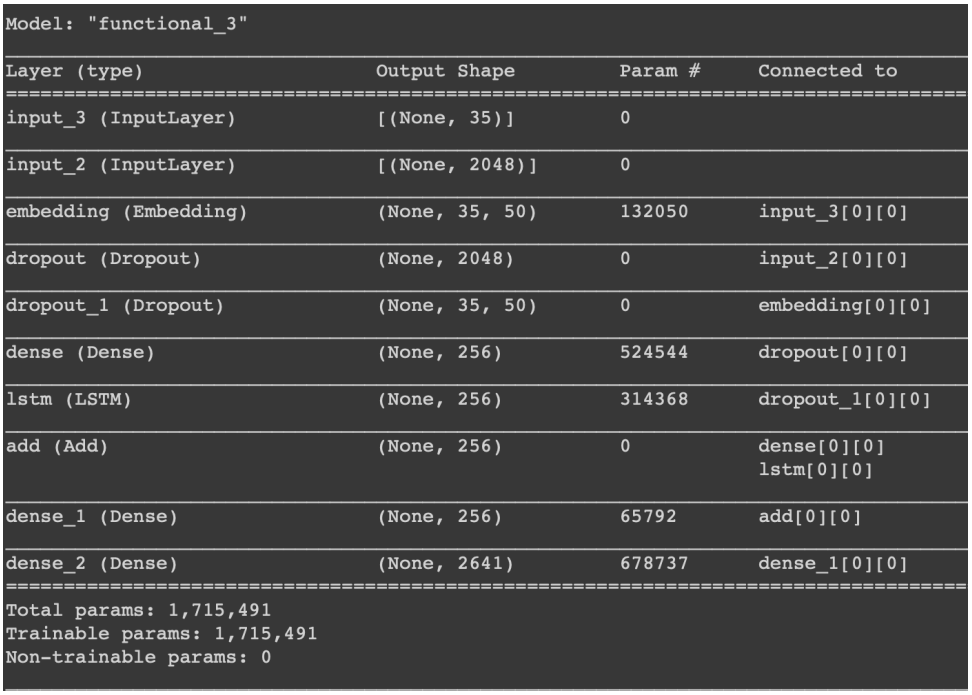
Since a pre-trained embedding layer is being used, the last two layers are made to freeze (trainable = False), before training the model, so that it does not get updated during the back propagation.

input_image (224x224 —> 2048 —> 256 dimensions)
input_caption(batch_size x 35 —> batch_size x 35 x 50 —> LSTM —> 256 dimensions)
input_image + input_caption —> 256 dimensions —> 2641 dimensions —> softmax —> probable_word

Simply presented as

Input_1 -> Partial Caption
Input_2 -> Image feature vector
Output -> An appropriate word, next in the sequence of partial caption provided in the input_1 (or in probability terms trained on image vector and the partial caption)

The model was compiled using the adam optimizer and categorical cross entropy as loss.



```
Model: "functional_3"
__________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
input_3 (InputLayer)            [(None, 35)]         0
__________________________________________________________________________________
input_2 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________
embedding (Embedding)           (None, 35, 50)       132050      input_3[0][0]
__________________________________________________________________________________
dropout (Dropout)               (None, 2048)         0           input_2[0][0]
__________________________________________________________________________________
dropout_1 (Dropout)             (None, 35, 50)       0           embedding[0][0]
__________________________________________________________________________________
dense (Dense)                   (None, 256)          524544      dropout[0][0]
__________________________________________________________________________________
lstm (LSTM)                     (None, 256)          314368      dropout_1[0][0]
__________________________________________________________________________________
add (Add)                       (None, 256)          0           dense[0][0]
                                                                 lstm[0][0]
__________________________________________________________________________________
dense_1 (Dense)                 (None, 256)          65792       add[0][0]
__________________________________________________________________________________
dense_2 (Dense)                 (None, 2641)         678737      dense_1[0][0]
==================================================================================
Total params: 1,715,491
Trainable params: 1,715,491
Non-trainable params: 0
__________________________________________________________________________________
```

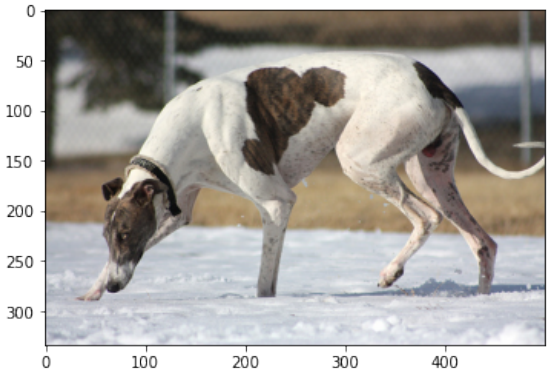
Combined Model

The model was then trained for 10 epochs and 7 pictures per batch (batch size).

Time Taken: I used the Google Colaboratory Notebook + GPU available with it and hence it took me approximately only 8 -12 minutes to train the model. However if you train it on a PC without GPU, it could take anywhere from 8 to 16 hours depending on the configuration of your system ,which personally, I would not recommend.

Inference

After preparing the data and building the model and training the model, the testing is performed by passing in new images to the model, i.e. how a caption is generated for a new test image. An example which is an output of my project is presented as follows.



Test image

Caption : black and white dog runs in the snow .

The vocabulary in the example = {and, black, dog, in, runs, snow, the, white}
The caption is generated iteratively, one word at a time as follows:

Iteration 1:
Input: Image vector + “start” (as partial caption)
Probable word: “black”

Iteration 2:
Input: Image vector + “start black”
Probable word: “and”

Iteration 3:
Input: Image vector + “start black and”
Probable word: “white”

Iteration 4:
Input: Image vector + “start black and white”
Probable word: “dog”

Iteration 5:
Input: Image vector + “start black and white dog”
Probable word: “runs”

Iteration 6:
Input: Image vector + “start black and white dog runs”
Probable word: “in”

Iteration 7:
Input: Image vector + “start black and white dog runs in”
Probable word: “the”

Iteration 8:
Input: Image vector + “start black and white dog runs in the”
Probable word: “snow”

Iteration 9:
Input: Image vector + “start black and white dog runs in the snow”
Probable word: “end”

Token ‘start’ which is used as the initial partial caption for any image. We stop when either we encounter an ‘end’ token which means the model thinks that this is the end of the caption or maximum threshold of the number of words generated by the model is reached.

Chapter 3

Implementation

The entire code of this project was implemented using Python 3 syntax on Google Colaboratory Notebook because of its easy and versatile interface and the free GPU which helps in performing computations within lesser time.

1. In the code snippet in figure (3.1), important libraries to perform basic operations are imported and Google Drive is mounted in my Project Notebook and Flicker 8k Dataset files are imported and unzipped.

```
import numpy as np
import matplotlib.pyplot as plt
import time
import pickle
from keras.utils import to_categorical
import json
import cv2

from google.colab import drive
drive.mount('/content/drive')

!unzip -uq "/content/drive/My Drive/Colab Notebooks/flicker8k-dataset.zip"
```

figure (3.1)

2. In the code snippet in figure (3.2), text file is read where image name with corresponding 5 captions is given.

```
[2] def readTextFile(path):

      with open(path) as f:
        captions = f.read()
      return captions

    captions = readTextFile("/content/flicker8k-dataset/Flickr8k_text/Flickr8k.token.txt")
    captions = captions.split("\n")[:-1]
    len(captions)

    40460
```

figure (3.2)

3. In figure (3.3), Dictionary to map each image with list of captions is created.

```
[2] def readTextFile(path):

      with open(path) as f:
        captions = f.read()
      return captions

    captions = readTextFile("/content/flicker8k-dataset/Flickr8k_text/Flickr8k.token.txt")
    captions = captions.split("\n")[:-1]
    len(captions)

    40460
```

figure (3.3)

4. In figure (3.4), clean_text function is uses to clean all the captions present in the descriptions dictionary i.e lower-casing all the words, removing punctuations, removing words of length less than 2 and special symbols like special symbols like @,<,# etc.

```
[4]  import re

     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
```

```
[5]  for key,cap_list in description.items():
         for i in range(len(cap_list)):
             cap_list[i] = clean_text(cap_list[i])
```

figure (3.4)

5. In figure (3.5), the cleaned up captions are stored in a descriptions.txt file in the disk for later use. It is reloaded again and converted into a dictionary.

```
[7]  with open("descriptions.txt","w") as f:
       f.write(str(description))
```

```
[8]  descriptions = None
     with open("descriptions.txt","r") as f:
       descriptions = f.read()
     json_acceptable_string = descriptions.replace("'","\"")
     descriptions = json.loads(json_acceptable_string)
     print(type(descriptions))
```

```
     <class 'dict'>
```

figure (3.5)

6. In figure (3.6), Vocabulary is a set of unique words in the descriptions. Sentences per image are split into words and Vocabulary is updated with a new word. total_words is all words in descriptions and des is one caption in list of captions and i is one word in that caption.

```
[9]  vocabulary = set()

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

     print("Vocab size : %d"%len(vocabulary))

     total_words = []

     for key in descriptions.keys():
       [total_words.append(i) for des in descriptions[key] for i in des.split()]
     print("Total words : %d"%len(total_words))
```

```
     Vocab size : 8424
     Total words : 373837
```

figure (3.6)

7. In figure (3.7), words from the vocabulary are filtered according to a certain threshold frequency. **sorted_freq_cnt** is a dictionary with words(keys) and frequency(values), sorted in reverse order of frequency. sorted_freq_cnt is then converted into a tuple (word, frequency) with all the words with frequency greater than the threshold. **total_words** list then is updated using sorted_freq_cnt.

```
from collections import Counter

counter = Counter(total_words)
freq_cnt = dict(counter)
len(freq_cnt.keys())

sorted_freq_cnt = sorted(freq_cnt.items(), key= lambda x: x[1],reverse=True)

threshold  = 10
sorted_freq_cnt  = [x for x in sorted_freq_cnt if x[1]>threshold]
total_words = [x[0] for x in sorted_freq_cnt]
```

figure (3.7)

8. Next the training data (6000 images and corresponding captions) and testing data (2000 images) was loaded.

```
[11] train_file_data = readTextFile("/content/flicker8k-dataset/Flickr8k_text/Flickr_8k.trainImages.txt")
     test_file_data = readTextFile("/content/flicker8k-dataset/Flickr8k_text/Flickr_8k.testImages.txt")

     train = [e[:-4] for e in train_file_data.split('\n')[:-1]]
     test = [e[:-4] for e in test_file_data.split('\n')[:-1]]
```

figure (3.8)

9. In figure (3.9), an empty dictionary train_descriptions is created which is used to store image name (key) and captions with “<s>" and “<e>” tokens (values).

```
[12] train_descriptions = {}
     for img_id in train:
         train_descriptions[img_id] = []
         for cap in descriptions[img_id]:
             cap_to_append = "<s> "+cap+" <e>"
             train_descriptions[img_id].append(cap_to_append)
```

figure (3.9)

10. In figure (3.10) required Keras libraries are imported and ResNet50 model , pretrained on ImageNet Dataset, is loaded. new_model is created without the last softmax layer of the model as it will be defined later when the Combined Model architecture is defined.

```
import tensorflow as tf

tf.compat.v1.disable_eager_execution()
from tensorflow.python.keras.layers import *
from tensorflow.python.keras.layers.advanced_activations import LeakyReLU
from tensorflow.python.keras.models import Sequential,Model
from tensorflow.compat.v1.keras.optimizers import Adam
from keras.preprocessing import image
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.applications.resnet50 import ResNet50,preprocess_input
```

```
[16] model = ResNet50(weights='imagenet',input_shape=(224,224,3))
     new_model = Model(inputs = model.input,outputs = model.layers[-2].output)
     new_model.summary()
```

figure (3.10)

11. In figure (3.11), function preprocess_image(), takes an image and converts it into a normalised array or target size. 1 is image number and (224, 224) is reshaping dimension and 3 is no. of channels while encode_image() returns feature vector of size 2048 per image.

```
[17] def preprocess_image(img):
       img = image.load_img(img,target_size=(224,224))
       img = image.img_to_array(img)
       img = np.expand_dims(img,axis=0)
       img = preprocess_input(img)
       return img

     def encode_image(img):
       img = preprocess_image(img)
       fea_vec = new_model.predict(img)
       fea_vec = fea_vec.reshape((fea_vec.shape[1],))
       return fea_vec
```

figure (3.11)

12. In encoding_train dictionary image_id is mapped to its 2048 vector feature vector & stored on the disk using pickle ext. file. The time taken was around 106 seconds. Similarly test data was encoded as well.

```
[17] start = time.time()
     encoding_train = {}
     for ix,img_id in enumerate(train):
       try:

         path = img_path+"/"+img_id+".jpg"
         encoding_train[img_id] = encode_image(path)
       except:
         continue

       if ix%1000 == 0:
         print("Encoding image :"+str(ix))
     print("Time taken to encode 6000 images in sec :" + str(time.time()-start))

     with open("./encoded_train_images.pkl",'wb') as f:
       pickle.dump(encoding_train,f)
```

figure (3.12)

13. Two dictionaries word_to_idx & idx_to_word are created to map words with their index and vice versa. Also maximum length of a caption comes out to be 35 characters.

```
[21]  word_to_idx = {}
      idx_to_word = {}

      for i,word in enumerate(total_words):
        word_to_idx[word] = i+1
        idx_to_word[i+1] = word

     word_to_idx['<s>'] = 1846
     word_to_idx['<e>'] = 1847
     idx_to_word[1846] = '<s>'
     idx_to_word[1847] = '<e>'

     vocab_size = len(idx_to_word) + 1
```

figure (3.12)

14. Data generator, intended to yield input sequence, output sequence and image encoding when n is equal to batch size.

```
def data_generator(train_descriptions,encoding_train,word_to_idx,max_len,batch_size):
  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 == batch_size:
        yield([np.array(X1),np.array(X2)],np.array(y))
        X1,X2,y = [],[],[]
        n=0
```

figure (3.13)

15. Glove file is loaded which contains words and their corresponding 50 dimensional vectors. Each word in the vocabulary is then mapped to its 50 dimensional vectors in the function get_embedding_matrix().

```
[56] f = open('/content/drive/My Drive/Colab Notebooks/glove.6B.50d.txt',encoding='utf-8')
     embedding_index = {}
     for line in f:
       values = line.split()
       word = values[0]
       word_embedding = np.asarray(values[1:],dtype='float')
       embedding_index[word] = word_embedding
     f.close()

     def get_embedding_matrix():
       emb_dim = 50
       embedding_output = np.zeros((vocab_size,emb_dim))

       for word,idx in word_to_idx.items():
          embedding_vector = embedding_index.get(word)
          if embedding_vector is not None:
              embedding_output[idx] = embedding_vector
       return embedding_output

     embedding_matrix = get_embedding_matrix()
```

figure (3.14)

16. Features are extracted and Dropout layers help in reducing excessive features to avoid overfitting.

```
[59] # Image feature extraction
     input_img_fea = Input(shape=(2048,))
     inp_img1 = Dropout(0.3)(input_img_fea)
     inp_img2 = Dense(256,activation='relu')(inp_img1)

     # 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)
```

figure (3.15)

17. A functional model is created to combine partial caption input and image features input in figure (3.16).

```
     decoder1 = add([inp_img2,inp_cap3])
     decoder2 = Dense(256,activation='relu')(decoder1)
     output = Dense(vocab_size,activation='softmax')(decoder2)

     # Combined Model
     model = Model(inputs= [input_img_fea,input_cap], outputs = output)
     model.summary()
```

figure (3.16)

18. The previously removed softmax layers are now replaced by embedded word matrix for partial caption prediction and freeze the trainable features since pretrained model is being used (Glove) .

```
[61] model.layers[2].set_weights([embedding_matrix])
     model.layers[2].trainable = False
     model.compile(loss='categorical_crossentropy',optimizer='adam')
```

figure (3.16)

19. Finally the model was trained using a batch size = 7 and epochs = 10 and steps were 500, and the step where the error is minimum was saved as the best model.

```
epochs = 10
batch_size = 7
steps = len(train_descriptions)//12

for i in range(epochs):
  gen = data_generator(train_descriptions,encoding_train,word_to_idx,max_len,batch_size)
  model.fit_generator(gen,epochs=1,steps_per_epoch=steps,verbose=1)
  model.save("best_model.h5")
```

figure (3.17)

The model took around 8 - 10 minutes to train.

```
WARNING:tensorflow:From <ipython-input-64-870c9dac8994>:3: Model.fit_generator (from tensorflow.python.keras.
Instructions for updating:
Please use Model.fit, which supports generators.
500/500 [==============================] - 37s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 4.6736
500/500 [==============================] - 37s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 3.8642
500/500 [==============================] - 38s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 3.5469
500/500 [==============================] - 38s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 3.3346
500/500 [==============================] - 37s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 3.1744
500/500 [==============================] - 37s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 3.0471
500/500 [==============================] - 37s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 2.9490
500/500 [==============================] - 38s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 2.8656
500/500 [==============================] - 38s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 2.7879
500/500 [==============================] - 37s 75ms/step - batch: 249.5000 - size: 344.2360 - loss: 2.7215
```

figure (3.18)

Predictions on the test data are shown in next chapter.

Chapter 4

Predictions

Predictions were made using the predict_caption() function which processes the partial caption ans image feature vector and breaks the loop when <e> (end) token is found. It joins the word to form a captions and returns it.

```
[1]  def predict_caption(photo):
       in_text = "<s>"
       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')

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

         if word == '<e>':
           break

       final_caption = in_text[3:-3]
       final_caption = ''.join(final_caption)

       return final_caption
```

predict_caption() function

To make predictions, at random 10 images from the test dataset are selected corresponding captions along with the image are shown.

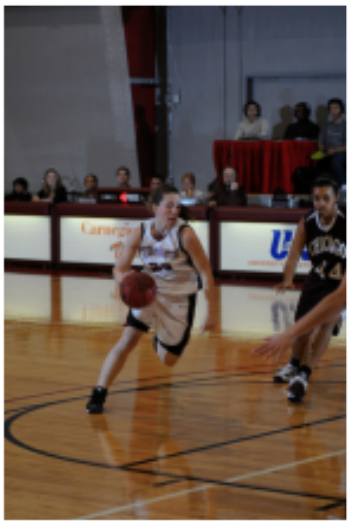
```
for i in range(15):
  idx = np.random.randint(0,1000)
  all_img_name = list(encoding_test.keys())
  img_name = all_img_name[idx]
  photo_2048 = encoding_test[img_name].reshape((1,2048))

  i = plt.imread(img_path+img_name+".jpg")
  caption = predict_caption(photo_2048)
  print(caption + ".")
  plt.axis('off')
  plt.imshow(i)
  plt.show()
```

Randomly selected image caption prediction

Outputs

two women dressed in dresses .

snowboarder is skiing down snowy hill .



man in blue shirt is jumping into the air .

black and white dog is running through the grass .

two children are skiing down snowy hill .

the dog jumps over bar .

Chapter 5
Future Scope & Conclusion

The captions predicted, though were relevant for most images but sometimes did not match the content of the image. This can be solved by training the model on larger dataset.

Due to the stochastic nature of the models, the captions generated by every time the code is implemented may not be exactly similar to those generated in the first time.

This was a basic project and a lot of modifications can be made to improve this program like:

- Using a larger dataset.
- Changing the model architecture, e.g. include an attention module.
- Doing more hyper parameter tuning (learning rate, batch size, number of layers, number of units, dropout rate, batch normalization etc.).
- Use the cross validation set to understand overfitting.
- Using Beam Search instead of Greedy Search during Inference.
- Using BLEU Score to evaluate and measure the performance of the model.
- Writing the code in a proper object oriented way

A similar, more advanced example of my automatic image captioning project is state-of-the-art system created by Microsoft called as Caption Bot.

Machine Learning and Deep Learning is an ever growing and has caught a lot of interest among students. Hence more and more viable and likewise more models are being developed regularly with higher accuracy. We can expect more advance image captioning since it has such important applications.

REFERENCES

- https://towardsdatascience.com
- https://medium.com/analytics-vidhya/basics-of-using-pre-trained-glove-vectors-in-python-d38905f356db
- https://keras.io/api/applications/
- https://www.kaggle.com/watts2/glove6b50dtxt
- https://medium.com/@hanify/sequential-api-vs-functional-api-model-in-keras-266823d7cd5e
- https://www.quora.com/What-are-10-15-applications-of-image-captioning-Deep-Learning
- https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/
- https://codingblocks.com/centres/pitampura.html
- https://ieeexplore.ieee.org/document/8276124