iNeuron

Detailed Project Report(DPR)

SUBMITTED BY

PRATEEK MAGDUM

(IMAGE CAPTIONING PROJECT)

INEURON INTERNSHIP

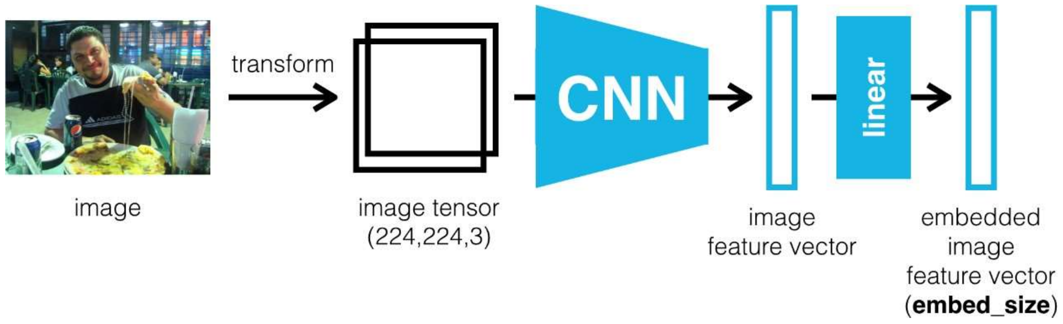
\

Domain : Security and Safety, Surveillance

IMAGE - CAPTIONING

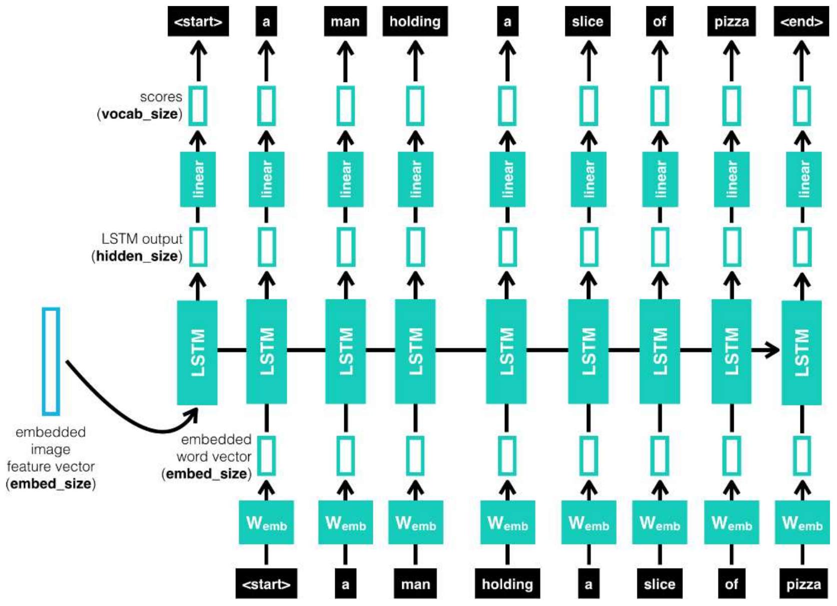
Explaining Encoder and Decoder

Encoder:



For Encoder we use VGG-16. VGG-16 is a convolutional neural network that is trained on more than a million images from the ImageNet database. The network is 50 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.

Decode:


<start>
a
man
holding
a
slice
of
pizza
<end>
scores (vocab_size)
linear
LSTM output (hidden_size)
LSTM
embedded image feature vector (embed_size)
embedded word vector (embed_size)
Wemb
<start>
a
man
holding
a
slice
of
pizza

For Decoder we use LSTM. Long Short-Term Memory (LSTM) networks are a modified version of recurrent neural networks, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation

Table of Contents

Image Caption Generator using Python | Flickr 8k Dataset

Image caption generator is a process of recognizing the context of an image and annotating it with relevant captions using deep learning, and computer vision. This is an advanced deep learning project where more than one model must be used for analysis and preprocessing the data to obtain the results.
In this project tutorial, we will build an image caption generator to load a random image and give some captions describing the image. We will use Convolutional Neural Network (CNN) for image feature extraction and Long Short-Term Memory Network (LSTM) for Natural Language Processing (NLP).

Dataset Information

The objective of the project is to predict the captions for the input image. The dataset consists of 8k images and 5 captions for each image. The features are extracted from both the image and the text captions for input.

The features will be concatenated to predict the next word of the caption. CNN is used for image and LSTM is used for text. BLEU Score is used as a metric to evaluate the performance of the trained model.

Import Modules

First, we have to import all the basic modules we will be needing for this project

```python
import os
import pickle
import numpy as np
from tqdm.notebook import tqdm
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
```

- **os -** used to handle files using system commands.
- **pickle** - used to store numpy features extracted

- **numpy** - used to perform a wide variety of mathematical operations on arrays
- **tqdm** - progress bar decorator for iterators. Includes a default range iterator printing to stderr.
- **VGG16, preprocess_input** - imported modules for feature extraction from the image data
- **load_img, img_to_array** - used for loading the image and converting the image to a numpy array
- **Tokenizer** - used for loading the text as convert them into a token
- **pad_sequences -** used for equal distribution of words in sentences filling the remaining spaces with zeros
- **plot_model** - used to visualize the architecture of the model through different images

Now we must set the directories to use the data

```
BASE_DIR = '/kaggle/input/flickr8k'
WORKING_DIR = '/kaggle/working'
```

Extract Image Features

We have to load and restructure the model

```
# load vgg16 model
model = VGG16()
# restructure the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
# summarize
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-
applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5 553467904/553467096
[==============================] - 3s 0us/step 553476096/553467096
[==============================] - 3s 0us/step Model: "model"
_________________________________________________________________ Layer (type)
Output Shape Param #
================================================================= input_1
(InputLayer) [(None, 224, 224, 3)] 0

_________________________________________________________________
block1_conv1 (Conv2D) (None, 224, 224, 64) 1792

_________________________________________________________________
block1_conv2 (Conv2D) (None, 224, 224, 64) 36928
_________________________________________________________________ block1_pool
(MaxPooling2D) (None, 112, 112, 64) 0

_________________________________________________________________
block2_conv1 (Conv2D) (None, 112, 112, 128) 73856

_________________________________________________________________
block2_conv2 (Conv2D) (None, 112, 112, 128) 147584
_________________________________________________________________ block2_pool
(MaxPooling2D) (None, 56, 56, 128) 0

_________________________________________________________________
block3_conv1 (Conv2D) (None, 56, 56, 256) 295168

_________________________________________________________________
```

```
block3_conv2 (Conv2D) (None, 56, 56, 256) 590080
_________________________________________________________________
block3_conv3 (Conv2D) (None, 56, 56, 256) 590080
_________________________________________________________________ block3_pool
(MaxPooling2D) (None, 28, 28, 256) 0
_________________________________________________________________
block4_conv1 (Conv2D) (None, 28, 28, 512) 1180160
_________________________________________________________________
block4_conv2 (Conv2D) (None, 28, 28, 512) 2359808
_________________________________________________________________
block4_conv3 (Conv2D) (None, 28, 28, 512) 2359808
_________________________________________________________________ block4_pool
(MaxPooling2D) (None, 14, 14, 512) 0
_________________________________________________________________
block5_conv1 (Conv2D) (None, 14, 14, 512) 2359808
_________________________________________________________________
block5_conv2 (Conv2D) (None, 14, 14, 512) 2359808
_________________________________________________________________
block5_conv3 (Conv2D) (None, 14, 14, 512) 2359808
_________________________________________________________________ block5_pool
(MaxPooling2D) (None, 7, 7, 512) 0
_________________________________________________________________ flatten
(Flatten) (None, 25088) 0
_________________________________________________________________ fc1 (Dense)
(None, 4096) 102764544
_________________________________________________________________ fc2 (Dense)
(None, 4096) 16781312
================================================================= Total
params: 134,260,544 Trainable params: 134,260,544 Non-trainable params: 0
_________________________________________________________________ None
```

- Fully connected layer of the VGG16 model is not needed, just the previous layers to extract feature results.
- By preference you may include more layers, but for quicker results avoid adding the unnecessary layers.

Now we extract the image features and load the data for preprocess

```
# extract features from image

features = {}

directory = os.path.join(BASE_DIR, 'Images')

for img_name in tqdm(os.listdir(directory)):
    # load the image from file
    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, verbose=0)
    # get image ID
    image_id = img_name.split('.')[0]
    # store feature
    features[image_id] = feature
```

- Dictionary 'features' is created and will be loaded with the extracted features of image data
- **load_img(img_path, target_size=(224, 224))** - custom dimension to resize the image when loaded to the array
- **image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))** - reshaping the image data to preprocess in a RGB type image.
- **model.predict(image, verbose=0)** - extraction of features from the image
- **img_name.split('.')[0]** - split of the image name from the extension to load only the image name.

```
# store features in pickle
pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'), 'wb'))
```

- Extracted features are not stored in the disk, so re-extraction of features can extend running time
- Dumps and store your dictionary in a pickle for reloading it to save time

```
# load features from pickle
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
```

- Load all your stored feature data to your project for quicker runtime

Load the Captions Data

Let us store the captions data from the text file

```
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
```

Now we split and append the captions data with the image

```
# create mapping of image to captions
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
```

- Dictionary 'mapping' is created with key as image_id and values as the corresponding caption text
- Same image may have multiple captions, **if image_id not in mapping: mapping[image_id] = []** creates a list for appending captions to the corresponding image

Now let us see the no. of images loaded

```
len(mapping)
```

```
    8091
```

Preprocess Text Data

```
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in          caption.split() if
len(word)>1]) + ' endseq'
            captions[i] = caption
```

- Defined to clean and convert the text for quicker process and better results

Let us visualize the text before and after cleaning

```
# before preprocess of text
mapping['1000268201_693b08cb0e']
```

```
['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 .']
```

```
# preprocess the text
clean(mapping)

# after preprocess of text
 mapping['1000268201_693b08cb0e']
```

```
['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']
```

- Words with one letter was deleted
- All special characters were deleted
- 'startseq' and 'endseq' tags were added to indicate the start and end of a caption for easier processing

Next we will store the preprocessed captions into a list

```
all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)


len(all_captions)
```

```
40455
```

- No. of unique captions stored

Let us see the first ten captions

```
all_captions[:10]
```

```
['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 tri-colored 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']
```

Now we start processing the text data

```
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
```

```
vocab_size
      8485
```

- No. of unique words

```
# get maximum length of the caption available
max_length = max(len(caption.split()) for caption in all_captions)
max_length
      35
```

- Finding the maximum length of the captions, used for reference for the padding sequence.

Train Test Split

After preprocessing the data now we will train, test and split

```
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
```

Explanatory example of the sequence split into pairs

```
# startseq girl going into wooden building endseq
#        X                   y
# startseq                   girl
# startseq girl              going
# startseq girl going        into
# ...........
# startseq girl going into wooden building      endseq
```

Now we will define a batch and include the padding sequence

```
# create data generator to get data in batch (avoids session crash)
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size,
batch_size):
    # loop over images
    X1, X2, y = list(), list(), list()
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence
            seq = tokenizer.texts_to_sequences([caption])[0]
            # split the sequence into X, y pairs
                for i in range(1, len(seq)):
                # split into input and output pairs
                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 the sequences
                X1.append(features[key][0])
                X2.append(in_seq)
                y.append(out_seq)
          if n == batch_size:
              X1, X2, y = np.array(X1), np.array(X2), np.array(y)
              yield [X1, X2], y
              X1, X2, y = list(), list(), list()
              n = 0
```

- Padding sequence normalizes the size of all captions to the max size filling them with zeros for better results.

Model Creation

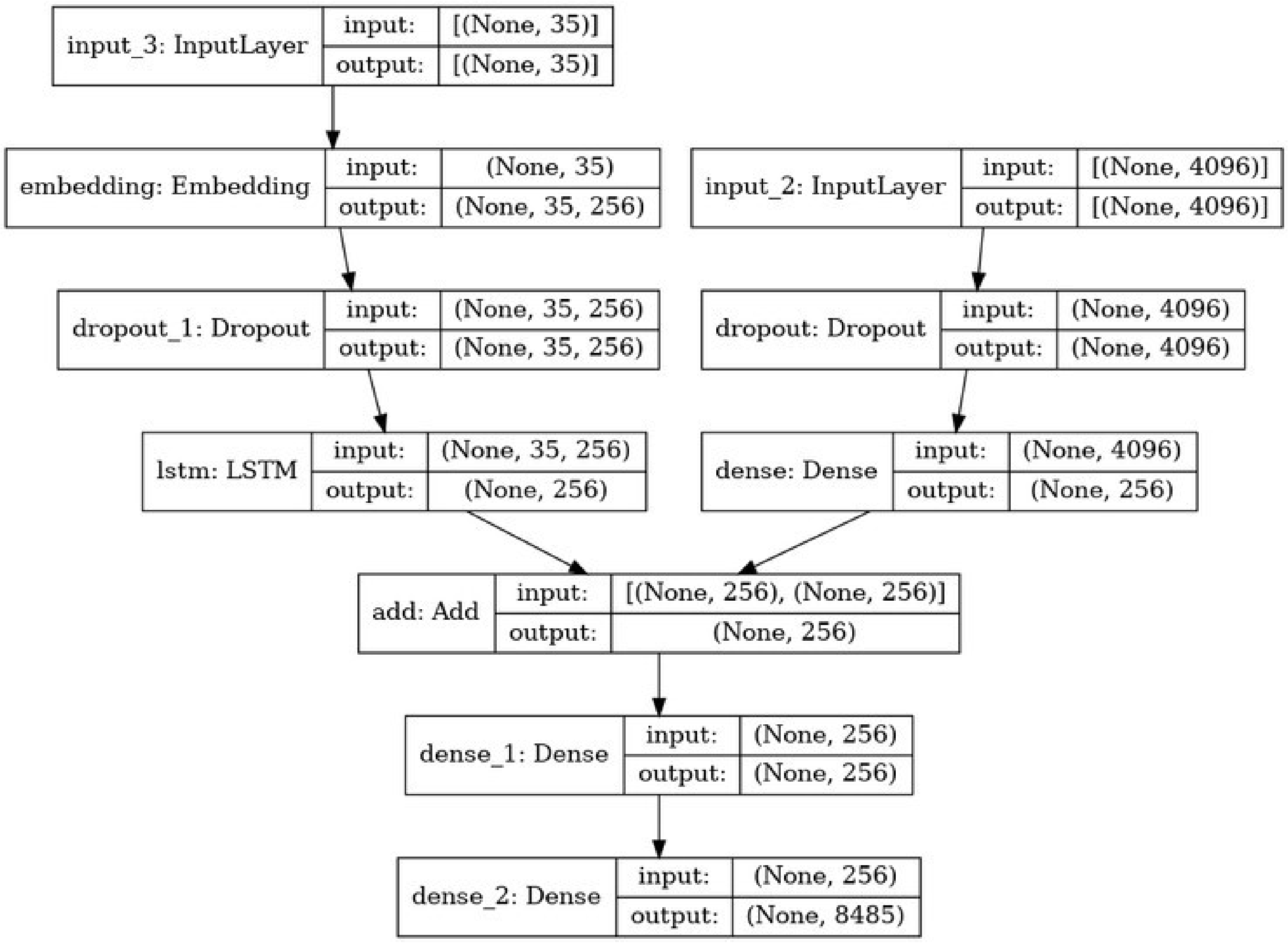
```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')

# plot the model
plot_model(model, show_shapes=True)
```

```
# plot the model
plot_model(model, show_shapes=True)
```



- **shape=(4096,)** - output length of the features from the VGG model
- **Dense** - single dimension linear layer array
- **Dropout()** - used to add regularization to the data, avoiding over fitting & dropping out a fraction of the data from the layers
- **model.compile()** - compilation of the model
- **loss='sparse_categorical_crossentropy'** - loss function for category outputs
- **optimizer='adam'** - automatically adjust the learning rate for the model over the no. of epochs
- Model plot shows the concatenation of the inputs and outputs into a single layer
- Feature extraction of image was already done using VGG, no CNN model was needed in this step.

Now let us train the model

```
# train the model
epochs = 20
batch_size = 32
steps = len(train) // batch_size

for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
```

```
227/227 [==============================] - 68s 285ms/step - loss: 5.2210 227/227
[==============================] - 66s 291ms/step - loss: 4.0199 227/227
```

```
[==============================] - 66s 292ms/step - loss: 3.5781 227/227
[==============================] - 65s 287ms/step - loss: 3.3090 227/227
[==============================] - 66s 292ms/step - loss: 3.1080 227/227
[==============================] - 65s 286ms/step - loss: 2.9619 227/227
[==============================] - 63s 276ms/step - loss: 2.8491 227/227
[==============================] - 64s 282ms/step - loss: 2.7516 227/227
[==============================] - 64s 282ms/step - loss: 2.6670 227/227
[==============================] - 65s 286ms/step - loss: 2.5966 227/227
[==============================] - 66s 290ms/step - loss: 2.5327 227/227
[==============================] - 61s 270ms/step - loss: 2.4774 227/227
[==============================] - 65s 288ms/step - loss: 2.4307 227/227
[==============================] - 66s 289ms/step - loss: 2.3873 227/227
[==============================] - 62s 274ms/step - loss: 2.3451 227/227
[==============================] - 65s 285ms/step - loss: 2.3081 227/227
[==============================] - 65s 288ms/step - loss: 2.2678 227/227
[==============================] - 66s 292ms/step - loss: 2.2323 227/227
[==============================] - 65s 285ms/step - loss: 2.1992 227/227
[==============================] - 66s 291ms/step - loss: 2.1702
```

- **steps = len(train) // batch_size** - back propagation and fetch the next data
- Loss decreases gradually over the iterations
- Increase the no. of epochs for better results
- Assign the no. of epochs and batch size accordingly for quicker results

You can save the model in the working directory for reuse

```
# save the model
model.save(WORKING_DIR+'/best_model.h5')
```

Generate Captions for the Image

```
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
        return word
    return None
```

- Convert the predicted index from the model into a word

```
# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break
    return in_text
```

- Caption generator appending all the words for an image
- The caption starts with 'startseq' and the model continues to predict the caption until the 'endseq' appeared

Now we validate the data using BLEU Score

```
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)
    # calcuate BLEU score
    print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
    print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

```
BLEU-1: 0.516880 BLEU-2: 0.293009
```

- BLEU Score is used to evaluate the predicted text against a reference text, in a list of tokens.
- The reference text contains all the words appended from the captions data (actual_captions)
- A BLEU Score more than 0.4 is considered a good result, for a better score increase the no. of epochs accordingly.

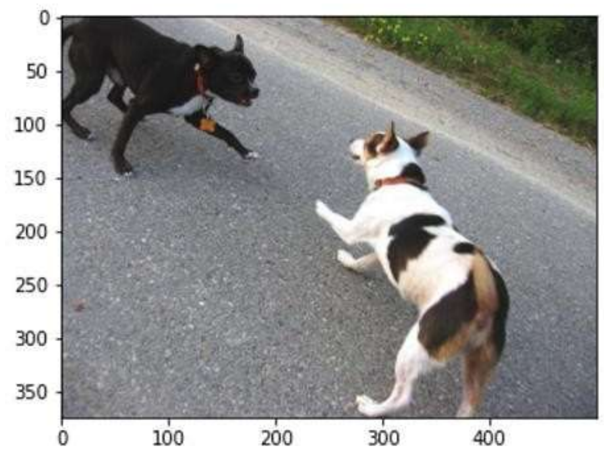
Visualize the Results

```python
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)
    image = Image.open(img_path)
    captions = mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
```

- Image caption generator defined
- First prints the actual captions of the image then prints a predicted caption of the image

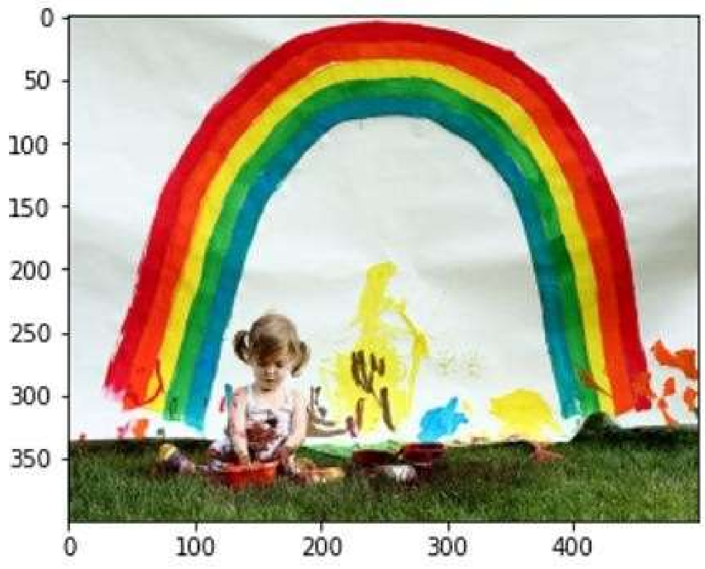
```python
generate_caption("1001773457_577c3a7d70.jpg")
```

```
_____________Actual_____________
startseq black dog and spotted dog are fighting endseq
startseq black dog and tri-colored 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
____________Predicted____________
startseq two dogs play with each other in the grass endseq
```



```
generate_caption("1002674143_1b742ab4b8.jpg")
```

```
____________Actual____________
startseq little girl covered in paint sits in front of painted rainbow with her hands in
bowl endseq startseq little girl is sitting in front of large painted rainbow endseq
startseq small girl in the grass plays with fingerpaints in front of white canvas with
rainbow on it endseq
startseq there is girl with pigtails sitting in front of rainbow painting endseq
startseq young girl with pigtails painting outside in the grass endseq
____________Predicted____________
startseq little girl in pink dress is lying on the side of the grass endseq
```



```
generate_caption("101669240_b2d3e7f17b.jpg")
```

```
____________Actual____________
startseq man in hat is displaying pictures next to skier in blue hat endseq
startseq man skis past another man displaying paintings in the snow endseq
startseq person wearing skis looking at framed pictures set up in the snow endseq
startseq skier looks at framed pictures in the snow next to trees endseq
startseq man on skis looking at artwork for sale in the snow endseq
____________Predicted____________
startseq two people are hiking up snowy mountain endseq
```

\