भारतीय प्रौद्योगिकी
संस्थान जम्मू
INDIAN INSTITUTE OF
TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

IMAGE-CAPTIONING

Project Report

Submitted to :

Prof. Virendra Singh

Indian Institute of Technology

Bombay

Summer Internship 2019

20th May – 10th July 2019

Submitted By:

Jatin Nandal , Rajat Khanna

भारतीय प्रौद्योगिकी
संस्थान जम्मू
INDIAN INSTITUTE OF
TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

Preface

This report document is the work done during the summer internship at IIT Bombay on Image Captioning, under the supervision of Dr. Virendra Singh. The report will give an overview of the tasks completed during the period of internship with technical details. Then the results obtained are discussed and analyzed. We hope we succeed in our attempt.

भारतीय प्रौद्योगिकी
संस्थान जम्मू
INDIAN INSTITUTE OF
TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

Acknowledgements

We take this opportunity to acknowledge all the people who have helped us whole heartedly in every stage of this project.

We are indebtedly grateful to Prof. Virendra Singh for his support. We are also grateful to our Project guide Dr. Chandramani Chaudhary for her valuable guidance. We also extend our sincere thanks to People Invited by Prof. Virendra Singh for their valuable inspiration. We also extend our sincere thanks to all other PhD scholars of Electrical Engineering Department and our peers who have helped us with this project.

We would like to thank authorities of Indian Institute of Technology Bombay to let us be a part of their institution and learn from it.

भारतीय प्रौद्योगिकी संस्थान जम्मू
INDIAN INSTITUTE OF TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

भारतीय प्रौद्योगिकी संस्थान जम्मू
INDIAN INSTITUTE OF TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

Introduction

We have designed a model which predicts a suitable caption for a given image . We have used Advanced search algorithms and they give better results compared to the naive ones . We tried to predict caption using zero shot learning , which is a technique in which we predict the classes which were unseen to the model . For this purpose we have used Flickr8k Data set and CUB Data set . We have used Google Colab GPU for the purpose of Training and Testing of our model .

भारतीय प्रौद्योगिकी संस्थान जम्मू
INDIAN INSTITUTE OF TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

IMAGE-CAPTIONING

Zero-Shot learning method aims to **Image Captioning** is the process of generating textual description of an image. It uses both **Natural Language Processing** and **Computer Vision** to generate the captions. The data set will be in the form [image→captions]. The data set consists of input images and their corresponding output captions.

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

1)Introduction:

What do we see in the below picture?

Well some of us might say “A white dog in a grassy area”, some may say “White dog with brown spots” and yet some others might say “A dog on grass and some pink flowers”.Definitely all of these captions are relevant

for this image and there may be some others also. But the point we want to make is; it’s so easy for us, as human beings, to just have a glance at a picture and describe it in an appropriate language. Even a 5 year old could do this with utmost ease.

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 data set.

This problem was well researched by **Andrej Karapathy** in his PhD thesis at Stanford , who is also now the Director of AI at Tesla **.** This deals that how Deep Learning can be used to solve this problem of generating a caption for a given image, hence the name **Image Captioning.**

2)Motivation:

We must first understand how important this problem is to real world scenarios. Let’s 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.

3) Prerequisites:

This project assumes familiarity with basic Deep Learning concepts like Multi-layered Perceptrons, Convolution Neural Networks, Recurrent Neural Networks, Transfer Learning, Gradient Descent, Backpropagation, Overfitting, Probability, Text Processing, Python syntax and data structures, Keras library, etc

4)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 project, we have used the **Flickr 8k dataset** which you can download provided by the University of Illinois at Urbana-Champaign. Also training a model with large number of images may not

be feasible on a system which is not a very high end PC/Laptop.This data set 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

5)Understanding the data:

If you have downloaded the data 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.

We can read this file as follows:

filename = "/content/drive/My Drive/Flickr8k.token.txt"

 # load descriptions

doc = load_doc(filename)

The text file looks as follows:

```
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 .
1001773457_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 .
1002674143_1b742ab4b8.jpg#1     A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2     A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
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 .
1003163366_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 sleeping 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 holding leash of dog sitting on ground
1007129816_e794419615.jpg#0     A man in an orange hat starring at something .
1007129816_e794419615.jpg#1     A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2     A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3     A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4     The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0     A child playing on a rope net .
1007320043_627395c3d8.jpg#1     A little girl climbing on red roping .
1007320043_627395c3d8.jpg#2     A little girl in pink climbs a rope bridge at the park .
1007320043_627395c3d8.jpg#3     A small child grips onto the red ropes at the playground .
1007320043_627395c3d8.jpg#4     The small child climbs on a red ropes on a playground .
```

Thus every line contains the <image name>#i <caption>, where 0≤i≤4i.e. the name of the image, caption number (0 to 4) and the actual caption.

Now, we create a dictionary named “descriptions” which contains the name of the image (without the .jpg extension) as keys and a list of the 5 captions for the corresponding image as values.

```
[ ]  def load_descriptions(doc):
       mapping = dict()
       # process lines
       for line in doc.split('\n'):
         # split line by white space
         tokens = line.split()
         if len(line) < 2:
           continue
         # take the first token as the image id, the rest as the description
         image_id, image_desc = tokens[0], tokens[1:]
         # extract filename from image id
         image_id = image_id.split('.')[0]
         # convert description tokens back to string
         image_desc = ' '.join(image_desc)
         # create the list if needed
         if image_id not in mapping:
           mapping[image_id] = list()
         # store description
         mapping[image_id].append(image_desc)
       return mapping

     # parse descriptions
     descriptions = load_descriptions(doc)
     print('Loaded: %d ' % len(descriptions))
```

```
Loaded: 8092
```

For example with reference to the above screenshot the dictionary will look as follows:

```
[ ]  descriptions['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 playhouse',
 'little girl climbing the stairs to her playhouse',
 'little girl in pink dress going into wooden cabin']
```

6)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:

```
[ ]  def clean_descriptions(descriptions):
       # prepare translation table for removing punctuation
       table = str.maketrans('', '', string.punctuation)
       for key, desc_list in descriptions.items():
         for i in range(len(desc_list)):
           desc = desc_list[i]
           # tokenize
           desc = desc.split()
           # convert to lower case
           desc = [word.lower() for word in desc]
           # remove punctuation from each token
           desc = [w.translate(table) for w in desc]
           # remove hanging 's' and 'a'
           desc = [word for word in desc if len(word)>1]
           # remove tokens with numbers in them
           desc = [word for word in desc if word.isalpha()]
           # store as string
           desc_list[i] =  ' '.join(desc)

     # clean descriptions
     clean_descriptions(descriptions)
```

Create a vocabulary of all the unique words present across all the 8000*5 (i.e. 40000) image captions (**corpus**) in the data set :

vocabulary = set()
for key in descriptions.keys():
[vocabulary.update(d.split()) for d in descriptions[key]]
print('Original Vocabulary Size: %d' % len(vocabulary))
Original Vocabulary Size: 8763

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 cloud.

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.

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

```
[ ]  # Consider only words which occur at least 10 times in the corpus
     word_count_threshold = 10
     word_counts = {}
     nsents = 0
     for sent in all_train_captions:
         nsents += 1
         for w in sent.split(' '):
             word_counts[w] = word_counts.get(w, 0) + 1

     vocab = [w for w in word_counts if word_counts[w] >= word_count_threshold]
     print('preprocessed words %d -> %d' % (len(word_counts), len(vocab)))
```

```
preprocessed words 7578 -> 1651
```

So now we have only 1651 unique words in our vocabulary. However, we will append 0's (zero padding explained later) and thus total words = 1651+1 = **1652** (one index for the 0).

7)Loading the DataSet:

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”.

```
filename = 'dataset/TextFiles/Flickr_8k.trainImages.txt'
doc = load_doc(filename)
train = list()
for line in doc.split('\n'):
identifier = line.split('.')[0]
train.append(identifier)
print('Dataset: %d' % len(train))
Dataset: 6000
```

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 colab) 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.

```
[ ]  # load clean descriptions into memory
     def load_clean_descriptions(filename, dataset):
       # load document
       doc = load_doc(filename)
       descriptions = dict()
       for line in doc.split('\n'):
         # split line by white space
         tokens = line.split()
         # split id from description
         image_id, image_desc = tokens[0], tokens[1:]
         # skip images not in the set
         if image_id in dataset:
           # create list
           if image_id not in descriptions:
             descriptions[image_id] = list()
           # wrap description in tokens
           desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
           # store
           descriptions[image_id].append(desc)
       return descriptions

     # descriptions
     train_descriptions = load_clean_descriptions('descriptions.txt', train)
     print('Descriptions: train=%d' % len(train_descriptions))
```

8) Data Processing:

Images are nothing but input (X) to our model. As we 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 InceptionV3 model (Convolutional Neural Network)

```
[ ]  def preprocess(image_path):
         # Convert all the images to size 299x299 as expected by the inception v3 model
         img = image.load_img(image_path, target_size=(299, 299))
         # Convert PIL image to numpy array of 3-dimensions
         x = image.img_to_array(img)
         # Add one more dimension
         x = np.expand_dims(x, axis=0)
         # preprocess the images using preprocess_input() from inception module
         x = preprocess_input(x)
         return x
```

```
[ ]  # Load the inception v3 model
     model = InceptionV3(weights='imagenet')
```

```
WARNING: Logging before flag parsing goes to stderr.
W0710 15:24:41.811226 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:74: The name tf.get_default_graph is deprecated. Please u

W0710 15:24:41.860867 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:517: The name tf.placeholder is deprecated. Please use tf

W0710 15:24:41.869579 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:4138: The name tf.random_uniform is deprecated. Please us

W0710 15:24:41.911515 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:174: The name tf.get_default_session is deprecated. Pleas

W0710 15:24:41.912709 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:181: The name tf.ConfigProto is deprecated. Please use tf

W0710 15:24:45.025610 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1834: The name tf.nn.fused_batch_norm is deprecated. Plea

W0710 15:24:45.426249 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3976: The name tf.nn.max_pool is deprecated. Please use t

W0710 15:24:46.215392 139904571324288 deprecation_wrapper.py:119] From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3980: The name tf.nn.avg_pool is deprecated. Please use t

Downloading data from https://github.com/fchollet/deep-learning-models/releases/download/v0.5/inception_v3_weights_tf_dim_ordering_tf_kernels.h5
96116736/96112376 [==============================] - 1s 0us/step
```

```
[ ]  # Create a new model, by removing the last layer (output layer) from the inception v3
     model_new = Model(model.input, model.layers[-2].output)
```

created by Google Research.This model was trained on Imagenet dataset to perform image classification on 1000 different classes of images.

Hence, we just remove the last **softmax** layer from the model and extract a 2048 length vector for every image as follows:We save all the bottleneck train 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**".

9)Data Cleaning:

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"and "ixtoword" . Stating simply, we will represent every unique word in the vocabulary by an integer (index). As seen above, we have 1652 unique words in the corpus and thus each word will be represented by an integer index between 1 to 1652.

These two Python dictionaries can be used as follows:

wordtoix['abc'] -> returns index of the word 'abc'

ixtoword[k] -> returns the word whose index is 'k'

The code used is as below:

```
[ ]  ixtoword = {}
     wordtoix = {}

     ix = 1
     for w in vocab:
         wordtoix[w] = ix
         ixtoword[ix] = w
         ix += 1
```

```
[ ]  wordtoix['endseq']
```

```
14
```

```
[ ]  # convert a dictionary of clean descriptions to a list of descriptions
     def to_lines(descriptions):
       all_desc = list()
       for key in descriptions.keys():
         [all_desc.append(d) for d in descriptions[key]]
       return all_desc

     # calculate the length of the description with the most words
     def max_length(descriptions):
       lines = to_lines(descriptions)
       return max(len(d.split()) for d in lines)

     # determine the maximum sequence length
     max_length = max_length(train_descriptions)
     print('Description Length: %d' % max_length)
```

```
Description Length: 34
```

There is one more parameter that we need to calculate, i.e., the maximum length of a caption and we do it as below

10)Data Prepration:

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.

Consider we have 3 images and their 3 corresponding captions as follows:

(Train image 1) Caption -> The black cat sat on grass

(Train image 2) Caption -> The white cat is walking on road

(Test image) Caption -> The black cat is walking on grass

Now, let’s say we use the first two images and their captions to train the model and the third image to test our model.

Now the questions that will be answered are: how do we frame this as a supervised learning problem?, what does the data matrix look like? how many data points do we have?, etc.

First we need to convert both the images to their corresponding 2048 length feature vector as discussed above. Let “Image_1” and “Image_2” be the feature vectors of the first two images respectively

Secondly, let’s build the vocabulary for the first two (train) captions by adding the two tokens “startseq” and “endseq” in both of them: (Assume we have already performed the basic cleaning steps)

Caption_1 -> “startseq the black cat sat on grass endseq”

Caption_2 -> “startseq the white cat is walking on road endseq”

vocab = {black, cat, endseq, grass, is, on, road, sat, startseq, the, walking, white}let's give an index to each word in the vocabulary:black -1, cat -2, endseq -3, grass -4, is -5, on -6, road -7, sat -8, startseq -9, the -10, walking -11, white -12

Now let's try to frame it as a **supervised learning problem** where we have a set of data points D = {Xi, Yi}, where Xi is the feature vector of data point 'i' and Yi is the corresponding target variable.

Let's take the first image vector Image_1 and its corresponding caption "startseq the black cat sat on grass endseq".

For the first time, we provide the image vector and the first word as input and try to predict the second word, i.e.:

Input = Image_1 + 'startseq'; Output = 'the'

Then we provide image vector and the first two words as input and try to predict the third word, i.e.:

Input = Image_1 + 'startseq the'; Output = 'cat'

And so on...

Thus, we can summarize the data matrix for one image and its corresponding caption as follows:

	Xi		Yi
i	Image feature vector	Partial Caption	Target word
1	Image_1	startseq	the
2	Image_1	startseq the	black
3	Image_1	startseq the black	cat
4	Image_1	startseq the black cat	sat
5	Image_1	startseq the black cat sat	on
6	Image_1	startseq the black cat sat on	grass
7	Image_1	startseq the black cat sat on grass	endseq

It must be noted that, one image+caption is not a single data point but are multiple data points depending on the length of the caption.

Need for a Data Generator:

I hope this gives you a good sense as to how we can prepare the dataset for this problem. However, there is a big catch in this. In the above example, we have only considered 2 images and captions which have lead to 15 data points.

However, in our actual training dataset we have 6000 images, each having 5 captions. This makes a total of 30000 images and captions.

Even if we assume that each caption on an average is just 7 words long, it will lead to a total of 30000*7 that is 210000 data points.

Size of the data matrix = n*m

Where n-> number of data points (assumed as 210000)

And m-> length of each data point

Clearly m= Length of image vector(2048) + Length of partial caption(x).

m = 2048 + x

But what is the value of x? Every word (or index) will be mapped (embedded) to higher dimensional space through one of the word embedding techniques.Later, during the model building stage, we will see that each word/index is mapped to a 200-long vector using a pre-trained GLOVE word embedding model.Now each sequence contains 34 indices, where each index is a vector of length 200. Therefore x = 34*200 = 6800

Hence, m = 2048 + 6800 = 8848.

Finally, size of data matrix= 210000 * 8848= 1858080000 blocks.

Now even if we assume that one block takes 2 byte, then, to store this data matrix, we will require more than 3 GB of main memory.This is pretty

huge requirement 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. Data Generators are a functionality which is natively implemented in Python. The ImageDataGenerator class provided by the Keras API is nothing but an implementation of generator function in Python.

So how does using a generator function solve this problem?

If you know the basics of Deep Learning, then you must know that to train a model on a particular dataset, we use some version of Stochastic Gradient Descent (SGD) like Adam, Adagrad, etc.

With SGD, we do not calculate the loss on the entire data set to update the gradients. Rather in every iteration, we calculate the loss on a batch of data points (typically 64, 128, 256, etc.) to update the gradients.This means that we do not require to store the entire dataset in the memory at once. Even if we have the current batch of points in the memory, it is sufficient for our purpose.A generator function in Python is used exactly for this purpose. It’s like an iterator which resumes the functionality from the point it left the last time it was called.

The code for data generator is as follows:

```
]  # data generator, intended to be used in a call to model.fit_generator()
   def data_generator(descriptions, photos, wordtoix, max_length, num_photos_per_batch):
       X1, X2, y = list(), list(), list()
       n=0
       # loop for ever over images
       while 1:
           for key, desc_list in descriptions.items():
               n+=1
               # retrieve the photo feature
               photo = photos[key+'.jpg']
               for desc in desc_list:
                   # encode the sequence
                   seq = [wordtoix[word] for word in desc.split(' ') if word in wordtoix]
                   # split one sequence into multiple X, y pairs
                   for i in range(1, len(seq)):
                       # split into input and output pair
                       in_seq, out_seq = seq[:i], seq[i]
                       # pad input sequence
                       in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                       # encode output sequence
                       out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                       # store
                       X1.append(photo)
                       X2.append(in_seq)
                       y.append(out_seq)
               # yield the batch data
               if n==num_photos_per_batch:
                   yield [[array(X1), array(X2)], array(y)]
                   X1, X2, y = list(), list(), list()
                   n=0
```

11)Word Embedding:

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:

```
[ ]  # Load Glove vectors
     # glove_dir = '../../storage/glove'
     embeddings_index = {} # empty dictionary
     # f = open(os.path.join(glove_dir, 'glove.6B.200d.txt'), encoding="utf-8")
     f = open("/content/drive/My Drive/glove.6B.200d.txt", encoding="utf-8")


     for line in f:
         values = line.split()
         word = values[0]
         coefs = np.asarray(values[1:], dtype='float32')
         embeddings_index[word] = coefs
     f.close()
     print('Found %s word vectors.' % len(embeddings_index))
```

```
Found 400000 word vectors.
```

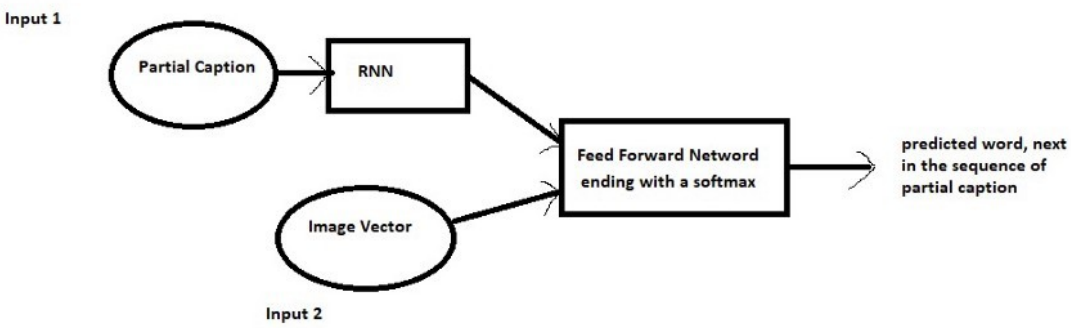
Now, for all the 1652 unique words in our vocabulary, we create an embedding matrix which will be loaded into the model before training

```
[ ]  embedding_dim = 200

     # Get 200-dim dense vector for each of the 10000 words in out vocabulary
     embedding_matrix = np.zeros((vocab_size, embedding_dim))

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

12)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:



We define the model as follows:

```
[ ]  inputs1 = Input(shape=(2048,))
     fe1 = Dropout(0.5)(inputs1)
     fe2 = Dense(256, activation='relu')(fe1)
     inputs2 = Input(shape=(max_length,))
     se1 = Embedding(vocab_size, embedding_dim, mask_zero=True)(inputs2)
     se2 = Dropout(0.5)(se1)
     se3 = LSTM(256)(se2)
     decoder1 = add([fe2, se3])
     decoder2 = Dense(256, activation='relu')(decoder1)
     outputs = Dense(vocab_size, activation='softmax')(decoder2)
     model = Model(inputs=[inputs1, inputs2], outputs=outputs)
```

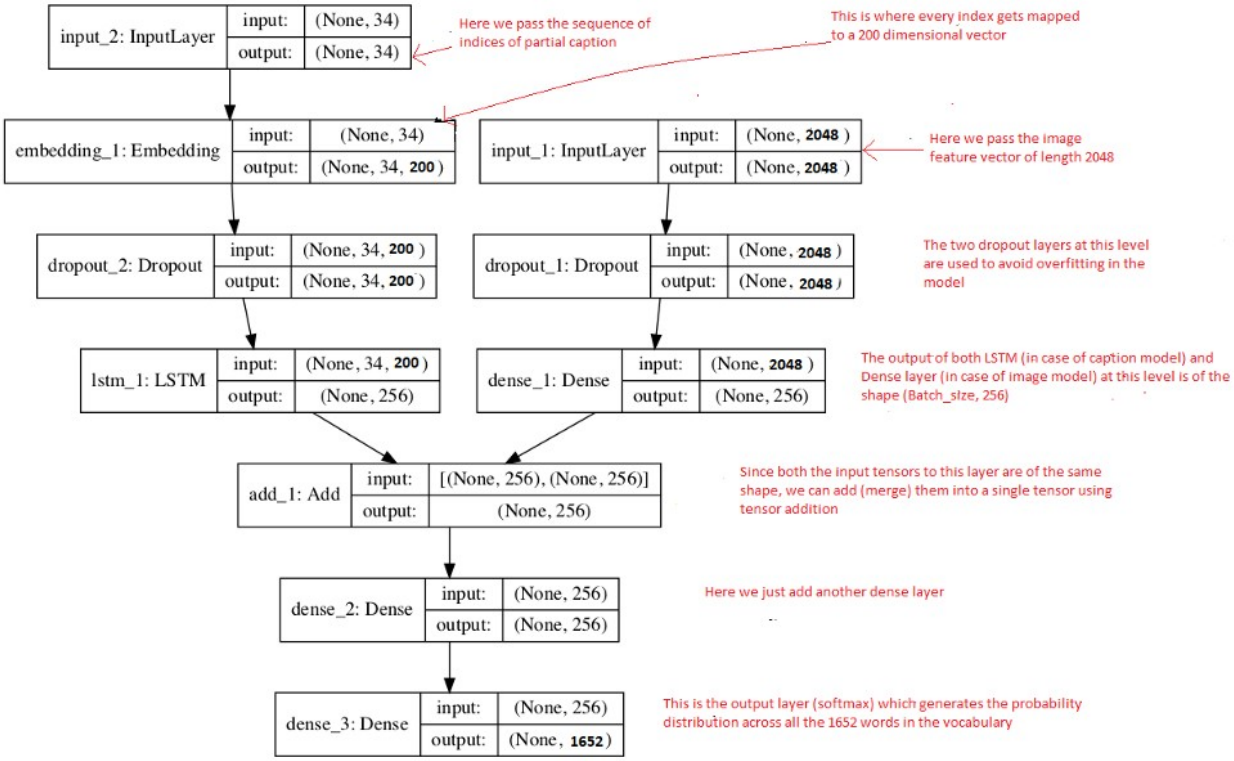
```
W0710 15:26:19.855201 139904571324288 deprecation.py:506] From /
```

Let’s look at the model summary:

```
[ ]  model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_3 (InputLayer)         (None, 34)                0         
_________________________________________________________________
input_2 (InputLayer)         (None, 2048)              0         
_________________________________________________________________
embedding_1 (Embedding)      (None, 34, 200)           330400    
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 34, 200)           0         
_________________________________________________________________
dense_1 (Dense)              (None, 256)               524544    
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               467968    
_________________________________________________________________
add_1 (Add)                  (None, 256)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792     
_________________________________________________________________
dense_3 (Dense)              (None, 1652)              424564    
=================================================================
Total params: 1,813,268
Trainable params: 1,813,268
Non-trainable params: 0
_________________________________________________________________
```

The below plot helps to visualize the structure of the network and better understand the two streams of input:



The text in red on the right side are the comments provided for you to map your understanding of the data preparation to model architecture.

The **LSTM (Long Short Term Memory)**layer is nothing but a specialized Recurrent Neural Network to process the sequence input.Since we are using a pre-trained embedding layer, we need to freeze it (trainable = False), before training the model, so that it does not get updated during the backpropagation.Finally we compile the model using the adam optimizer

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

Finally the weights of the model will be updated through backpropagation

algorithm and the model will learn to output a word, given an image feature vector and a partial caption. So in summary, we have:

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 we say conditioned on image vector and the partial caption)

The model was then trained for 30 epochs with the initial learning rate of 0.001 and 3 pictures per batch (batch size). However after 20 epochs, the learning rate was reduced to 0.0001 and the model was trained on 6 pictures per batch.

13)Inference

So till now, we have seen how to prepare the data and build the model. In the final step of this series, we will understand how do we test (infer) our model by passing in new images, i.e. how can we generate a caption for a new test image.

Caption -> the black cat is walking on grass

Also the vocabulary in the example was:

vocab = {black, cat, endseq, grass, is, on, road, sat, startseq, the, walking, white}We will generate the caption iteratively, one word at a time as follows:

Input: Image vector + “startseq” (as partial caption)

Expected Output word: “the”

The model generates a 12-long vector(in the sample example while 1652-long vector in the original example) which is a probability distribution across all the words in the vocabulary. For this reason we greedily select the word with the maximum probability, given the feature vector and partial caption.

If the model is trained well, we must expect the probability for the word “the” to be maximumThis is called as Maximum Likelihood Estimation (MLE) i.e. we select that word which is most likely according to the model for the given input. And sometimes this method is also called as **Greedy Search**, as we greedily select the word with maximum probability.We stop when either of the below two conditions is met:

- We encounter an ‘**endseq**’ token which means the model thinks that this is the end of the caption. (You should now understand the importance of the ‘endseq’ token)
- We reach a maximum **threshold** of the number of words generated by the model.

If any of the above conditions is met, we break the loop and report the generated caption as the output of the model for the given image. The code for inference is as follows:

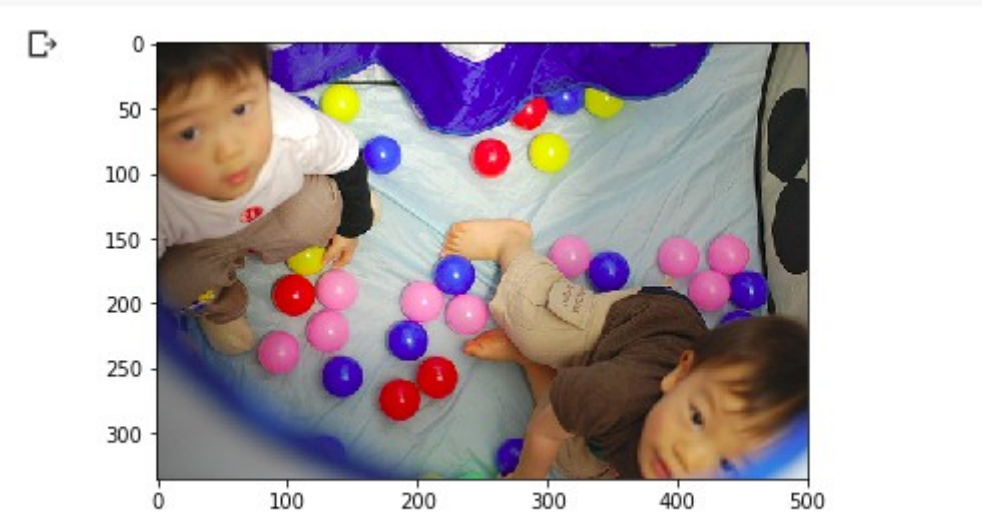
```
[ ]  def greedySearch(photo):
         in_text = 'startseq'
         for i in range(max_length):
             sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
             sequence = pad_sequences([sequence], maxlen=max_length)
             yhat = model.predict([photo,sequence], verbose=0)
             yhat = np.argmax(yhat)
             word = ixtoword[yhat]
             in_text += ' ' + word
             if word == 'endseq':
                 break
         final = in_text.split()
         final = final[1:-1]
         final = ' '.join(final)
         return final
```

We can also use a more advanced search algorithm which is known as Beam Search Algorithm in which we n captions at a time and decide the next word based on previous captions and not a single caption . This is a more accurate algorithm in making a sentence to form some sense.

```
[ ]  #TODO: Beam Search
     def beam_search_predictions(image, beam_index = 3):
         start = [wordtoix["startseq"]]

         start_word = [[start, 0.0]]

         while len(start_word[0][0]) < max_len:
             temp = []
             for s in start_word:
                 par_caps = sequence.pad_sequences([s[0]], maxlen=max_len, padding='post')
                 e = encoding_test[image[len(images):]]
                 preds = final_model.predict([np.array([e]), np.array(par_caps)])

                 word_preds = np.argsort(preds[0])[-beam_index:]

                 # Getting the top <beam_index>(n) predictions and creating a
                 # new list so as to put them via the model again
                 for w in word_preds:
                     next_cap, prob = s[0][:], s[1]
                     next_cap.append(w)
                     prob += preds[0][w]
                     temp.append([next_cap, prob])

             start_word = temp
             # Sorting according to the probabilities
             start_word = sorted(start_word, reverse=False, key=lambda l: l[1])
             # Getting the top words
             start_word = start_word[-beam_index:]

         start_word = start_word[-1][0]
         intermediate_caption = [idx2word[i] for i in start_word]

         final_caption = []

         for i in intermediate_caption:
             if i != 'endseq':
                 final_caption.append(i)
             else:
                 break

         final_caption = ' '.join(final_caption[1:])
         return final_caption
```

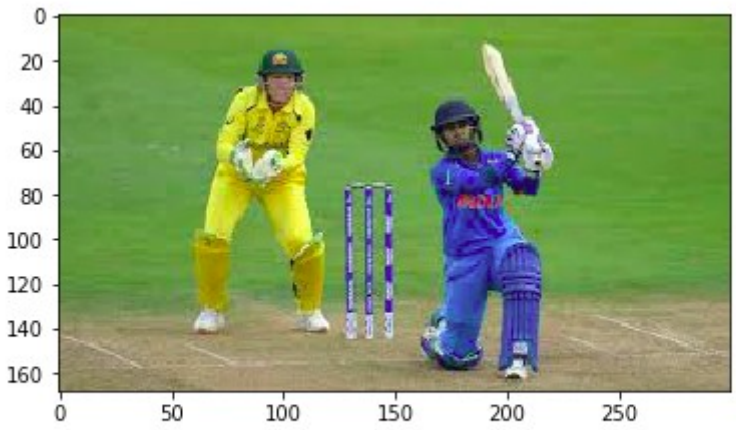
14) Evaluation

To understand how good the model is, let’s try to generate captions on images from the test dataset.




```
Greedy: baby in blue shirt is playing in the bed
```


```
Greedy: motorcyclist is riding an orange motorcycle
```



Greedy: two boys play on the grass

Greedy: black dog is running through the water

We have also printed the ***BLEU scores*** for the images which tells us the accuracy of the images . The BLEU score predicts results on the basis of n-grams . We have used n=1,n=2,n=3,n=4.

+ Code + Text

BLEU Score

```
[ ] from nltk.translate.bleu_score import sentence_bleu
```

```
[ ] # reference = [['this', 'is', 'a', 'test'], ['this', 'is' 'test']]
    # reference[0]
    # candidate = ['this', 'is', 'a', 'test']
    # score = sentence_bleu(reference, candidate)
    # print(score)
```

```
['this', 'is', 'a', 'test']
```

```
[ ] Candidate = Prediction.split()
    Candidate
```

```
['baby', 'in', 'blue', 'shirt', 'is', 'playing', 'in', 'the', 'bed']
```

```
[ ] Reference = [x.split() for x in descriptions[list(encoding_test.keys())[999].split(".")[0]]]
    Reference
```

```
[['two',
  'babies',
  'look',
  'up',
  'while',
  'they',
  'are',
  'playing',
  'in',
  'playpen',
  'with',
  'lot',
  'of',
  'balls'],
 ['two',
  'boys',
  'in',
  'tentlike',
  'area',
  'with',
  'colorful',
  'balls',
  'on',
  'floor'],
 ['two', 'children', 'look', 'up'],
 ['two', 'kids', 'are', 'in', 'pen', 'playing', 'with', 'colorful', 'balls'],
 ['two', 'young', 'children', 'play', 'with', 'colored', 'balls']]
```

```
[ ] score0 = sentence_bleu(Reference, Candidate)
    score1 = sentence_bleu(Reference, Candidate,weights=(1, 0, 0, 0))
    score2 = sentence_bleu(Reference, Candidate,weights=(0, 1, 0, 0))
    score3 = sentence_bleu(Reference, Candidate,weights=(0, 0, 1, 0))
    score4 = sentence_bleu(Reference, Candidate,weights=(0, 0, 0, 1))
    print(score0)
    print(score1)
    print(score2)
    print(score3)
    print(score4)
```

```
0.408248290463863
0.2222222222222222
0.12500000000000003
1.0
1.0
```

15) Conclusion

Due to the stochastic nature of the models, the captions generated by you (if you try to replicate the code) may not be exactly similar to those

generated in my case. Of course this is just a first-cut solution and a lot of modifications can be made to improve this solution like:

- Using a larger data set.
- Changing the model architecture, like 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**.

भारतीय प्रौद्योगिकी संस्थान जम्मू
INDIAN INSTITUTE OF TECHNOLOGY JAMMU
विद्याधनं सर्वधन प्रधानम्

ZERO SHOT LEARNING

Zero-Shot learning method aims to solve a task without receiving any example of that task at training phase. The task of recognizing an object from a given image where there weren't any example images of that object during training phase can be considered as an example of Zero-Shot Learning task. Actually, it simply allows us to recognize objects we have not seen before.

While reading about the image captioning, we observed that zero shot learning has not been used till now in the process of image captioning. So, we decided to use the zero-shot learning in our image caption generating model.

We came up with a real-world example where our model can be used. In conventional object recognition process, it is necessary to determine a certain number of object classes in order to be able to do object recognition with high success rate. It is also necessary to collect as many sample images as possible for selected object classes. Of course, these sample images should contain objects taken from diverse angles in various contexts/environments in order to be comprehensive. Although there exists lots of object classes that we can effortlessly gather sample images of, there also exists cases that we are not always so lucky for example we want to recognize animals that are on the edge of extinction or live in extreme environment.

This **Ili pika** was seen last summer in China's Tianshan Mountains

The image captioning model with zero shot can be used to recognize the endangered species as we don't have much photos of these species to train simple image captioning model. It is where zero shot learning comes into play as it helps to recognize the object that it has not already seen in the training process.

In addition to the difficulty of recognizing different object classes with a limited number of images, labeling for some object classes is not as easy as ordinary people can do. In some cases, labeling can only be done after the subject is truly mastered or in the presence of an expert. Fine grained object recognition tasks like recognition of fish species or tree species can be considered as examples of labelling under the supervision of an expert. An ordinary person will call/label all the tree she/he is viewed as ***tree*** or all the fish she/he is viewed as ***fish***. These are obviously true answers but imagine that you want to train a network in order to recognize tree or fish species. In that case, all true answers are useless, and you need an expert

to help you with labelling task. Again, you need to make a lot of effort to achieve that.

The research paper titled *Fine-Grained Object Recognition and Zero-Shot Learning in Remote Sensing Imagery* [1] is one of the interesting practical studies about the subject where, in the paper, trees are recognized and classified to species only using their aerial or satellite images which are hard to make sense of but easy to collect when compared to walking around a huge area to take the pictures of trees and label them.

Different datasets such as CUB dataset, Sun dataset are available where this difficult task of labeling is done along with the images. This helps us in training our zero-shot model.So, we build a simple zero shot learning model which we decided to integrate with our image captioning model latter. We decided to train our model for different types of bird species. We use the CUB dataset for the purpose of training our model. The complete description of the dataset and our methodology is discussed latter.

APPROCH FOR BUILDING MODEL OF ZERO SHOT LEARNING

Let us consider an example, a child would have no problem recognizing a zebra if it has seen a horse before and read somewhere that a zebra looks like a horse but has black-and-white stripes.

According to a research paper[2], the reason why humans can perform ZSL is because of their existing language knowledge base, which provides a high-level description of a new or unseen class (zebra) and makes a connection between it and seen classes and visual concepts (horse, stripes). Inspired by this human's ability, there is an increasing interest in machine ZSL for scaling up visual recognition.

Research paper[2], further explains, zero-shot learning also relies on the existence of a labelled training set of seen classes and unseen class. Both seen and unseen classes are related in a high dimensional vector space, called semantic space, where the knowledge from seen classes can be transferred to unseen classes.

So, two steps are involved in implementation of ZSL (Zero shot learning):

- A joint embedding space is learned where both the semantic vectors (prototypes) and the visual feature vectors can be projected to.

- Nearest neighbour (NN) search is performed in this embedding space to match the projection of an image feature vector against that of an unseen class prototype.

So, in zero shot learning we will make two embeddings space one will be the image embedding and second embedding we can get from the attributes and the labels that we will have for our images.

Image embedding is nothing special. It is a feature vector extracted from an image using a convolutional network. Convolutional network can be implemented from scratch or a pre-trained convolutional network that had already proven its success, can be used.

We collect image samples for training classes and naturally, we can get image embeddings for all these image samples. However, we don't have any image sample for zero-shot classes, and it is not possible to get image embeddings for zero-shot classes. At this point, we need another data representation which will function as a bridge between training and zero-shot classes. This data representation should be extracted from all data samples ignoring that they belong to training classes or zero-shot classes. Because of that, instead of focusing image itself, we should focus on labels and attributes which is a common property for all data samples.

It is a *representation which we can easily access for each class of objects* beside their image representations. We can use Google's *Word2Vecs* which will allow us to represent words as vectors. In Word2Vec space, two vectors are most likely to be positioned closely if two words — represented with two mentioned vectors — tend to be appear together in same documents or have semantic relations. Another research paper [3] has claimed to describe an extremely simple approach for ZSL that is able to outperform by a significant margin the current state of the art approaches on a standard collection of ZSL datasets.

We have tried to implement the approach that the research paper “An embarrassingly simple approach to zero-shot learning."[3] have described. The methods used are described in the next section.

Dataset used:

We follow setting provided by [4] for training/test splits. Under the old setting, adopted by most existing ZSL works before [4], some of the test classes also appear in the ImageNet 1K classes, which have been used to pretrain the image embedding network, thus violating the zero-shot assumption. In contrast, the new GBU setting ensures that none of the test classes of the datasets appear in the ImageNet 1K classes.

We have used the CUB dataset which is dataset of 200 different categories of birds with their images and annotation. The Caltech-UCSD Birds-200-2011 Dataset [5] is the paper about the complete detail if the CUB dataset.

The important features of this dataset are

- **Number of categories:** 200
- **Number of images:** 11,788
- **Annotations per image:** 15 Part Locations, 312 Binary Attributes, 1 Bounding Box

Attributes: A vocabulary of 28 attribute groupings and 312 binary attributes (e.g., the attribute group belly color contains 15 different color choices) are selected. All attributes are visual in nature, with most pertaining to a color, pattern, or shape of a particular part.

Part Locations: A total of 15 parts were annotated by pixel location and visibility in each image.

In the data split provided in [4] consists of:

- allclasses.txt: list of names of all classes in the dataset
- trainvalclasses.txt: seen classes
- testclasses.txt: unseen classes
- trainclasses1/2/3.txt: 3 different subsets of trainvalclasses used for tuning the hyperparameters
- valclasses1/2/3.txt: 3 different subsets of trainvalclasses used for tuning the hyperparameters

resNet101.mat includes the following fields:

- features: columns correspond to image instances
- labels: label number of a class is its row number in allclasses.txt
- image files: image sources

att_splits.mat includes the following fields:

- att: columns correspond to class attribute vectors normalized to have unit l2 norm, following the classes order in allclasses.txt
- original_att: the original class attribute vectors without normalization
- trainval_loc: instances indexes of train+val set features (for only seen classes) in resNet101.mat
- test_seen_loc: instances indexes of test set features for seen classes
- test_unseen_loc: instances indexes of test set features for unseen classes

- train_loc: instances indexes of train set features (subset of trainval_loc)
- val_loc: instances indexes of val set features (subset of trainval_loc)

In [4] there are both proposed splits (PS) and resNet101 features of CUB, SUN, AWA1, AWA2 and APY datasets. We use CUB dataset split and resNet 101 features for our model. Other dataset can also be used with a slight change in the code, i.e. we only have to change the name of dataset from “CUB” to any other like “SUN” or “AWA1”.

Extracting Image features:

To extract the image features we used the pretrained resNET101 features given along with the splits.

resNET101:

According to the author of paper Deep Residual Learning for Image Recognition [6]

“Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers---8x deeper than VGG nets but still having lower complexity.”

ResNet-101 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 101 layers deep and can classify images into 1000 object categories, such as

keyboard, mouse, pencil, and many animals. As a result, the network has learned rich feature representations for a wide range of images. The network has an image input size of 224-by-224.

A pre-trained model has been previously trained on a dataset and contains the weights and biases that represent the features of whichever dataset it was trained on. Learned features are often transferable to different data.

From the .mat files extract all the features from resnet and the attribute splits.

- The res101 contains features and the corresponding labels.
- att_splits contain the different splits for trainval, train, val and test set.

res101.keys():

dict_keys (['__header__', '__version__', '__globals__', 'image_files', 'features', 'labels'])

att_splits.key ():

dict_keys(['__header__', '__version__', '__globals__', 'allclasses_names', 'att', 'original_att', 'test_seen_loc', 'test_unseen_loc', 'train_loc', 'trainval_loc', 'val_loc'])

Labels/Classes:

We need the corresponding ground-truth labels/classes for each training example for all our train, val, trainval and test set according to the split locations provided. In this example we

have used the CUB dataset which has 200 unique classes overall.

We got the labels from

```
labels = res101['labels']
```

and now we can divide it into different test, train and validation according to the given splits.

Size of train validation and test set:

Let us denote the features X ∈ [d×m] available at training stage, where d is the dimensionality of the data, and m is the number of instances. We are using resnet features which are

extracted from CUB dataset.

```
X_features = res101['features']
```

and again, these features can be divided into test and training according to the splits. After dividing the features following are the sizes:

Features for train: (2048, 5875)

Features for val: (2048, 2946)

Features for trainval: (2048, 7057)

Features for test: (2048, 2967)

Signature Matrix:

Each of the classes in the dataset have an attribute (a) description. This vector is known as

the Signature matrix of dimension S ∈ [0, 1] a×z. For training stage there are z classes and z' classes for test S ∈ [0, 1] a×z'.

```
signature = att_splits['att']
```

This is a signature matrix, where the occurrence of an attribute corresponding to the class is given. For instance, if the classes

are horse and zebra and the corresponding attributes are [wild animal, 4_legged, carnivore]

Horse Zebra

[0.00354613 0.] Domestic animal

[0.13829921 0.20209503] 4_legged

[0.06560347 0.04155225] carnivore

Signature for train: (312, 100)

Signature for val: (312, 50)

Signature for trainval: (312, 150)

Signature for test: (312, 50).

Using the feature vector and signature matrix which are the attributes represented in form of matrix we will first find the weights using train and validation set and then use these weights to do predictions on our test set.

Proposed Solution:

The one-line code solution proposed.

V = inverse (XX' + γI) XYS' inverse (SS' + λI) where

X: feature vector obtained from resNET101

X’: Transpose of X

I: identity matrix

Y: ground truth-- The ground truth is a one-hot encoded vector

S: Signature matrix

γ, λ: hyperparameters

V: weights that will be used to do prediction on test set.

We calculate V from train and val set.

Using V, we can make predictions on test set as:

argmax (X'VS) where

X: feature vector of test set,

V: weights as calculated above from the train and val set,

S: signature matrix of the test set.

We can tweak the hyperparameters and check for which set we can get the best

accuracy.

We have written a small script that iterates over for different sets of alpha and gamma

and gives us the best set.

The set for which we found the highest accuracy is:

Alpha = 3, Gamma= -1

Results and Conclusion:

We got an accuracy of 51.3% from our model. Although this is not much but for a very

simple model like we think it has done quite well.

Further Improvements and Scope:

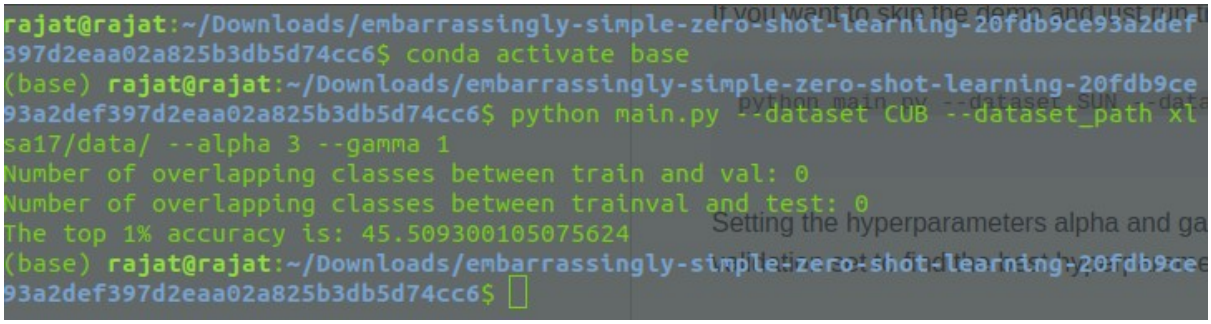
In this model we have used the resNET101 model for our image feature extraction.

There is a scope of trying different neural network architecture like inception, VGG and then the best architecture can be used.

Secondly, as we have the features and attributes in order to map them and getting better

result, we may use different algorithms like K-nearest neighbors and can try to increase the

accuracy of the model.



Uses:

- With a better accuracy model of Zero shot learning as discussed earlier we can use it with an image caption generating model which can provide us a solution to keep track of the endangered species.
- Zero shot learning can also be used in the autonomous driving where it is very important to detect the objects.

Conclusion

In the first part we made image captioning model in which we used beam search on Flickr8k data set .In the second part we have designed zero shot learning model using CUB data set . Since zero shot learning is never combined with image captioning , we propose to combine the above two models to work in a way such that our model outputs the required class in our caption . This model can be used in many real life problems like in case of endangered species . We can deploy this model and it will predict the class of the species along with caption.

References:

[1] Gencer Sumbul, Ramazan Gokberk Cinbis, Selim Aksoy**:** Fine-Grained Object Recognition and Zero-Shot Learning in Remote Sensing Imagery.

[2] Li Zhang, Tao Xiang, Shaogang Gong: LearningaDeepEmbeddingModelforZero-ShotLearning.

[3] Bernardino Romera-Paredes, Philip H. S. Torr: An embarrassingly simple approach to zero-shot learning.

[4] Yongquin Xian**,** Christoph H. Lampert, Bernt Schiele, Zeynep Akata: Zero-Shot Learning – A comprehensive evaluation of the good, bad and the Ugly.

[5] Catherine Wah, Steve Branson, Peter Welinder, Pietro Perona, Serge Belongie: The Caltech-UCSD Birds-200-2011 Dataset.

[6] Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun: Deep Residual Learning for Image Recognition.

[7] https://cs.stanford.edu/people/karpathy/cvpr2015.pdf

[8] https://towardsdatascience.com/image-captioning-with-keras-teaching-computers-to-describe-pictures-c88a46a311b8

[9] https://arxiv.org/abs/1411.4555

[10] https://arxiv.org/abs/1703.09137

[11] https://arxiv.org/abs/1708.02043

[12] https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/

[13] https://www.appliedaicourse.com/