```
import warnings
warnings.filterwarnings("ignore")
import numpy as np
from numpy import array
import pandas as pd
from tqdm import tqdm
import datetime as datetime
import tensorflow as tf
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from wordcloud import WordCloud
import glob
import string
import cv2
import random
import imgaug.augmenters as iaa
from pickle import dump, load
from keras.preprocessing import image
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import VGG16, preprocess_input
from keras.layers import LSTM, Embedding, Dense, Activation, Dropout, Bidirectional, Flatten
from keras_self_attention import SeqSelfAttention
from keras.callbacks import ModelCheckpoint, ModelCheckpoint, TensorBoard
from keras.layers.merge import Concatenate
from keras.preprocessing import sequence
from keras.models import Model
from keras import Input, layers
from keras import optimizers
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.translate.bleu_score import sentence_bleu
tf.random.set_seed(3)
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
image_path = '/content/Flicker8k_Dataset/'
dir_Flickr_text = '/content/Flickr_txt/Flickr8k.token.txt'
images_lst = glob.glob(image_path + '*.jpg')
no_of_image = len(images_lst)
print(f'Total Images in Datset : {no_of_image}')
```

```
Total Images in Datset : 8091
```

Utility Fuctions:

- **Load text**
- **Preparing Text data from Flickr8k.txt file**
- **Prepare DataFrames**

```
def load_txt(txt_file):
  with open(txt_file,'r') as file:
    text = file.read()
  return text

def prepare_clean_txt(text):
  text_data = []
  table = str.maketrans('', '', string.punctuation)
  for line in tqdm(text.split('\n')):                          # spiltting over each line
    col = line.split('\t')                                     # splitting over each space ir
    if len(col) == 1:                                          # neglecting data if full infc
        continue
    filename, index = col[0].split("#")
    filename_path = image_path + filename                      # splitting to gather informat
    txt = col[1].split()
    txt = [word.lower() for word in txt]                       # convert each word to lower c
    txt = [w.translate(table) for w in txt]                    # to remove punctuations from
    txt = [word for word in txt if len(word)>1]                # to discard words of length 1
    txt = [word for word in txt if word.isalpha()]             # to discard numeric string
    txt = ' '.join(txt)
    text_data.append([index]+ [filename_path] + [txt])
  return text_data                                              # [index, filename, caption t


def prepare_dataframes(text_data):
  data_df = pd.DataFrame(text_data, columns=[ 'index', 'filename', 'caption'])
  data_df = data_df[data_df.filename != '2258277193_586949ec62.jpg.1']
  data_df['caption_with_tags'] = data_df['caption'].apply(lambda txt : 'startseq ' + txt + '
  tokenizer = Tokenizer()
  tokenizer.fit_on_texts(data_df.caption.values)
  data_df['caption_word_count'] = data_df['caption'].apply(lambda caption: len(caption.split(
  word = []
  frequency = []
  for k,v in tokenizer.word_counts.items():
    word.append(k)
    frequency.append(v)
  df_word_count = pd.DataFrame({'word': word, 'frequency': frequency})
  df_word_count.sort_values('frequency', ascending=False, inplace=True, ignore_index=True)
  return data_df, df_word_count


text = load_txt(dir_Flickr_text)
text_data = prepare_clean_txt(text)
data_df, df_word_count = prepare_dataframes(text_data)
```

```
100%|██████████| 40461/40461 [00:00<00:00, 68995.80it/s]
```

```
data_df.head(3)
```

	index	filename	caption	caption_with_tags	cap
0	0	/content/Flicker8k_Dataset/1000268201_693b08cb...	child in pink dress is climbing up set of	startseq child in pink dress is climbing up se...	

Exploratory Data Analysis

```
from collections import Counter
def utility_counter(data):
  filename = data_df.filename.values
  caption_count = Counter(filename).values()
  uni_filenames = np.unique(filename)
  print(f'Number of unique file names : {len(uni_filenames)}')
  print(f'Number of captions per image : {set(caption_count)}')
  print(f'All image have same number of captions: {all(caption_count)}')

utility_counter(data_df)
```

```
Number of unique file names : 8092
Number of captions per image : {5}
All image have same number of captions: True
```

Image - Caption visualization

```
def image_caption_plotter(data):
  npic = 3
  target_size = (350, 500, 3)
  count = 1
  fig = plt.figure(figsize=(25, 22))
  for file in tqdm(random.sample(list(data['filename']), npic)):
    captions = list(data['caption'].loc[data['filename'] == file].values)
    image_load = load_img(file, target_size = target_size)
    ax = fig.add_subplot(npic, 2, count)
    ax.axis('off')
    ax.imshow(image_load, interpolation='nearest')
    count += 1
    ax = fig.add_subplot(npic, 2, count)
    ax.axis('off')
    ax.set_ylim(0, len(captions))
    for i, caption in enumerate(captions):
```

```
          ax.text(0, i, caption, fontsize = 30, fontfamily= 'fantasy', bbox=dict(facecolor='black
        count += 1
        plt.grid(None)
      plt.show()

image_caption_plotter(data_df)
```

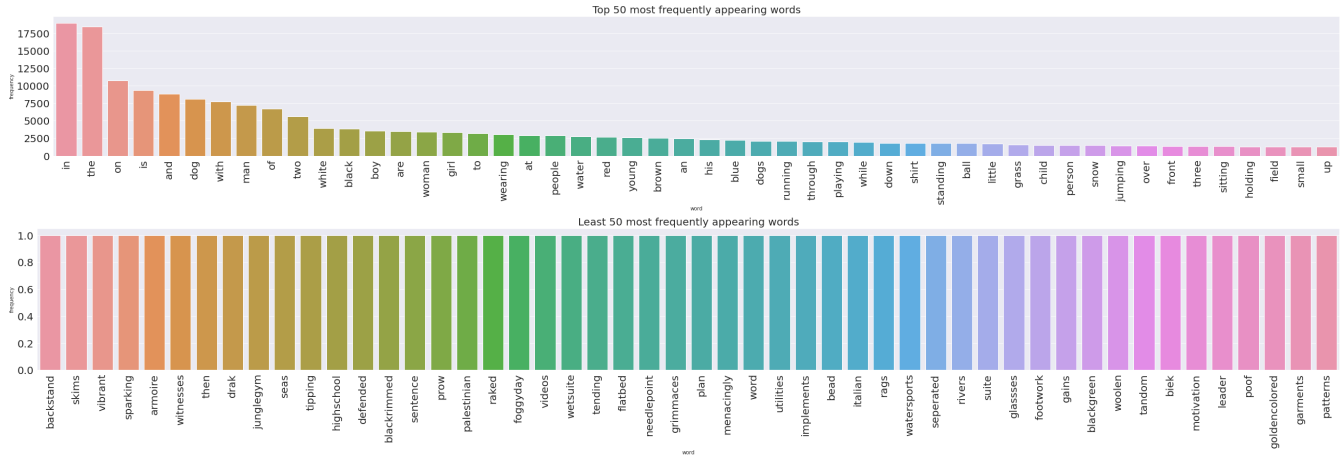
```
100%|██████████| 3/3 [00:00<00:00, 25.84it/s]
```

Observation

- We can see each image has 5 captions associated with it.

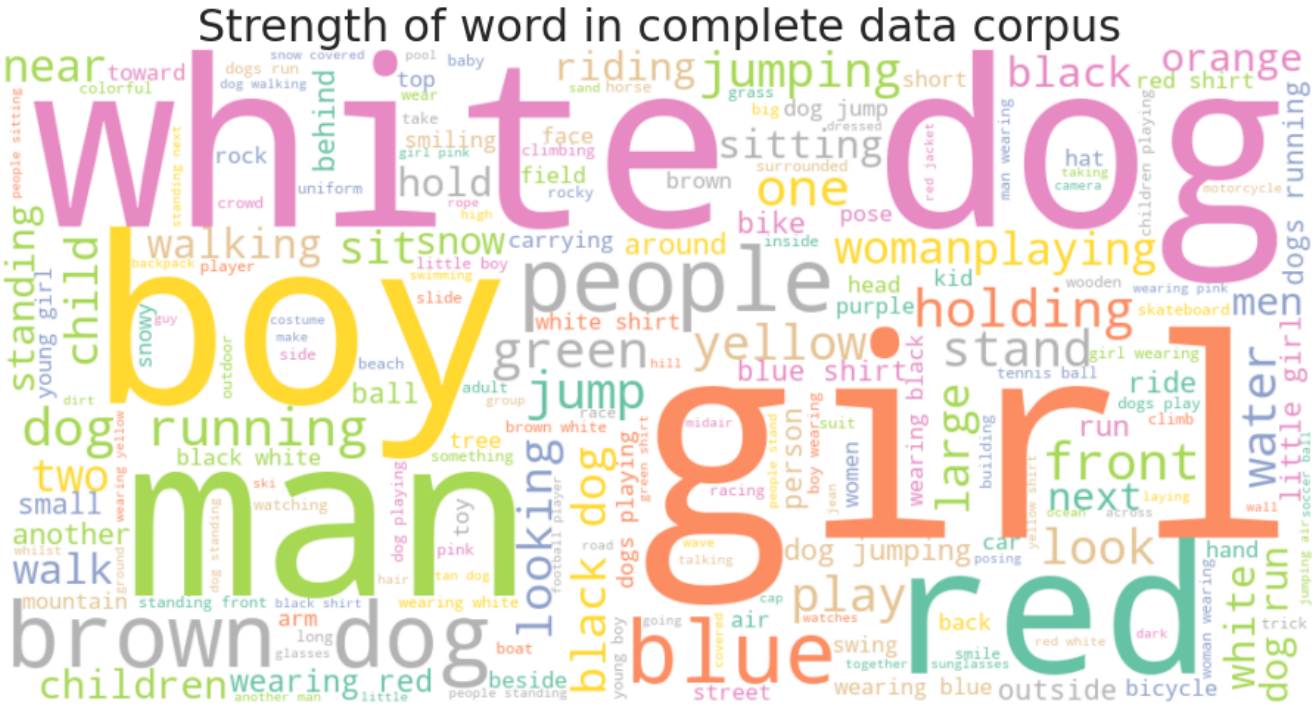
Plot for word - frequency

```
count = 50

def plot_word_count(df, title = ''):
    sns.set_style('darkgrid')
    plt.figure(figsize = (45, 5))
    sns.barplot(x=df['word'], y=df["frequency"])
    plt.yticks(fontsize = 20)
    plt.xticks( rotation = 'vertical', fontsize = 20)
    plt.title(title, fontsize = 20)
    plt.show()

plot_word_count(df_word_count.iloc[:count, :], title= 'Top 50 most frequently appearing words
plot_word_count(df_word_count.iloc[-count:, :], title= 'Least 50 most frequently appearing wo
```



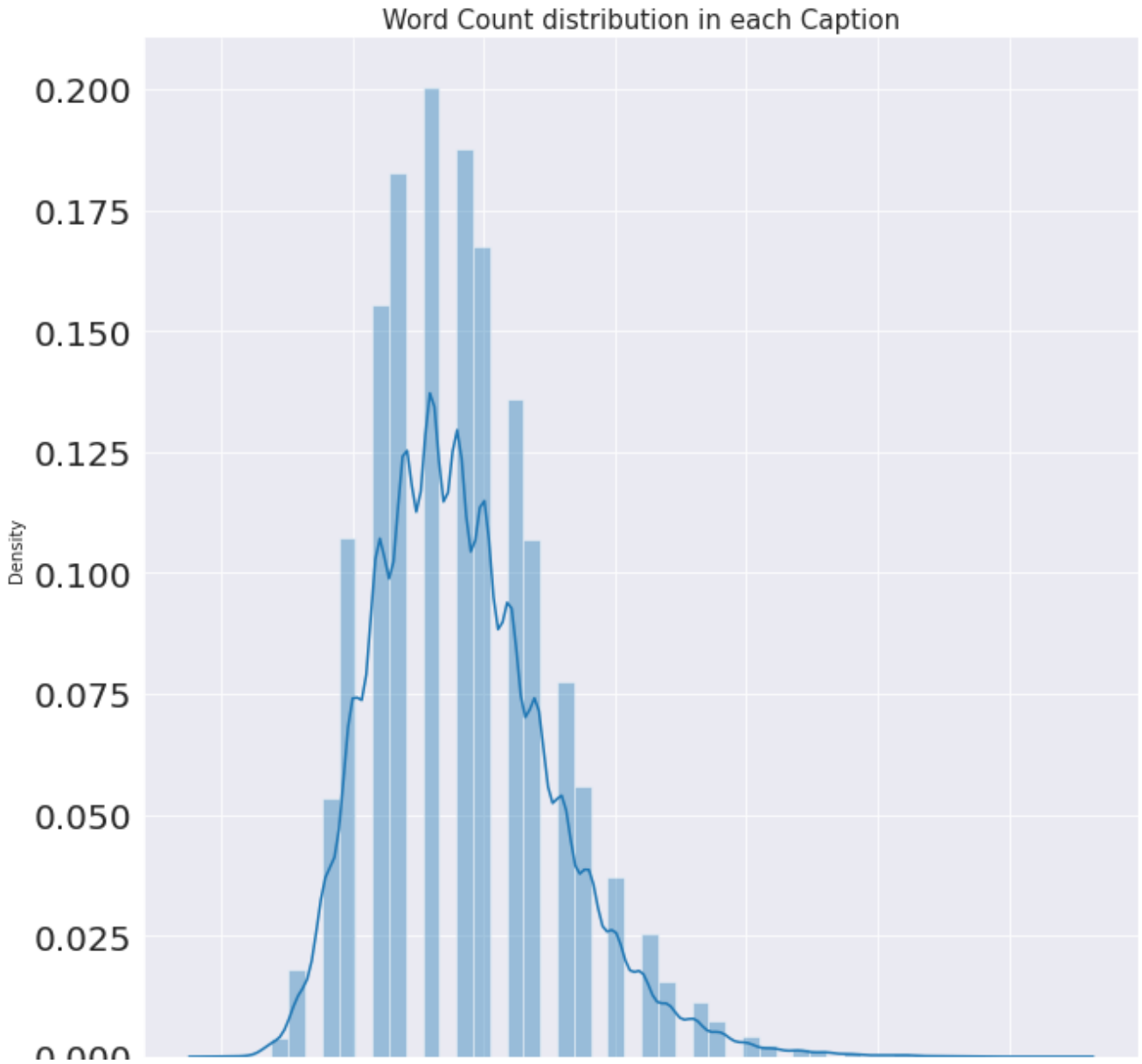
Word Cloud Visualization

```
text = ' '
for caption in data_df.caption:
  text = text + caption
plt.figure(figsize=(16, 12))
word_cloud = WordCloud(width = 1000, height = 500, random_state=1, collocations= True, backgr
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.title('Strength of word in complete data corpus', fontsize = 30)
plt.show()
```



Distribution of number of words in each caption

```
sns.set_style('darkgrid')
sns.FacetGrid(data_df, height = 9).map(sns.distplot, 'caption_word_count').add_legend()
plt.yticks(fontsize = 20)
plt.xticks(fontsize = 20)
plt.title('Word Count distribution in each Caption', fontsize = 15)
plt.show()
```



Observation:

- We can see most of captions contain words in range of 7 to 10

- **Train_filename_container : Train image path**
- Test_filename_container : Test image path

```
train_images_file = '/content/Flickr_txt/Flickr_8k.trainImages.txt'
train_images = set(open(train_images_file, 'r').read().strip().split('\n'))
train_filename_container = [image_path+filename for filename in train_images if image_path+fi

validation_images_file = '/content/Flickr_txt/Flickr_8k.devImages.txt'
validation_images = set(open(validation_images_file, 'r').read().strip().split('\n'))
validation_filename_container = [image_path+filename for filename in validation_images if ima

train_filename_container = train_filename_container+validation_filename_container
```

```
test_images_file = '/content/Flickr_txt/Flickr_8k.testImages.txt'
test_images = set(open(test_images_file, 'r').read().strip().split('\n'))
test_filename_container = [image_path+filename for filename in test_images if image_path+file
```

Augumentation on images

```
def augument_image(image_path, aug=True, visualize=False):
  if visualize:
    image = cv2.imread(image_path, cv2.IMREAD_UNCHANGED)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, (224, 224))
  else:
    image = load_img(image_path, target_size=(224, 224))
    image = img_to_array(image)
  if aug:
    aug2 = iaa.Fliplr(1)
    aug3 = iaa.Flipud(1)
    aug4 = iaa.Emboss(alpha=(1), strength=1)
    aug5 = iaa.DirectedEdgeDetect(alpha=(0.8), direction=(1.0))
    aug6 = iaa.Sharpen(alpha=(1.0), lightness=(1.5))

    a = np.random.uniform()
    if a<0.2:
      aug_image = aug2.augment_image(image)
    elif a<0.4:
      aug_image = aug3.augment_image(image)
    elif a<0.6:
      aug_image = aug4.augment_image(image)
    elif a<0.8:
      aug_image = aug5.augment_image(image)
    else:
      aug_image = aug6.augment_image(image)
    return image, aug_image
  else:
    return image



row = 3
col = 2
dataset = [augument_image(train_filename_container[np.random.randint(0, 5000)], aug=True, vis
fig = plt.figure(figsize=(15, 20))
for counter in range(0, row*col):
  fig.add_subplot(row, col, counter+1)
  plt.imshow((dataset[int(counter/2)][counter%2]))
  plt.axis('off')
  plt.grid(None)
```

- train_caption_mapping: Train image caption
- test_caption_mapping: Train image caption

```
train_caption_mapping = {}
for file in tqdm(train_filename_container):
    train_caption_mapping[file]= list(data_df.loc[data_df['filename'] == file].caption_with_t

test_caption_mapping = {}
for file in tqdm(test_filename_container):
    test_caption_mapping[file]= list(data_df.loc[data_df['filename'] == file].caption_with_ta
```

```
 100%|██████████| 7000/7000 [00:22<00:00, 315.84it/s]
 100%|██████████| 1000/1000 [00:03<00:00, 300.38it/s]
```

Model load VGG16

```
def encode_image(filename_container, aug=True):
  encoding = {}
  augmented_image_path = []
  for image_path in tqdm(filename_container):
    image = augument_image(image_path, aug=aug)
    if aug:
      image = preprocess_input(image[0])
      aug_image = preprocess_input(image[1])

      features = model_new(np.expand_dims(image, axis=0))
      features = np.reshape(features, (features.shape[1],))
      features_aug = model_new(np.expand_dims(image, axis=0))
      features_aug = np.reshape(features_aug, (features_aug.shape[1],))

      encoding[image_path] = features
      aug_path = image_path[:-4]+'_aug.jpg'
      augmented_image_path.append(aug_path)
      train_caption_mapping[aug_path] = train_caption_mapping[image_path]
      encoding[aug_path] = features_aug
    else:
      image = preprocess_input(image)
      features = model_new(np.expand_dims(image, axis=0))
      fearures = np.reshape(features, (features.shape[1],))
      encoding[image_path] = features
  global train_filename_container
```

```
  train_filename_container += augmented_image_path
  return encoding

model_vgg16 = VGG16(weights='imagenet', include_top=True)
model_new = Model(model_vgg16.input, model_vgg16.layers[-2].output)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg1
553467904/553467096 [==============================] - 5s 0us/step
553476096/553467096 [==============================] - 5s 0us/step
```

Save the bottleneck train and test features to disk

```
encoding_train = encode_image(train_filename_container)
with open('/content/encoded_train_image_features.pkl', 'wb') as p:
    dump(encoding_train, p)
```

```
100%|██████████| 7000/7000 [03:45<00:00, 30.98it/s]
```

```
encoding_test = encode_image(test_filename_container, aug=False)
with open('/content/encoded_test_image_feature.pkl', 'wb') as p:
    dump(encoding_test, p)
```

```
100%|██████████| 1000/1000 [00:16<00:00, 59.20it/s]
```

Loading saved features

```
train_features = load(open("/content/encoded_train_image_features.pkl", "rb"))
print('Number of train image features', len(train_features))

test_features = load(open("/content/encoded_test_image_feature.pkl", "rb"))
print('Number of test image features', len(test_features))
```

```
Number of train image features 14000
Number of test image features 1000
```

List of all the train, test captions

```
all_train_captions = set()
for _, caption_lst in tqdm(train_caption_mapping.items()):
    for caption in caption_lst:
        all_train_captions.add(caption)

all_test_captions = set()
for _, caption_lst in tqdm(test_caption_mapping.items()):
```

```
    for caption in caption_lst:
        all_test_captions.add(caption)
```

```
100%|██████████| 14000/14000 [00:00<00:00, 391009.59it/s]
100%|██████████| 1000/1000 [00:00<00:00, 314038.93it/s]
```

Frequency of words in train, test caption

```
train_tokenizer = Tokenizer(oov_token = '<UNK>')
train_tokenizer.fit_on_texts(all_train_captions)
words = []
frequency = []
for word,freq in tqdm(train_tokenizer.word_counts.items()):
  words.append(word)
  frequency.append(freq)
df_wc_train = pd.DataFrame({'word': words, 'frequency': frequency})
df_wc_train.sort_values('frequency', ascending=False, inplace=True, ignore_index=True)
```

```
100%|██████████| 8169/8169 [00:00<00:00, 1076445.79it/s]
```

Word-Index/Index-Word Mapping

```
ixtoword = dict((index, word) for word, index in train_tokenizer.word_index.items())
wordtoix = train_tokenizer.word_index
max_length = max(map(lambda caption: len(caption.split()), all_train_captions))
vocab_size = len(wordtoix)+1
```

Dataset Preparation/Dataloader

```
def dataset_loader(tokenizer, image_caption_mapping, image_features, max_length, images_per_b
    image_feature, input_seq, output_seq = list(), list(), list()
    image_count=0
    while 1:
        for image_path, captions in image_caption_mapping.items():
            image_count+=1
            feature = image_features[image_path]
            sequences = tokenizer.texts_to_sequences(captions)
            for seq in sequences:
              for i in range(1, len(seq)):
                  in_seq, out_seq = seq[:i], seq[i]
                  in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                  out_seq = tf.keras.utils.to_categorical([out_seq], num_classes=vocab_size)[
                  image_feature.append(feature)
                  input_seq.append(in_seq)
                  output_seq.append(out_seq)
            if image_count==images_per_batch:
                yield [[array(image_feature), array(input_seq)], array(output_seq)]
```

```
image_feature, input_seq, output_seq = list(), list(), list()
image_count=0
```

Model Architecture

```
input_image = Input(shape=(4096, ), name='Image_Feature_input')
fe1 = Dropout(0.5, name='Dropout_image')(input_image)
fe2 = Dense(256, activation='relu', name='Activation_Encoder')(fe1)
input_text = Input(shape=(max_length,), name='Text_input')
se1 = Embedding(vocab_size, 500, mask_zero=True, name='Text_Feature')(input_text)
se2 = Dropout(0.5, name='Dropout_text')(se1)
se3 = Bidirectional(LSTM(512, name='Bidirectional-LSTM', return_sequences=True))(se2)
se4 = SeqSelfAttention(attention_activation='sigmoid', name='Self-Attention')(se3)
se5 = Flatten()(se4)
se6 = Dense(256, activation='relu')(se5)
decoder1 = Concatenate(name='Concatenate')([fe2, se6])
decoder2 = Dense(256, activation='relu', name='Activation_Decoder')(decoder1)
output = Dense(vocab_size, activation='softmax',name='Output')(decoder2)
model = Model(inputs=[input_image, input_text], outputs=output)
```

Model Summary

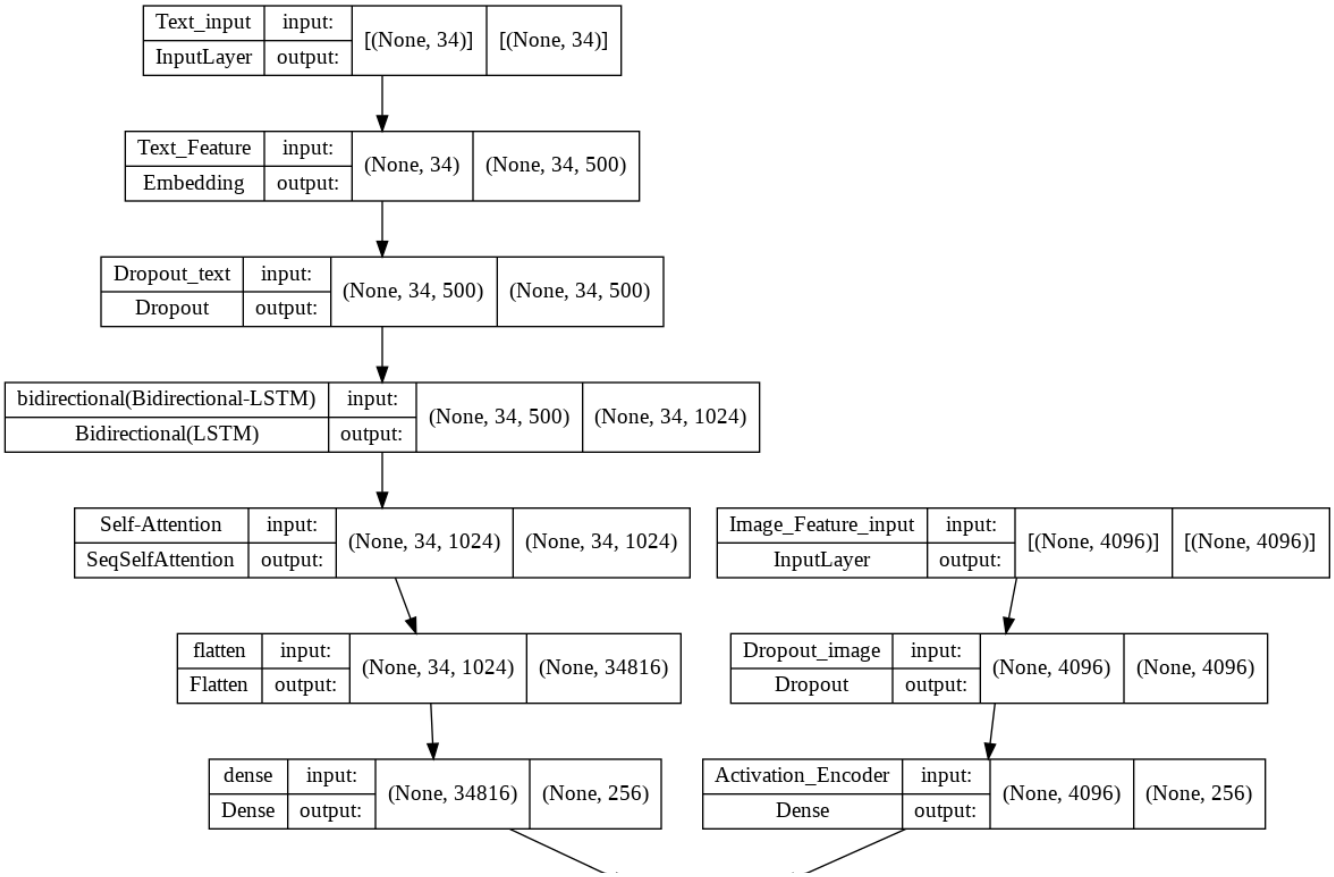
```
model.summary()
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 Text_input (InputLayer)        [(None, 34)]         0           []

 Text_Feature (Embedding)       (None, 34, 500)      4085500     ['Text_input[0][0]']

 Dropout_text (Dropout)         (None, 34, 500)      0           ['Text_Feature[0][0]']

 bidirectional (Bidirectional)  (None, 34, 1024)     4149248     ['Dropout_text[0][0]']

 Image_Feature_input (InputLaye  [(None, 4096)]      0           []
 r)

 Self-Attention (SeqSelfAttenti  (None, 34, 1024)    65601       ['bidirectional[0][0]'
 on)

 Dropout_image (Dropout)        (None, 4096)         0           ['Image_Feature_input[

 flatten (Flatten)              (None, 34816)        0           ['Self-Attention[0][0]

 Activation_Encoder (Dense)     (None, 256)          1048832     ['Dropout_image[0][0]'

 dense (Dense)                  (None, 256)          8913152     ['flatten[0][0]']

 Concatenate (Concatenate)      (None, 512)          0           ['Activation_Encoder[0
```

```
                                                                 'dense[0][0]']

 Activation_Decoder (Dense)     (None, 256)          131328      ['Concatenate[0][0]']

 Output (Dense)                 (None, 8171)         2099947     ['Activation_Decoder[0

==================================================================================================
Total params: 20,493,608
Trainable params: 20,493,608
Non-trainable params: 0
__________________________________________________________________________________________________
```

Model Architecture

```
tf.keras.utils.plot_model(model, to_file='model.png', show_shapes = True, show_layer_names =
```



Compile/Callbacks

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
filepath = 'model-ep{epoch:03d}-loss{loss:.3f}.h5'
checkpoint = ModelCheckpoint(filepath, monitor='loss', verbose=1, save_best_only=True, mode='
log_dir = "logs/fit/"+ datetime.datetime.now().strftime("%Y%m%d-%H%M%S")
tensorboard = TensorBoard(log_dir = log_dir, histogram_freq=1)
```

Model Fitting

```
number_pics_per_bath = 16
steps = len(train_caption_mapping)//number_pics_per_bath
learning_rate = 0.001

train_generator = dataset_loader(train_tokenizer, train_caption_mapping, train_features, max_
history = model.fit(train_generator, epochs=70, steps_per_epoch=steps, verbose=1, callbacks =
```

```
875/875 [==============================] - 349s 398ms/step - loss: 1.0497 - accuracy
Epoch 57/70
875/875 [==============================] - ETA: 0s - loss: 1.0496 - accuracy: 0.6937
Epoch 57: loss improved from 1.04966 to 1.04958, saving model to model-ep057-loss1.0
875/875 [==============================] - 348s 398ms/step - loss: 1.0496 - accuracy
Epoch 58/70
875/875 [==============================] - ETA: 0s - loss: 1.0486 - accuracy: 0.6943
Epoch 58: loss improved from 1.04958 to 1.04858, saving model to model-ep058-loss1.04
875/875 [==============================] - 348s 398ms/step - loss: 1.0486 - accuracy
Epoch 59/70
```

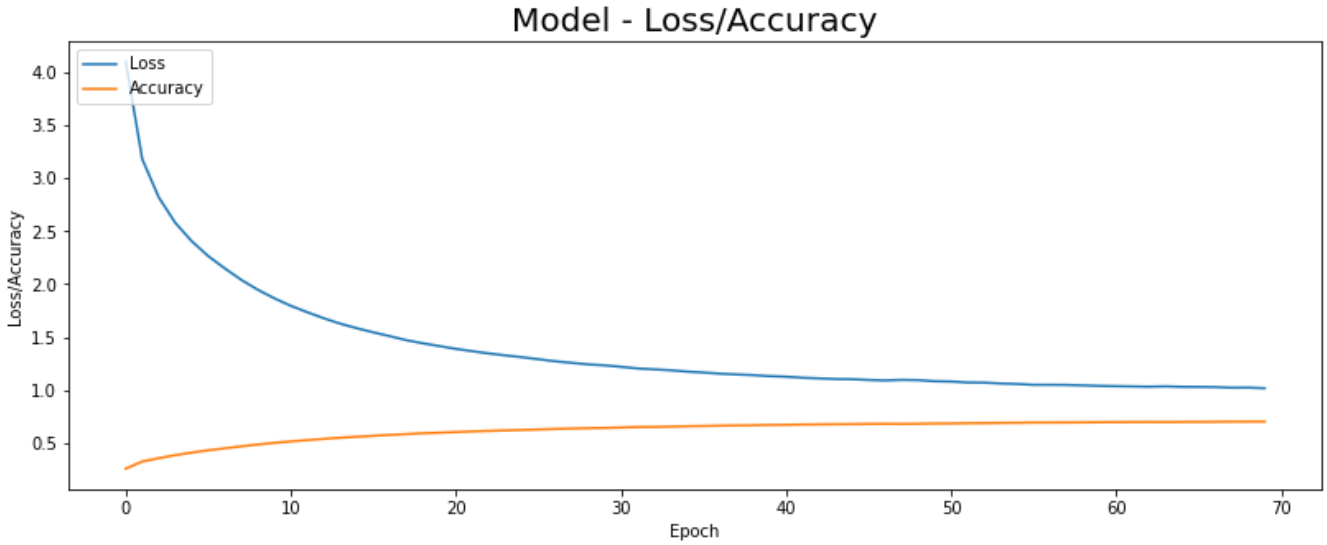
```
Epoch 59/70
875/875 [==============================] - ETA: 0s - loss: 1.0445 - accuracy: 0.6953
Epoch 59: loss improved from 1.04858 to 1.04450, saving model to model-ep059-loss1.04
875/875 [==============================] - 347s 397ms/step - loss: 1.0445 - accuracy
Epoch 60/70
875/875 [==============================] - ETA: 0s - loss: 1.0395 - accuracy: 0.6968
Epoch 60: loss improved from 1.04450 to 1.03948, saving model to model-ep060-loss1.0
875/875 [==============================] - 348s 398ms/step - loss: 1.0395 - accuracy
Epoch 61/70
875/875 [==============================] - ETA: 0s - loss: 1.0368 - accuracy: 0.6975
Epoch 61: loss improved from 1.03948 to 1.03677, saving model to model-ep061-loss1.0
875/875 [==============================] - 349s 399ms/step - loss: 1.0368 - accuracy
Epoch 62/70
875/875 [==============================] - ETA: 0s - loss: 1.0345 - accuracy: 0.6982
Epoch 62: loss improved from 1.03677 to 1.03449, saving model to model-ep062-loss1.0
875/875 [==============================] - 349s 399ms/step - loss: 1.0345 - accuracy
Epoch 63/70
875/875 [==============================] - ETA: 0s - loss: 1.0319 - accuracy: 0.6991
Epoch 63: loss improved from 1.03449 to 1.03194, saving model to model-ep063-loss1.0
875/875 [==============================] - 349s 399ms/step - loss: 1.0319 - accuracy
Epoch 64/70
875/875 [==============================] - ETA: 0s - loss: 1.0352 - accuracy: 0.6980
Epoch 64: loss did not improve from 1.03194
875/875 [==============================] - 348s 398ms/step - loss: 1.0352 - accuracy
Epoch 65/70
875/875 [==============================] - ETA: 0s - loss: 1.0307 - accuracy: 0.6998
Epoch 65: loss improved from 1.03194 to 1.03070, saving model to model-ep065-loss1.0
875/875 [==============================] - 349s 398ms/step - loss: 1.0307 - accuracy
Epoch 66/70
875/875 [==============================] - ETA: 0s - loss: 1.0298 - accuracy: 0.6999
Epoch 66: loss improved from 1.03070 to 1.02984, saving model to model-ep066-loss1.0
875/875 [==============================] - 349s 399ms/step - loss: 1.0298 - accuracy
Epoch 67/70
875/875 [==============================] - ETA: 0s - loss: 1.0281 - accuracy: 0.7008
Epoch 67: loss improved from 1.02984 to 1.02807, saving model to model-ep067-loss1.0
875/875 [==============================] - 349s 399ms/step - loss: 1.0281 - accuracy
Epoch 68/70
875/875 [==============================] - ETA: 0s - loss: 1.0221 - accuracy: 0.7023
Epoch 68: loss improved from 1.02807 to 1.02214, saving model to model-ep068-loss1.0
875/875 [==============================] - 350s 400ms/step - loss: 1.0221 - accuracy
Epoch 69/70
875/875 [==============================] - ETA: 0s - loss: 1.0232 - accuracy: 0.7026
Epoch 69: loss did not improve from 1.02214
875/875 [==============================] - 348s 397ms/step - loss: 1.0232 - accuracy
Epoch 70/70
875/875 [==============================] - ETA: 0s - loss: 1.0174 - accuracy: 0.7036
Epoch 70: loss improved from 1.02214 to 1.01742, saving model to model-ep070-loss1.0
875/875 [==============================] - 350s 400ms/step - loss: 1.0174 - accuracy
```

Loss/Accuracy Plot

```
plt.figure(figsize=(30, 5))
plt.subplot(121)
```

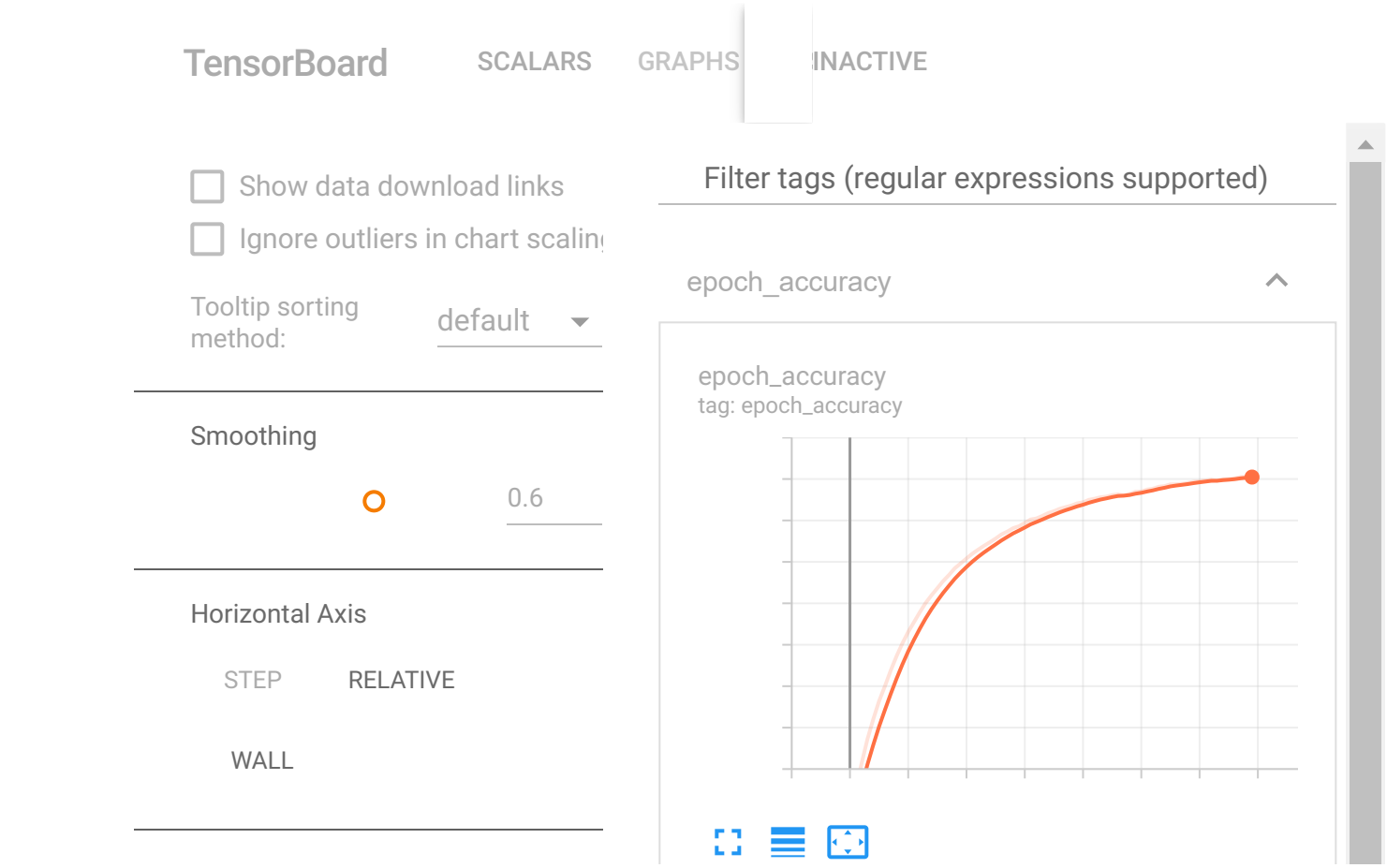
```
plt.plot(history.history['loss'])
plt.plot(history.history['accuracy'])
plt.title('Model - Loss/Accuracy', fontsize=20)
plt.ylabel('Loss/Accuracy')
plt.xlabel('Epoch')
plt.legend(['Loss', 'Accuracy'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7f8b4d2b5f90>
```



Tensorboard Plot

```
%load_ext tensorboard
%tensorboard --logdir logs/fit
```



Inference

```
with open("/content/encoded_test_image_feature.pkl", "rb") as encoded_pickle:
    encoding_test = load(encoded_pickle)
```

```
def greedySearch(image):
    image = encoding_test[image]
    image = np.reshape(image, (1,4096))
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [wordtoix[word] for word in in_text.split()]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([image,sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final


def image_caption_score_extractor(data):
```

```
    npic = 6
    target_size = (224, 224, 3)
    count = 1
    fig = plt.figure(figsize=(60,65))
    for file in tqdm(random.sample(test_filename_container, npic)):
      predicted_caption = greedySearch(file)
      captions = list(data['caption'].loc[data['filename'] == file].values)
      image_load = load_img(file, target_size = target_size)
      ax = fig.add_subplot(npic, 2, count)
      ax.axis('off')
      ax.imshow(image_load, interpolation='nearest')
      reference = [caption.split() for caption in captions]
      candidate = predicted_caption.split()
      score = round(sentence_bleu(reference, candidate), 3)
      count += 1
      ax = fig.add_subplot(npic, 2, count)
      ax.axis('off')
      ax.set_ylim(0, len(captions)+2)
      for i, caption in enumerate(captions):
        ax.text(0, i, caption, fontsize = 40, bbox=dict(facecolor='blue', alpha=0.1), color='re
      ax.text(0, i+1, greedySearch(file),  fontsize =40, fontfamily= 'fantasy', bbox=dict(facec
      ax.text(0, i+2, 'BLEU Score: '+str(score),  fontsize = 40, fontfamily= 'fantasy', bbox=di
      count += 1
    plt.grid(None)
    plt.show()

image_caption_score_extractor(data_df)
```

100%|██████████| 6/6 [00:07<00:00, 1.24s/it]

BLEU SCORE: 0.76

MAN DRIVING GREEN MOTORCYCLE IS DRIVING DOWN DIRT PATH

the motocross rider wearing blue and black pants

the biker in blue is going around corner

person in blue jacket on bike on dirt course with people looking from behind red and white barrier

person on bmx bike

biker with blue jacket black pants and white helmet driving on dirt while people watch

BLEU SCORE: 0.551

DOG BITING INTO BASKETBALL

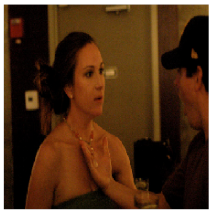
dog retrieving stick in water

wet dog carrying stick in its mouth

large brown dog with stick in his mouth coming out of water

dog with stick in his mouth walking through water

dog carrying stick while walking in water



BLEU SCORE: 0.816

MAN AND WOMAN ARE STANDING IN FRONT OF STORE

there is man looking girls necklace

the man in the black hat holds the brown hair woman necklace

man with shocked expression on his face is holding beaded necklace that is around woman neck

man stands across from woman and holds her necklace while she looks at him

man checks out the beautiful necklace woman is wearing

BLEU SCORE: 0.604

DOG JUMPS IN THE AIR WITH BALL IN HIS MOUTH

dog stands on his hind legs amid shower of tennis balls

dog jumps for several tennis balls thrown at him

brown and white dogs stands in front of wooden building while tennis balls fly through the air

brown and white dog in front of shed overwhelmed by the onslaught of tennis balls

big dog stands on his hand leg as tennis balls are thrown his direction

BLEU SCORE: 0.669

TWO CHILDREN ARE PLAYING IN LARGE PUDDLE IN THE DIRT

two people work with long ribbons in chinese courtyard

lady waving two long red streamers around

woman plays with long red ribbons in an empty square

person waving long red flags

an asian girl performs routine with two red flags in square

BLEU SCORE: 0.76

BOY IN BLUE SHIRT PLAYING BASKETBALL

where is the rest of his racket

the boy is jumping in the air

child jumping on tennis court

boy wearing jeans leaps in the air and shows his shadow below

boy jumping to hit tennis ball with his racket