IMAGE CAPTION GENERATION

Team:

Venkata Sai Naveena Bathula	(A20451804)
Jesleen Sonia Pradeep Kamalesh	(A20448891)
Bhanuja Arekatla	(A20449753)
Venkata Manikanta Monic Kamisetty	(A20446683)

Illinois Institute of Technology
CSP571-Data Preparation and Analysis
Professor: Jawahar Panchal

ABSTRACT

In recent years, with the rapid growth of artificial intelligence, the caption of images has steadily attracted the attention of many artificial intelligence researchers and has become a fascinating and arduous mission. A significant part of scene comprehension, which incorporates the knowledge of computer vision and natural language processing is image captioning which automatically produces natural language explanations according to the content observed in an image. The application of the image caption is extensive and significant, for example, the realization of human-computer interaction. We are performing the task of image captioning which is composed of Image-based model and a Language-based model on the Flickr8k dataset. These models are responsible for extracting the features out of a given image and translates the features and objects provided by the image-based model to a natural sentence. Furthermore, the advantages and the shortcomings of these approaches are addressed, providing the widely used datasets and assessment criteria.

OVERVIEW

Introduction:

Image Captioning refers to the process of generating a textual description from a given image based on the objects and actions in the image. The task of providing a natural language description of the content within an image lies at the intersection of computer vision and natural language processing. On the computer vision side, improved convolutional neural network and object detection architectures have contributed to improved image captioning systems. On the natural language processing side, more sophisticated sequential models, such as attention-based recurrent neural networks, have similarly resulted in more accurate caption generation. Inspired by neural machine translation, most conventional image captioning systems utilize an encoder-decoder framework, in which an input image is encoded into an intermediate representation of the information contained within the image, and subsequently decoded into a descriptive text sequence. The generation of captions from images has various practical benefits, ranging from aiding visually impaired, to enabling the automatic and cost-saving labeling of the millions of images uploaded to the Internet every day.

Proposed methodology:

The image captioning task consists of two logical models, namely an image-based model and a language-based model. The former is responsible for extracting the characteristics from a given image, while the latter converts the image-based model's characteristics and objects into a natural sentence.

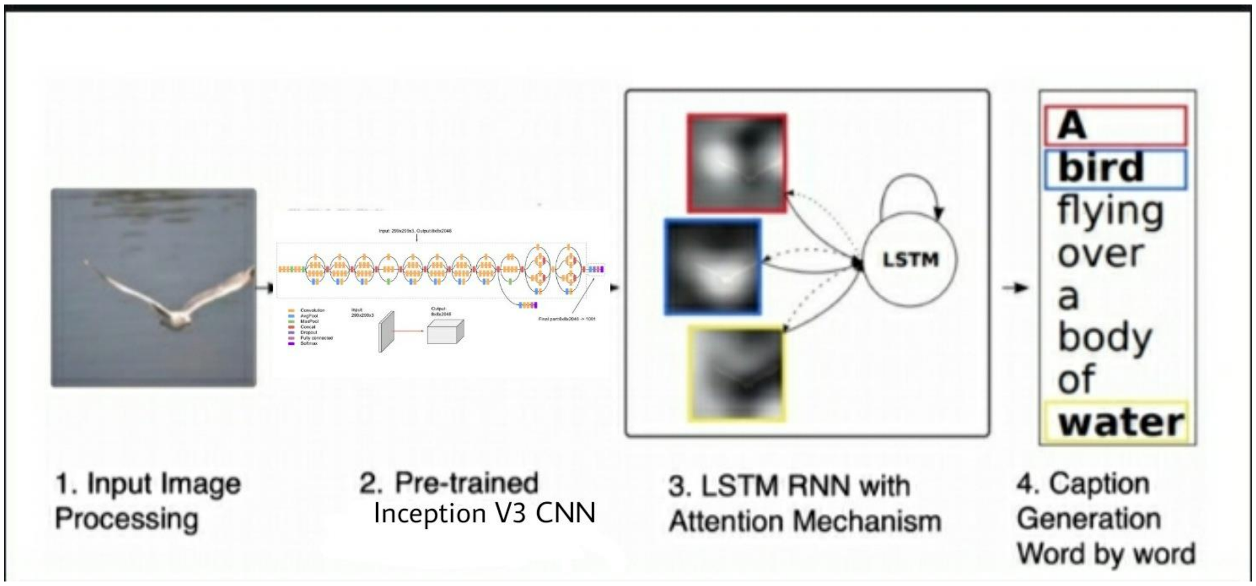
The Convolutionary Neural Network (CNN) does the work related to image processing and feature extraction, while the Recurrent Neural Network (RNN) produces the necessary textual description (captions) using the features learned by CNN. This overall model/architecture is usually referred to as the model Encoder-Decoder.

The Encoder is the image-based model (CNN) here while the Decoder is the language-based model (RNN). An Encoder-Decoder model maps a fixed-length input with a fixed-length output in a nutshell, and the input and output length may/may not vary. The last layer of the CNN model, in general, is a Dense/Fully-connected layer that has a number of neurons equal to the number of target classes possible. For a multi-class classification problem, this layer has a softmax activation function, where the expected class is a probability allocated to each of the classes and the class with the highest probability value. We are not involved in classifying images and need only CNN's rich encoding of the image (learned features). These learned characteristics are then transmitted to the decoder (RNN language generation) to generate the appropriate captions. The features are extracted from the image by a pre-trained CNN model (by dropping the last layer-Full connected/Dense layer). Then the function vector is transformed linearly to have the same dimension as the LSTM(RNN) network's input dimension. This network on the feature vector is trained as a language model.

Therefore, to train a model which is capable of captioning images, we need a dataset with a large number of images along with the corresponding caption (s).

Architecture:

We have written data preprocessing scripts to process raw input data (both images and captions) into proper format; A pre-trained Convolutional Neural Network architecture as an encoder to extract and encode image features into a higher dimensional vector space; An LSTM-based Recurrent Neural Network as a decoder to convert encoded features to natural language descriptions; Attention mechanism which allows the decoder to see features from a specifically highlighted region of the input image to improve the overall performance; Beam Search to figure out a caption with the highest likelihood.


LSTM
A
bird
flying
over
a
body
of
water
1. Input Image Processing
2. Pre-trained Inception V3 CNN
3. LSTM RNN with Attention Mechanism
4. Caption Generation Word by word

DATA

Data Properties:

A new benchmark set of 8,000 images for sentence-based image definition and search, each coupled with five separate captions that provide concise explanations of the salient entities and events. For this mission, we present a strong baseline that combines image-text embedding, common object detectors, a color classifier, and a bias towards selecting larger objects. Although our baseline rivals more complex state-of-the-art models in precision, we show that its gains can not easily be parlayed into enhancements on tasks such as image-sentence retrieval, illustrating the shortcomings of current methods and the need for further study.

In the Computer Vision research domain, we identified the three most widely used image caption training datasets - COCO dataset[6], Flickr8k[3], and Flickr30k[8]. These datasets include annotated photos of 123,000, 31,000 and 8,000 captions, respectively, and each picture is labeled with 5 distinct details. At present, because of our limited storage and computational power ,we have found the Flickr8k dataset

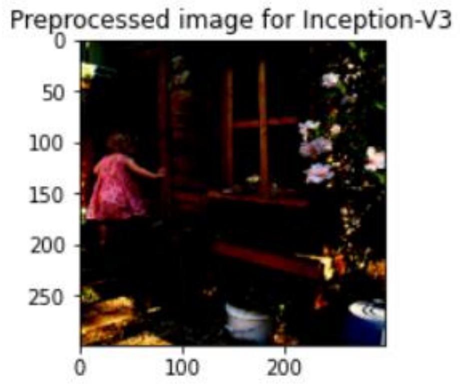
The Flickr-8k dataset comprises 8000 images from the Flickr website and can be found here. There are 6000 training images , 1000 images for validation and 1000 testing images. There are 5 captions explaining each image. For the images, these captions serve as labels. For the objects contained within an image, there is no class information.

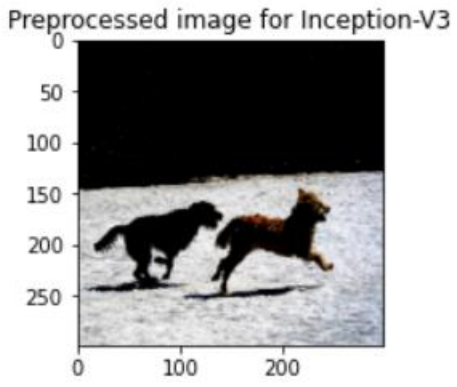
Data Preprocessing:

For images as well as names, we conduct data cleaning. Label cleaning is achieved by converting all words to lowercase, deleting stop words and punctuations for improved text processing. Photos and captions consist of input data. We need, therefore to process CNN network for processing images and RNN network to process text captions. As the pipeline of the image caption generator leverages the pre-trained state of the CNN network, we need to turn the images into the correct format.

We will first transform images into fixed size vectors for image prediction and we will perform analysis on these vectors. We will encode each and every word into a fixed size vector in order to predict captions word by word. A final matrix is generated using these vectors.

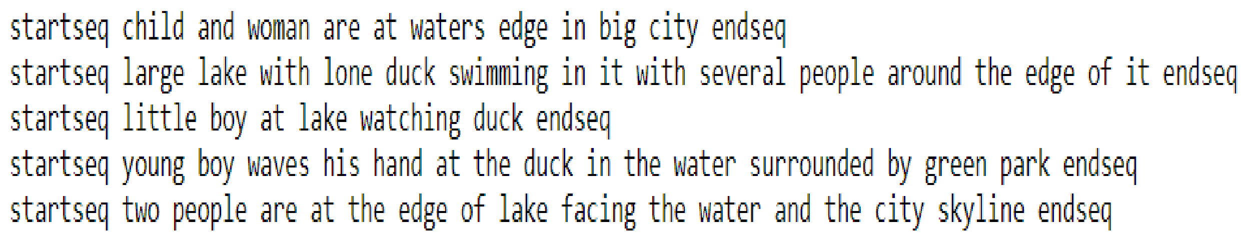
We will be using Inception V3 model in order to classify every image. We will be extracting features from the last layer of convolutional layer. The steps needed to be followed by Inception V3 in order to convert into images.





A caption given to a picture will be generated by the model we will create, and the caption will be generated one word at a time. As input, the sequence of words previously generated will be given. We would also need a 'first word' to set off the process of generation and a last word' to signify the end of the caption.

For this process, we'll use the 'startseq' and 'endseq' strings. When they are loaded, these tokens are added to the loaded details. Before we encrypt the text so that the tokens are also correctly encoded, it is necessary to do this now.



```
startseq child and woman are at waters edge in big city endseq
startseq large lake with lone duck swimming in it with several people around the edge of it endseq
startseq little boy at lake watching duck endseq
startseq young boy waves his hand at the duck in the water surrounded by green park endseq
startseq two people are at the edge of lake facing the water and the city skyline endseq
```

DATA ANALYSIS

Feature extraction:

The input of the model is a single raw image and the output is a caption y encoded as a sequence of 1-of-K encoded words. The extractor produces L vectors and each element corresponds to a part of the image as a D-dimensional representation. The feature vectors will be extracted from the convolutional layer before the fully connected layer. We will try different layers such as convolutional layers to compare the result and try to choose the best layers to produce feature vectors that contains most precise information about relationship between salient features and clusters in the image.

For the encoding part of our encoder-decoder network, we will make use of InceptionV3 to extract image features. In principle, which features to extract is up to experimentation, - here we just use the last layer before the fully connected top.

For an image size of 299x299, the output will be of size (batch_size, 8, 8, 2048), that is, we are making use of 2048 feature maps.

Using the preprocess_input method, the images are preprocessed to normalize the image so that it contains pixels in the -1 to 1 range that fit the image format which used to train InceptionV3.

We will now build the tf.keras model where the output layer in the InceptionV3 architecture is the last convolutionary layer. The output form of this layer is 8*8*2048.

You forward each image through the network and store the resulting vector in a dictionary

(image_name --> feature_vector).

InceptionV3 being a “big model”, where every pass through the model takes time, we will precompute features in advance and store them on disk. We’ll use tfdatasets to stream images to the model. This means all our preprocessing has to employ tensorflow functions:

Caption generation:

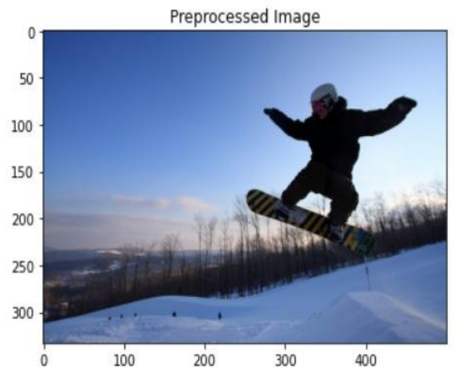
The model uses a network of long short-term memory (LSTM) that generates a caption. We will generate one word based on a context vector at each time level, the previous hidden state and the words previously generated. The multi-model pipeline proposed the groundbreaking use of neural networks for image caption creation, which showed that neural networks could decode image representations from a CNN encoder, and which also showed that the resulting hidden dimensions and word embeddings contained semantic meaning.

Caption generation is a challenging problem of artificial intelligence in which a textual explanation for a given photograph must be produced. It involves both computer vision methods to understand the image content and a language model from the natural language processing field to transform the image's understanding into words in the correct order.

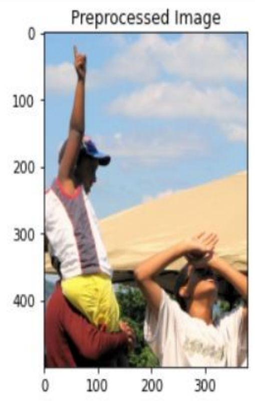
Using a pre-trained Convolutional Neural Network (ENCODER) that would generate a hidden state h, a "classic" image captioning system would encode the image.

This hidden state would then be decoded by using an LSTM(DECODER) and each caption word would be generated recursively. So, we will be combining these architectures to make our image caption generator model. It's also called a model for CNN-RNN.

CNN is used for extracting features from the image. We will use the pre-trained model Inception V3. LSTM will use the information from CNN to help generate a description of the image.



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Greedy: man in red jacket is snowboarding down snowy hill
```



```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Greedy: two men lift their arms as they are sitting on the ground
```

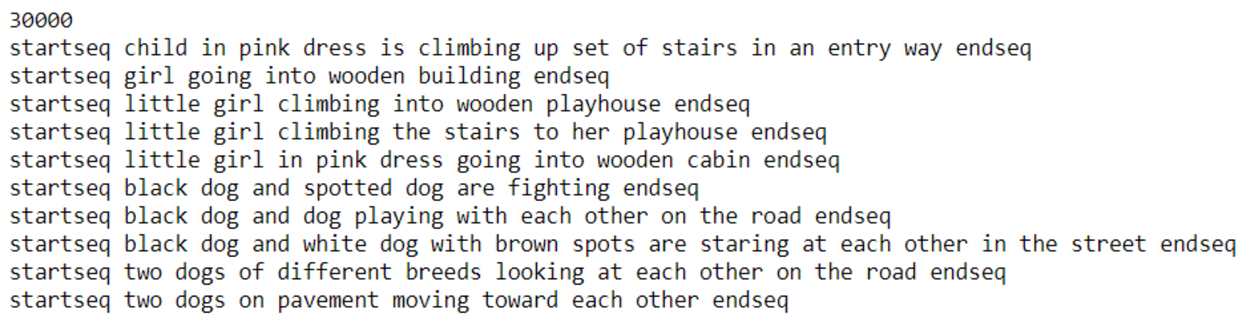
Loss function:

We use a word-wise cross entropy as the basic loss function. At this point, we know that we cannot directly feed words into an LSTM and expect it to be able to train or produce the correct output. These words first must be turned into a numerical representation so that a network can use normal loss functions and optimizers to calculate how "close" a predicted word and ground truth word (from a known, training caption) are? So, we typically turn a sequence of words into a sequence of numerical values; a vector of numbers where each number maps to a specific word in our vocabulary.

MODEL TRAINING

Feature engineering:

The Convolutional Neural Network (CNN) does the image processing and feature extraction related work while on the other hand Recurrent Neural Network (RNN) subsequently generates the required textual description (caption) using the features learned by CNN. This overall feature/architecture is commonly termed as Encoder-Decoder model. Most of the images correctly state what objects appear in the scene, count number of appearance and gives an intuitively correct verb to logically complete the sentence. Colors of the object and spatial relationships between object are also well captured. For example, the bottom right image has a caption " A man in a red shirt is standing on a grassy hill". Objects in the images are identified as "A Man", "Shirt" and "Grassy hill". Then CNN is also able to capture properties of objects such as shirt's color is "red". Lastly, RNN and Attention Mechanism will connect these words logically by conjunctions into a valid sentence.



```
30000
startseq child in pink dress is climbing up set of stairs in an entry way endseq
startseq girl going into wooden building endseq
startseq little girl climbing into wooden playhouse endseq
startseq little girl climbing the stairs to her playhouse endseq
startseq little girl in pink dress going into wooden cabin endseq
startseq black dog and spotted dog are fighting endseq
startseq black dog and dog playing with each other on the road endseq
startseq black dog and white dog with brown spots are staring at each other in the street endseq
startseq two dogs of different breeds looking at each other on the road endseq
startseq two dogs on pavement moving toward each other endseq
```

Training Captions

Evaluation metrics:

Qualitative:

We looked at model performance as one of the factors. This includes the size of the network, the rate at which the network learns, how early it plateaus and how resource intensive the model is.

Quantitative:

To quantifiably measure the accuracy of the models, we will be using the BLEU (bilingual evaluation understudy) score and the METEOR. BLEU was originally developed to assess performance with language translation problems. The primary task for a BLEU implementer is to compare n-grams of the candidate with the n-grams of the reference translation and count the number of matches. These matches are position-independent. The more the matches, the better the candidate translation is. The Flickr-8k dataset after preprocessing, provides data in the form of a dictionary where, the key is an image and the value for that image is a set of 5 captions. The BLEU metric is used to compare the predicted caption from the model to all the given caption labels. The best match from all these labels is picked as the final BLEU score.

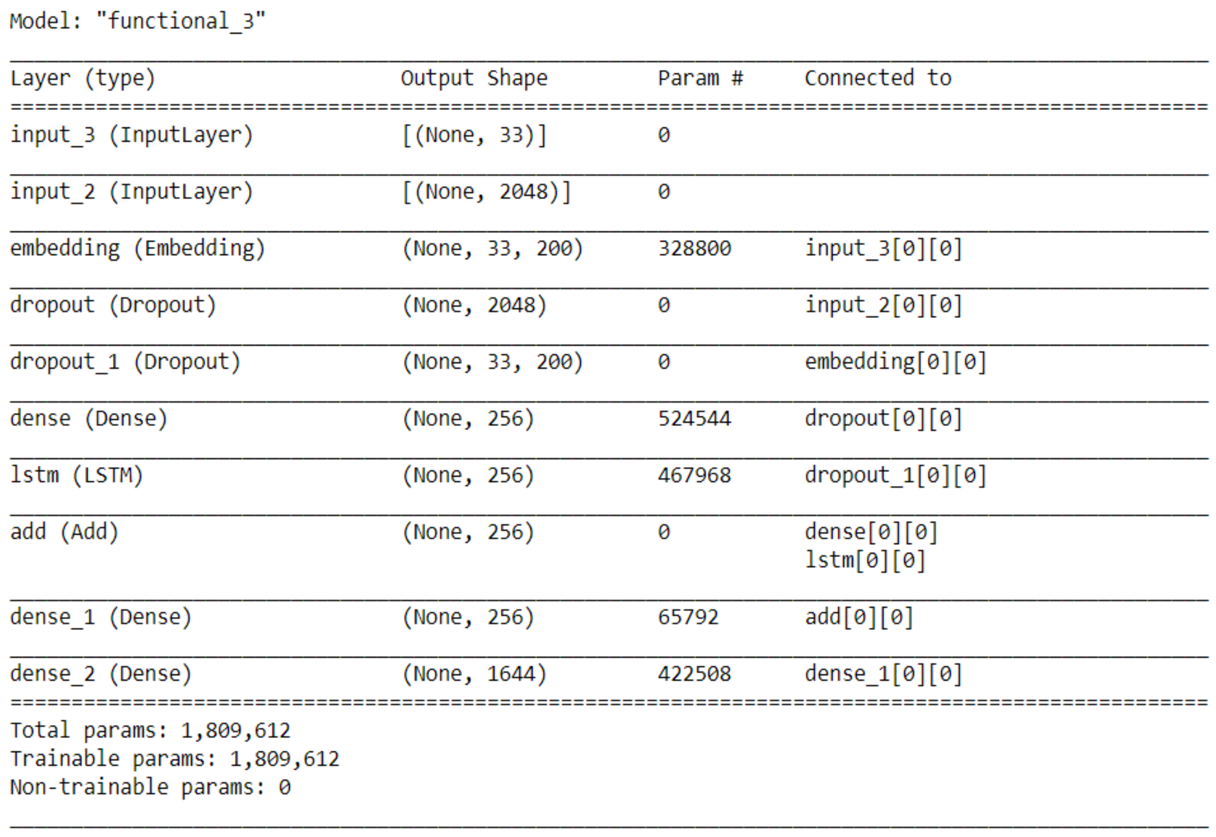
Model selection:

To extract the features from the image, we are considering a pre-trained Inception V3 model. On the ImageNet dataset, which can be directly imported from Keras.applications based on results, the Inception V3 model has been trained.

The focus of the ImageNet Project is to mark and categorize images. Szegedy et al proposed Inception V3 model that has a CNN-based architecture and has contributed to a new state of the art for classification and detection. The model's main feature is its architecture, which has enhanced the usage of computing resources. The design accomplishes this by allowing the model to have increased depth and width. There are smaller weights for Inception V3

When we actually dig deeper into the layers of these networks, we find that each layer is somehow assigned with extracting unique features during training. Therefore in one network, we have a stack of function extractors. In this tutorial, we will use Inception V3, a powerful model that Google has built as our feature extractor.

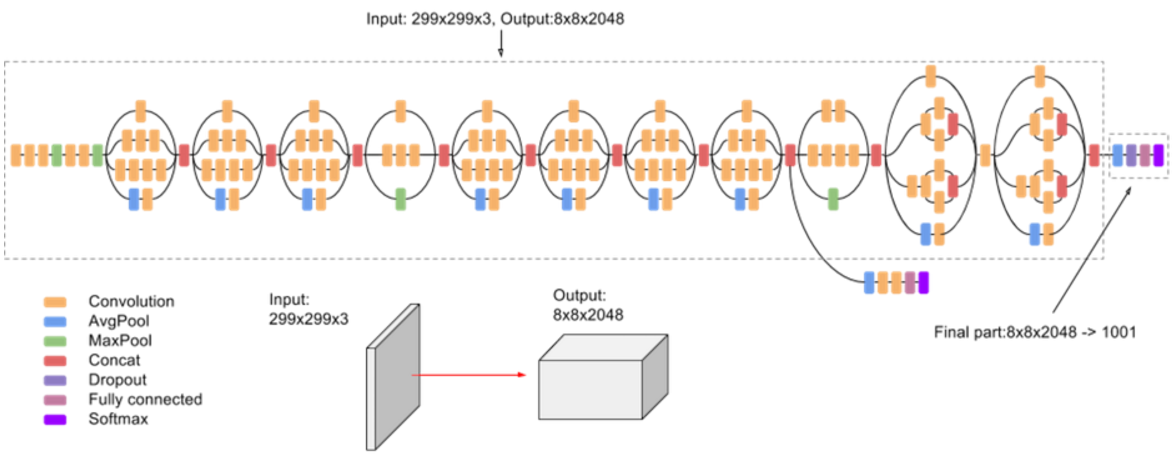
- In this project, we have extracted the features from the lower convolutional layer of model InceptionV3 that has a vector of shape (8, 8, 2048).
- Then we reduce to a shape of (64, 2048).
- The CNN Encoder then passes this vector through the (which consists of a single Fully connected layer).
- In order to predict the next word, the RNN (here GRU) goes over the image.
- We need a way of streaming them to our captioning model now that we have taken care of pre-extracting the features and preprocessing the captions.



```
Model: "functional_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 33)]         0
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 33, 200)      328800      input_3[0][0]
__________________________________________________________________________________________________
dropout (Dropout)               (None, 2048)         0           input_2[0][0]
__________________________________________________________________________________________________
dropout_1 (Dropout)             (None, 33, 200)      0           embedding[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 256)          524544      dropout[0][0]
__________________________________________________________________________________________________
lstm (LSTM)                     (None, 256)          467968      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, 1644)         422508      dense_1[0][0]
==================================================================================================
Total params: 1,809,612
Trainable params: 1,809,612
Non-trainable params: 0
__________________________________________________________________________________________________
```

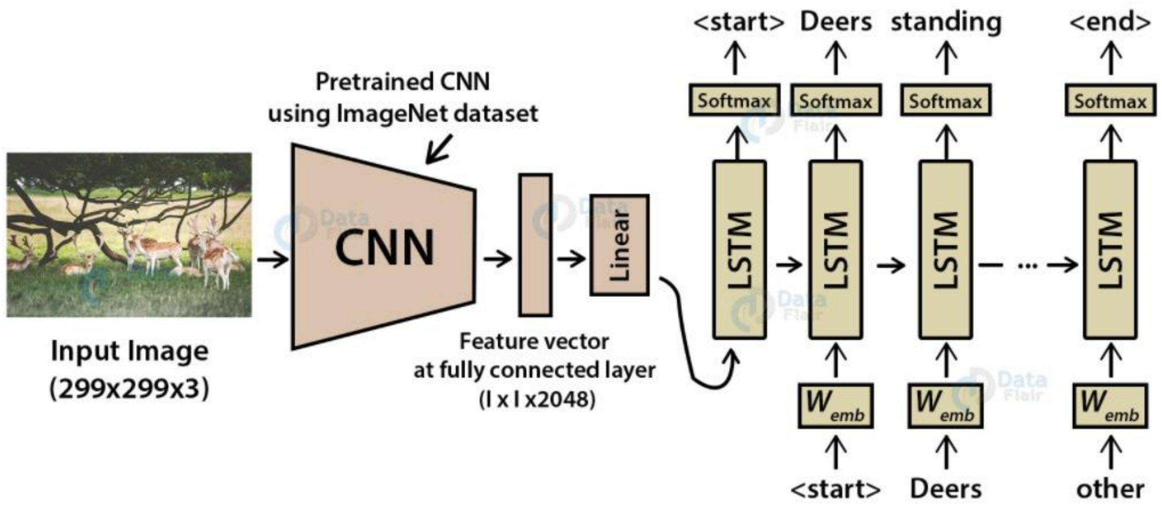
Architecture:

The Inception module is designed as a “multi-level feature extractor” which is implemented by computing 1×1, 3×3, and 5×5 convolutions within the same module of the network. The network is built in such a way that the result obtained from the convolutions is being stacked along the channel dimension and then fed into layer in the network. Figure 2 depicts the architecture of the InceptionV3 model.



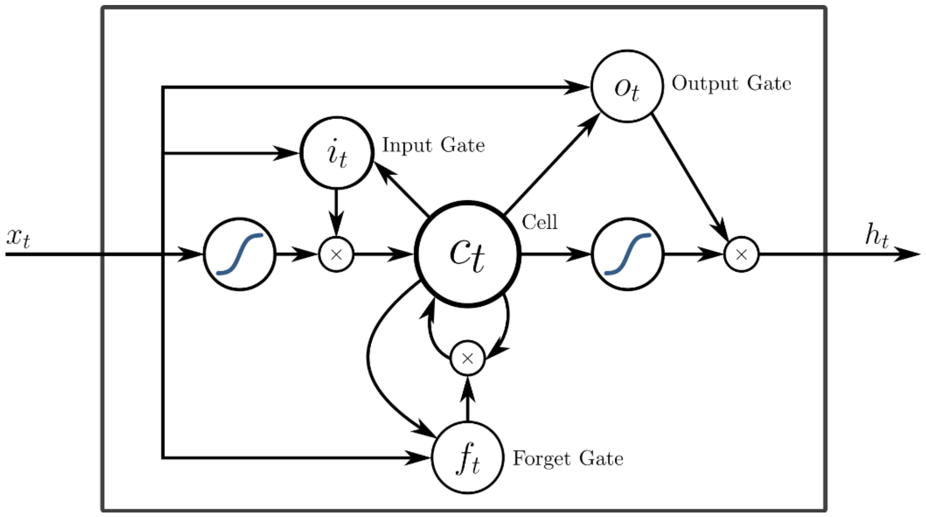
CONVOLUTIONAL NEURAL NETWORKS(CNN) :

Convolutional Neural networks [10] are specialized deep neural networks which processes the data that has input shape like a 2D matrix. CNN works well with images and are easily represented as a 2D matrix. Image classification and identification can be easily done using CNN. It can determine whether an image is a bird, a plane or Superman, etc. [7] Important features of an image can be extracted by scanning the image from left to right and top to bottom and finally the features are combined together to classify images. It can deal with the images that have been translated, rotated, scaled and changes in perspective.



LSTM (RNN):

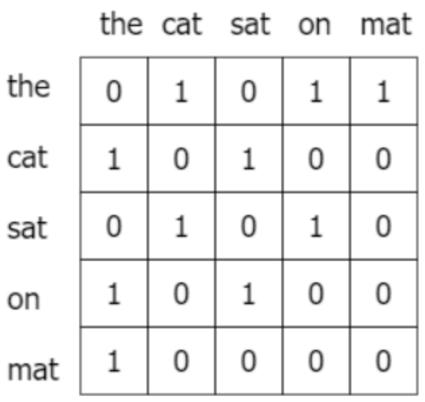
Long short-term memory (LSTM) units are a building unit for layers of a recurrent neural network (RNN). An RNN composed of LSTM units is often called an LSTM network. A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The cell is responsible for “remembering” values over arbitrary time intervals; hence the word “memory” in LSTM. Each of the three gates can be thought of as a “conventional” artificial neuron, as in a multi-layer (or feedforward) neural network: that is, they compute an activation (using an activation function) of a weighted sum. Intuitively, they can be thought as regulators of the flow of values that goes through the connections of the LSTM; hence the denotation “gate”. There are connections between these gates and the cell. This is detailed in the following figure .



Glove dataset

GloVe stands for “Global Vectors”. And as mentioned earlier, GloVe captures both global statistics and local statistics of a corpus, in order to come up with word vectors. Turns out, it each type of statistic has their own advantage. For example, Word2vec which captures local statistics do very well in analogy tasks. However a method like LSA which uses only global statistics does not do that well in analogy tasks. However since Word2vec method suffers from certain limitations (like what we discussed above) due to using local statistics only. Considering the corpus having V words, the co-occurrence matrix X will be a

V *x* V matrix, where the i th row and j th column of *X*, X_ij denotes how many times word ***i*** has co-occurred with word ***j***. An example co-occurrence matrix might look as follows.



	the	cat	sat	on	mat
the	0	1	0	1	1
cat	1	0	1	0	0
sat	0	1	0	1	0
on	1	0	1	0	0
mat	1	0	0	0	0

MODEL VALIDATION

Testing results:

Unlike with other machine learning validation pipelines, in this case we're not going to validate our model against a certain metric. Instead, we're going to validate our model based on whether it's generating the correct captions and, most importantly, whether it's paying attention to the correct features when generating those captions. We can achieve this by overlaying the attention matrix weights generated when producing the caption for a particular image on the image itself. This produces an image with some spots which indicate what the network was paying attention to when generating the caption.

Greedy Approach: This 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.

Encoder:

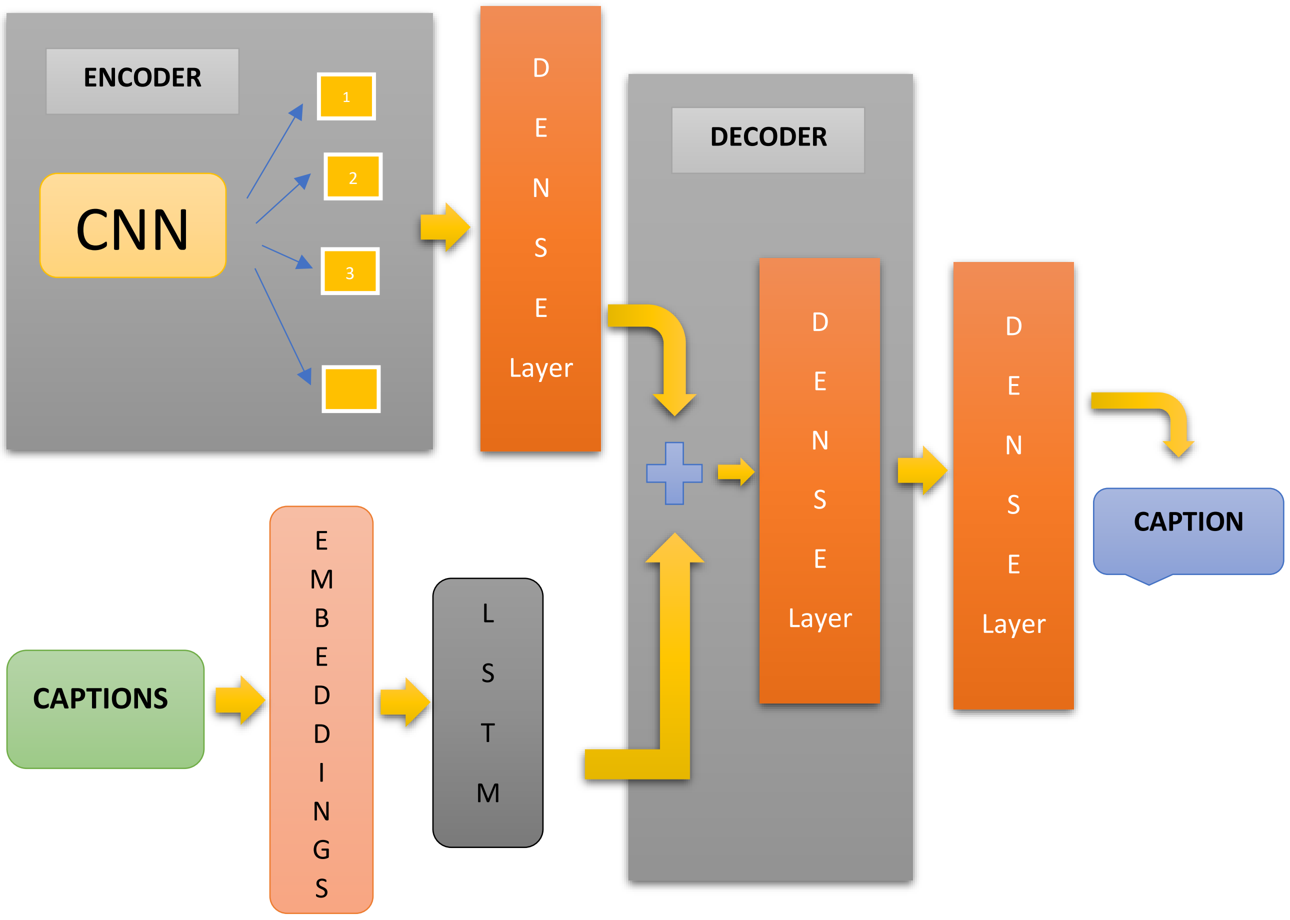
We use convolutional neural network to generate feature maps. By using Inception V3, we collect 2048 feature maps for each image. The encoder in this case is just a fully connected layer, taking in the features extracted from Inception V3 (in flattened form, as they were written to disk), and embedding them in 256-dimensional space.

Attention module:

The attention module is separated out into its own custom model. With an Attention mechanism, the image is first divided into n parts, and we compute with a Convolutional Neural Network (CNN) representations of each part h1,…, hn. When the RNN is generating a new word, the attention mechanism is focusing on the relevant part of the image, so the decoder only uses specific parts of the image. We can recognize the figure of the "classic" model for image captioning, but with a new layer of attention model. What is happening when we want to predict the new word of the caption? If we have predicted i words, the hidden state of the LSTM is hi. We select the « relevant » part of the image by using hi as the context. Then, the output of the attention model zi, which is the representation of the image filtered such that only the relevant parts of the image remains, is used as an input for the LSTM. Then, the LSTM predicts a new word and returns a new hidden state hi+1.

Decoder:

By considering the caption after the construction of the model, based on the co-occurences formed in the adjancency matrix using the Glove dataset the feature vectors are constructed. The decoder at each time step calls the attention module with the features it got from the encoder and its last hidden state, and receives back an attention vector. The attention vector gets concatenated with the current input and further processed by a GRU and two fully connected layers, the last of which gives us the (unnormalized) probabilities for the next word in the caption.



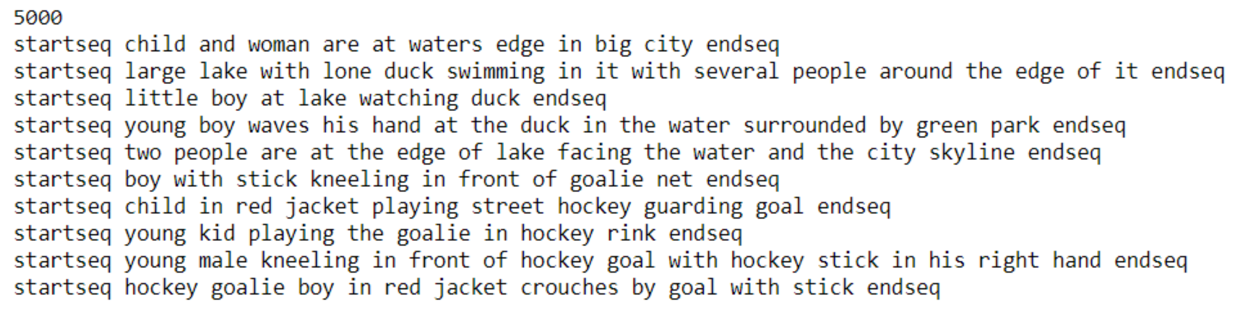
Performance criteria:

The experimental quantitative results are validated for the caption generation by using the BLEU metric and METEOR metric for evaluating the performance. We report the results with the frequently used metric BLEU which is the standard metric for the caption generation literature with various scores from 1 to 4 without the brevity penalty for BLEU, but due to criticism on BLEU, another common metric METEOR is considered for the comparison performance.

Bleu- Output is the geometric mean of n-gram score with a brevity penalty to discourage shorter translation. Meteor: It is based on an explicit word-to-word matching between the MT output being evaluated and one or more reference translations. It can also match synonyms.

The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text.

BLEU scores are used in text translation for evaluating translated text against one or more reference translations which is the geometric mean of n-gram score.



```
5000
startseq child and woman are at waters edge in big city endseq
startseq large lake with lone duck swimming in it with several people around the edge of it endseq
startseq little boy at lake watching duck endseq
startseq young boy waves his hand at the duck in the water surrounded by green park endseq
startseq two people are at the edge of lake facing the water and the city skyline endseq
startseq boy with stick kneeling in front of goalie net endseq
startseq child in red jacket playing street hockey guarding goal endseq
startseq young kid playing the goalie in hockey rink endseq
startseq young male kneeling in front of hockey goal with hockey stick in his right hand endseq
startseq hockey goalie boy in red jacket crouches by goal with stick endseq
```

Validating Captions

CONCLUSION AND FUTURE WORK:

An image caption generator has been developed using a CNN-RNN model. Some key aspects about our project to note are that our model depends on the data, so, it cannot predict the words that are out of its vocabulary. A dataset consisting of 8000 images is used here. But for production-level models i.e. higher accuracy models, we need to train the model on larger than 100,000 images datasets so that better accuracy models can be developed.

Automatically describing the content of an image using properly arranged English sentences is a tough challenging task, but it could is something very necessary for helping visually impaired people. Modern smartphones are able to take the photographs, which can help in taking surrounding images for visually impaired peoples. Here images as input can generate captions that can be loud enough so that visually impaired can hear, so that they can get a better sense of things present in there surrounding. Here we use a CNN model to extract features of an image. These features are then fed into a RNN or a LSTM model to generate a description of the image in grammatically correct English sentences describing the surroundings

DATA SOURCES

Links:

https://www.kaggle.com/hsankesara/flickr-image-dataset

https://nlp.stanford.edu/projects/glove/

Downloads:

Flickr 8k dataset is collected from the Kaggle dataset from the provided above link.

Data Description:

Dataset size: 1 GB

Number of images: 8000

These 8000 images are divided into three sets, which are namely-

Training set- 6000 images for training the model.
Validation set- 1000 images for assessing the model's performance while training.
Testing set- 1000 images for assessing the model's performance after training.

SOURCE CODE

# Image Captioning

### Imports

```python
import numpy as np
import keras
import matplotlib.pyplot as plt
%matplotlib inline
from time import time
from collections import Counter
from pickle import dump, load
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from numpy import array
import nltk


import warnings
warnings.filterwarnings(action='ignore')
```

### Variables

```python
path = "E:/All Docs/Masters/CSP 571 Data Prep and Analysis/Project/Data/8k/"
images_path = 'E:/All Docs/Masters/CSP 571 Data Prep and
Analysis/Project/Data/8k/Flicker8k_Dataset/'
model_path = 'E:/All Docs/Masters/CSP 571 Data Prep and Analysis/Project/Python/Image-Captioning-
master/model_weights/3/'
```

## Preprocessing

### Loading Captions into a Dictionary

```python
img_to_captions = {}
filename = 'Flickr8k.token.txt'
captions = f'{path}{filename}'
with open(captions) as file:
        captions_ = file.readlines()
for line in captions_:
    line_list = line.split()
    image_id = line_list[0].split('.')[0]
    img_cap = ' '.join(line_list[1:])
    if image_id in img_to_captions:
        img_to_captions[image_id].append(img_cap)
    else:
        img_to_captions[image_id] = [img_cap]
# Printing the first image captions
print(*img_to_captions[list(img_to_captions.keys())[0]],sep='\n')
```

```
    A child in a pink dress is climbing up a set of stairs in an entry way .
    A girl going into a wooden building .
    A little girl climbing into a wooden playhouse .
    A little girl climbing the stairs to her playhouse .
    A little girl in a pink dress going into a wooden cabin .
```

### Caption Processing

```python
# Removing the single length strings and punctuations
for key in img_to_captions.keys():
    for i in range(len(img_to_captions[key])):
        cap_list = img_to_captions[key][i].split()
        temp = []
        for j in cap_list:
            if j.isalpha() and len(j) > 1:
                temp.append(j.lower())
        img_to_captions[key][i] = ' '.join(temp)
print(*img_to_captions[list(img_to_captions.keys())[0]],sep='\n')
```

```
    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
```

### Saving the captions to a text file for future reference

```python
captions_ = []
for key, captions in img_to_captions.items():
    for cap in captions:
        captions_.append(key + " " + cap + '\n')
with open("preprocessed_captions.txt", 'w') as f:
    for i in captions_:
        f.write(i)
```

### Creating Word Corpus

```python
word_corpus = set()
for key in img_to_captions.keys():
    for cap in img_to_captions[key]:
        for word in cap.split():
            word_corpus.add(word)
print(f'The size of the Corpus is {len(word_corpus)}')
```

```

    The size of the Corpus is 8357
```

### Creating List of Train and Test Images

```python
train = 'Flickr_8k.trainImages.txt'
train_images = []
train_path = f'{path}{train}'
with open(train_path) as file:
    captions_ = file.readlines()
    for line in captions_:
        image_id = line.split('.')[0]
        train_images.append(image_id)

test = 'Flickr_8k.testImages.txt'
test_images = []
test_path = f'{path}{test}'
with open(test_path) as file:
    captions_ = file.readlines()
    for line in captions_:
        image_id = line.split('.')[0]
        test_images.append(image_id)
test = 'Flickr_8k.devImages.txt'
test_path = f'{path}{test}'
with open(test_path) as file:
    captions_ = file.readlines()
    for line in captions_:
        image_id = line.split('.')[0]
        test_images.append(image_id)
```

### Adding start and end seq for Training

```python
training_data = {}
for img in img_to_captions:
    if img in train_images:
        cap_list = []
        for cap in img_to_captions[img]:
            cap_list.append('startseq ' + cap + ' endseq')
        training_data[img] = cap_list
print(*training_data[list(training_data.keys())[0]],sep='\n')
```

```
    startseq child in pink dress is climbing up set of stairs in an entry way endseq
    startseq girl going into wooden building endseq
    startseq little girl climbing into wooden playhouse endseq
    startseq little girl climbing the stairs to her playhouse endseq
    startseq little girl in pink dress going into wooden cabin endseq
```

```python
testing_data = {}
for img in img_to_captions:
    if img in test_images:
        cap_list = []
        for cap in img_to_captions[img]:
            cap_list.append('startseq ' + cap + ' endseq')
        testing_data[img] = cap_list
print(*testing_data[list(testing_data.keys())[0]],sep='\n')
```

```
    startseq child and woman are at waters edge in big city endseq
    startseq large lake with lone duck swimming in it with several people around the edge of it endseq
    startseq little boy at lake watching duck endseq
    startseq young boy waves his hand at the duck in the water surrounded by green park endseq
    startseq two people are at the edge of lake facing the water and the city skyline endseq
```

### Preprocessing the Image

```python
def image_preprocessing(image_path):
    from keras.preprocessing import image
    reszed_image = image.load_img(image_path,target_size=(299,299))
    image_array = image.img_to_array(reszed_image)
    expanded_image = np.expand_dims(image_array, axis=0)
    final_image = keras.applications.inception_v3.preprocess_input(expanded_image)
    plt.imshow(final_image[0])
    plt.title('Preprocessed Image')
```

    return final_image

```python
img = image_preprocessing('E:/All Docs/Masters/CSP 571 Data Prep and Analysis/Project/Data/8k/Flicker8k_Dataset/1000268201_693b08cb0e.jpg')
```

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

![png](output_21_1.png)

```python

```

## Model

### Importing the InceptionV3 Model

```python
inception = keras.applications.inception_v3.InceptionV3(weights='imagenet')
model_1 = keras.models.Model(inputs=inception.input, outputs=inception.layers[-2].output)
```

### Creating Training and Testing Image paths

```python
train_images_paths = []
for i in train_images:
    train_images_paths.append(f'{images_path}{i}.jpg')
```

```python
test_images_paths = []
```

```
for i in test_images:
    test_images_paths.append(f'{images_path}{i}.jpg')
```

### Encoding Images

```python
def image_encode(image):
    image = image_preprocessing(image)
    features = model_1.predict(image)
    features = features.reshape(features.shape[1], )
    return features
```

```python
a = image_encode(train_images_paths[0])
```

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

![png](output_31_1.png)

```python
len(a)
```

    2048

### Encoding Train Images

```python
```

```
    t = time()
    encoded_train_images = {}
    for index, image in enumerate(train_images_paths):
        encoded_train_images[image] = image_encode(image)
        if((index) % 200 == 0):
            print(f'Completed Encoding {index + 1} images')
    print(f'Time taken {time()-t}')
```

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    ** Time taken for encoding train images 1.7288079261779785 seconds **
```

![png](output_34_2.png)

### Encoding Test Images

```python
t = time()
encoded_test_images = {}
```

for index, image in enumerate(test_images_paths):
    encoded_test_images[image] = image_encode(image)
    if((index) % 200 == 0):
        print(f'Completed Encoding {index + 1} images')
print(f'Time taken {time()-t}')

### Saving the Encoded Images to a File

```python
with open("encoded_train_images.pkl", "wb") as encoded_pickle:
    dump(encoded_train_images, encoded_pickle)

with open("encoded_test_images.pkl", "wb") as encoded_pickle:
    dump(encoded_test_images, encoded_pickle)
```

### Creating Training Captions List

```python
training_captions_list = []
for key in training_data.keys():
    for cap in training_data[key]:
        training_captions_list.append(cap)
```

### Creating Testing Captions List

```python
testing_captions_list = []
for key in testing_data.keys():
    for cap in testing_data[key]:
        testing_captions_list.append(cap)
```

```python
train_word_corpus = []
for cap in training_captions_list:
    for word in cap.split():
        train_word_corpus.append(word)
```

```python
train_corpus_counter = Counter(train_word_corpus)
train_corpus = [word for word, count in train_corpus_counter.items() if count >= 10]
```

```python
test_word_corpus = []
for cap in testing_captions_list:
    for word in cap.split():
        test_word_corpus.append(word)
test_corpus_counter = Counter(test_word_corpus)
test_corpus = [word for word, count in test_corpus_counter.items() if count >= 10]
```

### Calculating Maximum Length of all captions

```python
maximum_length_caption = 0
for caption in training_captions_list:
    temp_len = len(caption.split())
    if temp_len > maximum_length_caption:
        maximum_length_caption = temp_len
```

### Creating Word and Index maps

```python
word_dict = {}
index_dict = {}

for index, word in enumerate(train_corpus):
    word_dict[index+1] = word
    index_dict[word] = index+1
```

```python
train_corpus_size = len(word_dict) + 1
```

### Creating Data Generator for Model Training

```python
def generator_training(image_captions, image_features, index_dict, maximum_length_of_caption,
pics_per_batch):
    a = list()
    b = list()
    c = list()
    counter = 0
    while True:
        for image, captions in image_captions.items():
            counter += 1
            img = f'{images_path}{image}.jpg'
            features = image_features[img]
            for caption in captions:
                sequence = [index_dict[word] for word in caption.split(' ') if word in index_dict]
                for i in range(len(sequence)):
                    s1 = sequence[:i]
                    s1 = pad_sequences([s1], maxlen=maximum_length_of_caption)[0]
                    s2 = sequence[i]
                    s2 = to_categorical([s2], num_classes=train_corpus_size)[0]
                    c.append(s2)
                    b.append(s1)
                    a.append(features)
            if counter == pics_per_batch:
                counter = 0
                a = array(a)
                b = array(b)
                c = array(c)
                yield(([a, b], c))
                a = list()
                b = list()
                c = list()
```

### Loading GloVe

```python
weights = {}
with open('E:/All Docs/Masters/CSP 571 Data Prep and Analysis/Project/Data/glove.6B.200d.txt',
encoding="utf-8") as file:
    for line in file:
        weight = line.split()
        word = weight[0]
        wts = np.asarray(weight[1:], dtype='float32')
        weights[word] = wts
```

```python
weights_dimension = 200
```

```python
weight_matrix = np.zeros((train_corpus_size, weights_dimension))
for word, index in index_dict.items():
    if word in weights:
        weight_matrix[index] = weights[word]
```

### Model Creation

```python
input_1 = keras.Input(shape=(2048,))
image_layer_1 = keras.layers.Dropout(0.5)(input_1)
inputs_2 = keras.Input(shape=(maximum_length_caption,))
caption_layer_1 = keras.layers.Embedding(train_corpus_size, weights_dimension,
mask_zero=True)(inputs_2)
caption_layer_2 = keras.layers.Dropout(0.5)(caption_layer_1)
image_layer_2 = keras.layers.Dense(256, activation='relu')(image_layer_1)
caption_layer_3 = keras.layers.LSTM(256)(caption_layer_2)
decoder = keras.layers.merge.add([image_layer_2, caption_layer_3])
fully_connected_layer = keras.layers.Dense(256, activation='relu')(decoder)
fully_connected_layer_2 = keras.layers.Dense(train_corpus_size,
activation='softmax')(fully_connected_layer)
model = keras.models.Model(inputs=[input_1, inputs_2], outputs=fully_connected_layer_2)
model.summary()
```

```
    Model: "functional_7"

__________________________________________________________________________________________
______________
    Layer (type)                Output Shape        Param #     Connected to
```

```
==================================================================================
=============
 input_9 (InputLayer)         [(None, 33)]        0

__________________________________________________________________________________
_____________
 input_8 (InputLayer)         [(None, 2048)]      0

__________________________________________________________________________________
_____________
 embedding_2 (Embedding)       (None, 33, 200)     328800      input_9[0][0]

__________________________________________________________________________________
_____________
 dropout_4 (Dropout)           (None, 2048)        0           input_8[0][0]

__________________________________________________________________________________
_____________
 dropout_5 (Dropout)           (None, 33, 200)     0           embedding_2[0][0]

__________________________________________________________________________________
_____________
 dense_4 (Dense)              (None, 256)         524544      dropout_4[0][0]

__________________________________________________________________________________
_____________
 lstm_2 (LSTM)                (None, 256)         467968      dropout_5[0][0]

__________________________________________________________________________________
_____________
 add_1 (Add)                  (None, 256)         0           dense_4[0][0]
                                                               lstm_2[0][0]

__________________________________________________________________________________
_____________
 dense_5 (Dense)              (None, 256)         65792       add_1[0][0]

__________________________________________________________________________________
_____________
 dense_6 (Dense)              (None, 1644)        422508      dense_5[0][0]

==================================================================================
=============
 Total params: 1,809,612
 Trainable params: 1,809,612
 Non-trainable params: 0
______________________________________________________________________________________________________
```

### Assigning weights to the Model

```python
model.layers[2].trainable = False
model.layers[2].set_weights([weight_matrix])
```

### Compiling the Model

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

### Training the Model

```python
batch_size = 3
epochs = 1
steps = len(training_data) // batch_size
training_features = load(open("encoded_train_images.pkl", "rb"))
for i in range(epochs):
    train_generator = generator_training(training_data, training_features, index_dict, maximum_length_caption, batch_size)
    trainig = model.fit_generator(train_generator, epochs=2, steps_per_epoch=steps, verbose=1)
    model.save(f'{model_path}model_{str(i)}.h5')
```

```
    Epoch 1/2
    2000/2000 [==============================] - 287s 143ms/step - loss: 2.1580
    Epoch 2/2
    2000/2000 [==============================] - 267s 134ms/step - loss: 2.0817
```

```python
model.optimizer.lr = 0.0001
batch_size = 6
epochs = 30
steps = len(training_data) // batch_size
for i in range(epochs):
    train_generator = generator_training(training_data, training_features, index_dict,
maximum_length_caption, batch_size)
    trainig = model.fit_generator(train_generator, epochs=2, steps_per_epoch=steps, verbose=1)
    model.save(f'{model_path}model_{str(30+i)}.h5')
```

### Saving the Model

```python
# model.save_weights(f'{model_path}model_{str(100)}.h5')
```

### Loading the Model

```python
model.load_weights(f'{model_path}model_{str(100)}.h5')
```

```python
def caption_generator(image):
    start = 'startseq'
    for i in range(maximum_length_caption):
        word_sequence = [index_dict[word] for word in start.split() if word in index_dict]
        word_sequence = pad_sequences([word_sequence], maxlen=maximum_length_caption)
        wts = model.predict([image,word_sequence], verbose=0)
        index = np.argmax(wts)
        word = word_dict[index]
        start += ' ' + word
        if word == 'endseq':
            break
    return ' '.join(start.split()[1:-1])
```

```python
pic = "E:/All Docs/Masters/CSP 571 Data Prep and Analysis/Project/Python/Image-Captioning-master/images/112245524.jpg"
# print(pic)
pic1 = image_encode(pic)
image = pic1.reshape((1,2048))
# image = encoding_test[pic].reshape((1,2048))
x=plt.imread(pic)
plt.imshow(x)
plt.show()
out = caption_generator(image)
print("Greedy:",out)
```

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

![png](output_70_1.png)

```
    Greedy: woman in pink shirt is sitting next to man in black shirt
```

```python
maximum_length_caption
```

```
    33
```

### Model Evaluation

```python
def evaluate(captions, obtaind_op):
    c1 = []
    c2 = obtaind_op.split()
    for i in captions:
        c1.append(i.split())
    max_len = len(max(c1, key=len))
    for i in c1:
        i += [' '] * (max_len - len(i))
    c2 += [' '] * (max_len - len(c2))
    me = nltk.translate.meteor_score.meteor_score(captions, obtaind_op)
    bl = nltk.translate.bleu_score.sentence_bleu(c1, c2)
    return me, bl
```

```python
opt = []
for i in range(100):
    pic = test_images_paths[i]
    pic1 = image_encode(pic)
    image = pic1.reshape((1,2048))
    x=plt.imread(pic)
    plt.imshow(x)
    plt.show()
    out = caption_generator(image)
    print("Greedy:",out)
    captions = testing_data[str(pic[85:-4])]
#     print('a', captions)
    opt.append(evaluate(captions, out))
```

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

![png](output_74_1.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
    Greedy: two dogs are wrestling in the snow
```

![png](output_74_4.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: two girls are playing in the water

![png](output_74_7.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: woman in string uniform is walking along the street

![png](output_74_10.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: two girls are sitting on park bench

![png](output_74_13.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: football player in red uniform is running down the field

![png](output_74_16.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: dog is running through the grass

![png](output_74_19.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two women in identical costumes are posing for picture

![png](output_74_22.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: dog is running through the grass

![png](output_74_25.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: black dog is running through the snow

![png](output_74_28.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: the opposing team are competing in the basketball court

![png](output_74_31.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket is standing next to large fountain

![png](output_74_34.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in shirt and glasses is talking to man wearing red hat

![png](output_74_37.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two dogs are shown on each side of tank

![png](output_74_40.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket and tie is standing next to woman in black jacket

![png](output_74_43.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt is standing on top of mountain

![png](output_74_46.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket is standing outside of shop

![png](output_74_49.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: black dog is running through the grass

![png](output_74_52.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black shirt is sitting on chair with his dog

![png](output_74_55.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two dogs are playing together in the grass

![png](output_74_58.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket is sitting on the ground reading book

![png](output_74_61.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man is standing on snowcapped mountain looking at the mountains surrounding him

![png](output_74_64.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two girls are sitting on bed in front of the door

![png](output_74_67.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket waves cigarette in crowd of people

![png](output_74_70.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two children are playing on tire

![png](output_74_73.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt is pulling cart on the beach

![png](output_74_76.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: people walking along the street seen some are walking past

![png](output_74_79.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket is eating

![png](output_74_82.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black shirt is eating piece of

![png](output_74_85.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man is climbing up huge boulder

![png](output_74_88.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black shirt and sunglasses is smiling

![png](output_74_91.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: football players huddle

![png](output_74_94.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red climbs sheer cliff face

![png](output_74_97.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: dog is running through doorway leading

![png](output_74_100.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt is attached to rope whilst camping

![png](output_74_103.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: dog is running through the grass

![png](output_74_106.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt and jeans follows

![png](output_74_109.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two boys are playing with fallen football in the grass

![png](output_74_112.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: dog is running through the grass

![png](output_74_115.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two men are playing basketball on court

![png](output_74_118.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: woman in black and white dress is walking across the street

![png](output_74_121.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: baby in red shirt is sitting on wooden bench

![png](output_74_124.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: group of people stand in front of large crowd

![png](output_74_127.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two girls lay on bed

![png](output_74_130.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt is climbing up rock

![png](output_74_133.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt is standing on rock looking at the view

![png](output_74_136.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: baby in red pajamas plays with toy

![png](output_74_139.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: boy in red shirt is jumping off of bar

![png](output_74_142.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red skiing gear facing down snowy hill

![png](output_74_145.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two dogs play in the water

![png](output_74_148.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two people are walking along the beach near the ocean

![png](output_74_151.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two girls are sitting on bed in front of the window

![png](output_74_154.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two men are playing basketball

![png](output_74_157.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: young boy wearing red shirt is playing in the water

![png](output_74_160.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```

```
    Greedy: woman in black and white print jacket is standing next to two other people standing by table
```

![png](output_74_163.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```

```
    Greedy: man in black shirt is standing onstage
```

![png](output_74_166.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```

```
    Greedy: dog swims through the water
```

![png](output_74_169.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for
integers).
```

```
    Greedy: young boy wearing red shirt is swinging on swing
```

![png](output_74_172.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt is standing on rock looking out at the sunset

![png](output_74_175.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: black dog is walking along the beach

![png](output_74_178.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black coat is standing in front of an machine

![png](output_74_181.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket is standing on rock overlooking the water

![png](output_74_184.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket is sitting on wooden bench

![png](output_74_187.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red uniform is riding red motorbike

![png](output_74_190.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two men are playing basketball on court

![png](output_74_193.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt and jeans follows uphill on fancy bike

![png](output_74_196.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two girls are sitting in front of bus

![png](output_74_199.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black shirt and knee is standing on the shore

![png](output_74_202.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two white and brown poodles are playing on the sand

![png](output_74_205.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket is standing outside smoking cigarette

![png](output_74_208.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    Greedy: black and white dog is running through the grass
```

![png](output_74_211.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    Greedy: man in red jacket is walking along rocky path
```

![png](output_74_214.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    Greedy: two dogs are playing together on the grass
```

![png](output_74_217.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    Greedy: group of children are lined in line of an adults
```

![png](output_74_220.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two people are fishing on the end of the water

![png](output_74_223.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: boy in striped shirt is jumping off of playground equipment

![png](output_74_226.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black shirt is doing stunt on bike

![png](output_74_229.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: boy in red shirt and black shorts is smiling whilst looking away from the camera

![png](output_74_232.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red jacket is snowboarding down snowy hill

![png](output_74_235.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: boy in red shirt is running through water

![png](output_74_238.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two girls are doing handstand on trampoline

![png](output_74_241.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt climbs rock cliff

![png](output_74_244.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two men lift their arms as they are sitting on the ground

![png](output_74_247.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red uniform is riding bike down dirt hill

![png](output_74_250.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in red shirt climbs sheer cliff face

![png](output_74_253.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man is climbing up cliff

![png](output_74_256.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: girl in pink shirt and jean shorts makeup watches

![png](output_74_259.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: dog runs through the water

![png](output_74_262.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: man in black shirt is standing next to statue

![png](output_74_265.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: two dogs are playing together on the grass

![png](output_74_268.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in plaid shirt is eating piece of

![png](output_74_271.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: woman in pink shirt is eating with blonde girl in shirt

![png](output_74_274.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: two dogs play in the snow

![png](output_74_277.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: black and white dog is running through the water

![png](output_74_280.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: woman in skimpy bikini is throwing her wet into the air

![png](output_74_283.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: two men in gym outfits are playing

![png](output_74_286.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: man in black shirt is helping child on his back in the seat

![png](output_74_289.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: young boy is running on the grass with his arms in the air

![png](output_74_292.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black jacket and tie hat stands in front of roller coaster

![png](output_74_295.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: man in black shirt is riding surfboard on the beach

![png](output_74_298.png)

    Greedy: dog is licking its nose

```python
opt = []
for i in range(10):
    pic = test_images_paths[i]
    pic1 = image_encode(pic)
    image = pic1.reshape((1,2048))
    x=plt.imread(pic)
    plt.imshow(x)
    plt.show()
    out = caption_generator(image)
    print("Greedy:",out)
    captions = testing_data[str(pic[85:-4])]
#     print('a', captions)
    opt.append(evaluate(captions, out))
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

![png](output_75_1.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    Greedy: two dogs are wrestling in the snow
```

![png](output_75_4.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    Greedy: two girls are playing in the water
```

![png](output_75_7.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).


    Greedy: woman in string uniform is walking along the street
```

![png](output_75_10.png)

```
    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

Greedy: two girls are sitting on park bench

![png](output_75_13.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: football player in red uniform is running down the field

![png](output_75_16.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: dog is running through the grass

![png](output_75_19.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Greedy: two women in identical costumes are posing for picture

![png](output_75_22.png)

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: dog is running through the grass

![png](output_75_25.png)

    Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

    Greedy: black dog is running through the snow

![png](output_75_28.png)

    Greedy: the opposing team are competing in the basketball court

```python
opt2 = []
for i in opt:
    opt2.append(i[0])
opt3 = []
for i in opt:
    opt3.append(i[1])
```

```python
b_score = sum(opt3) / float(len(opt3))
print(f'Average Bleu Score for the model is - {b_score}')
```

    Average Bleu Score for the model is - 0.41089122809839507

```python
```

~~~
```
~~~

REFERENCES:

https://ieeexplore.ieee.org/abstract/document/8844921

https://blogs.rstudio.com/ai/posts/2018-07-30-attention-layer/

https://ieeexplore.ieee.org/document/9105191

https://ieeexplore.ieee.org/abstract/document/9071372