```
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
from keras.callbacks import ModelCheckpoint, ModelCheckpoint, TensorBoard
from keras.layers.merge import Add
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)


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 in
    if len(col) == 1:                                          # neglecting data if full info
        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, 64152.52it/s]
```

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

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, 29.35it/s]
```

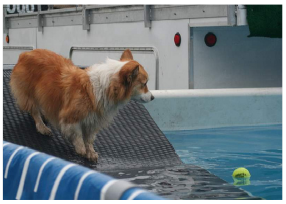
THE MAN IS INTERVIEWING TEENAGE BOY IN FRONT OF AN AUDIENCE

MAN INTERVIEWS BOY IN FRONT OF CROWD OF PEOPLE

MAN INTERVIEWS BOY IN GATHERING OF PEOPLE

GROUP OF PEOPLE WATCHING BOY GETTING INTERVIEWED BY MAN

CHILD BEING INTERVIEWED BY MAN WITH MICROPHONE



DOG LOOKING AT TENNIS BALL FLOATING IN POOL

DOG DESCENDS RAMP INTO THE POOL TO GET TENNIS BALL

BROWN AND WHITE DOG WALKS DOWN RAMP TOWARD POOL OF WATER IN WHICH GREEN TENNIS BALL FLOATS

BROWN AND WHITE DOG LOOKING INTO POOL AT TENNIS BALL

BROWN AND WHITE DOG IS LOOKING AT TENNIS BALL FLOATING IN POOL

THE GIRL DRESSED IN PINK AND WHITE AND WEARING KNIT CAP IS REACHING TOWARD FLOWERING TREE

YOUNG GIRL WITH WHITE VEST PINK SLEEVES AND PINK KNIT HAT WITH FLOWER IS LOOKING AT THE FLOWER BLOSSOMS

YOUNG GIRL IS PICKING BLOSSOMS FROM TREE

GIRL IN PINK HAT PICK FLOWER FROM TREE

CHILD IN WHITE AND PINK IS PICKING WHITE FLOWERED TREE NEXT TO FENCE

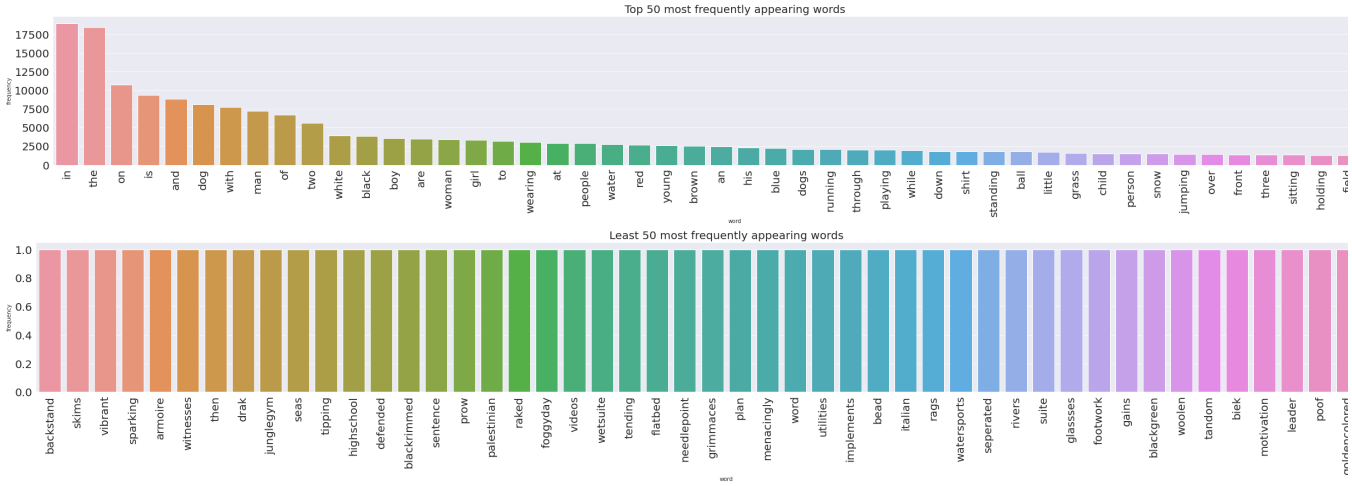
Observation

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

Plot for word - frequency

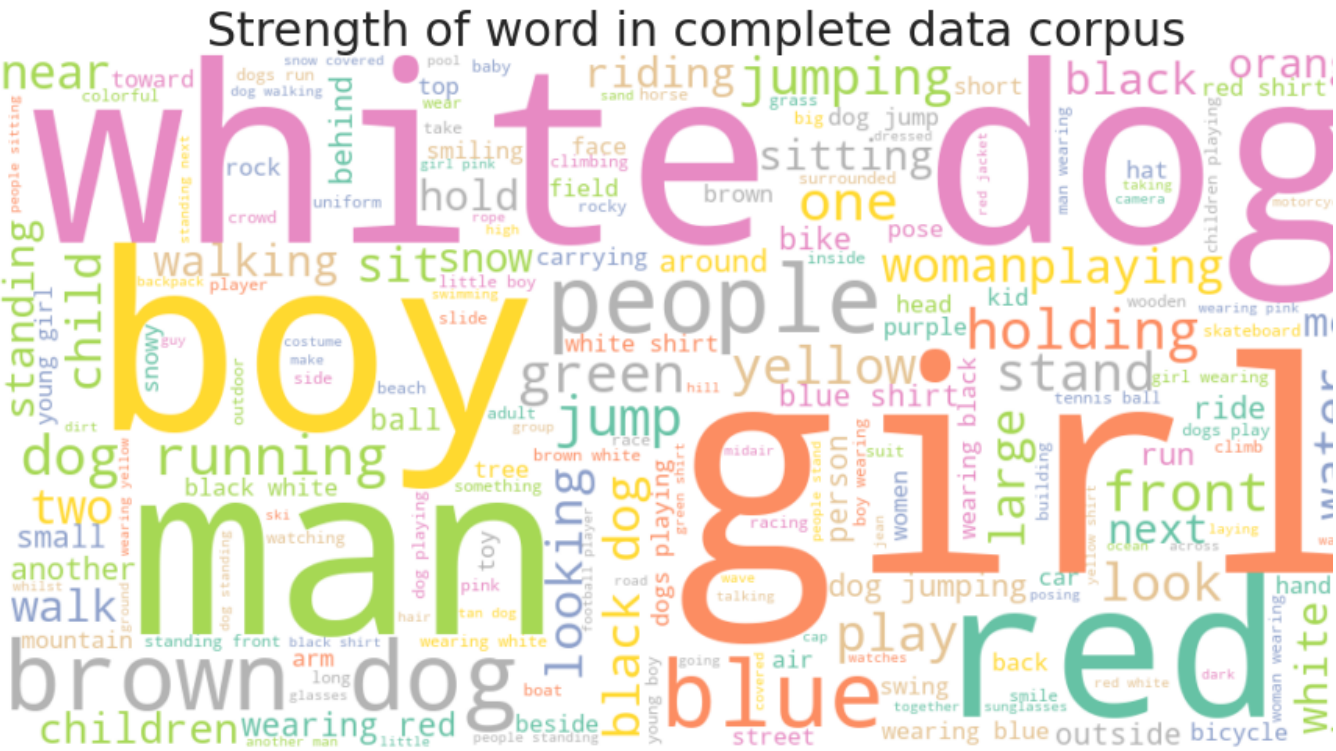
```
count = 50

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
```



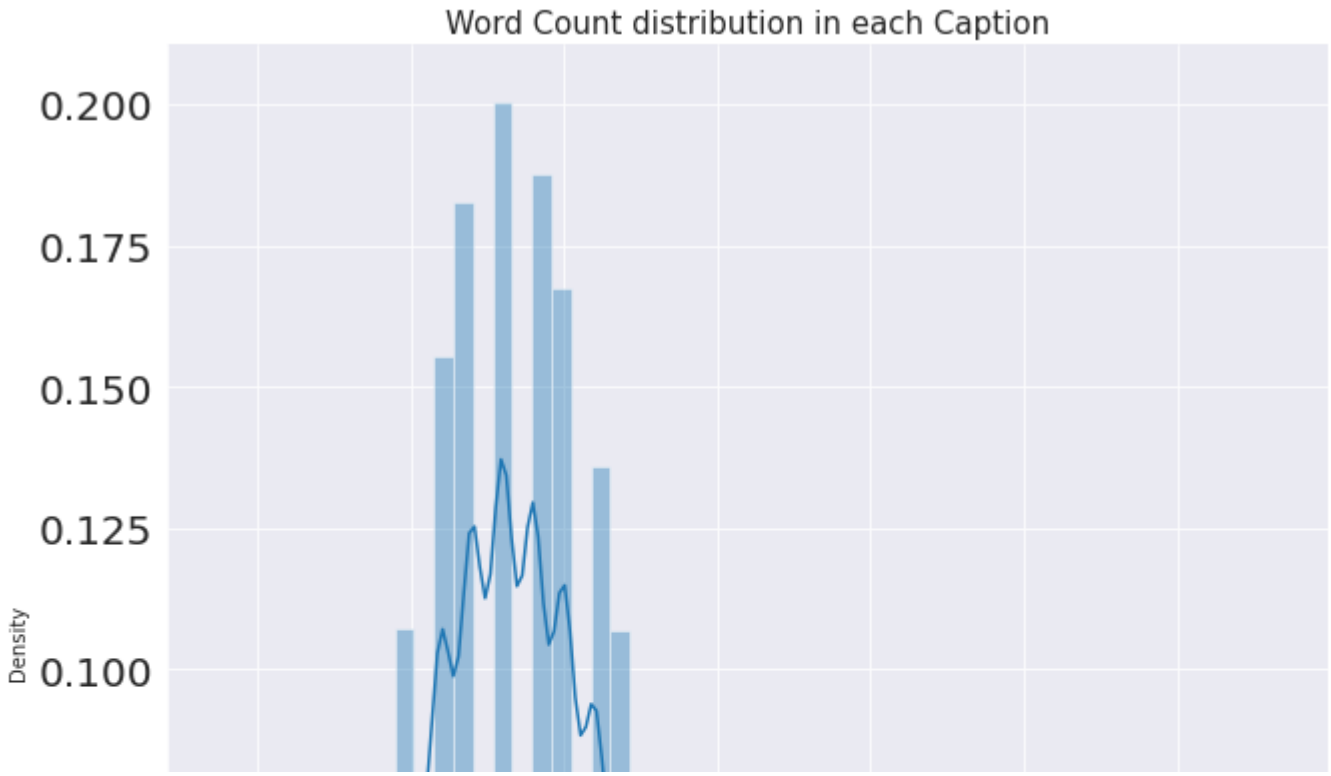
Word Cloud Visualization

```
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()
```



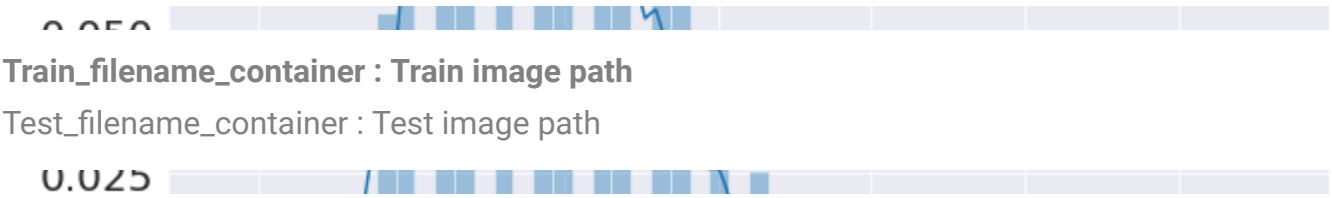
Distribution of number of words in each caption

```
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
```

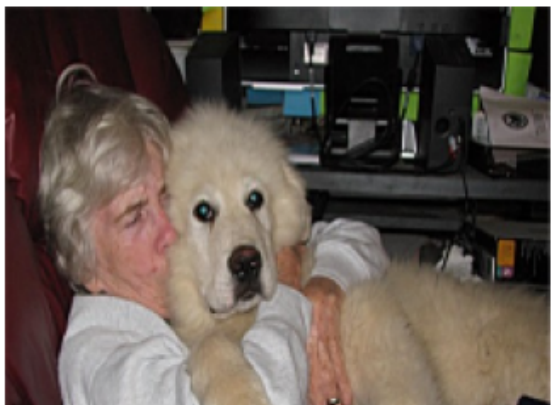
Augumentation on images

```
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 [00:20<00:00, 336.60it/s]
100%|██████████| 1000/1000 [00:02<00:00, 374.27it/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/vgg16
553467904/553467096 [==============================] - 10s 0us/step
553476096/553467096 [==============================] - 10s 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 [03:18<00:00, 35.26it/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 [00:19<00:00, 50.46it/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, 520994.57it/s]
 100%|██████████| 1000/1000 [00:00<00:00, 498195.04it/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, 1144130.28it/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
```

Embeddings: Load Glove vectors

```
embeddings_index = {}
f = open('/content/glove.6B.200d.txt', encoding="utf-8")

for line in tqdm(f):
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
```

```
    embeddings_index[word] = coefs
f.close()
print(f'\n Number of words in glove vector: {len(embeddings_index)}')
```

```
400000it [00:14, 27148.35it/s]
 Number of words in glove vector: 400000
```

Embedding matrix

```
embedding_dim = 200
embedding_matrix = np.zeros((vocab_size, embedding_dim))
for word, i in wordtoix.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector

embedding_matrix.shape
```

```
(8171, 200)
```

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, embedding_dim, mask_zero=True, weights=[embedding_matrix], traina
se2 = Dropout(0.5, name='Dropout_text')(se1)
se3 = LSTM(256, name='LSTM')(se2)
decoder1 = Add(name='Add')([fe2, se3])
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

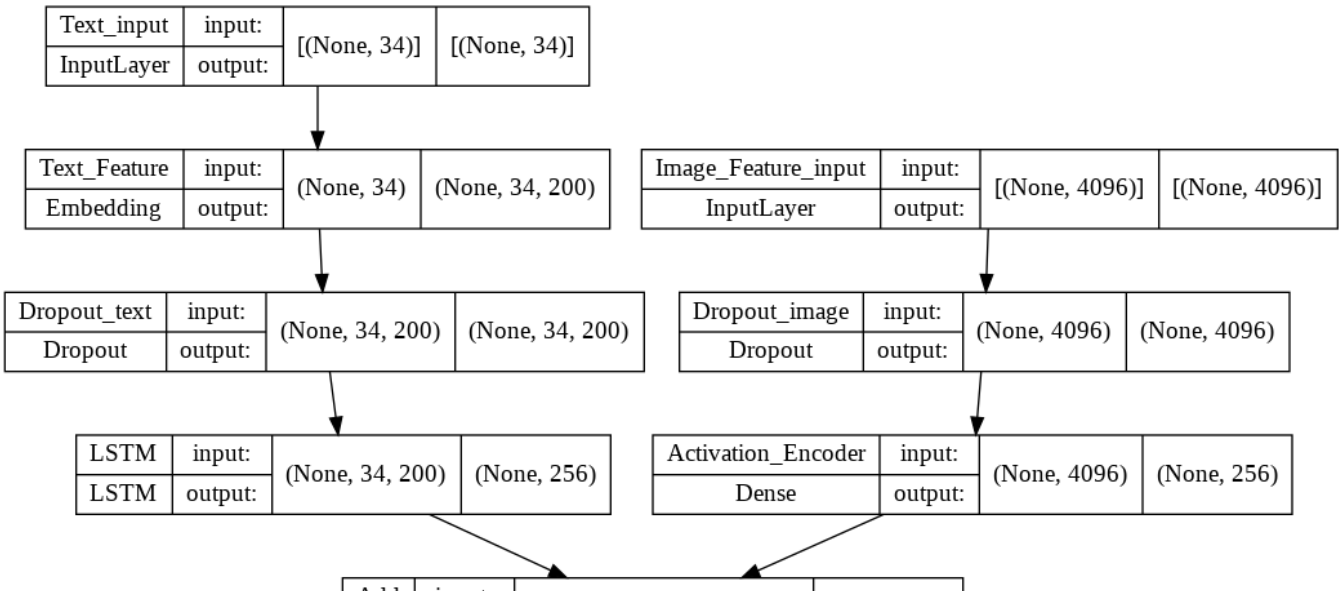
```
model.summary()
```

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

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

```
 Text_Feature (Embedding)       (None, 34, 200)      1634200     ['Text_input[0][0]']

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

 Dropout_text (Dropout)         (None, 34, 200)      0           ['Text_Feature[0][0]']

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

 LSTM (LSTM)                    (None, 256)          467968      ['Dropout_text[0][0]']

 Add (Add)                      (None, 256)          0           ['Activation_Encoder[0]
                                                                  'LSTM[0][0]']

 Activation_Decoder (Dense)     (None, 256)          65792       ['Add[0][0]']

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

==================================================================================================
Total params: 5,316,739
Trainable params: 3,682,539
Non-trainable params: 1,634,200
__________________________________________________________________________________________________
```

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 =
```

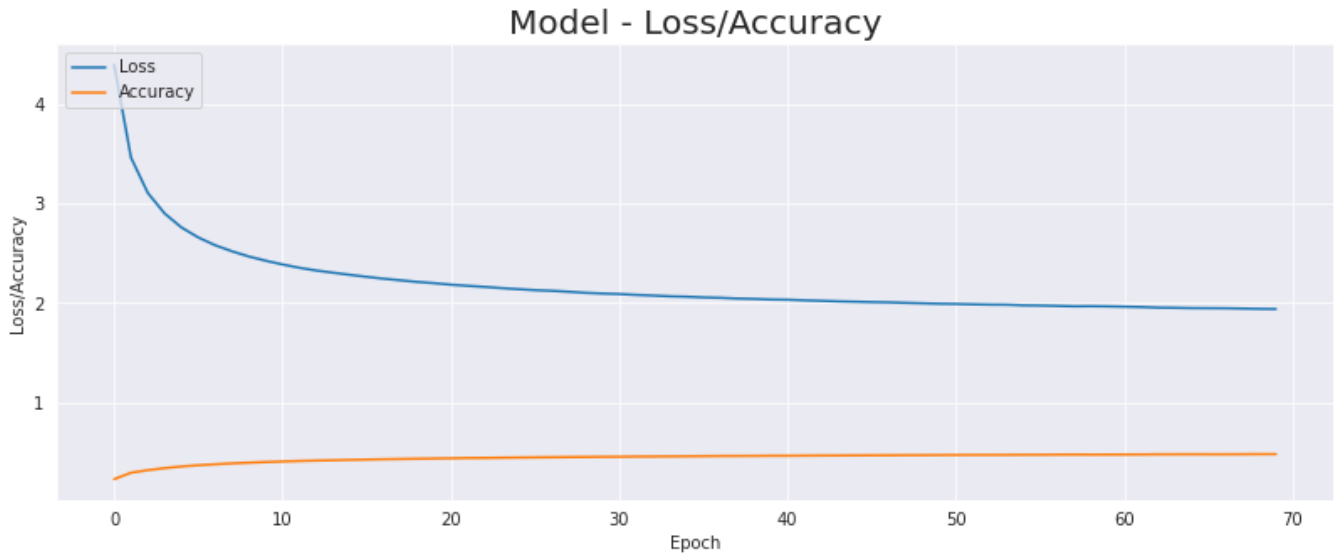
```
Epoch 57/70
875/875 [==============================] - ETA: 0s - loss: 1.9713 - accuracy: 0.4743
Epoch 57: loss improved from 1.97496 to 1.97126, saving model to model-ep057-loss1.97
875/875 [==============================] - 151s 173ms/step - loss: 1.9713 - accuracy:
Epoch 58/70
875/875 [==============================] - ETA: 0s - loss: 1.9672 - accuracy: 0.4755
Epoch 58: loss improved from 1.97126 to 1.96715, saving model to model-ep058-loss1.96
875/875 [==============================] - 150s 171ms/step - loss: 1.9672 - accuracy:
Epoch 59/70
875/875 [==============================] - ETA: 0s - loss: 1.9683 - accuracy: 0.4745
Epoch 59: loss did not improve from 1.96715
875/875 [==============================] - 150s 172ms/step - loss: 1.9683 - accuracy:
Epoch 60/70
875/875 [==============================] - ETA: 0s - loss: 1.9661 - accuracy: 0.4750
Epoch 60: loss improved from 1.96715 to 1.96613, saving model to model-ep060-loss1.96
875/875 [==============================] - 150s 171ms/step - loss: 1.9661 - accuracy:
Epoch 61/70
```

```
875/875 [==============================] - ETA: 0s - loss: 1.9631 - accuracy: 0.4755
Epoch 61: loss improved from 1.96613 to 1.96310, saving model to model-ep061-loss1.96
875/875 [==============================] - 150s 171ms/step - loss: 1.9631 - accuracy:
Epoch 62/70
875/875 [==============================] - ETA: 0s - loss: 1.9596 - accuracy: 0.4760
Epoch 62: loss improved from 1.96310 to 1.95958, saving model to model-ep062-loss1.96
875/875 [==============================] - 152s 173ms/step - loss: 1.9596 - accuracy:
Epoch 63/70
875/875 [==============================] - ETA: 0s - loss: 1.9536 - accuracy: 0.4775
Epoch 63: loss improved from 1.95958 to 1.95356, saving model to model-ep063-loss1.95
875/875 [==============================] - 150s 172ms/step - loss: 1.9536 - accuracy:
Epoch 64/70
875/875 [==============================] - ETA: 0s - loss: 1.9519 - accuracy: 0.4778
Epoch 64: loss improved from 1.95356 to 1.95187, saving model to model-ep064-loss1.95
875/875 [==============================] - 152s 173ms/step - loss: 1.9519 - accuracy:
Epoch 65/70
875/875 [==============================] - ETA: 0s - loss: 1.9488 - accuracy: 0.4782
Epoch 65: loss improved from 1.95187 to 1.94877, saving model to model-ep065-loss1.94
875/875 [==============================] - 149s 171ms/step - loss: 1.9488 - accuracy:
Epoch 66/70
875/875 [==============================] - ETA: 0s - loss: 1.9481 - accuracy: 0.4777
Epoch 66: loss improved from 1.94877 to 1.94810, saving model to model-ep066-loss1.94
875/875 [==============================] - 152s 173ms/step - loss: 1.9481 - accuracy:
Epoch 67/70
875/875 [==============================] - ETA: 0s - loss: 1.9472 - accuracy: 0.4776
Epoch 67: loss improved from 1.94810 to 1.94718, saving model to model-ep067-loss1.94
875/875 [==============================] - 151s 173ms/step - loss: 1.9472 - accuracy:
Epoch 68/70
875/875 [==============================] - ETA: 0s - loss: 1.9443 - accuracy: 0.4784
Epoch 68: loss improved from 1.94718 to 1.94434, saving model to model-ep068-loss1.94
875/875 [==============================] - 151s 172ms/step - loss: 1.9443 - accuracy:
Epoch 69/70
875/875 [==============================] - ETA: 0s - loss: 1.9414 - accuracy: 0.4796
Epoch 69: loss improved from 1.94434 to 1.94136, saving model to model-ep069-loss1.94
875/875 [==============================] - 150s 172ms/step - loss: 1.9414 - accuracy:
Epoch 70/70
875/875 [==============================] - ETA: 0s - loss: 1.9394 - accuracy: 0.4792
Epoch 70: loss improved from 1.94136 to 1.93944, saving model to model-ep070-loss1.93
875/875 [==============================] - 149s 171ms/step - loss: 1.9394 - accuracy:
```

Loss/Accuracy Plot

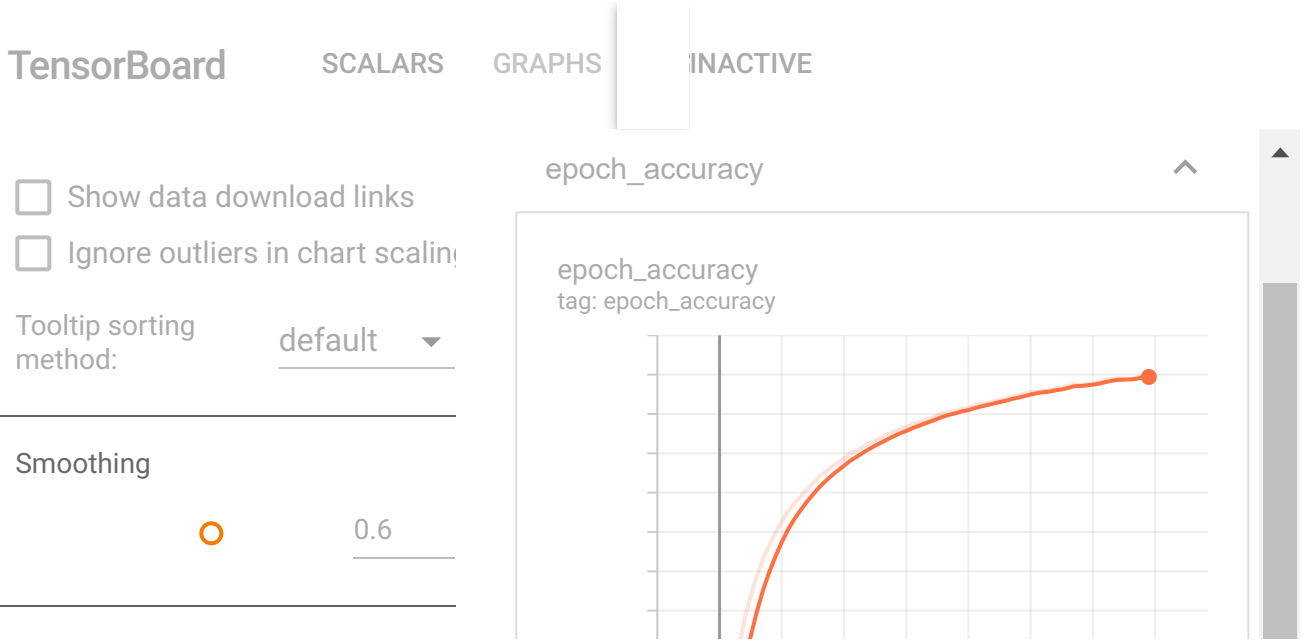
```
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history.history['loss'])
plt.plot(history.history['accuracy'])
plt.title('Model - Loss/Accuracy', fontsize=20)
plt.ylabel('Loss/Accuracy')
plt.xlabel('Epoch')
plt.legend(['Loss', 'Accuracy'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7f37c50d3ed0>
```



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)
```

```
        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 [00:06<00:00, 1.08s/it]

BLEU SCORE: 0.434

PERSON IN RED JACKET IS CLIMBING STEEP MOUNTAINSIDE COVERED

skier wearing white sunglasses is skiing down snowy mountain very fast

skier skiing downhill in cloud of snow

skier is throwing up snow as he skis off piste

person dressed in red and grey is skiing down snowy slope

man is skiing down mountain

BLEU SCORE: 0.481

MAN IN BLACK COAT AND KHAKI EATS CIGARETTE

the small child bangs his drum whil looking at an older man doing the same

the man with the big drum is marching next to boy with small drum

man plays drum and little boy hits his own little drum

man and little boy beating drums

large man in leather costume plays big drum beside little boy playing little drum

BLEU SCORE: 0.482

BOY HOLDS HIS CELLPHONE TO THE CAMERA

young child holds spoon to its mouth while sitting in chair

small child dressed in green is eating with spoon

little boy is eating his food off of spoon while sitting on patio

little boy holds spoon up to his mouth

boy eats with spoon

BLEU SCORE: 0.687

TWO PEOPLE IN RED SHIRTS WALK DOWN THE STREET

the man is wearing green shirt and blue jeans and carrying free hugs sign

young man in black shirt is holding free hugs sign

man is holding sign that says free hugs

man holds up free hugs sign above his head

man carrying sign that says free hug along the sidewalk

BLEU SCORE: 0.359

LITTLE GIRL IN PINK SHIRT IS CLIMBING THROUGH THE TUB

the little girl in the green outfit is playing with piece of orange material

young girl plays outside

little girl in green outfit wringing out an orange rag

girl wearing green twists something in her hands

girl wrings out wet shirt

BLEU SCORE: 0.788

WOMAN IN COWBOY SUIT AND MAN IN WETSUIT STANDING IN FRONT OF MOUNTAIN

woman taking pictures of the surf and child in the foreground

woman is taking photograph on beach at the edge of the water while girl stands in the for

man and girl are at the shoreline while the man takes picture and the girl hods doll

little girl in red at the beach while adult points camera

child on the beach while woman takes pictures nearby