IMAGE CAPTION GENERATOR

2019/2020 – Third Year_6th Semester

EC9170 - Deep Learning

Submitted By:

Gunarathna L.P.N (2020/E/046)

Lakshan W.G. (2020/E/079)

Somarathna S.V.A.P.K. (2020/E/212)

DEPARTMENT OF ELECTRICAL ENGINEERING

FACULTY OF ENGINEERING

UNIVERSITY OF JAFFNA

14 JULY 2024

TABLE OF CONTENTS

EXCLUSIVE SUMMERY

Project Overview:

The Objective of this Project is to develop two Distinct Models to be used in generating image Captions by using Image caption generator Dataset. For the image feature extraction, we have applied a pretrained CNN model in this work and, for the caption generation, we have used two variants of LSTM structures. Performance metrics and qualitative analysis will be used to assess the models.

Dataset :

- Contains 8,091 images, each paired with five different captions describing the scene

Models :

1. Feature Extraction
    - VGG16: A pre-trained convolutional neural network (CNN) used for extracting features from images.
2. Caption Generation
    - Uni-directional LSTM: Processes input sequences in a single direction.
    - Bidirectional LSTM: Processes input sequences in both forward and backward directions.

Steps :

1. Data Preparation:
    - Load and Preprocess the images and Captions on Google Colab .
    - Resize and normalize images for Compatible with input layer of the VGG16 model .
    - Tokenize and encode captions for compatible with LSTM models.
2. Feature Extraction:
    - Use VGG16 to extract features from each image.
    - Store extracted features to be used as input for the LSTM models.
3. Model Training :
    - Train the Uni-directional LSTM model using extracted features and corresponding captions.
    - Train the Bidirectional LSTM model using previously extracted features and captions.

4. Evaluation and Comparison:
    - Evaluate model performance using metrics (BLEU scores, training loss, and validation loss.)
    - Generate captions for random test images and images outside the dataset

5. Generate Captions :

    - Generate captions for random test images.
    - Generate captions for images from outside the Dataset.

6. Report Preparation :

    - The final step would be to write a comprehensive report documenting the entire project.
    - Summarize the models' performances.

Evaluation Metrics:

BLEU Score :

- Determines the accuracy of the generated captions . It is done by comparing them to reference captions, that is captions that were previously written for this text. The overall performance is key graded by the higher overall BLEU scores.

Training Loss:

- Suggests how effectively the model is memorizing the training data in each epoch on the training set. The lower training loss gives a good fit to the training data set.

Validation Loss:

- Describes how the model is doing concerning unseen elements in the data. When validation loss is low generalization is good.

INTRODUCTION

Background and Information:

Image captioning is the procedure of developing a written description for a given picture. A lot of enhance has been made concerning the image captioning technologies more so with the progress of Computer vision along with the natural language processing. Here, the attention is paid to the construction of various image captioning models and selected methods comparison to define the most efficient for producing the desired expected output.

The Process of build a Image caption generator is split into two stages.

- Pretrained CNN model for Feature Extraction :

CNN is one of the subfield of Deep learning and specific kind of deep neural networks used for image recognition and classification. In case of Extract features of each image we have used the VGG16 pre-trained convolutional neural network (CNN). VGG16 is a deep net which is trained over ImageNet dataset and is beneficial in various image recognition problems. It comprises of 16 layers where 13 of them are convolutional layers and 3 are the fully connected layers.

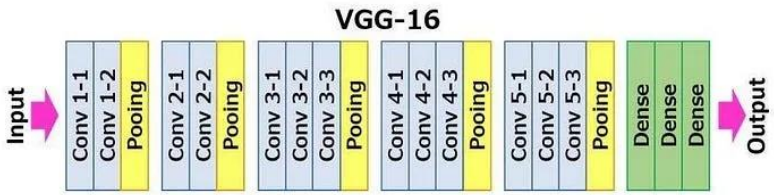


Figure 1 - VGG16 Pretrained Model Architecture

The last two layers of VGG16 are fully connected (dense) layers followed by a SoftMax classification layer. These layers are specifically designed for the Classify images in to categories. For image captioning, the specific classes that the final layer outputs are not relevant. Instead, we need a high-level feature representation of the image that can be used by another model, like an LSTM, to generate captions. Therefore remove the last two layers to get a manageable, fixed-size feature vector for Caption Generator. This Extracted features contains rich representation of the visual content in each image .

```
IMAGES_DIR = '/content/drive/MyDrive/Image_caption_generator_Dataset/Image caption generator Dataset/Images'
CAPTIONS_FILE = '/content/drive/MyDrive/Image_caption_generator_Dataset/Image caption generator Dataset/captions.txt'

# Load the VGG16 model
model = VGG16()
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
```

Figure 2 - Model define and remove Last two layers from the model

- LSTM model for Caption Generation :

Recurrent Neural Networks (RNNs) are a type of neural network designed to handle sequential data by maintaining a hidden state that captures information from previous steps in the sequence. This makes them suitable for tasks such as time series prediction, language modeling, and sequence classification. LSTM networks are a type of recurrent neural network (RNN) capable of learning long-term dependencies, making them well-suited for sequence prediction tasks like Image Captioning. In this Project We have Experimented with two type of LSTM Architectures.

1. Uni-Directional LSTM Architecture
2. Bi-Directional LSTM Architecture

Uni-Directional LSTM Architecture

- The Uni-directional LSTM analyzes the input sequence in one direction manner, commonly from the first element to the last.
- Mainly Suitable for tasks where future context is not necessary to predict the current state.
- Comprises a single LSTM layer and it takes input from the previous time step and provides output for the next time step.

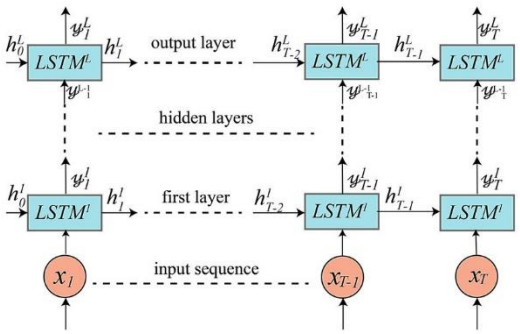


Figure 3 - Uni directional LSTM architecture

Bi-Directional LSTM Architecture

- The Bi-directional LSTM Processes the input sequence in both directions, from the first element to the last and from the last element to the first, and then combines the information from both directions.
- Useful for tasks where context from both past and future elements is important Such as image Captioning , Sentiment Analysis.
- Consists of two LSTM layers, one processing the sequence forward and the other processing it backward. The outputs of these two layers are then concatenated.

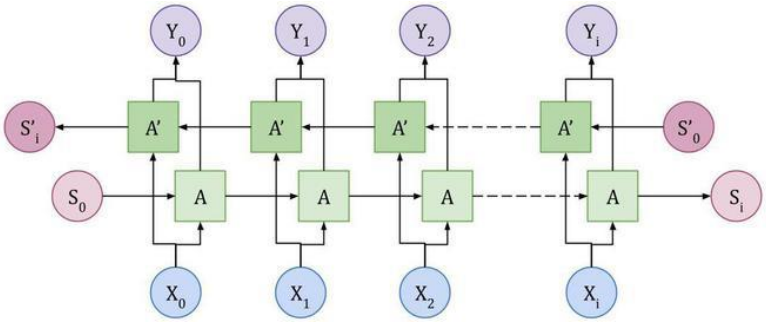


Figure 4 - Bi-Directional LSTM architecture

Our primary goal is to assess the performance of these models using various metrics such as BLEU scores, training loss, and validation loss and Select the most Suitable model for image Captioning tasks.

Methodology

1.Observe the data set

we use a dataset that consists of an image folder containing 8,091 images and a captions.txt file. The dataset is structured as follows:

- Image Folder: This folder contains 8,091 images in jpg formats. These images serve as the input data for our pretrained model.
- captions.txt: This file contains the captions corresponding to each image. Each line in the text file includes an image name with the .jpg extension followed by a caption associated with that image.

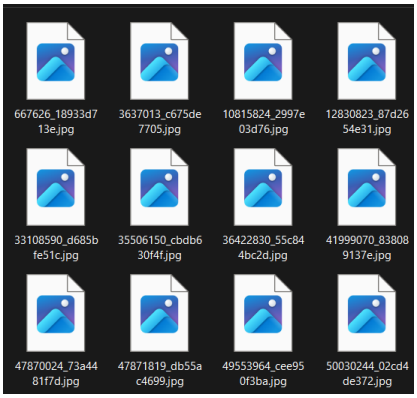


Figure 5 - Image Folder

```
image , caption
1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg,A girl going into a wooden building .
1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg,A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg,A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg,A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg,A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg,A black dog and a white dog with brown spots are staring at each other in the street .
1001773457_577c3a7d70.jpg,Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg,Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg,A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
1002674143_1b742ab4b8.jpg,A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg,A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
1002674143_1b742ab4b8.jpg,There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg,Young girl with pigtails painting outside in the grass .
```

Figure 6 - Captions.txt file

2.Preprocessing

Image Preprocessing And Feature Extraction

- Finally, it is noted that all the Images in the Data set are in different sizes. But the input of trained model is of VGG16 pretrained 224x224. Thus, there is a need to consider Image Size in order to resize.
- There is a function prescribed to the Keras framework for VGG16 model that preprocessing the input image based on the VGG model's prerequisites.
- After these Preprocessing Steps are done , Pretrained model with extract features from each image.

Code

```
# Extract features from images and save them
features = {}
for img_name in tqdm(os.listdir(IMAGES_DIR)):
    # Load the image from file
    img_path = os.path.join(IMAGES_DIR, 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 the 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
    # Save feature to a numpy file
    feature_path = os.path.join(FEATURES_DIR, f'{image_id}.npy')
    np.save(feature_path, feature)

print(f"Features saved to {FEATURES_DIR}")
```

Output

```
100%
 8091/8091 [17:10<00:00,  9.82it/s]
Features saved to /content/drive/MyDrive/ML/Image caption generator Dataset/Image caption generator Dataset/features
```

Captions Preprocessing

- From the above input, Load the captions and Mapping the Captions with Corresponding images.
- The captions have included ,
    - A mix of upper and lower case letters,
    - Multiple spaces,
    - No indicator for the start and end of each sentence,
    - Special characters,
    - Single-letter words which do not contribute meaningful content.
- When preparing captions for insertion into the model, it's crucial to clean and standardize them to ensure neglect this special contents.

Code

```
# 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)
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)

# Verify the result
print(mapping)
```

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

Outputs

Before Clean Captions

```
# before preprocess of text
mapping['1009434119_febe49276a']
```

```
['A black and white dog is running in a grassy garden surrounded by a white fence .',
 'A black and white dog is running through the grass .',
 'A Boston terrier is running in the grass .',
 'A Boston Terrier is running on lush green grass in front of a white fence .',
 'A dog runs on the green grass near a wooden fence .']
```

After Clean Captions

```
# preprocess the text
clean(mapping)
# after preprocess of text
mapping['1009434119_febe49276a']
```

```
['<start> black and white dog is running in grassy garden surrounded by white fence <end>',
 '<start> black and white dog is running through the grass <end>',
 '<start> boston terrier is running in the grass <end>',
 '<start> boston terrier is running on lush green grass in front of white fence <end>',
 '<start> dog runs on the green grass near wooden fence <end>']
```

- Tokenize Captions - Tokenizing is the process of breaking any text of to several components known as tokens. It is advisable that these tokens can be words, sub words, or characters. Thus, while converting the text to tokens and numbers, we can use this token for Natural Language Processing Purpose like caption generation.

Code

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
```

Split the Data in to Train Data and Test Data

Splitting the Data set in to Training Data and Testing Data is essential to test the Model's ability to perform on unseen data. maintaining a separate set for testing is useful in instances where they help track the progress of the model during training and If the model starts to memorize the training data' a holdout set is ideal since the training is early stopping.

Code

```
image_ids = list(mapping.keys())
splitData = int(len(image_ids) * 0.90)
train = image_ids[:splitData]
test = image_ids[splitData:]
train_features = {}
```

```
for i in tqdm(train):
    train_features[i] = features[i]
```

```
test_features = {}
for i in tqdm(test):
    test_features[i] = features[i]
```

```
import random
# Setting random seed for reproducibility of results
random.seed('1000')
# Shuffle train data

random.shuffle(train)
train_captions = {_id: mapping[_id] for _id in train}

test_captions = {_id: mapping[_id] for _id in test}
```

Outputs

100%|██████████| 7281/7281 [00:00<00:00, 754005.42it/s]

100%|██████████| 810/810 [00:00<00:00, 630663.86it/s]

3.Model Define

Model with Uni directional LSTM

CNN+ Uni directional Model Architecture

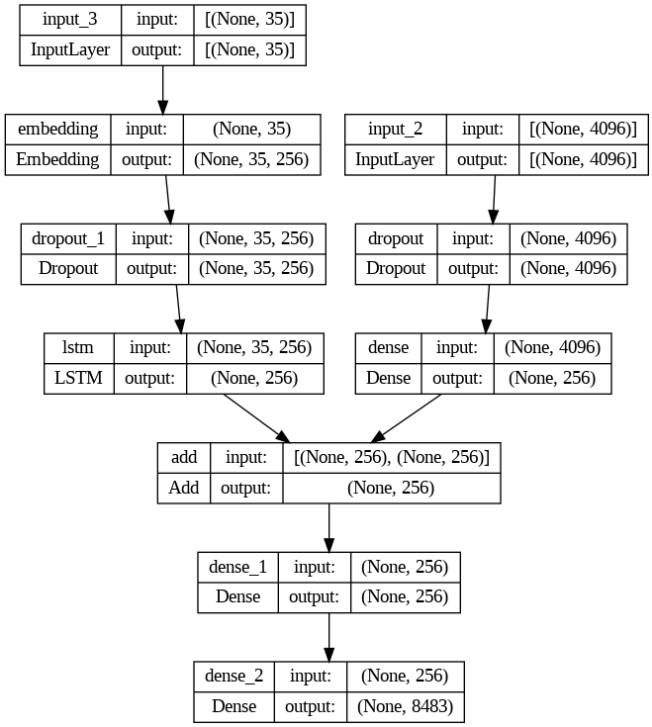


Figure 7 - CNN + Uni-Directional LSTM Architecture

Code

```
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
```

```
# define the captioning model1
def RNN_model_UniDirectLSTM(vocab_size, max_length, embedding_size, units,
input_size):
    # feature extractor model
    imageInput = Input(shape=(input_size,))
    flattenImg = Dropout(0.4)(imageInput)
    fullyConnected = Dense(embedding_size, activation='relu')(flattenImg)
    # sequence model
    captionInput = Input(shape=(max_length,))
    sequence1 = Embedding(vocab_size, embedding_size, mask_zero=True)(captionInput)
    sequence2 = Dropout(0.3)(sequence1)
    sequence3 = LSTM(units)(sequence2)
    # decoder model
    decoder1 = add([fullyConnected, sequence3])
    decoder2 = Dense(units, activation='relu')(decoder1)
    outputs = Dense(vocab_size, activation='softmax')(decoder2)
    # tie it together [image, seq] [word]
    model = Model(inputs=[imageInput, captionInput], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    print(model.summary())
    return model
```

```
# define the model
embedding_size = 256
units = 256
input_size = 4096
finalmodel = RNN_model_UniDirectLSTM(vocab_size, max_length, embedding_size, units,
input_size)
```

Output

```
Model: "model_1"
__________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==========================================================================================
 input_3 (InputLayer)        [(None, 35)]              0         []

 input_2 (InputLayer)        [(None, 4096)]            0         []

 embedding (Embedding)       (None, 35, 256)           2171648   ['input_3[0][0]']

 dropout (Dropout)           (None, 4096)              0         ['input_2[0][0]']

 dropout_1 (Dropout)         (None, 35, 256)           0
['embedding[0][0]']

 dense (Dense)               (None, 256)               1048832   ['dropout[0][0]']

 lstm (LSTM)                 (None, 256)               525312
['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, 8483)              2180131   ['dense_1[0][0]']

==========================================================================================
Total params: 5991715 (22.86 MB)
Trainable params: 5991715 (22.86 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________
None
```

Model with Bi directional LSTM

CNN+ Bi directional Model Architecture

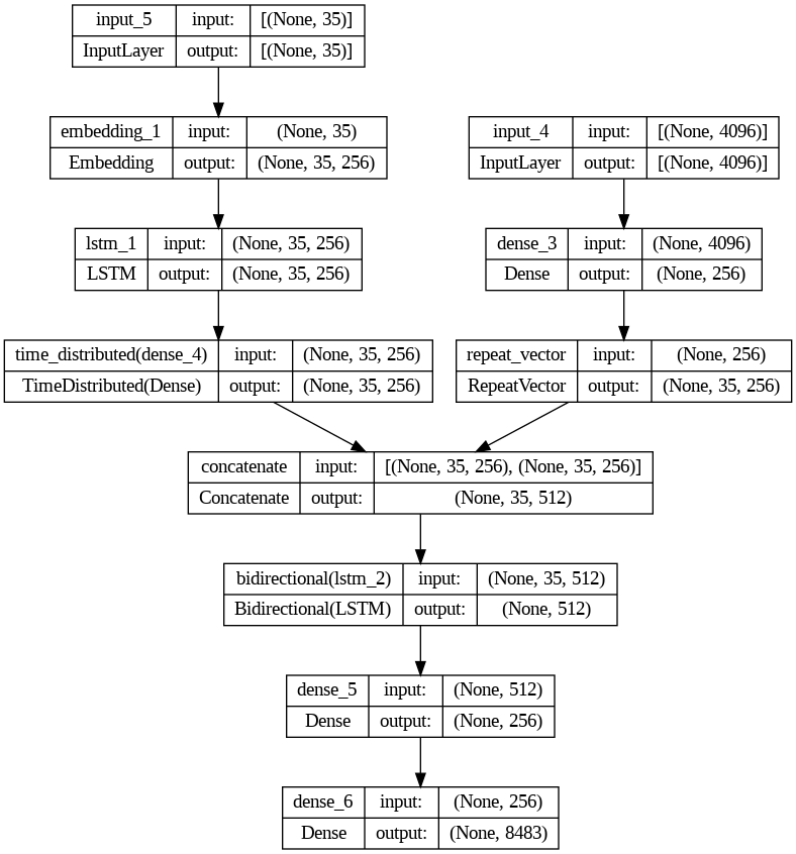


Figure 8 - CNN+ Bi directional Model Architecture

Code

```
from keras.layers import Input, Dense, Embedding, LSTM, RepeatVector,
TimeDistributed, Bidirectional, concatenate
```

```
# define the captioning model2
def RNN_model_Bidirect(vocab_size, max_length, embedding_size, units, input_size):
    image_input = Input(shape=(input_size,))
    image_model_1 = Dense(embedding_size, activation='relu')(image_input)
    image_model = RepeatVector(max_length)(image_model_1)

    caption_input = Input(shape=(max_length,))
    # mask_zero: We zero pad inputs to the same length, the zero mask ignores those
inputs
    caption_model_1 = Embedding(vocab_size, embedding_size,
mask_zero=True)(caption_input)
    # Since we are going to predict the next word using the previous words, we have
to set return_sequences = True.
    caption_model_2 = LSTM(units, return_sequences=True)(caption_model_1)
    caption_model = TimeDistributed(Dense(embedding_size))(caption_model_2)
```

```
    # Merging the models and creating a softmax classifier
    final_model_1 = concatenate([image_model, caption_model])
    final_model_2 = Bidirectional(LSTM(units, return_sequences=False))(final_model_1)
    final_model_3 = Dense(units, activation='relu')(final_model_2)
    final_model = Dense(vocab_size, activation='softmax')(final_model_3)

    model = Model(inputs=[image_input, caption_input], outputs=final_model)
    model.compile(loss='categorical_crossentropy', optimizer='adam')
    # summarize model
    print(model.summary())
    return model
```

```
# define the model
embedding_size = 256
units = 256
input_size = 4096
finalmodel2 = RNN_model_Bidirect(vocab_size, max_length, embedding_size, units,
input_size)
```

Outputs

```
Model: "model_2"
__________________________________________________________________________________________
_____________
 Layer (type)                Output Shape              Param #   Connected to
==========================================================================================
=============
 input_5 (InputLayer)        [(None, 35)]              0         []

 input_4 (InputLayer)        [(None, 4096)]            0         []

 embedding_1 (Embedding)     (None, 35, 256)           2171648   ['input_5[0][0]']

 dense_3 (Dense)             (None, 256)               1048832   ['input_4[0][0]']

 lstm_1 (LSTM)               (None, 35, 256)           525312
['embedding_1[0][0]']

 repeat_vector (RepeatVecto  (None, 35, 256)           0         ['dense_3[0][0]']
 r)

 time_distributed (TimeDist  (None, 35, 256)           65792     ['lstm_1[0][0]']
 ributed)

 concatenate (Concatenate)   (None, 35, 512)           0
['repeat_vector[0][0]',

'time_distributed[0][0]']

 bidirectional (Bidirection  (None, 512)               1574912
['concatenate[0][0]']
```

```
 al)

 dense_5 (Dense)             (None, 256)               131328
['bidirectional[0][0]']

 dense_6 (Dense)             (None, 8483)              2180131   ['dense_5[0][0]']

==================================================================================
=============
Total params: 7697955 (29.37 MB)
Trainable params: 7697955 (29.37 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________
_____________
None
```

4.Data Initialization And Rearrangement For Model Training

Code

```python
# Create sequences of images, input sequences and output words for an image
def create_sequences(tokenizer, max_length, captions_list, image):
    # In1 : input for image features
    # In2 : input for text features
    # Out  : output word
    In1, In2, Out = list(), list(), list()
    vocab_size = len(tokenizer.word_index) + 1
    # loop each caption for the image
    for caption in captions_list:
        # Encode the sequence
        seq = tokenizer.texts_to_sequences([caption])[0]
        # Split one sequence into multiple In1,Out pairs
        for i in range(1, len(seq)):
            # Split into input and output pair
            in_seq, out_seq = seq[:i], seq[i]
            # Pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            # Encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # Store data
            In1.append(image)
            In2.append(in_seq)
            Out.append(out_seq)
    return In1, In2, Out
```

```python
# Data generator function  to be used in a call to model.fit()
def data_process(images, captions, tokenizer, max_length, batch_size, random_seed):
    # Setting random seed for reproducibility of results
    random.seed(random_seed)
    # Image ids
```

```
    image_ids = list(captions.keys())
    _count=0
    while True:
        if _count >= len(image_ids):
            # restart
            _count = 0
        # Batch list to store data
        input_img_batch, input_sequence_batch, output_word_batch = list(), list(),
list()
        for i in range(_count, min(len(image_ids), _count+batch_size)):

            image_id = image_ids[i]

            image = images[image_id][0]
            # Retrieve the captions list
            captions_list = captions[image_id]
            # Shuffle captions list
            random.shuffle(captions_list)
            input_img, input_sequence, output_word = create_sequences(tokenizer,
max_length, captions_list, image)
            # Add to batch
            for j in range(len(input_img)):
                input_img_batch.append(input_img[j])
                input_sequence_batch.append(input_sequence[j])
                output_word_batch.append(output_word[j])
        _count = _count + batch_size
        yield ([np.array(input_img_batch), np.array(input_sequence_batch)],
np.array(output_word_batch))
```

```
#  train data generator
generator_train = data_process(train_features, train_captions, tokenizer, max_length,
batch_size, random_seed='1000')
# test data generator
generator_val = data_process(test_features, test_captions, tokenizer, max_length,
batch_size, random_seed='1000')
```

```
# define parameters
num_of_epochs = 10
batch_size = 32
train_length = len(train)
val_length = len(test)
steps_train = train_length // batch_size
if train_length % batch_size != 0:
    steps_train = steps_train+1
steps_val = val_length // batch_size
if val_length % batch_size != 0:
```

```
    steps_val = steps_val+1
```

Introduce Early Stopping

Early stopping is a regularization strategy helped to prevent overfitting when training neural networks. It involves monitoring the model's performance on validation set during training and stop the training process once the model performance start to perform poorly.

Code

```
from keras.callbacks import ModelCheckpoint
from keras.callbacks import EarlyStopping
# define checkpoint callback
filepath = '/content/drive/MyDrive/ML/RNN_model_UniDirectLSTM_vgg16.h5'
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1,
save_best_only=True, mode='min')
# define early stopping callback
early = EarlyStopping(patience=5, verbose=1)
```

5.Model Training

Model 1 Training

```
# Fit
history = finalmodel.fit(generator_train,
            epochs=num_of_epochs,
            steps_per_epoch=steps_train,
            validation_data=generator_val,
            validation_steps=steps_val,
            callbacks=[checkpoint, early],
            verbose=1)
```

Model 2 Training

```
# Fit
history = finalmodel2.fit(generator_train,
            epochs=num_of_epochs,
            steps_per_epoch=steps_train,
            validation_data=generator_val,
            validation_steps=steps_val,
            callbacks=[checkpoint, early],
            verbose=1)
```

6.Model Performance Plot

During the model training step, we have stored the training and testing losses, In later steps, plotted graph of both the training and validation losses against epochs for Both models.

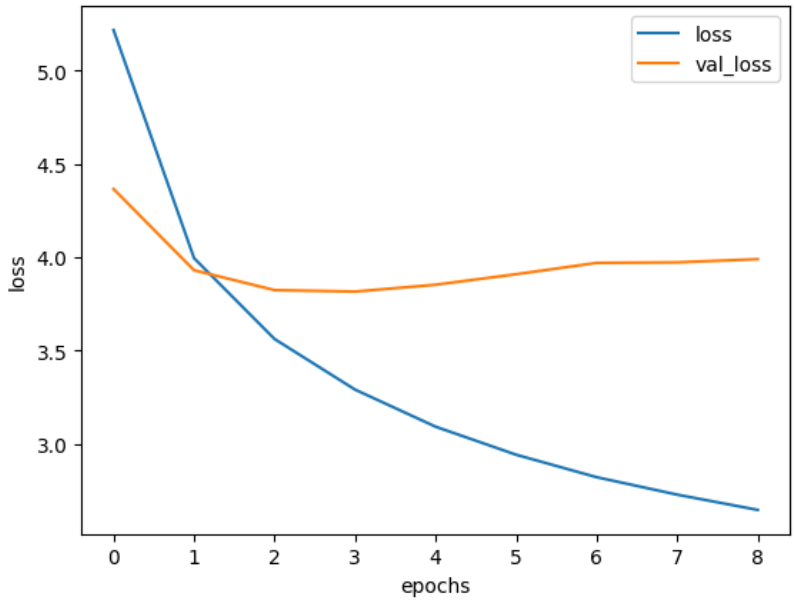


Figure 9 - Training and Validation Loss Curves For Model 1

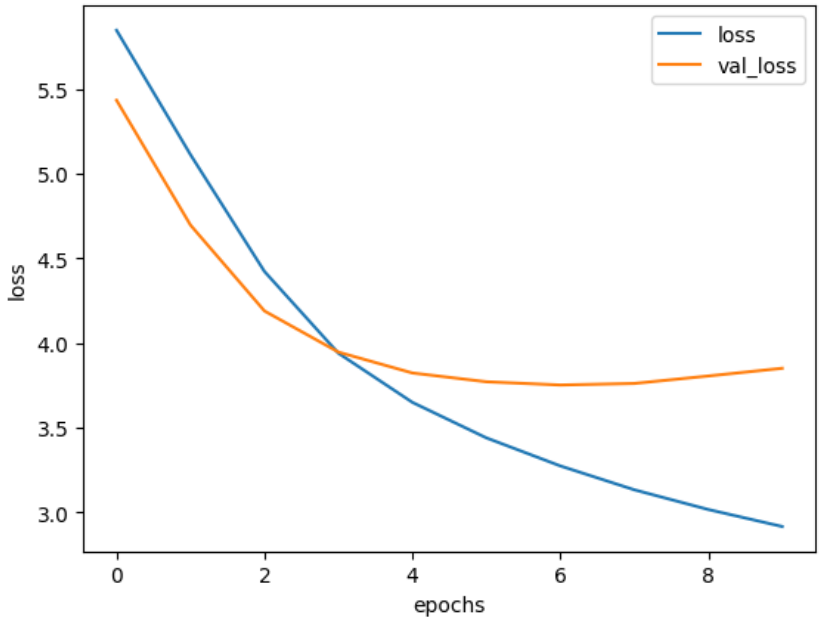


Figure 10 - Training and Validation Loss Curves For Model 2

7.Evaluate Model with BLEU Score

BLEU Score

In Our Case We used the BLEU (Bilingual Evaluation Understudy) score as the metric to evaluate the quality of text generated by each model . It measures how many n- grams in the generated text matched with reference text. We have used two BLEU score values for evaluate models.

- Unigram Precision (BLEU-1): Measures the overlap of individual words (unigrams). A high BLEU-1 score indicates that the generated text contains the correct words.
- Bigram Precision (BLEU-2): Measures the overlap of two-word sequences (bigrams). A high BLEU-2 score indicates that the generated text maintains some correct word sequences.

Code

```
from nltk.translate.bleu_score import corpus_bleu, SmoothingFunction
```

```
def calculate_scores(actual, predicted):
    # calculate BLEU score
    smooth = SmoothingFunction().method4
    bleu1 = corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0),
smoothing_function=smooth)*100
    bleu2 = corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0),
smoothing_function=smooth)*100

    print('BLEU-1: %f' % bleu1)
    print('BLEU-2: %f' % bleu2)
```

```
# evaluate the model
def evaluate_model(model, descriptions, features, tokenizer, max_length):
    actual, predicted = list(), list()
    # step over the whole set
    for key, desc_list in tqdm(descriptions.items(), position=0, leave=True):
        # generate description
        yhat = predict_caption(model, features[key], tokenizer, max_length)
        # store actual and predicted
        references = [d.split() for d in desc_list]
        actual.append(references)
        predicted.append(yhat.split())
    print('Blue Score:')
    calculate_scores(actual, predicted)
```

Model 01

```
# load the model
filepath = '/content/drive/MyDrive/ML/RNN_model_UniDirectLSTM_vgg16.h5'
Caption_Generator_Model = load_model(filepath)
# evaluate model
evaluate_model(Caption_Generator_Model, test_captions, test_features, tokenizer,
max_length)
```

Output

```
100%|■■■■■■■■■■| 810/810 [10:45<00:00, 1.25it/s]
Blue Score:
BLEU-1: 45.085770
BLEU-2: 27.208923
```

Model 02

```
filepath = '/content/drive/MyDrive/ML/RNN_model_bidirect_vgg16.h5'
Caption_Generator_Model2 = load_model(filepath)

evaluate_model(Caption_Generator_Model2, test_captions, test_features, tokenizer,
max_length)
```

Output

```
100%|■■■■■■■■■■| 810/810 [13:03<00:00, 1.03it/s]
Blue Score:
BLEU-1: 47.245241
BLEU-2: 28.995790
```

8.Generate Predictions

Define Functions For Model Prediction

Code

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

```
# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = '<start>'
    # 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 reach end tag
        if word == 'end':
            break
    return in_text
```

```
from PIL import Image
import matplotlib.pyplot as plt
def captionGenerator(image_name):
    # load the image
    image_id = image_name.split('.')[0]
    img_path =  os.path.join(IMAGES_DIR, 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(finalmodel,  features[image_id],tokenizer , max_length)
    y_pred = cleanCap(y_pred)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
```

```
def cleanCap(in_text):
    words = in_text.split()
    if len(words) > 2:
        return ' '.join(words[1:-1])
    else:
        return ''
```

Generate Captions For Random Selected Test Data

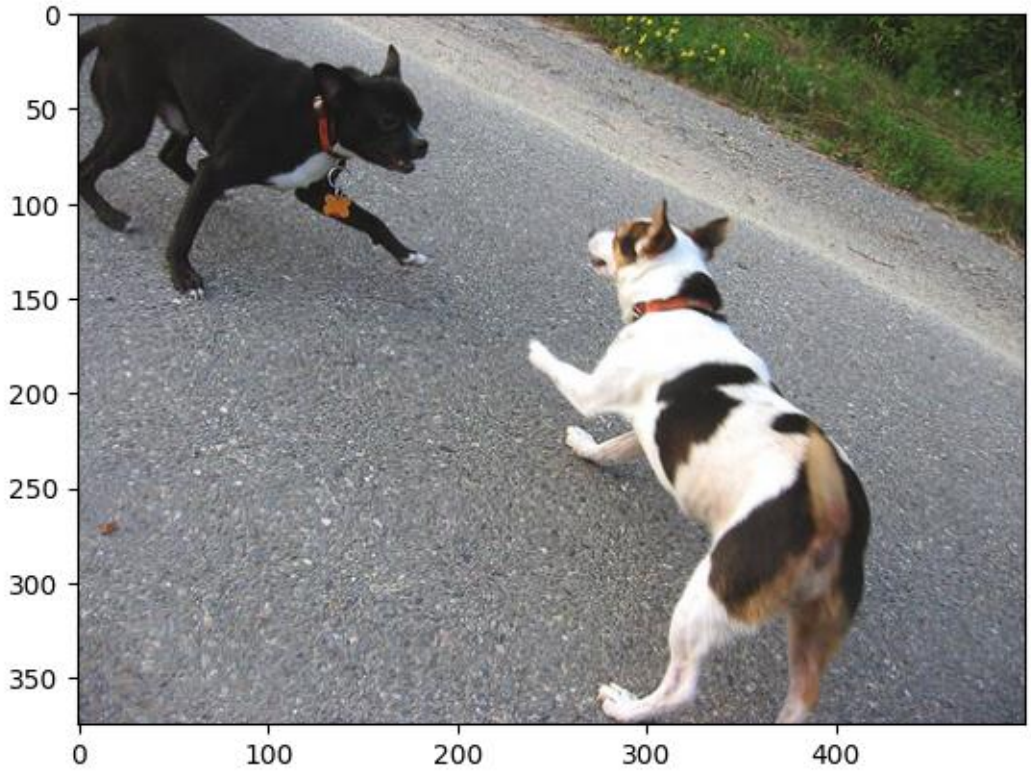
Model 01

```
captionGenerator("1001773457_577c3a7d70.jpg")
captionGenerator("1002674143_1b742ab4b8.jpg")
captionGenerator("1237985362_dbafc59280.jpg")
captionGenerator("1131932671_c8d17751b3.jpg")
captionGenerator("1234817607_924893f6e1.jpg")
```

Sample Output

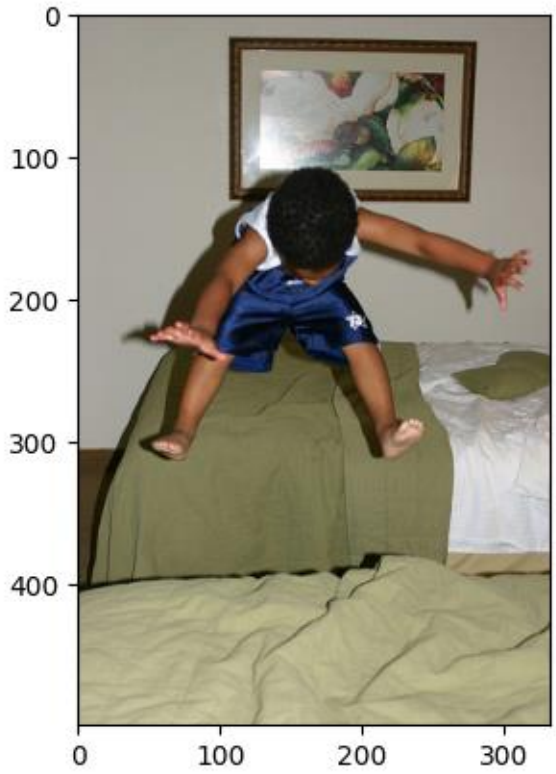
1001773457_577c3a7d70.jpg

```
----------------------Actual----------------------
<start> black dog and spotted dog are fighting <end>
<start> two dogs on pavement moving toward each other <end>
<start> black dog and tri-colored dog playing with each other on the road <end>
<start> two dogs of different breeds looking at each other on the road <end>
<start> black dog and white dog with brown spots are staring at each other in the street
<end>
--------------------Predicted--------------------
two dogs are playing with red toy in the grass
```



1131932671_c8d17751b3.jpg

```
----------------------Actual----------------------
<start> boy jumped up from the green bed <end>
<start> small child is jumping on bed <end>
<start> the boy in blue shorts is bouncing on the bed <end>
<start> boy is jumping on bed <end>
<start> boy jumps from one bed to another <end>
--------------------Predicted--------------------
young boy jumping on bed
```



1234817607_924893f6e1.jpg

```
----------------------Actual----------------------
<start> an elderly man is smiling while sitting in front of row of soda cans <end>
<start> man wearing glasses with aluminum cans lined up in front of him <end>
<start> man is sitting at an outside bar near many soda and beer cans <end>
<start> vendor selling drinks in stall <end>
<start> korean man sells soda <end>
--------------------Predicted--------------------
korean man sells soda cans
```

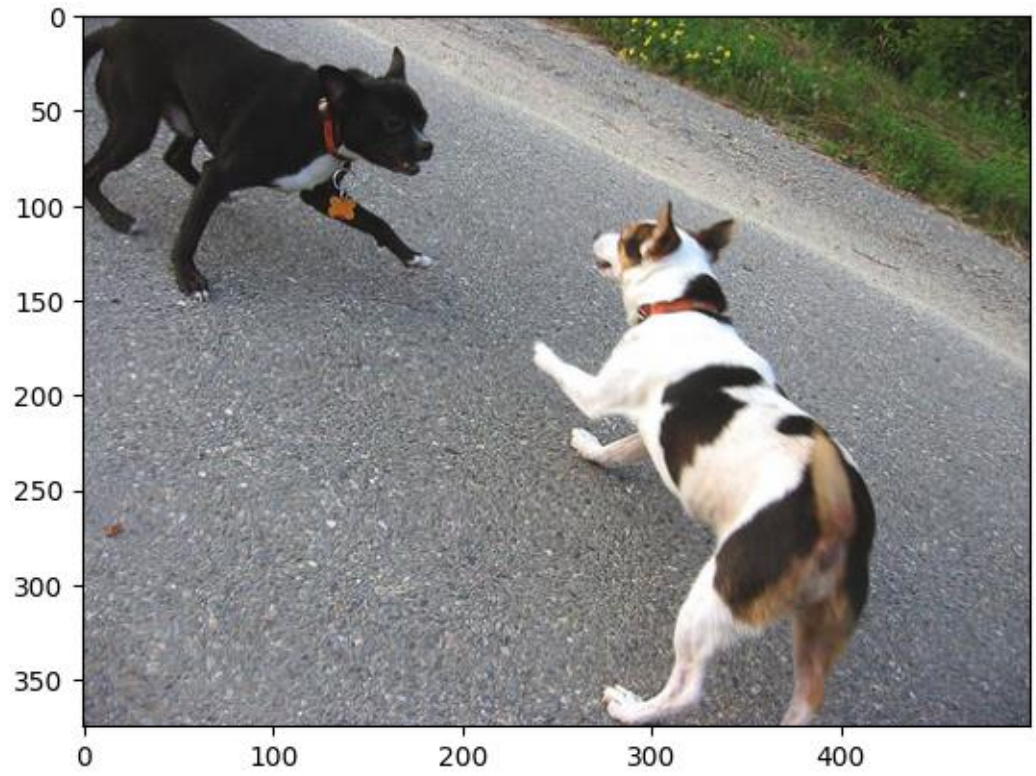
Model 02

```
captionGenerator2("1001773457_577c3a7d70.jpg")
captionGenerator2("1002674143_1b742ab4b8.jpg")
captionGenerator2("1237985362_dbafc59280.jpg")
captionGenerator2("1131932671_c8d17751b3.jpg")
captionGenerator2("1234817607_924893f6e1.jpg")
```

Sample Output

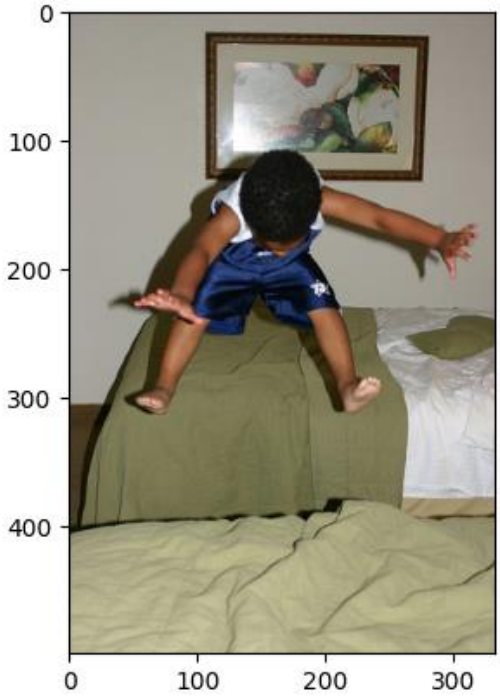
1001773457_577c3a7d70.jpg

```
----------------------Actual----------------------
<start> two dogs on pavement moving toward each other <end>
<start> two dogs of different breeds looking at each other on the road <end>
<start> black dog and spotted dog are fighting <end>
<start> black dog and white dog with brown spots are staring at each other in the street
<end>
<start> black dog and tri-colored dog playing with each other on the road <end>
--------------------Predicted--------------------
two dogs are playing in the grass
```



1131932671_c8d17751b3.jpg

```
----------------------Actual----------------------
<start> boy jumps from one bed to another <end>
<start> boy jumped up from the green bed <end>
<start> small child is jumping on bed <end>
<start> the boy in blue shorts is bouncing on the bed <end>
<start> boy is jumping on bed <end>
--------------------Predicted--------------------
the boy is jumping on the slide
```



1234817607_924893f6e1.jpg

```
----------------------Actual----------------------
<start> vendor selling drinks in stall <end>
<start> man wearing glasses with aluminum cans lined up in front of him <end>
<start> korean man sells soda <end>
<start> an elderly man is smiling while sitting in front of row of soda cans <end>
<start> man is sitting at an outside bar near many soda and beer cans <end>
--------------------Predicted--------------------
man in black and white shirt is sitting in front of the building
```

Generate captions for images imported from outside of the dataset

Code

```
def preprocess_image(image_path):
    # Load the image with the required target size
    image = load_img(image_path, target_size=(224, 224))
    # Convert the image to an array
    image = img_to_array(image)
    # Reshape data for the model
    image = np.expand_dims(image, axis=0)
    # Prepare the image for the InceptionV3 model
    image = preprocess_input(image)
    return image
```

```
def extract_features(image_path, model):
    image = preprocess_image(image_path)
    features = model.predict(image, verbose=0)
    return features
```

```
# Path to the image
image_path = '/content/drive/MyDrive/family-creating-digital-content.jpg'

# Extract features
photo = extract_features(image_path, model)
```

```
caption = predict_caption(Caption_Generator_Model,photo, tokenizer , max_length=35)
clean_caption = cleanCap(caption)
print('Generated Caption:', clean_caption)
```

Output

Model 01

```
Generated Caption: two girls dressed in red costume are playing in the air
```

Model 2

```
Generated Caption: two women in costume are smiling
```

9.Model Comparison

In this Section We have Compared the performances of this two model . According to the Observations we have gathered, One Factor For the comparison of two models training and validation losses .

Table 1 - Training Loss and Validation Loss Of Models

	Uni Directional LSTM based Model	Bi directional LSTM based Model
Training Loss	2.6455	2.9150
Validation Loss	3.9896	3.8038

The validation loss for the Bidirectional LSTM is lower than that of the Uni-directional LSTM, suggesting that the Bidirectional LSTM model generalizes better to the validation data.

The shape of the epoch vs. training loss and validation loss curves for the Bidirectional LSTM indicates a more general form and better performance.(Graphs Are Shown in Figure 9 and Figure 10)

And Another Matrix we have chosen for evaluation the model is BLEU Score. The BLEU score results of Both models are Shown in following Table

Table 2 - BLEU score for models

	Uni Directional LSTM based Model	Uni Directional LSTM based Model
BLEU-1	45.085770	47.245241
BLEU-2	27.208923	28.995790

Bidirectional LSTM model produced higher BLEU score than Uni-directional LSTM model also. To the Bidirectional LSTM to be exact, the BLEU-1 score achieved is 47. 245241 compared to total loss value is 45.085770 for the Uni-directional LSTM. Again the BLEU-2 score for the proposed Bidirectional LSTM is 28.

995790 compared to 27. 59322 for bi-directional LSTM, while the Uni-directional LSTM is equal to 208923 . Thus, the Bidirectional LSTM model yields higher BLEU scores suggesting better accuracy and relevancy of the captions.

Analyzing the graphs of training and validation losses, as well as BLEU scores it can been seen that the Bidirectional LSTM model is better than Uni-directional LSTM model.

10.Conclusion

In this project, we got down to the challenge of creating and comparing image caption generator models using VGG16 pre-trained model as the feature extractor model and Uni-directional LSTM as well as Bidirectional LSTM as the generator models. Our aim was to analyze the efficiency of these models for providing the appropriate and correct descriptions of images.

For this task, we used the Image caption generator Dataset where we converted into numpy format and for the feature extraction of the images, we used VGG16 by taking the help of which we were able to get the pre trained features which helped the LSTM networks to work in an efficient way. We built two models: one with Uni-directional LSTM and the other with Bidirectional LSTM and their performance comparison was done with the help of training and validation loss and the BLEU score.

Finally, the training loss of the selected model , the Uni-directional LSTM is showed a training loss of 2. 6455, and the validation loss was 3. 9896. Consequently, its BLEU-1 and BLEU-2 scores were 45. 085770 and 27. 208923, respectively. However, training outcomes of Bidirectional LSTM model are slightly low achieving a training loss of 2.9150 with a validation loss of 3. 8038, while BLEU values for BLEU-1 and BLEU-2 came out to be 47. 245241 and 28.995790, respectively. Hence, the Bidirectional LSTM model was more generalizable with the lowest validation loss, which also pointed to more accurate and relevant captions as supported by the above higher BLEU scores.

In summary, the project was aimed to present how to construct and assess the image caption generator models. After comparing both the models the Bidirectional LSTM model was selected as the best model as it was observed that it has a relatively less loss as compared to the Uni-directional LSTM model and has higher BLEU scores. This asserts that integrating the bidirectional context in LSTM networks enhances the understanding as well as the generation of captions for images.