

Deep Learning course project - 2

Title : Automatic Image Caption Generation



Sl.no	Name	Roll no.
-------	------	----------

1	G. Tanmay	B210753EC
---	-----------	-----------

2	C. Nithin Bhavan	B210705EC
---	------------------	-----------

3	Ch. Karthik	B210711EC
---	-------------	-----------

Importing necessary libraries

```
In [76]: import tensorflow as tf
import keras
import sys, time, os, warnings
import numpy as np
import pandas as pd
from collections import Counter
warnings.filterwarnings("ignore")
print("python {}".format(sys.version))
print("keras version {}".format(keras.__version__)); del keras
print("tensorflow version {}".format(tf.__version__))

config = tf.compat.v1.ConfigProto()
config.gpu_options.per_process_gpu_memory_fraction = 0.95
```

```

config.gpu_options.visible_device_list = "0"

def set_seed(sd=123):
    from numpy.random import seed
    from tensorflow import set_random_seed
    import random as rn
    ## numpy random seed
    seed(sd)
    ## core python's random number
    rn.seed(sd)
    ## tensor flow's random number
    set_random_seed(sd)

python 3.11.5 (tags/v3.11.5:cce6ba9, Aug 24 2023, 14:38:34) [MSC v.1936 64 bit (AMD64)]
keras version 3.1.1
tensorflow version 2.16.1

```

Introduction

Caption generation is a complex task in the field of artificial intelligence that involves creating a written description for a given image.

In order to comprehend the content of a picture, it is necessary to utilize computer vision techniques. Additionally, a language model from the field of natural language processing is required to accurately convert the image's knowledge into coherent and properly ordered words.

For the current study, we utilised a pretrained model VGG16 to extract the features from the picture. We then employed an LSTM (Long Short-Term Memory) model to predict the captions using the retrieved features. We utilized the BLEU score for evaluation purposes.

Dataset

We will be using the Flickr8k Dataset for our project.

It contains around 8091 images and 5 captions for each image ($5 * 8091 = 40455$ captions)

Dataset Source: KAGGLE LINK <https://www.kaggle.com/shadabhussain/flickr8k>

Downloadiing and importing the dataset to the notebook

```

In [77]: dir_Flickr_jpg = "C:\\\\Users\\\\karth\\\\OneDrive\\\\Documents\\\\Desktop\\\\jp\\\\Flickr8k_Dataset"

dir_Flickr_text = "C:\\\\Users\\\\karth\\\\OneDrive\\\\Documents\\\\Desktop\\\\jp\\\\Flickr8k_text\\\\Flickr8k.token.txt"

jpgs = os.listdir(dir_Flickr_jpg)
print("The number of jpg files in Flicker8k: {}".format(len(jpgs)))

```

The number of jpg files in Flicker8k: 8091

Loading the text and adding it into pandas dataframe

Here, the data is divided into separate parts by using the newline, tab, and '#' characters. These parts are then stored in a new dataframe. Considering the computational complexity, we have utilized only the initial caption from the dataset, disregarding the remaining four captions.

```

In [78]: #Load the text data and save it into a panda dataframe df_txt.

## read in the Flickr caption data
file = open(dir_Flickr_text,'r')
text = file.read()
file.close()

datatxt = []
for line in text.split('\\n'):
    col = line.split('\\t')
    if len(col) == 1:
        continue
    w = col[0].split("#")
    datatxt.append(w + [col[1].lower()])

df_txt = pd.DataFrame(datatxt,columns=["filename","index","caption"])

uni_filenames = np.unique(df_txt.filename.values)
print("The number of unique file names : {}".format(len(uni_filenames)))
print("The distribution of the number of captions for each image:")

# Counting number of captions for each image using counter

```

```
Counter(Counter(df_txt.filename.values).values())
```

The number of unique file names : 8092

The distribution of the number of captions for each image:

```
Out[78]: Counter({5: 8092})
```

```
In [79]: df_txt[:5]
```

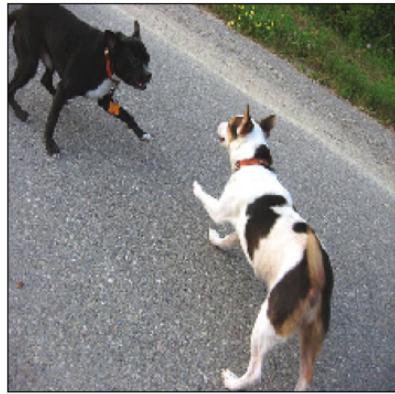
	filename	index	caption
0	1000268201_693b08cb0e.jpg	0	a child in a pink dress is climbing up a set o...
1	1000268201_693b08cb0e.jpg	1	a girl going into a wooden building .
2	1000268201_693b08cb0e.jpg	2	a little girl climbing into a wooden playhouse .
3	1000268201_693b08cb0e.jpg	3	a little girl climbing the stairs to her play...
4	1000268201_693b08cb0e.jpg	4	a little girl in a pink dress going into a woo...

```
In [80]: from keras.preprocessing.image import load_img, img_to_array  
import matplotlib.pyplot as plt
```

```
# Displaying (npic = 3) pics from the dataset  
npic = 3  
npix = 224  
target_size = (npix,npix,3)  
  
count = 1  
fig = plt.figure(figsize=(10,20))  
for jpgfnm in uni_filenames[:npic]:  
    filename = dir_Flickr_jpg + '/' + jpgfnm  
    captions = list(df_txt["caption"].loc[df_txt["filename"]==jpgfnm].values)  
    image_load = load_img(filename, target_size=target_size)  
  
    ax = fig.add_subplot(npic,2,count,xticks=[],yticks=[])  
    ax.imshow(image_load)  
    count += 1  
  
    ax = fig.add_subplot(npic,2,count)  
    plt.axis('off')  
    ax.plot()  
    ax.set_xlim(0,1)  
    ax.set_ylim(0,len(captions))  
    for i, caption in enumerate(captions):  
        ax.text(0,i,caption,fontsize=20)  
    count += 1  
plt.show()
```



a little girl in a pink dress going into a wooden cabin .
a little girl climbing the stairs to her playhouse .
a little girl climbing into a wooden playhouse .
a girl going into a wooden building .
a child in a pink dress is climbing up a set of stairs in an entry way .



two dogs on pavement moving toward each other .
two dogs of different breeds looking at each other on the road .
a black dog and a white dog with brown spots are staring at each other in the street .
a black dog and a tri-colored dog playing with each other on the road .
a black dog and a spotted dog are fighting



young girl with pigtails painting outside in the grass .
there is a girl with pigtails sitting in front of a rainbow painting .
a small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
a little girl is sitting in front of a large painted rainbow .
a little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .

Data Preparation

Here we create a new dataframe dfword to visualize distribution of the words. It contains each word and its frequency in the entire tokens in decreasing order.

```
In [6]: def df_word(df_txt):  
    vocabulary = []  
    for txt in df_txt.caption.values:  
        vocabulary.extend(txt.split())  
    print('Vocabulary Size: %d' % len(set(vocabulary)))  
    ct = Counter(vocabulary)  
    dfword = pd.DataFrame(list(ct.items()), columns=['word', 'count'])  
    dfword.sort_values(by='count', ascending=False, inplace=True)  
    dfword = dfword.reset_index()[["word", "count"]]  
    return(dfword)  
dfword = df_word(df_txt)  
dfword.head(3)
```

Vocabulary Size: 8918

```
Out[6]:   word  count  
0      a  62989  
1      .  36581  
2     in  18975
```

Visuvalizing 50 most frequently appearing words

```
In [7]: topn = 50  
  
def plthist(dfsub, title="The top 50 most frequently appearing words"):  
    plt.figure(figsize=(20,3))
```

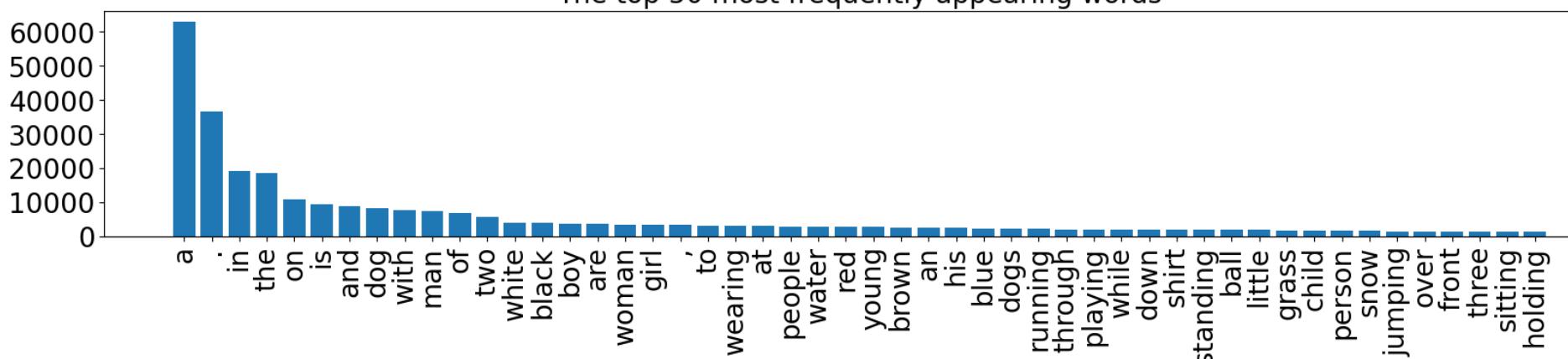
```

plt.bar(dfsub.index,dfsub["count"])
plt.yticks(fontsize=20)
plt.xticks(dfsub.index,dfsub["word"],rotation=90,fontsize=20)
plt.title(title,fontsize=20)
plt.show()

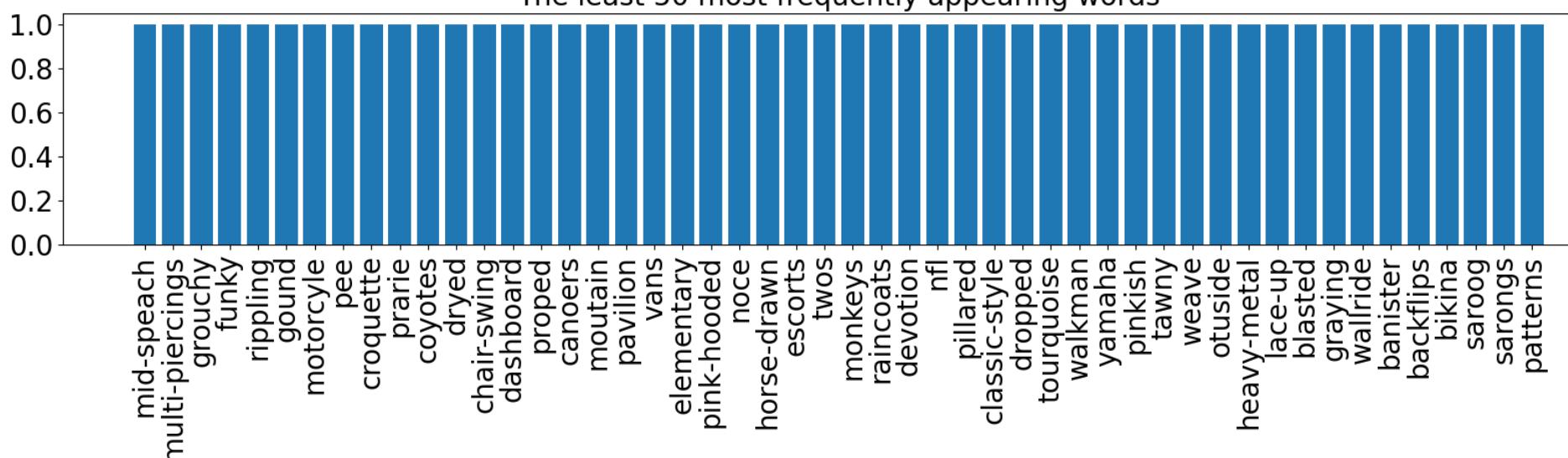
plthist(dfword.iloc[:topn,:],
        title="The top 50 most frequently appearing words")
plthist(dfword.iloc[-topn:,:],
        title="The least 50 most frequently appearing words")

```

The top 50 most frequently appearing words



The least 50 most frequently appearing words



In order to clean the caption, We will create three functions :

```

remove punctuation
remove single character
remove numeric characters

```

```

In [8]: import string
text_original = "I scored 21 in NLP, but Akshat topped. I have python v3.8. It's 4:20 pm. Could you give us 20 on 20?"

print(text_original)
print("\nRemove punctuations..")
def remove_punctuation(text_original):
    text_no_punctuation = text_original.translate(str.maketrans(' ', ' ', string.punctuation))
    return(text_no_punctuation)
text_no_punctuation = remove_punctuation(text_original)
print(text_no_punctuation)

print("\nRemove a single character word..")
def remove_single_character(text):
    text_len_more_than1 = ""
    for word in text.split():
        if len(word) > 1:
            text_len_more_than1 += " " + word
    return(text_len_more_than1)
text_len_more_than1 = remove_single_character(text_no_punctuation)
print(text_len_more_than1)

print("\nRemove words with numeric values..")
def remove_numeric(text,printTF=False):
    text_no_numeric = ""
    for word in text.split():
        isalpha = word.isalpha()
        if printTF:
            print("  {:10} : {}".format(word,isalpha))
        if isalpha:
            text_no_numeric += " " + word
    return(text_no_numeric)
text_no_numeric = remove_numeric(text_len_more_than1,printTF=True)
print(text_no_numeric)

```

I scored 21 in NLP, but Akshat topped. I have python v3.8. It's 4:20 pm. Could you give us 20 on 20?

Remove punctuations..

I scored 21 in NLP but Akshat topped I have python v38 Its 420 pm Could you give us 20 on 20

Remove a single character word..

scored 21 in NLP but Akshat topped have python v38 Its 420 pm Could you give us 20 on 20

Remove words with numeric values..

```
scored      : True
21         : False
in          : True
NLP         : True
but         : True
Akshat      : True
topped      : True
have        : True
python       : True
v38         : False
Its          : True
420         : False
pm           : True
Could       : True
you          : True
give          : True
us            : True
20            : False
on            : True
20            : False
```

scored in NLP but Akshat topped have python Its pm Could you give us on

Cleaning the captions in dataset

Using the functions defined in above cell we clean all the punctuations, Single characters and numerical characters.

```
In [9]: def text_clean(text_original):
    text = remove_punctuation(text_original)
    text = remove_single_character(text)
    text = remove_numeric(text)
    return(text)

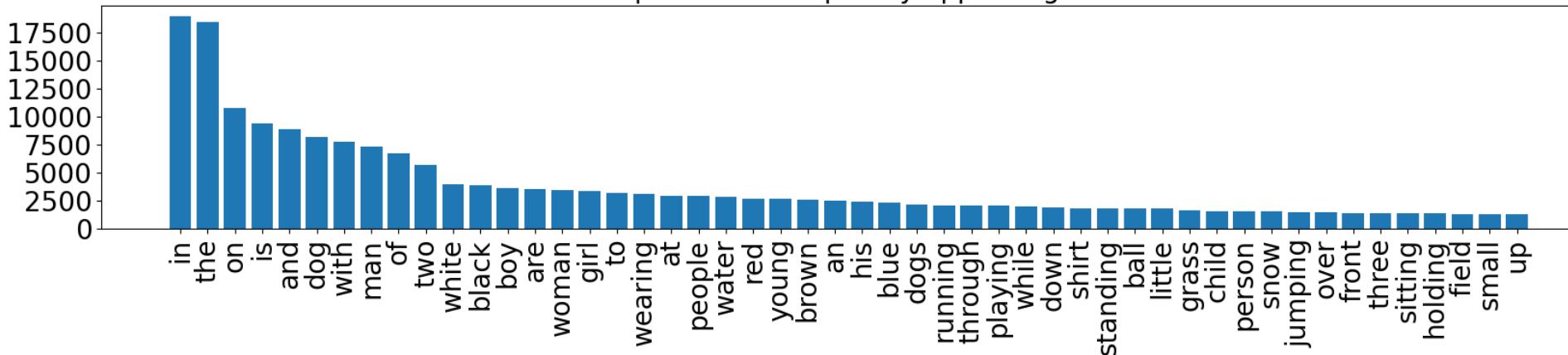
for i, caption in enumerate(df_txt.caption.values):
    newcaption = text_clean(caption)
    df_txt["caption"].iloc[i] = newcaption
```

After cleaning visualizing 50 most frequently occurring words

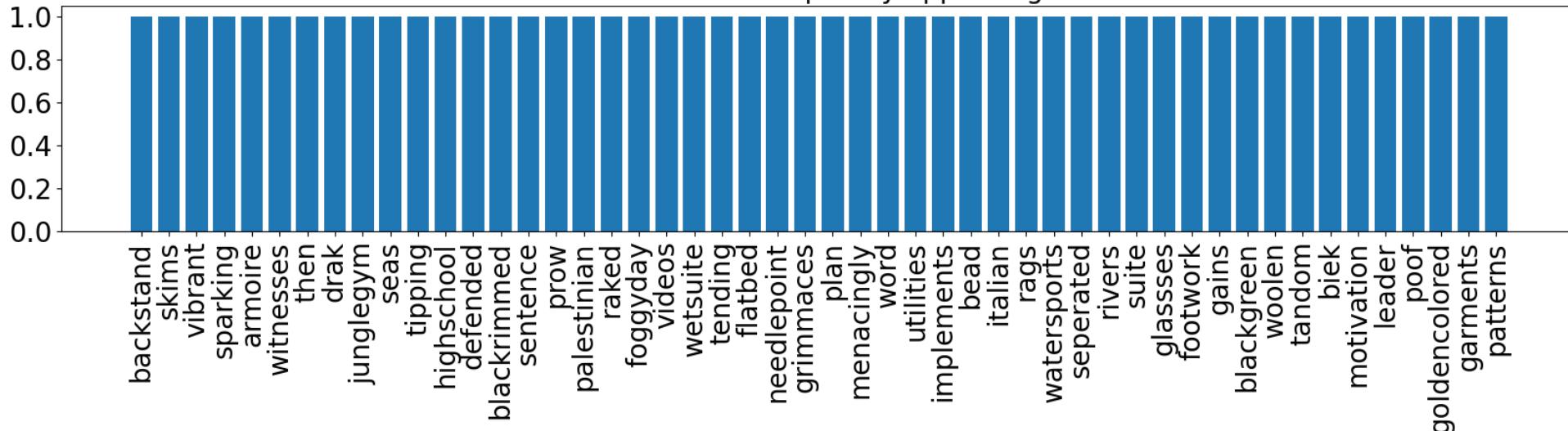
```
In [10]: dfword = df_word(df_txt)
plthist(dfword.iloc[:topn,:],
        title="The top 50 most frequently appearing words")
plthist(dfword.iloc[-topn:,:],
        title="The least 50 most frequently appearing words")
```

Vocabulary Size: 8763

The top 50 most frequently appearing words



The least 50 most frequently appearing words



Adding start and end sequences in every caption

```
In [11]: from copy import copy
def add_start_end_seq_token(captions):
    caps = []
    for txt in captions:
        txt = 'startseq ' + txt + ' endseq'
        caps.append(txt)
    return(caps)
df_txt0 = copy(df_txt)
df_txt0["caption"] = add_start_end_seq_token(df_txt["caption"])
df_txt0.head(5)
del df_txt
```

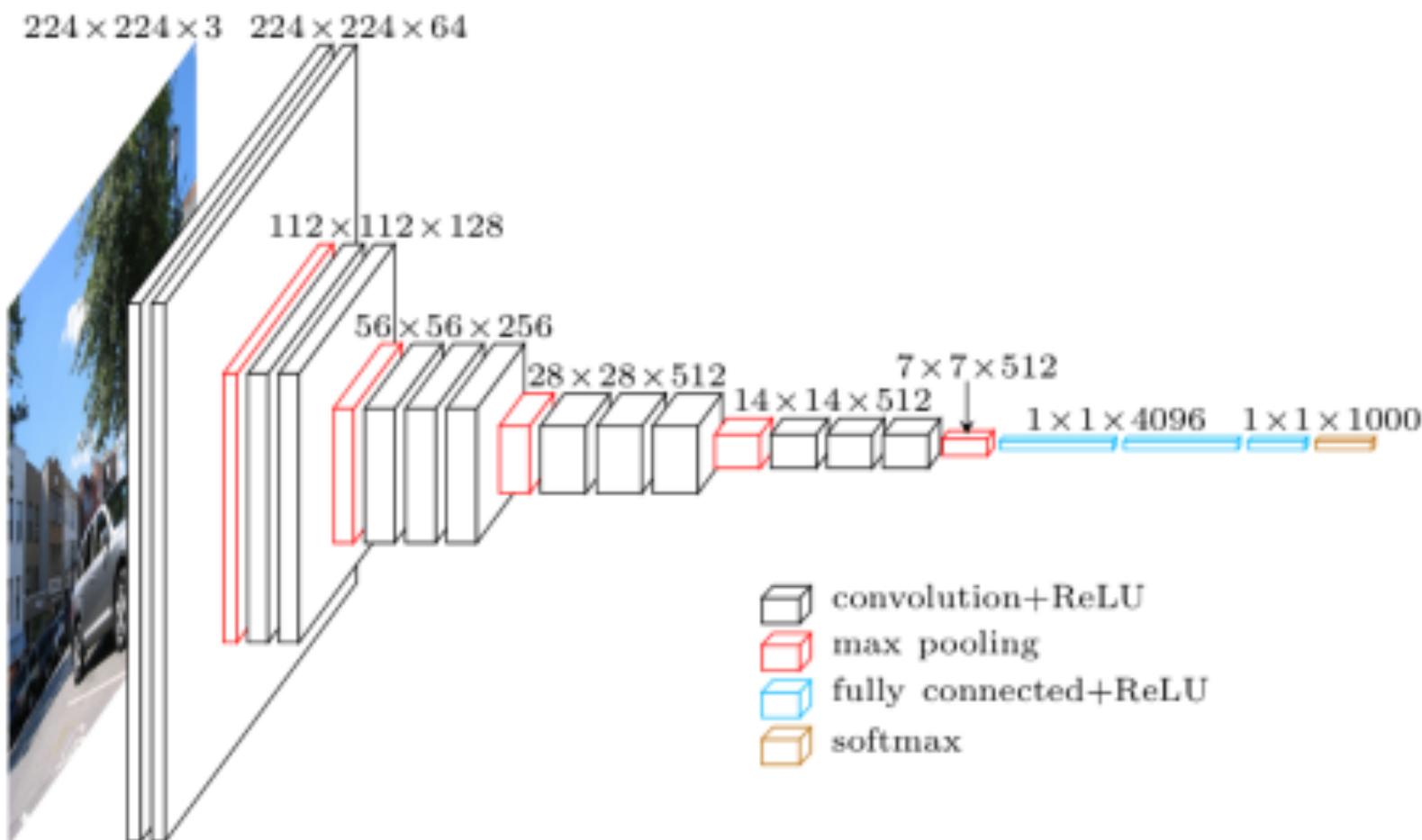
Importing VGG16 Model and its weights

We will be using pre-trained network VGG16. This model takes input image of size (224, 224, 3).

We choose this model because it achieves 92.7% (top-5 test accuracy) in ImageNet, which is a dataset of over 14 million images belonging to 1000 classes.

VGG16 is a type of CNN (Convolutional Neural Network) that is considered to be one of the best computer vision models to date. The creators of this model evaluated the networks and increased the depth using an architecture with very small (3×3) convolution filters, which showed a significant improvement on the prior-art configurations. They pushed the depth to 16–19 weight layers making it approx — 138 trainable parameters.

VGG16 Architecture



The network architecture consists of 13 convolutional layers, three fully connected layers, and five pooling layers [19], a diagram of which is shown in Fig. 11. The size of the convolution kernel in the convolutional layers is 3×3 with stride fixed at 1. The size of the kernel in the pool layers is 2×2 with step size 2. The convolutional layers use the rectified linear unit (ReLU) as the activation function. The VGG-16 network receives input as a three-channel 224×224 -pixel image. The first part contains two convolutional layers followed by a pooling layer. These convolution layers contain 64 kernels, each at 224×224 pixels.

```
In [12]: modelvgg = tf.keras.applications.VGG16(include_top=True, weights=None)
modelvgg.load_weights("C:\\\\Users\\\\karth\\\\Downloads\\\\vgg16_weights_tf_dim_ordering_tf_kernels.h5")
modelvgg.summary()
Model: "vgg16"
```

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312
predictions (Dense)	(None, 1000)	4,097,000

Total params: 138,357,544 (527.79 MB)

Trainable params: 138,357,544 (527.79 MB)

Non-trainable params: 0 (0.00 B)

```
In [13]: modelvgg.layers.pop()
modelvgg = tf.keras.Model(inputs=modelvgg.inputs, outputs=modelvgg.layers[-2].output)
## show the deep Learning model
modelvgg.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 224, 224, 3)	0
block1_conv1 (Conv2D)	(None, 224, 224, 64)	1,792
block1_conv2 (Conv2D)	(None, 224, 224, 64)	36,928
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0
block2_conv1 (Conv2D)	(None, 112, 112, 128)	73,856
block2_conv2 (Conv2D)	(None, 112, 112, 128)	147,584
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0
block3_conv1 (Conv2D)	(None, 56, 56, 256)	295,168
block3_conv2 (Conv2D)	(None, 56, 56, 256)	590,080
block3_conv3 (Conv2D)	(None, 56, 56, 256)	590,080
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0
block4_conv1 (Conv2D)	(None, 28, 28, 512)	1,180,160
block4_conv2 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_conv3 (Conv2D)	(None, 28, 28, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0
block5_conv1 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv2 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_conv3 (Conv2D)	(None, 14, 14, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
fc1 (Dense)	(None, 4096)	102,764,544
fc2 (Dense)	(None, 4096)	16,781,312

Total params: 134,260,544 (512.16 MB)

Trainable params: 134,260,544 (512.16 MB)

Non-trainable params: 0 (0.00 B)

Transforming images into features using VGG16 model

Here we process every image into a feature vector. While keeping computation complexity in mind we do it batch wise.

```
In [26]: from keras.preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input
from collections import OrderedDict

images = OrderedDict()
npix = 224
target_size = (npix, npix, 3)
batch_size = 64 # Adjust batch size based on your system's memory capacity
num_images = len(jpgs)

for i in range(0, num_images, batch_size):
    batch_images = OrderedDict()
    batch_data = np.zeros((min(batch_size, num_images - i), npix, npix, 3))

    for j, name in enumerate(jpgs[i:i+batch_size]):
        # Load an image from file
        filename = dir_Flickr_jpg + '/' + 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)
        batch_data[j] = nimage

    batch_predictions = modelvgg.predict(batch_data)

    for j, name in enumerate(jpgs[i:i+batch_size]):
        images[name] = batch_predictions[j].flatten()
```


2/2	12s	6s/step
2/2	13s	6s/step
2/2	12s	6s/step
2/2	13s	6s/step
2/2	13s	6s/step
2/2	12s	6s/step
2/2	11s	5s/step
2/2	11s	6s/step
2/2	12s	6s/step
2/2	11s	6s/step
2/2	11s	6s/step
2/2	12s	6s/step
1/1	5s	5s/step

```
In [27]: images['1000268201_693b08cb0e.jpg']
```

```
Out[27]: array([2.507474, 0.        , 0.        , ..., 0.        , 0.        , 0.        ])
```

```
In [28]: len(images)
```

Out[28]: 8091

Linking text to corresponding images

```
In [29]: dimages, keepindex = [], []
df_txt0 = df_txt0.loc[df_txt0["index"].values == "0", :]
for i, fnm in enumerate(df_txt0.filename):
    if fnm in images.keys():
        dimages.append(images[fnm])
        keepindex.append(i)

fnames = df_txt0["filename"].iloc[keepindex].values
dcaptions = df_txt0["caption"].iloc[keepindex].values
dimages = np.array(dimages)
```

```
In [30]: dimages.shape
```

Out[30]: (8091, 4096)

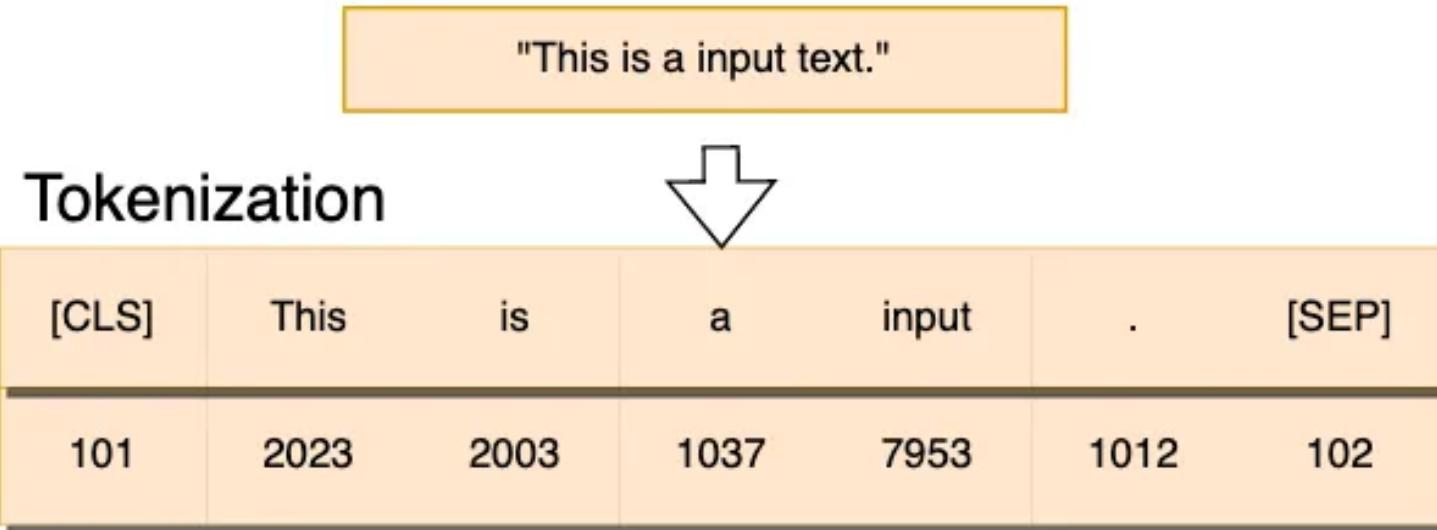
In [35]: dcaptions[:5]

```
Out[35]: array(['startseq child in pink dress is climbing up set of stairs in an entry way endseq',  
   'startseq black dog and spotted dog are fighting endseq',  
   'startseq little girl covered in paint sits in front of painted rainbow with her hands in bowl endseq',  
   'startseq man lays on bench while his dog sits by him endseq',  
   'startseq man in an orange hat staring at something endseq'],  
  dtype=object)
```

Tokenizing

Tokenization is the process of breaking down a text into smaller units called tokens. These tokens can be words, subwords, or even characters, depending on the granularity required for a particular task or analysis. Tokenization is a fundamental preprocessing step in NLP, and its primary

goal is to prepare text data for further analysis or processing by converting it into a format suitable for machine learning algorithms or linguistic analysis.



```
In [44]: from tensorflow.keras.preprocessing.text import Tokenizer
## the maximum number of words in dictionary
nb_words = 8000
tokenizer = Tokenizer(nb_words=nb_words)
tokenizer.fit_on_texts(dcaptions)
vocab_size = len(tokenizer.word_index) + 1
print("vocabulary size : {}".format(vocab_size))
dtexts = tokenizer.texts_to_sequences(dcaptions)
```

vocabulary size : 4476

```
In [45]: dtexts[:5]
```

```
Out[45]: [[1, 38, 3, 66, 144, 7, 124, 52, 406, 9, 367, 3, 24, 2351, 522, 2],
[1, 12, 8, 5, 752, 8, 17, 368, 2],
[1, 48, 15, 170, 3, 584, 101, 3, 41, 9, 551, 1198, 11, 55, 213, 3, 1076, 2],
[1, 10, 621, 6, 150, 27, 23, 8, 101, 46, 112, 2],
[1, 10, 3, 24, 82, 96, 1199, 19, 162, 2]]
```

Spliting the dataset

```
In [46]: # We will split the dataset in ratio 6:2:2 (train:valid:test)

prop_test, prop_val = 0.2, 0.2

N = len(dtexts)
Ntest, Nval = int(N*prop_test), int(N*prop_val)

def split_test_val_train(dtexts,Ntest,Nval):
    return(dtexts[:Ntest],
           dtexts[Ntest:Ntest+Nval],
           dtexts[Ntest+Nval:])

dt_test, dt_val, dt_train = split_test_val_train(dtexts,Ntest,Nval)
di_test, di_val, di_train = split_test_val_train(dimages,Ntest,Nval)
fnm_test, fnm_val, fnm_train = split_test_val_train(fnames,Ntest,Nval)
```

```
In [47]: maxlen = np.max([len(text) for text in dtexts])
```

Here we need to maintain uniformity of length in the captions

So we find the max length out of all captions and the we pad zeros to remaining captions until they reach to the max length in the data set

```
In [48]: from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical

def preprocessing(dtexts,dimages):
    N = len(dtexts)
    print("# captions/images = {}".format(N))

    assert(N==len(dimages))
    Xtext, Ximage, ytext = [],[],[]
    for text,image in zip(dtexts,dimages):

        for i in range(1,len(text)):
            in_text, out_text = text[:i], text[i]
            in_text = pad_sequences([in_text],maxlen=maxlen).flatten()
            out_text = to_categorical(out_text,num_classes = vocab_size)

            Xtext.append(in_text)
            Ximage.append(image)
            ytext.append(out_text)
```

```
Xtext = np.array(Xtext)
Ximage = np.array(Ximage)
ytext = np.array(ytext)
print(" {} {} {}".format(Xtext.shape,Ximage.shape,ytext.shape))
return(Xtext,Ximage,ytext)

Xtext_train, Ximage_train, ytext_train = preprocessing(dt_train,di_train)
Xtext_val, Ximage_val, ytext_val = preprocessing(dt_val,di_val)
# pre-processing is not necessary for testing data
#Xtext_test, Ximage_test, ytext_test = preprocessing(dt_test,di_test)

# captions/images = 4855
(49631, 30) (49631, 4096) (49631, 4476)
# captions/images = 1618
(16353, 30) (16353, 4096) (16353, 4476)
```

```
In [49]: Xtext_train[:14]
```

```
In [51]: Ximage_train.shape
```

Out[51]: (49631, 4096)

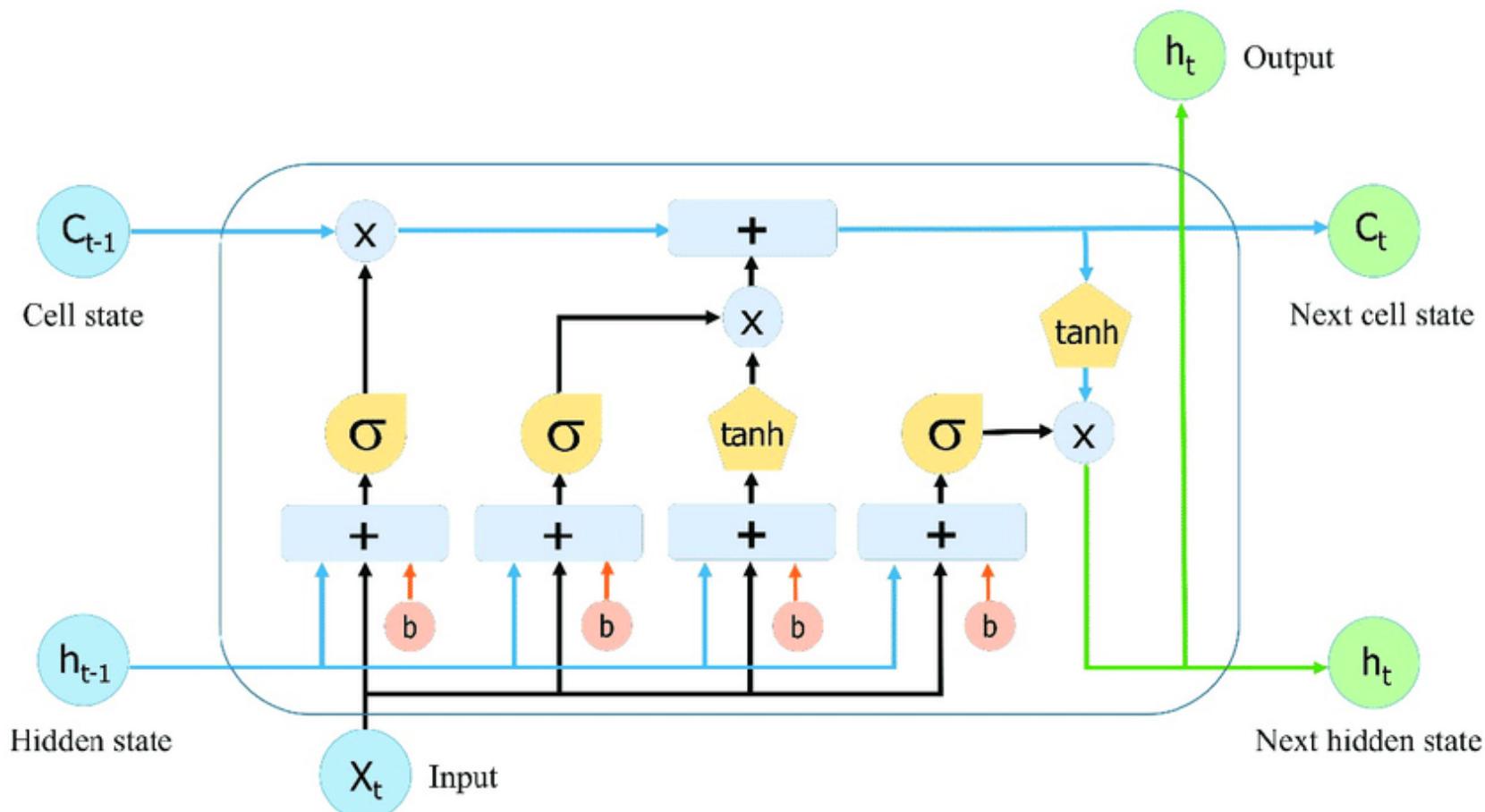
```
In [52]: ytext_train.shape
```

Out[52]: (49631, 4476)

Model

LSTM

Long Short-Term Memory (LSTM) is a type of artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTMs have feedback connections, allowing them to exploit temporal dependencies across sequences of data. LSTM is designed to handle the issue of vanishing or exploding gradients, which can occur when training traditional RNNs on sequences of data. LSTM networks introduce memory cells, which have the ability to retain information over long sequences. Each memory cell has three main components: an input gate, a forget gate, and an output gate. These gates help regulate the flow of information in and out of the memory cell.



Inputs:

X_t Current input

C_{t-1} Memory from last LSTM unit

h_{t-1} Output of last LSTM unit

Outputs:

C_t New updated memory

h_t Current output

Nonlinearities:

σ Sigmoid layer

\tanh Tanh layer

b Bias

Vector operations:

x Scaling of information

$+$ Adding information

combination of the cell state, hidden state, and gates allows the LSTM to selectively "remember" or "forget" information over time, making it well-suited for tasks that require modeling long-term dependencies or sequences.

```
In [53]: from tensorflow.keras import layers
print(vocab_size)
## image feature

dim_embedding = 64

input_image = layers.Input(shape=(Ximage_train.shape[1],))
fimage = layers.Dense(256, activation='relu', name="ImageFeature")(input_image)

## sequence model
input_txt = layers.Input(shape=(maxlen,))
### The embedding Layer in Keras can be used when we want to create the embeddings to embed higher dimensional data into lower
ftxt = layers.Embedding(vocab_size, dim_embedding, mask_zero=True)(input_txt)
ftxt = layers.LSTM(256, name="CaptionFeature")(ftxt)

## combined model for decoder
decoder = layers.add([ftxt,fimage])
decoder = layers.Dense(256, activation='relu')(decoder)
output = layers.Dense(vocab_size, activation='softmax')(decoder)
model = tf.keras.Model(inputs=[input_image, input_txt],outputs=output)

model.compile(loss='categorical_crossentropy', optimizer='adam')

print(model.summary())
4476
Model: "functional_3"
```

Layer (type)	Output Shape	Param #	Connected to
input_layer_2 (InputLayer)	(None, 30)	0	-
embedding (Embedding)	(None, 30, 64)	286,464	input_layer_2[0][0]
not_equal (NotEqual)	(None, 30)	0	input_layer_2[0][0]
input_layer_1 (InputLayer)	(None, 4096)	0	-
CaptionFeature (LSTM)	(None, 256)	328,704	embedding[0][0], not_equal[0][0]
ImageFeature (Dense)	(None, 256)	1,048,832	input_layer_1[0][0]
add (Add)	(None, 256)	0	CaptionFeature[0][0], ImageFeature[0][0]
dense (Dense)	(None, 256)	65,792	add[0][0]
dense_1 (Dense)	(None, 4476)	1,150,332	dense[0][0]

Total params: 2,880,124 (10.99 MB)

Trainable params: 2,880,124 (10.99 MB)

Non-trainable params: 0 (0.00 B)

None

We are giving our model an image and a word (`Ximage_train` & `Xtext_train` respectively) and we want it to predict the next word for it which is our target `ytext_train`

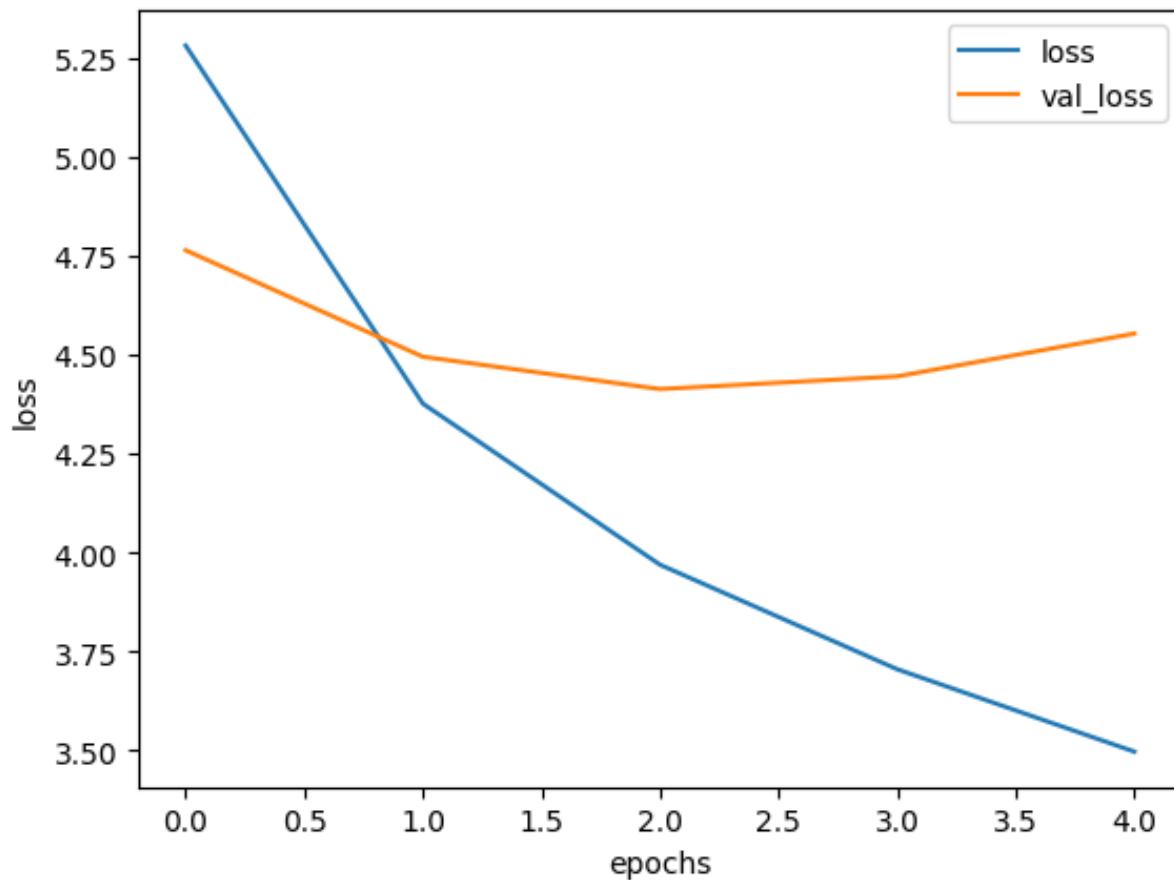
```
In [61]: # fit model
start = time.time()

hist = model.fit([Ximage_train, Xtext_train], ytext_train,
                 epochs=5, verbose=2,
                 batch_size=32,
                 validation_data=([Ximage_val, Xtext_val], ytext_val))

end = time.time()
print("TIME TOOK {:.2f}MIN".format((end - start )/60))

Epoch 1/5
1551/1551 - 156s - 100ms/step - loss: 5.2804 - val_loss: 4.7632
Epoch 2/5
1551/1551 - 101s - 65ms/step - loss: 4.3760 - val_loss: 4.4944
Epoch 3/5
1551/1551 - 86s - 55ms/step - loss: 3.9691 - val_loss: 4.4127
Epoch 4/5
1551/1551 - 85s - 55ms/step - loss: 3.7051 - val_loss: 4.4448
Epoch 5/5
1551/1551 - 85s - 55ms/step - loss: 3.4969 - val_loss: 4.5528
TIME TOOK 8.59MIN
```

```
In [62]: for label in ["loss", "val_loss"]:
    plt.plot(hist.history[label], label=label)
plt.legend()
plt.xlabel("epochs")
plt.ylabel("loss")
plt.show()
```



```
In [69]: # Define a List of indexes for images you want to display
indexes_to_display = [100, 22, 324, 6, 8] # Modify this list as needed

# Assuming fnm_test and di_test are your lists of filenames and image features respectively
npic = len(indexes_to_display)
npix = 224
target_size = (npix, npix, 3)

count = 1
fig = plt.figure(figsize=(10, 20))
for idx in indexes_to_display:
    jpgfnm = fnm_test[idx]
    image_feature = di_test[idx]

    # Load the image
    filename = dir_Flickr_jpg + '/' + jpgfnm
    image_load = load_img(filename, target_size=target_size)
    ax = fig.add_subplot(npic, 2, count, xticks=[], yticks[])
    ax.imshow(image_load)
    count += 1

    # Predict and display caption
    caption = predict_caption(image_feature.reshape(1, len(image_feature)))
    ax = fig.add_subplot(npic, 2, count)
    plt.axis('off')
    ax.plot()
    ax.set_xlim(0, 1)
    ax.set_ylim(0, 1)
    ax.text(0, 0.5, caption, fontsize=20)
    count += 1

plt.show()
```



startseq brown and white dog is running through the grass endseq



startseq black and white dog is running through the grass endseq



startseq group of people are sitting on the beach endseq



startseq dog is jumping through the grass endseq



startseq man in black shirt is standing in front of the street endseq

BLEU (Bilingual Evaluation Understudy) Score :

BLEU score measures the similarity between the machine-translated text and the reference translations using n-grams, which are contiguous sequences of n words. BLEU score calculates the precision of n-grams in the machine-generated translation by comparing them to the reference translations. The precision is then modified by a brevity penalty to account for translations that are shorter than the reference translations.

The formula for BLEU score is as follows:

$$\text{BLEU} = \text{BP} * \exp(\sum p_n)$$

Where:

- p_n is the precision of n-grams
- BP (Brevity Penalty) is a penalty term that adjusts the score for translations that are shorter than the reference translations.

BLEU score ranges from 0 to 1, with higher values indicating better translation quality. A perfect translation would have a BLEU score of 1, while

a completely incorrect translation would have a BLEU score of 0.

```
In [65]: start = time.time()

from nltk.translate.bleu_score import sentence_bleu

index_word = dict([(index,word) for word, index in tokenizer.word_index.items()])

nkeep = 5
pred_good, pred_bad, bleus = [], [], []
count = 0
for jpgfnm, image_feature, tokenized_text in zip(fnm_test,di_test,dt_test):
    count += 1
    if count % 200 == 0:
        print(" {:4.2f}% is done..".format(100*count/float(len(fnm_test)))))

    caption_true = [ index_word[i] for i in tokenized_text ]
    caption_true = caption_true[1:-1] ## remove startreg, and endreg
    ## captions
    caption = predict_caption(image_feature.reshape(1,len(image_feature)))
    caption = caption.split()
    caption = caption[1:-1]## remove startreg, and endreg

    bleu = sentence_bleu([caption_true],caption)
    bleus.append(bleu)
    if bleu > 0.6 and len(pred_good) < nkeep:
        pred_good.append((bleu,jpgfnm,caption_true,caption))
    elif bleu < 0.3 and len(pred_bad) < nkeep:
        pred_bad.append((bleu,jpgfnm,caption_true,caption))

end = time.time()
print((start-end)/60)
```

12.36% is done..
24.72% is done..
37.08% is done..
49.44% is done..
61.80% is done..
74.17% is done..
86.53% is done..
98.89% is done..
-33.92743647098541

```
In [72]: pred_good[:2]
```

```
Out[72]: [(0.8091067115702212,
    '1117972841_2b9261f95f.jpg',
    ['brown', 'dog', 'is', 'running', 'in', 'the', 'sand'],
    ['brown', 'dog', 'is', 'running', 'in', 'the', 'snow']),
(0.8633400213704505,
    '1119418776_58e4b93eac.jpg',
    ['black', 'and', 'white', 'dog', 'is', 'running', 'through', 'the', 'field'],
    ['black',
     'and',
     'white',
     'dog',
     'is',
     'running',
     'through',
     'the',
     'grass'])]
```

```
In [71]: pred_bad[:2]
```

```
Out[71]: [(0,
  '1000268201_693b08cb0e.jpg',
  ['child',
   'in',
   'pink',
   'dress',
   'is',
   'climbing',
   'up',
   'set',
   'of',
   'stairs',
   'in',
   'an',
   'entry',
   'way'],
  ['two', 'girls', 'are', 'sitting', 'on', 'the', 'camera']),
 (1.384292958842266e-231,
  '1001773457_577c3a7d70.jpg',
  ['black', 'dog', 'and', 'spotted', 'dog', 'are', 'fighting'],
  ['black', 'and', 'white', 'dog', 'is', 'running', 'in', 'the', 'grass'])]
```

```
In [73]: print("Mean BLEU {:.4f}".format(np.mean(bleus)))
```

```
Mean BLEU 0.012
```

```
In [74]: def plot_images(pred_bad):
    def create_str(caption_true):
        sttrue = ""
        for s in caption_true:
            sttrue += " " + s
        return(sttrue)
    npix = 224
    target_size = (npix,npix,3)
    count = 1
    fig = plt.figure(figsize=(10,20))
    npic = len(pred_bad)
    for pb in pred_bad:
        bleu,jpgfnm,caption_true,caption = pb
        ## images
        filename = dir_Flickr_jpg + '/' + jpgfnm
        image_load = load_img(filename, target_size=target_size)
        ax = fig.add_subplot(npic,2,count,xticks=[],yticks[])
        ax.imshow(image_load)
        count += 1

        caption_true = create_str(caption_true)
        caption = create_str(caption)

        ax = fig.add_subplot(npic,2,count)
        plt.axis('off')
        ax.plot()
        ax.set_xlim(0,1)
        ax.set_ylim(0,1)
        ax.text(0,0.7,"true:" + caption_true,fontsize=20)
        ax.text(0,0.4,"pred:" + caption,fontsize=20)
        ax.text(0,0.1,"BLEU: {}".format(bleu),fontsize=20)
        count += 1
    plt.show()

print("Bad Caption")
plot_images(pred_bad)
print("Good Caption")
plot_images(pred_good)
```

```
Bad Caption
```



true: child in pink dress is climbing up set of stairs in an entry way

pred: two girls are sitting on the camera

BLEU: 0



true: black dog and spotted dog are fighting

pred: black and white dog is running in the grass

BLEU: 1.384292958842266e-231



true: little girl covered in paint sits in front of painted rainbow with her hands in bowl

pred: man in red shirt is standing on the beach

BLEU: 4.832402486973385e-232



true: man lays on bench while his dog sits by him

pred: boy is jumping to the grass

BLEU: 0



true: man in an orange hat staring at something

pred: man in black shirt is smiling

BLEU: 5.431059661402121e-155

Good Caption



true: brown dog is running in the sand

pred: brown dog is running in the snow

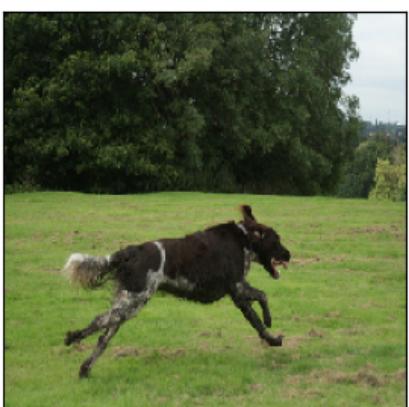
BLEU: 0.8091067115702212



true: black and white dog is running through the field

pred: black and white dog is running through the grass

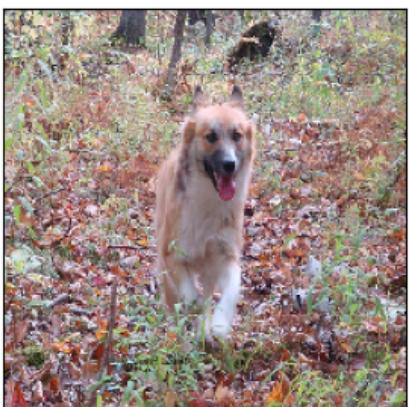
BLEU: 0.8633400213704505



true: black and white dog is running in the grass

pred: black and white dog is running through the grass

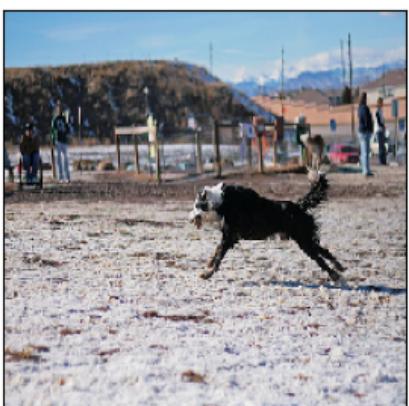
BLEU: 0.6606328636027614



true: brown and white dog is running through woodland

pred: brown and white dog is running through the grass

BLEU: 0.7259795291154771



true: black and white dog is running through the sand at beach

pred: black and white dog is running through the snow

BLEU: 0.6913086465463161

Conclusion :

- We started this project with the goal of creating captions for images by using a mix of computer vision and natural language processing techniques. We used the Flickr8k Dataset, which has a wide range of images and captions for each image, to develop our solution.
- We used a pre-trained VGG16 model to extract features from images and an LSTM (Long Short-Term Memory) model to generate captions based on those features. We combined these models to understand image content and create textual descriptions that capture each image's essence.
- We evaluated our caption generation model by using the BLEU (Bilingual Evaluation Understudy) metric. This metric measures how similar the generated captions are to the reference captions.
- In the future, there are many ways we can improve and expand. By trying out various architectures, including attention mechanisms, and experimenting with transfer learning techniques.

References :

- dataset : <https://www.kaggle.com/datasets/sayanf/flickr8k>
- Model Source(VGG16): https://github.com/fchollet/deep-learning-models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels.h5
- Most of the code is inspired from : <https://github.com/akshatchaturvedi28/Image-Caption-Generator-with-GUI/blob/main/README.md>
- Other References <https://medium.com/@sthanikamsanthosh1994/understanding-bleu-and-rouge-score-for-nlp-evaluation-1ab334ecadcb>
- Other References <https://medium.com/@rebeen.jaff/what-is-lstm-introduction-to-long-short-term-memory-66bd3855b9ce>