AI-Transl:A Multilingual Assistive Device Using Multi Modal Machine Learning

Lekshmy H O

Supervisor Dr.Swaminathan

c

Amrita School of Engineering ,Amrita Vishwa vidyapeedam

November 1,2021

Outline

Background Information

Bind statistics

Country Name	Blind Statistics
India	9.2 M
China	8.9 M
Indonesia	3.7 M
Pakistan	1.8 M
Brazil	1.8 M
Nigeria	1.3 M
Bangladesh	0.9 M
USA	0.6 M
Russia	0.6 M
Mexico	0.5 M

Figure 1: Top countries having highest blind population

Issues faced by blinds in Developing Countries

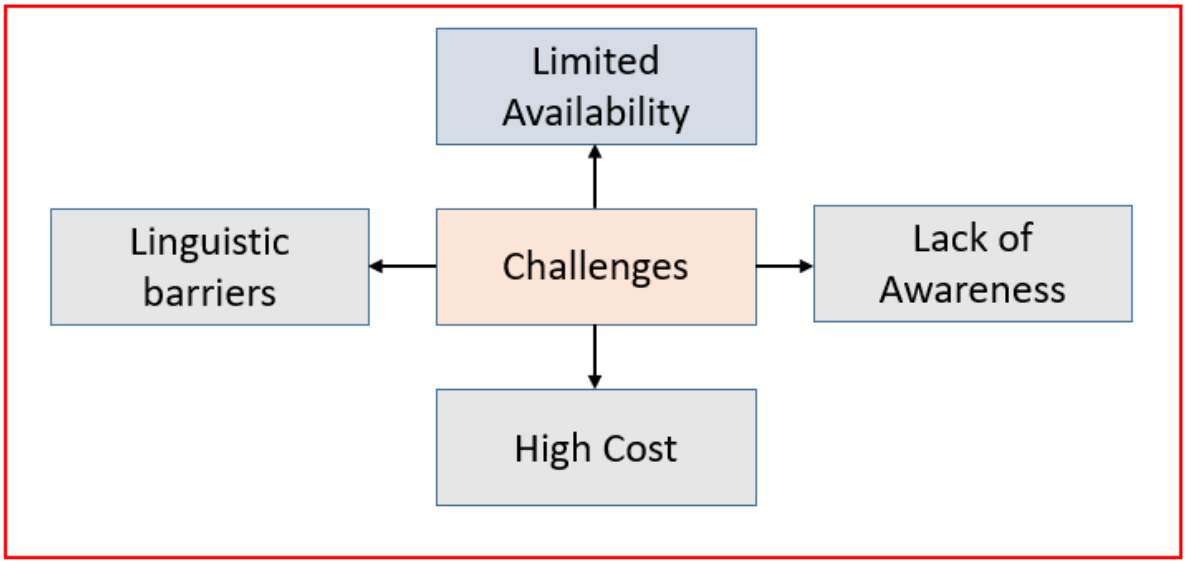


Figure 2: Linguistic barrier is a key problem in multilingual developing countries.Translation APIs are often used to solve this problem.

Common Practices

Widely used Machine Translation Algorithms

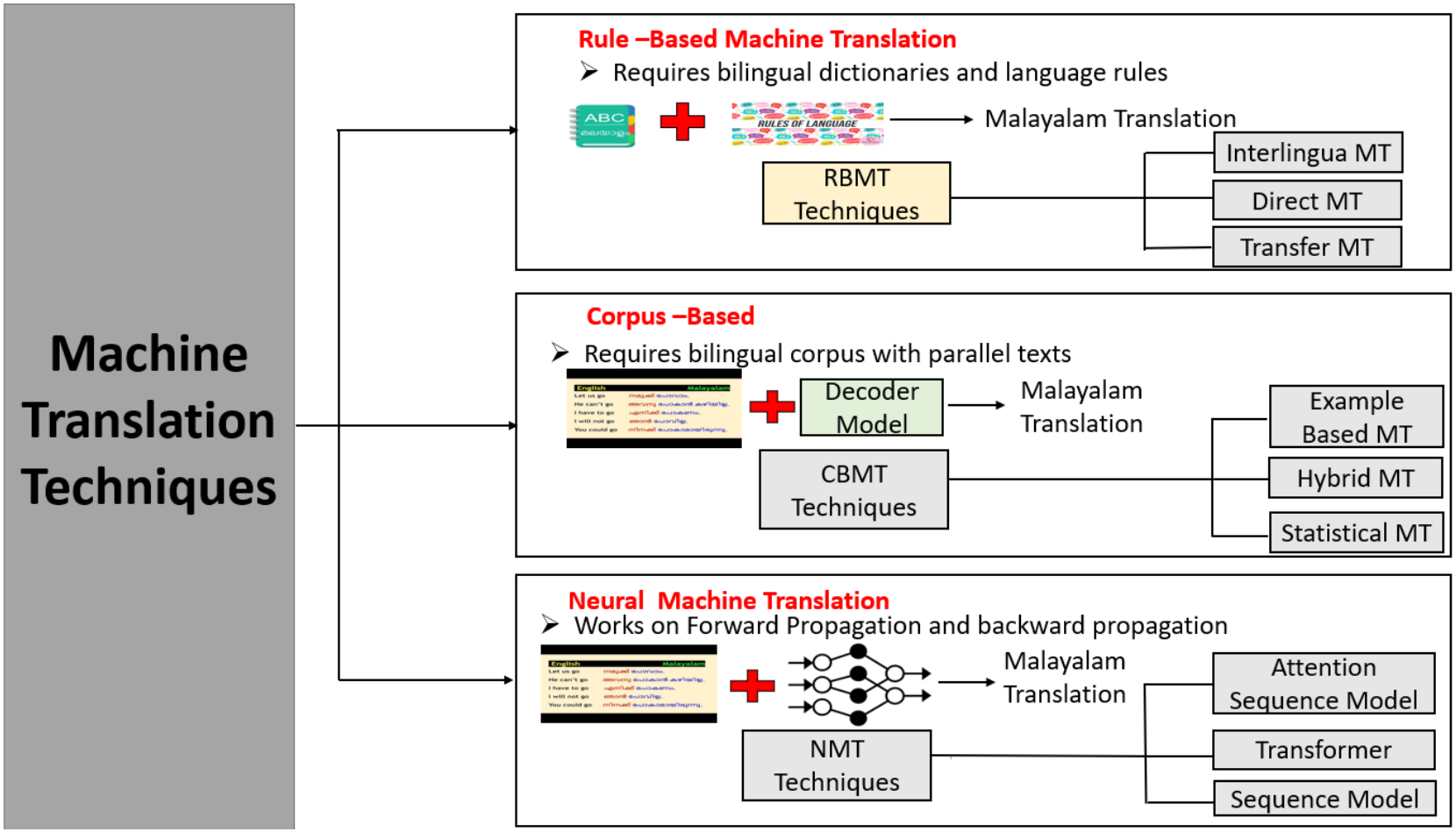


Figure 3: Indicates most commonly used machine translation techniques.

Outline

Machine Translation-Challenges

Challenges involved in Low Resource Language Translation

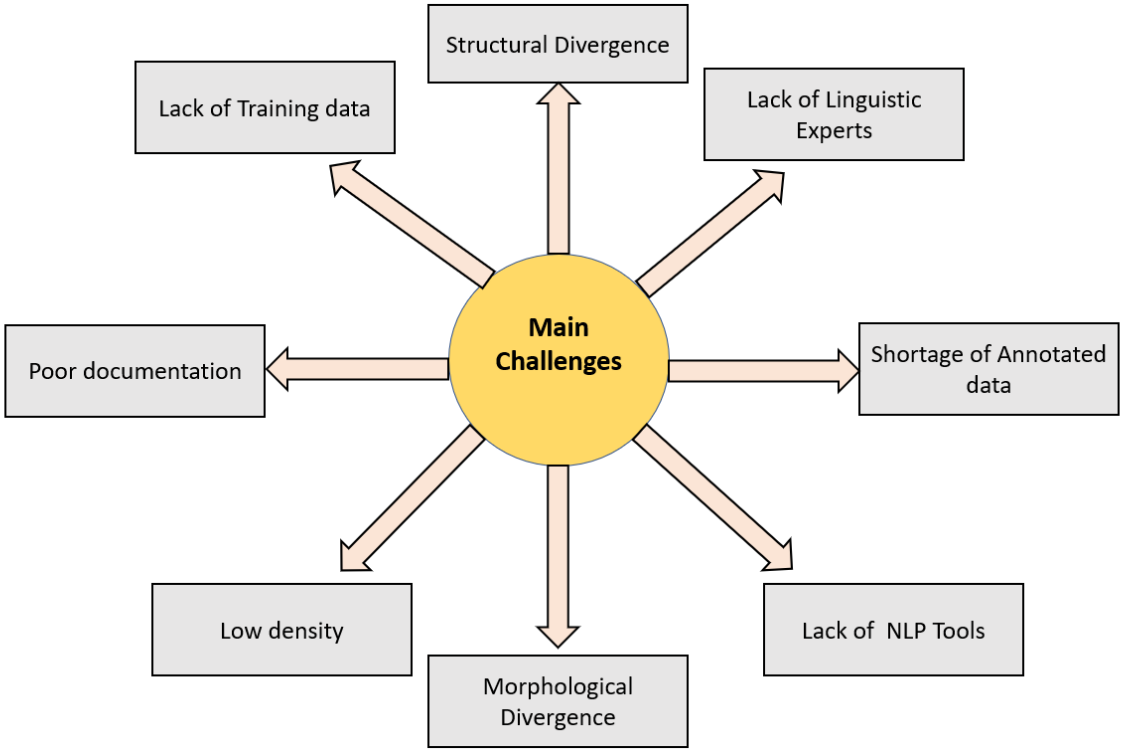
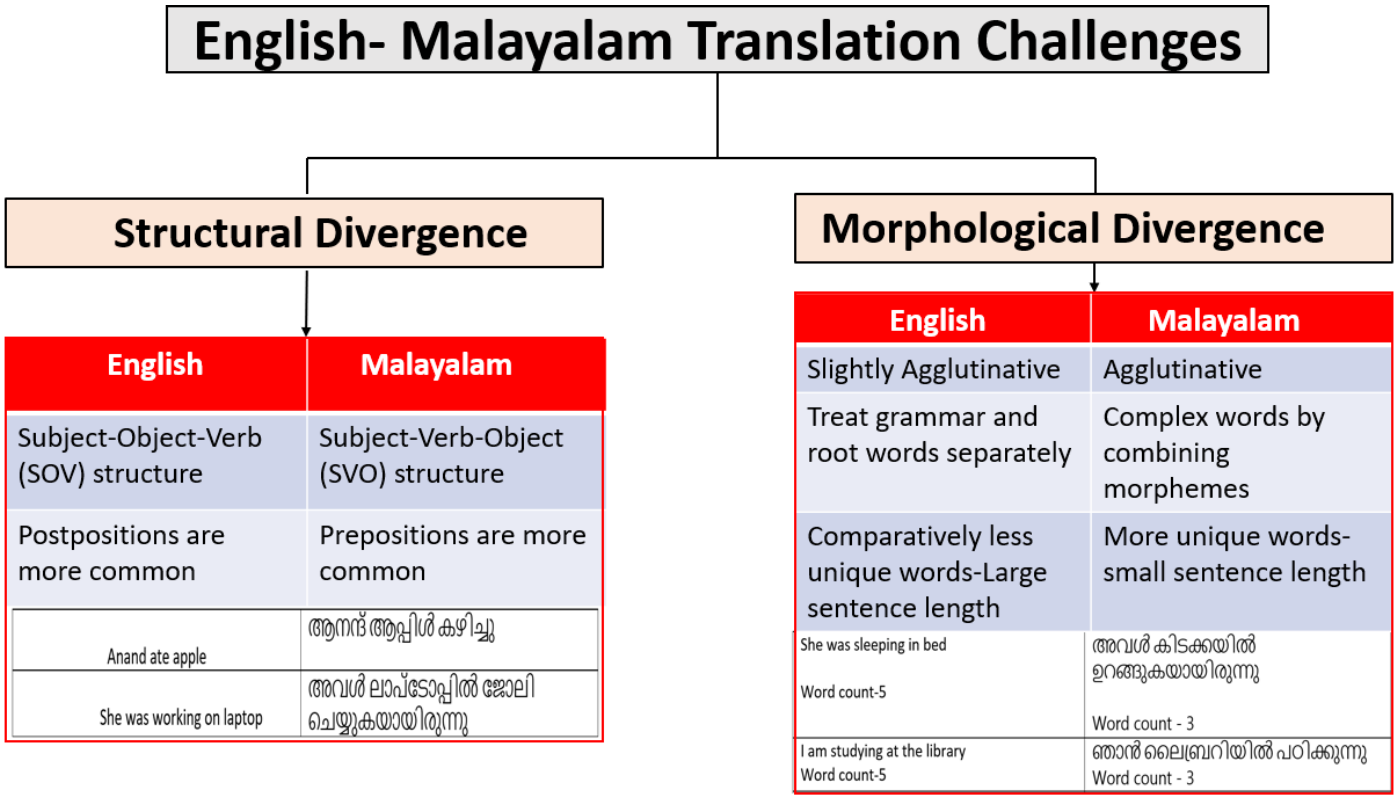


Figure 4: Major challenges associated with the translation of low resource languages like Indian languages.

English -Dravidian Translation Challenges

Challenges involved in English - Dravidian Language Translation



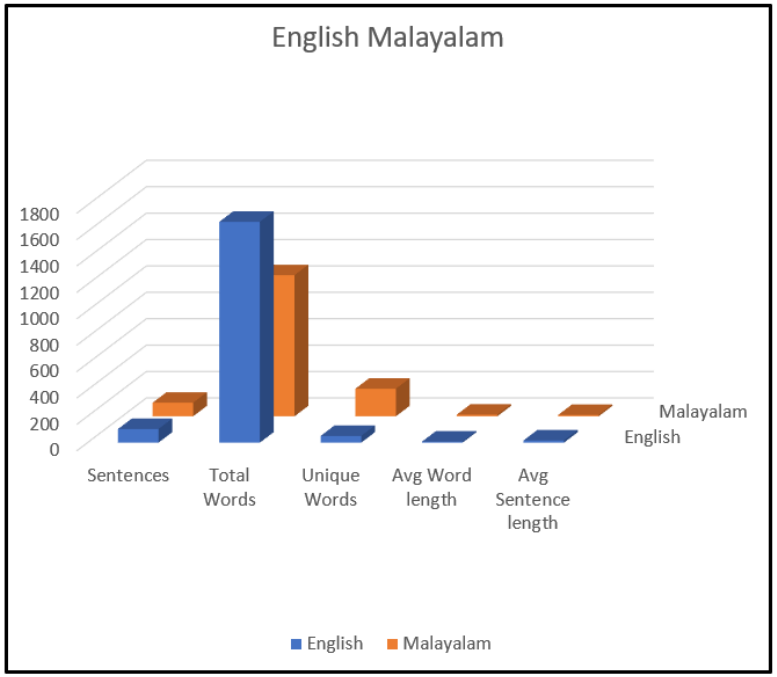
English	Malayalam
Subject-Object-Verb (SOV) structure	Subject-Verb-Object (SVO) structure
Postpositions are more common	Prepositions are more common
Anand ate apple	ആനന്ദ് ആപ്പിൾ കഴിച്ചു
She was working on laptop	അവൾ ലാപ്ടോപ്പിൽ ജോലി ചെയ്യുകയായിരുന്നു

English	Malayalam
Slightly Agglutinative	Agglutinative
Treat grammar and root words separately	Complex words by combining morphemes
Comparatively less unique words-Large sentence length	More unique words-small sentence length
She was sleeping in bed Word count-5	അവൾ കിടക്കയിൽ ഉറങ്ങുകയായിരുന്നു Word count - 3
I am studying at the library Word count-5	ഞാൻ ലൈബ്രറിയിൽ പഠിക്കുന്നു Word count - 3

Figure 5: Challenges involved in English -Dravidian Translation is explained with the help of Malayalam language which belongs to Dravidian linguistic group

English -Dravidian Translation Challenges

Comparative Analysis of English and Malayalam



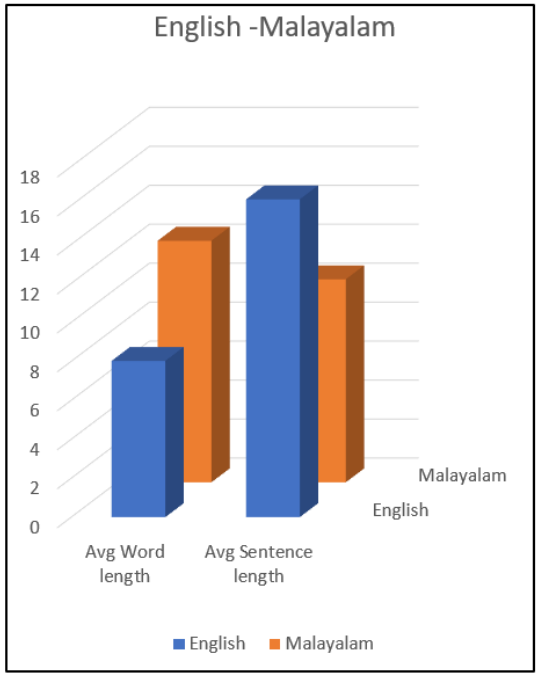


Figure 6: Comparative analysis performed on 109 k sentences in both English and Malayalam languages. Graph indicates difference arises due to morphological as well as structural divergence.

Problem

Drawback of Existing Machine Translation Models

Rule Based MT	Corpus Based MT	Neural Machine Translation
Insufficient number of good dictionaries	Corpus creation can be costly.	Corpus creation can be costly.
Hard to deal with rule interactions in big systems, ambiguity, and idiomatic expressions	Specific errors are hard to predict and fix.	Structural and morphological divergence effects the translation quality
Failure to adapt to new domains.	Works less well for language pairs with significantly different word order.	Large amount of data is essential for training.
Need to linguistic experts to derive language rules.	Doesn't work well on low resource language.	Produce literal translations, ignores context .

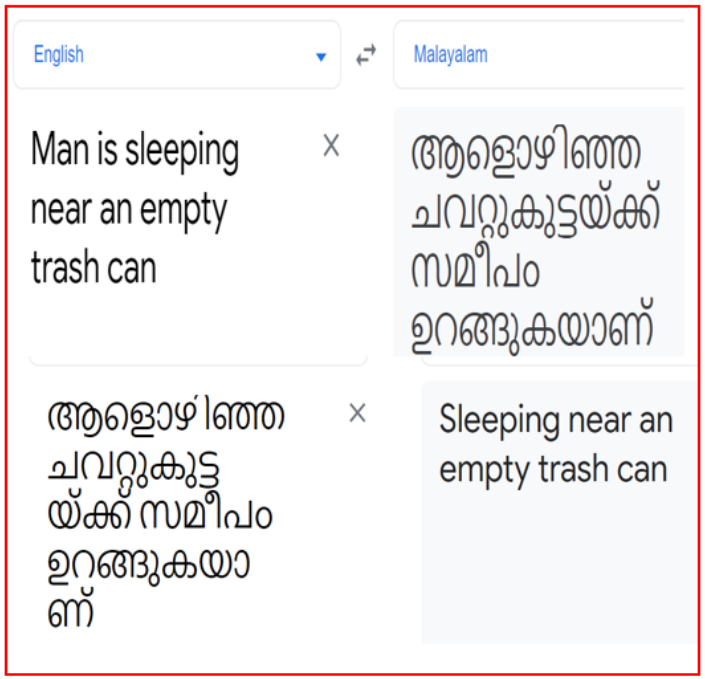


Figure indicates wrong translations produced by existing translation API. The model was not able to identify the semantic difference between the words man and empty.

Figure 7: Shows problems associated with exiting machine translation systems

Outline

Solution Approach

Multi modal Machine Translation vs Traditional Neural Machine Translation

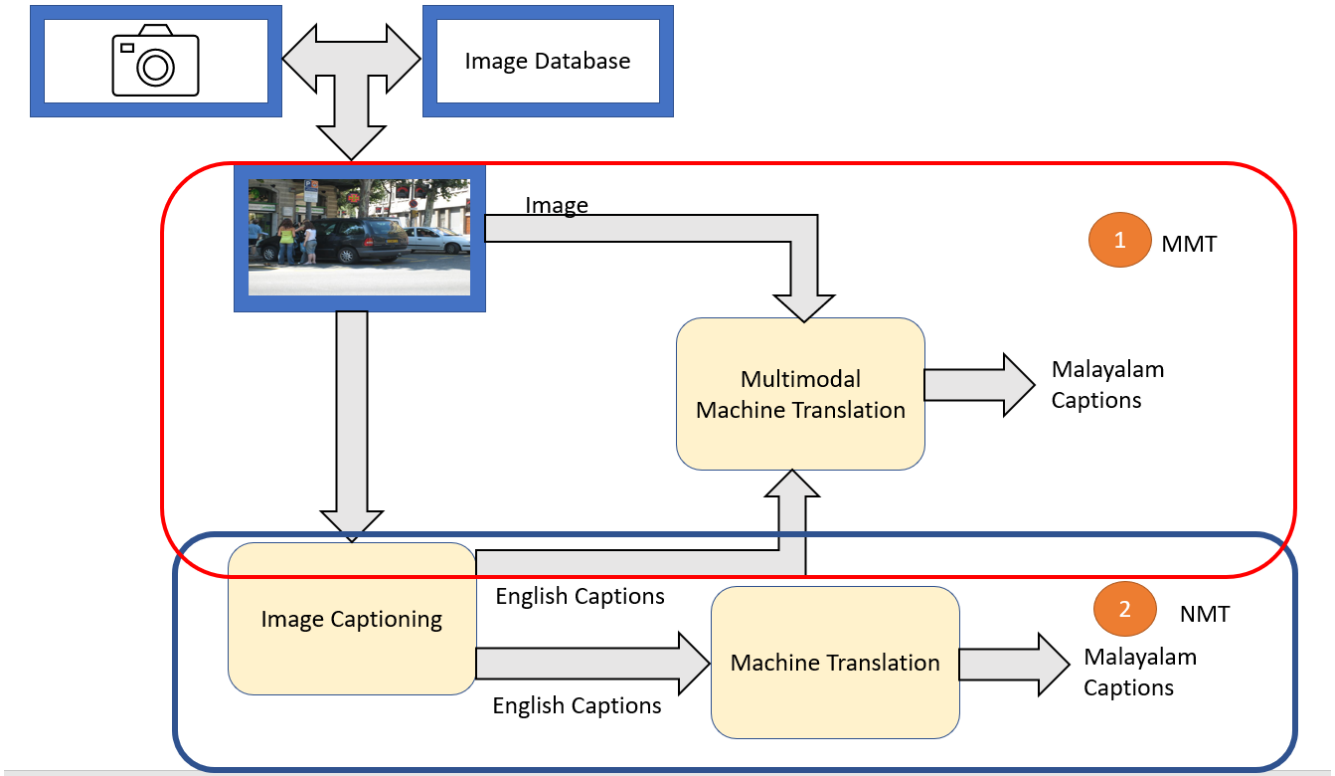


Figure 8: Difference between Multi modal machine Translation and Traditional Neural Machine Translation .

Methodology

Encoder-Decoder Architecture For Multi modal Machine Translation

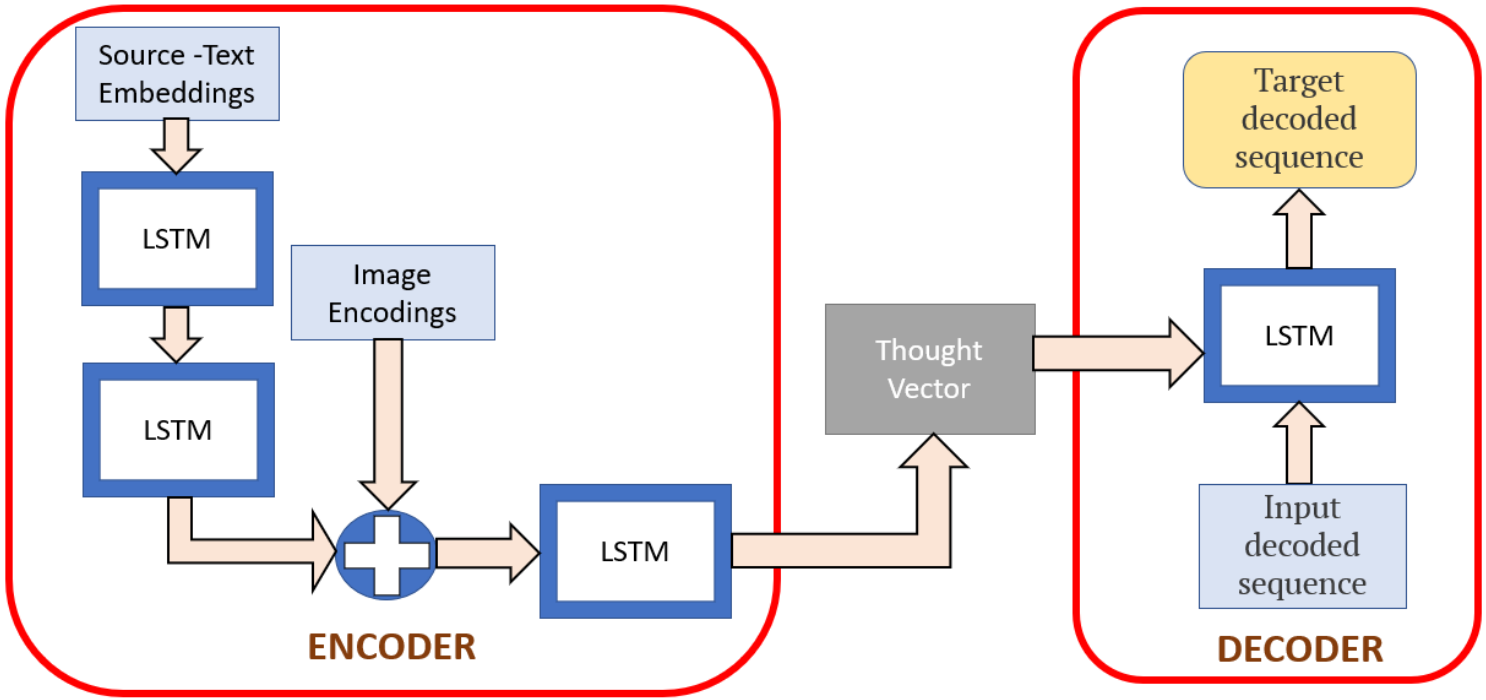


Figure 9: Thought vector is formed by concatenating text embedding with image encoding. This is used to initialize decoder.

Detailed Methodology

Encoder-Decoder Architecture For Multi modal Machine Translation

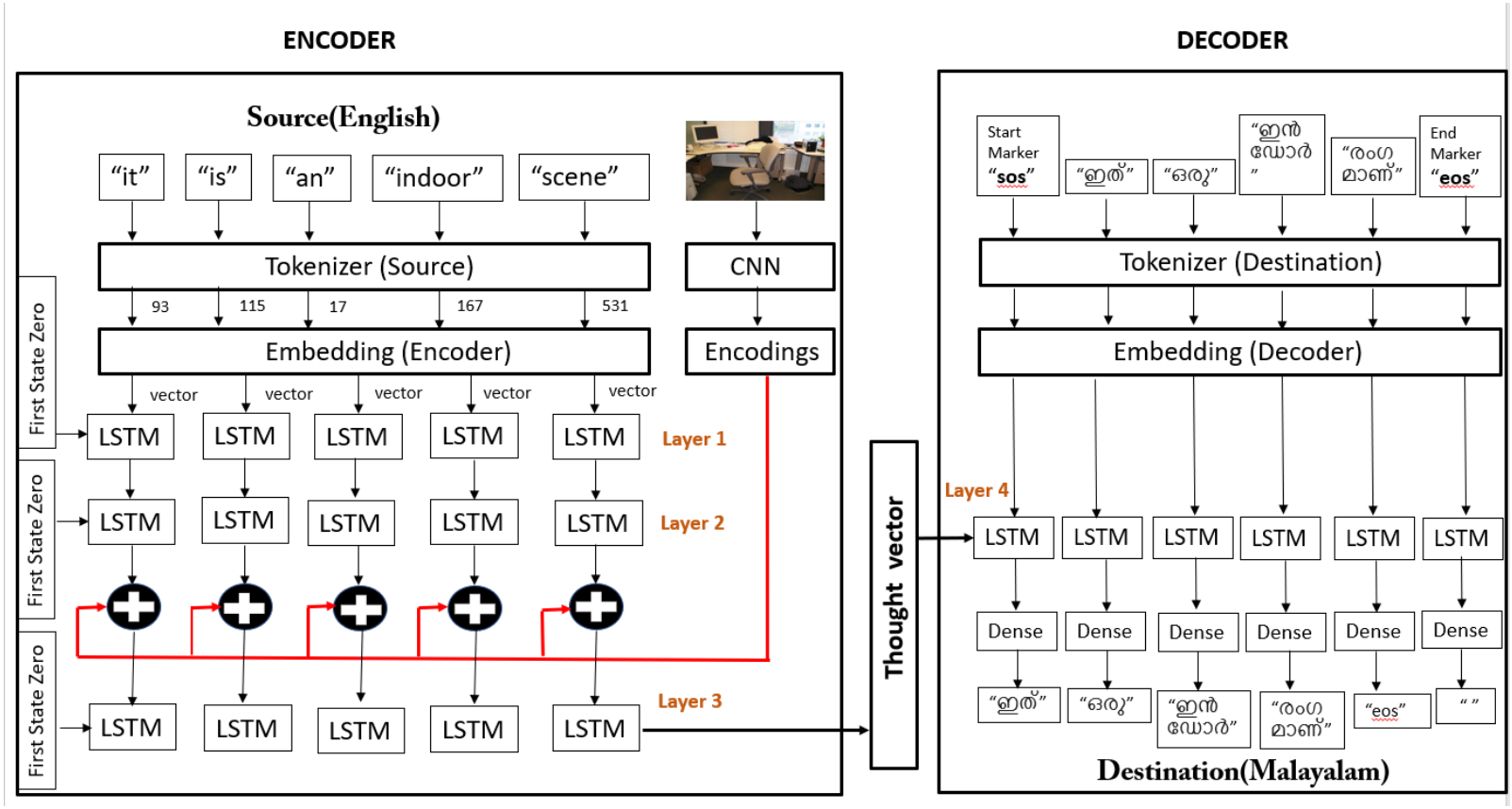


Figure 10: Insights of Encoder-decoder architecture

Detailed Methodology

Encoder-Decoder Architecture(Training phase)

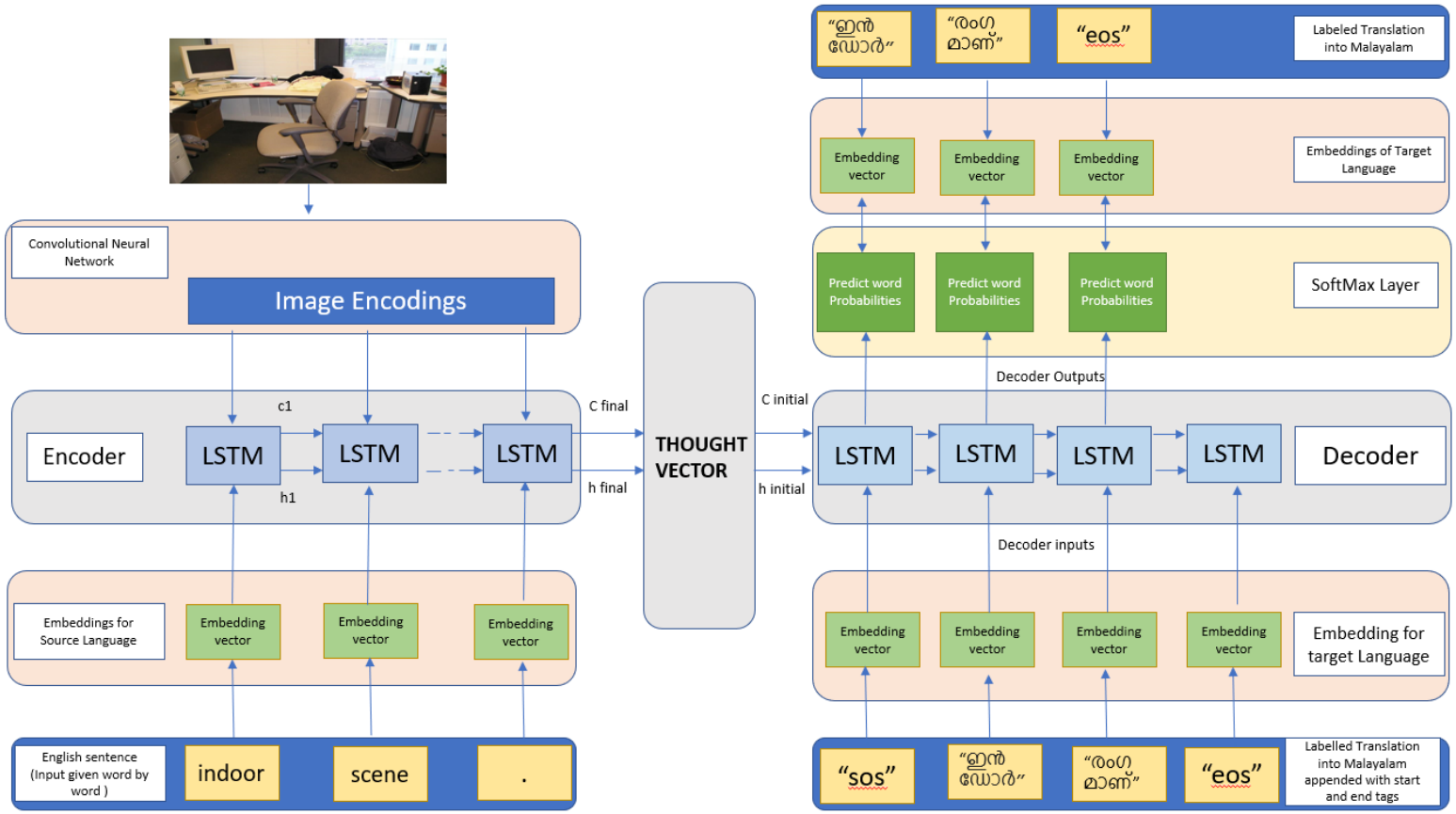


Figure 11: Insights of Encoder-decoder architecture

Detailed Methodology

Encoder-Decoder Architecture(Inference phase)

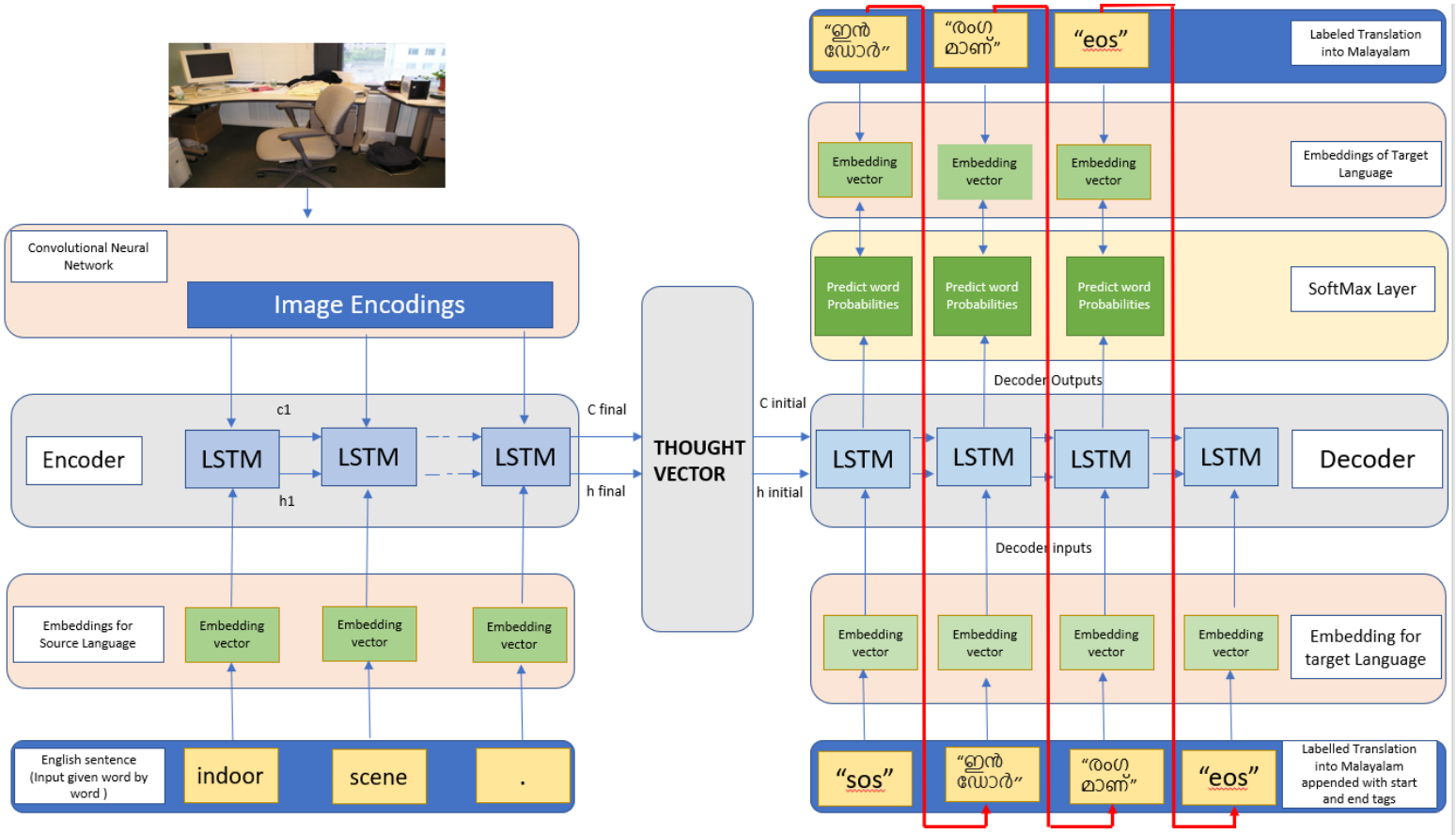


Figure 12: Insights of Encoder-decoder architecture

Detailed Methodology

Encoder-Decoder Architecture with Attention

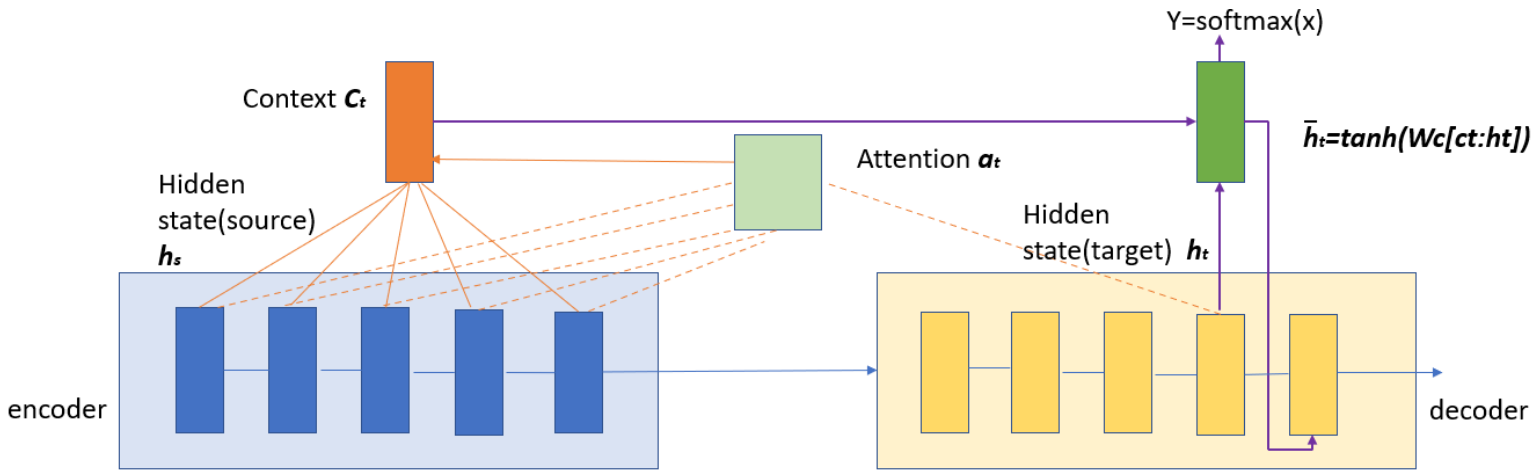


Figure 13: Insights of Encoder-decoder architecture

Outline

Proposed Model using MMT technology

Schematic Diagram

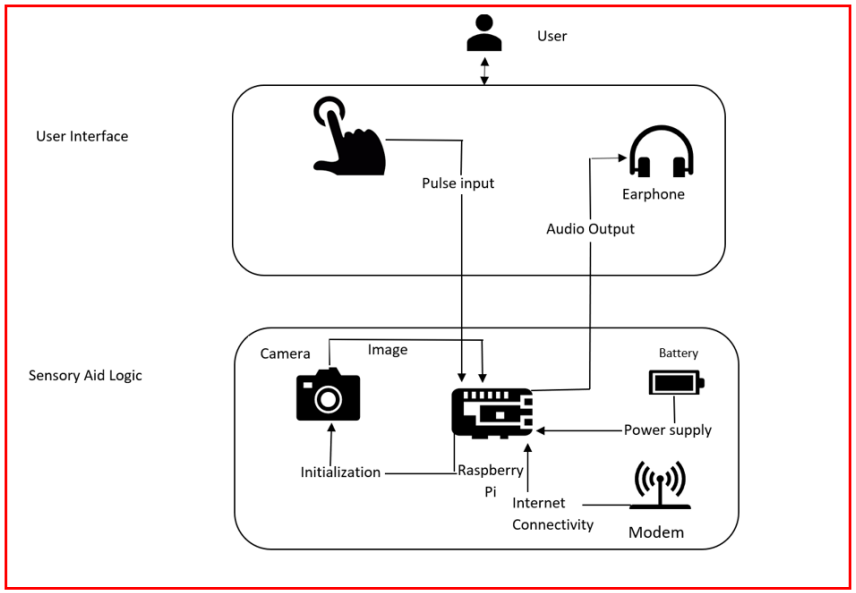


Figure 14: Schematic Diagram of AI-Transl

Work Flow

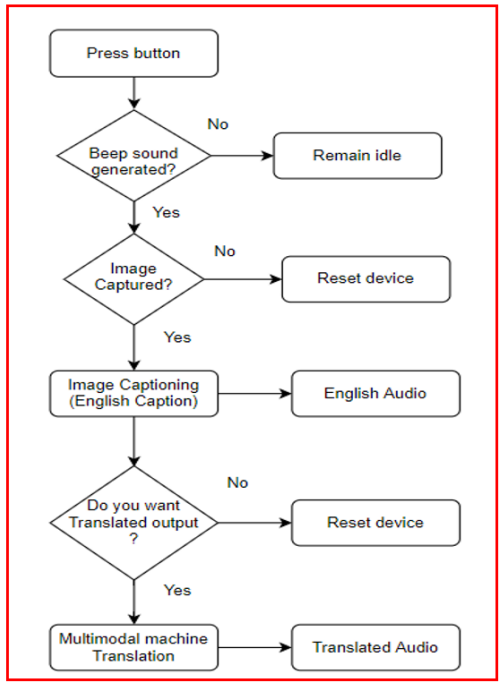


Figure 15: Work Flow Diagram of AI-Transl

Implementation technique used in AI-Transl

Techniques used inside AI-Transl

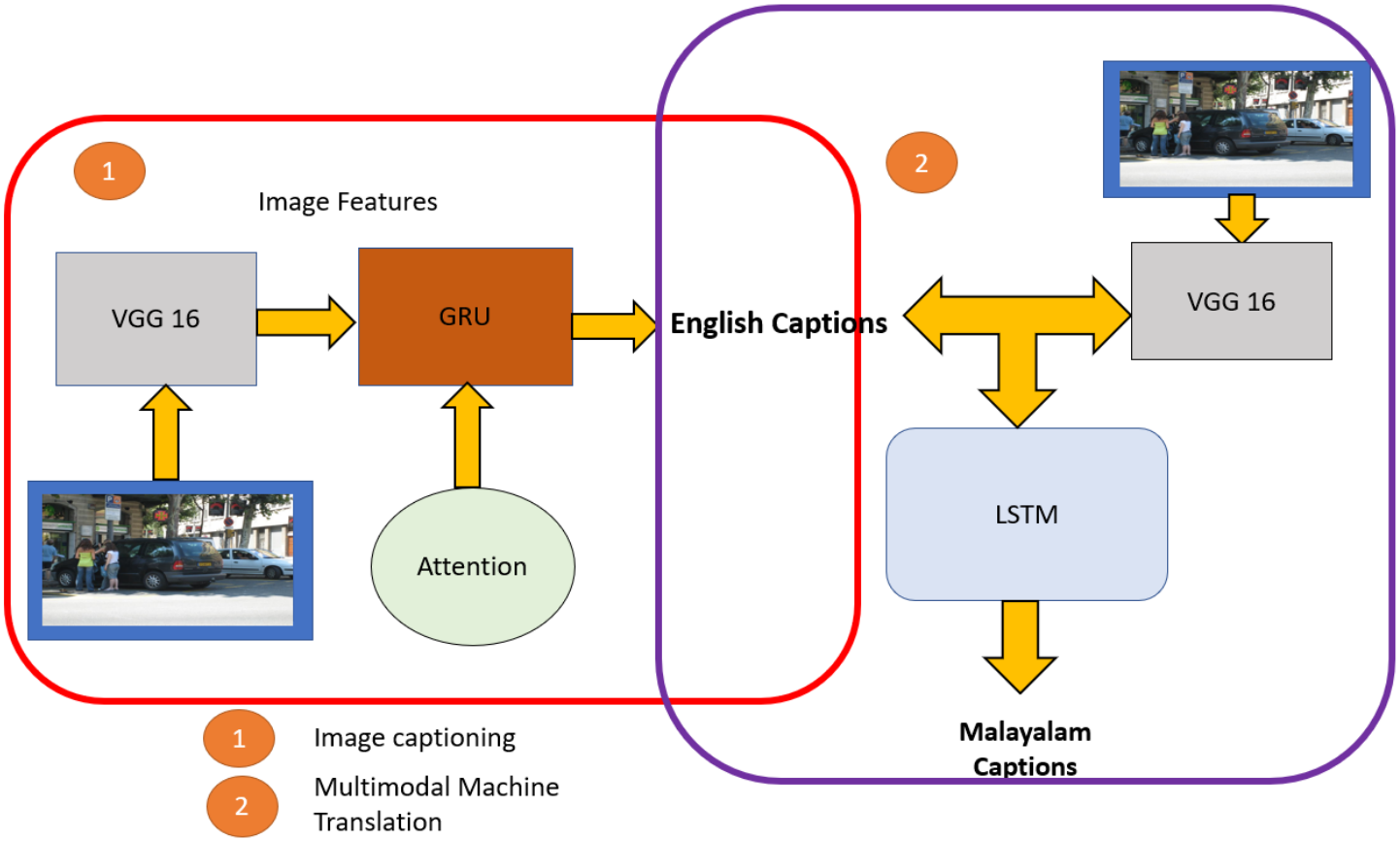


Figure 16: AI Transl incorporate Image Captioning and Multi modal Machine Translation Techniques to produce efficient translations

Working of Model

Detailed Working of Model

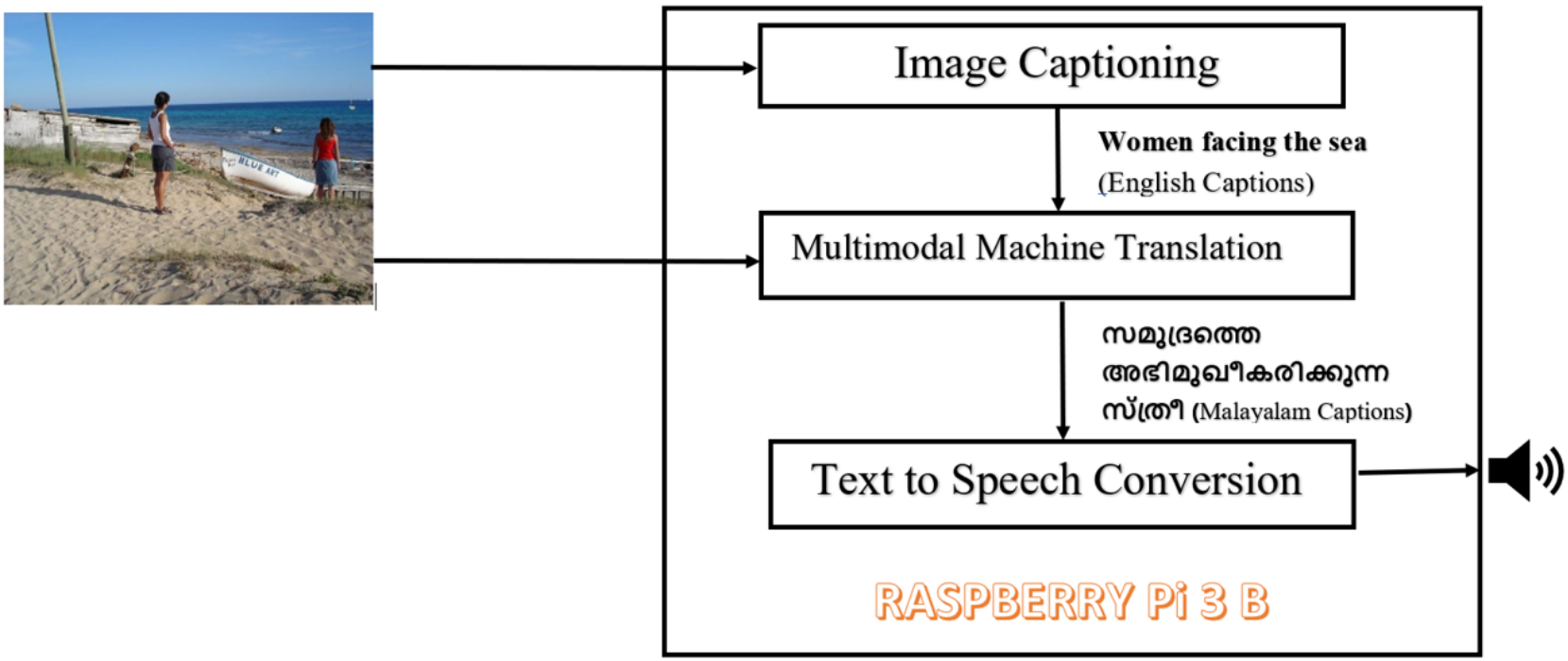


Figure 17: Image captured will undergoes image captioning and Multi modal machine translation to produce English caption and corresponding Malayalam captions.

Outline

Dataset

Details of Dataset

	English – Malayalam Multimodal MT	Image Captioning
Dataset	Visual Genome Malayalam	Coco
Size	Training- 20k Testing -1.6k Validation-1 k	330 k
Release Year	2021	2014
Content	Images+English captions+Malayalam captions	Images+English Captions
Root Dataset	Coco dataset	

Figure 18: DataSets used for training Models

Results

Multi modal Machine Translation- Without Attention

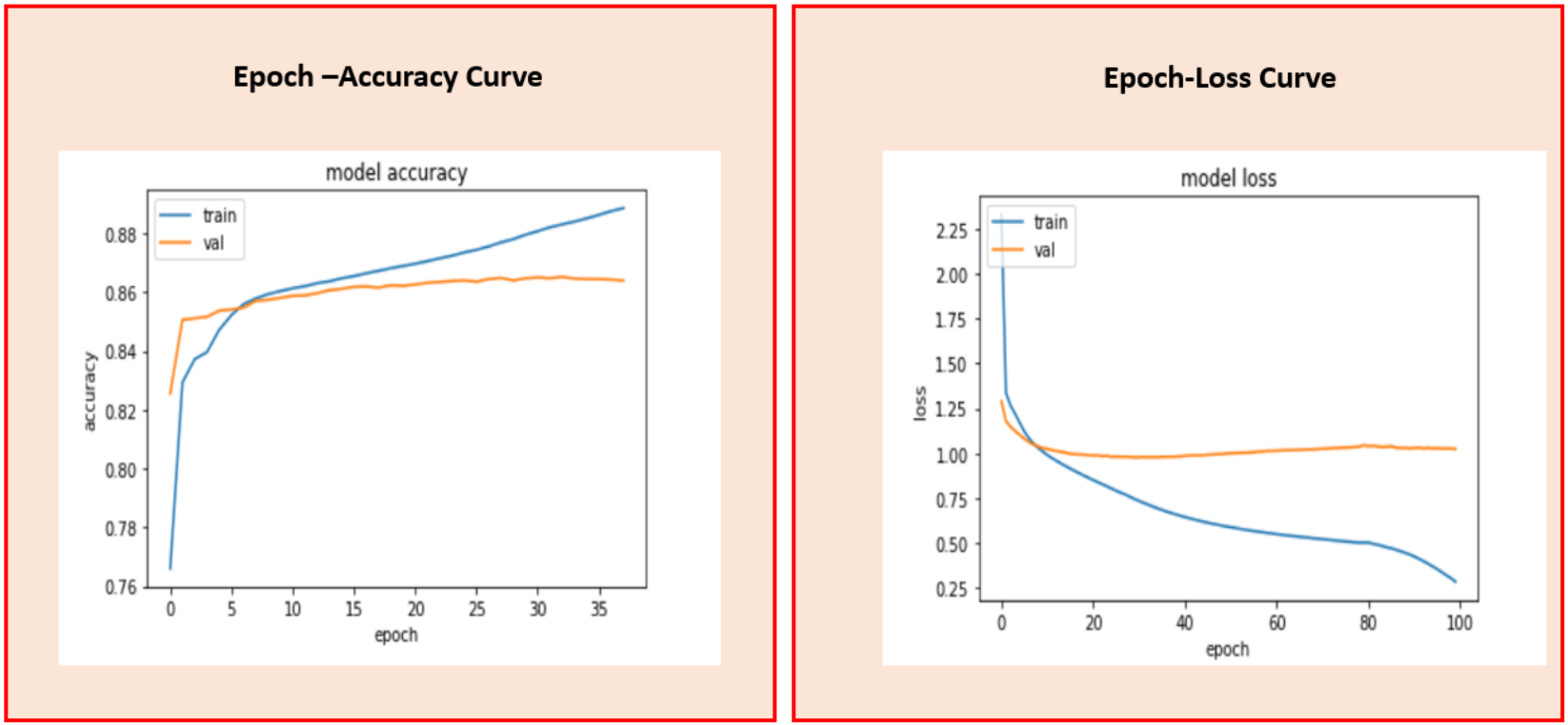
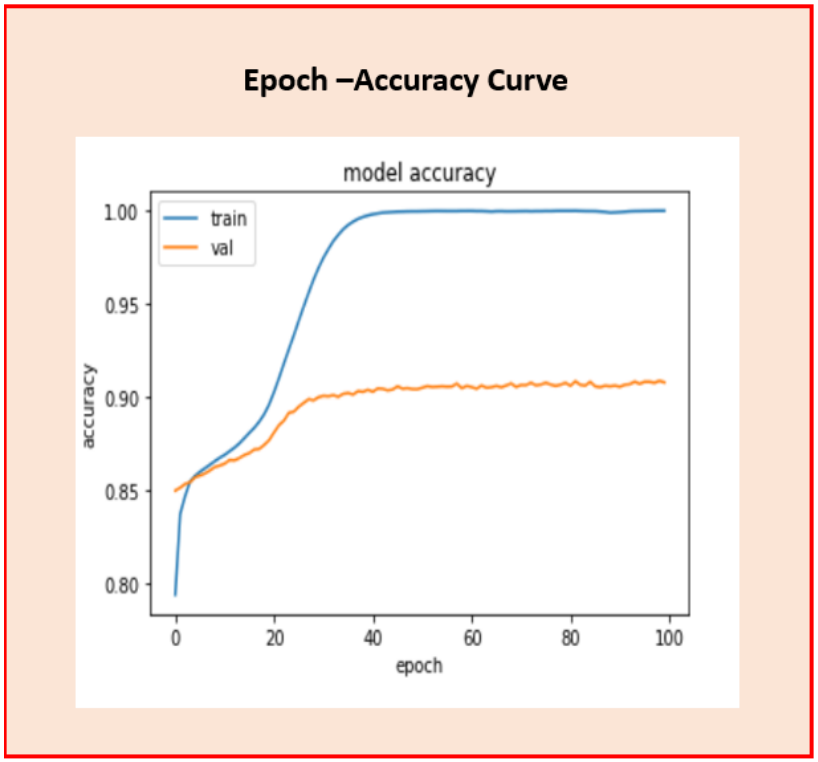


Figure 19: Accuracy and loss curves

Results

Multi modal Machine Translation- With Attention



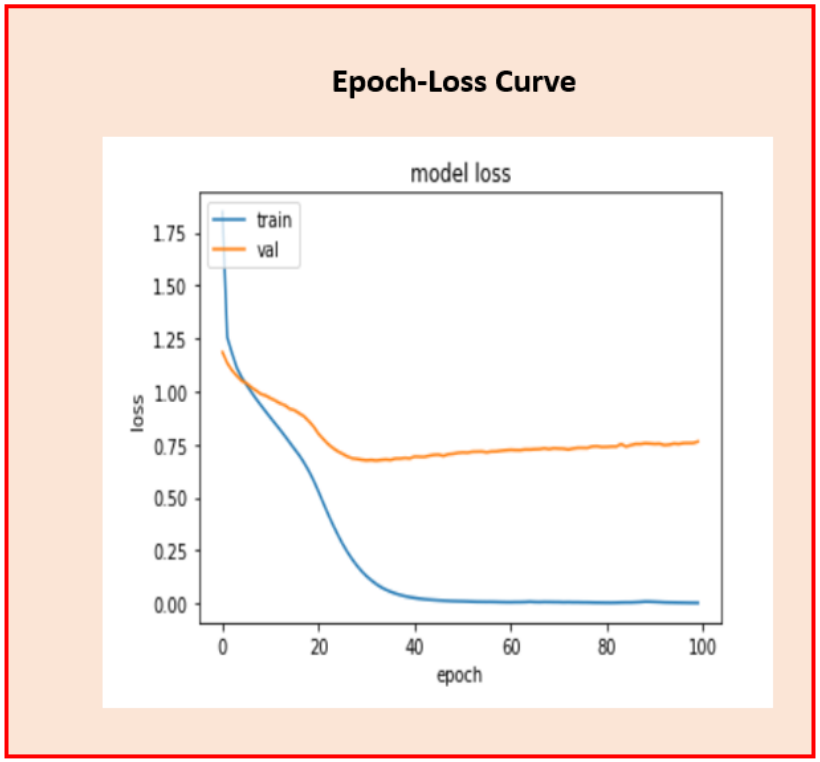
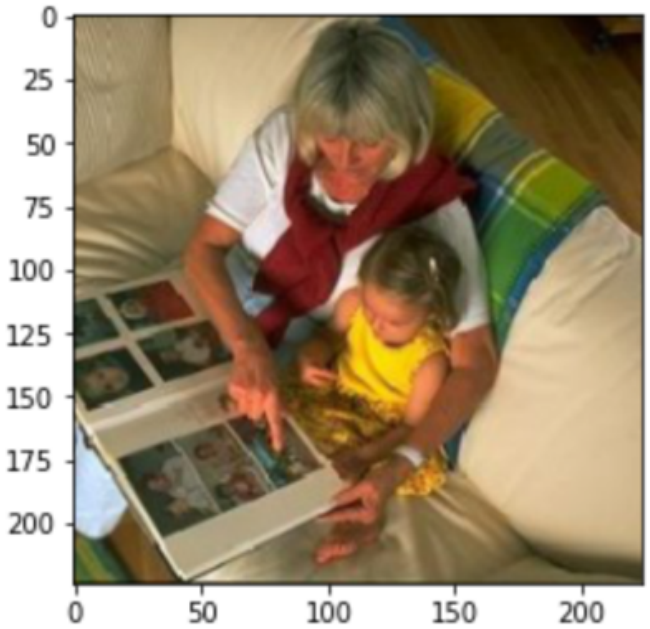


Figure 20: Accuracy and loss curves

Results

Multi modal machine Translation

Review: photo album open on an adults lap
Predicted summary: ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
<matplotlib.image.AxesImage at 0x7ff356884110>



Average BLEU Score
44

Figure 21: Shows results of Multi modal Machine Translation

Results

Table

BLEU Score

100

BEST CASE

Review: there is a group of girls beside the black car
Predicted summary: കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട്
<matplotlib.image.AxesImage at 0x7f1b1331c090>

BLEU Score

10

WORST CASE

Review: male surfer surfing in still in the ocean
Original summary: ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
Predicted summary: സമുദ്രത്തിലെ നല്ല തിരകൾ
<matplotlib.image.AxesImage at 0x7ff35686af90>

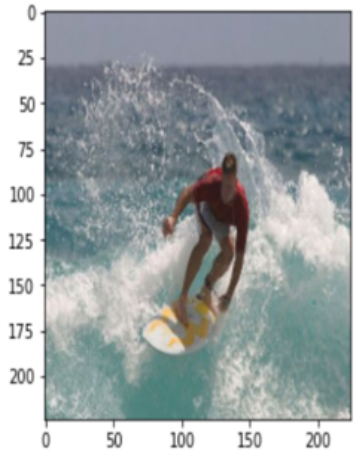


Figure 22: Best Case and worst scenerios genearted by our Model

AI-TRANSL:A MULTILINGUAL ASSISTIVE DEVICE USING MULTI MODAL MACHINE LEARNING

Lekshmy H O,Dr.Swaminathan

Amrita School of Engineering

Problem

Due to linguistic barriers blind people in multilingual developing often have trouble in accessing vison aids . Some products have machine translation modules that use statistical and neural machine technology to deal with this. However, in morphologically rich languages like Dravidian languages, these algorithms give poor translations. This is primarily due to differences in syntactic and morphological structures between languages. This literal translation frequently leads to misunderstandings and has a negative impact on assistive aid performance. Through our project we are presenting an effective English to Dravidian(Malayalam) machine translation methodology and vision aid based on it.

Background Information

Low resource machine translation poses the challenge of structural and morphological divergence. Initially, translation was performed with the help of rule based and traditional Corpus based translation techniques. However, this method leads to ignorance of key syntax distinctions called linguistics . Later neural machine translation systems were introduced. Standard NMT was insufficient to deal with translation ambiguity. As a result, multimodal machine translation is being developed.

Related Work

Po-Yao Huang [4] was the first to present a multimodal Neural Machine Translation based on Attention in 2016. Later,Iacer Calixto [3] discovers that in multimodal machine translation, the doubly-attentive Decoder is the most optimal technique. Studies revealed that multimodal machine translation with attention significantly improves quality of translation [2]. Most of these works were conducted on European dialects. In 2020 Lasker[5] was able to perform a English -Hindi multimodal machine translation with double attentive decoder. Even though effect of MMT on Dravidian languages remained understudied.

Problem Solution

IoT networks and multi modal machine learning techniques like image captioning and multi modal machine learning can be used to create an effective multilingual vision.

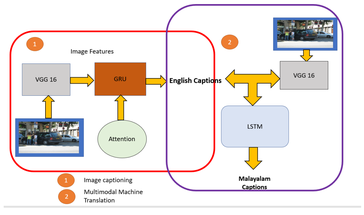


Fig. 1: Proposed Model AI-Transl Architecture.

Linguistic barriers can be solved by with multi modal machine translation techniques. Image guided translation is a subset of multimodal machine translation techniques that employs images as an additional modality to enhance translation quality.

English -Malayalam Multi modal Machine Translation

We have built our basic model on top of encoder-decoder sequence to sequence model.

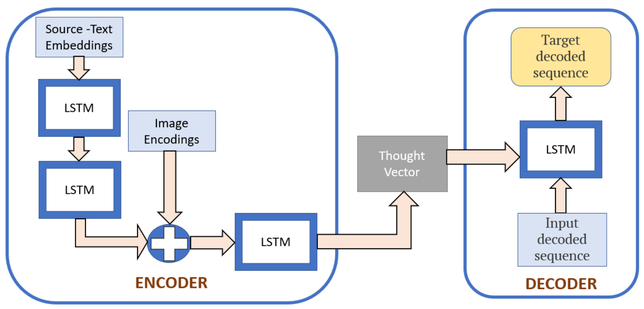


Fig. 3: Methodology adopted in English Malayalam Translation

Our Encoder incorporates three layers of LSTM cells and decoder one layer of LSTM cells. Encoder model is initially fed with source language text embedding . A pretrained CNN model is used for image feature extraction. These Image encoding are concatenated with encoder output from second layer and then it is passed through the third encoder layer. This layer is responsible for producing thought vector from image encodings and text encodings.Decoder is initialized using hidden states from encoder,

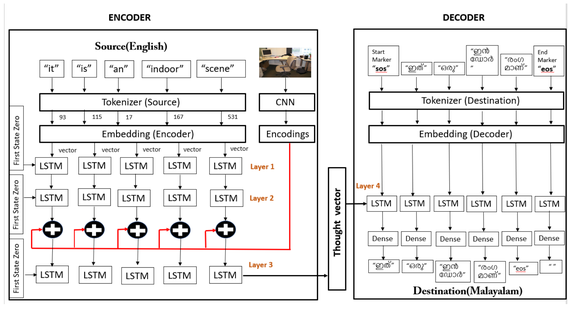


Fig. 4: Encoder-Decoder Architecture

More advanced version of this model is created by using bidirectional LSTM cells on encoder side and LSTM cells on decoder side with Bahdanau[1] attention mechanism.

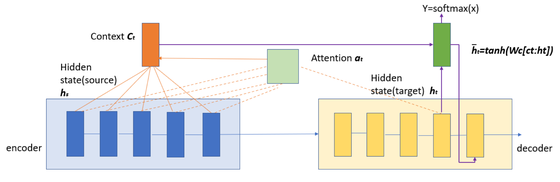


Fig. 5: Encoder-Decoder Architecture with attention

Here context vector $c_t = h_s * a_t, where h_s is is the weighted sum of the source states.$

Results

The main goal of our project was to develop a multilingual vision aid for Malayalam blinds using image guided multi modal machine learning technologies. An effective tion module was lacks in most of the assitive aids available in market. Our Model to take image as an extra modality to improve translation quality. We have used t to improve our model performance

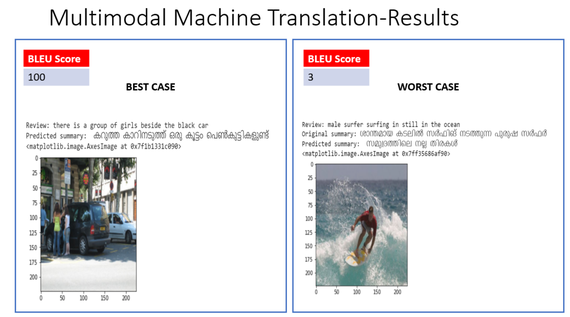


Fig. 6: Results

We were able to get an average blue score of 44.

Remarks

As per our knowledge this work is the state of art model in English- Malayalam M machine translation. We are trying to improve model performance using transfo chitechture.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Y. Bengio. “Neural Machine Translation by Jointly L Align and Translate”. In: *ArXiv* 1409 (Sept. 2014).

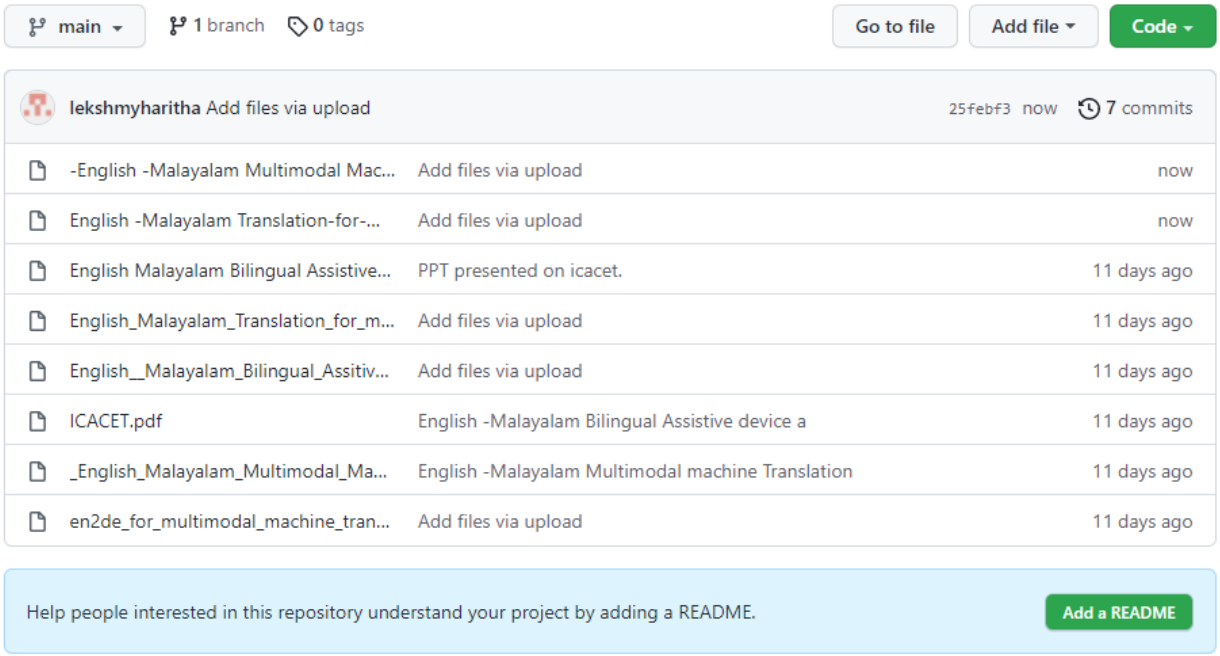
[2] Ozan Caglayan et al. “Probing the Need for Visual Context in Multimodal Machine Translatio *ceedings of the 2019 Conference of the North American Chapter of the Association for Cor Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. Minneapolis, I Association for Computational Linguistics, June 2019, pp. 4159–4170. DOI: 10.18653/v1/N19- https://aclanthology.org/N19-1422.

[3] Iacer Calixto, Qun Liu, and Nick Campbell. “Doubly-Attentive Decoder for Multi-modal Neura Translation”. In: *Proceedings of the 55th Annual Meeting of the Association for Computation tics (Volume 1: Long Papers)*. Vancouver, Canada: Association for Computational Linguistics, pp. 1913–1924. DOI: 10.18653/v1/P17-1175. URL: https://aclanthology.org/P17-1175.

[4] Po-Yao Huang et al. “Attention-based Multimodal Neural Machine Translation”. In: Jan. 2016 645. DOI: 10.18653/v1/W16-2360.

[5] Sahinur Laskar et al. *Improved English to Hindi Multi-modal Neural Machine Translation and H Captioning*. July 2021. URL: https://aclanthology.org/2021.wat-1.17.pdf.

Link: https://github.com/lekshmyharitha/AI-Transl.git



English -Malayalam Multimodal Machine Translation-5000 images

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.image as mp
```

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

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

```
with open('/content/drive/My Drive/Main/train.mn.txt') as file:
    mal_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/train.en.txt') as file:
    eng_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/train_images.txt') as file:
    train_images = file.read().split('\n')
```

```
train_images[-1]
```

```
''
```

```
#removing last elements which containing special characters
mal_txt.pop()
mal_txt.pop()
eng_txt.pop()
eng_txt.pop()
train_images.pop()
print(len(mal_txt))
print(len(eng_txt))
print(len(train_images))
img_path=[]
for s in train_images:
    img_path.append("/content/drive/My Drive/Main/trainimages/train/"+s)
```

```
28930
28931
28931
```

```
print(mal_txt[1])
print(eng_txt[1])
```

```
ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
it is an indoor scene
```

```
mal_txt[0:15]
```

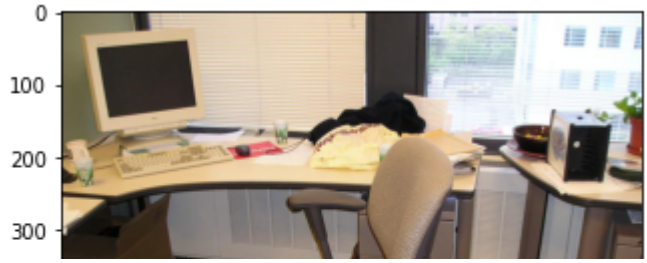
```
['ശാന്തമായ  കടലിൽ സർഫിങ്ങ് നടത്തുന്ന പുരുഷ സർഫർ',
 'ഇത് ഒരു ഇൻഡോർ രംഗമാണ്',
 'കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി',
 'മനുഷ്യന് ചെറിയ മുടിയുണ്ട്',
 'ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു',
 'കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട്',
 'ഒരു ഉന്തുവണ്ടിയിലെ കുട്ടി',
 'ഉയരമുള്ള മെറ്റൽ ലൈറ്റ്പാസ്റ്റ്',
 'മതിൽ വെളുത്ത ചായം പൂശി',
 'ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പുകൾ',
 'സമുദ്രം അഭിമുഖീകരിക്കുന്ന സ്ത്രീ',
 'ഇതൊരു ഓഫീസ് രേഖാചിത്രം',
 'നാല് ലോഹത്തിന്റെ കസേരകൾ',
 'കോലാഹലം ഒരു മേശപ്പുറത്താണ്',
 'ഒരു വെളുത്ത മൈക്രോവേവ് ഓവൻ']
```

```
eng_txt[0:15]
```

```
['Male surfer surfing in still in the ocean',
 'it is an indoor scene\t\t\t\t\t\t',
 'Computer screens turned on\t\t\t\t\t\t',
 'man has short hair\t\t\t\t\t\t',
 "photo album open on an adult's lap\t\t\t\t\t\t",
 'there is a group of girls beside the black car\t\t\t\t\t\t',
 'Child in a stroller\t\t\t\t\t\t',
 'Tall metal lightpost\t\t\t\t\t\t',
 'wall is painted white\t\t\t\t\t\t',
 'there are several pictures on the wall\t\t\t\t\t\t',
 'woman facing the ocean\t\t\t\t\t\t',
 'this is an office layout\t\t\t\t\t\t',
 'four metallic chairs\t\t\t\t\t\t',
 'Clutter is on a table\t\t\t\t\t\t',
 'a white microwave oven\t\t\t\t\t\t']
```

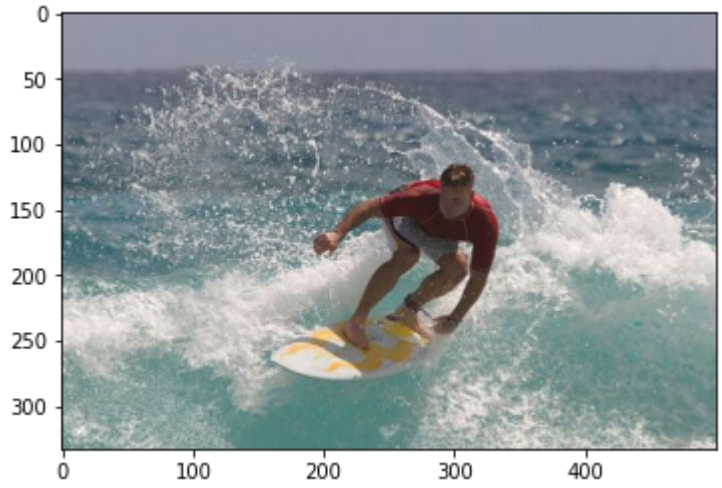
```
im=mp.imread(img_path[1])
plt.imshow(im)
print(img_path[1])
print("mal:"+mal_txt[1])
print("eng:"+eng_txt[1])
```

```
/content/drive/My Drive/Main/trainimages/train/11.jpg
mal:ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
eng:it is an indoor scene
```



```
im=mp.imread(img_path[0])
plt.imshow(im)
print(img_path[0])
print("mal:"+mal_txt[0])
print("eng:"+eng_txt[0])
```

```
/content/drive/My Drive/Main/trainimages/train/10.jpg
mal:ശാന്തമായ  കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
eng:Male surfer surfing in still in the ocean
```



```
#Manually splitting data for training and texting-due to presence of 3 inputs split using Skl
splits=10000
mal_train=mal_txt[:splits]
eng_train=eng_txt[:splits]


mal_df = pd.DataFrame(mal_train, columns=['Malayalam'])
eng_df = pd.DataFrame(eng_train, columns=['English'])



mal_df.head(10)
```

	Malayalam
0	ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
1	ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
2	കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി
3	മനുഷ്യന് ചെറിയ മുടിയുണ്ട്
4	ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
5	കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട്
6	ഒരു ഉന്തുവണ്ടിയിലെ കുട്ടി
7	ഉയരമുള്ള മെറ്റൽ ലൈറ്റ്പാസ്റ്റ്
8	മതിൽ വെളുത്ത ചായം പൂശി

```
eng_df.head(10)
```

	English
0	Male surfer surfing in still in the ocean
1	it is an indoor scene\t\t\t\t\t\t
2	Computer screens turned on\t\t\t\t\t\t
3	man has short hair\t\t\t\t\t\t
4	photo album open on an adult's lap\t\t\t\t\t\t
5	there is a group of girls beside the black car...
6	Child in a stroller\t\t\t\t\t\t
7	Tall metal lightpost\t\t\t\t\t\t
8	wall is painted white\t\t\t\t\t\t
9	there are several pictures on the wall\t\t\t\t...

```
#Datacleaning by removing special characters
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r" '", "", text)
    text = re.sub(r" '", "", text)
    text = re.sub(r"-", " ", text)
    text = re.sub(r"<5>", "5", text)
    text = re.sub(r"‘ ", "", text)
    text = re.sub(r" ’", "", text)
    text = re.sub(r"[+\.\!\/_,$%^*(+\"\']+|[+——！ ，  〈〉 《》 。 ｜？ ？、 . %~@#￥%……&* () ’]", "", t
    text=text.rstrip()
```

```
    return text
```

```
mal_text1 = mal_df["Malayalam"].apply(clean_text)
eng_text1 = eng_df["English"].apply(clean_text)
mal_text2 = list(mal_text1.values)
eng_text2 = list(eng_text1.values)
```

```
#cleaned Malayalm data
mal_text1
```

```
0        ശാന്തമായ  കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
1                            ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
2                    കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി
3                        മനുഷ്യന് ചെറിയ മുടിയുണ്ട്
4       ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
                           ...
4995          ഫ്രിസ്ബീ ഉള്ള പുൽത്തകിടിയിൽ ഒരു നായ
4996       ഒരു ഉദ്യാനത്തിൽ ചാരനിറത്തിലുള്ള ലോഹ വേലി
4997            തവിട്ടുനിറമുള്ള മുടിയുള്ള മനുഷ്യൻ
4998                 ഒരു ജിറാഫ് പുല്ല് തിന്നുന്നു
4999                  മുൻവശത്തുള്ള ഒരു പെൺകുട്ടി
Name: Malayalam, Length: 5000, dtype: object
```

```
#cleaned English data
eng_text1
```

```
0       male surfer surfing in still in the ocean
1                           it is an indoor scene
2                     computer screens turned on
3                             man has short hair
4               photo album open on an adults lap
                           ...
4995               a dog on a lawn with a frisbee
4996                 a gray metal fence in a park
4997                           a brown haired man
4998                           the floor is tiled
4999               a young girl in the foreground
Name: English, Length: 5000, dtype: object
```

```
mal_text2[1:5]
```

```
['ഇത് ഒരു ഇൻഡോർ രംഗമാണ്',
 'കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി',
 'മനുഷ്യന് ചെറിയ മുടിയുണ്ട്',
 'ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു']
```

```
eng_text2[1:5]
```

```
['it is an indoor scene',
```

```
       'computer screens turned on',
       'man has short hair',
       'photo album open on an adults lap']


#Adding starting and ending tokens
mal_temp=[]
for s in mal_text2:
    temp="sos "+s+" eos"
    mal_temp.append(temp)
#text2=[]
mal_text2=mal_temp
mal_text2[1:10]
```

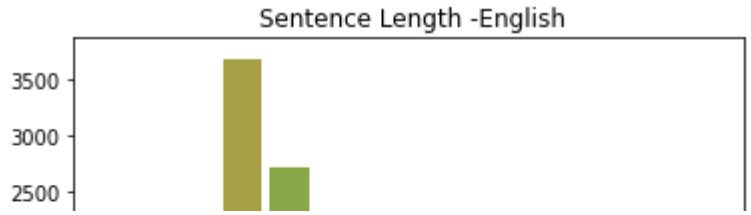
```
['sos ഇത് ഒരു ഇൻഡോർ രംഗമാണ് eos',
 'sos കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി eos',
 'sos മനുഷ്യന് ചെറിയ മുടിയുണ്ട് eos',
 'sos ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു eos',
 'sos കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട് eos',
 'sos ഒരു ഉന്തുവണ്ടിയിലെ കുട്ടി eos',
 'sos ഉയരമുള്ള മെറ്റൽ ലൈറ്റ്പോസ്റ്റ് eos',
 'sos മതിൽ വെളുത്ത ചായം പൂശി eos',
 'sos ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പുകൾ eos']
```

```
import seaborn as sn
import matplotlib.pyplot as plt
malayalam_words = []
for i in mal_text2:
    malayalam_words.append(len(i.split()))
sn.countplot(malayalam_words).set(title=' Sentence Length -Malayalam')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```

```
english_words = []
for j in eng_text2:
    english_words.append(len(j.split()))
sn.countplot(english_words).set(title=' Sentence Length -English')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
maxlen_malayalam = max(malayalam_words)
maxlen_english = max(english_words)
print('Maximum sentence length-Malayalam :',maxlen_malayalam)
print('Maximum sentence length-English :',maxlen_english)
```

```
Maximum sentence length-Malayalam : 13
Maximum sentence length-English : 14
```

```
#splitting training data into training and validation data
x_tr=eng_text2[:splits-500]
y_tr=mal_text2[:splits-500]
x_val=eng_text2[splits-500:]
y_val=mal_text2[splits-500:]
```

```
x_tr[1:5]
```

```
['it is an indoor scene',
 'computer screens turned on',
 'man has short hair',
 'photo album open on an adults lap']
```

```
y_tr[1:5]
```

```
['sos ഇത് ഒരു ഇൻഡോർ രംഗമാണ് eos',
 'sos കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി eos',
 'sos മനുഷ്യന് ചെറിയ മുടിയുണ്ട് eos',
 'sos ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു eos']
```

```
len(x_tr)
```

```
9500
```

```
len(x_val)
```

```
500
```

```
#Tokening the sentences using Keras tokenizer -Malayalam data
from keras.preprocessing.text import Tokenizer
x_tokens = Tokenizer()
```

```
x_tokens.fit_on_texts(x_tr)
x_tr = x_tokens.texts_to_sequences(x_tr)
x_val = x_tokens.texts_to_sequences(x_val)
print('x_tr:',x_tr)
print('x_val:',x_val)
```

```
x_tr: [[463, 238, 230, 6, 817, 6, 2, 239], [149, 5, 32, 1173, 209], [118, 1604, 464, 3],
x_val: [[2, 18, 6, 2, 262], [188, 6, 30], [26, 64, 3, 55], [1, 326, 4, 1, 33], [341, 3,
```

```
#padding with post (appending zeros at the end to equalize sentence length)
from keras.preprocessing.sequence import pad_sequences
x_tr = pad_sequences(x_tr,maxlen = maxlen_english,padding = 'post')
x_val = pad_sequences(x_val,maxlen = maxlen_english,padding = 'post')

# +1 for padding
x_voc_size   =  len(x_tokens.word_index) +1
print("No of unique words in English",x_voc_size)
```

```
No of unique words in English 3031
```

```
# English data preprocessing
from keras.preprocessing.text import Tokenizer
y_tokens = Tokenizer()
y_tokens.fit_on_texts(y_tr)

y_tr = y_tokens.texts_to_sequences(y_tr)
y_val = y_tokens.texts_to_sequences(y_val)

from keras.preprocessing.sequence import pad_sequences
y_tr = pad_sequences(y_tr,maxlen = maxlen_malayalam,padding = 'post')
y_val = pad_sequences(y_val,maxlen = maxlen_malayalam,padding = 'post')

# +1 for padding
y_voc_size   =  len(y_tokens.word_index) +1
print("No of unique words in Malyalam",y_voc_size)
```

```
No of unique words in Malyalam 5674
```

```
pip install keras-applications
```

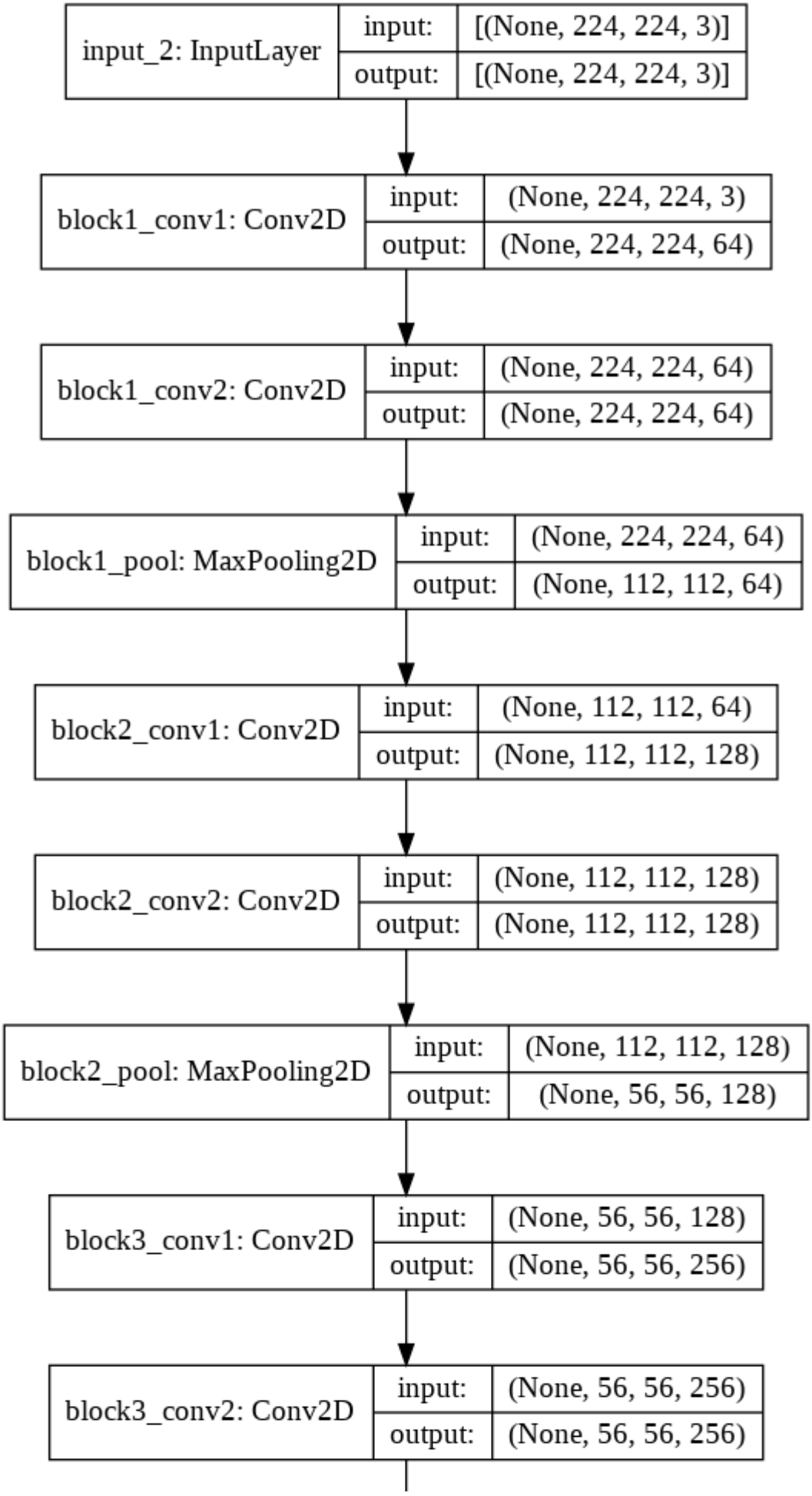
```
Collecting keras-applications
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     |████████████████████████████████| 50 kB 4.0 MB/s
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from kera
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages
Installing collected packages: keras-applications
Successfully installed keras-applications-1.0.8
```

```
import pandas as pd
import pickle
import numpy as np
import os
import keras
import tensorflow
from keras_applications.resnet import ResNet50
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, GlobalAveragePooling2D,BatchNormalization,Flatten,Input, Conv
from keras.models import Sequential, Model
from keras.utils import np_utils
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
import keras
from keras import backend as K
import gensim
from numpy import *
import numpy as np
import pandas as pd
import re
from tensorflow.keras.applications.vgg16 import VGG16
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistribut
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings



#Loading VGG model for Feature Extraction-Removing classification layers from memory
modelvgg = VGG16(include_top=True,weights="imagenet")
modelvgg.layers.pop()
modelvgg = Model(inputs=modelvgg.inputs, outputs=modelvgg.layers[-2].output)
modelvgg.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16
553467904/553467096 [==============================] - 3s 0us/step
553476096/553467096 [==============================] - 3s 0us/step
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
```

```
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```

```
from keras.utils.vis_utils import plot_model
import tensorflow as tf


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


input_2: InputLayer
input: [(None, 224, 224, 3)]
output: [(None, 224, 224, 3)]
block1_conv1: Conv2D
input: (None, 224, 224, 3)
output: (None, 224, 224, 64)
block1_conv2: Conv2D
input: (None, 224, 224, 64)
output: (None, 224, 224, 64)
block1_pool: MaxPooling2D
input: (None, 224, 224, 64)
output: (None, 112, 112, 64)
block2_conv1: Conv2D
input: (None, 112, 112, 64)
output: (None, 112, 112, 128)
block2_conv2: Conv2D
input: (None, 112, 112, 128)
output: (None, 112, 112, 128)
block2_pool: MaxPooling2D
input: (None, 112, 112, 128)
output: (None, 56, 56, 128)
block3_conv1: Conv2D
input: (None, 56, 56, 128)
output: (None, 56, 56, 256)
block3_conv2: Conv2D
input: (None, 56, 56, 256)
output: (None, 56, 56, 256)

```
pip install cv
```

```
Collecting cv
  Downloading cv-1.0.0-py3-none-any.whl (7.3 kB)
Installing collected packages: cv
Successfully installed cv-1.0.0
```

```
import cv2
import cv
```

```
ERROR:root:Error disabling cv.imshow().
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/google/colab/_import_hooks/_cv2.py", line
    cv_module.imshow,
AttributeError: module 'cv' has no attribute 'imshow'
```

```
#tRY RESNET


#Resizing image and converting grey scale images into RGB images
#split=5000
imagedata=np.zeros(shape=(splits,224,224,3))
for i in range(splits):
    temp=mp.imread(img_path[i])
    if (len(temp.shape)==3):
       temp=cv2.resize(temp,(224,224))
       imagedata[i]=temp
    elif (len(temp.shape)<3):
       #plt.imshow(temp)
       temp=cv2.cvtColor(temp, cv2.COLOR_BGR2RGB)
       temp=cv2.resize(temp,(224,224))
       imagedata[i]=temp
imagedata=imagedata/255
imagedata=imagedata.astype(np.float16)


imagedata[1:10]
```

```
array([[[[0.4626  , 0.443   , 0.3254  ],
         [0.4548  , 0.4353  , 0.3176  ],
         [0.4666  , 0.447   , 0.3293  ],
         ...,
         [0.902   , 0.933   , 0.9453  ],
         [0.906   , 0.937   , 0.949   ],
         [0.906   , 0.937   , 0.949   ]],

        [[0.4666  , 0.447   , 0.3293  ],
         [0.4587  , 0.4392  , 0.3215  ],
         [0.4707  , 0.451   , 0.3333  ],
```

```
                ...,
                [0.949   , 0.9805  , 0.9883  ],
                [0.9453  , 0.9766  , 0.9883  ],
                [0.9453  , 0.9766  , 0.9883  ]],

               [[0.4707  , 0.451   , 0.3333  ],
                [0.4626  , 0.443   , 0.3254  ],
                [0.4785  , 0.4587  , 0.341   ],
                ...,
                [0.961   , 0.992   , 1.      ],
                [0.9453  , 0.9766  , 0.992   ],
                [0.9453  , 0.9766  , 0.992   ]],

               ...,

               [[0.3098  , 0.2864  , 0.2393  ],
                [0.341   , 0.3098  , 0.2666  ],
                [0.3647  , 0.3254  , 0.2864  ],
                ...,
                [0.1686  , 0.153   , 0.1059  ],
                [0.1647  , 0.149   , 0.102   ],
                [0.1569  , 0.1412  , 0.0941  ]],

               [[0.3098  , 0.2864  , 0.2393  ],
                [0.3215  , 0.2903  , 0.2471  ],
                [0.3608  , 0.3176  , 0.2783  ],
                ...,
                [0.1608  , 0.1451  , 0.098   ],
                [0.1647  , 0.149   , 0.102   ],
                [0.153   , 0.1372  , 0.0902  ]],

               [[0.3176  , 0.298   , 0.2471  ],
                [0.3215  , 0.2903  , 0.2471  ],
                [0.349   , 0.3098  , 0.2705  ],
                ...,
                [0.153   , 0.1372  , 0.0902  ],
                [0.1686  , 0.153   , 0.1059  ],
                [0.149   , 0.1333  , 0.0863  ]]],


              [[[0.51    , 0.502   , 0.506   ],
                [0.506   , 0.506   , 0.51    ],
                [0.51    , 0.506   , 0.5215  ],
                ...,
                [0.392   , 0.408   , 0.447   ],
                [0.3843  , 0.408   , 0.4548  ],
                [0.3765  , 0.408   , 0.4587  ]],
```

```
with open('/content/drive/My Drive/Main/imagedatas.txt', 'w') as writefile:
    writefile.write("imagedata")
```

```
#preprocessing images
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input
from collections import OrderedDict
```

```
jpgs=img_path[:splits]
images_new = OrderedDict()
npix = 224
target_size = (npix,npix,3)
for i,name in enumerate(jpgs):
    filename = name
    image = load_img(filename, target_size=target_size)
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    nimage = preprocess_input(image)
    y_pred = modelvgg.predict(nimage.reshape( (1,) + nimage.shape[:3]))
    images_new [name] = y_pred.flatten()
    if i%200==0:
        print(i,filename)
```

```
0 /content/drive/My Drive/Main/trainimages/train/10.jpg
200 /content/drive/My Drive/Main/trainimages/train/739.jpg
400 /content/drive/My Drive/Main/trainimages/train/1529.jpg
600 /content/drive/My Drive/Main/trainimages/train/2238.jpg
800 /content/drive/My Drive/Main/trainimages/train/2970.jpg
1000 /content/drive/My Drive/Main/trainimages/train/3693.jpg
1200 /content/drive/My Drive/Main/trainimages/train/4450.jpg
1400 /content/drive/My Drive/Main/trainimages/train/150275.jpg
1600 /content/drive/My Drive/Main/trainimages/train/497937.jpg
1800 /content/drive/My Drive/Main/trainimages/train/713300.jpg
2000 /content/drive/My Drive/Main/trainimages/train/1159284.jpg
2200 /content/drive/My Drive/Main/trainimages/train/1160083.jpg
2400 /content/drive/My Drive/Main/trainimages/train/1592294.jpg
2600 /content/drive/My Drive/Main/trainimages/train/1592957.jpg
2800 /content/drive/My Drive/Main/trainimages/train/2315779.jpg
3000 /content/drive/My Drive/Main/trainimages/train/2316528.jpg
3200 /content/drive/My Drive/Main/trainimages/train/2317337.jpg
3400 /content/drive/My Drive/Main/trainimages/train/2318048.jpg
3600 /content/drive/My Drive/Main/trainimages/train/2318796.jpg
3800 /content/drive/My Drive/Main/trainimages/train/2319593.jpg
4000 /content/drive/My Drive/Main/trainimages/train/2320436.jpg
4200 /content/drive/My Drive/Main/trainimages/train/2321247.jpg
4400 /content/drive/My Drive/Main/trainimages/train/2322001.jpg
4600 /content/drive/My Drive/Main/trainimages/train/2322763.jpg
4800 /content/drive/My Drive/Main/trainimages/train/2323538.jpg
5000 /content/drive/My Drive/Main/trainimages/train/2324400.jpg
5200 /content/drive/My Drive/Main/trainimages/train/2325137.jpg
5400 /content/drive/My Drive/Main/trainimages/train/2325912.jpg
5600 /content/drive/My Drive/Main/trainimages/train/2326777.jpg
5800 /content/drive/My Drive/Main/trainimages/train/2327564.jpg
6000 /content/drive/My Drive/Main/trainimages/train/2328291.jpg
6200 /content/drive/My Drive/Main/trainimages/train/2329112.jpg
6400 /content/drive/My Drive/Main/trainimages/train/2329883.jpg
6600 /content/drive/My Drive/Main/trainimages/train/2330601.jpg
6800 /content/drive/My Drive/Main/trainimages/train/2331402.jpg
7000 /content/drive/My Drive/Main/trainimages/train/2332176.jpg
7200 /content/drive/My Drive/Main/trainimages/train/2333032.jpg
7400 /content/drive/My Drive/Main/trainimages/train/2333847.jpg
7600 /content/drive/My Drive/Main/trainimages/train/2334602.jpg
7800 /content/drive/My Drive/Main/trainimages/train/2335353.jpg
```

```
8000 /content/drive/My Drive/Main/trainimages/train/2336144.jpg
8200 /content/drive/My Drive/Main/trainimages/train/2336923.jpg
8400 /content/drive/My Drive/Main/trainimages/train/2337672.jpg
8600 /content/drive/My Drive/Main/trainimages/train/2338520.jpg
8800 /content/drive/My Drive/Main/trainimages/train/2339266.jpg
9000 /content/drive/My Drive/Main/trainimages/train/2339991.jpg
9200 /content/drive/My Drive/Main/trainimages/train/2340756.jpg
9400 /content/drive/My Drive/Main/trainimages/train/2341498.jpg
9600 /content/drive/My Drive/Main/trainimages/train/2342276.jpg
9800 /content/drive/My Drive/Main/trainimages/train/2343119.jpg
```

```
print(list(images_new.values())[1])
```

```
[0.6324536 1.3856603 0.        ... 0.        0.        2.182263 ]
```

```
#storing image pixels sepearetely
vgg_feature=np.zeros(shape=(len(jpgs),4096))
for i in range(len(jpgs)):
    vgg_feature[i]=images_new[jpgs[i]]
vgg_feature[1:10]
```

```
array([[0.63245362, 1.38566029, 0.        , ..., 0.        , 0.        ,
        2.1822629 ],
       [0.54209262, 0.        , 0.        , ..., 0.        , 0.        ,
        0.93312246],
       [1.5215044 , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.11626244, 1.3521657 , 0.05647588, ..., 0.        , 2.70661497,
        0.        ],
       [2.4945612 , 1.68215179, 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.26692578, 4.36360025, 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

```
#splitting image pixels for training and validation
vgg_train_=vgg_feature[:splits-500]
vgg_val=vgg_feature[splits-500:]
```

```
#Generating a repeat vector from image pixels
img_inputs=Input(shape=(4096,))
d_1=Dense(512, activation='relu')(img_inputs)
r_1=RepeatVector(maxlen_english)(d_1)
vf_model = Model(img_inputs, r_1)
vf_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         [(None, 4096)]            0
_________________________________________________________________
```

```
dense (Dense)                (None, 512)               2097664
_________________________________________________________________
repeat_vector (RepeatVector) (None, 14, 512)           0
=================================================================
Total params: 2,097,664
Trainable params: 2,097,664
Non-trainable params: 0
_________________________________________________________________
```

```
x_voc=x_voc_size
y_voc=y_voc_size
```

```
#Model
x_voc=x_voc_size
y_voc=y_voc_size
latent_dim = 512
embedding_dim=512
#Encoder
encoder_inputs = Input(shape=(maxlen_english,))
#The model will take as input an integer matrix of size (batch,input_length)and the largest i
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)
print(encoder_inputs.get_shape)
print(enc_emb.get_shape)
```

```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, 14) dtype=float32 (cre
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, 14, 512) dtype=float32
```

```
#encoder LSTM Layer 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent
#The dimension of each state equals to the LSTM unit number
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)
print(encoder_lstm1.output_shape)
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the crite
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the crite
[(None, 14, 512), (None, 512), (None, 512)]
```

```
#LSTM layer 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)
print(encoder_lstm2.output_shape)
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the cri
[(None, 14, 512), (None, 512), (None, 512)]
```

```
#Concatenating image features with text input
encoder_output2=Concatenate(axis=-1)([encoder_output2,r_1])


#LSTM layer 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)
print(encoder_lstm3.output_shape)
```

```
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't meet the cri
[(None, 14, 512), (None, 512), (None, 512)]
```

```
#Decoder
# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))
#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)
print(decoder_inputs.get_shape)
print(dec_emb.get_shape)
```

```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None) dtype=float32 (
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None, 512) dtype=float
```

```
#Decoder LSTM layer1
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurren
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[s
print(decoder_lstm.output_shape)
```

```
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernels since it doesn't meet the cri
[(None, None, 512), (None, 512), (None, 512)]
```

```
#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_outputs)
print(decoder_dense.output_shape)
```

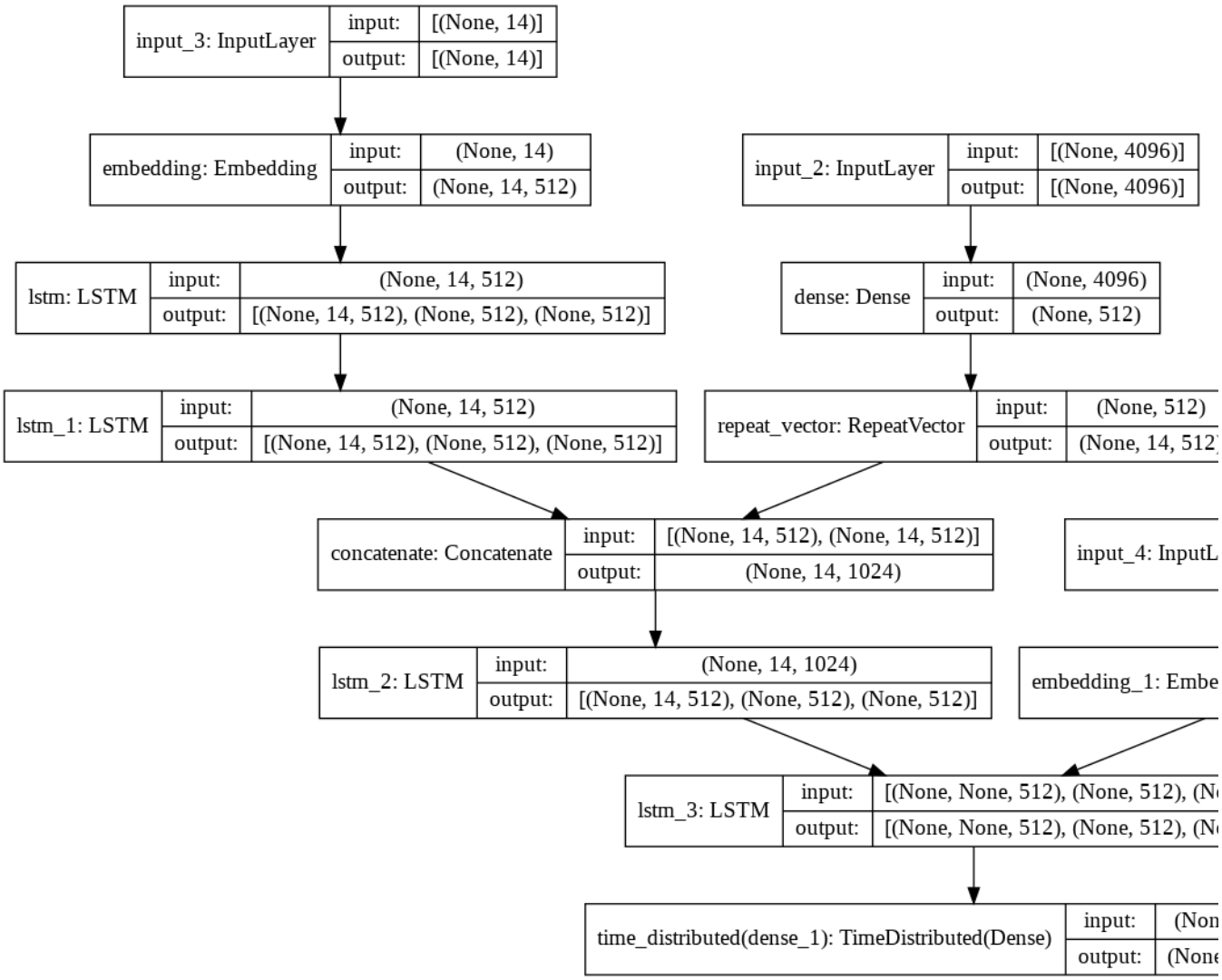
```
(None, None, 5674)
```

```
model = Model([encoder_inputs,decoder_inputs,img_inputs], decoder_outputs)
model.summary()
```

```
Model: "model_2"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
```

```
==================================================================================================
input_3 (InputLayer)            [(None, 14)]         0
__________________________________________________________________________________________________
embedding (Embedding)           (None, 14, 512)      1551872     input_3[0][0]
__________________________________________________________________________________________________
input_2 (InputLayer)            [(None, 4096)]       0
__________________________________________________________________________________________________
lstm (LSTM)                     [(None, 14, 512), (N 2099200     embedding[0][0]
__________________________________________________________________________________________________
dense (Dense)                   (None, 512)          2097664     input_2[0][0]
__________________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 14, 512), (N 2099200     lstm[0][0]
__________________________________________________________________________________________________
repeat_vector (RepeatVector)    (None, 14, 512)      0           dense[0][0]
__________________________________________________________________________________________________
input_4 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________________
concatenate (Concatenate)       (None, 14, 1024)     0           lstm_1[0][0]
                                                                 repeat_vector[0][0]
__________________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 512)    2905088     input_4[0][0]
__________________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 14, 512), (N 3147776     concatenate[0][0]
__________________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 512),  2099200     embedding_1[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 5674)   2910762     lstm_3[0][0]
==================================================================================================
Total params: 18,910,762
Trainable params: 18,910,762
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
from keras.utils.vis_utils import plot_model
import tensorflow as tf


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



in

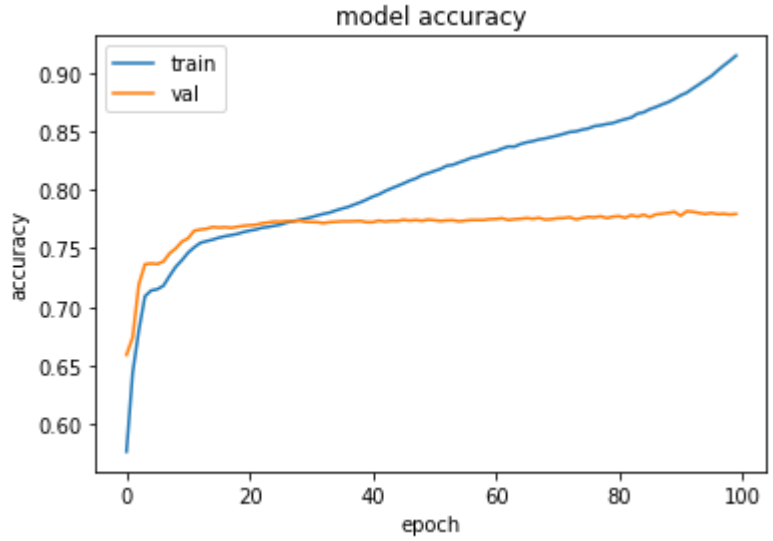
```
#compiling model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history=model.fit([x_tr,y_tr[:,:-1],vgg_train_], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)
```

```
Epoch 1/100
19/19 [==============================] - 15s 324ms/step - loss: 4.0751 - accuracy: 0.
Epoch 2/100
19/19 [==============================] - 5s 271ms/step - loss: 2.4583 - accuracy: 0.6
Epoch 3/100
19/19 [==============================] - 5s 263ms/step - loss: 2.2810 - accuracy: 0.6
```

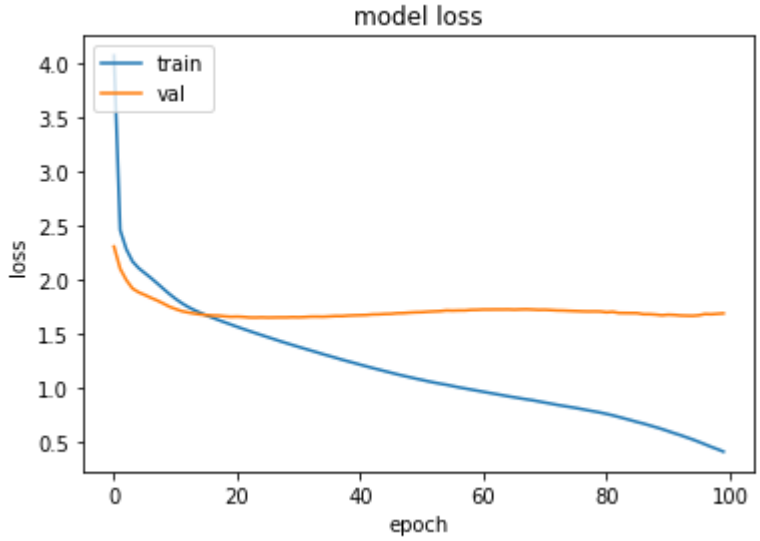
```
Epoch 4/100
19/19 [==============================] - 5s 274ms/step - loss: 2.1676 - accuracy: 0.7
Epoch 5/100
19/19 [==============================] - 5s 270ms/step - loss: 2.1056 - accuracy: 0.7
Epoch 6/100
19/19 [==============================] - 5s 268ms/step - loss: 2.0594 - accuracy: 0.7
Epoch 7/100
19/19 [==============================] - 5s 263ms/step - loss: 2.0141 - accuracy: 0.7
Epoch 8/100
19/19 [==============================] - 5s 273ms/step - loss: 1.9649 - accuracy: 0.7
Epoch 9/100
19/19 [==============================] - 5s 276ms/step - loss: 1.9147 - accuracy: 0.7
Epoch 10/100
19/19 [==============================] - 5s 276ms/step - loss: 1.8653 - accuracy: 0.7
Epoch 11/100
19/19 [==============================] - 5s 272ms/step - loss: 1.8208 - accuracy: 0.7
Epoch 12/100
19/19 [==============================] - 5s 273ms/step - loss: 1.7808 - accuracy: 0.7
Epoch 13/100
19/19 [==============================] - 5s 271ms/step - loss: 1.7465 - accuracy: 0.7
Epoch 14/100
19/19 [==============================] - 5s 274ms/step - loss: 1.7173 - accuracy: 0.7
Epoch 15/100
19/19 [==============================] - 5s 271ms/step - loss: 1.6913 - accuracy: 0.7
Epoch 16/100
19/19 [==============================] - 5s 274ms/step - loss: 1.6674 - accuracy: 0.7
Epoch 17/100
19/19 [==============================] - 5s 279ms/step - loss: 1.6450 - accuracy: 0.7
Epoch 18/100
19/19 [==============================] - 5s 279ms/step - loss: 1.6233 - accuracy: 0.7
Epoch 19/100
19/19 [==============================] - 5s 279ms/step - loss: 1.6023 - accuracy: 0.7
Epoch 20/100
19/19 [==============================] - 5s 275ms/step - loss: 1.5819 - accuracy: 0.7
Epoch 21/100
19/19 [==============================] - 5s 270ms/step - loss: 1.5611 - accuracy: 0.7
Epoch 22/100
19/19 [==============================] - 5s 281ms/step - loss: 1.5411 - accuracy: 0.7
Epoch 23/100
19/19 [==============================] - 5s 267ms/step - loss: 1.5216 - accuracy: 0.7
Epoch 24/100
19/19 [==============================] - 5s 282ms/step - loss: 1.5021 - accuracy: 0.7
Epoch 25/100
19/19 [==============================] - 5s 270ms/step - loss: 1.4835 - accuracy: 0.7
Epoch 26/100
19/19 [==============================] - 5s 270ms/step - loss: 1.4647 - accuracy: 0.7
Epoch 27/100
19/19 [==============================] - 5s 277ms/step - loss: 1.4463 - accuracy: 0.7
Epoch 28/100
19/19 [==============================] - 5s 270ms/step - loss: 1.4275 - accuracy: 0.7
Epoch 29/100
19/19 [==============================] - 5s 275ms/step - loss: 1.4097 - accuracy: 0.7
```

```
import keras
from matplotlib import pyplot as plt
```

```
#history = model1.fit(train_x, train_y,validation_split = 0.1, epochs=50, batch_size=4)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
reverse_target_word_index=y_tokens.index_word
```

```
reverse_target_word_index
```

```
{1: 'sos',
 2: 'eos',
 3: 'ഒരു',
 4: 'കറുത്ത',
 5: 'വെളുത്ത',
 6: 'സ്ത്രീ',
 7: 'ധരിച്ച',
 8: 'മനുഷ്യൻ',
 9: 'പച്ച',
 10: 'വലിയ',
 11: 'നീല',
 12: 'ബാഗ്',
 13: 'വർണ്ണാഭമായ',
 14: 'ചുവന്ന',
 15: 'ചാരനിറത്തിലുള്ള',
 16: 'സ്കാർഫ്',
 17: 'ട്രാഷ്',
 18: 'റോഡിന്റെ',
 19: 'വിൻഡോ',
 20: 'പുല്ലിന്റെ',
 21: 'മഞ്ഞ',
 22: 'വശങ്ങളിൽ',
 23: 'സ്ട്രിപ്പുകൾ',
 24: 'വ്യക്തി',
 25: 'ഇതൊരു',
 26: 'രണ്ട്',
 27: 'ചെറിയ',
 28: 'കെട്ടിടത്തിലെ',
 29: 'നിലത്ത്',
 30: 'ആന',
 31: 'വെള്ള',
 32: 'ഒരാൾ',
 33: 'ഉള്ള',
 34: 'ചുവപ്പ്',
 35: 'തവിട്ട്',
 36: 'കുഞ്ഞ്',
 37: 'ജിറാഫ്',
 38: 'വെള്ളയും',
 39: 'ഓറഞ്ച്',
 40: 'മുകളിൽ',
 41: 'പുല്ല്',
 42: 'എഞ്ചിൻ',
 43: 'ചിഹ്നം',
 44: 'കാർ',
 45: 'കെട്ടിടത്തിന്റെ',
 46: 'മേശപ്പുറത്ത്',
 47: 'പുള്ളി',
 48: 'ആളുകൾ',
 49: 'നിറമുള്ള',
 50: 'ടെന്നീസ്',
 51: 'രംഗം',
 52: 'ട്രെയിൻ',
 53: 'തലം',
 54: 'ഇത്',
 55: 'ബീച്ച്',
```

```
 56: 'നിൽക്കുന്നു',
 57: 'തിന്നുന്നു',
 58: 'നിൽക്കുന്ന',
 59: 'ദ്വീപ്',
```

```
reverse_source_word_index=x_tokens.index_word
reverse_source_word_index
```

```
{1: 'a',
 2: 'the',
 3: 'on',
 4: 'of',
 5: 'is',
 6: 'in',
 7: 'white',
 8: 'man',
 9: 'and',
 10: 'black',
 11: 'with',
 12: 'red',
 13: 'this',
 14: 'person',
 15: 'blue',
 16: 'building',
 17: 'wall',
 18: 'woman',
 19: 'brown',
 20: 'wearing',
 21: 'green',
 22: 'window',
 23: 'yellow',
 24: 'head',
 25: 'sign',
 26: 'two',
 27: 'train',
 28: 'street',
 29: 'water',
 30: 'sky',
 31: 'side',
 32: 'an',
 33: 'table',
 34: 'car',
 35: 'standing',
 36: 'light',
 37: 'large',
 38: 'clock',
 39: 'people',
 40: 'shirt',
 41: 'sitting',
 42: 'holding',
 43: 'are',
 44: 'small',
 45: 'plate',
 46: 'has',
 47: 'bus',
 48: 'road',
```

```
 49: 'to',
 50: 'dog',
 51: 'grass',
 52: 'orange',
 53: 'tennis',
 54: 'top',
 55: 'ground',
 56: 'silver',
 57: 'cat',
 58: 'plane',
 59: 'at',
```

```
target_word_index=y_tokens.word_index
target_word_index
```

```
{'sos': 1,
 'eos': 2,
 'ഒരു': 3,
 'കറുത്ത': 4,
 'വെളുത്ത': 5,
 'സ്ത്രീ': 6,
 'ധരിച്ച': 7,
 'മനുഷ്യൻ': 8,
 'പച്ച': 9,
 'വലിയ': 10,
 'നീല': 11,
 'ബാഗ്': 12,
 'വർണ്ണാഭമായ': 13,
 'ചുവന്ന': 14,
 'ചാരനിറത്തിലുള്ള': 15,
 'സ്കാർഫ്': 16,
 'ട്രാഷ്': 17,
 'റോഡിന്റെ': 18,
 'വിൻഡോ': 19,
 'പുല്ലിന്റെ': 20,
 'മഞ്ഞ': 21,
 'വശങ്ങളിൽ': 22,
 'സ്ട്രിപ്പുകൾ': 23,
 'വ്യക്തി': 24,
 'ഇതൊരു': 25,
 'രണ്ട്': 26,
 'ചെറിയ': 27,
 'കെട്ടിടത്തിലെ': 28,
 'നിലത്ത്': 29,
 'ആന': 30,
 'വെള്ള': 31,
 'ഒരാൾ': 32,
 'ഉള്ള': 33,
 'ചുവപ്പ്': 34,
 'തവിട്ട്': 35,
 'കുഞ്ഞ്': 36,
 'ജിറാഫ്': 37,
 'വെള്ളയും': 38,
 'ഓറഞ്ച്': 39,
 'മുകളിൽ': 40,
```

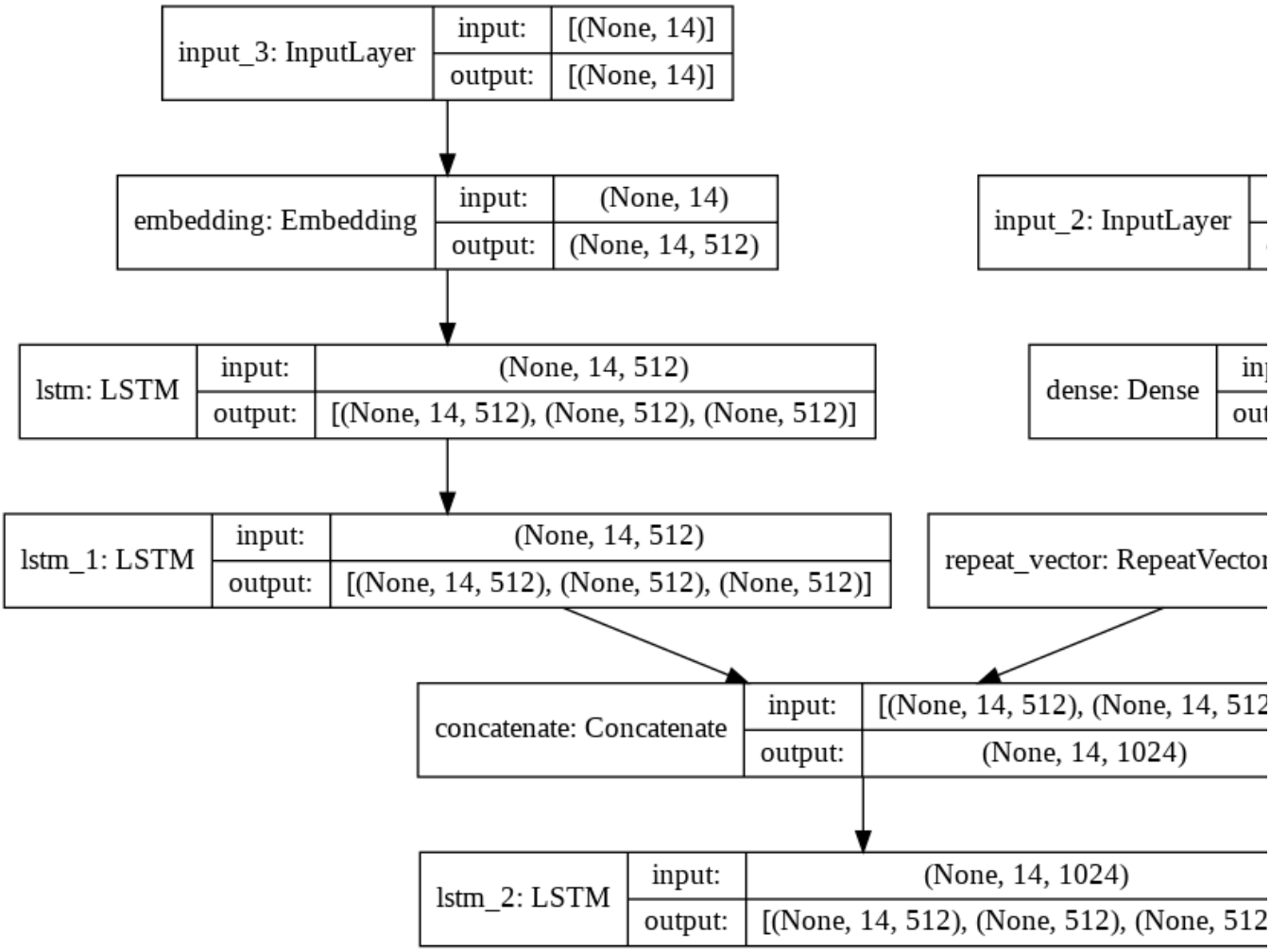
```
 'പുല്ല്': 41,
 'എഞ്ചിൻ': 42,
 'ചിഹ്നം': 43,
 'കാർ': 44,
 'കെട്ടിടത്തിന്റെ': 45,
 'മേശപ്പുറത്ത്': 46,
 'പുള്ളി': 47,
 'ആളുകൾ': 48,
 'നിറമുള്ള': 49,
 'ടെന്നീസ്': 50,
 'രംഗം': 51,
 'ട്രെയിൻ': 52,
 'തലം': 53,
 'ഇത്': 54,
 'ബീച്ച്': 55,
 'നിൽക്കുന്നു': 56,
 'തിന്നുന്നു': 57,
 'നിൽക്കുന്ന': 58,
 'ദ്വീപ്': 59,
```

```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=[encoder_inputs,img_inputs],outputs=[encoder_outputs, state_h, s
encoder_model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_3 (InputLayer)            [(None, 14)]         0
__________________________________________________________________________________________
embedding (Embedding)           (None, 14, 512)      1551872     input_3[0][0]
__________________________________________________________________________________________
input_2 (InputLayer)            [(None, 4096)]       0
__________________________________________________________________________________________
lstm (LSTM)                     [(None, 14, 512), (N 2099200     embedding[0][0]
__________________________________________________________________________________________
dense (Dense)                   (None, 512)          2097664     input_2[0][0]
__________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 14, 512), (N 2099200     lstm[0][0]
__________________________________________________________________________________________
repeat_vector (RepeatVector)    (None, 14, 512)      0           dense[0][0]
__________________________________________________________________________________________
concatenate (Concatenate)       (None, 14, 1024)     0           lstm_1[0][0]
                                                                 repeat_vector[0][0]
__________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 14, 512), (N 3147776     concatenate[0][0]
==========================================================================================
Total params: 10,995,712
Trainable params: 10,995,712
Non-trainable params: 0
__________________________________________________________________________________________
```

```
tf.keras.utils.plot_model(
```

```
    encoder_model,
    to_file='model.png',
    show_shapes=True,
    show_layer_names=True,
    rankdir='TB'
)
```



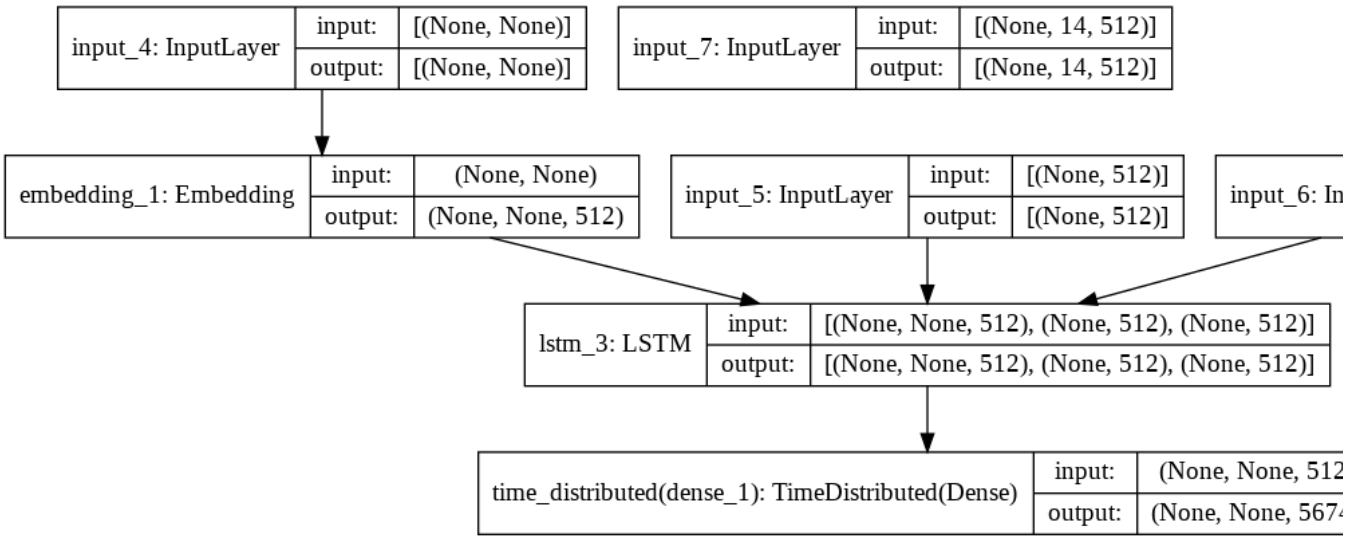

```
# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(maxlen_english,latent_dim))
print(decoder_inputs.get_shape)
#print(dec_emb.get_shape)
```

```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None) dtype=float32 (c
```

```
# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the pre
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_in

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_outputs2)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input
    [decoder_outputs2] + [state_h2, state_c2])
decoder_model.summary()
```

```
Model: "model_4"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_4 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 512)    2905088     input_4[0][0]
__________________________________________________________________________________________
input_5 (InputLayer)            [(None, 512)]        0
__________________________________________________________________________________________
input_6 (InputLayer)            [(None, 512)]        0
__________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 512),  2099200     embedding_1[1][0]
                                                                 input_5[0][0]
                                                                 input_6[0][0]
__________________________________________________________________________________________
input_7 (InputLayer)            [(None, 14, 512)]    0
__________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 5674)   2910762     lstm_3[1][0]
==========================================================================================
Total params: 7,915,050
Trainable params: 7,915,050
Non-trainable params: 0
__________________________________________________________________________________________
```

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




```
def decode_sequence(input_seq,img):
    img=img[np.newaxis,:]
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict([input_seq,img])

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['sos']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='eos'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eos'  or len(decoded_sentence.split()) >= (maxlen_malayalam -1)
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index
```

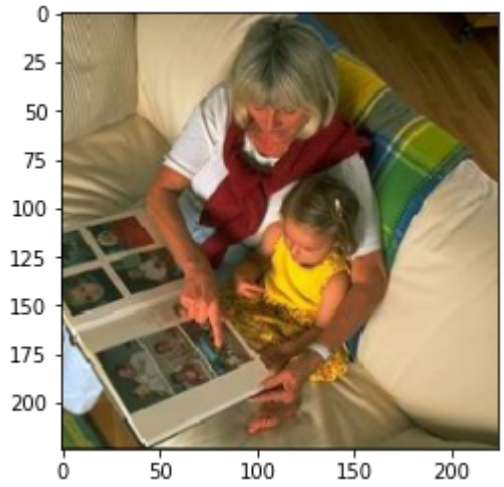
```
        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence


def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['sos']) and i!=target_word_index['eos']):
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString

for i in range(5):
    print("Review:",seq2text(x_tr[i]))
    print("Original summary:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,maxlen_english),vgg_train_[i
    print("\n")
```

```
Review: male surfer surfing in still in the ocean
Original summary: ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
Predicted summary:  തിരമാലയിൽ കയറുന്ന ഒരാൾ


Review: it is an indoor scene
Original summary: ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
Predicted summary:  മനുഷ്യന്റെ തലയിൽ കറുത്ത മാസ്ക്


Review: computer screens turned on
Original summary: കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി
Predicted summary:  കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി


Review: man has short hair
Original summary: മനുഷ്യന് ചെറിയ മുടിയുണ്ട്
Predicted summary:  മനുഷ്യൻ സ്റ്റ യിൽ നിൽക്കുന്നു


Review: photo album open on an adults lap
Original summary: ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
Predicted summary:  ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
```

```
i=4
```

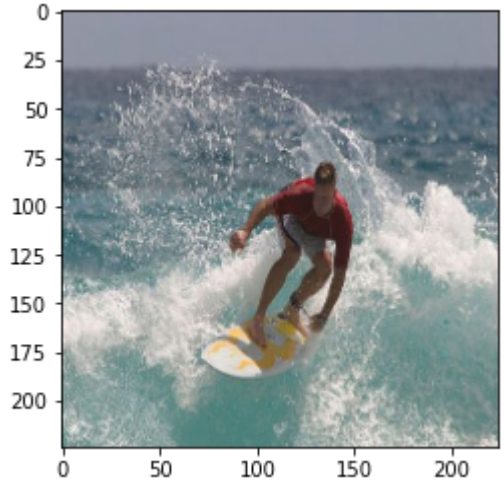
```
print("Review:",seq2text(x_tr[i]))
#print("Original summary:",seq2summary(y_tr[i]))
print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,maxlen_english),vgg_train_[i]))
plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: photo album open on an adults lap
Predicted summary:  ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
<matplotlib.image.AxesImage at 0x7ff53f2a7150>
```



```
i=0
print("Review:",seq2text(x_tr[i]))
print("Original summary:",seq2summary(y_tr[i]))
print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,maxlen_english),vgg_train_[i]))
plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: male surfer surfing in still in the ocean
Original summary: ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
Predicted summary:  തിരമാലയിൽ കയറുന്ന ഒരാൾ
<matplotlib.image.AxesImage at 0x7ff53edf4e10>
```



```
!pip install sacrebleu
import sacrebleu
import random
```

```
Collecting sacrebleu
  Downloading sacrebleu-2.0.0-py3-none-any.whl (90 kB)
     |████████████████████████████████| 90 kB 5.7 MB/s
Collecting colorama
  Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from sac
Collecting portalocker
  Downloading portalocker-2.3.2-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: tabulate>=0.8.9 in /usr/local/lib/python3.7/dist-packages
Installing collected packages: portalocker, colorama, sacrebleu
Successfully installed colorama-0.4.4 portalocker-2.3.2 sacrebleu-2.0.0
```

```
temp_o=[]
temp_p=[]
for i in range(50):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,maxlen_english),vgg_train_[s]))

bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
25.403027683651814
```

```
temp_o=[]
temp_p=[]
for i in range(10000):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,maxlen_english),vgg_train_[s]))

bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

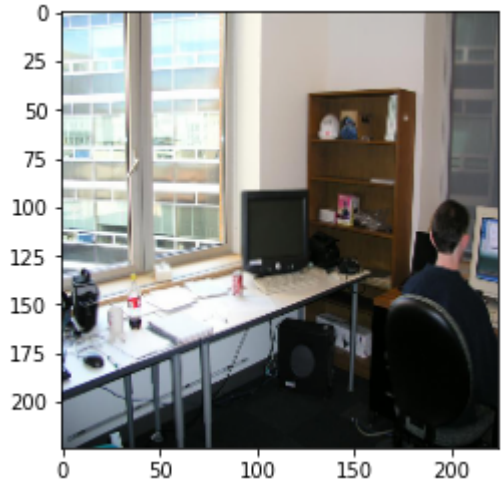
```
32.396260550602484
```

```
i=6
print("Review:",seq2text(x_tr[i]))
print("Original summary:",seq2summary(y_tr[i]))
print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,maxlen_english),vgg_train_[i]))
plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: child in a stroller
Original summary: ഒരു ഉന്തുവണ്ടിയിലെ കുട്ടി
Predicted summary:  ഒരു കറുത്ത ട്രാഷ് ബാഗ്
<matplotlib.image.AxesImage at 0x7ff53f6d18d0>
```

```
i=3
print("Review:",seq2text(x_tr[i]))
print("Original summary:",seq2summary(y_tr[i]))
print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,maxlen_english),vgg_train_[i]))
plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: man has short hair
Original summary: മനുഷ്യന് ചെറിയ മുടിയുണ്ട്
Predicted summary:  മനുഷ്യൻ സ്റ്റ യിൽ നിൽക്കുന്നു
<matplotlib.image.AxesImage at 0x7ff53f43a850>
```



Model Trained with 10000 images

```
#1000 images
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.image as mp
```

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

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

```
s=open('/content/drive/My Drive/Main/train.mn.txt')
```

```
s=open("/content/drive/My Drive/Multi/train.mn.txt")
```

```
with open('/content/drive/My Drive/Main/train.mn.txt') as f:
    train_ml = f.read().split('\n')
with open('/content/drive/My Drive/Main/train.en.txt') as f:
    train_l = f.read().split('\n')
with open('/content/drive/My Drive/Main/train_images.txt') as f:
    train_img_name = f.read().split('\n')
train_ml.pop()
train_ml.pop()
train_en.pop()
train_en.pop()
train_img_name.pop()
print(len(train_ml))
print(len(train_en))
print(len(train_img_name))
img_path=[]
for s in train_img_name:
    img_path.append("/content/drive/My Drive/Main/trainimages/train/"+s)
```

```
28930
28931
28931
```

```
train_img_name[0]
```

```
'10.jpg'
```

```
train_en[1]
```

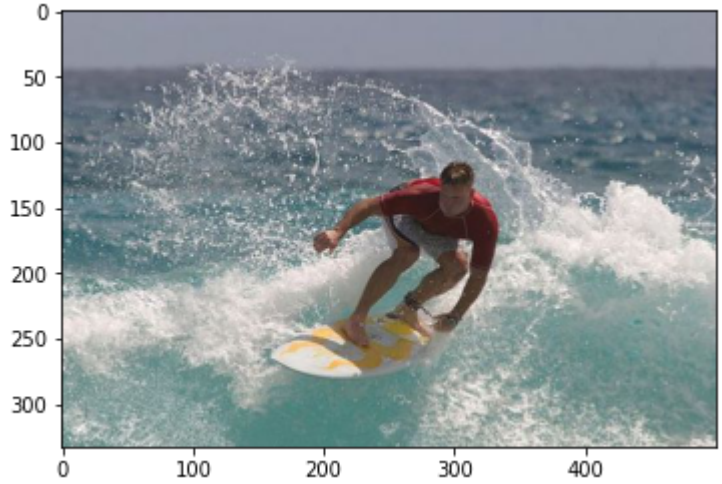
```
'it is an indoor scene\t\t\t\t\t\t'
```

```
im=mp.imread(img_path[1])
plt.imshow(im)
print(img_path[1])
print("ml:"+train_ml[1])
print("en:"+train_en[1])
```

```
/content/drive/My Drive/Main/trainimages/train/11.jpg
de:ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
en:it is an indoor scene
```


```
im=mp.imread(img_path[0])
plt.imshow(im)
print(img_path[0])
print("ml:"+train_ml[0])
print("en:"+train_en[0])
```

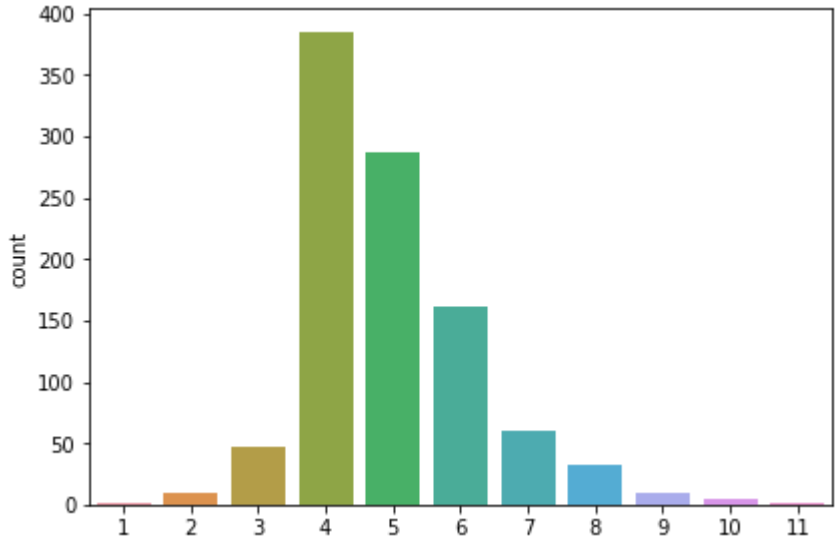
```
/content/drive/My Drive/Main/trainimages/train/10.jpg
de:ശാന്തമായ  കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
en:Male surfer surfing in still in the ocean
```



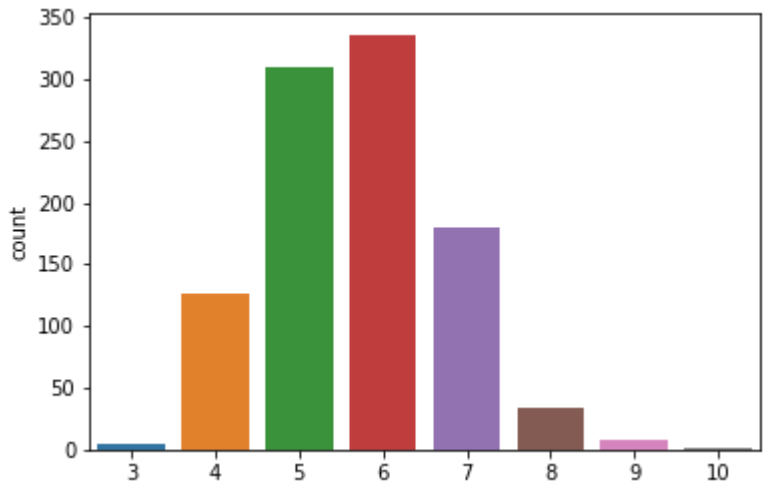
```
choicenum=1000
train_de=train_de[:choicenum]
train_en=train_en[:choicenum]


de_df = pd.DataFrame(train_ml, columns=['ml'])
en_df = pd.DataFrame(train_en, columns=['en'])
import re
def clean_text(text):
    '''Clean text by removing unnecessary characters and altering the format of words.'''
    text = text.lower()
    text = re.sub(r" ‘", "", text)
    text = re.sub(r" ‘", "", text)
    text = re.sub(r"-", " ", text)
    text = re.sub(r"<5>", "5", text)
    text = re.sub(r"“ ", "", text)
    text = re.sub(r" ”", "", text)
    text = re.sub(r"[+\.\!\/_,$%^*(+\"\']+|[+——！ ，  〈〉 《》 。 ｜？ ？、 . %~@#￥%……&* () ’]", "", t
    text=text.rstrip()
    #text=' '.join(text.split())
    return text

text1 = en_df["en"].apply(clean_text)
text2 = ml_df["ml"].apply(clean_text)
text1 = list(text1.values)
text2 = list(text2.values)


text1[1]

     'it is an indoor scene'


texttemp=[]
for s in text2:
    temp="cls "+s+" eos"
    texttemp.append(temp)
text2=[]
text2=texttemp
from sklearn.model_selection import train_test_split
english_words = []
malayalam_words = []

for i in text1:
    english_words.append(len(i.split()))

for j in text2:
    malayalam_words.append(len(j.split()))
import seaborn as sn
import matplotlib.pyplot as plt

sn.countplot(english_words)
```

```
#et_xlabel( "GFG X")
plt.tight_layout()
plt.show()

sn.countplot(malayalam_words)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
text2[1]
```

```
'cls ഇത് ഒരു ഇൻഡോർ രംഗമാണ് eos'
```

```
max_len_english = max(english_words)
max_len_malayalam = max(malayalam_words)

#from sklearn.model_selection import train_test_split
#x_tr,x_val,y_tr,y_val=train_test_split(text1,text2,test_size=0.3,random_state=12,shuffle=Tru
#print(len(x_tr))
```

```
#print(len(x_val))
x_tr=text1[:choicenum-500]
y_tr=text2[:choicenum-500]
x_val=text1[choicenum-500:]
y_val=text2[choicenum-500:]
```

```
len(x_tr)
```

```
2000
```

```
x_tr[1]
```

```
'it is an indoor scene'
```

```
len(text2)
```

```
1013
```

```
print(choicenum)
```

```
2000
```

```
a=text1[choicenum-500:]
a
```

```
[]
```

```
from keras.preprocessing.text import Tokenizer
x_tokens = Tokenizer()
x_tokens.fit_on_texts(x_tr)

x_tr = x_tokens.texts_to_sequences(x_tr)
x_val = x_tokens.texts_to_sequences(x_val)

from keras.preprocessing.sequence import pad_sequences
x_tr = pad_sequences(x_tr,maxlen = max_len_english,padding = 'post')
x_val = pad_sequences(x_val,maxlen = max_len_english,padding = 'post')

# +1 for padding
x_voc_size   =  len(x_tokens.word_index) +1

# y data
from keras.preprocessing.text import Tokenizer
y_tokens = Tokenizer()
y_tokens.fit_on_texts(y_tr)

y_tr = y_tokens.texts_to_sequences(y_tr)
y_val = y_tokens.texts_to_sequences(y_val)
```

```
from keras.preprocessing.sequence import pad_sequences
y_tr = pad_sequences(y_tr,maxlen = max_len_malayalam,padding = 'post')
y_val = pad_sequences(y_val,maxlen = max_len_malayalam,padding = 'post')

# +1 for padding
y_voc_size   =  len(y_tokens.word_index) +1


print("The document count",x_tokens.word_counts)
```

```
The document count OrderedDict([('male', 1), ('surfer', 1), ('surfing', 1), ('in', 75),
```

```
print("The document count",x_tokens.word_docs)
```

```
The document count defaultdict(<class 'int'>, {'the': 174, 'ocean': 2, 'in': 74, 'surfer
```

```
print("The document count",x_tokens.document_count)
```

```
The document count 500
```

```
print("The document count",x_tokens.word_index)
```

```
The document count {'the': 1, 'a': 2, 'on': 3, 'in': 4, 'of': 5, 'is': 6, 'white': 7, 'b
```

```
y_voc_size
```

```
3530
```

```
print("The sequences generated from text are : ",x_tr[1])
```

```
The sequences generated from text are :  [157   6  48 231 232   0   0   0   0   0   0]
```

```
len(x_tr)
```

```
500
```

```
pip install keras-applications
```

```
Collecting keras-applications
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     |████████████████████████████████| 50 kB 2.6 MB/s
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from kera
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages
```

```
Installing collected packages: keras-applications
Successfully installed keras-applications-1.0.8
```

```
import pandas as pd
import pickle
import numpy as np
import os
import keras
import tensorflow
from keras_applications.resnet import ResNet50
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, GlobalAveragePooling2D,BatchNormalization,Flatten,Input, Conv
from keras.models import Sequential, Model
from keras.utils import np_utils
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
import keras
from keras import backend as K
import gensim
from numpy import *
import numpy as np
import pandas as pd
import re
from tensorflow.keras.applications.vgg16 import VGG16
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistribut
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings
modelvgg = VGG16(include_top=True,weights="imagenet")
## load the locally saved weights
modelvgg.layers.pop()
modelvgg = Model(inputs=modelvgg.inputs, outputs=modelvgg.layers[-2].output)
modelvgg.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16
553467904/553467096 [==============================] - 6s 0us/step
553476096/553467096 [==============================] - 6s 0us/step
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
```

```
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
=================================================================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________________________________________
```

```
import cv2
import cv
```

```
ERROR:root:Error disabling cv.imshow().
Traceback (most recent call last):
  File "/usr/local/lib/python3.7/dist-packages/google/colab/_import_hooks/_cv2.py", line
    cv_module.imshow,
AttributeError: module 'cv' has no attribute 'imshow'
```

```
pip install cv
```

```
     Collecting cv
       Downloading cv-1.0.0-py3-none-any.whl (7.3 kB)
     Installing collected packages: cv
     Successfully installed cv-1.0.0
```

```
imagedata=np.zeros(shape=(choicenum,224,224,3))
for i in range(choicenum):
    temp=mp.imread(img_path[i])
    if (len(temp.shape)==3):
       temp=cv2.resize(temp,(224,224))
       imagedata[i]=temp
    elif (len(temp.shape)<3):
       temp=cv2.cvtColor(temp, cv2.COLOR_BGR2RGB)
       temp=cv2.resize(temp,(224,224))
       imagedata[i]=temp
    #print(temp)
```

```
imagedata=imagedata/255
imagedata=imagedata.astype(np.float16)
```

```
from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input
from collections import OrderedDict
jpgs=img_path[:choicenum]

images = OrderedDict()
npix = 224
target_size = (npix,npix,3)
for i,name in enumerate(jpgs):
    filename = name
    image = load_img(filename, target_size=target_size)
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    nimage = preprocess_input(image)
    y_pred = modelvgg.predict(nimage.reshape( (1,) + nimage.shape[:3]))
    images[name] = y_pred.flatten()
    if i%200==0:
        print(i,filename)
```

```
 0 /content/drive/My Drive/Main/trainimages/train/10.jpg
 200 /content/drive/My Drive/Main/trainimages/train/739.jpg
 400 /content/drive/My Drive/Main/trainimages/train/1529.jpg
 600 /content/drive/My Drive/Main/trainimages/train/2238.jpg
 800 /content/drive/My Drive/Main/trainimages/train/2970.jpg
```

```
vgg_imfea=np.zeros(shape=(len(jpgs),4096))
for i in range(len(jpgs)):
    vgg_imfea[i]=images[jpgs[i]]
```

```
train_vggf=vgg_imfea[:choicenum-500]
val_vggf=vgg_imfea[choicenum-500:]


#g_1=GlobalAveragePooling2D()(conv_3)
img_inputs=Input(shape=(4096,))
d_1=Dense(512, activation='relu')(img_inputs)
r_1=RepeatVector(max_len_english)(d_1)
vf_model = Model(img_inputs, r_1)
vf_model.summary()
```

```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, 4096)]            0         
_________________________________________________________________
dense (Dense)                (None, 512)               2097664   
_________________________________________________________________
repeat_vector (RepeatVector) (None, 11, 512)           0         
=================================================================
Total params: 2,097,664
Trainable params: 2,097,664
Non-trainable params: 0
_________________________________________________________________
```

```
x_voc=x_voc_size
y_voc=y_voc_size

# img_inputs=Input(shape=(224,224,3))
# conv_1=Conv2D(filters=64, kernel_size=(3,3), strides=(1, 1), padding='valid')(img_inputs)
# m_pool=MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')(conv_1)
# bn_1=BatchNormalization()(m_pool)
# conv_2=Conv2D(filters=128, kernel_size=(3,3), strides=(1, 1), padding='valid')(bn_1)
# conv_2=Conv2D(filters=128, kernel_size=(3,3), strides=(1, 1), padding='valid')(conv_2)
# m_pool1=MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')(conv_2)
# conv_3=Conv2D(filters=256, kernel_size=(3,3), strides=(1, 1), padding='valid')(m_pool1)
# conv_3=Conv2D(filters=256, kernel_size=(3,3), strides=(1, 1), padding='valid')(conv_3)
# m_pool2=MaxPooling2D(pool_size=(2, 2), strides=None, padding='valid')(conv_3)
# bn_2=BatchNormalization()(m_pool2)
# conv_3=Conv2D(filters=512, kernel_size=(3,3), strides=(1, 1), padding='valid')(bn_2)
# conv_3=Conv2D(filters=512, kernel_size=(3,3), strides=(1, 1), padding='valid')(conv_3)
# g_1=GlobalAveragePooling2D()(conv_3)
# d_1=Dense(512, activation='relu')(g_1)
# r_1=RepeatVector(max_len_english)(d_1)
# vf_model = Model(img_inputs, r_1)
#vf_model.summary()


latent_dim = 512
embedding_dim=512
```

```
# Encoder
encoder_inputs = Input(shape=(max_len_english,))

#embedding layer
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)

#encoder lstm 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)

#encoder lstm 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)

encoder_output2=Concatenate(axis=-1)([encoder_output2,r_1])

#encoder lstm 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)

# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))

#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)

decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurren
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[s

#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_outputs)

# Define the model
#model = Model([encoder_inputs, decoder_inputs], decoder_outputs)
model = Model([encoder_inputs,decoder_inputs,img_inputs], decoder_outputs)
model.summary()
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the crite
WARNING:tensorflow:Layer lstm will not use cuDNN kernels since it doesn't meet the crite
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernels since it doesn't meet the cri
Model: "model_2"
__________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==========================================================================================
input_3 (InputLayer)         [(None, 11)]              0
```

```
embedding (Embedding)           (None, 11, 512)      294400      input_3[0][0]
__________________________________________________________________________________________
input_2 (InputLayer)            [(None, 4096)]       0
__________________________________________________________________________________________
lstm (LSTM)                     [(None, 11, 512), (N 2099200     embedding[0][0]
__________________________________________________________________________________________
dense (Dense)                   (None, 512)          2097664     input_2[0][0]
__________________________________________________________________________________________
lstm_1 (LSTM)                   [(None, 11, 512), (N 2099200     lstm[0][0]
__________________________________________________________________________________________
repeat_vector (RepeatVector)    (None, 11, 512)      0           dense[0][0]
__________________________________________________________________________________________
input_4 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________
concatenate (Concatenate)       (None, 11, 1024)     0           lstm_1[0][0]
                                                                 repeat_vector[0][0]
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 512)    396800      input_4[0][0]
__________________________________________________________________________________________
lstm_2 (LSTM)                   [(None, 11, 512), (N 3147776     concatenate[0][0]
__________________________________________________________________________________________
lstm_3 (LSTM)                   [(None, None, 512),  2099200     embedding_1[0][0]
                                                                 lstm_2[0][1]
                                                                 lstm_2[0][2]
__________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 775)    397575      lstm_3[0][0]
==========================================================================================
Total params: 12,631,815
Trainable params: 12,631,815
Non-trainable params: 0
__________________________________________________________________________________________
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
history=model.fit([x_tr,y_tr[:,:-1],train_vggf], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)
```

```
Epoch 1/100
19/19 [==============================] - 14s 314ms/step - loss: 4.0959 - val_loss: 2.
Epoch 2/100
19/19 [==============================] - 5s 254ms/step - loss: 2.5005 - val_loss: 2.1
Epoch 3/100
19/19 [==============================] - 5s 256ms/step - loss: 2.3060 - val_loss: 2.0
Epoch 4/100
19/19 [==============================] - 5s 262ms/step - loss: 2.1863 - val_loss: 1.9
Epoch 5/100
19/19 [==============================] - 5s 255ms/step - loss: 2.1131 - val_loss: 1.8
Epoch 6/100
19/19 [==============================] - 5s 264ms/step - loss: 2.0635 - val_loss: 1.8
Epoch 7/100
19/19 [==============================] - 5s 254ms/step - loss: 2.0152 - val_loss: 1.8
Epoch 8/100
19/19 [==============================] - 5s 258ms/step - loss: 1.9631 - val_loss: 1.8
Epoch 9/100
19/19 [==============================] - 5s 256ms/step - loss: 1.9116 - val_loss: 1.7
Epoch 10/100
```

```
19/19 [==============================] - 5s 256ms/step - loss: 1.8635 - val_loss: 1.7
Epoch 11/100
19/19 [==============================] - 5s 255ms/step - loss: 1.8183 - val_loss: 1.7
Epoch 12/100
19/19 [==============================] - 5s 256ms/step - loss: 1.7786 - val_loss: 1.7
Epoch 13/100
19/19 [==============================] - 5s 255ms/step - loss: 1.7454 - val_loss: 1.6
Epoch 14/100
19/19 [==============================] - 5s 258ms/step - loss: 1.7166 - val_loss: 1.6
Epoch 15/100
19/19 [==============================] - 5s 251ms/step - loss: 1.6910 - val_loss: 1.6
Epoch 16/100
19/19 [==============================] - 5s 256ms/step - loss: 1.6670 - val_loss: 1.6
Epoch 17/100
19/19 [==============================] - 5s 256ms/step - loss: 1.6443 - val_loss: 1.6
Epoch 18/100
19/19 [==============================] - 5s 260ms/step - loss: 1.6228 - val_loss: 1.6
Epoch 19/100
19/19 [==============================] - 5s 257ms/step - loss: 1.6018 - val_loss: 1.6
Epoch 20/100
19/19 [==============================] - 5s 258ms/step - loss: 1.5818 - val_loss: 1.6
Epoch 21/100
19/19 [==============================] - 5s 252ms/step - loss: 1.5615 - val_loss: 1.6
Epoch 22/100
19/19 [==============================] - 5s 251ms/step - loss: 1.5420 - val_loss: 1.6
Epoch 23/100
19/19 [==============================] - 5s 252ms/step - loss: 1.5228 - val_loss: 1.6
Epoch 24/100
19/19 [==============================] - 5s 256ms/step - loss: 1.5044 - val_loss: 1.6
Epoch 25/100
19/19 [==============================] - 5s 254ms/step - loss: 1.4860 - val_loss: 1.6
Epoch 26/100
19/19 [==============================] - 5s 248ms/step - loss: 1.4675 - val_loss: 1.6
Epoch 27/100
19/19 [==============================] - 5s 255ms/step - loss: 1.4503 - val_loss: 1.6
Epoch 28/100
19/19 [==============================] - 5s 255ms/step - loss: 1.4330 - val_loss: 1.6
Epoch 29/100
19/19 [==============================] - 5s 250ms/step - loss: 1.4160 - val_loss: 1.6
```

```
reverse_target_word_index=y_tokens.index_word
reverse_source_word_index=x_tokens.index_word
target_word_index=y_tokens.word_index

# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=[encoder_inputs,img_inputs],outputs=[encoder_outputs, state_h, s
encoder_model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
==========================================================================================
input_3 (InputLayer)         [(None, 14)]              0
__________________________________________________________________________________________
embedding (Embedding)        (None, 14, 512)           1551872     input_3[0][0]
```

```
input_2 (InputLayer)            [(None, 4096)]       0
lstm (LSTM)                     [(None, 14, 512), (N 2099200     embedding[0][0]
dense (Dense)                   (None, 512)          2097664     input_2[0][0]
lstm_1 (LSTM)                   [(None, 14, 512), (N 2099200     lstm[0][0]
repeat_vector (RepeatVector)    (None, 14, 512)      0           dense[0][0]
concatenate (Concatenate)       (None, 14, 1024)     0           lstm_1[0][0]
                                                                 repeat_vector[0][0]
lstm_2 (LSTM)                   [(None, 14, 512), (N 3147776     concatenate[0][0]
==================================================================================
Total params: 10,995,712
Trainable params: 10,995,712
Non-trainable params: 0
```

```
# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(max_len_english,latent_dim))

# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the pre
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_in

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_outputs2)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input
    [decoder_outputs2] + [state_h2, state_c2])
decoder_model.summary()
```

```
Model: "model_4"
__________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==========================================================================================
input_4 (InputLayer)            [(None, None)]       0
__________________________________________________________________________________________
embedding_1 (Embedding)         (None, None, 512)    2905088     input_4[0][0]
__________________________________________________________________________________________
input_5 (InputLayer)            [(None, 512)]        0
__________________________________________________________________________________________
input_6 (InputLayer)            [(None, 512)]        0
```

```
lstm_3 (LSTM)                   [(None, None, 512),  2099200     embedding_1[1][0]
                                                                 input_5[0][0]
                                                                 input_6[0][0]
__________________________________________________________________________________________
input_7 (InputLayer)            [(None, 14, 512)]    0
__________________________________________________________________________________________
time_distributed (TimeDistribut (None, None, 5674)   2910762     lstm_3[1][0]
==========================================================================================
Total params: 7,915,050
Trainable params: 7,915,050
Non-trainable params: 0
__________________________________________________________________________________________
```

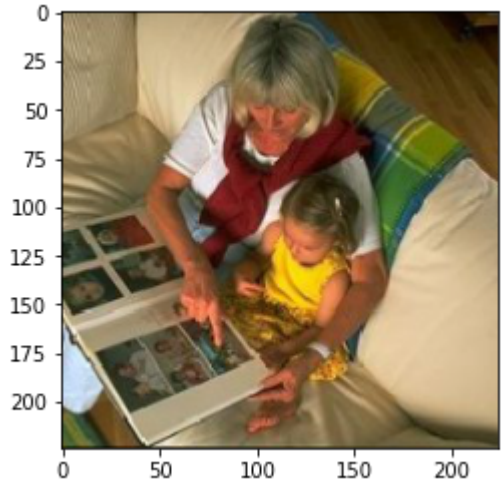
```
def decode_sequence(input_seq,img):
    img=img[np.newaxis,:]
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict([input_seq,img])

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['cls']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])

        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='eos'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eos'  or len(decoded_sentence.split()) >= (max_len_malayalam -1
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence
```

```
def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['cls']) and i!=target_word_index['eos']):
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString

for i in range(5):
    print("Review:",seq2text(x_tr[i]))
    print("Original summary:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,max_len_english),train_vggf[
    print("\n")
```

```
Review: male surfer surfing in still in the ocean
Original summary: ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
Predicted summary:  സമുദ്രത്തിലെ നല്ല തിരകൾ


Review: it is an indoor scene
Original summary: ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
Predicted summary:  ഇതൊരു കുളിമുറിയാണ്


Review: computer screens turned on
Original summary: കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി
Predicted summary:  ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പ


Review: man has short hair
Original summary: മനുഷ്യന് ചെറിയ മുടിയുണ്ട്
Predicted summary:  മനുഷ്യന് ചെറിയ മുടിയുണ്ട്


Review: photo album open on an adults lap
Original summary: ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
Predicted summary:  ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
```

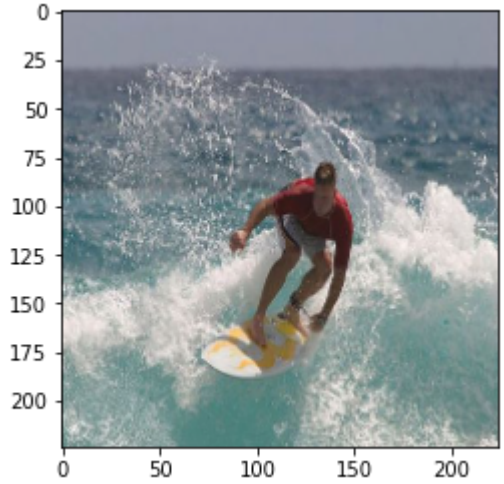
```
i=4
print("Review:",seq2text(x_tr[i]))
#print("Original summary:",seq2summary(y_tr[i]))
print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,max_len_english),vgg_imfea[i]))
plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: photo album open on an adults lap
Predicted summary:  ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
<matplotlib.image.AxesImage at 0x7ff356884110>
```



```
i=0
print("Review:",seq2text(x_tr[i]))
print("Original summary:",seq2summary(y_tr[i]))
print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,max_len_english),vgg_imfea[i]))
plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: male surfer surfing in still in the ocean
Original summary: ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
Predicted summary:  സമുദ്രത്തിലെ നല്ല തിരകൾ
<matplotlib.image.AxesImage at 0x7ff35686af90>
```



```
!pip install sacrebleu
import sacrebleu
import random
```

```
Requirement already satisfied: sacrebleu in /usr/local/lib/python3.7/dist-packages (2.0
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: colorama in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: portalocker in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from sac
Requirement already satisfied: tabulate>=0.8.9 in /usr/local/lib/python3.7/dist-packages
```

```
temp_o=[]
temp_p=[]
for i in range(50):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,max_len_english),train_vggf[s]))

bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
44.835907663936744
```

```
temp_o=[]
temp_p=[]
for i in range(10000):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,max_len_english),train_vggf[s]))

bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
44.08897167055731
```

```
temp_o=[]
temp_p=[]
for i in range(100):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,max_len_english),vgg_imfea[s]))

bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
34.843988395192
```

English_Malayalam_Multimodal_Machine_Translation

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.image as mp
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mour
```

```
with open('/content/drive/My Drive/Main/train.mn.txt') as file:
    train_mal_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/train.en.txt') as file:
    train_eng_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/train_images.txt') as file:
    train_images = file.read().split('\n')
```

```
with open('/content/drive/My Drive/Main/maldev.txt') as file:
    dev_mal_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/endev.txt') as file:
    dev_eng_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/devimage.txt') as file:
    dev_images = file.read().split('\n')
```

```
with open('/content/drive/My Drive/Main/maltest.txt') as file:
   test_mal_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/engtest.txt') as file:
   test_eng_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/testimages.txt') as file:
    test_images = file.read().split('\n')
```

```
def remove(mal_txt,eng_txt):
        mal_txt.pop()
        mal_txt.pop()
        eng_txt.pop()
        eng_txt.pop()
        #trainimages.pop()
        #trainimages,link
```

```
        print(len(mal_txt))
        print(len(eng_txt))
        #print(len(trainimages))
        #img_path=[]
        #for s in trainimages:
                #img_path.append(link+s)
        #return mal_txt,eng_txt,train_images,img_path
        return mal_txt,eng_txt
```

Double-click (or enter) to edit

```
##training image
#link="/content/drive/My Drive/Main/trainimages/train/"
train_mal_txt,train_eng_txt=remove(train_mal_txt,train_eng_txt)
```

```
28930
28931
```

```
testvgg_feature=np.load('/content/drive/My Drive/Main/testfeature.npy', encoding='bytes')
```

```
#dev image
#link="/content/drive/My Drive/Main/trainimages/train/"
dev_mal_txt,dev_eng_txt=remove(dev_mal_txt,dev_eng_txt)
```

```
997
998
```

```
test_mal_txt,test_eng_txt=remove(test_mal_txt,test_eng_txt)
```

```
1399
1399
```

```
ttmal_df = pd.DataFrame(test_mal_txt, columns=['Malayalam'])
tteng_df = pd.DataFrame(test_eng_txt, columns=['English'])
```

```
ttmal_text1 = ttmal_df["Malayalam"].apply(clean_text)
tteng_text1 = tteng_df["English"].apply(clean_text)
ttmal_text2 = list(ttmal_text1.values)
tteng_text2 = list(tteng_text1.values)
```

```
x_tt=tteng_text2
y_tt=ttmal_text2
```

```
x_tt[1]
```

```
'knife block sitting on counter with knives in it'
```

```
y_tt[1]
```

```
'sos കത്തികൊണ്ട് കൗണ്ടറിൽ ഇരിക്കുന്ന കത്തി ബ്ലോക്ക് eos'
```

```
ttmal_temp=[]
for s in ttmal_text2:
    tttemp="sos "+s+" eos"
    ttmal_temp.append(tttemp)
#text2=[]
ttmal_text2=ttmal_temp
ttmal_text2[1:10]
```

```
['sos കത്തികൊണ്ട് കൗണ്ടറിൽ ഇരിക്കുന്ന കത്തി ബ്ലോക്ക് eos',
 'sos ചട്ടിയിൽ രണ്ടാമത്തെ പിസ്സ eos',
 'sos ബീജ് സ്റ്റെയർ വേ രണ്ടാം ലെവലിലേക്ക് പോകുന്നു eos',
 'sos ടാൻ ഹൗസിലെ ഇളം നിറമുള്ള രണ്ടാം നില eos',
 'sos കെട്ടിടങ്ങളുടെ രണ്ടാമത്തെ നിലയിലെ ബാൽക്കണി eos',
 'sos രാത്രി സ്റ്റാന്റിലെ വിളക്ക് കട്ടിലിന്റെ ഇടതുവശത്ത് നിൽക്കുന്നു eos',
 'sos നൈറ്റ് സ്റ്റാൻഡിന് മുകളിൽ ഒരു വെളുത്ത അലാറം ക്ലോക്ക് ഉണ്ട് eos',
 'sos ഒരു കറുത്ത സംഗീത സ്റ്റാൻഡ് eos',
 'sos തെരുവ് മുറിച്ചുകടക്കാൻ കാത്തിരിക്കുന്ന സ്ത്രീ eos']
```

```
trmal_df = pd.DataFrame(train_mal_txt, columns=['Malayalam'])
treng_df = pd.DataFrame(train_eng_txt, columns=['English'])
```

```
#Datacleaning by removing special characters
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r" ‘", "", text)
    text = re.sub(r" ‘", "", text)
    text = re.sub(r"-", " ", text)
    text = re.sub(r"<5>", "5", text)
    text = re.sub(r"“ ", "", text)
    text = re.sub(r" ”", "", text)
    text = re.sub(r"[+\.\!\/_,$%^*(+\"\']+|[+——！ ， 〈〉 《》 。 |？ ?、 . %~@#￥%……&* () ’]", "", t
    text=text.rstrip()
    return text
```

```
trmal_text1 = trmal_df["Malayalam"].apply(clean_text)
treng_text1 = treng_df["English"].apply(clean_text)
trmal_text2 = list(trmal_text1.values)
treng_text2 = list(treng_text1.values)
```

```
trmal_text2[0]
```

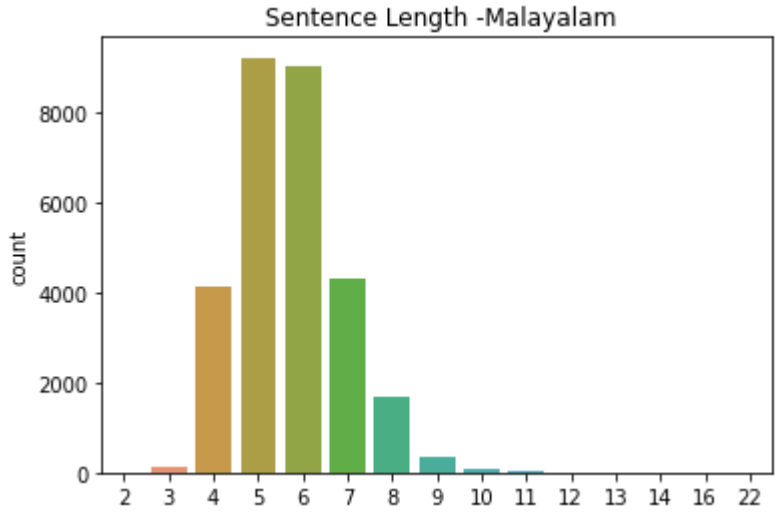
```
'ശാന്തമായ  കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ'
```

```
trmal_temp=[]
for s in trmal_text2:
    trtemp="sos "+s+" eos"
    trmal_temp.append(trtemp)
#text2=[]
trmal_text2=trmal_temp
trmal_text2[1:10]
```

```
['sos ഇത് ഒരു ഇൻഡോർ രംഗമാണ് eos',
 'sos കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി eos',
 'sos മനുഷ്യന് ചെറിയ മുടിയുണ്ട് eos',
 'sos ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു eos',
 'sos കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട് eos',
 'sos ഒരു ഉന്തുവണ്ടിയിലെ കുട്ടി eos',
 'sos ഉയരമുള്ള മെറ്റൽ ലൈറ്റ്പോസ്റ്റ് eos',
 'sos മതിൽ വെളുത്ത ചായം പൂശി eos',
 'sos ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പുകൾ eos']
```

```
import seaborn as sn
import matplotlib.pyplot as plt
trmalayalam_words = []
for i in trmal_text2:
    trmalayalam_words.append(len(i.split()))
sn.countplot(trmalayalam_words).set(title=' Sentence Length -Malayalam')
plt.show()
```

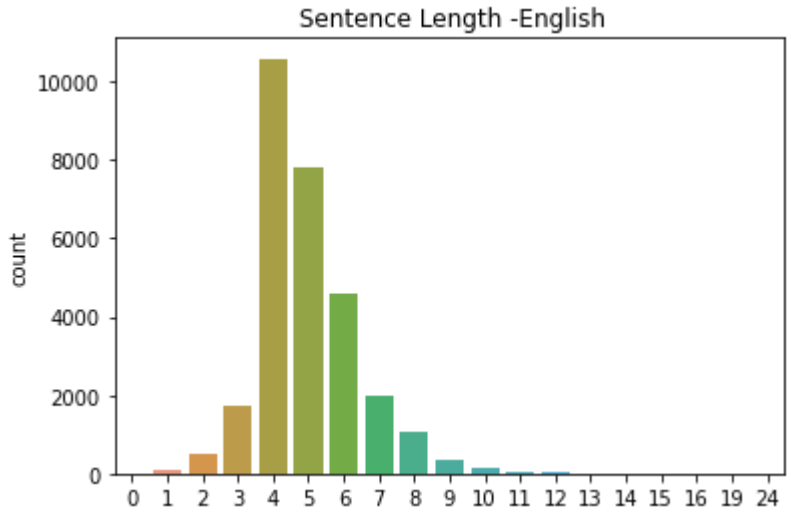
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
trenglish_words = []
```

```
for j in treng_text2:
    trenglish_words.append(len(j.split()))
sn.countplot(trenglish_words).set(title=' Sentence Length -English')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
trmaxlen_malayalam = max(trmalayalam_words)
trmaxlen_english = max(trenglish_words)
print('Maximum sentence length-Malayalam :',trmaxlen_malayalam)
print('Maximum sentence length-English :',trmaxlen_english)
```

```
Maximum sentence length-Malayalam : 22
Maximum sentence length-English : 24
```

```
#dev

#Datacleaning by removing special characters
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r" ‘", "", text)
    text = re.sub(r" ‘", "", text)
    text = re.sub(r"-", " ", text)
    text = re.sub(r"<5>", "5", text)
    text = re.sub(r"“ ", "", text)
    text = re.sub(r" ”", "", text)
    text = re.sub(r"[+\.\!\/_,$%^*(+\"\']+|[+——！ ，  〈〉 《》 。 ｜？ ？、 . %~@#￥%……&* () ’]", "", t
    text=text.rstrip()
    return text


dmal_df = pd.DataFrame(dev_mal_txt, columns=['Malayalam'])
deng_df = pd.DataFrame(dev_eng_txt, columns=['English'])
```

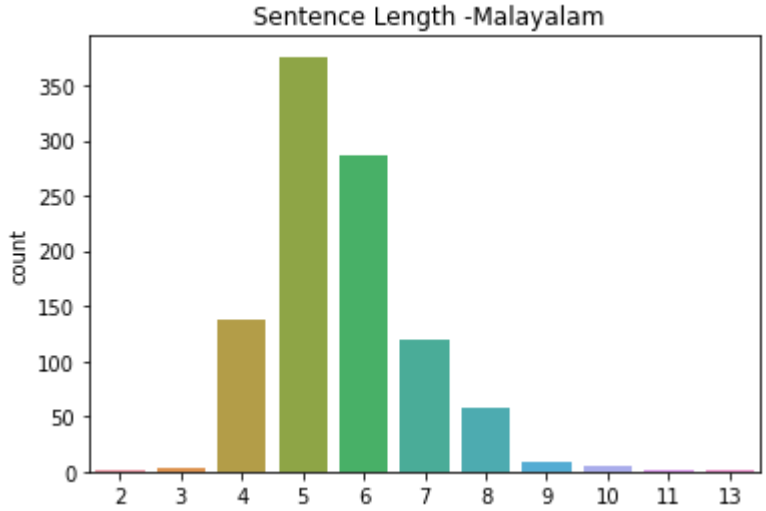
```
dmal_text1 = dmal_df["Malayalam"].apply(clean_text)
deng_text1 = deng_df["English"].apply(clean_text)
dmal_text2 = list(dmal_text1.values)
deng_text2 = list(deng_text1.values)
```

```
dmal_temp=[]
for s in dmal_text2:
    dtemp="sos "+s+" eos"
    dmal_temp.append(dtemp)
#text2=[]
dmal_text2=dmal_temp
dmal_text2[1:10]
```

```
['sos ഒരു കെട്ടിടത്തിന്റെ വിൻഡോ eos',
 'sos ഡ്രോയറുകളുള്ള ഇരുണ്ട ചാരനിറത്തിലുള്ള കമ്പ്യൂട്ടർ ഡെസ്ക് eos',
 'sos നിലത്ത് നാല് കാലുകളുള്ള ഉരുക്ക് കസേര eos',
 'sos തെരുവിൽ സൈക്കിൾ ചവിട്ടുന്ന മനുഷ്യൻ eos',
 'sos ഈ കാറുകൾ അഴുക്കിൽ പാർക്ക് ചെയ്തിരിക്കുന്നു eos',
 'sos ആകാശത്ത് വെളുത്ത മേഘങ്ങൾ eos',
 'sos ഗ്ലാസ് വാതിലുകളുടെ ഗണം eos',
 'sos പ്രൊജക്ടർ സ്ക്രീൻ ചുരുട്ടി വെച്ചിരിക്കുന്നു eos',
 'sos ചാരനിറത്തിലുള്ള സെറ്റർ ധരിച്ച മനുഷ്യൻ eos']
```

```
import seaborn as sn
import matplotlib.pyplot as plt
dmalayalam_words = []
for i in dmal_text2:
    dmalayalam_words.append(len(i.split()))
sn.countplot(dmalayalam_words).set(title=' Sentence Length -Malayalam')
plt.show()
```

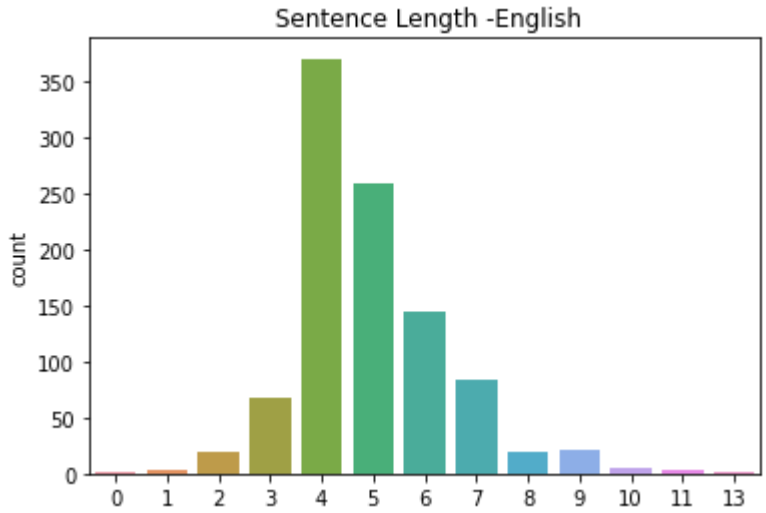
```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
denglish_words = []
```

```
for j in deng_text2:
    denglish_words.append(len(j.split()))
sn.countplot(denglish_words).set(title=' Sentence Length -English')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
dmaxlen_malayalam = max(dmalayalam_words)
dmaxlen_english = max(denglish_words)
print('Maximum sentence length-Malayalam :',dmaxlen_malayalam)
print('Maximum sentence length-English :',dmaxlen_english)
```

```
Maximum sentence length-Malayalam : 13
Maximum sentence length-English : 13
```

```
ttmal_df = pd.DataFrame(test_mal_txt, columns=['Malayalam'])
tteng_df = pd.DataFrame(test_eng_txt, columns=['English'])
```

```
ttmal_text1 = ttmal_df["Malayalam"].apply(clean_text)
tteng_text1 = tteng_df["English"].apply(clean_text)
ttmal_text2 = list(ttmal_text1.values)
tteng_text2 = list(tteng_text1.values)
```

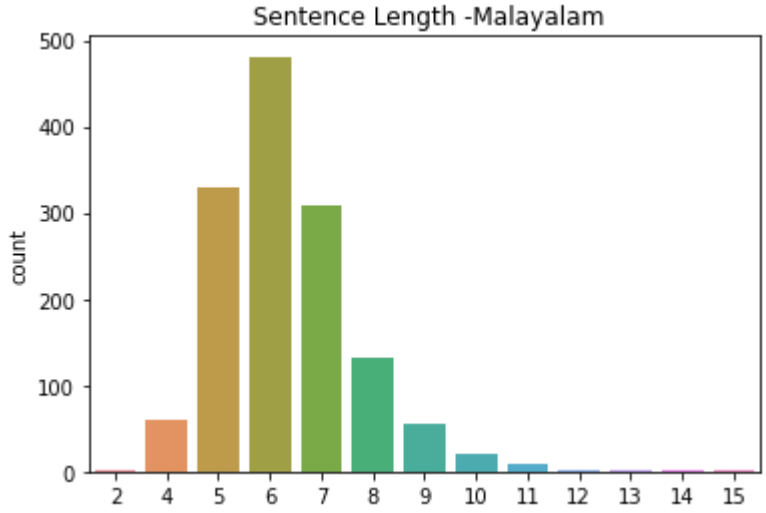
```
ttmal_temp=[]
for s in ttmal_text2:
    tttemp="sos "+s+" eos"
    ttmal_temp.append(tttemp)
#text2=[]
ttmal_text2=ttmal_temp
ttmal_text2[1:10]
```

```
['sos കത്തികൊണ്ട് കൗണ്ടറിൽ ഇരിക്കുന്ന കത്തി ബ്ലോക്ക് eos',
 'sos ചട്ടിയിൽ രണ്ടാമത്തെ പിസ്സ eos',
```

```
'sos ബീജ് സ്റ്റെയർ വേ രണ്ടാം ലെവലിലേക്ക് പോകുന്നു eos',
'sos ടാൻ ഹൗസിലെ ഇളം നിറമുള്ള രണ്ടാം നില eos',
'sos കെട്ടിടങ്ങളുടെ രണ്ടാമത്തെ നിലയിലെ ബാൽക്കണി eos',
'sos രാത്രി സ്റ്റാന്റിലെ വിളക്ക് കട്ടിലിന്റെ ഇടതുവശത്ത് നിൽക്കുന്നു eos',
'sos നൈറ്റ് സ്റ്റാൻഡിന് മുകളിൽ ഒരു വെളുത്ത അലാറം ക്ലോക്ക് ഉണ്ട് eos',
'sos ഒരു കറുത്ത സംഗീത സ്റ്റാൻഡ് eos',
'sos തെരുവ് മുറിച്ചുകടക്കാൻ കാത്തിരിക്കുന്ന സ്ത്രീ eos']
```

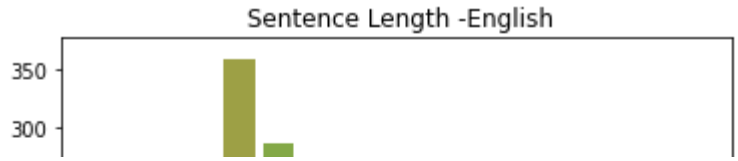
```
import seaborn as sn
import matplotlib.pyplot as plt
ttmalayalam_words = []
for i in ttmal_text2:
    ttmalayalam_words.append(len(i.split()))
sn.countplot(ttmalayalam_words).set(title=' Sentence Length -Malayalam')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
ttenglish_words = []
for j in tteng_text2:
    ttenglish_words.append(len(j.split()))
sn.countplot(ttenglish_words).set(title=' Sentence Length -English')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
ttmaxlen_malayalam = max(ttmalayalam_words)
ttmaxlen_english = max(ttenglish_words)
print('Maximum sentence length-Malayalam :',ttmaxlen_malayalam)
print('Maximum sentence length-English :',ttmaxlen_english)
```

```
Maximum sentence length-Malayalam : 15
Maximum sentence length-English : 17
```

```
x_tr=treng_text2
y_tr=trmal_text2
x_val=deng_text2
y_val=dmal_text2



#Tokening the sentences using Keras tokenizer -Malayalam data
from keras.preprocessing.text import Tokenizer
x_tokens = Tokenizer()
x_tokens.fit_on_texts(x_tr)
x_tr = x_tokens.texts_to_sequences(x_tr)
x_val = x_tokens.texts_to_sequences(x_val)
print('x_tr:',x_tr)
print('x_val:',x_val)
```

```
x_tr: [[432, 237, 187, 6, 1418, 6, 2, 173], [145, 5, 29, 2227, 252], [146, 1864, 506, 3
x_val: [[1, 561, 13, 24], [50, 4, 1, 28], [1, 113, 106, 146, 164, 11, 1754], [212, 1871,
```

```
#padding with post (appending zeros at the end to equalize sentence length)
from keras.preprocessing.sequence import pad_sequences
x_tr = pad_sequences(x_tr,maxlen = trmaxlen_english,padding = 'post')
x_val = pad_sequences(x_val,maxlen = trmaxlen_english,padding = 'post')

# +1 for padding
x_voc_size   =  len(x_tokens.word_index) +1
print("No of unique words in English",x_voc_size)
```

```
No of unique words in English 5570
```

```
#Tokening the sentences using Keras tokenizer -Malayalam data
from keras.preprocessing.text import Tokenizer
y_tokens = Tokenizer()
y_tokens.fit_on_texts(y_tr)
```

```
y_tr = y_tokens.texts_to_sequences(y_tr)
y_val = y_tokens.texts_to_sequences(y_val)
print('x_tr:',y_tr)
print('x_val:',y_val)
```

```
x_tr: [[1, 974, 1376, 4856, 1158, 672, 287, 2], [1, 62, 3, 3335, 3336, 2], [1, 147, 2631
x_val: [[1, 343, 293, 2], [1, 3, 111, 31, 2], [1, 129, 12, 147, 1388, 2], [1, 45, 206, 3
```

```
#padding with post (appending zeros at the end to equalize sentence length)
from keras.preprocessing.sequence import pad_sequences
y_tr = pad_sequences(y_tr,maxlen = trmaxlen_malayalam,padding = 'post')
y_val = pad_sequences(y_val,maxlen = trmaxlen_malayalam,padding = 'post')

# +1 for padding
y_voc_size   =  len(y_tokens.word_index) +1
print("No of unique words in English",y_voc_size)
```

```
No of unique words in English 11314
```

```
x_voc_size   =  len(x_tokens.word_index) +1
print("No of unique words in English",x_voc_size)
```

```
No of unique words in English 5570
```

```
y_voc_size   =  len(y_tokens.word_index) +1
print("No of unique words in English",y_voc_size)
```

```
No of unique words in English 11314
```

```
#Tokening the sentences using Keras tokenizer -Malayalam data
from keras.preprocessing.text import Tokenizer
x_tokens = Tokenizer()
x_tokens.fit_on_texts(x_tt)
x_tt = x_tokens.texts_to_sequences(x_tt)
print('x_tt:',x_tt)
```

```
x_tt: [[121, 12, 596, 35], [131, 12, 56, 3, 597, 16, 132, 7, 93], [2, 11, 400, 7, 2, 598
```

```
from keras.preprocessing.sequence import pad_sequences
x_tt = pad_sequences(x_tt,maxlen = trmaxlen_english,padding = 'post')
```

```
from keras.preprocessing.text import Tokenizer
y_tokens = Tokenizer()
y_tokens.fit_on_texts(y_tt)
y_tt = y_tokens.texts_to_sequences(y_tt)
```

```
y_tt = pad_sequences(y_tt,maxlen = trmaxlen_english,padding = 'post')
```

```
pip install keras-applications
```

```
Collecting keras-applications
  Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
     |████████████████████████████████| 50 kB 2.6 MB/s
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from kera
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-packages (f
Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages
Installing collected packages: keras-applications
Successfully installed keras-applications-1.0.8
```

```
import pandas as pd
import pickle
import numpy as np
import os
import keras
import tensorflow
from keras_applications.resnet import ResNet50
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, GlobalAveragePooling2D,BatchNormalization,Flatten,Input, Conv
from keras.models import Sequential, Model
from keras.utils import np_utils
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
import keras
from keras import backend as K
import gensim
from numpy import *
import numpy as np
import pandas as pd
import re
from tensorflow.keras.applications.vgg16 import VGG16
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistribut
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings
```

```
trvgg_feature=np.load('/content/drive/My Drive/Main/combine.npy', encoding='bytes')
```

```
dvgg_feature=np.load('/content/drive/My Drive/Main/valfeature.npy', encoding='bytes')
```

```
print(len(trvgg_feature))
print(len(x_tr))
print(len(y_tr))
```

```
28929
28933
28932
```

```
print(len(trvgg_feature))
x_tr=x_tr[:-4]
print(len(x_tr))
y_tr=y_tr[:-2]
print(len(y_tr))
```

```
28929
28929
28930
```

```
y_tr=y_tr[:-1]
print(len(y_tr))
```

```
28929
```

```
dvgg_feature=dvgg_feature[:-1]
print(len(dvgg_feature))
```

```
998
```

```
x_val=q[:-2]
print(len(x_val))
```

```
998
```

```
y_val=z[:-1]
print(len(y_val))
```

```
998
```

```
#y_tr=y_tr[:-2]
print(len(y_val))
```

```
999
```

```
print(len(dvgg_feature))
print(len(x_val))
print(len(y_val))
```

```
998
998
999
```

```
p=dvgg_feature
q=x_val
z=y_val
```

```
#splitting image pixels for training and validation
vgg_train_=trvgg_feature
vgg_val=dvgg_feature
```

```
#Generating a repeat vector from image pixels
img_inputs=Input(shape=(4096,))
d_1=Dense(512, activation='relu')(img_inputs)
r_1=RepeatVector(trmaxlen_english)(d_1)
vf_model = Model(img_inputs, r_1)
vf_model.summary()
```

```
Model: "model_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_12 (InputLayer)       [(None, 4096)]            0

 dense_4 (Dense)             (None, 512)               2097664

 repeat_vector_2 (RepeatVect  (None, 24, 512)          0
 or)

=================================================================
Total params: 2,097,664
Trainable params: 2,097,664
Non-trainable params: 0
_________________________________________________________________
```

```
x_voc=x_voc_size
y_voc=y_voc_size
```

```
y_voc
```

```
11314
```

```
#Model
x_voc=x_voc_size
```

```
y_voc=y_voc_size
latent_dim = 512
embedding_dim=512
#Encoder
encoder_inputs = Input(shape=(trmaxlen_english,))
#The model will take as input an integer matrix of size (batch,input_length)and the largest i
enc_emb =  Embedding(x_voc, embedding_dim,trainable=True)(encoder_inputs)
print(encoder_inputs.get_shape)
print(enc_emb.get_shape)
```

```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, 24) dtype=float32 (cre
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, 24, 512) dtype=float32
```

```
#encoder LSTM Layer 1
encoder_lstm1 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent
#The dimension of each state equals to the LSTM unit number
encoder_output1, state_h1, state_c1 = encoder_lstm1(enc_emb)
print(encoder_lstm1.output_shape)
```

```
WARNING:tensorflow:Layer lstm_8 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_8 will not use cuDNN kernels since it doesn't meet the cri
[(None, 24, 512), (None, 512), (None, 512)]
```

```
#LSTM layer 2
encoder_lstm2 = LSTM(latent_dim,return_sequences=True,return_state=True,dropout=0.4,recurrent
encoder_output2, state_h2, state_c2 = encoder_lstm2(encoder_output1)
print(encoder_lstm2.output_shape)
```

```
WARNING:tensorflow:Layer lstm_9 will not use cuDNN kernels since it doesn't meet the cri
WARNING:tensorflow:Layer lstm_9 will not use cuDNN kernels since it doesn't meet the cri
[(None, 24, 512), (None, 512), (None, 512)]
```

```
#Concatenating image features with text input
encoder_output2=Concatenate(axis=-1)([encoder_output2,r_1])
```

```
#LSTM layer 3
encoder_lstm3=LSTM(latent_dim, return_state=True, return_sequences=True,dropout=0.4,recurrent
encoder_outputs, state_h, state_c= encoder_lstm3(encoder_output2)
print(encoder_lstm3.output_shape)
```

```
WARNING:tensorflow:Layer lstm_10 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_10 will not use cuDNN kernels since it doesn't meet the cr
[(None, 24, 512), (None, 512), (None, 512)]
```

```
#Decoder
# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))
#embedding layer
dec_emb_layer = Embedding(y_voc, embedding_dim,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)
print(decoder_inputs.get_shape)
print(dec_emb.get_shape)
```

```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None) dtype=float32 (c
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None, 512) dtype=float
```

```
#Decoder LSTM layer1
decoder_lstm = LSTM(latent_dim, return_sequences=True, return_state=True,dropout=0.4,recurren
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=[s
print(decoder_lstm.output_shape)
```

```
WARNING:tensorflow:Layer lstm_11 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_11 will not use cuDNN kernels since it doesn't meet the cr
[(None, None, 512), (None, 512), (None, 512)]
```

```
#dense layer
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_outputs)
print(decoder_dense.output_shape)
```

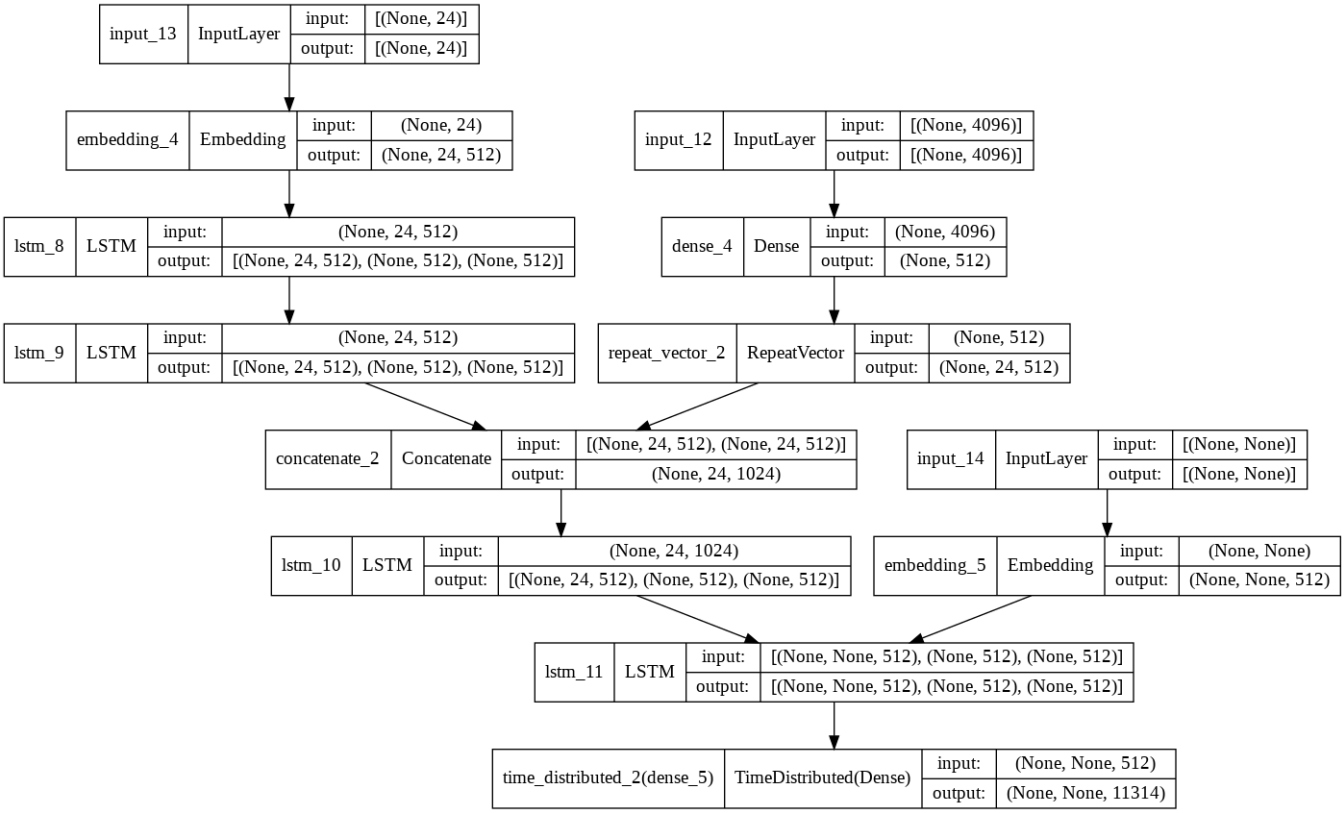
```
(None, None, 11314)
```

```
model = Model([encoder_inputs,decoder_inputs,img_inputs], decoder_outputs)
model.summary()
```

```
Model: "model_7"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_13 (InputLayer)          [(None, 24)]         0           []

 embedding_4 (Embedding)        (None, 24, 512)      2851840     ['input_13[0][0]']

 input_12 (InputLayer)          [(None, 4096)]       0           []

 lstm_8 (LSTM)                  [(None, 24, 512),    2099200     ['embedding_4[0][0]']
                                 (None, 512),
                                 (None, 512)]

 dense_4 (Dense)                (None, 512)          2097664     ['input_12[0][0]']

 lstm_9 (LSTM)                  [(None, 24, 512),    2099200     ['lstm_8[0][0]']
                                 (None, 512),
                                 (None, 512)]
```

```
repeat_vector_2 (RepeatVector)  (None, 24, 512)     0           ['dense_4[0][0]']

input_14 (InputLayer)           [(None, None)]      0           []

concatenate_2 (Concatenate)     (None, 24, 1024)    0           ['lstm_9[0][0]',
                                                                 'repeat_vector_2[0][0]

embedding_5 (Embedding)         (None, None, 512)   5792768     ['input_14[0][0]']

lstm_10 (LSTM)                  [(None, 24, 512),   3147776     ['concatenate_2[0][0]']
                                 (None, 512),
                                 (None, 512)]

lstm_11 (LSTM)                  [(None, None, 512),  2099200    ['embedding_5[0][0]',
                                 (None, 512),                    'lstm_10[0][1]',
                                 (None, 512)]                    'lstm_10[0][2]']

time_distributed_2 (TimeDistri  (None, None, 11314)  5804082    ['lstm_11[0][0]']
buted)

==================================================================================================
Total params: 25,991,730
Trainable params: 25,991,730
Non-trainable params: 0
__________________________________________________________________________________________________
```

```
from keras.utils.vis_utils import plot_model
import tensorflow as tf


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




```
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint("/content/drive/My Drive/Main/modelfull2", monitor='val_accuracy

early_stopping = EarlyStopping(monitor='val_accuracy', patience=5)

callbacks_list = [checkpoint, early_stopping]

model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])

len(y_tr)
```

```
28930
```

```
y_val=y_val[-1]

len(y_val)
```

```
22
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history=model.fit([x_tr,y_tr[:,:-1],vgg_train_], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)
```

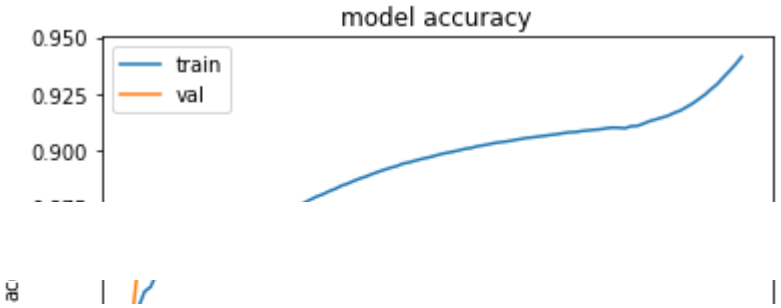
```
Epoch 1/100
57/57 [==============================] - 47s 635ms/step - loss: 2.3283 - accuracy: 0.
Epoch 2/100
57/57 [==============================] - 35s 618ms/step - loss: 1.3360 - accuracy: 0.
Epoch 3/100
57/57 [==============================] - 35s 617ms/step - loss: 1.2648 - accuracy: 0.
Epoch 4/100
57/57 [==============================] - 35s 615ms/step - loss: 1.2192 - accuracy: 0.
Epoch 5/100
57/57 [==============================] - 35s 614ms/step - loss: 1.1697 - accuracy: 0.
Epoch 6/100
57/57 [==============================] - 35s 614ms/step - loss: 1.1217 - accuracy: 0.
Epoch 7/100
57/57 [==============================] - 35s 614ms/step - loss: 1.0838 - accuracy: 0.
Epoch 8/100
57/57 [==============================] - 35s 610ms/step - loss: 1.0546 - accuracy: 0.
Epoch 9/100
57/57 [==============================] - 35s 612ms/step - loss: 1.0307 - accuracy: 0.
Epoch 10/100
57/57 [==============================] - 35s 613ms/step - loss: 1.0097 - accuracy: 0.
Epoch 11/100
57/57 [==============================] - 35s 611ms/step - loss: 0.9907 - accuracy: 0.
Epoch 12/100
57/57 [==============================] - 35s 613ms/step - loss: 0.9737 - accuracy: 0.
Epoch 13/100
57/57 [==============================] - 35s 608ms/step - loss: 0.9582 - accuracy: 0.
Epoch 14/100
57/57 [==============================] - 35s 608ms/step - loss: 0.9430 - accuracy: 0.
Epoch 15/100
57/57 [==============================] - 35s 610ms/step - loss: 0.9288 - accuracy: 0.
Epoch 16/100
57/57 [==============================] - 35s 605ms/step - loss: 0.9149 - accuracy: 0.
Epoch 17/100
57/57 [==============================] - 35s 609ms/step - loss: 0.9016 - accuracy: 0.
Epoch 18/100
57/57 [==============================] - 35s 608ms/step - loss: 0.8887 - accuracy: 0.
Epoch 19/100
57/57 [==============================] - 35s 609ms/step - loss: 0.8761 - accuracy: 0.
Epoch 20/100
57/57 [==============================] - 35s 608ms/step - loss: 0.8634 - accuracy: 0.
Epoch 21/100
57/57 [==============================] - 35s 611ms/step - loss: 0.8513 - accuracy: 0.
Epoch 22/100
57/57 [==============================] - 35s 608ms/step - loss: 0.8395 - accuracy: 0.
Epoch 23/100
57/57 [==============================] - 35s 609ms/step - loss: 0.8280 - accuracy: 0.
Epoch 24/100
57/57 [==============================] - 35s 609ms/step - loss: 0.8164 - accuracy: 0.
Epoch 25/100
57/57 [==============================] - 35s 610ms/step - loss: 0.8041 - accuracy: 0.
Epoch 26/100
57/57 [==============================] - 35s 612ms/step - loss: 0.7923 - accuracy: 0.
Epoch 27/100
57/57 [==============================] - 35s 614ms/step - loss: 0.7806 - accuracy: 0.
```

```
Epoch 28/100
57/57 [==============================] - 35s 611ms/step - loss: 0.7713 - accuracy: 0.
Epoch 29/100
57/57 [==============================] - 35s 609ms/step - loss: 0.7580 - accuracy: 0.
```

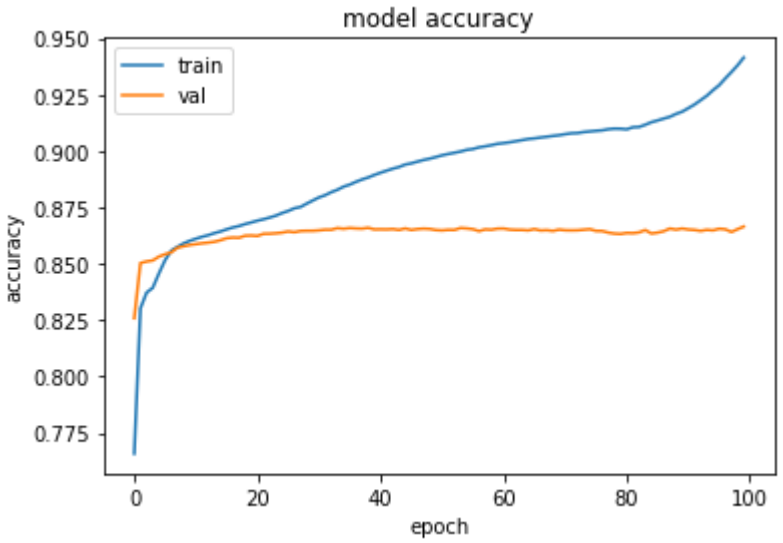
```
#compiling model

history=model.fit([x_tr,y_tr,vgg_train_], y_tr ,validation_data=([x_val,y_val,vgg_val], y_val
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-253-77cd7e4c4882> in <module>()
      1 #compiling model
      2
----> 3 history=model.fit([x_tr,y_tr,vgg_train_], y_tr ,validation_data=
([x_val,y_val,vgg_val], y_val),epochs=100,batch_size=512,callbacks=[callbacks_list ])

 1 frames
/usr/local/lib/python3.7/dist-packages/keras/engine/data_adapter.py in
_check_data_cardinality(data)
   1655                           for i in tf.nest.flatten(single_data)))
   1656     msg += "Make sure all arrays contain the same number of samples."
-> 1657     raise ValueError(msg)
   1658
   1659

ValueError: Data cardinality is ambiguous:
  x sizes: 28929, 28930, 28929
  y sizes: 28930
Make sure all arrays contain the same number of samples.
```

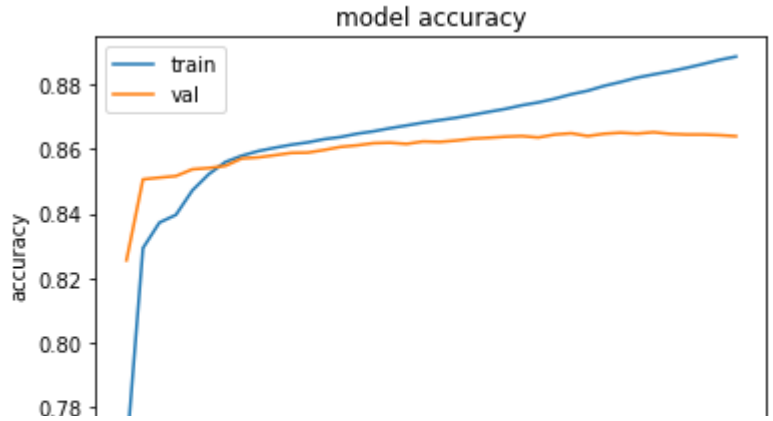
```
import keras
from matplotlib import pyplot as plt
#history = model1.fit(train_x, train_y,validation_split = 0.1, epochs=50, batch_size=4)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



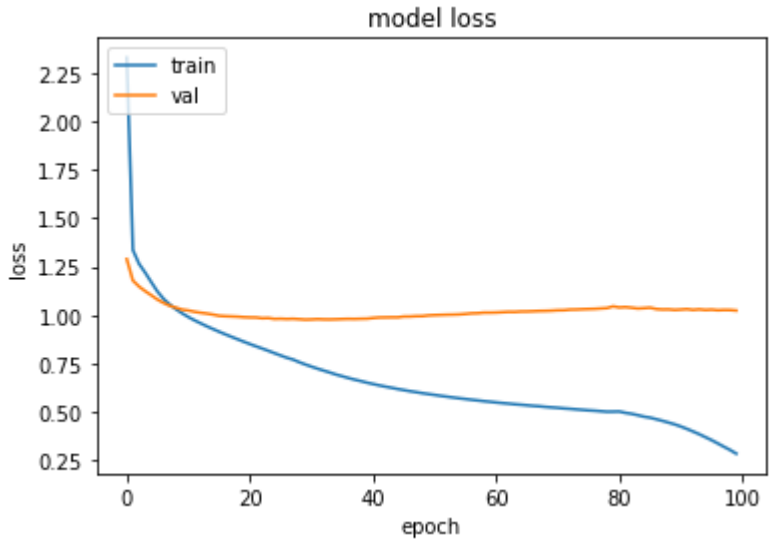

```
import keras
from matplotlib import pyplot as plt
#history = model1.fit(train_x, train_y,validation_split = 0.1, epochs=50, batch_size=4)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



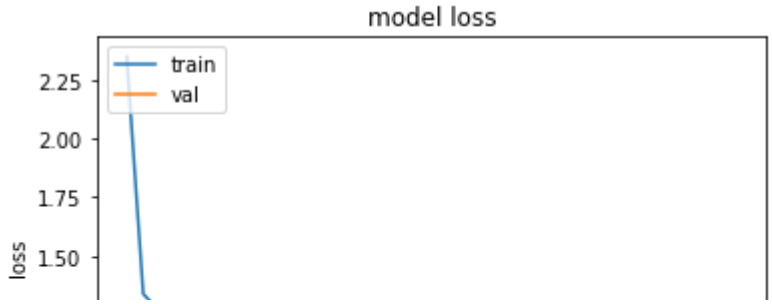

```
import keras
from matplotlib import pyplot as plt
#history = model1.fit(train_x, train_y,validation_split = 0.1, epochs=50, batch_size=4)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
model.save_weights("/content/drive/My Drive/Main/modelfull.h5")
```

```
reverse_target_word_index=y_tokens.index_word
```

```
reverse_source_word_index=x_tokens.index_word
```

```
target_word_index=y_tokens.word_index
```

```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=[encoder_inputs,img_inputs],outputs=[encoder_outputs, state_h, s
encoder_model.summary()
```

```
Model: "model_8"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_13 (InputLayer)          [(None, 24)]         0           []

 embedding_4 (Embedding)        (None, 24, 512)      2851840     ['input_13[0][0]']

 input_12 (InputLayer)          [(None, 4096)]       0           []

 lstm_8 (LSTM)                  [(None, 24, 512),    2099200     ['embedding_4[0][0]']
                                 (None, 512),
                                 (None, 512)]

 dense_4 (Dense)                (None, 512)          2097664     ['input_12[0][0]']

 lstm_9 (LSTM)                  [(None, 24, 512),    2099200     ['lstm_8[0][0]']
                                 (None, 512),
                                 (None, 512)]

 repeat_vector_2 (RepeatVector)  (None, 24, 512)     0           ['dense_4[0][0]']

 concatenate_2 (Concatenate)    (None, 24, 1024)     0           ['lstm_9[0][0]',
                                                                  'repeat_vector_2[0][0]
```

```
 lstm_10 (LSTM)              [(None, 24, 512),    3147776     ['concatenate_2[0][0]']
                              (None, 512),
                              (None, 512)]

==================================================================================================
Total params: 12,295,680
Trainable params: 12,295,680
Non-trainable params: 0
__________________________________________________________________________________________________
```

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

input_13	InputLayer	input:	[(None, 24)]
		output:	[(None, 24)]

```
# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(latent_dim,))
decoder_state_input_c = Input(shape=(latent_dim,))
decoder_hidden_state_input = Input(shape=(trmaxlen_english,latent_dim))
print(decoder_inputs.get_shape)
#print(dec_emb.get_shape)
```

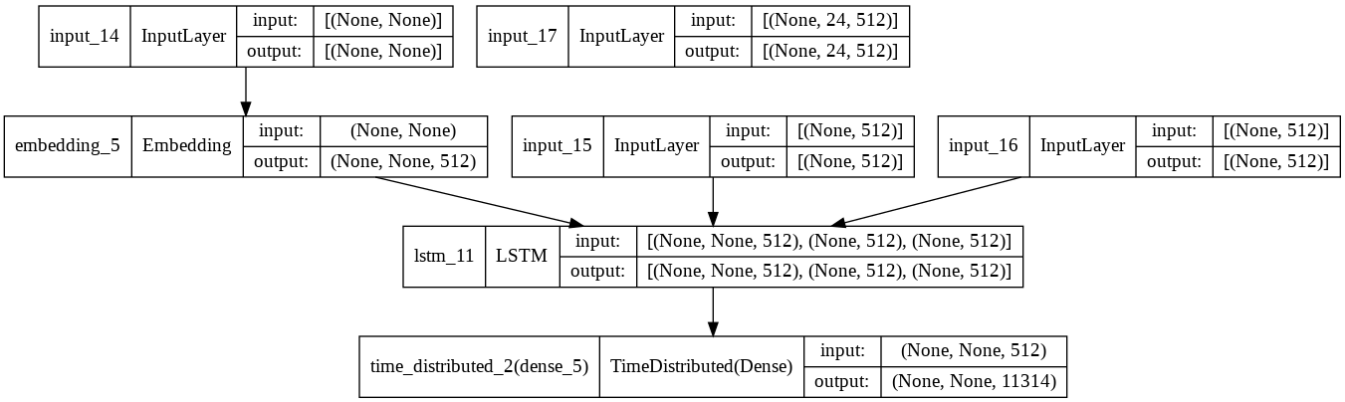
```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None) dtype=float32 (c
```

```
# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the pre
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=[decoder_state_in

# A dense softmax layer to generate prob dist. over the target vocabulary
decoder_outputs2 = decoder_dense(decoder_outputs2)

# Final decoder model
decoder_model = Model(
    [decoder_inputs] + [decoder_hidden_state_input,decoder_state_input_h, decoder_state_input
    [decoder_outputs2] + [state_h2, state_c2])
decoder_model.summary()
```

```
Model: "model_9"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_14 (InputLayer)          [(None, None)]       0           []

 embedding_5 (Embedding)        (None, None, 512)    5792768     ['input_14[0][0]']

 input_15 (InputLayer)          [(None, 512)]        0           []

 input_16 (InputLayer)          [(None, 512)]        0           []

 lstm_11 (LSTM)                 [(None, None, 512),  2099200     ['embedding_5[1][0]',
                                 (None, 512),                     'input_15[0][0]',
                                 (None, 512)]                     'input_16[0][0]']

 input_17 (InputLayer)          [(None, 24, 512)]    0           []

 time_distributed_2 (TimeDistri  (None, None, 11314)  5804082    ['lstm_11[1][0]']
 buted)

==========================================================================================
Total params: 13,696,050
Trainable params: 13,696,050
```

```
Non-trainable params: 0
_________________________________________________________________________________________________
```

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




```
def decode_sequence(input_seq,img):
    img=img[np.newaxis,:]
    # Encode the input as state vectors.
    e_out, e_h, e_c = encoder_model.predict([input_seq,img])

    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['sos']

    stop_condition = False
    decoded_sentence = ''
    while not stop_condition:

        output_tokens, h, c = decoder_model.predict([target_seq] + [e_out, e_h, e_c])
```

```
        # Sample a token
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        sampled_token = reverse_target_word_index[sampled_token_index]

        if(sampled_token!='eos'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eos'  or len(decoded_sentence.split()) >= (trmaxlen_malayalam -
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        e_h, e_c = h, c

    return decoded_sentence


def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['sos']) and i!=target_word_index['eos']):
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString




for i in range(5):
    print("Review:",seq2text(x_val[i]))
    print("Original summary:",seq2summary(y_val[i]))
    print("Predicted summary:",decode_sequence(x_val[i].reshape(1,trmaxlen_english),vgg_val[i
    print("\n")
```

```
Review: a cloudy blue sky
Original summary: തെളിഞ്ഞ നീലാകാശം
Predicted summary:  ഒരു ബാഗ് കോട്ടൺ മിഠായി


Review: window of a building
Original summary: ഒരു കെട്ടിടത്തിന്റെ വിൻഡോ
Predicted summary:  ഒരു കെട്ടിടത്തിലെ വലിയ വിൻഡോ
```

```
Review: a dark grey computer desk with drawers
Original summary: ഇരുണ്ട ചാരനിറത്തിലുള്ള കമ്പ്യൂട്ടർ ഡെസ്ക്
Predicted summary:  ഒരു മേശപ്പുറത്ത് ഒരു ലാപ്ടോപ്പ് കമ്പ്യൂട്ടർ


Review: four legged steel chair on ground
Original summary: നിലത്ത് നാല് കാലുകളുള്ള ഉരുക്ക് കസേര
Predicted summary:  ഒരു കെട്ടിടത്തിലെ വലിയ വിൻഡോ


Review: man riding a bicycle down a street
Original summary: തെരുവിൽ സൈക്കിൾ ചവിട്ടുന്ന മനുഷ്യൻ
Predicted summary:  മനുഷ്യൻ ഒരു ലോംഗ്ബോർഡിൽ
```

```
i=4
print("Review:",seq2text(x_tr[i]))
#print("Original summary:",seq2summary(y_tr[i]))
print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,trmaxlen_english),vgg_train_[i])
#plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: photo album open on an adults lap
Predicted summary:  ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പ
```

```
for i in range(5):
    print("Review:",seq2text(x_tr[i]))
    print("Original summary:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,trmaxlen_english),vgg_train_
    print("\n")
```

```
Review: male surfer surfing in still in the ocean
Original summary: ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
Predicted summary:  വെള്ളത്തിൽ ചെറിയ അലകൾ


Review: it is an indoor scene
Original summary: ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
Predicted summary:  ഇത് ഒരു വ്യക്തിയാണ്


Review: computer screens turned on
Original summary: കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി
Predicted summary:  കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി


Review: man has short hair
Original summary: മനുഷ്യന് ചെറിയ മുടിയുണ്ട്
Predicted summary:  മനുഷ്യൻ ആനയെ അലങ്കരിക്കുന്നു
```

```
Review: photo album open on an adults lap
Original summary: ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
Predicted summary:  ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പ
```

```
for i in range(10):
    print("Review:",seq2text(x_tr[i]))
    print("Original summary:",seq2summary(y_tr[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,trmaxlen_english),vgg_train_
    print("\n")
```

```
Review: male surfer surfing in still in the ocean
Original summary: ശാന്തമായ കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ
Predicted summary:  വെള്ളത്തിൽ ചെറിയ അലകൾ


Review: it is an indoor scene
Original summary: ഇത് ഒരു ഇൻഡോർ രംഗമാണ്
Predicted summary:  ഇത് ഒരു വ്യക്തിയാണ്


Review: computer screens turned on
Original summary: കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി
Predicted summary:  കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി


Review: man has short hair
Original summary: മനുഷ്യന് ചെറിയ മുടിയുണ്ട്
Predicted summary:  മനുഷ്യൻ ആനയെ അലങ്കരിക്കുന്നു


Review: photo album open on an adults lap
Original summary: ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു
Predicted summary:  ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പ


Review: there is a group of girls beside the black car
Original summary: കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട്
Predicted summary:  ഒരു കെട്ടിടത്തിലെ വലിയ വിൻഡോ


Review: child in a stroller
Original summary: ഒരു ഉന്തുവണ്ടിയിലെ കുട്ടി
Predicted summary:  ഒരു കെട്ടിടത്തിലെ വലിയ വിൻഡോ


Review: tall metal lightpost
Original summary: ഉയരമുള്ള മെറ്റൽ ലൈറ്റ്പാസ്റ്റ്
Predicted summary:  ഉയരമുള്ള ജിറാഫ്
```

```
Review: wall is painted white
Original summary: മതിൽ വെളുത്ത ചായം പൂശി
Predicted summary:  ചുവരിൽ വെളുത്ത പെയിന്റ്


Review: there are several pictures on the wall
Original summary: ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പുക
Predicted summary:  മൂന്ന് പേർ മേശയ്ക്കു ചുറ്റും ഇരിക്കുന്നു
```

```
x_tt[1]
```

```
'knife block sitting on counter with knives in it'
```

```
x_tr[2]
```

```
array([ 146, 1864,  506,    3,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0], dtype=int32)
```

```
for i in range(1):
    print("Review:",seq2text(x_tt[i]))
    print("Original summary:",seq2summary(y_tt[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,trmaxlen_english),testvgg_fe
    print("\n")
```

```
Review: hanging red palm tennis
Original summary: കളിക്കാരൻ പച്ച പിച്ചർ കെട്ടിടത്തിലെ
Predicted summary:  മേശപ്പുറത്ത് ഒരു പ്ലേറ്റ്
```

```
#error due to probelm in test data
for i in range(10):
    print("Review:",seq2text(x_tt[i]))
    print("Original summary:",seq2summary(y_tt[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,trmaxlen_english),testvgg_fe
    print("\n")
```

```
Review: hanging red palm tennis
Original summary: കളിക്കാരൻ പച്ച പിച്ചർ കെട്ടിടത്തിലെ
Predicted summary:  മേശപ്പുറത്ത് ഒരു പ്ലേറ്റ്


Review: bowl red plane on plaid green bathroom white ball
Original summary: കഴുത്ത് മുറിയിലെ രംഗം ഘടികാരം പച്ച
Predicted summary:  ഇതൊരു ഓഫീസ് കസേരയാണ്


Review: the with frame white the tarmac
```

```
Original summary: ഘടിപ്പിച്ചിരിക്കുന്നു വെളുത്ത കുതിരകൾ
Predicted summary:  ഉയരമുള്ള ടവറിൽ ക്ലോക്ക് ഫെയ്സ്


Review: coat print luggage hand yellow with cake
Original summary: സർഫ് അകലെ സോസ് വ്യക്തി പെയിന്റ് ഗ്ലാസ്
Predicted summary:  ചാരനിറത്തിലുള്ള ജാക്കറ്റ് ധരിച്ച മനുഷ്യൻ


Review: door nose green object room with plate
Original summary: പയ്യൻ സ്ക്വീസ് തവിട്ടുനിറത്തിലുള്ള തെരുവിൽ വ്യക്തി ഫോട്ടോ
Predicted summary:  ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പ


Review: colorful on a with boat of a vehicle
Original summary: ആൻഡ് വെളുത്ത മേശപ്പുറത്ത് പിന്നിൽ
Predicted summary:  ഒരു കെട്ടിടത്തിലെ വലിയ വിൻഡോ


Review: by on a baseball is yellow a grey of a dark
Original summary: ചിത്രം കളിക്കുന്ന ഇരുണ്ട അലങ്കരിച്ച ലോഗോ വിൻഡോ
Predicted summary:  ഒരു കട്ടിലിൽ തലയിണ


Review: the blue mouth bench this on clock of a baseball is
Original summary: ഇഷ്ടിക ഉപയോഗിച്ച് ജിറാഫ് ഒരു സ്ത്രീ തറ നിന്ന് ചിഹ്നം
Predicted summary:  മേശപ്പുറത്ത് വ്യക്തമായ ഗ്ലാസ്


Review: the sign stripes is
Original summary: ഒരു ബാഗ് ടെലിവിഷൻ ചുവന്ന
Predicted summary:  ചുമരിൽ തൂക്കിയിട്ടിരിക്കുന്ന സ്കൈ ലൈനിന്റെ പെയിന്റിംഗ്


Review: a side this umbrella yellow and a bus
Original summary: ഭാഗം ചുവപ്പും നടക്കുന്നു എഞ്ചിൻ
Predicted summary:  ബസിന്റെ മുൻ ടയർ
```

```
i=25
print("Review:",seq2text(x_val[i]))
print("Original summary:",seq2summary(y_val[i]))
print("Predicted summary:",decode_sequence(x_val[i].reshape(1,trmaxlen_english),vgg_val[i]))
#plt.imshow(imagedata[i].astype(np.float32))
```

```
Review: metal pole is arched
Original summary: വളഞ്ഞു നില്ക്കുന്ന മെറ്റൽ പോൾ
Predicted summary:  കെട്ടിടത്തിന്റെ വശത്ത് ഉണക്കുക
```

```
!pip install sacrebleu
```

```
import sacrebleu
import random
```

```
Requirement already satisfied: sacrebleu in /usr/local/lib/python3.7/dist-packages (2.0
Requirement already satisfied: colorama in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: tabulate>=0.8.9 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from sac
Requirement already satisfied: portalocker in /usr/local/lib/python3.7/dist-packages (fr
```

```
temp_o=[]
temp_p=[]
for i in range(50):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,trmaxlen_english),vgg_train_[s]))

bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
35.60890683230081
```

```
temp_o=[]
temp_p=[]
for i in range(1000):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,trmaxlen_english),vgg_train_[s]))


bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
24.718587015299352
```

```
temp_o=[]
temp_p=[]
for i in range(1000):
    s=random.randint(0,len(y_tt)-1)
    temp_o.append(seq2summary(y_tt[s]))
    temp_p.append(decode_sequence(x_tt[s].reshape(1,trmaxlen_english),testvgg_feature[s]))



bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
0.04524625603529401
```

English_Malayalam_Multimodal_Machine_Translation with attention

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
import cv2
import matplotlib.image as mp
```

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

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

```
with open('/content/drive/My Drive/Main/train.mn.txt') as file:
    train_mal_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/train.en.txt') as file:
    train_eng_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/train_images.txt') as file:
    train_images = file.read().split('\n')
```

```
with open('/content/drive/My Drive/Main/maldev.txt') as file:
    dev_mal_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/endev.txt') as file:
    dev_eng_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/devimage.txt') as file:
    dev_images = file.read().split('\n')
```

```
with open('/content/drive/My Drive/Main/maltest.txt') as file:
   test_mal_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/engtest.txt') as file:
   test_eng_txt = file.read().split('\n')
with open('/content/drive/My Drive/Main/testimages.txt') as file:
    test_images = file.read().split('\n')
```

```
def remove(mal_txt,eng_txt):
        mal_txt.pop()
        mal_txt.pop()
        eng_txt.pop()
        eng_txt.pop()
        #trainimages.pop()
        #trainimages,link
        print(len(mal_txt))
        print(len(eng_txt))
```

```
        #print(len(trainimages))
        #img_path=[]
        #for s in trainimages:
                #img_path.append(link+s)
        #return mal_txt,eng_txt,train_images,img_path
        return mal_txt,eng_txt


##training image
#link="/content/drive/My Drive/Main/trainimages/train/"
train_mal_txt,train_eng_txt=remove(train_mal_txt,train_eng_txt)
```

```
28930
28931
```

```
testvgg_feature=np.load('/content/drive/My Drive/Main/testfeature.npy', encoding='bytes')


#dev image
#link="/content/drive/My Drive/Main/trainimages/train/"
dev_mal_txt,dev_eng_txt=remove(dev_mal_txt,dev_eng_txt)
```

```
997
998
```

```
test_mal_txt,test_eng_txt=remove(test_mal_txt,test_eng_txt)
```

```
1399
1399
```

```
ttmal_df = pd.DataFrame(test_mal_txt, columns=['Malayalam'])
tteng_df = pd.DataFrame(test_eng_txt, columns=['English'])


ttmal_text1 = ttmal_df["Malayalam"].apply(clean_text)
tteng_text1 = tteng_df["English"].apply(clean_text)
ttmal_text2 = list(ttmal_text1.values)
tteng_text2 = list(tteng_text1.values)


x_tt=tteng_text2
y_tt=ttmal_text2


x_tt[1]
```

```
'knife block sitting on counter with knives in it'
```

```
y_tt[1]
```

```
'sos കത്തികൊണ്ട് കൗണ്ടറിൽ ഇരിക്കുന്ന കത്തി ബ്ലോക്ക് eos'
```

```
ttmal_temp=[]
for s in ttmal_text2:
    tttemp="sos "+s+" eos"
    ttmal_temp.append(tttemp)
#text2=[]
ttmal_text2=ttmal_temp
ttmal_text2[1:10]
```

```
['sos കത്തികൊണ്ട് കൗണ്ടറിൽ ഇരിക്കുന്ന കത്തി ബ്ലോക്ക് eos',
 'sos ചട്ടിയിൽ രണ്ടാമത്തെ പിസ്സ eos',
 'sos ബീജ് സ്റ്റെയർ വേ രണ്ടാം ലെവലിലേക്ക് പോകുന്നു eos',
 'sos ടാൻ ഹൗസിലെ ഇളം നിറമുള്ള രണ്ടാം നില eos',
 'sos കെട്ടിടങ്ങളുടെ രണ്ടാമത്തെ നിലയിലെ ബാൽക്കണി eos',
 'sos രാത്രി സ്റ്റാന്റിലെ വിളക്ക് കട്ടിലിന്റെ ഇടതുവശത്ത് നിൽക്കുന്നു eos',
 'sos നൈറ്റ് സ്റ്റാൻഡിന് മുകളിൽ ഒരു വെളുത്ത അലാറം ക്ലോക്ക് ഉണ്ട് eos',
 'sos ഒരു കറുത്ത സംഗീത സ്റ്റാൻഡ് eos',
 'sos തെരുവ് മുറിച്ചുകടക്കാൻ കാത്തിരിക്കുന്ന സ്ത്രീ eos']
```

```
trmal_df = pd.DataFrame(train_mal_txt, columns=['Malayalam'])
treng_df = pd.DataFrame(train_eng_txt, columns=['English'])
```

```
#Datacleaning by removing special characters
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r" '", "", text)
    text = re.sub(r" '", "", text)
    text = re.sub(r"-", " ", text)
    text = re.sub(r"<5>", "5", text)
    text = re.sub(r"‘ ", "", text)
    text = re.sub(r" ’", "", text)
    text = re.sub(r"[+\.\!\/_,$%^*(+\"\']+|[+—！ ，  〈〉 《》 。 |？ ？、 . %~@#￥%……&* () ’]", "", t
    text=text.rstrip()
    return text
```

```
trmal_text1 = trmal_df["Malayalam"].apply(clean_text)
treng_text1 = treng_df["English"].apply(clean_text)
trmal_text2 = list(trmal_text1.values)
treng_text2 = list(treng_text1.values)
```

```
trmal_text2[0]
```

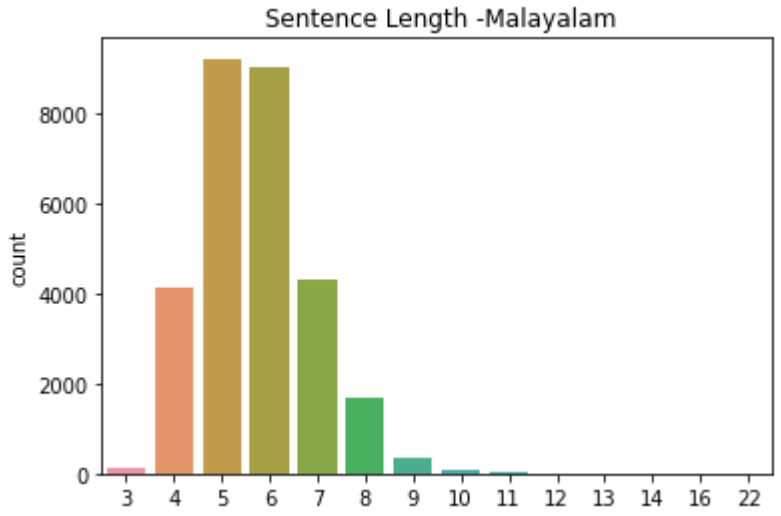
```
'ശാന്തമായ  കടലിൽ സർഫിങ് നടത്തുന്ന പുരുഷ സർഫർ'
```

```
trmal_temp=[]
for s in trmal_text2:
    trtemp="sos "+s+" eos"
    trmal_temp.append(trtemp)
#text2=[]
trmal_text2=trmal_temp
trmal_text2[1:10]
```

```
['sos ഇത് ഒരു ഇൻഡോർ രംഗമാണ് eos',
 'sos കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാക്കി eos',
 'sos മനുഷ്യന് ചെറിയ മുടിയുണ്ട് eos',
 'sos ഫോട്ടോ ആൽബം മുതിർന്നവരുടെ മടിയിൽ തുറക്കുന്നു eos',
 'sos കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട് eos',
 'sos ഒരു ഉന്തുവണ്ടിയിലെ കുട്ടി eos',
 'sos ഉയരമുള്ള മെറ്റൽ ലൈറ്റ്പോസ്റ്റ് eos',
 'sos മതിൽ വെളുത്ത ചായം പൂശി eos',
 'sos ചാരനിറത്തിലുള്ള റോഡിന്റെ വശങ്ങളിൽ പച്ച പുല്ലിന്റെ സ്ട്രിപ്പുകൾ eos']
```

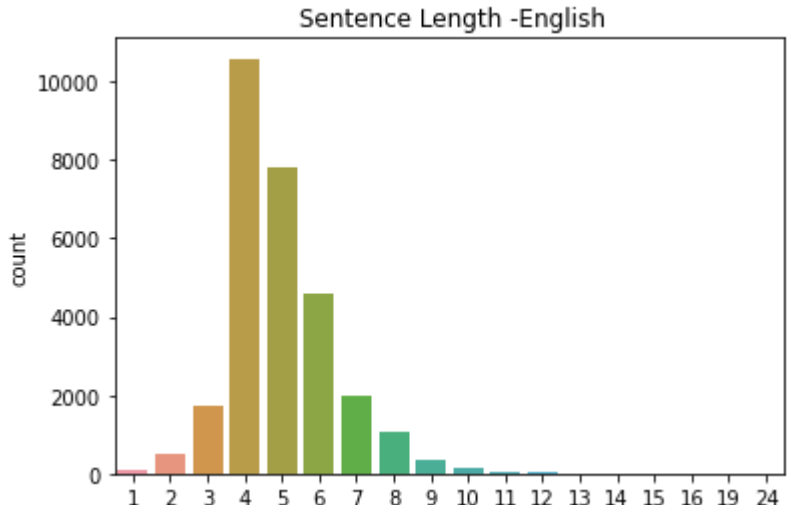
```
import seaborn as sn
import matplotlib.pyplot as plt
trmalayalam_words = []
for i in trmal_text2:
    trmalayalam_words.append(len(i.split()))
sn.countplot(trmalayalam_words).set(title=' Sentence Length -Malayalam')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
trenglish_words = []
for j in treng_text2:
    trenglish_words.append(len(j.split()))
sn.countplot(trenglish_words).set(title=' Sentence Length -English')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
trmaxlen_malayalam = max(trmalayalam_words)
trmaxlen_english = max(trenglish_words)
print('Maximum sentence length-Malayalam :',trmaxlen_malayalam)
print('Maximum sentence length-English :',trmaxlen_english)
```

```
Maximum sentence length-Malayalam : 22
Maximum sentence length-English : 24
```

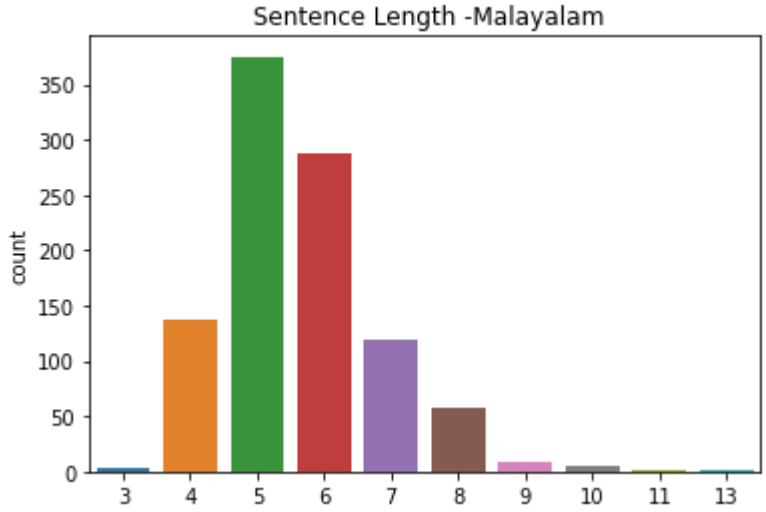
```
#dev

#Datacleaning by removing special characters
import re
def clean_text(text):
    text = text.lower()
    text = re.sub(r" ‘", "", text)
    text = re.sub(r" ‘", "", text)
    text = re.sub(r"-", " ", text)
    text = re.sub(r"<5>", "5", text)
    text = re.sub(r"“ ", "", text)
    text = re.sub(r" ”", "", text)
    text = re.sub(r"[+\.\!\/_,$%^*(+\"\']+|[+——！ ，  〈〉 《》 。 ｜？ ?、 . %~@#￥%……&* () ’]", "", t
    text=text.rstrip()
    return text


dmal_df = pd.DataFrame(dev_mal_txt, columns=['Malayalam'])
deng_df = pd.DataFrame(dev_eng_txt, columns=['English'])


dmal_text1 = dmal_df["Malayalam"].apply(clean_text)
deng_text1 = deng_df["English"].apply(clean_text)
dmal_text2 = list(dmal_text1.values)
deng_text2 = list(deng_text1.values)
```

```
dmal_temp=[]
for s in dmal_text2:
    dtemp="sos "+s+" eos"
    dmal_temp.append(dtemp)
#text2=[]
dmal_text2=dmal_temp
dmal_text2[1:10]
```

```
['sos ഒരു കെട്ടിടത്തിന്റെ വിൻഡോ eos',
 'sos ഡ്രോയറുകളുള്ള ഇരുണ്ട ചാരനിറത്തിലുള്ള കമ്പ്യൂട്ടർ ഡെസ്ക് eos',
 'sos നിലത്ത് നാല് കാലുകളുള്ള ഉരുക്ക് കസേര eos',
 'sos തെരുവിൽ സൈക്കിൾ ചവിട്ടുന്ന മനുഷ്യൻ eos',
 'sos ഈ കാറുകൾ അഴുക്കിൽ പാർക്ക് ചെയ്തിരിക്കുന്നു eos',
 'sos ആകാശത്ത് വെളുത്ത മേഘങ്ങൾ eos',
 'sos ഗ്ലാസ് വാതിലുകളുടെ ഗണം eos',
 'sos പ്രൊജക്ടർ സ്ക്രീൻ ചുരുട്ടി വെച്ചിരിക്കുന്നു eos',
 'sos ചാരനിറത്തിലുള്ള സ്വെറ്റർ ധരിച്ച മനുഷ്യൻ eos']
```

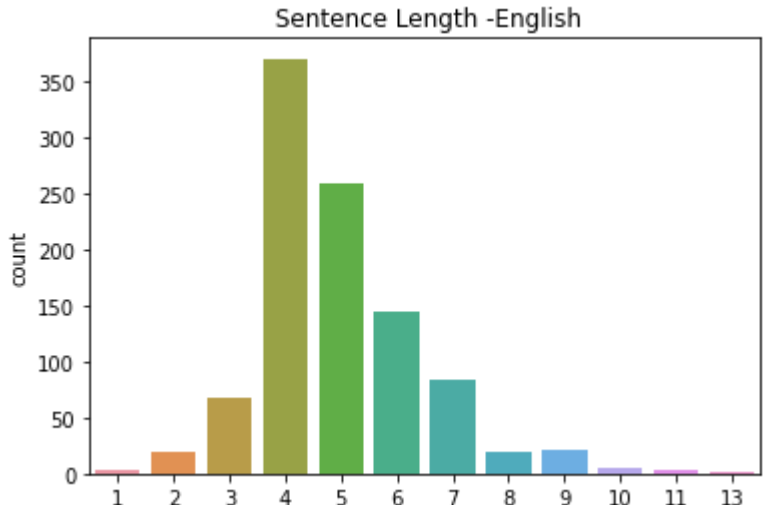
```
import seaborn as sn
import matplotlib.pyplot as plt
dmalayalam_words = []
for i in dmal_text2:
    dmalayalam_words.append(len(i.split()))
sn.countplot(dmalayalam_words).set(title=' Sentence Length -Malayalam')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
denglish_words = []
for j in deng_text2:
    denglish_words.append(len(j.split()))
sn.countplot(denglish_words).set(title=' Sentence Length -English')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
dmaxlen_malayalam = max(dmalayalam_words)
dmaxlen_english = max(denglish_words)
print('Maximum sentence length-Malayalam :',dmaxlen_malayalam)
print('Maximum sentence length-English :',dmaxlen_english)
```

```
Maximum sentence length-Malayalam : 13
Maximum sentence length-English : 13
```

```
ttmal_df = pd.DataFrame(test_mal_txt, columns=['Malayalam'])
tteng_df = pd.DataFrame(test_eng_txt, columns=['English'])
```

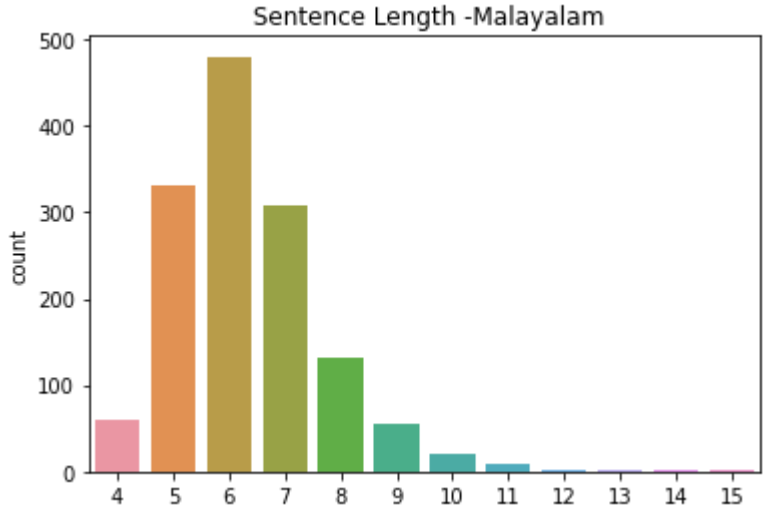
```
ttmal_text1 = ttmal_df["Malayalam"].apply(clean_text)
tteng_text1 = tteng_df["English"].apply(clean_text)
ttmal_text2 = list(ttmal_text1.values)
tteng_text2 = list(tteng_text1.values)
```

```
ttmal_temp=[]
for s in ttmal_text2:
    tttemp="sos "+s+" eos"
    ttmal_temp.append(tttemp)
#text2=[]
ttmal_text2=ttmal_temp
ttmal_text2[1:10]
```

```
['sos കത്തികൊണ്ട് കൗണ്ടറിൽ ഇരിക്കുന്ന കത്തി ബ്ലോക്ക് eos',
 'sos ചട്ടിയിൽ രണ്ടാമത്തെ പിസ്സ eos',
 'sos ബീജ് സ്റ്റെയർ വേ രണ്ടാം ലെവലിലേക്ക് പോകുന്നു eos',
 'sos ടാൻ ഹൗസിലെ ഇളം നിറമുള്ള രണ്ടാം നില eos',
 'sos കെട്ടിടങ്ങളുടെ രണ്ടാമത്തെ നിലയിലെ ബാൽക്കണി eos',
 'sos രാത്രി സ്റ്റാന്റിലെ വിളക്ക് കട്ടിലിന്റെ ഇടതുവശത്ത് നിൽക്കുന്നു eos',
 'sos നൈറ്റ് സ്റ്റാൻഡിന് മുകളിൽ ഒരു വെളുത്ത അലാറം ക്ലോക്ക് ഉണ്ട് eos',
 'sos ഒരു കറുത്ത സംഗീത സ്റ്റാൻഡ് eos',
 'sos തെരുവ് മുറിച്ചുകടക്കാൻ കാത്തിരിക്കുന്ന സ്ത്രീ eos']
```

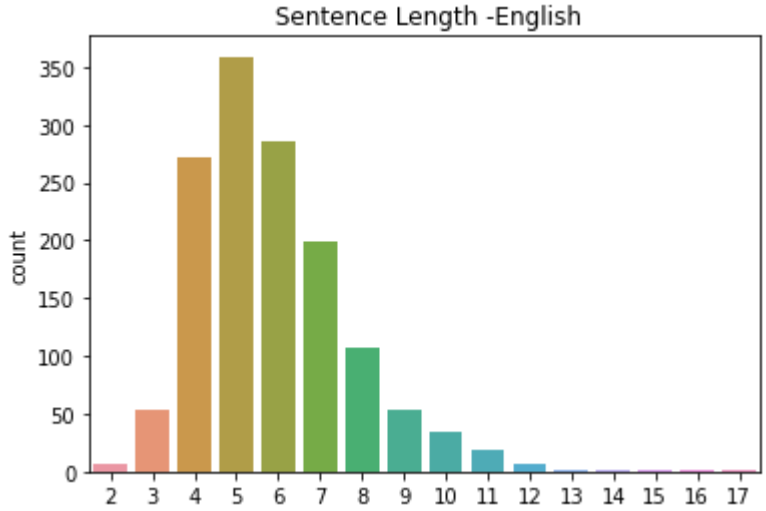
```
import seaborn as sn
import matplotlib.pyplot as plt
ttmalayalam_words = []
for i in ttmal_text2:
    ttmalayalam_words.append(len(i.split()))
sn.countplot(ttmalayalam_words).set(title=' Sentence Length -Malayalam')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```




```
ttenglish_words = []
for j in tteng_text2:
    ttenglish_words.append(len(j.split()))
sn.countplot(ttenglish_words).set(title=' Sentence Length -English')
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass th
  FutureWarning
```



```
ttmaxlen_malayalam = max(ttmalayalam_words)
ttmaxlen_english = max(ttenglish_words)
print('Maximum sentence length-Malayalam :',ttmaxlen_malayalam)
print('Maximum sentence length-English :',ttmaxlen_english)
```

```
Maximum sentence length-Malayalam : 15
Maximum sentence length-English : 17
```

```
x_tr=treng_text2
y_tr=trmal_text2
x_val=deng_text2
y_val=dmal_text2
```

```
#Tokening the sentences using Keras tokenizer -Malayalam data
from keras.preprocessing.text import Tokenizer
x_tokens = Tokenizer()
x_tokens.fit_on_texts(x_tr)
x_tr = x_tokens.texts_to_sequences(x_tr)
x_val = x_tokens.texts_to_sequences(x_val)
print('x_tr:',x_tr)
print('x_val:',x_val)
```

```
x_tr: [[432, 237, 187, 6, 1418, 6, 2, 173], [145, 5, 29, 2227, 252], [146, 1864, 506, 3]
x_val: [[1, 561, 13, 24], [50, 4, 1, 28], [1, 113, 106, 146, 164, 11, 1754], [212, 1871,
```

```
#padding with post (appending zeros at the end to equalize sentence length)
from keras.preprocessing.sequence import pad_sequences
x_tr = pad_sequences(x_tr,maxlen = trmaxlen_english,padding = 'post')
x_val = pad_sequences(x_val,maxlen = trmaxlen_english,padding = 'post')

# +1 for padding
x_voc_size   =  len(x_tokens.word_index) +1
print("No of unique words in English",x_voc_size)
```

```
No of unique words in English 5570
```

```
#Tokening the sentences using Keras tokenizer -Malayalam data
from keras.preprocessing.text import Tokenizer
y_tokens = Tokenizer()
y_tokens.fit_on_texts(y_tr)
y_tr = y_tokens.texts_to_sequences(y_tr)
y_val = y_tokens.texts_to_sequences(y_val)
print('x_tr:',y_tr)
print('x_val:',y_val)
```

```
x_tr: [[1, 974, 1376, 4856, 1158, 672, 287, 2], [1, 62, 3, 3335, 3336, 2], [1, 147, 2631
x_val: [[1, 343, 293, 2], [1, 3, 111, 31, 2], [1, 129, 12, 147, 1388, 2], [1, 45, 206, 3
```

```
#padding with post (appending zeros at the end to equalize sentence length)
from keras.preprocessing.sequence import pad_sequences
y_tr = pad_sequences(y_tr,maxlen = trmaxlen_malayalam,padding = 'post')
y_val = pad_sequences(y_val,maxlen = trmaxlen_malayalam,padding = 'post')

# +1 for padding
y_voc_size   =  len(y_tokens.word_index) +1
print("No of unique words in English",y_voc_size)
```

```
     No of unique words in English 11314
```

```
x_voc_size   =  len(x_tokens.word_index) +1
print("No of unique words in English",x_voc_size)
```

```
     No of unique words in English 5570
```

```
y_voc_size   =  len(y_tokens.word_index) +1
print("No of unique words in English",y_voc_size)
```

```
     No of unique words in English 11314
```

```
from keras.preprocessing.text import Tokenizer
y_tokens = Tokenizer()
y_tokens.fit_on_texts(y_tt)
y_tt = y_tokens.texts_to_sequences(y_tt)


y_tt = pad_sequences(y_tt,maxlen = trmaxlen_english,padding = 'post')
```

```
pip install keras-applications
```

```
     Collecting keras-applications
       Downloading Keras_Applications-1.0.8-py3-none-any.whl (50 kB)
          |████████████████████████████████| 50 kB 4.7 MB/s
     Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from kera
     Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-packages (f
     Requirement already satisfied: cached-property in /usr/local/lib/python3.7/dist-packages
     Installing collected packages: keras-applications
     Successfully installed keras-applications-1.0.8
```

```
import pandas as pd
import pickle
import numpy as np
import os
import keras
import tensorflow
```

```
from keras_applications.resnet import ResNet50
from tensorflow.keras.optimizers import Adam
from keras.layers import Dense, GlobalAveragePooling2D,BatchNormalization,Flatten,Input, Conv
from keras.models import Sequential, Model
from keras.utils import np_utils
import random
from keras.preprocessing import image, sequence
import matplotlib.pyplot as plt
import keras
from keras import backend as K
import gensim
from numpy import *
import numpy as np
import pandas as pd
import re
from tensorflow.keras.applications.vgg16 import VGG16
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from nltk.corpus import stopwords
from tensorflow.keras.layers import Input, LSTM, Embedding, Dense, Concatenate, TimeDistribut
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping
import warnings
```

```
trvgg_feature=np.load('/content/drive/My Drive/Main/combine.npy', encoding='bytes')
```

```
dvgg_feature=np.load('/content/drive/My Drive/Main/valfeature.npy', encoding='bytes')
```

```
print(len(trvgg_feature))
print(len(x_tr))
print(len(y_tr))
```

```
28929
28931
28930
```

```
print(len(trvgg_feature))
x_tr=x_tr[:-2]
print(len(x_tr))
y_tr=y_tr[:-1]
print(len(y_tr))
```

```
28929
28929
28929
```

```
print(len(dvgg_feature))
print(len(x_val))
print(len(y_val))
```

```
997
997
997
```

```
y_tr=y_tr[:-1]
print(len(y_tr))
```

```
28927
```

```
dvgg_feature=dvgg_feature[:-1]
print(len(dvgg_feature))
```

```
997
```

```
x_val=x_val[:-1]
print(len(x_val))
```

```
997
```

```
y_val=z[:-1]
print(len(y_val))
```

```
998
```

```
#y_tr=y_tr[:-2]
print(len(y_val))
```

```
999
```

```
print(len(dvgg_feature))
print(len(x_val))
print(len(y_val))
```

```
998
998
997
```

```
p=dvgg_feature
q=x_val
z=y_val
```

```
#splitting image pixels for training and validation
vgg_train_=trvgg_feature
vgg_val=dvgg_feature
```

```
#Generating a repeat vector from image pixels
img_inputs=Input(shape=(4096,))
d_1=Dense(512, activation='relu')(img_inputs)
r_1=RepeatVector(trmaxlen_english)(d_1)
vf_model = Model(img_inputs, r_1)
vf_model.summary()
```

```
Model: "model_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_8 (InputLayer)        [(None, 4096)]            0

 dense_4 (Dense)             (None, 512)               2097664

 repeat_vector_3 (RepeatVect  (None, 24, 512)          0
 or)

=================================================================
Total params: 2,097,664
Trainable params: 2,097,664
Non-trainable params: 0
_________________________________________________________________
```

```
#Model
x_voc=x_voc_size
y_voc=y_voc_size
latent_dim = 512
embedding_dim=512
#Encoder
encoder_inputs = Input(shape=(trmaxlen_english,))
#The model will take as input an integer matrix of size (batch,input_length)and the largest i
enc_emb =  Embedding(x_voc, 1024,trainable=True)(encoder_inputs)
print(encoder_inputs.get_shape)
print(enc_emb.get_shape)
```

```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, 24) dtype=float32 (cre
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, 24, 1024) dtype=float3
```

```
#encoder LSTM Layer 1#merge_mode='sum'
encoder_lstm1 = Bidirectional(LSTM(256,return_sequences=True,return_state=True,dropout=0.4,re
#The dimension of each state equals to the LSTM unit number
encoder_output1= encoder_lstm1(enc_emb)
print(encoder_lstm1.output_shape)
```

```
WARNING:tensorflow:Layer lstm_13 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_13 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_13 will not use cuDNN kernels since it doesn't meet the cr
[(None, 24, 512), (None, 256), (None, 256), (None, 256), (None, 256)]
```

```
#LSTM layer 2
encoder_lstm2 = keras.layers.wrappers.Bidirectional(LSTM(256,return_sequences=True,return_sta
encoder_output2 = encoder_lstm2(encoder_output1)
print(encoder_lstm2.output_shape)
```

```
WARNING:tensorflow:Layer lstm_14 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_14 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_14 will not use cuDNN kernels since it doesn't meet the cr
[(None, 24, 512), (None, 256), (None, 256), (None, 256), (None, 256)]
```

```
type(encoder_output2)
```

```
list
```

```
encoder_output=encoder_output2[0]
encoder_output
```

```
<KerasTensor: shape=(None, 24, 512) dtype=float32 (created by layer 'bidirectional_7')>
```

```
img_inputs=Input(shape=(4096,))
d_1=Dense(512, activation='relu')(img_inputs)
r_1=RepeatVector(trmaxlen_english)(d_1)
vf_model = Model(img_inputs, r_1)
vf_model.summary()
```

```
Model: "model_16"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_13 (InputLayer)       [(None, 4096)]            0

 dense_10 (Dense)            (None, 512)               2097664

 repeat_vector_6 (RepeatVect  (None, 24, 512)          0
 or)

=================================================================
Total params: 2,097,664
Trainable params: 2,097,664
Non-trainable params: 0
_________________________________________________________________
```

```
#Concatenating image features with text input
encoder_output2=Concatenate(axis=-1)([encoder_output,r_1])


#layer 3
encoder_lstm3= Bidirectional(LSTM(256, return_state=True, return_sequences=True,dropout=0.4,r
encoder_outputs, forward_h, forward_c, backward_h, backward_c= encoder_lstm3(encoder_output2)
encoder_states = [forward_h, forward_c, backward_h, backward_c]
```

```
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it doesn't meet the cr
WARNING:tensorflow:Layer lstm_15 will not use cuDNN kernels since it doesn't meet the cr
```

```
state_h = Concatenate()([forward_h, backward_h])
state_c = Concatenate()([forward_c, backward_c])


encoder_states =[state_h, state_c]


#Decoder
# Set up the decoder, using `encoder_states` as initial state.
decoder_inputs = Input(shape=(None,))
#embedding layer
dec_emb_layer = Embedding(y_voc, 1024,trainable=True)
dec_emb = dec_emb_layer(decoder_inputs)
print(decoder_inputs.get_shape)
print(dec_emb.get_shape)
```

```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None) dtype=float32 (c
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None, 1024) dtype=floa
```

```
#Decoder LSTM layer1
decoder_lstm = LSTM(512, return_sequences=True, return_state=True,dropout=0.4,recurrent_dropo
decoder_outputs,decoder_fwd_state, decoder_back_state = decoder_lstm(dec_emb,initial_state=en
print(decoder_lstm.output_shape)
```

```
WARNING:tensorflow:Layer lstm_16 will not use cuDNN kernels since it doesn't meet the cr
[(None, None, 512), (None, 512), (None, 512)]
```

```
import tensorflow as tf
import os
from tensorflow.keras.layers import Layer
from tensorflow.keras import backend as K
import numpy as np
from matplotlib import pyplot as plt
from tqdm import tqdm
```

```
import keras
from keras import backend as K
from keras import activations, initializers, regularizers, constraints, metrics
from keras.datasets import cifar10
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential, Model
from keras.layers import (Dense, Dropout, Activation, Flatten, Reshape, Layer,
                          BatchNormalization, LocallyConnected2D,
                          ZeroPadding2D, Conv2D, MaxPooling2D, Conv2DTranspose,
                          GaussianNoise, UpSampling2D, Input)
#from keras.utils import conv_utils, multi_gpu_model
from keras.layers import Lambda
#from keras.engine import Layer, InputSpec
#from keras.legacy import interfaces

class AttentionLayer(Layer):
    """
    This class implements Bahdanau attention (https://arxiv.org/pdf/1409.0473.pdf).
    There are three sets of weights introduced W_a, U_a, and V_a
     """

    def __init__(self, **kwargs):
        super(AttentionLayer, self).__init__(**kwargs)

    def build(self, input_shape):
        assert isinstance(input_shape, list)
        # Create a trainable weight variable for this layer.

        self.W_a = self.add_weight(name='W_a',
                                   shape=tf.TensorShape((input_shape[0][2], input_shape[0][2]
                                   initializer='uniform',
                                   trainable=True)
        self.U_a = self.add_weight(name='U_a',
                                   shape=tf.TensorShape((input_shape[1][2], input_shape[0][2]
                                   initializer='uniform',
                                   trainable=True)
        self.V_a = self.add_weight(name='V_a',
                                   shape=tf.TensorShape((input_shape[0][2], 1)),
                                   initializer='uniform',
                                   trainable=True)

        super(AttentionLayer, self).build(input_shape)  # Be sure to call this at the end

    def call(self, inputs, verbose=False):
        """
        inputs: [encoder_output_sequence, decoder_output_sequence]
        """
        assert type(inputs) == list
        encoder_out_seq, decoder_out_seq = inputs
        if verbose:
            print('encoder_out_seq>', encoder_out_seq.shape)
```

```
        print('decoder_out_seq>', decoder_out_seq.shape)

    def energy_step(inputs, states):
        """ Step function for computing energy for a single decoder state
        inputs: (batchsize * 1 * de_in_dim)
        states: (batchsize * 1 * de_latent_dim)
        """

        assert_msg = "States must be an iterable. Got {} of type {}".format(states, type(
        assert isinstance(states, list) or isinstance(states, tuple), assert_msg

        """ Some parameters required for shaping tensors"""
        en_seq_len, en_hidden = encoder_out_seq.shape[1], encoder_out_seq.shape[2]
        de_hidden = inputs.shape[-1]

        """ Computing S.Wa where S=[s0, s1, ..., si]"""
        # <= batch size * en_seq_len * latent_dim
        W_a_dot_s = K.dot(encoder_out_seq, self.W_a)

        """ Computing hj.Ua """
        U_a_dot_h = K.expand_dims(K.dot(inputs, self.U_a), 1)  # <= batch_size, 1, latent
        if verbose:
            print('Ua.h>', U_a_dot_h.shape)

        """ tanh(S.Wa + hj.Ua) """
        # <= batch_size*en_seq_len, latent_dim
        Ws_plus_Uh = K.tanh(W_a_dot_s + U_a_dot_h)
        if verbose:
            print('Ws+Uh>', Ws_plus_Uh.shape)

        """ softmax(va.tanh(S.Wa + hj.Ua)) """
        # <= batch_size, en_seq_len
        e_i = K.squeeze(K.dot(Ws_plus_Uh, self.V_a), axis=-1)
        # <= batch_size, en_seq_len
        e_i = K.softmax(e_i)

        if verbose:
            print('ei>', e_i.shape)

        return e_i, [e_i]

    def context_step(inputs, states):
        """ Step function for computing ci using ei """

        assert_msg = "States must be an iterable. Got {} of type {}".format(states, type(
        assert isinstance(states, list) or isinstance(states, tuple), assert_msg

        # <= batch_size, hidden_size
        c_i = K.sum(encoder_out_seq * K.expand_dims(inputs, -1), axis=1)
        if verbose:
            print('ci>', c_i.shape)
```

```
            return c_i, [c_i]

        fake_state_c = K.sum(encoder_out_seq, axis=1)
        fake_state_e = K.sum(encoder_out_seq, axis=2)  # <= (batch_size, enc_seq_len, latent_

        """ Computing energy outputs """
        # e_outputs => (batch_size, de_seq_len, en_seq_len)
        last_out, e_outputs, _ = K.rnn(
            energy_step, decoder_out_seq, [fake_state_e],
        )

        """ Computing context vectors """
        last_out, c_outputs, _ = K.rnn(
            context_step, e_outputs, [fake_state_c],
        )

        return c_outputs, e_outputs

    def compute_output_shape(self, input_shape):
        """ Outputs produced by the layer """
        return [
            tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[1][2])),
            tf.TensorShape((input_shape[1][0], input_shape[1][1], input_shape[0][1]))
        ]


attn_layer = AttentionLayer(name='attention_layer')
attn_out, attn_states = attn_layer([encoder_outputs, decoder_outputs])


attn_out
```

```
<KerasTensor: shape=(None, None, 512) dtype=float32 (created by layer 'attention_layer')
```

```
decoder_concat_input = Concatenate(axis=-1, name='concat_layer')([decoder_outputs, attn_out])


decoder_concat_input
```

```
<KerasTensor: shape=(None, None, 1024) dtype=float32 (created by layer 'concat_layer')>
```

```
#dense = Dense(fr_vsize, activation='softmax', name='softmax_layer')
#dense_time = TimeDistributed(dense, name='time_distributed_layer')
decoder_dense =  TimeDistributed(Dense(y_voc, activation='softmax'))
decoder_outputs = decoder_dense(decoder_concat_input)
#decoder_output = dense_time(decoder_concat_input)
print(decoder_dense.output_shape)
```

```
(None, None, 11314)
```

```
model = Model([encoder_inputs,decoder_inputs,img_inputs], decoder_outputs)
model.summary()
```

```
input_12 (InputLayer)          [(None, 24)]         0           []

embedding_4 (Embedding)        (None, 24, 1024)     5703680     ['input_12[0][0]']

input_13 (InputLayer)          [(None, 4096)]       0           []

bidirectional_6 (Bidirectional  [(None, 24, 512),   2623488     ['embedding_4[0][0]'
)                               (None, 256),
                                (None, 256),
                                (None, 256),
                                (None, 256)]

dense_10 (Dense)               (None, 512)          2097664     ['input_13[0][0]']

bidirectional_7 (Bidirectional  [(None, 24, 512),   1574912     ['bidirectional_6[0]
)                               (None, 256),                     'bidirectional_6[0]
                                (None, 256),                     'bidirectional_6[0]
                                (None, 256),                     'bidirectional_6[0]
                                (None, 256)]                     'bidirectional_6[0]

repeat_vector_6 (RepeatVector)  (None, 24, 512)     0           ['dense_10[0][0]']

concatenate_15 (Concatenate)   (None, 24, 1024)     0           ['bidirectional_7[0]
                                                                 'repeat_vector_6[0]

input_5 (InputLayer)           [(None, None)]       0           []

encoder_lstm_3 (Bidirectional)  [(None, 24, 512),   2623488     ['concatenate_15[0][
                                (None, 256),
                                (None, 256),
                                (None, 256),
                                (None, 256)]

embedding_1 (Embedding)        (None, None, 1024)   11585536    ['input_5[0][0]']

concatenate_16 (Concatenate)   (None, 512)          0           ['encoder_lstm_3[0][
                                                                 'encoder_lstm_3[0][

concatenate_17 (Concatenate)   (None, 512)          0           ['encoder_lstm_3[0][
                                                                 'encoder_lstm_3[0][

lstm_16 (LSTM)                 [(None, None, 512),  3147776     ['embedding_1[0][0]'
                                (None, 512),                     'concatenate_16[0][
                                (None, 512)]                     'concatenate_17[0][

attention_layer (AttentionLaye  ((None, None, 512),  524800     ['encoder_lstm_3[0][
r)                              (None, None, 24))                'lstm_16[0][0]']

concat_layer (Concatenate)     (None, None, 1024)   0           ['lstm_16[0][0]',

                                                                 'attention_layer[0]

time_distributed_3 (TimeDistri  (None, None, 11314)  11596850   ['concat_layer[0][0]
buted)
```

```
==================================================================================================
Total params: 41,478,194
Trainable params: 41,478,194
Non-trainable params: 0
```

```
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',metrics=['accuracy'])
history=model.fit([x_tr,y_tr[:,:-1],vgg_train_], y_tr.reshape(y_tr.shape[0],y_tr.shape[1], 1)
```

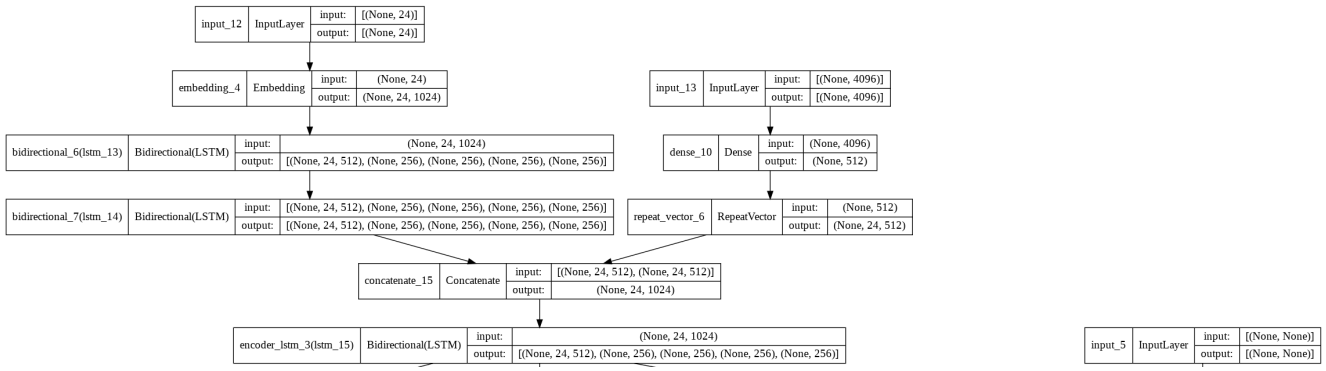
```
Epoch 1/100
57/57 [==============================] - 67s 884ms/step - loss: 1.8472 - accuracy: 0.
Epoch 2/100
57/57 [==============================] - 49s 857ms/step - loss: 1.2564 - accuracy: 0.
Epoch 3/100
57/57 [==============================] - 49s 855ms/step - loss: 1.1806 - accuracy: 0.
Epoch 4/100
57/57 [==============================] - 49s 854ms/step - loss: 1.1118 - accuracy: 0.
Epoch 5/100
57/57 [==============================] - 49s 855ms/step - loss: 1.0672 - accuracy: 0.
Epoch 6/100
57/57 [==============================] - 49s 854ms/step - loss: 1.0316 - accuracy: 0.
Epoch 7/100
57/57 [==============================] - 48s 850ms/step - loss: 0.9983 - accuracy: 0.
Epoch 8/100
57/57 [==============================] - 49s 853ms/step - loss: 0.9658 - accuracy: 0.
Epoch 9/100
57/57 [==============================] - 49s 854ms/step - loss: 0.9355 - accuracy: 0.
Epoch 10/100
57/57 [==============================] - 48s 849ms/step - loss: 0.9056 - accuracy: 0.
Epoch 11/100
57/57 [==============================] - 48s 848ms/step - loss: 0.8768 - accuracy: 0.
Epoch 12/100
57/57 [==============================] - 48s 845ms/step - loss: 0.8482 - accuracy: 0.
Epoch 13/100
57/57 [==============================] - 48s 846ms/step - loss: 0.8198 - accuracy: 0.
Epoch 14/100
57/57 [==============================] - 48s 847ms/step - loss: 0.7902 - accuracy: 0.
Epoch 15/100
57/57 [==============================] - 48s 848ms/step - loss: 0.7595 - accuracy: 0.
Epoch 16/100
57/57 [==============================] - 48s 850ms/step - loss: 0.7292 - accuracy: 0.
Epoch 17/100
57/57 [==============================] - 49s 851ms/step - loss: 0.6989 - accuracy: 0.
Epoch 18/100
57/57 [==============================] - 48s 846ms/step - loss: 0.6639 - accuracy: 0.
Epoch 19/100
57/57 [==============================] - 48s 846ms/step - loss: 0.6245 - accuracy: 0.
Epoch 20/100
57/57 [==============================] - 48s 847ms/step - loss: 0.5793 - accuracy: 0.
Epoch 21/100
57/57 [==============================] - 48s 846ms/step - loss: 0.5277 - accuracy: 0.
Epoch 22/100
57/57 [==============================] - 48s 847ms/step - loss: 0.4740 - accuracy: 0.
Epoch 23/100
57/57 [==============================] - 48s 848ms/step - loss: 0.4211 - accuracy: 0.
Epoch 24/100
```

```
57/57 [==============================] - 48s 848ms/step - loss: 0.3714 - accuracy: 0.
Epoch 25/100
57/57 [==============================] - 48s 849ms/step - loss: 0.3249 - accuracy: 0.
Epoch 26/100
57/57 [==============================] - 48s 847ms/step - loss: 0.2816 - accuracy: 0.
Epoch 27/100
57/57 [==============================] - 48s 849ms/step - loss: 0.2432 - accuracy: 0.
Epoch 28/100
57/57 [==============================] - 48s 849ms/step - loss: 0.2087 - accuracy: 0.
Epoch 29/100
57/57 [==============================] - 48s 849ms/step - loss: 0.1785 - accuracy: 0.
```

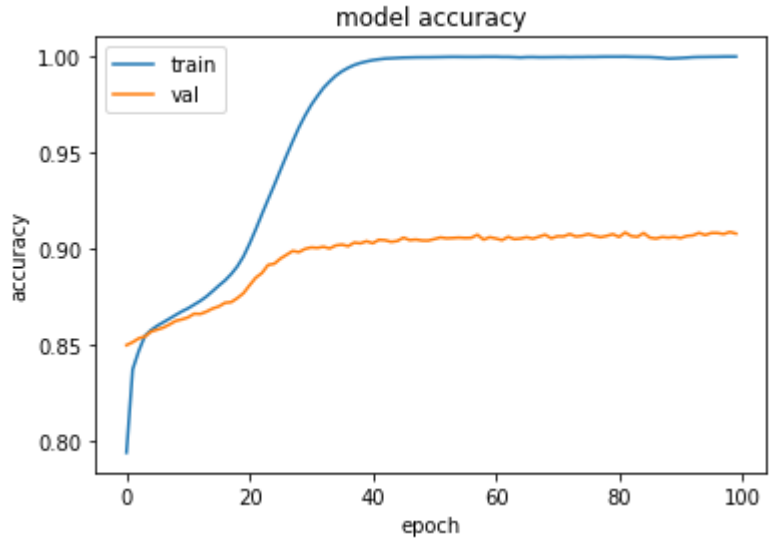
```
model.save_weights("/content/drive/My Drive/Main/bidemodel2.h5")
```

```
from keras.utils.vis_utils import plot_model
import tensorflow as tf
```

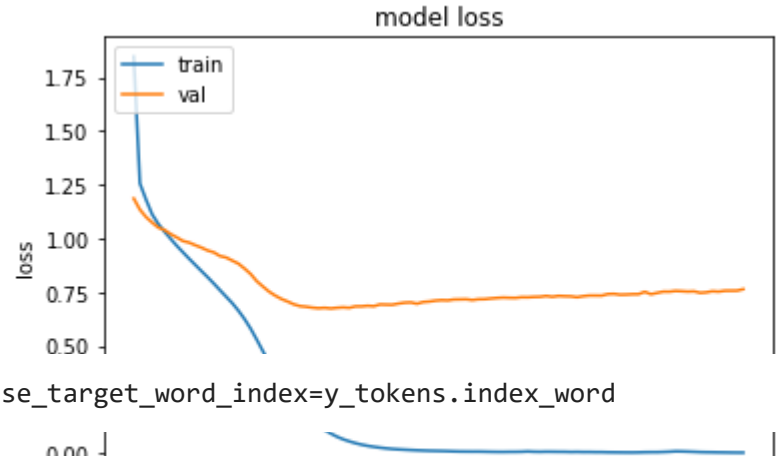
```
tf.keras.utils.plot_model(
    model,
    to_file='model.png',
    show_shapes=True,
    show_layer_names=True,
    rankdir='TB'
)
```




```
import keras
from matplotlib import pyplot as plt
#history = model1.fit(train_x, train_y,validation_split = 0.1, epochs=50, batch_size=4)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```




```
reverse_target_word_index=y_tokens.index_word
```

```
reverse_source_word_index=x_tokens.index_word
```

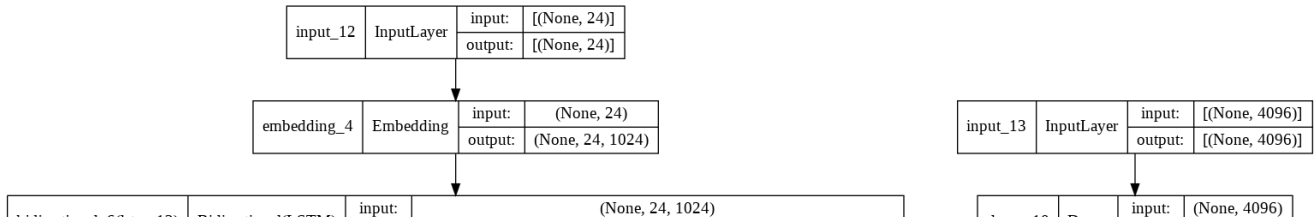
```
target_word_index=y_tokens.word_index
```

```
# Encode the input sequence to get the feature vector
encoder_model = Model(inputs=[encoder_inputs,img_inputs],outputs=[encoder_outputs,state_h,sta
encoder_model.summary()
```

```
Model: "model_18"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_12 (InputLayer)          [(None, 24)]         0           []

 embedding_4 (Embedding)        (None, 24, 1024)     5703680     ['input_12[0][0]']

 input_13 (InputLayer)          [(None, 4096)]       0           []

 bidirectional_6 (Bidirectional  [(None, 24, 512),   2623488     ['embedding_4[0][0]']
 )                               (None, 256),
                                 (None, 256),
                                 (None, 256),
                                 (None, 256)]

 dense_10 (Dense)               (None, 512)          2097664     ['input_13[0][0]']

 bidirectional_7 (Bidirectional  [(None, 24, 512),   1574912     ['bidirectional_6[0][0]
 )                               (None, 256),                     'bidirectional_6[0][1]
                                 (None, 256),                     'bidirectional_6[0][2]
                                 (None, 256),                     'bidirectional_6[0][3]
                                 (None, 256)]                     'bidirectional_6[0][4]

 repeat_vector_6 (RepeatVector)  (None, 24, 512)     0           ['dense_10[0][0]']

 concatenate_15 (Concatenate)   (None, 24, 1024)     0           ['bidirectional_7[0][0]
                                                                  'repeat_vector_6[0][0]

 encoder_lstm_3 (Bidirectional)  [(None, 24, 512),   2623488     ['concatenate_15[0][0]
```

```
                                    (None, 256),
                                    (None, 256),
                                    (None, 256),
                                    (None, 256)]

 concatenate_16 (Concatenate)   (None, 512)          0           ['encoder_lstm_3[0][1]
                                                                  'encoder_lstm_3[0][3]

 concatenate_17 (Concatenate)   (None, 512)          0           ['encoder_lstm_3[0][2]
                                                                  'encoder_lstm_3[0][4]

==================================================================================================
Total params: 14,623,232
Trainable params: 14,623,232
Non-trainable params: 0
__________________________________________________________________________________________________
```

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




```
# Decoder setup
# Below tensors will hold the states of the previous time step
decoder_state_input_h = Input(shape=(512,))
decoder_state_input_c = Input(shape=(512,))
encoder_inf_states=Input(shape=(trmaxlen_english,512,))
decoder_hidden_state_input = Input(shape=(trmaxlen_english,512))
print(decoder_inputs.get_shape)
#print(dec_emb.get_shape)
```

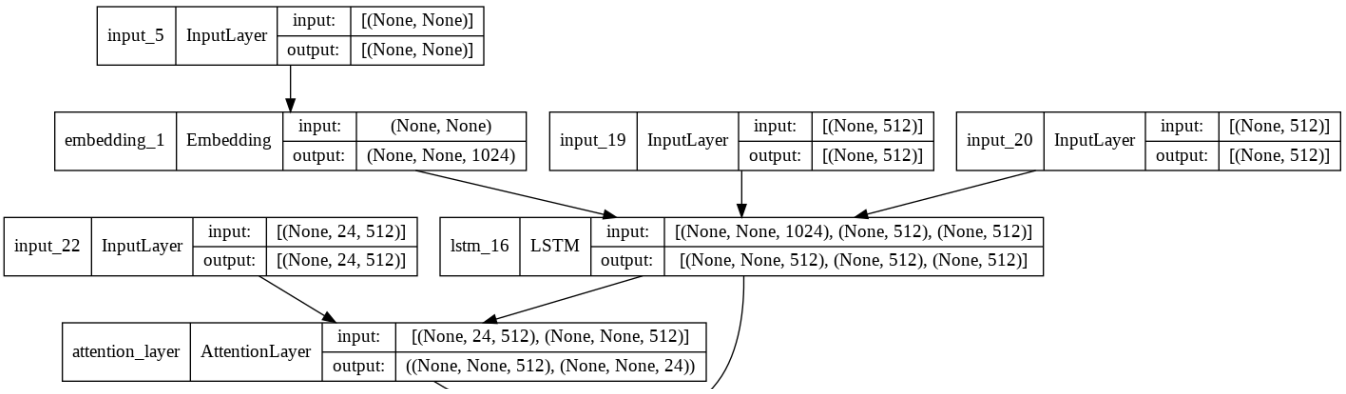
```
<bound method KerasTensor.get_shape of <KerasTensor: shape=(None, None) dtype=float32 (c
```

```
dec_states = [decoder_state_input_h, decoder_state_input_c]



# Get the embeddings of the decoder sequence
dec_emb2= dec_emb_layer(decoder_inputs)
# To predict the next word in the sequence, set the initial states to the states from the pre
decoder_outputs2, state_h2, state_c2 = decoder_lstm(dec_emb2, initial_state=dec_states)
attn_inf_out, attn_inf_states = attn_layer([decoder_hidden_state_input, decoder_outputs2])
decoder_inf_concat = Concatenate(axis=-1, name='concat')([decoder_outputs2, attn_inf_out])



dec_states2= [state_h2, state_c2]

decoder_outputs2 = decoder_dense(decoder_inf_concat)


decoder_model= Model(
                    [decoder_inputs] + [decoder_hidden_state_input, decoder_state_input_h, de
                     [decoder_outputs2]+ dec_states2)


decoder_model.summary()
```

```
Model: "model_19"
__________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==========================================================================================
 input_5 (InputLayer)           [(None, None)]       0           []

 embedding_1 (Embedding)        (None, None, 1024)   11585536    ['input_5[0][0]']

 input_19 (InputLayer)          [(None, 512)]        0           []
```

```
input_20 (InputLayer)          [(None, 512)]        0           []

lstm_16 (LSTM)                 [(None, None, 512),  3147776     ['embedding_1[1][0]',
                                (None, 512),                     'input_19[0][0]',
                                (None, 512)]                     'input_20[0][0]']

input_22 (InputLayer)          [(None, 24, 512)]    0           []

attention_layer (AttentionLaye  ((None, None, 512),  524800     ['input_22[0][0]',
r)                              (None, None, 24))                'lstm_16[1][0]']

concat (Concatenate)           (None, None, 1024)   0           ['lstm_16[1][0]',
                                                                 'attention_layer[1][0]

time_distributed_3 (TimeDistri  (None, None, 11314)  11596850   ['concat[0][0]']
buted)

==================================================================================================
Total params: 26,854,962
Trainable params: 26,854,962
Non-trainable params: 0
__________________________________________________________________________________________________
```

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




```
def decode_sequence(input_seq,img):
    img=img[np.newaxis,:]
    # Encode the input as state vectors.
    enc_output, enc_h, enc_c   = encoder_model.predict([input_seq,img])


    # Generate empty target sequence of length 1.
    target_seq = np.zeros((1,1))

    # Populate the first word of target sequence with the start word.
    target_seq[0, 0] = target_word_index['sos']

    stop_condition = False
    decoded_sentence = ''
    attention_weights=[]
    while not stop_condition:

        output_tokens, h, c  = decoder_model.predict([target_seq] + [enc_output, enc_h, enc_c
        sampled_token_index = np.argmax(output_tokens[0, -1, :])
        if sampled_token_index == 0:
            break
        else:
            sampled_token = reverse_target_word_index[sampled_token_index]
        #attention_weights.append((sampled_token_index, attention))
        if(sampled_token!='eos'):
            decoded_sentence += ' '+sampled_token

        # Exit condition: either hit max length or find stop word.
        if (sampled_token == 'eos'  or len(decoded_sentence.split()) >= (trmaxlen_malayalam -
            stop_condition = True

        # Update the target sequence (of length 1).
        target_seq = np.zeros((1,1))
        target_seq[0, 0] = sampled_token_index

        # Update internal states
        enc_h, enc_c = h, c

    return decoded_sentence
```

```
def seq2summary(input_seq):
    newString=''
    for i in input_seq:
        if((i!=0 and i!=target_word_index['sos']) and i!=target_word_index['eos']):
            newString=newString+reverse_target_word_index[i]+' '
    return newString

def seq2text(input_seq):
    newString=''
    for i in input_seq:
        if(i!=0):
            newString=newString+reverse_source_word_index[i]+' '
    return newString


from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping

checkpoint = ModelCheckpoint("/content/drive/My Drive/Main/modelfull2", monitor='val_accuracy

early_stopping = EarlyStopping(monitor='val_accuracy', patience=5)

callbacks_list = [checkpoint, early_stopping]


for i in range(1):
    #print("Review:",seq2text(x_tt[i]))
    print("Original summary:",seq2summary(y_tt[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,trmaxlen_english),testvgg_fe
    print("\n")
```

```
Original summary: മിഠായി വെളുത്ത സ്നോബോർഡിംഗ് പുല്ല്
Predicted summary:  സമുദ്രത്തിൽ തരംഗദൈർഘ്യം
```

```
#error due to probelm in test data
for i in range(10):
    #print("Review:",seq2text(x_tt[i]))
    print("Original summary:",seq2summary(y_tt[i]))
    print("Predicted summary:",decode_sequence(x_tr[i].reshape(1,trmaxlen_english),testvgg_fe
    print("\n")
```

```
Original summary: മിഠായി വെളുത്ത സ്നോബോർഡിംഗ് പുല്ല്
Predicted summary:  സമുദ്രത്തിൽ തരംഗദൈർഘ്യം


Original summary: ക്യാമറയിലേക്ക് പിച്ചർ ചെയ്യുന്നു നിറത്തിലുള്ള വെളുത്ത
Predicted summary:  ഭാഗികമായി കഴിച്ച പിസ്സ


Original summary: കഴുത്ത് ഒരു കോഫി
Predicted summary:  കമ്പ്യൂട്ടർ സ്ക്രീനുകൾ ഓണാണ്
```

```
Original summary: ധരിക്കുന്ന മുറിയിലെ ഘടിപ്പിച്ചിരിക്കുന്നു ഇതൊരു അകലെ ഭക്ഷണ
Predicted summary:  മനുഷ്യൻ ചെറിയ വണ്ടി വലിക്കുന്നു


Original summary: റാക്കറ്റ് സോസ് ഘടികാരം പാർക്ക് ഇതൊരു സ്കേറ്റ്ബോർഡിൽ
Predicted summary:  ചില ആളുകൾ ഒരു റാലിയിലാണ്


Original summary: വര ഒരു ആളുകൾ കിടക്കുന്നു
Predicted summary:  കറുത്ത കാറിനടുത്ത് ഒരു കൂട്ടം പെൺകുട്ടികളുണ്ട്


Original summary: പുല്ലിൽ പോൾ ഗ്ലാസ് പെയിന്റ് ബോട്ടിൽ വ്യക്തി
Predicted summary:  ഒരു കുട്ടിയുടെ കാൽ


Original summary: ബോൾ സർഫർ വെള്ള കറുത്ത ഷർട്ടിൽ വയലിൽ ഉള്ള
Predicted summary:  ഉയരമുള്ള മെറ്റൽ ലൈറ്റ്പോസ്റ്റ്


Original summary: ചാരനിറത്തിലുള്ള കുതിരകൾ വലിയ
Predicted summary:  മതിൽ വെളുത്ത നിറമാണ്


Original summary: ബസ് തൊപ്പി തെരുവിൽ കറുപ്പും
Predicted summary:  ആകാശത്ത് നിരവധി ഇലകളാണ്
```

```
!pip install sacrebleu
import sacrebleu
import random
```

```
Collecting sacrebleu
  Downloading sacrebleu-2.0.0-py3-none-any.whl (90 kB)
     |████████████████████████████████| 90 kB 5.8 MB/s
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (fr
Collecting colorama
  Downloading colorama-0.4.4-py2.py3-none-any.whl (16 kB)
Collecting portalocker
  Downloading portalocker-2.3.2-py2.py3-none-any.whl (15 kB)
Requirement already satisfied: tabulate>=0.8.9 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: regex in /usr/local/lib/python3.7/dist-packages (from sac
Installing collected packages: portalocker, colorama, sacrebleu
Successfully installed colorama-0.4.4 portalocker-2.3.2 sacrebleu-2.0.0
```

```
temp_o=[]
```

```
temp_p=[]
for i in range(50):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,trmaxlen_english),vgg_train_[s]))

bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
100.00000000000004
```

```
temp_o=[]
temp_p=[]
for i in range(1000):
    s=random.randint(0,len(y_tr)-1)
    temp_o.append(seq2summary(y_tr[s]))
    temp_p.append(decode_sequence(x_tr[s].reshape(1,trmaxlen_english),vgg_train_[s]))


bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
24.718587015299352
```

```
temp_o=[]
temp_p=[]
for i in range(1000):
    s=random.randint(0,len(y_tt)-1)
    temp_o.append(seq2summary(y_tt[s]))
    temp_p.append(decode_sequence(x_tt[s].reshape(1,trmaxlen_english),testvgg_feature[s]))



bleu = sacrebleu.corpus_bleu(temp_o, [temp_p],lowercase=True, tokenize='intl')
print(bleu.score)
```

```
0.08486582955725616
```