Configuration Manual

MSc Research Project
Programme Name

Amit Vajpeyee
Student ID: x19218397

School of Computing
National College of Ireland

Supervisor: Dr. Abid Yaqoob

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Amit Vajpeyee
Student ID:	x19218397
Programme:	Data Analytics
Year:	2022
Module:	MSc Research Project
Supervisor:	Dr. Abid Yaqoob
Submission Due Date:	15/12/2022
Project Title:	Configuration Manual
Word Count:	577
Page Count:	12

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th December 2022

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Amit Vajpeyee
19218397

1 Introduction

This report, step by step instructions and code have been presented so that the authorized people can reproduce the same code and similar results on their systems. These have been elaborated in cogent and concise manner below.

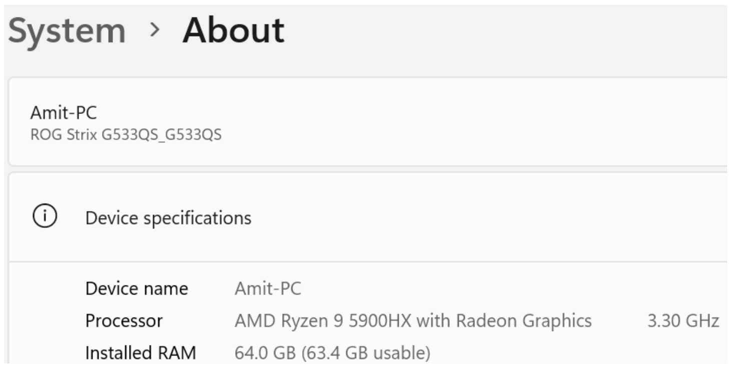
2 System Specifications and Configurations

Specifications for the hardware as well as the setup of software have been elucidated below with corresponding images or diagrams wherever necessary.

2.1 Hardware Specifications

- Brand: Asus
- Model: G533QS-HQ236TS
- Processor: AMD Ryzen 9 5900HX
- GPU: Nvidia RTX 3080 (16 GB)
- RAM: 64 GB

A screenshot of the same can be seen in Figure in Figure. 1



Device name	Amit-PC	
Processor	AMD Ryzen 9 5900HX with Radeon Graphics	3.30 GHz
Installed RAM	64.0 GB (63.4 GB usable)	

Figure 1: System configuration

2.2 Software Configuration

For software configuration, Windows 11 Pro, Python 3.10, Jupyter notebook, Pycharm Figure. 2 and Notepad has been used.

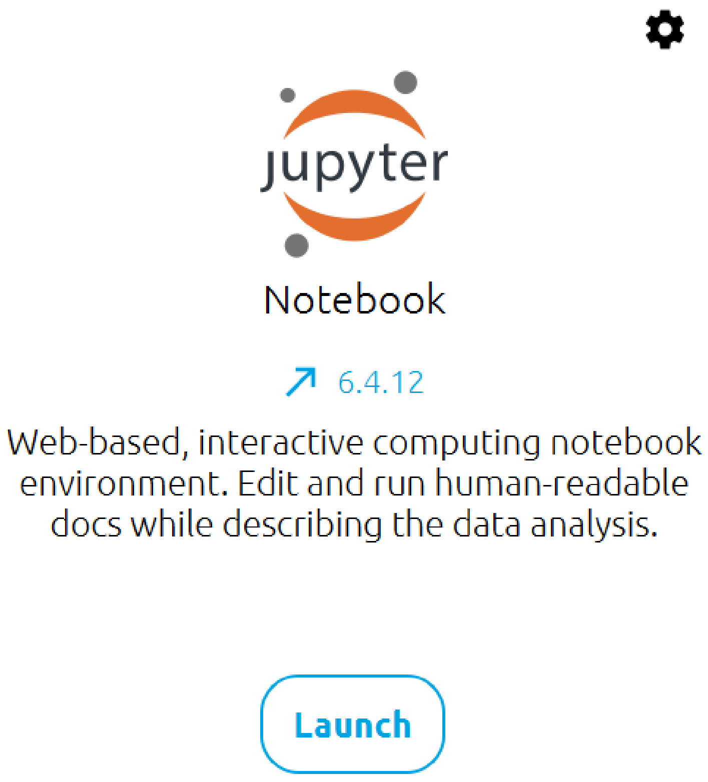

jupyter
Notebook
6.4.12
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.
Launch

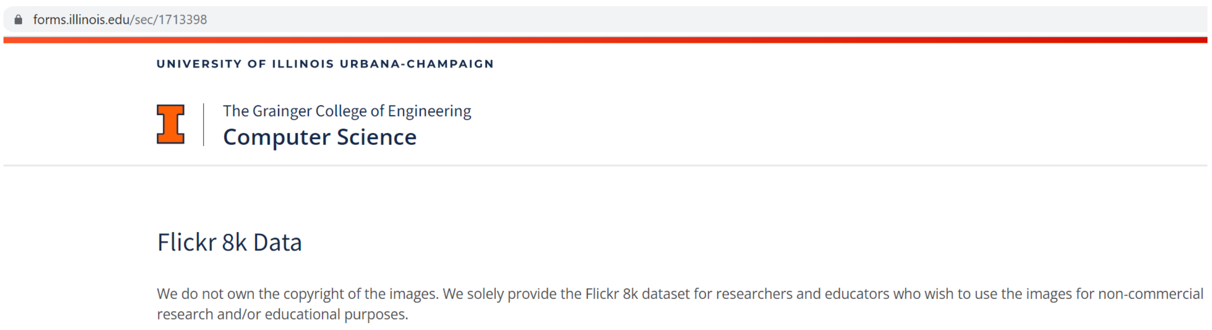
Figure 2: Jupyter notebook

- Windows 11 Pro - The entire research was conducted on Windows 11 Pro.
- Notepad - The raw dataset for captions is in the format of text file.
- Jupyter Notebook – It was used as the primary GUI for model development purposes.
- Python 3.10 – Python was used as the main programming language.
- PyCharm Community Edition - It is the IDE which was used for Robotic Process Automation and manipulation of the raw dataset.

3 Implementation

3.1 Data Source

Flickr 8k dataset can be downloaded from the link provided by the University of Ilinois [1]. Refer figure 3

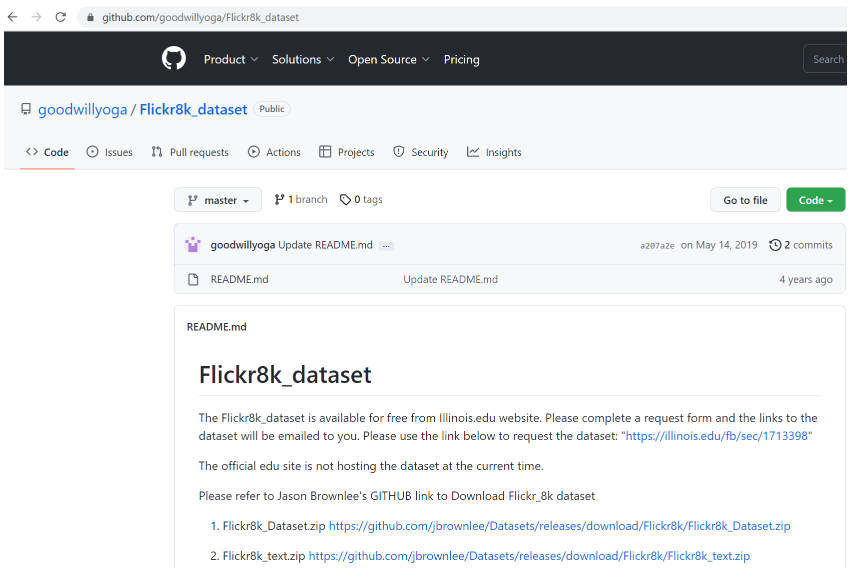

forms.illinois.edu/sec/1713398
UNIVERSITY OF ILLINOIS URBANA-CHAMPAIGN
The Grainger College of Engineering
Computer Science

Flickr 8k Data

We do not own the copyright of the images. We solely provide the Flickr 8k dataset for researchers and educators who wish to use the images for non-commercial research and/or educational purposes.

Figure 3: Flickr 8K Dataset Download Page 1

In this work, the dataset was downloaded from the GitHub link [2] in order for the automation to download the file without having to wait for the link to be sent via email from the University of Ilinois. Refer figure 4



goodwillyoga Update README.md ... a207a2e on May 14, 2019 2 commits

README.md Update README.md 4 years ago

README.md

Flickr8k_dataset

The Flickr8k_dataset is available for free from Illinois.edu website. Please complete a request form and the links to the dataset will be emailed to you. Please use the link below to request the dataset: "https://illinois.edu/fb/sec/1713398"

The official edu site is not hosting the dataset at the current time.

Please refer to Jason Brownlee's GITHUB link to Download Flickr_8k dataset

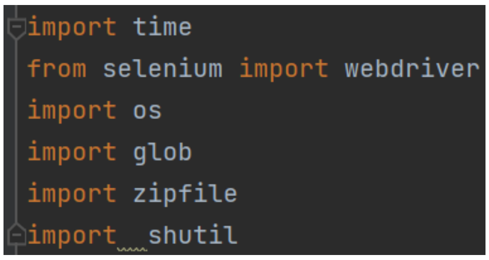
1. Flickr8k_Dataset.zip https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip
2. Flickr8k_text.zip https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip

Figure 4: Flickr 8K Dataset Download Page 2

3.2 Downloading and Manipulating the Dataset

- Open the Pycharm Community Edition
- Import the requisite libraries Figure. 5

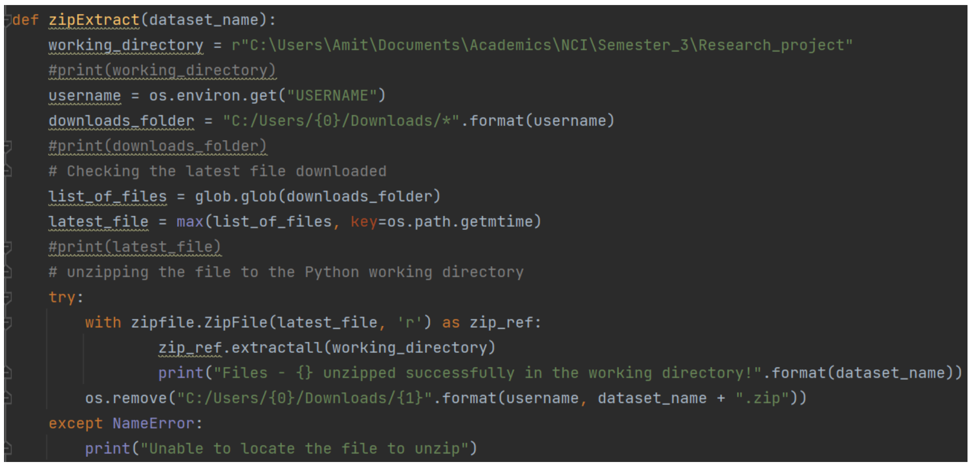
[1]https://illinois.edu/fb/sec/1713398
[2]https://github.com/goodwillyoga/Flickr8k_dataset



```python
import time
from selenium import webdriver
import os
import glob
import zipfile
import  shutil
```

Figure 5: Importing libraries

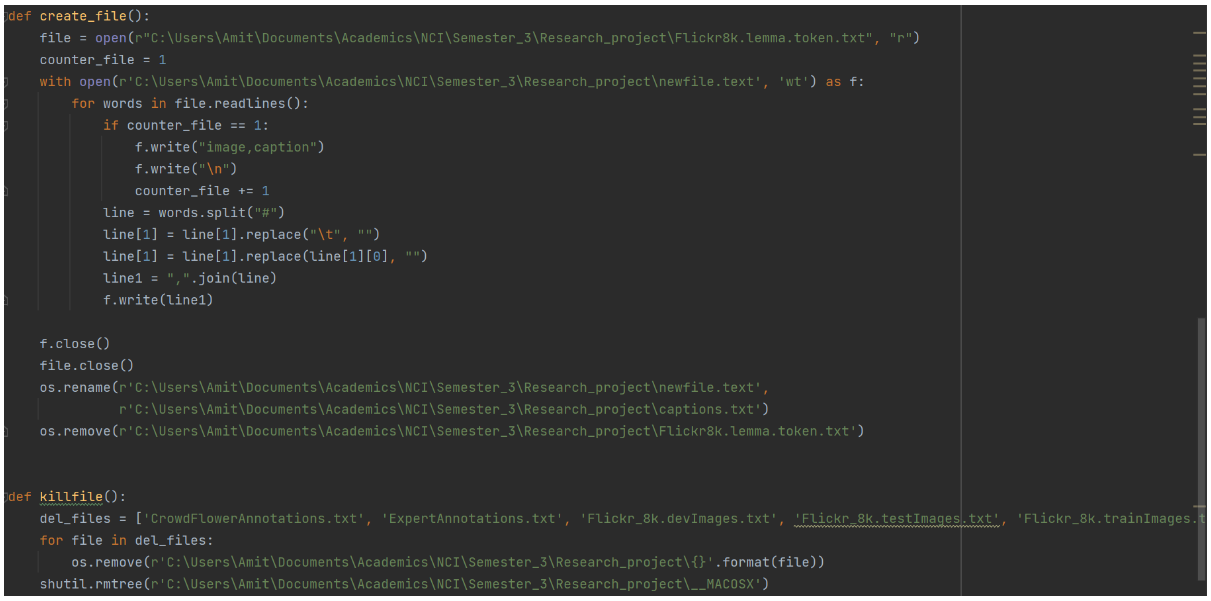
- Type the code for custom function to extract the dataset in to the correct working directory. Figure. 6



```python
def zipExtract(dataset_name):
    working_directory = r"C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project"
    #print(working_directory)
    username = os.environ.get("USERNAME")
    downloads_folder = "C:/Users/{0}/Downloads/*".format(username)
    #print(downloads_folder)
    # Checking the latest file downloaded
    list_of_files = glob.glob(downloads_folder)
    latest_file = max(list_of_files, key=os.path.getmtime)
    #print(latest_file)
    # unzipping the file to the Python working directory
    try:
        with zipfile.ZipFile(latest_file, 'r') as zip_ref:
                zip_ref.extractall(working_directory)
                print("Files - {} unzipped successfully in the working directory!".format(dataset_name))
        os.remove("C:/Users/{0}/Downloads/{1}".format(username, dataset_name + ".zip"))
    except NameError:
        print("Unable to locate the file to unzip")
```

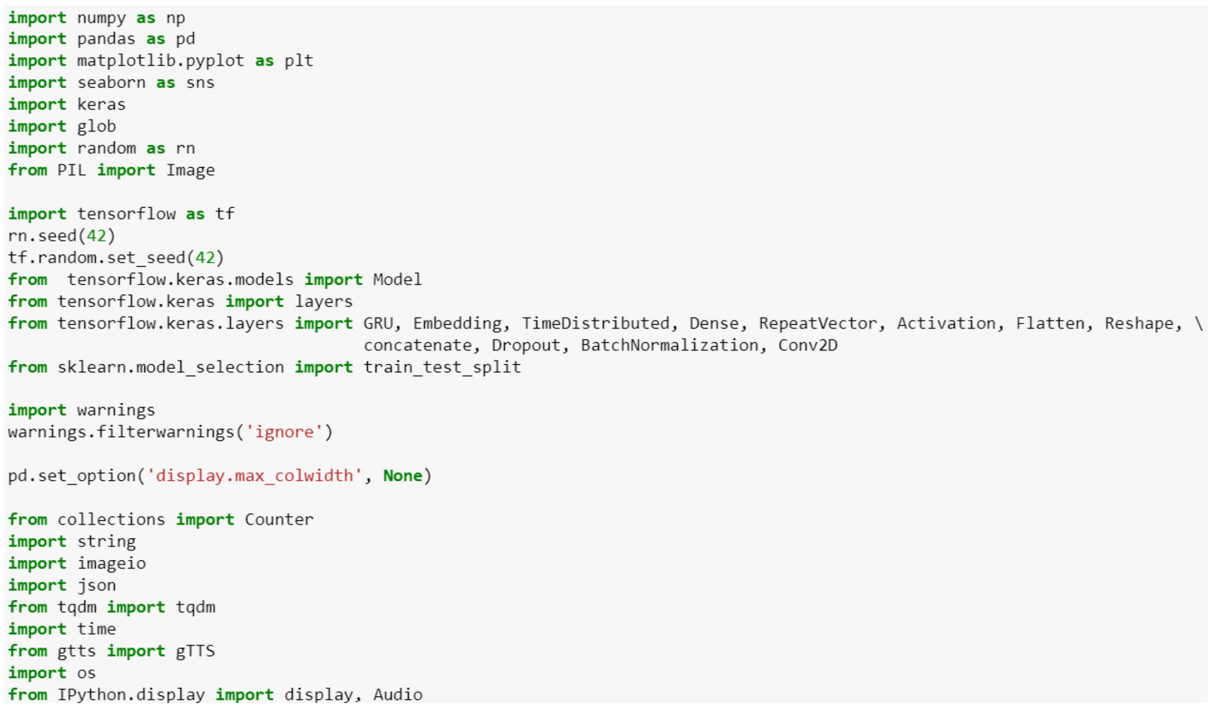
Figure 6: Code to Extract the raw data in working directory

- Type the code for manipulating the extracted data and to delete the unnecessary files Figure. 7
- Open Jupyter Notebook.
- Type the code to import requisite libraries. Figure. 8
- Create a Pandas DataFrame to contain the image ID's, image paths and captions. Figure. 9
- Create a vocabulary without punctuation marks. Figure. 10
- Type the code for Caption Pre-processing. Figure. 11
- Invoke Keras to define the InceptionV3 model. Figure. 12
- Process the output of the input_pipeline in the InceptionV3. Figure. 13
- Type the code for the custom function that maps the image path to their respective features. Figure. 14
- Type the code to generate the train and test dataset. Figure. 15



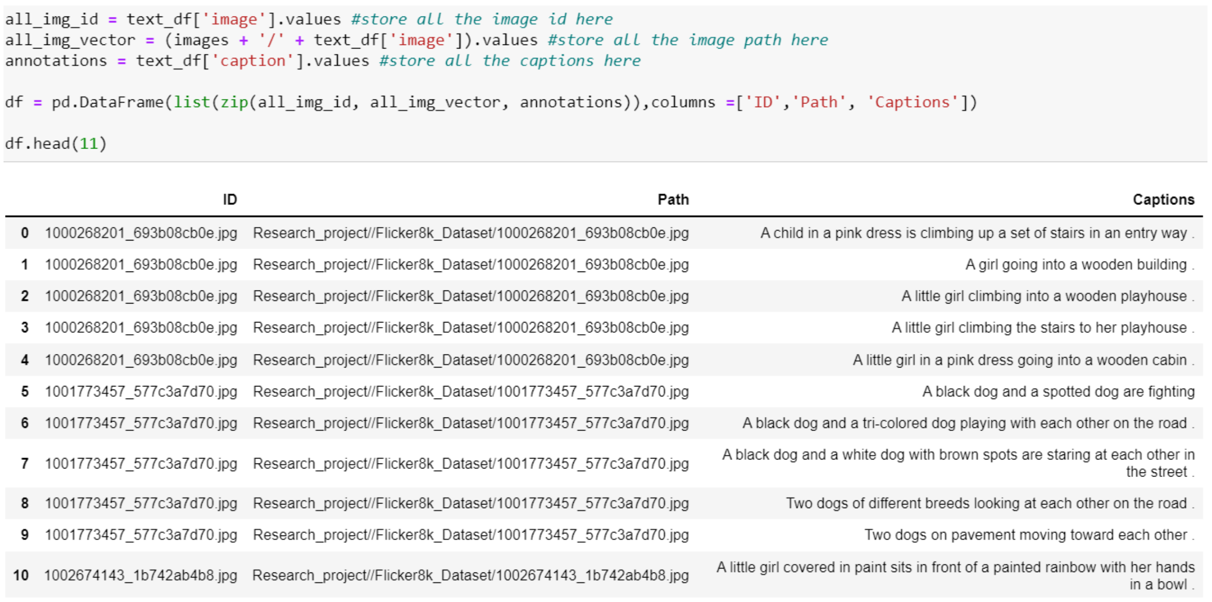
```
def create_file():
    file = open(r"C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project\Flickr8k.lemma.token.txt", "r")
    counter_file = 1
    with open(r'C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project\newfile.text', 'wt') as f:
        for words in file.readlines():
            if counter_file == 1:
                f.write("image,caption")
                f.write("\n")
                counter_file += 1
            line = words.split("#")
            line[1] = line[1].replace("\t", "")
            line[1] = line[1].replace(line[1][0], "")
            line1 = ",".join(line)
            f.write(line1)

    f.close()
    file.close()
    os.rename(r'C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project\newfile.text',
              r'C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project\captions.txt')
    os.remove(r'C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project\Flickr8k.lemma.token.txt')


def killfile():
    del_files = ['CrowdFlowerAnnotations.txt', 'ExpertAnnotations.txt', 'Flickr_8k.devImages.txt', 'Flickr_8k.testImages.txt', 'Flickr_8k.trainImages.t
    for file in del_files:
        os.remove(r'C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project\{}'.format(file))
    shutil.rmtree(r'C:\Users\Amit\Documents\Academics\NCI\Semester_3\Research_project\__MACOSX')
```

Figure 7: Code to Manipulate and Delete Data



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import keras
import glob
import random as rn
from PIL import Image

import tensorflow as tf
rn.seed(42)
tf.random.set_seed(42)
from  tensorflow.keras.models import Model
from tensorflow.keras import layers
from tensorflow.keras.layers import GRU, Embedding, TimeDistributed, Dense, RepeatVector, Activation, Flatten, Reshape, \
                                    concatenate, Dropout, BatchNormalization, Conv2D
from sklearn.model_selection import train_test_split

import warnings
warnings.filterwarnings('ignore')

pd.set_option('display.max_colwidth', None)

from collections import Counter
import string
import imageio
import json
from tqdm import tqdm
import time
from gtts import gTTS
import os
from IPython.display import display, Audio
```

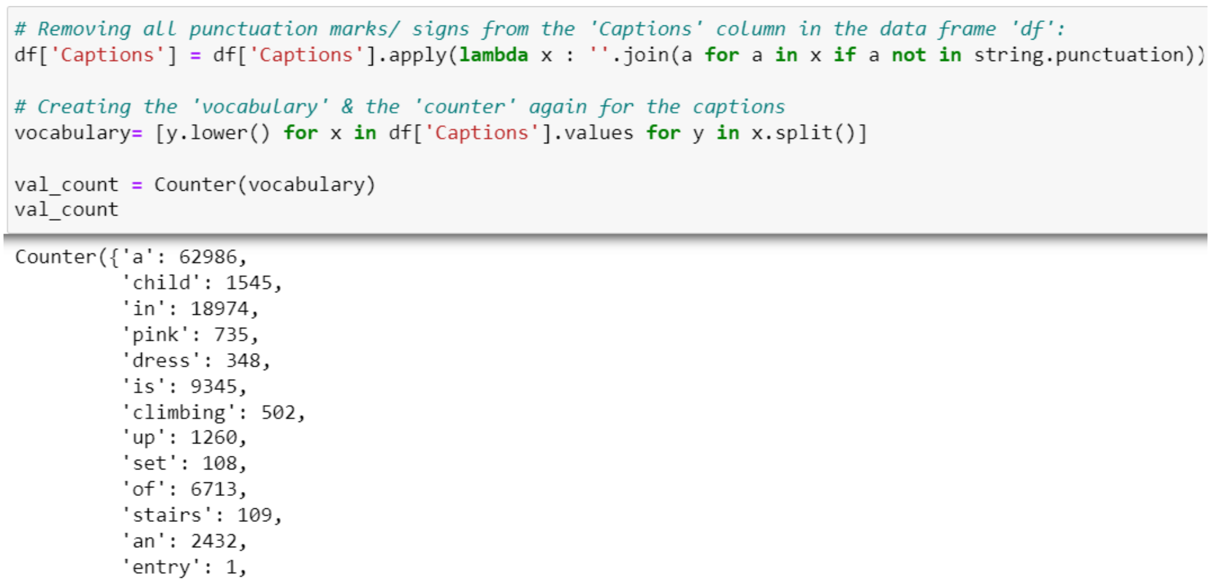
Figure 8: Importing Requisite Libraries in Jupyter



```
all_img_id = text_df['image'].values #store all the image id here
all_img_vector = (images + '/' + text_df['image']).values #store all the image path here
annotations = text_df['caption'].values #store all the captions here

df = pd.DataFrame(list(zip(all_img_id, all_img_vector, annotations)),columns =['ID','Path', 'Captions'])

df.head(11)
```

	ID	Path	Captions
0	1000268201_693b08cb0e.jpg	Research_project//Flicker8k_Dataset/1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set of stairs in an entry way .
1	1000268201_693b08cb0e.jpg	Research_project//Flicker8k_Dataset/1000268201_693b08cb0e.jpg	A girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	Research_project//Flicker8k_Dataset/1000268201_693b08cb0e.jpg	A little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	Research_project//Flicker8k_Dataset/1000268201_693b08cb0e.jpg	A little girl climbing the stairs to her playhouse .
4	1000268201_693b08cb0e.jpg	Research_project//Flicker8k_Dataset/1000268201_693b08cb0e.jpg	A little girl in a pink dress going into a wooden cabin .
5	1001773457_577c3a7d70.jpg	Research_project//Flicker8k_Dataset/1001773457_577c3a7d70.jpg	A black dog and a spotted dog are fighting
6	1001773457_577c3a7d70.jpg	Research_project//Flicker8k_Dataset/1001773457_577c3a7d70.jpg	A black dog and a tri-colored dog playing with each other on the road .
7	1001773457_577c3a7d70.jpg	Research_project//Flicker8k_Dataset/1001773457_577c3a7d70.jpg	A black dog and a white dog with brown spots are staring at each other in the street .
8	1001773457_577c3a7d70.jpg	Research_project//Flicker8k_Dataset/1001773457_577c3a7d70.jpg	Two dogs of different breeds looking at each other on the road .
9	1001773457_577c3a7d70.jpg	Research_project//Flicker8k_Dataset/1001773457_577c3a7d70.jpg	Two dogs on pavement moving toward each other .
10	1002674143_1b742ab4b8.jpg	Research_project//Flicker8k_Dataset/1002674143_1b742ab4b8.jpg	A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .

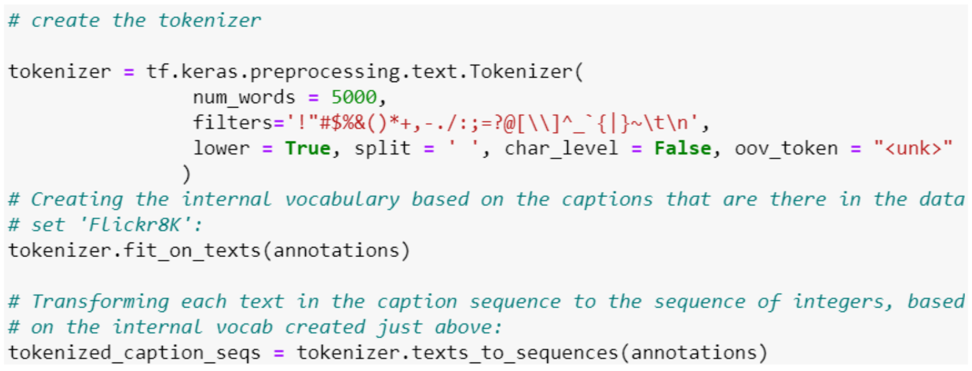
Figure 9: Create Pandas DataFrame to Contain ID's, Paths and captions



```
# Removing all punctuation marks/ signs from the 'Captions' column in the data frame 'df':
df['Captions'] = df['Captions'].apply(lambda x : ''.join(a for a in x if a not in string.punctuation))

# Creating the 'vocabulary' & the 'counter' again for the captions
vocabulary= [y.lower() for x in df['Captions'].values for y in x.split()]

val_count = Counter(vocabulary)
val_count
```

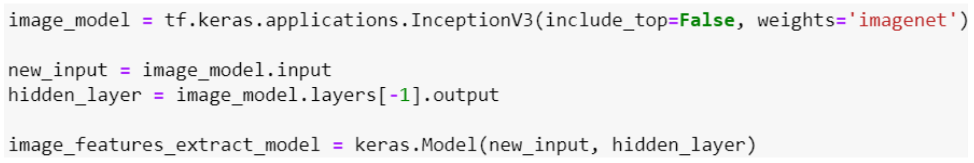
```
Counter({'a': 62986,
         'child': 1545,
         'in': 18974,
         'pink': 735,
         'dress': 348,
         'is': 9345,
         'climbing': 502,
         'up': 1260,
         'set': 108,
         'of': 6713,
         'stairs': 109,
         'an': 2432,
         'entry': 1,
```

Figure 10: Code for Vocab without Punctuation Marks



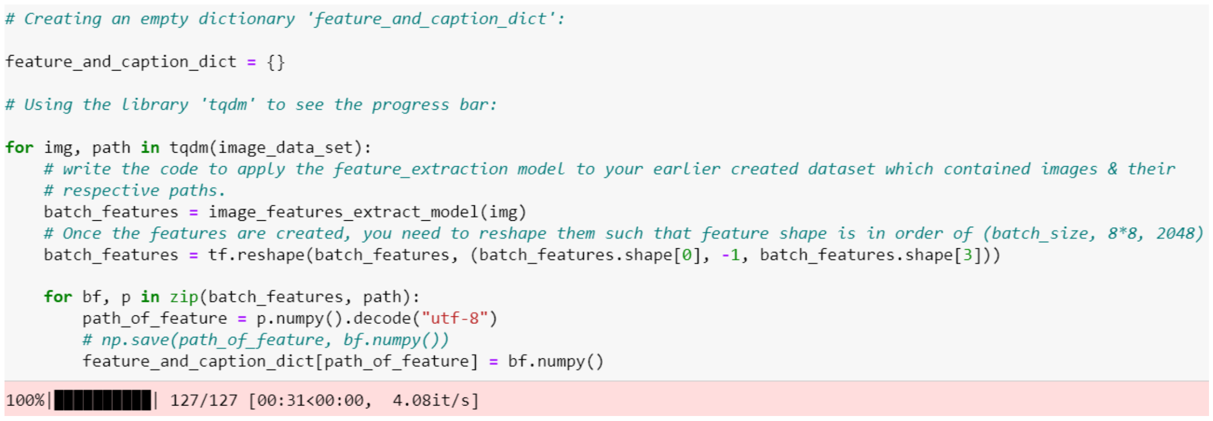
```
# create the tokenizer

tokenizer = tf.keras.preprocessing.text.Tokenizer(
                num_words = 5000,
                filters='!"#$%&()*+,-./:;=?@[\\]^_`{|}~\t\n',
                lower = True, split = ' ', char_level = False, oov_token = "<unk>"
               )
# Creating the internal vocabulary based on the captions that are there in the data
# set 'Flickr8K':
tokenizer.fit_on_texts(annotations)

# Transforming each text in the caption sequence to the sequence of integers, based
# on the internal vocab created just above:
tokenized_caption_seqs = tokenizer.texts_to_sequences(annotations)
```

Figure 11: Code for caption pre-processing



```
image_model = tf.keras.applications.InceptionV3(include_top=False, weights='imagenet')

new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = keras.Model(new_input, hidden_layer)
```

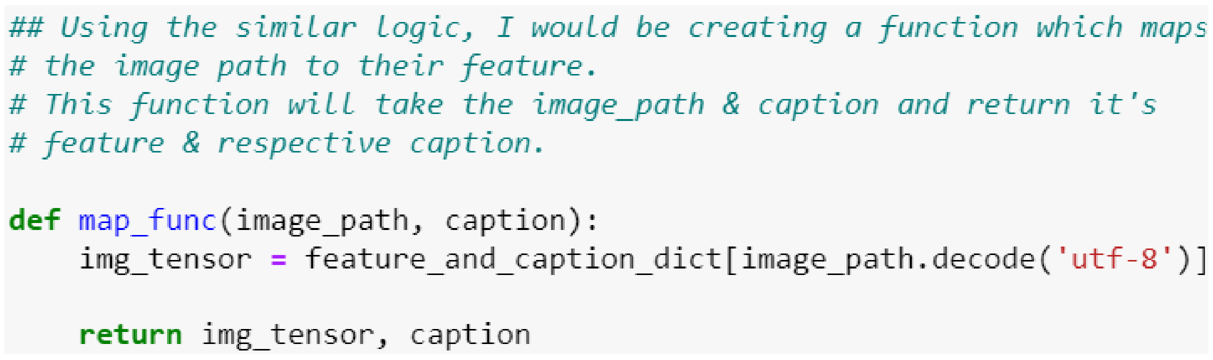
Figure 12: Code for InceptionV3 model



```
# Creating an empty dictionary 'feature_and_caption_dict':

feature_and_caption_dict = {}

# Using the library 'tqdm' to see the progress bar:

for img, path in tqdm(image_data_set):
    # write the code to apply the feature_extraction model to your earlier created dataset which contained images & their
    # respective paths.
    batch_features = image_features_extract_model(img)
    # Once the features are created, you need to reshape them such that feature shape is in order of (batch_size, 8*8, 2048)
    batch_features = tf.reshape(batch_features, (batch_features.shape[0], -1, batch_features.shape[3]))

    for bf, p in zip(batch_features, path):
        path_of_feature = p.numpy().decode("utf-8")
        # np.save(path_of_feature, bf.numpy())
        feature_and_caption_dict[path_of_feature] = bf.numpy()
```

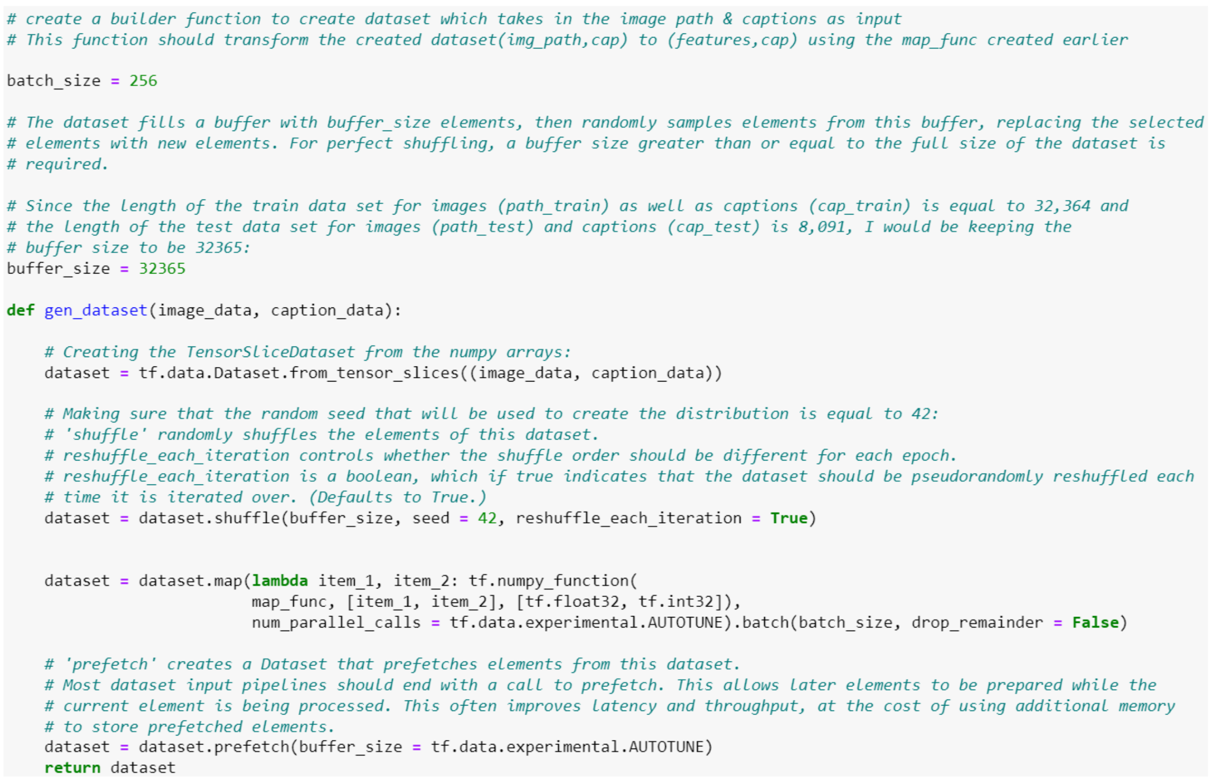
```
100%|██████████| 127/127 [00:31<00:00,  4.08it/s]
```

Figure 13: Code for processing the data in the base model



```
## Using the similar logic, I would be creating a function which maps
# the image path to their feature.
# This function will take the image_path & caption and return it's
# feature & respective caption.

def map_func(image_path, caption):
    img_tensor = feature_and_caption_dict[image_path.decode('utf-8')]

    return img_tensor, caption
```

Figure 14: Custom Function for Mapping image path to their features



```python
# create a builder function to create dataset which takes in the image path & captions as input
# This function should transform the created dataset(img_path,cap) to (features,cap) using the map_func created earlier

batch_size = 256

# The dataset fills a buffer with buffer_size elements, then randomly samples elements from this buffer, replacing the selected
# elements with new elements. For perfect shuffling, a buffer size greater than or equal to the full size of the dataset is
# required.

# Since the length of the train data set for images (path_train) as well as captions (cap_train) is equal to 32,364 and
# the length of the test data set for images (path_test) and captions (cap_test) is 8,091, I would be keeping the
# buffer size to be 32365:
buffer_size = 32365

def gen_dataset(image_data, caption_data):

    # Creating the TensorSliceDataset from the numpy arrays:
    dataset = tf.data.Dataset.from_tensor_slices((image_data, caption_data))

    # Making sure that the random seed that will be used to create the distribution is equal to 42:
    # 'shuffle' randomly shuffles the elements of this dataset.
    # reshuffle_each_iteration controls whether the shuffle order should be different for each epoch.
    # reshuffle_each_iteration is a boolean, which if true indicates that the dataset should be pseudorandomly reshuffled each
    # time it is iterated over. (Defaults to True.)
    dataset = dataset.shuffle(buffer_size, seed = 42, reshuffle_each_iteration = True)


    dataset = dataset.map(lambda item_1, item_2: tf.numpy_function(
                      map_func, [item_1, item_2], [tf.float32, tf.int32]),
                      num_parallel_calls = tf.data.experimental.AUTOTUNE).batch(batch_size, drop_remainder = False)

    # 'prefetch' creates a Dataset that prefetches elements from this dataset.
    # Most dataset input pipelines should end with a call to prefetch. This allows later elements to be prepared while the
    # current element is being processed. This often improves latency and throughput, at the cost of using additional memory
    # to store prefetched elements.
    dataset = dataset.prefetch(buffer_size = tf.data.experimental.AUTOTUNE)
    return dataset
```

Figure 15: Function to generate test and train datasets

3.3 Model Training and Testing

- Type the code to train and test the model and calculate the train and test losses over 50 epochs. Figure. 16

3.4 Model Evaluation

- Code to plot the graph for train and test losses. Figure. 17
- Plot the train and tess loss graph Figure. 18
- Type the code for predicting the captions (Greedy Search). Figure. 19
- Type the code for predicting the captions (Beam Search). Figure. 20 and 21
- Type the code to predict using greedy search and evaluate using the BLEU score. 22
- Type the code to predict using beam search. 23
- Type the code for transforming the caption in to speech. 24

```
# Code to train and test the model based on the processed data set:

loss_plot = []
test_loss_plot = []
epochs = 50

best_test_loss = 100
for epoch in tqdm(range(0, epochs)):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(train_dataset):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss
        avg_train_loss=total_loss / train_num_steps

    loss_plot.append(avg_train_loss)
    test_loss = test_loss_cal(test_dataset)
    test_loss_plot.append(test_loss)

    print ('For epoch: {}, the train loss is {:.3f}, & test loss is {:.3f}'.format(epoch+1,avg_train_loss,test_loss))
    print ('Time taken for 1 epoch {} sec\n'.format(time.time() - start))

    if test_loss < best_test_loss:
        print('Test loss has been reduced from %.3f to %.3f' % (best_test_loss, test_loss))
        best_test_loss = test_loss
        ckpt_manager.save()
```

Figure 16: Training and Testing the Model and calculating Losses

```
plt.figure(figsize=(12, 6))
plt.plot(loss_plot)
plt.plot(test_loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
# plt.legend(loc='best')
plt.show()
```

Figure 17: Code to Plot the train and test losses

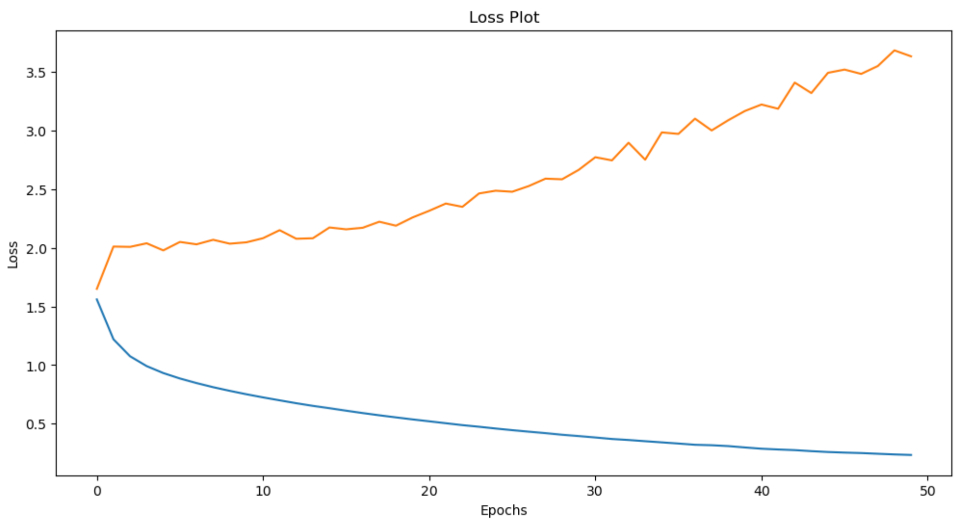
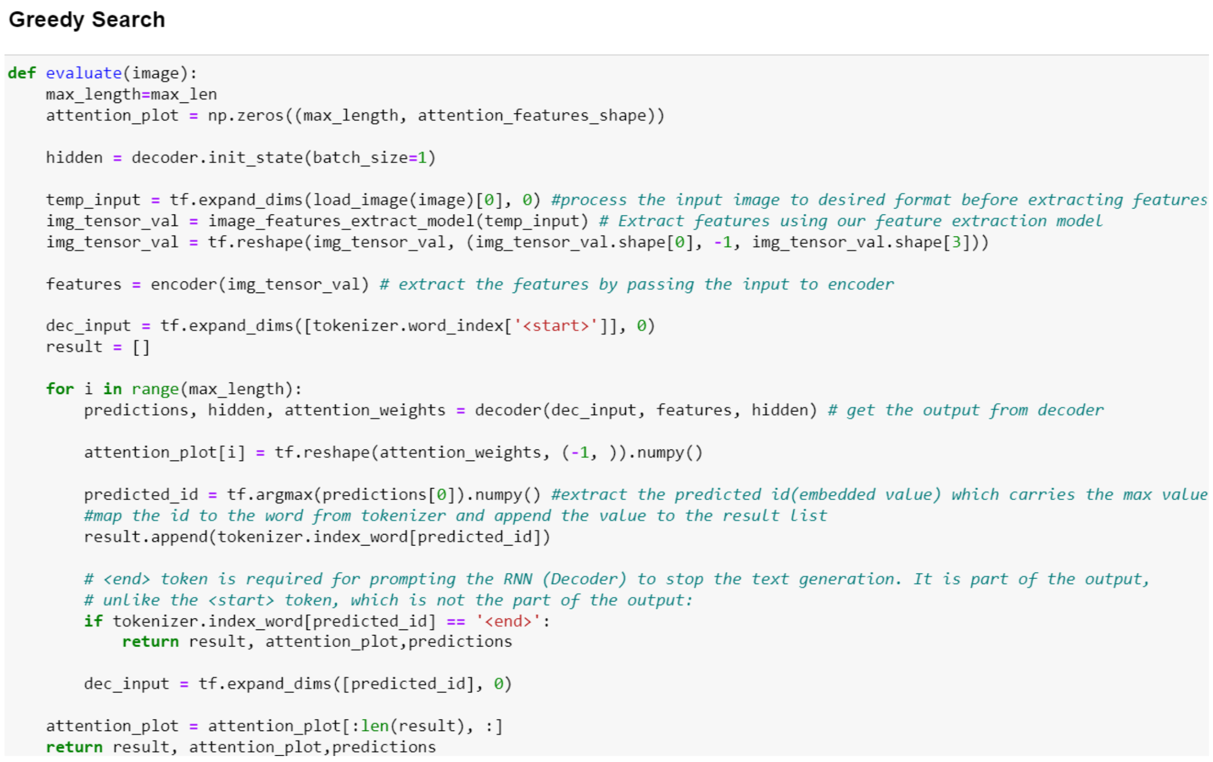


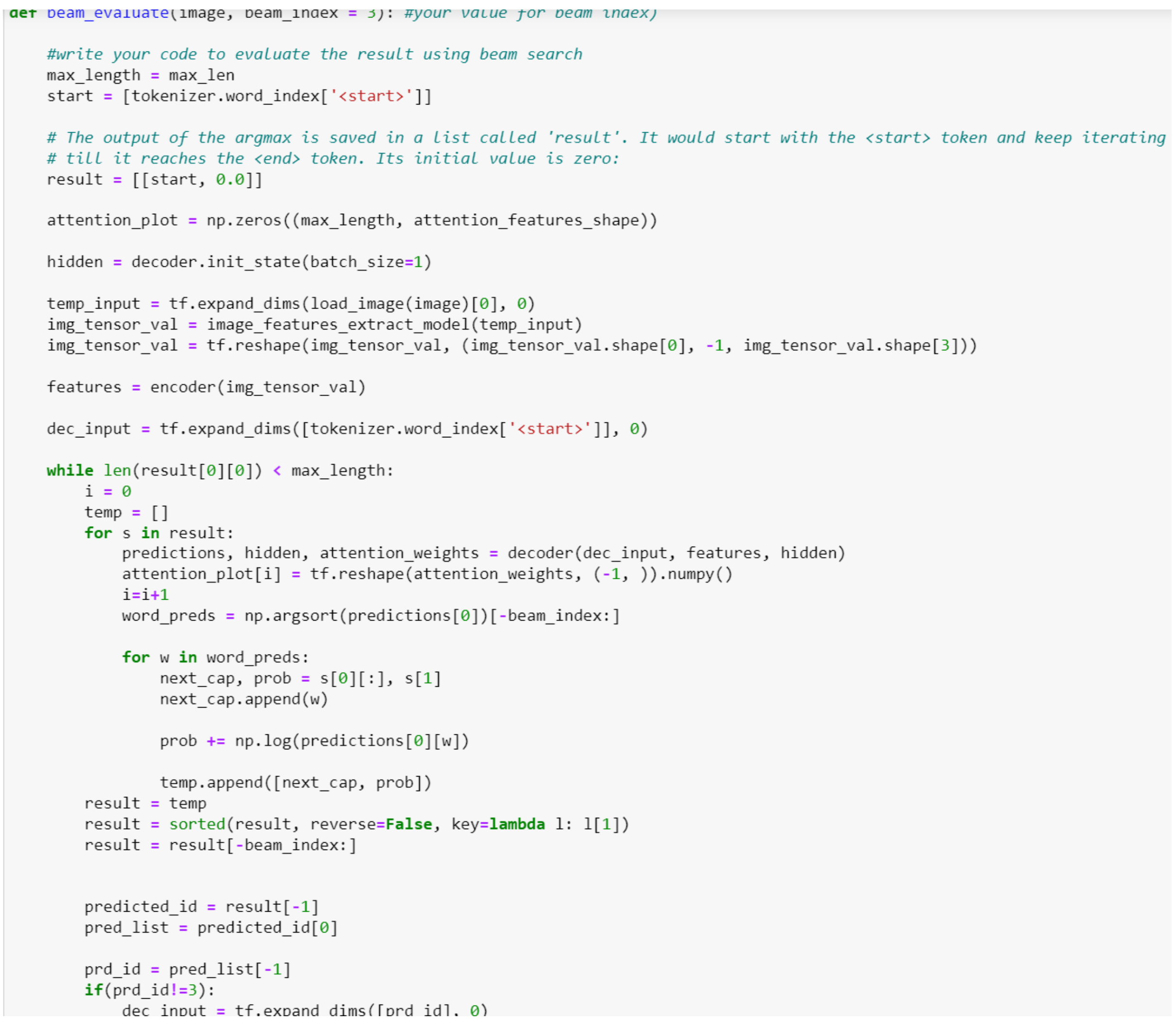
Figure 18: Graph of the train and test losses



Greedy Search

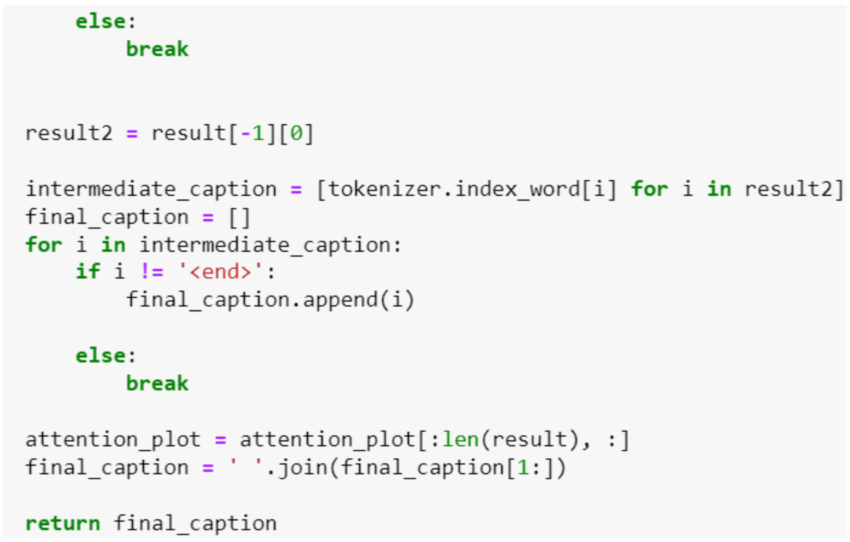
```python
def evaluate(image):
    max_length=max_len
    attention_plot = np.zeros((max_length, attention_features_shape))

    hidden = decoder.init_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0) #process the input image to desired format before extracting features
    img_tensor_val = image_features_extract_model(temp_input) # Extract features using our feature extraction model
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0], -1, img_tensor_val.shape[3]))

    features = encoder(img_tensor_val) # extract the features by passing the input to encoder

    dec_input = tf.expand_dims([tokenizer.word_index['<start>']], 0)
    result = []

    for i in range(max_length):
        predictions, hidden, attention_weights = decoder(dec_input, features, hidden) # get the output from decoder

        attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()

        predicted_id = tf.argmax(predictions[0]).numpy() #extract the predicted id(embedded value) which carries the max value
        #map the id to the word from tokenizer and append the value to the result list
        result.append(tokenizer.index_word[predicted_id])

        # <end> token is required for prompting the RNN (Decoder) to stop the text generation. It is part of the output,
        # unlike the <start> token, which is not the part of the output:
        if tokenizer.index_word[predicted_id] == '<end>':
            return result, attention_plot,predictions

        dec_input = tf.expand_dims([predicted_id], 0)

    attention_plot = attention_plot[:len(result), :]
    return result, attention_plot,predictions
```

Figure 19: Code for Prediction - Greedy Search



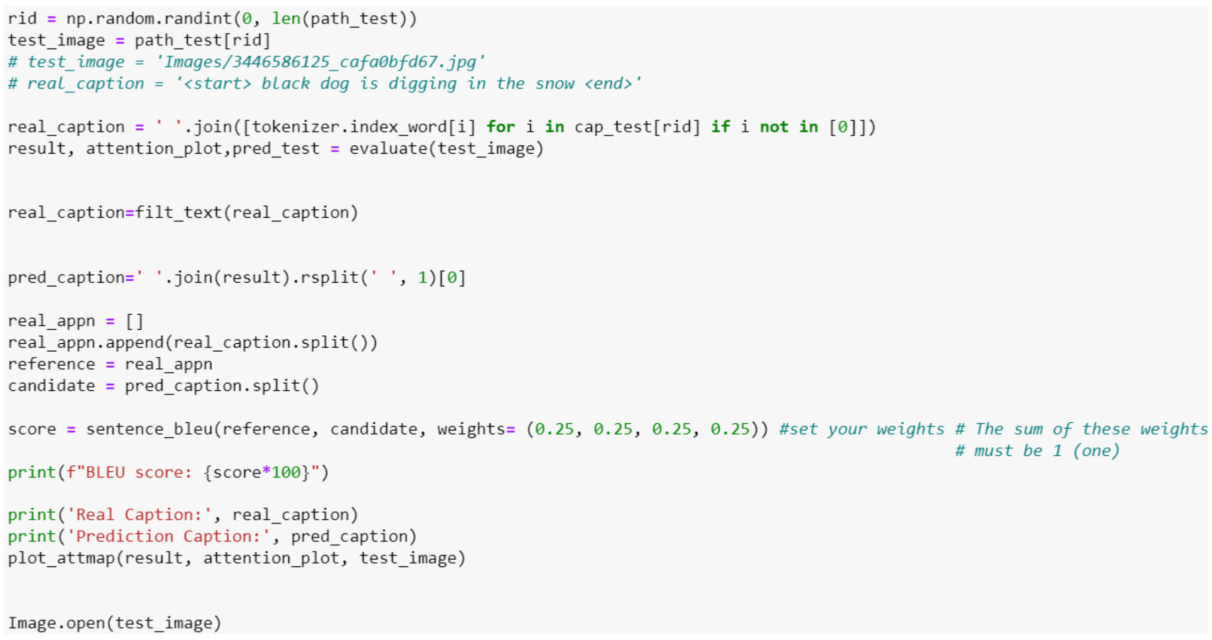
```
def beam_evaluate(image, beam_index = 3): #your value for beam index)

    #write your code to evaluate the result using beam search
    max_length = max_len
    start = [tokenizer.word_index['<start>']]

    # The output of the argmax is saved in a list called 'result'. It would start with the <start> token and keep iterating
    # till it reaches the <end> token. Its initial value is zero:
    result = [[start, 0.0]]

    attention_plot = np.zeros((max_length, attention_features_shape))

    hidden = decoder.init_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0)
    img_tensor_val = image_features_extract_model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0], -1, img_tensor_val.shape[3]))

    features = encoder(img_tensor_val)

    dec_input = tf.expand_dims([tokenizer.word_index['<start>']], 0)

    while len(result[0][0]) < max_length:
        i = 0
        temp = []
        for s in result:
            predictions, hidden, attention_weights = decoder(dec_input, features, hidden)
            attention_plot[i] = tf.reshape(attention_weights, (-1, )).numpy()
            i=i+1
            word_preds = np.argsort(predictions[0])[-beam_index:]

            for w in word_preds:
                next_cap, prob = s[0][:], s[1]
                next_cap.append(w)

                prob += np.log(predictions[0][w])

                temp.append([next_cap, prob])
        result = temp
        result = sorted(result, reverse=False, key=lambda l: l[1])
        result = result[-beam_index:]


        predicted_id = result[-1]
        pred_list = predicted_id[0]

        prd_id = pred_list[-1]
        if(prd_id!=3):
            dec_input = tf.expand_dims([prd_id], 0)
```

Figure 20: Code for Prediction - Beam Search - Part 1



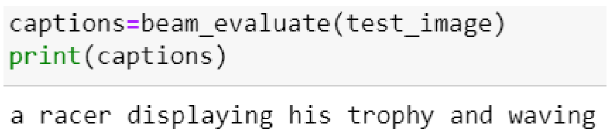
```
        else:
            break


    result2 = result[-1][0]

    intermediate_caption = [tokenizer.index_word[i] for i in result2]
    final_caption = []
    for i in intermediate_caption:
        if i != '<end>':
            final_caption.append(i)

        else:
            break

    attention_plot = attention_plot[:len(result), :]
    final_caption = ' '.join(final_caption[1:])

    return final_caption
```

Figure 21: Code for Prediction - Beam Search - Part 2



```python
rid = np.random.randint(0, len(path_test))
test_image = path_test[rid]
# test_image = 'Images/3446586125_cafa0bfd67.jpg'
# real_caption = '<start> black dog is digging in the snow <end>'

real_caption = ' '.join([tokenizer.index_word[i] for i in cap_test[rid] if i not in [0]])
result, attention_plot,pred_test = evaluate(test_image)


real_caption=filt_text(real_caption)


pred_caption=' '.join(result).rsplit(' ', 1)[0]

real_appn = []
real_appn.append(real_caption.split())
reference = real_appn
candidate = pred_caption.split()

score = sentence_bleu(reference, candidate, weights= (0.25, 0.25, 0.25, 0.25)) #set your weights # The sum of these weights
                                                                                              # must be 1 (one)
print(f"BLEU score: {score*100}")

print('Real Caption:', real_caption)
print('Prediction Caption:', pred_caption)
plot_attmap(result, attention_plot, test_image)


Image.open(test_image)
```

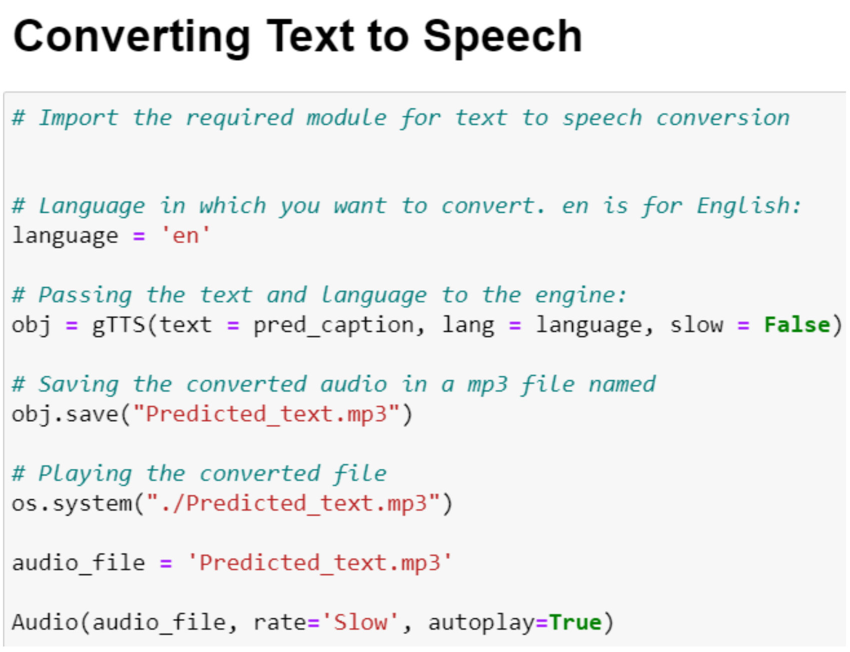
Figure 22: Code for execution of prediction - Greedy Search - And Evaluate - BLEU



```python
captions=beam_evaluate(test_image)
print(captions)
```

```
a racer displaying his trophy and waving
```

Figure 23: Code for the execution of Prediction - Beam Search



Converting Text to Speech

```python
# Import the required module for text to speech conversion


# Language in which you want to convert. en is for English:
language = 'en'

# Passing the text and language to the engine:
obj = gTTS(text = pred_caption, lang = language, slow = False)

# Saving the converted audio in a mp3 file named
obj.save("Predicted_text.mp3")

# Playing the converted file
os.system("./Predicted_text.mp3")

audio_file = 'Predicted_text.mp3'

Audio(audio_file, rate='Slow', autoplay=True)
```

Figure 24: Code for text to speech generation