```
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, 63581.54it/s]
```

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

index	filename	caption	caption_with_tags	cap
		child in pink dress is	startseq child in pink	

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, 12.02it/s]

THE GIRL IN THE BROWN SHIRT IS HANGING ONTO BLUE POLE

PEOPLE PUT UP VOLLEYBALL NET AT THE BEACH

PEOPLE HANGING UP VOLLEYBALL NET

GIRL CLIMBS VOLLEYBALL NET AT BEACH

WOMAN IS CLIMBING VOLLEYBALL NET ON BEACH AS TWO OTHERS WATCH

SNOWBOARDER JUMPS OFF OF RAMP IN THE SNOWY MOUNTAINS

SNOWBOARDER IS LEAPING OF JUMP IN THE MOUNTAINS

SNOWBOARDER IN MIDAIR HOVERING OVER SNOW RAMP

SNOWBOARDER IN MIDAIR

SNOWBOARDER FLIES THROUGH THE AIR

PERSON ON BICYCLE JUMPING HIGH IN THE AIR OVER THE SIGN THAT READS EASTPAK OTHERS LOOK UP AT HIM

STUNTBICYCLIST FLIES ABOVE ONLOOKERS

MAN IN THE AIR ON BMX BICYCLE WITH PEOPLE SITTING WATCHING HIM IN FRONT OF AN EASTPAK BANNER

MAN DOES JUMP TRICK ON BIKE

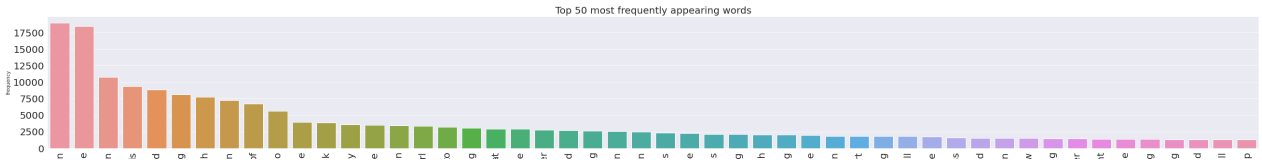
GUY PERFORMING BICYCLE JUMP TRICK FOR AN AUDIENCE

Observation

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

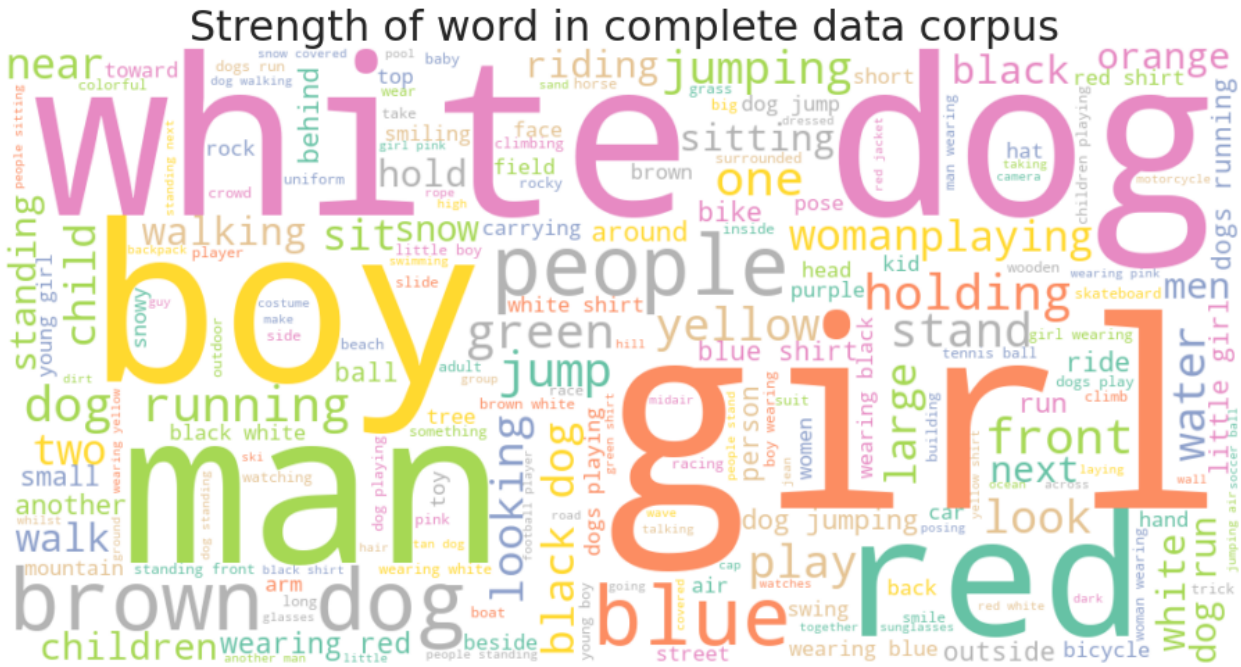
Plot for word - frequency

```
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

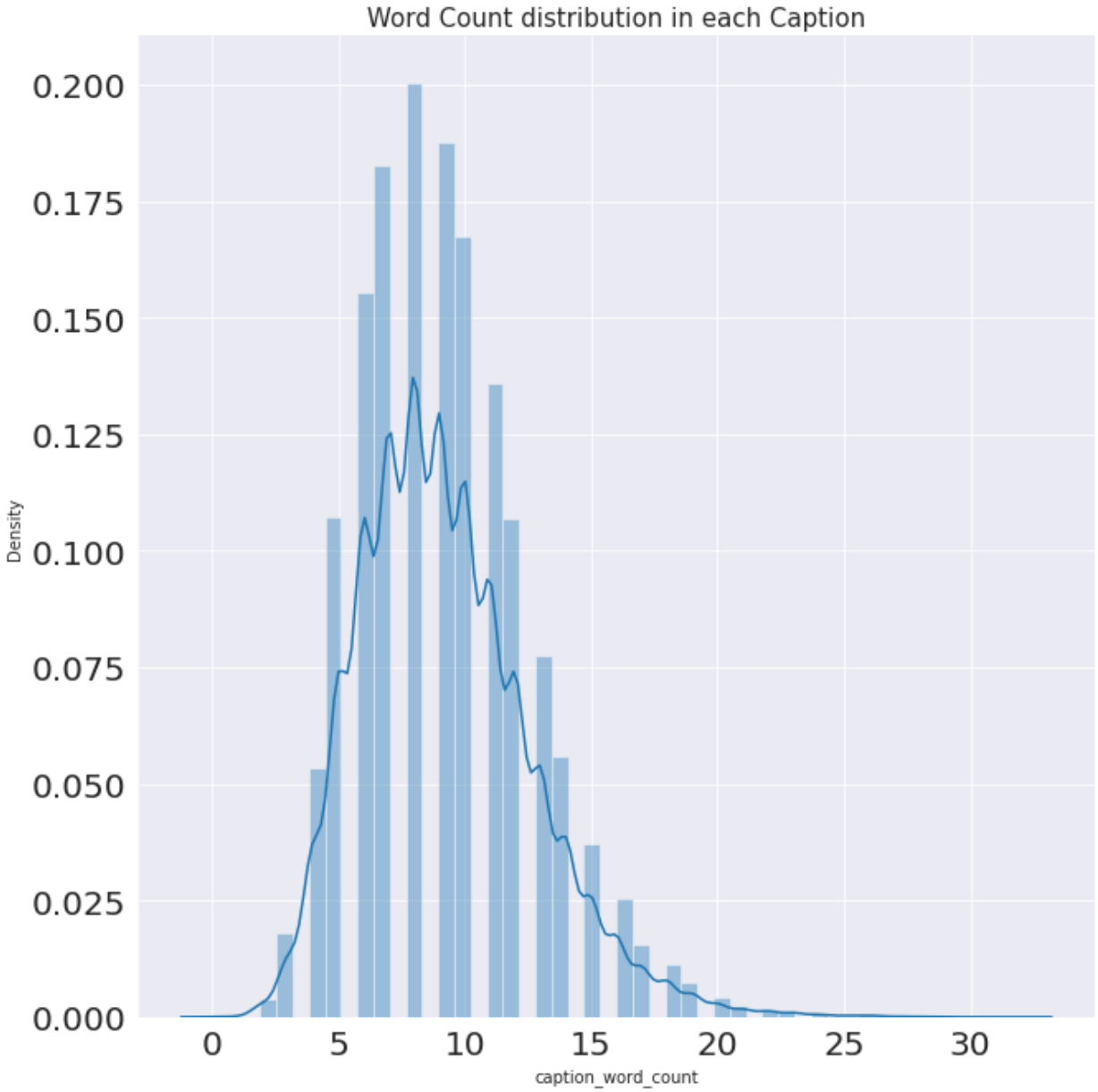

```
text = ' '
for caption in data_df.caption:
  text = text + caption
plt.figure(figsize=(16, 12))
word_cloud = WordCloud(width = 1000, height = 500, random_state=1, collocations= True, backgr
plt.imshow(word_cloud, interpolation='bilinear')
plt.axis("off")
plt.title('Strength of word in complete data 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 [00:19<00:00, 362.68it/s]
 100%|██████████| 1000/1000 [00:02<00:00, 357.64it/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 [==============================] - 4s 0us/step
553476096/553467096 [==============================] - 4s 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 [03:47<00:00, 30.74it/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 [00:18<00:00, 53.68it/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, 100699.09it/s]
 100%|██████████| 1000/1000 [00:00<00:00, 71444.70it/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, 261913.56it/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 [00:16, 24719.47it/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)
```

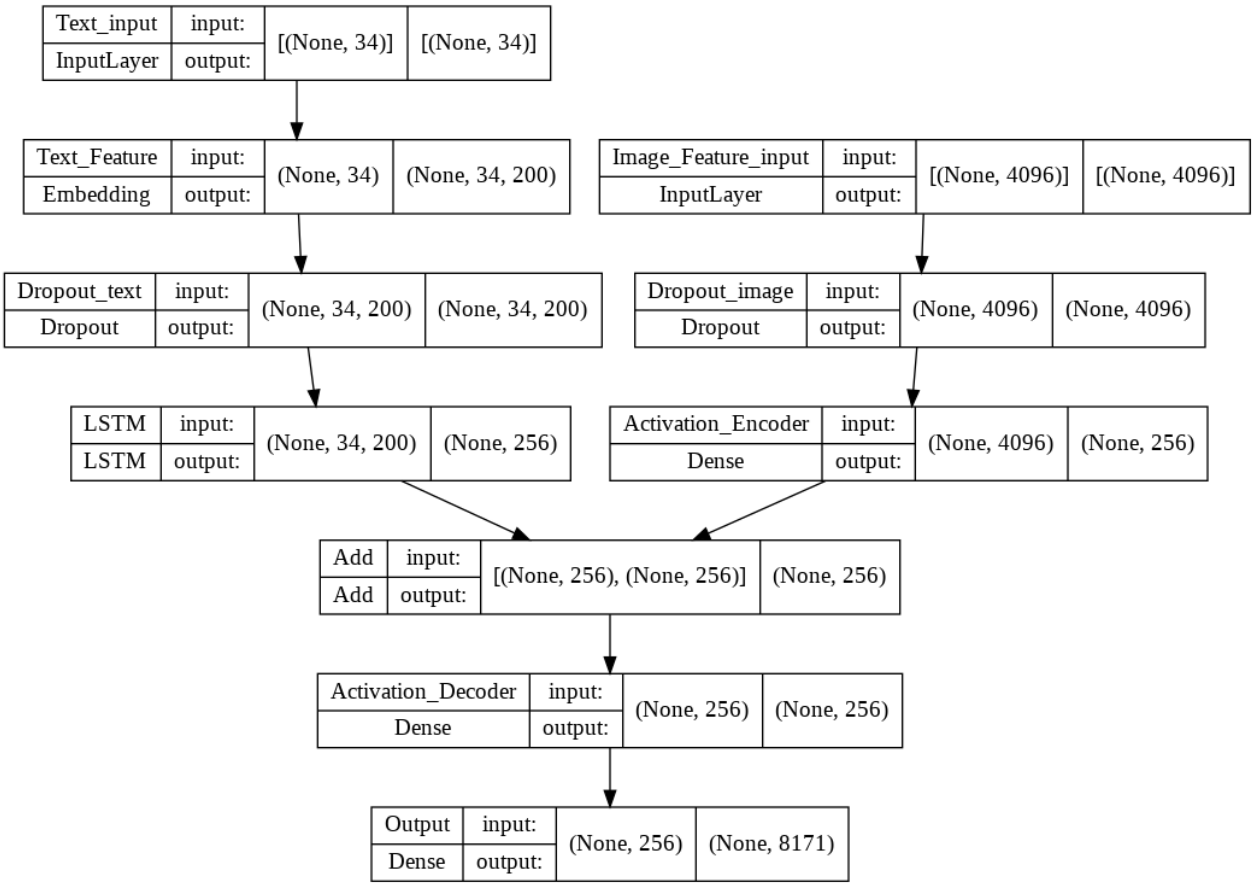
Model Summary

```
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 =
```

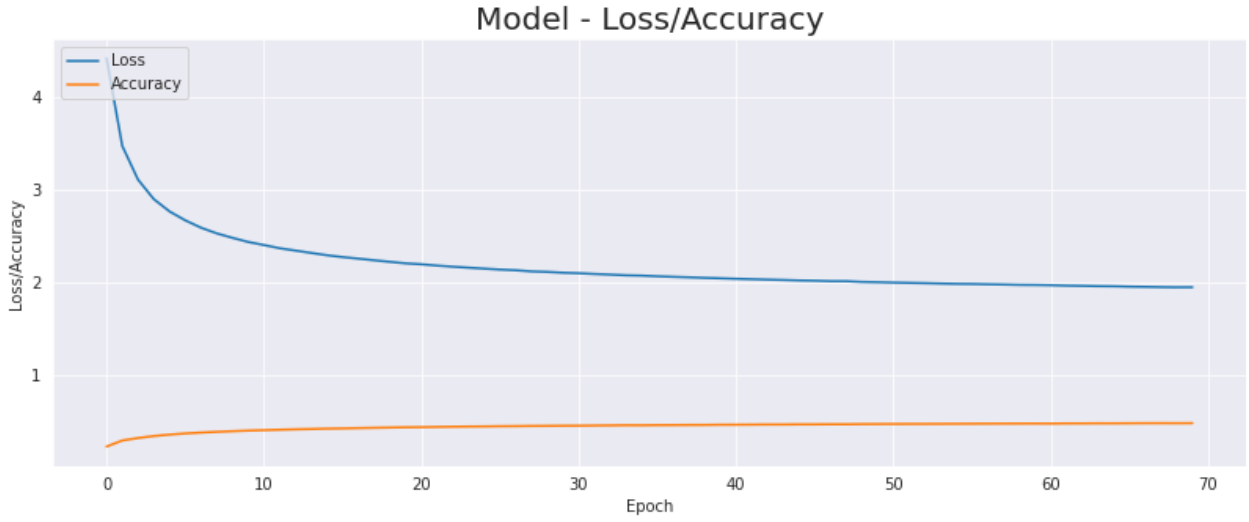
```
875/875 [==============================] - ETA: 0s - loss: 2.1292 - accuracy: 0.4466
Epoch 27: loss improved from 2.13657 to 2.12920, saving model to model-ep027-loss2.12
875/875 [==============================] - 150s 171ms/step - loss: 2.1292 - accuracy:
Epoch 28/70
875/875 [==============================] - ETA: 0s - loss: 2.1167 - accuracy: 0.4481
Epoch 28: loss improved from 2.12920 to 2.11672, saving model to model-ep028-loss2.11
875/875 [==============================] - 151s 173ms/step - loss: 2.1167 - accuracy:
Epoch 29/70
875/875 [==============================] - ETA: 0s - loss: 2.1114 - accuracy: 0.4487
Epoch 29: loss improved from 2.11672 to 2.11145, saving model to model-ep029-loss2.11
875/875 [==============================] - 150s 172ms/step - loss: 2.1114 - accuracy:
Epoch 30/70
875/875 [==============================] - ETA: 0s - loss: 2.1023 - accuracy: 0.4503
Epoch 30: loss improved from 2.11145 to 2.10228, saving model to model-ep030-loss2.10
875/875 [==============================] - 153s 174ms/step - loss: 2.1023 - accuracy:
Epoch 31/70
875/875 [==============================] - ETA: 0s - loss: 2.0970 - accuracy: 0.4508
Epoch 31: loss improved from 2.10228 to 2.09700, saving model to model-ep031-loss2.09
875/875 [==============================] - 151s 172ms/step - loss: 2.0970 - accuracy:
Epoch 32/70
875/875 [==============================] - ETA: 0s - loss: 2.0890 - accuracy: 0.4526
Epoch 32: loss improved from 2.09700 to 2.08899, saving model to model-ep032-loss2.08
875/875 [==============================] - 151s 172ms/step - loss: 2.0890 - accuracy:
Epoch 33/70
875/875 [==============================] - ETA: 0s - loss: 2.0823 - accuracy: 0.4541
Epoch 33: loss improved from 2.08899 to 2.08230, saving model to model-ep033-loss2.08
875/875 [==============================] - 150s 171ms/step - loss: 2.0823 - accuracy:
Epoch 34/70
875/875 [==============================] - ETA: 0s - loss: 2.0746 - accuracy: 0.4560
Epoch 34: loss improved from 2.08230 to 2.07460, saving model to model-ep034-loss2.07
875/875 [==============================] - 150s 171ms/step - loss: 2.0746 - accuracy:
Epoch 35/70
875/875 [==============================] - ETA: 0s - loss: 2.0711 - accuracy: 0.4554
Epoch 35: loss improved from 2.07460 to 2.07112, saving model to model-ep035-loss2.07
875/875 [==============================] - 151s 172ms/step - loss: 2.0711 - accuracy:
Epoch 36/70
875/875 [==============================] - ETA: 0s - loss: 2.0638 - accuracy: 0.4571
Epoch 36: loss improved from 2.07112 to 2.06382, saving model to model-ep036-loss2.06
875/875 [==============================] - 150s 171ms/step - loss: 2.0638 - accuracy:
Epoch 37/70
875/875 [==============================] - ETA: 0s - loss: 2.0583 - accuracy: 0.4577
Epoch 37: loss improved from 2.06382 to 2.05829, saving model to model-ep037-loss2.05
875/875 [==============================] - 151s 173ms/step - loss: 2.0583 - accuracy:
Epoch 38/70
875/875 [==============================] - ETA: 0s - loss: 2.0530 - accuracy: 0.4588
Epoch 38: loss improved from 2.05829 to 2.05296, saving model to model-ep038-loss2.05
875/875 [==============================] - 151s 173ms/step - loss: 2.0530 - accuracy:
Epoch 39/70
875/875 [==============================] - ETA: 0s - loss: 2.0472 - accuracy: 0.4592
Epoch 39: loss improved from 2.05296 to 2.04716, saving model to model-ep039-loss2.04
```

```
875/875 [==============================] - 152s 174ms/step - loss: 2.0472 - accuracy:
Epoch 40/70
875/875 [==============================] - ETA: 0s - loss: 2.0422 - accuracy: 0.4610
Epoch 40: loss improved from 2.04716 to 2.04218, saving model to model-ep040-loss2.04
875/875 [==============================] - 150s 172ms/step - loss: 2.0422 - accuracy:
Epoch 41/70
875/875 [==============================] - ETA: 0s - loss: 2.0368 - accuracy: 0.4616
```

Loss/Accuracy Plot

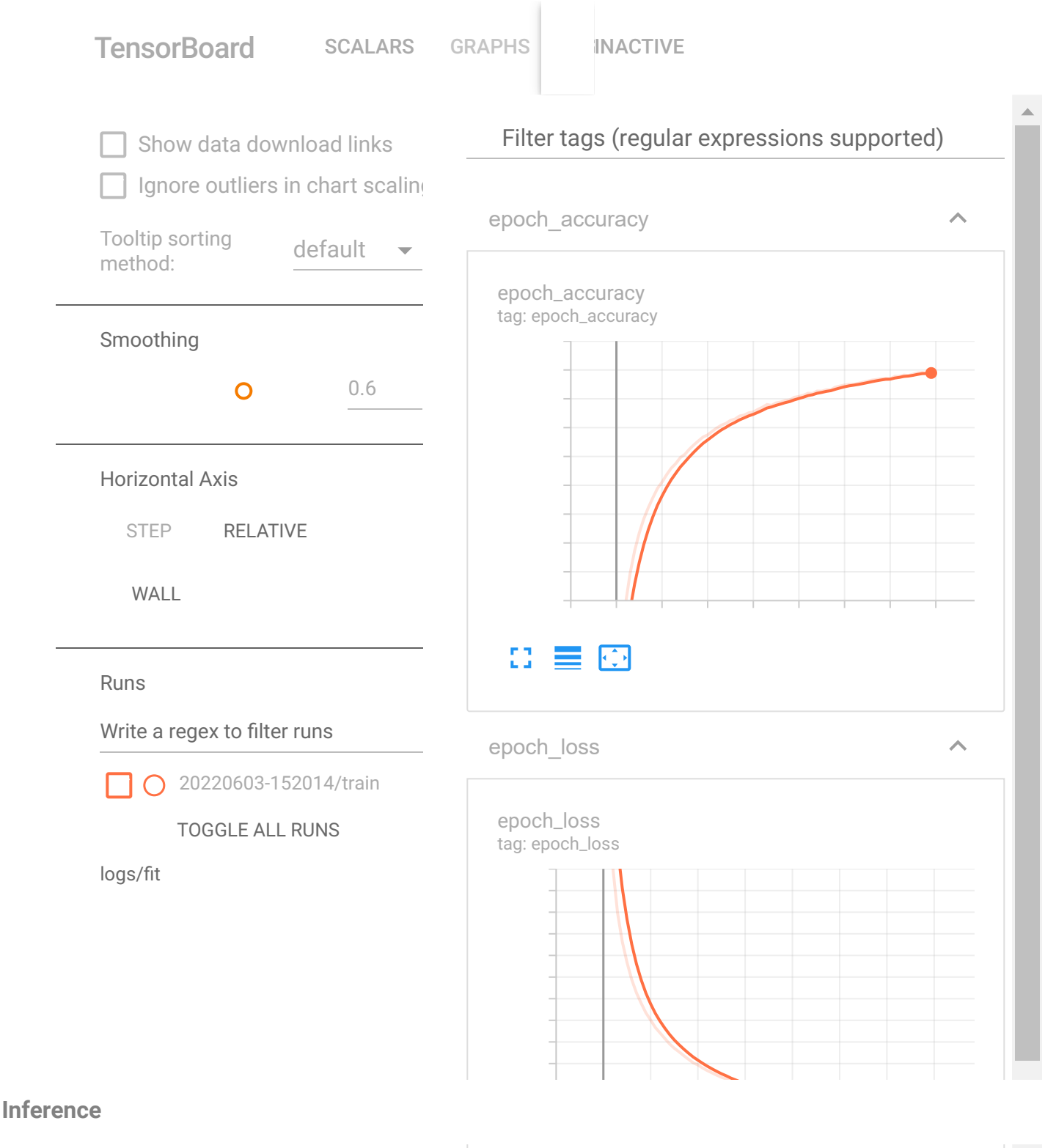
```
plt.figure(figsize=(30, 5))
plt.subplot(121)
plt.plot(history.history['loss'])
plt.plot(history.history['accuracy'])
plt.title('Model - Loss/Accuracy', fontsize=20)
plt.ylabel('Loss/Accuracy')
plt.xlabel('Epoch')
plt.legend(['Loss', 'Accuracy'], loc='upper left')
```

```
<matplotlib.legend.Legend at 0x7fdfe66bb150>
```



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 [00:05<00:00,  1.13it/s]
```

BLEU SCORE: 0.485

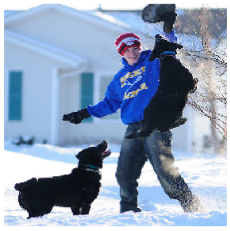
MAN IN RED JACKET AND BACKPACK WALKS DOWN CITY STREET
woman tries to cross the road before man on red scooter
woman in coat and man in tracksuit talking on phone cross the street
woman crosses the street near man who is on his cellphone
man talks on his cellphone and another man rides wheelchair
man and woman wait to cross street

BLEU SCORE: 0.795

YOUNG BOY JUMPS INTO THE WATER
young man leaps into the water
boy jumps into the water at beach
boy is about to jump into the water
boy in blue camo pants jumping into water
boy in swimsuit jumping into water

BLEU SCORE: 0.485

GROUP OF FRIENDS ARE ALL PERFORMING IN THE GROUP
group of kids singing on stage
group of performers sing and work to entertain the audience
group of performers dancing in front of two djs
group of people singing in front of dj at his station
group of people sing and dance at concert



BLEU SCORE: 0.339

BOY IN RED COAT PLAYS IN THE SNOW
two dogs play with human and disc in the snow
the man is playing with two dogs in the snow
young person playing in the snow with two black dogs
man stands in the snow and holds toy up while black dog jumps for it
boy in blue sweatshirt playing frisbee with two black dogs

BLEU SCORE: 0.364

TWO DOGS RUN THROUGH THE WOODS
tan and white dog retrieving ball on gravel path
large dog walks along gravel path in the woods
dog with ball in his mouth running down road covered in leaves
brown dog is carrying ball up the road
brown and white dog trotting down the road in autumn

BLEU SCORE: 0.783

PERSON IN RED JACKET IS CLIMBING ON MOUNTAINTOP
the person is wearing shorts and climbing gray sand hill under blue sky
person walks up white sandy hill against the blue sky
person is hiking to the top of hill
man reaches the top of tall sand dune
hiker ascends snowy hill