Attention Mechanism For Image Caption Generation

MIS-64061-002
Advanced Machine Learning

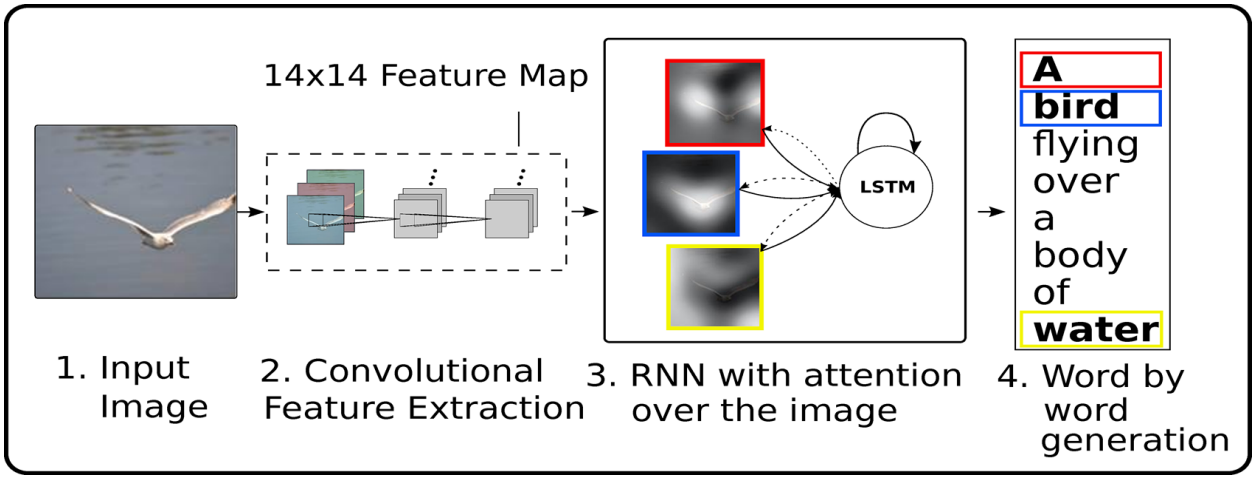
Abhishek Sau

Introduction and Theory

Attention mechanism is a complex cognitive ability that human beings possess. When people receive information, they can consciously focus on some of the main information while ignoring other secondary information. This ability of human beings to see some part of the images or textual information in high resolution and blur out the rest is known as attention.

Image caption generator is a model which generates caption based on the features present in the input image. In this project, we tend to amalgamate Image caption generation and attention mechanism to output caption for input images.

The below mentioned figure shows the architecture used in this project.

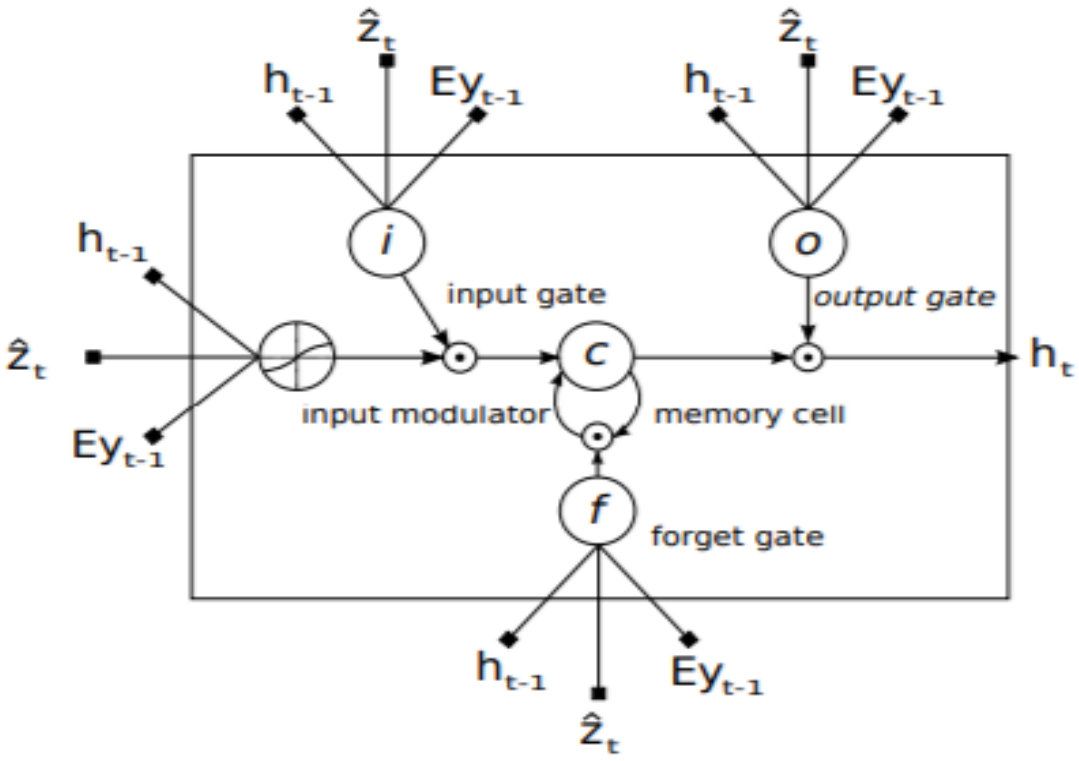


The model takes a raw image as its input, the images are then processed using a convolutional neural network through three features. In an encoder decoder model this CNN acts as an encoder.

The CNN used in this project uses the VGG architecture. VGG architecture processes an image through sequential convolutional layers followed by Max pooling and finally outputs a vector that captures the meaning of the image.

In order to obtain a correspondence between the feature vectors and the portions of the 2D image, we extract features from a lower convolutional layer instead of using a fully connected layer. We take a 14X14X512 feature map. And by flattering each one of the fourteen by 14 matrices in the feature map we turn them into a set of vectors which we refer to as annotation vectors. The extractor produces L vectors each of which is a D-dimensional representation corresponding to part of the image. By concatenating these annotation vectors we generate matrix A, which is the output of the CNN encoder, which is then used by the attention model to determine which sections of the image are more relevant in generating the next word in the image caption generation task.

The decoder in this project is RNN using the LSTM architecture. LSTMs were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information. The core concept of LSTM is the cell state, and it's various gates. Consider the cell state acts as a transport highway that helps transfer the relative info to the sequence chain. It acts as the Memory of the network. The cell state can carry relevant information throughout the processing of the sequence. The information from the earlier time steps can be carried to the time steps which helps diminish the short-term memory. The cell state transforms when information gets added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training.

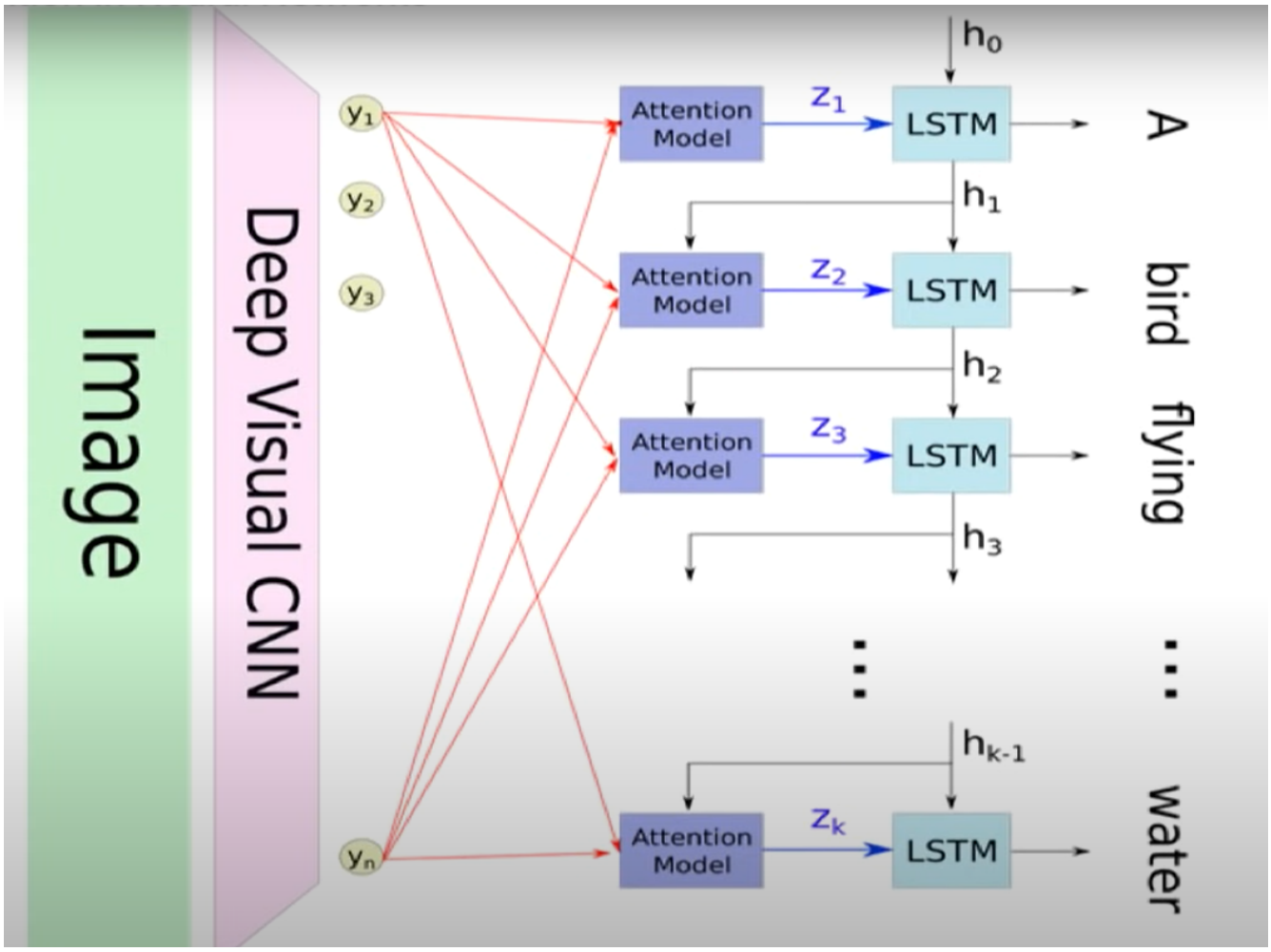


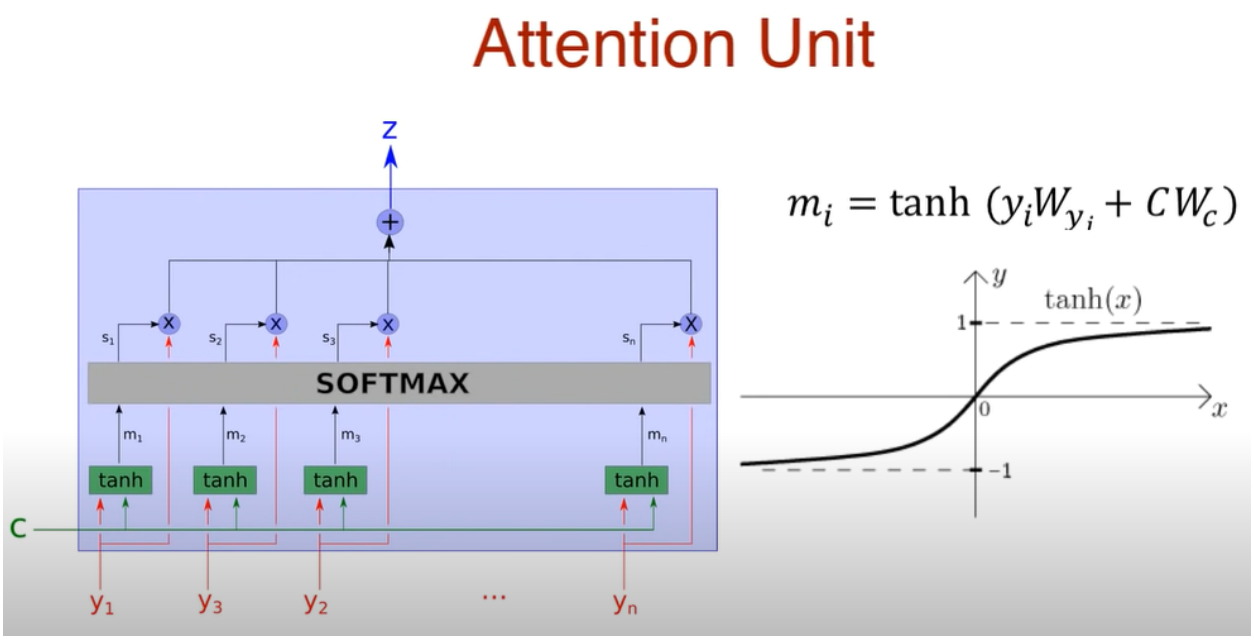
The importance of this LSTM model is the previously generated word y_t which is multiplied by an embedding matrix E, previous decoder hidden states h_{t-1} and the context vectors Z_t which is a dynamic representation of the relevant part of the image input at time t.

The different gates and their functions are mentioned below:

1. **Forget gate**: This gate decides what information should be removed or retained. Information from the preceeding hidden state and information from the ongoing input is passed through the sigmoid function where the values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.

2. **Input Gate: Input gates are used to update the cell state.** We pass the preceding hidden state and Ongoing input into a sigmoid function which decides what values will be updated by transforming the values to be between 0 and 1. You also pass the hidden state and current input into the tanh function to make its values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.
3. **Output Gate:**The output gate determines what the next hidden state should be. The hidden state also can be used for predictions.We pass the previous hidden state and the current input into a sigmoid function. Then the newly modified cell state is passed through a tanh function and multiplied with the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.





Dataset

The dataset used here is the **FLICKR 8K** which consists of around 8091 images along with 5 captions for each image. I have used Google colab pro to run the code since it needs high computational power.

The code can be found here.

Dependencies

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

Steps to follow:

1. **Importing the required libraries, the image dataset along with Captions and Preprocessing**

```
The number of jpg files in Flicker8k: 8118
```

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

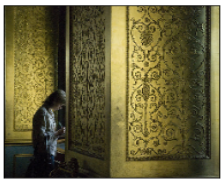
	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	a child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	1	a girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	2	a little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	3	a little girl climbing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

Plotting few images and their captions from the dataset

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

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

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



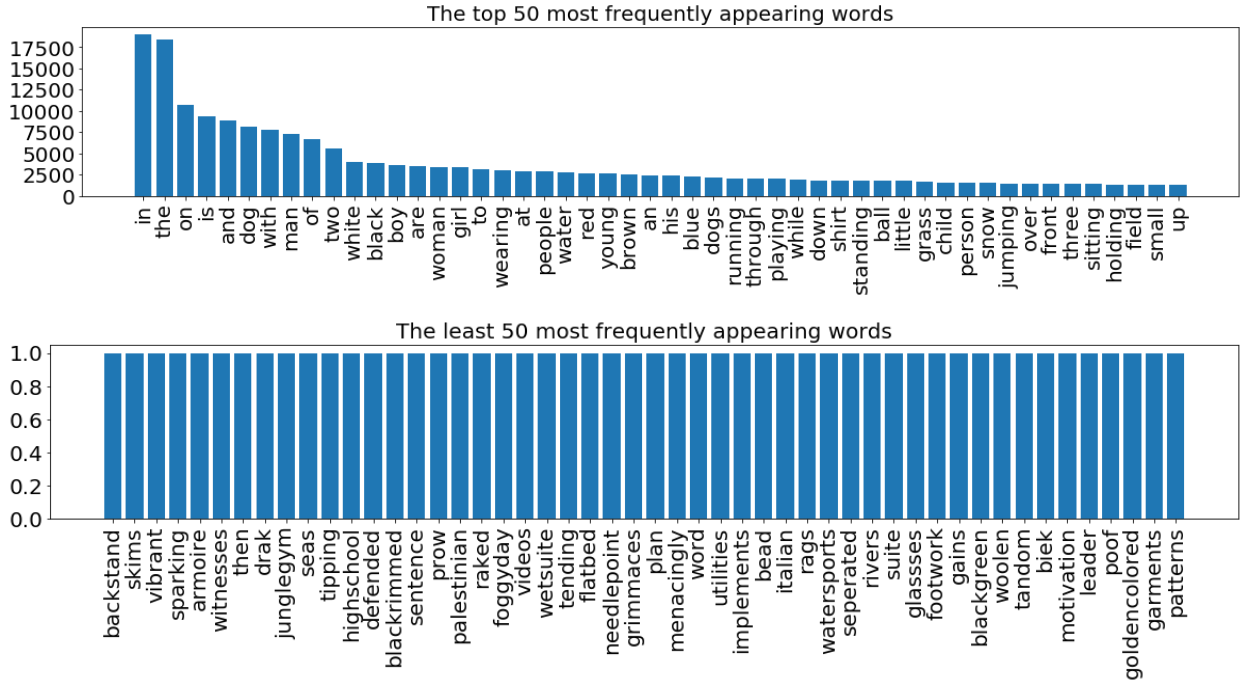
woman writing on a pad in room with gold , decorated walls .
the walls are covered in gold and patterns .
a woman standing near a decorated wall writes .
a woman behind a scrolled wall is writing
a person stands near golden walls .

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

Cleaning the captions for further analysis & processing

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

Plotting the top 50 words that appear in the cleaned dataset



We save all the captions and image paths in two lists so that we can load the images at once using the path set. We also add ‘< start >’ and ‘< end >’ tags to every caption so that the model understands the starting and end of each caption.

```
[ ]  PATH = "/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/"
     all_captions = []
     for caption  in data["caption"].astype(str):
        caption = '<start> ' + caption+ ' <end>'
        all_captions.append(caption)

     all_captions[:10]
```

```
['<start>  child in pink dress is climbing up set of stairs in an entry way <end>',
 '<start>  girl going into wooden building <end>',
 '<start>  little girl climbing into wooden playhouse <end>',
 '<start>  little girl climbing the stairs to her playhouse <end>',
 '<start>  little girl in pink dress going into wooden cabin <end>',
 '<start>  black dog and spotted dog are fighting <end>',
 '<start>  black dog and dog playing with each other on the road <end>',
 '<start>  black dog and white dog with brown spots are staring at each other in the street <end>',
 '<start>  two dogs of different breeds looking at each other on the road <end>',
 '<start>  two dogs on pavement moving toward each other <end>']
```

```
[ ]  all_img_name_vector = []
     for annot in data["filename"]:
        full_image_path = PATH + annot
        all_img_name_vector.append(full_image_path)

     all_img_name_vector[:10]
```

```
['/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1000268201_693b08cb0e.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1001773457_577c3a7d70.jpg',
 '/content/drive/MyDrive/Flickr8k_Dataset/Flicker8k_Dataset/1001773457_577c3a7d70.jpg']
```

```
print(f"len(all_img_name_vector) : {len(all_img_name_vector)}")
print(f"len(all_captions) : {len(all_captions)}")
```

```
len(all_img_name_vector) : 40455
len(all_captions) : 40455
```

We will take only 40000 of each so that we can select batch size properly i.e. 625 batches if batch size= 64. To do this we define a function to limit the dataset to 40000 images and captions.

2. Model Definition :

Let's define the image feature extraction model using VGG 16. We must remember that we do not need to classify the images here, we only need to extract an image vector for our images. Hence we remove the softmax layer from the model.

We extract the features and store them in the respective .npy files and then pass those features through the encoder.NPY files store all the information required to reconstruct an array on any computer, which includes dtype and shape information.

Next, we tokenize the captions and build a vocabulary of all the unique words in the data. We will also limit the vocabulary size to the top 5000 words to save memory. We will replace words not in vocabulary with the token < unk

```
top_k = 5000
tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=top_k,
                                                  oov_token="<unk>",
                                                  filters='!"#$%&()*+.,-/:;=?@[\]^_`{|}~ ')

tokenizer.fit_on_texts(train_captions)
train_seqs = tokenizer.texts_to_sequences(train_captions)
tokenizer.word_index['<pad>'] = 0
tokenizer.index_word[0] = '<pad>'

train_seqs = tokenizer.texts_to_sequences(train_captions)
cap_vector = tf.keras.preprocessing.sequence.pad_sequences(train_seqs, padding='post')
```

Next, Create training and validation sets using an 80-20 split:

```
img_name_train, img_name_val, cap_train, cap_val = train_test_split(img_name_vector,cap_vector, test_size=0.2, random_state=0)
```

Next, let's create a tf.data dataset to use for training our model.

```
def map_func(img_name, cap):
   img_tensor = np.load(img_name.decode('utf-8')+'.npy')
   return img_tensor, cap

dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))
dataset = dataset.map(lambda item1, item2: tf.numpy_function(map_func, [item1, item2], [tf.float32, tf.int32]),num_parallel_calls=tf.data.experimental.AUTOTUNE)
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.experimental.AUTOTUNE)
```

The encoder-decoder architecture with attention. The architecture defined in this article is similar to the one described in the paper "Show and Tell: A Neural Image Caption Generator"

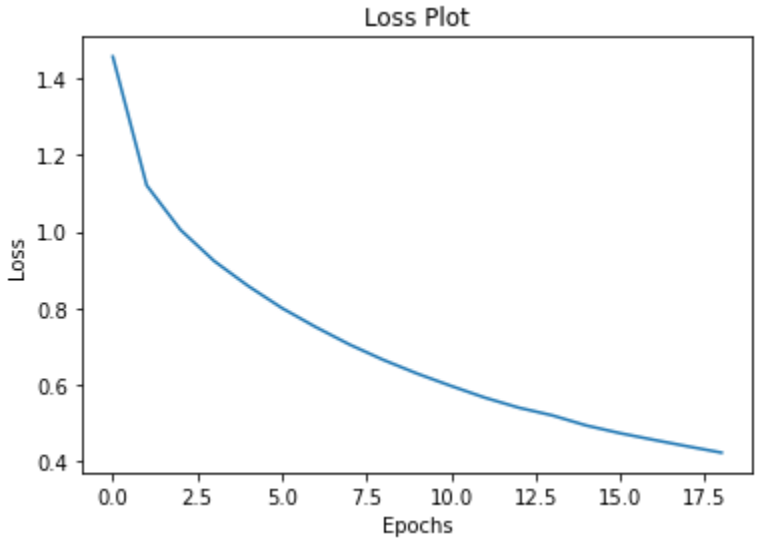
```
[32] class VGG16_Encoder(tf.keras.Model):
        # This encoder passes the features through a Fully connected layer
        def __init__(self, embedding_dim):
            super(VGG16_Encoder, self).__init__()
            # shape after fc == (batch_size, 49, embedding_dim)
            self.fc = tf.keras.layers.Dense(embedding_dim)
            self.dropout = tf.keras.layers.Dropout(0.5, noise_shape=None, seed=None)

        def call(self, x):
            #x= self.dropout(x)
            x = self.fc(x)
            x = tf.nn.relu(x)
            return x
```

3. Model Training

We use a technique called Teacher Forcing.In this technique, the target word is passed as the next input to the decoder. It helps in learning the correct sequence or correct statistical properties for the sequence in a fast manner.

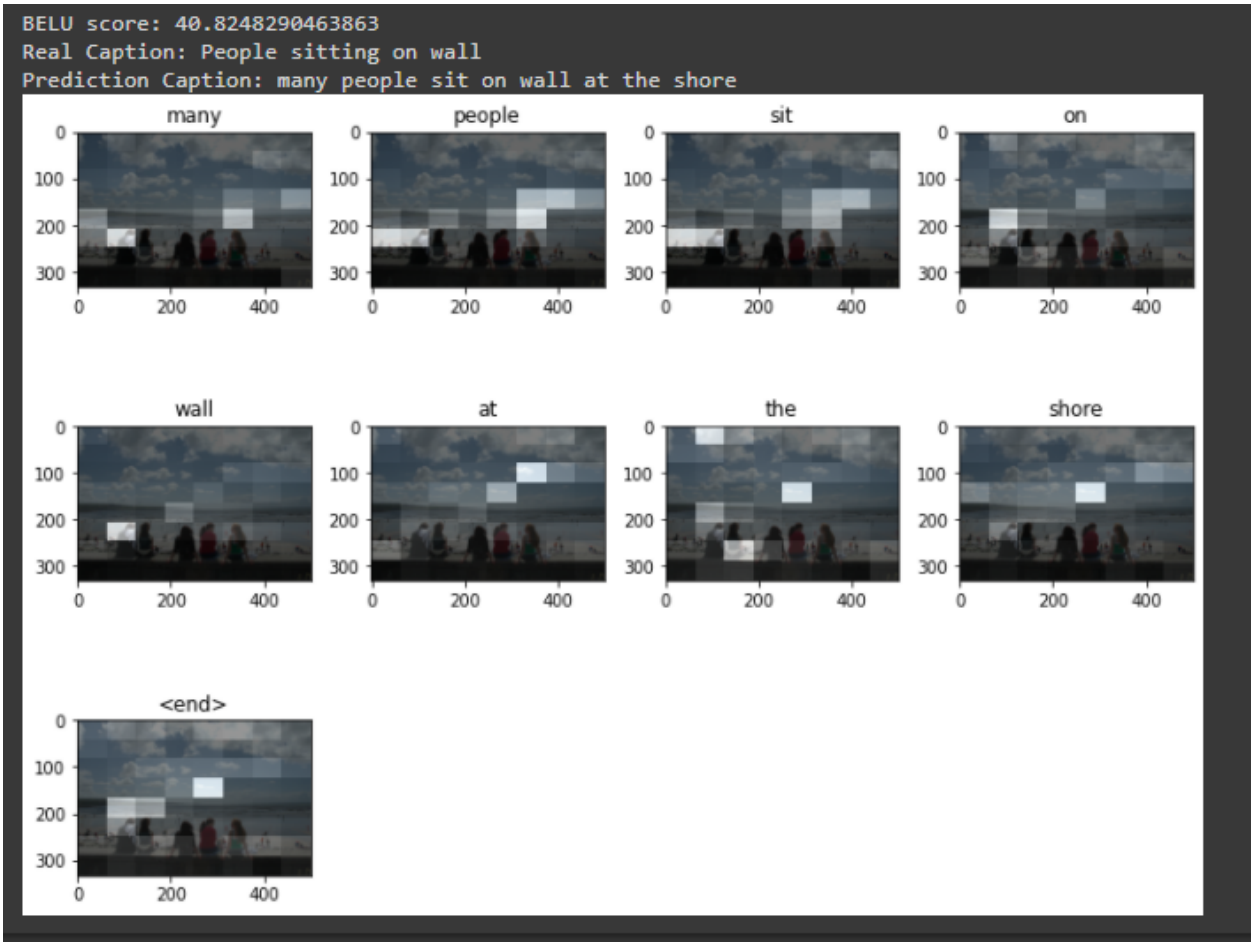
Then we train the model with 20 epochs and plot the error graph.



4. BLEU Evaluation

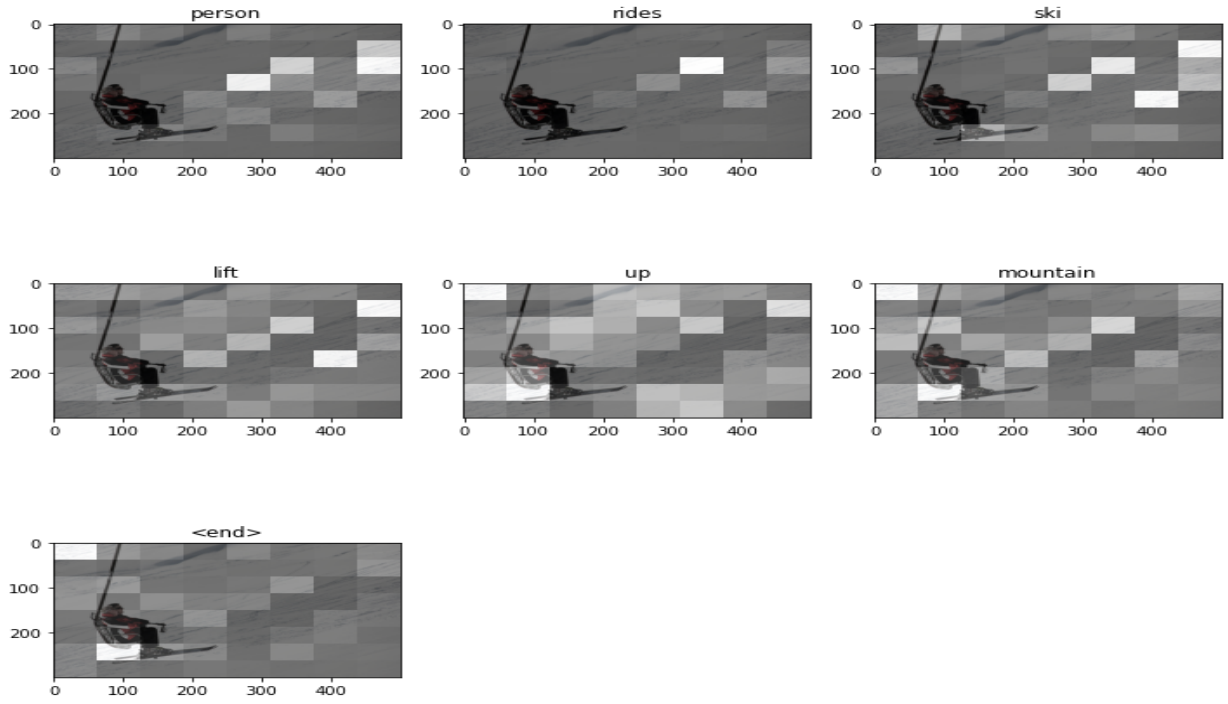
Now we will try to generate a caption for the image at the start of the article and see what the attention mechanism focuses on and generates.

In the below mentioned image the caption very similar to the real caption and suits the image. You can see even though our caption is quite different from the real caption, it is still very accurate.

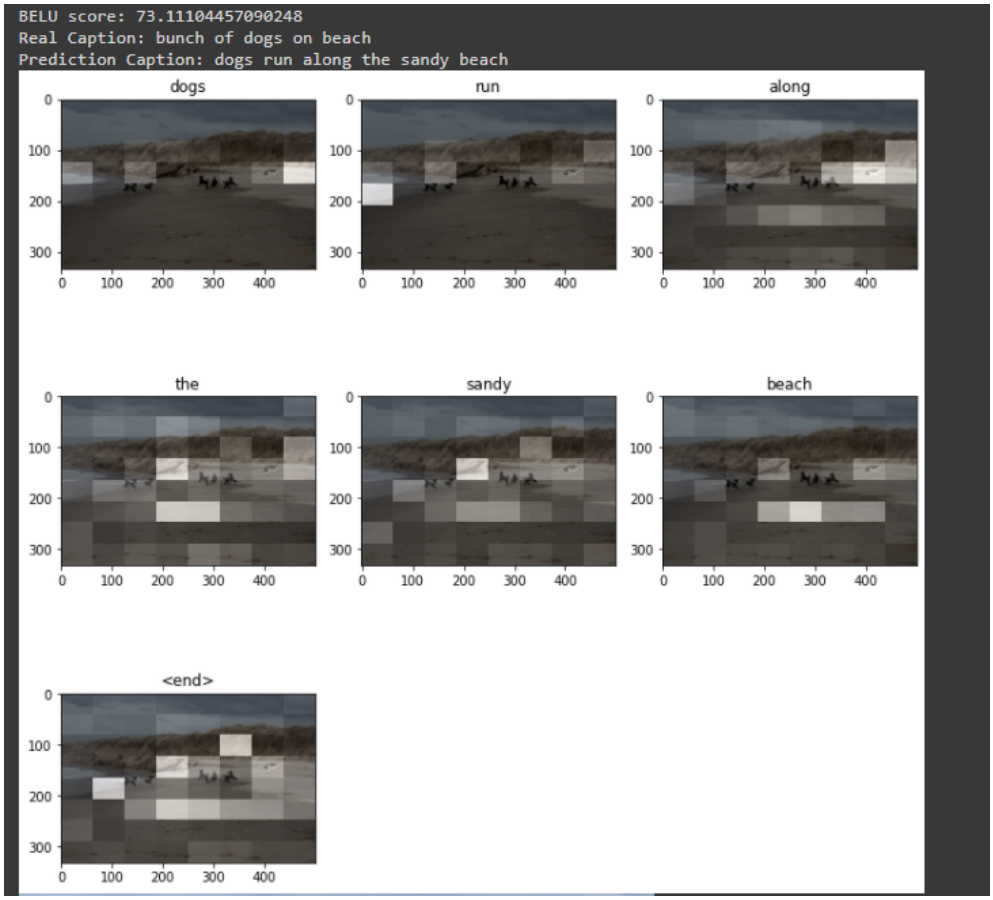


Now We will try to test it with another two pictures.

Picture number 1 :



Picture Number 2:



Hence, by the above two pictures I can conclude that the image caption generator is working pretty well with the attention mechanism implemented.

References:

1. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A.C., Salakhutdinov, R., Zemel, R.S., & Bengio, Y. (2015). Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *ICML*.
2. https://www.analyticsvidhya.com/blog/2017/12/fundamentals-of-deep-learning-introduction-to-lstm/
3. https://www.analyticsvidhya.com/blog/2020/11/attention-mechanism-for-caption-generation/
4. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21
5. https://www.analyticsvidhya.com/blog/2019/11/comprehensive-guide-attention-mechanism-deep-learning/