IMAGE CAPTION GENERATOR WITH DEEP LEARNING

(COURSE CODE: 23UPCSC4P01)

A Professional Competency Skill – Mini Project record submitted to Periyar University, Salem.
In partial fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE
IN
DATA SCIENCE

BY

MADESH M
REG NO: U23PG507DTS018

DEPARTMENT OF COMPUTER SCIENCE
PERIYAR UNIVERSITY

NAAC 'A++' Grade with CGPA 3.61 (Cycle - 3)

State University - NIRF Rank 59 - NIRF Innovation Band of 11-50

Salem-636011, Tamilnadu, India.

April – 2024.

Dr. K. SASIREKHA,
Teaching Assistant,
Department of Computer Science,
Periyar University,
Salem-11.

CERTIFICATE

This is to certify that the report of **“Professional Competency Skill - Mini Project” (23UPCSC4P01)** entitled **"Image Caption Generation with Deep Learning"** submitted in partial fulfilment of the requirement for the degree of Master of Science in Data Science to the Periyar University, Salem is a record of bonafide work carried out by **MADESH M,** Register No. **U23PG507DTS018** under my supervision and guidance.

Signature of the Guide Signature of the HOD

Submitted of the Viva-Voce Examination held on ________________.

Internal Examiner External Examiner

DECLARATION

I hereby, declare that the project work entitled **"Image caption generation with Deep learning"** submitted to Periyar University in partial fulfilment of the requirement for the award of the Degree of **Master of Science in Data Science** is the record work carried out by me, under the supervision **Dr. K. SASIREKHA, Teaching Assistant of Computer Science, Periyar University**. To the best of my knowledge, the work reported here is not a part of any other work on the basis of which a degree or award was conferred on an earlier to one or any other candidate.

Place: Salem **Signature of Student**

Date:

ACKNOWLEDGMENT

First, I would like to thank the almighty for providing mw with everything that I requested to completing the project.

I would like to extend my sincere thanks to **Dr. R. JAGANNATHAN**, Vice Chancellor Periyar University, Salem who has been an invaluable source of providing the facility to complete the mini project successfully.

I would like to extend my sincere thanks to **Dr. C. CHANDRASEKAR**, Head of the Department, Department of Computer Science, Periyar University, Salem, For the support and encourage.

I express my sincere thanks to my guide **Dr. K. SASIREKHA**, Teaching Assistant, Department of Computer Science, Periyar University, Salem for the motivation and kind suggestion given in every step throughout this dissertation work and for valuable support in finishing this dissertation.

I extend my thanks to my parents and friends for their constant support and encouragement.

Place: Salem **Signature of the Student**

Date:

ABSTRACT

This project implements an end-to-end solution for automatic image caption generation using deep learning techniques, integrating computer vision and natural language processing. Key components include VGG16-based image feature extraction, text preprocessing, and an encoder-decoder architecture for caption generation.

Image features are extracted using VGG16 and preprocess captions are paired with these features. An encoder extracts image features, while a decoder generates captions by merging these features with input text sequences.

Model training utilizes a data generator to prevent session crashes, with the model trained using categorical cross-entropy loss and the Adam optimizer. Evaluation employs BLEU scores on a test dataset.

Additionally, a **web application is created using Streamlit** to provide a user-friendly interface for generating captions.

This project demonstrates the integration of deep learning techniques for automated caption generation, with applications in image indexing, retrieval, and accessibility enhancement.

INDEX

SO.NO	TITLE	PAGE NO
1	Introduction	1
2	Problem statement	2
3	Objectives	3
4	System configuration	4
5	Data exploration	6
6	Working mechanism	8
7	Methodology	9
8	Scores	19
9	Source code	20
10	Conclusion	35
11	Bibliography	36

CHAPTER - 1

INTRODUCTION

Our project focuses on automatic image captioning, a crucial task in AI and computer vision. Using deep learning techniques, we aim to develop an end-to-end solution, primarily utilizing the Flickr 8k dataset. This dataset offers thousands of images, each with multiple descriptive captions, making it an ideal benchmark.

Our approach integrates advanced methods from computer vision and NLP. We leverage the VGG16 convolutional neural network to extract high-level features from images, forming the basis for our caption generation model. This model employs an encoder-decoder architecture with LSTM layers for coherent caption generation.

Data preprocessing, model training, and evaluation are central to our project. We preprocess both image data and captions, tokenizing and cleaning the text for training. Model training utilizes a data generator to efficiently handle large training volumes, and evaluation employs metrics such as BLEU scores.

Beyond training and evaluation, our system allows for practical caption generation for new images. Users can input images and receive descriptive captions, enhancing accessibility and user experience. Our aim is to showcase the effectiveness of deep learning in automatic image captioning across various domains.

CHAPTER - 2

PROBLEM STATEMENT

This project aims to automate the generation of descriptive captions for images using deep learning techniques. It begins by extracting image features through pre-trained convolutional neural networks like VGG16 and standardizing textual captions. The dataset is partitioned into training and testing subsets, and an encoder-decoder architecture with LSTM networks is devised to produce captions based on image features. Model performance is evaluated using metrics such as BLEU scores. Additionally, the project focuses on integrating user-friendly interfaces, particularly utilizing Streamlit, to create interactive applications for caption generation. The objective is to contribute to various applications, including image retrieval and other domains requiring automated captioning.

OBJECTIVES

Firstly, we aim to create a comprehensive solution for automatic image caption generation by leveraging deep learning methodologies. This involves utilizing the Flickr 8k dataset as our primary source for training data. Secondly, we intend to integrate advanced techniques from both computer vision and natural language processing to effectively extract image features and generate descriptive captions. Central to this is the design and training of an encoder-decoder architecture with LSTM layers, which will be pivotal in processing textual information and producing coherent captions. Additionally, we emphasize the importance of thorough data preprocessing, including tokenization and cleaning, to ensure the quality of training inputs. We also plan to employ a data generator to efficiently handle the large volumes of training data required for model training. Evaluation of model performance will be conducted using metrics such as BLEU scores to gauge the accuracy and quality of generated captions. Furthermore, we aim to develop user-friendly interfaces, with a particular focus on integrating a Streamlit application for ease of use and accessibility in generating captions for new images. Lastly, our overarching objective is to demonstrate the efficacy of deep learning techniques in automatic image captioning and highlight potential applications across various domains.

CHAPTER - 3

SYSTEM CONFIGURATION

Requirement System Specification:

The Image Caption Generator project requires a system with the following specifications:

Operating System: The project can be developed and deployed on various operating systems, including Windows, macOS, and Linux.

Programming Language: Python is the primary programming language used for this project due to its extensive support for deep learning libraries such as TensorFlow and PyTorch, as well as tools for image processing, natural language processing, and model deployment.

Deep Learning Framework: TensorFlow or PyTorch can be used as the primary deep learning framework for implementing the image captioning model.

Libraries: The project relies on several Python libraries, including TensorFlow or PyTorch for deep learning, NumPy for numerical computations, pandas for data manipulation, and NLTK or spaCy for natural language processing tasks.

Image Processing Libraries: OpenCV is used for image loading, preprocessing, and feature extraction tasks.

Development Environment: The project can be developed using popular integrated development environments (IDEs) such as PyCharm, Jupyter Notebook, or Visual Studio Code.

Hardware Requirements: While the project can be developed and tested on standard personal computers are 8GB or more, training deep learning models may require access to hardware with GPU support to accelerate computation, especially for larger datasets and complex models.

Deployment Options: The trained image captioning model can be deployed as a standalone application or integrated into web or mobile applications using frameworks like Flask or Django for web deployment. Additionally, the model can be deployed using Streamlit to create interactive web applications for easy deployment and usage.

These specifications provide guidance on the necessary components and environment for developing, training, and deploying the Image Caption Generator project.

MY SYSTEM SPECIFICATION:

SOFTWARE SPECIFICATION

Operating System	Windows 11
Coding Language	Python Language
Software Tool	Anaconda, Jupyter Notebook.

HARDWARE CONFIGURATION

Processor	12th Gen Intel(R) Core(TM) i5-1240P 1.70 GHz
Installed RAM	16.0 GB
Hard Disk	512 GB

CHAPTER - 4

DATA EXPLORATION

The dataset was found in the following Kaggle page Flickr 8k Dataset (kaggle.com). The Flickr8k dataset, available on Kaggle, offers a valuable resource for image captioning tasks, comprising 8,000 images, each paired with five captions. Initial exploration involves inspecting samples to ensure alignment between images and captions. Basic statistics, like total images and average caption length, provide insights into dataset characteristics. Analyzing caption distribution reveals linguistic diversity. Concurrently, examining images uncovers visual patterns. Textual analysis identifies prevalent words, aiding semantic understanding. Data cleaning ensures consistency, addressing issues like missing values. Visualization techniques, such as histograms, succinctly present findings, aiding comprehension of the dataset's potential in machine learning and computer vision research.

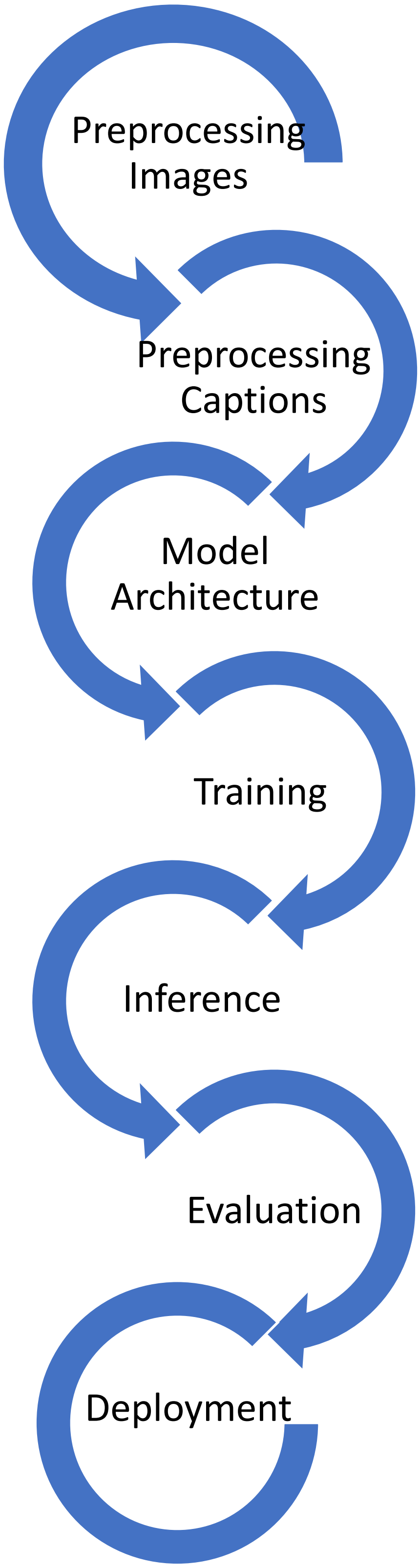
TABLE OF COLUMN

COLUMN NAMES	DESCRIPTION
IMAGE ID	Unique identifier for each image in the Flickr 8k dataset
CAPTION	Descriptive caption associated with the image
FEATURES	Extracted features from the input image using VGG16
PREPROCESS	Preprocess of caption text, including lowercase and removal of special characters, addition of start and end tags
TOKENIZER	Tokenized text data for training the model
VOCAB SIZE	Size of the vocabulary obtained from the tokenizer
MAX LENGTH	Maximum length of sequences in the preprocessed captions
MODEL	Maximum length of sequences in the preprocessed captions
EVALUATION	Evaluation metric (e.g., BLEU scores) for model performance
PREDICTION	Predicted caption for new images

These columns are utilized in the project with the Flickr 8k dataset to process image-caption pairs, train the model, evaluate its performance, and generate captions for new images

CHAPTER - 5

WORKING MECHANISM



CHAPTER - 6

METHODOLOGY

Libraries

[36]:
```
# Libraries
import pandas as pd
import numpy as np
import sklearn
import os
import pickle
from tqdm.notebook import tqdm
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mping

from tensorflow.keras.applications.vgg16 import VGG16,preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model

from tensorflow.keras.layers import Input,Dense, LSTM, Embedding,Dropout,add
```

Directory

*[2]:
```
# Directory
BASE_DIR ="D:\\Data_science\\Mini Project\\image caption generater"

# %pwd
WORKING_DIR ='C:\\Users\\mades\\Documents\\Image Caption Generater'
```

Create Model

[3]:
```
# load vgg16 model
model = VGG16()
# restructure the model
model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
# summarize
print(model.summary())
```

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)
None
```

Preprocessing Images:

Extract features from image

[ ]:
```
# extract features from image
features = {}
directory = os.path.join(BASE_DIR, 'Images')

for img_name in tqdm(os.listdir(directory)):
    # load the image from file
    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, verbose=0)
    # get image ID
    image_id = img_name.split('.')[0]
    # store feature
    features[image_id] = feature
```

Feature files:

Store the features in pickel

```
[ ]: # store features in pickle
pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'), 'wb'))
```

load features from pickle

```
[4]: # load features from pickle
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
```

```
[5]: len(features)
```

```
[5]: 8091
```

Load captions_doc

```
[6]: with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
```

```
[7]: len(captions_doc)
```

```
[7]: 3319280
```

Preprocessing Captions:

create mapping of image to captions

```
[8]: # create mapping of image to captions
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
```

100% 40456/40456 [00:00<00:00, 420499.94it/s]

```
[9]: def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            caption = caption.replace(' +', ' ')

            # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + ' endseq'
            captions[i] = caption
```

Cleaning:

sample testing to clean(mapping)

```
[10]: mapping['1000268201_693b08cb0e']
```

```
[10]: ['A child in a pink dress is climbing up a set of stairs in an entry way .',
 'A girl going into a wooden building .',
 'A little girl climbing into a wooden playhouse .',
 'A little girl climbing the stairs to her playhouse .',
 'A little girl in a pink dress going into a wooden cabin .']
```

```
[11]: clean(mapping)
```

```
[12]: mapping['1000268201_693b08cb0e']
```

```
[12]: ['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']
```

```
[13]: all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
```

```
[14]: len(all_captions)
```

```
[14]: 40455
```

```
[15]: all_captions[:10]
```

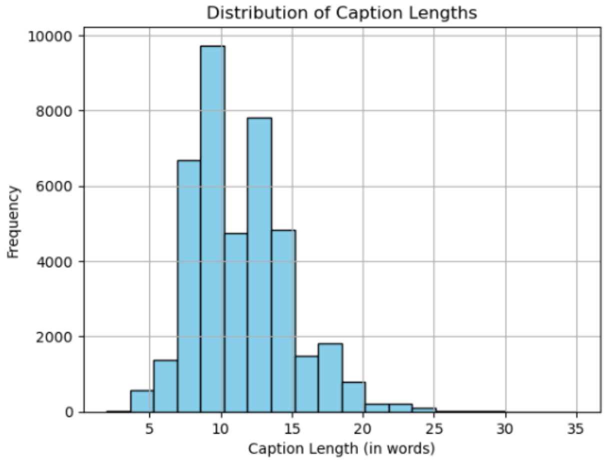
```
[15]: ['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq',
 'startseq black dog and spotted dog are fighting endseq',
 'startseq black dog and tri-colored dog playing with each other on the road endseq',
 'startseq black dog and white dog with brown spots are staring at each other in the street endseq',
 'startseq two dogs of different breeds looking at each other on the road endseq',
 'startseq two dogs on pavement moving toward each other endseq']
```

Histogram:

```
[31]: import matplotlib.pyplot as plt

# Assuming caption_lengths is a list containing the lengths of all captions
caption_lengths = [len(caption.split()) for caption in all_captions]

plt.hist(caption_lengths, bins=20, color='skyblue', edgecolor='black')
plt.xlabel('Caption Length (in words)')
plt.ylabel('Frequency')
plt.title('Distribution of Caption Lengths')
plt.grid(True)
plt.show()
```



Tokenize the text

[16]:
```
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
```

[17]:
```
vocab_size
```

[17]: 8485

[18]:
```
max_length = max(len(caption.split()) for caption in all_captions)
max_length
```

[18]: 35

split data - train, test

[19]:
```
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
```

[20]:
```
len(image_ids)
```

[20]: 8091

[21]:
```
len(train)
```

[21]: 7281

create data generator to get data in batch (avoids session crash)

[22]:
```
import tensorflow as tf

def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):
    def generator():
        while True:
            for key in data_keys:
                captions = mapping[key]
                for caption in captions:
                    seq = tokenizer.texts_to_sequences([caption])[0]
                    for i in range(1, len(seq)):
                        in_seq, out_seq = seq[:i], seq[i]
                        in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                        out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                        yield (features[key][0], in_seq), out_seq

    dataset = tf.data.Dataset.from_generator(
        generator,
        output_types=((tf.float32, tf.int32), tf.float32),
        output_shapes=(((4096,), (max_length,)), (vocab_size,))
    ).batch(batch_size)

    return dataset
```

Model Architecture Code:

encoder model

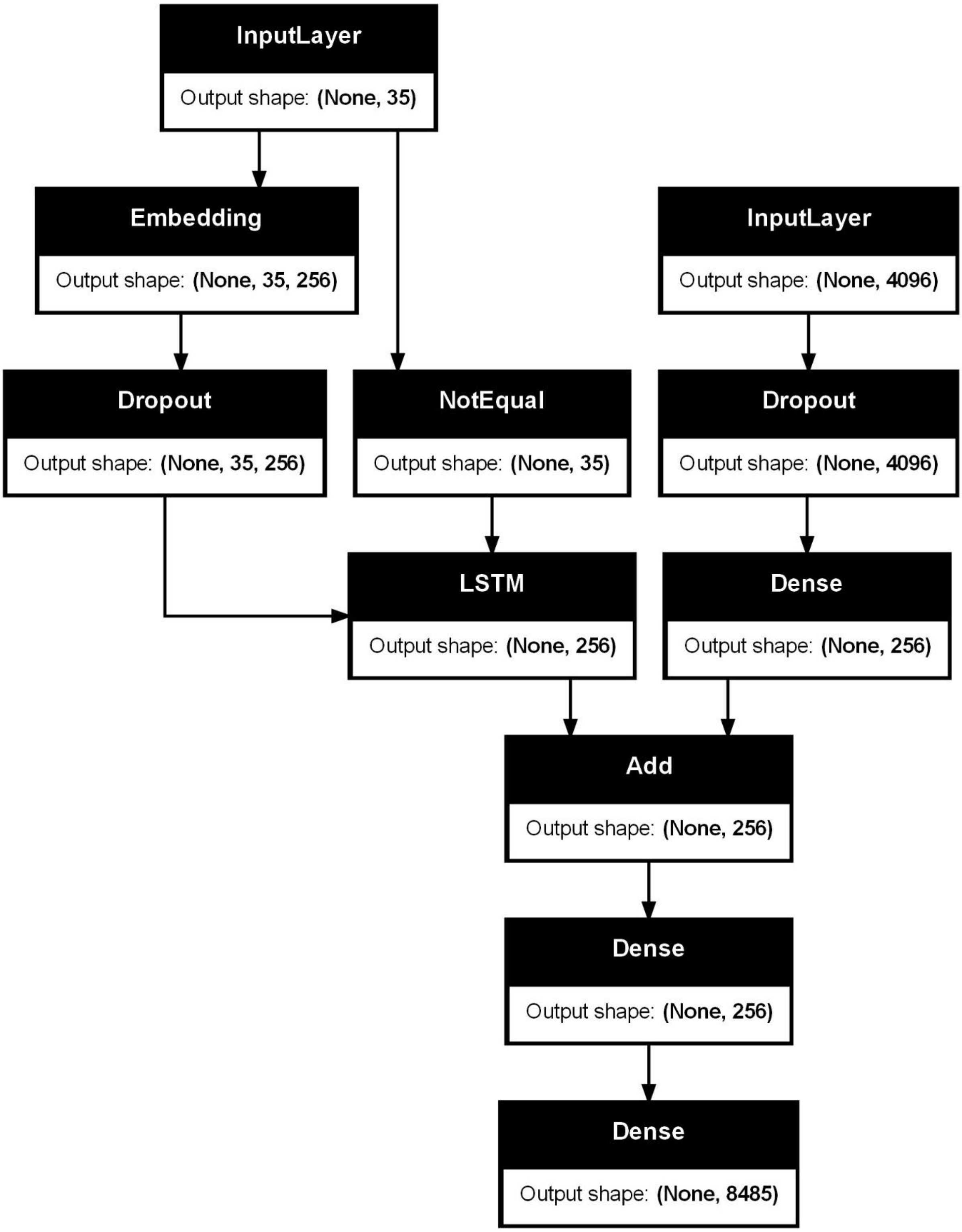
[23]:
```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)

# decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')

# plot the model
plot_model(model, show_shapes=True)
```

MODEL ARCHITECTURE



Training:

Train the model

[31]:
```python
# train the model
epochs = 150
batch_size = 64
for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size, batch_size)
    # calculate steps per epoch
    steps_per_epoch = len(train) // batch_size
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps_per_epoch, verbose=1)
```

```
113/113 ━━━━━━━━━━━━━━━━━━━━ 16s 141ms/step - loss: 0.2250
113/113 ━━━━━━━━━━━━━━━━━━━━ 16s 142ms/step - loss: 0.2248
113/113 ━━━━━━━━━━━━━━━━━━━━ 16s 144ms/step - loss: 0.2157
113/113 ━━━━━━━━━━━━━━━━━━━━ 16s 144ms/step - loss: 0.1943
113/113 ━━━━━━━━━━━━━━━━━━━━ 16s 143ms/step - loss: 0.1796
113/113 ━━━━━━━━━━━━━━━━━━━━ 17s 147ms/step - loss: 0.1745
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 162ms/step - loss: 0.1756
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 163ms/step - loss: 0.1704
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 168ms/step - loss: 0.1712
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 169ms/step - loss: 0.1686
113/113 ━━━━━━━━━━━━━━━━━━━━ 20s 176ms/step - loss: 0.1684
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 164ms/step - loss: 0.1656
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 168ms/step - loss: 0.1642
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 168ms/step - loss: 0.1697
113/113 ━━━━━━━━━━━━━━━━━━━━ 18s 158ms/step - loss: 0.1737
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 166ms/step - loss: 0.1841
113/113 ━━━━━━━━━━━━━━━━━━━━ 19s 163ms/step - loss: 0.1870
113/113 ━━━━━━━━━━━━━━━━━━━━ 16s 145ms/step - loss: 0.2152
```

store the trained model

[34]:
```python
# Save the model in the native Keras format
model.save(os.path.join(WORKING_DIR, "best200_model.keras"))
```

[56]:
```python
from keras.models import load_model
# Load the model
model = load_model(os.path.join(WORKING_DIR, "best200_model.keras"))
# Print model summary
print(model.summary())
```

Model: "functional_7"

Layer (type)	Output Shape	Param #	Connected to
input_layer_5 (InputLayer)	(None, 35)	0	-
input_layer_4 (InputLayer)	(None, 4096)	0	-
embedding_1 (Embedding)	(None, 35, 256)	2,172,160	input_layer_5[0]…
dropout_2 (Dropout)	(None, 4096)	0	input_layer_4[0]…
dropout_3 (Dropout)	(None, 35, 256)	0	embedding_1[0][0]
not_equal_2 (NotEqual)	(None, 35)	0	input_layer_5[0]…
dense_3 (Dense)	(None, 256)	1,048,832	dropout_2[0][0]
lstm_1 (LSTM)	(None, 256)	525,312	dropout_3[0][0], not_equal_2[0][0]
add_1 (Add)	(None, 256)	0	dense_3[0][0], lstm_1[0][0]
dense_4 (Dense)	(None, 256)	65,792	add_1[0][0]
dense_5 (Dense)	(None, 8485)	2,180,645	dense_4[0][0]

Total params: 17,978,225 (68.58 MB)

Trainable params: 5,992,741 (22.86 MB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 11,985,484 (45.72 MB)

None

[25]:
```python
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

Inference:

Generate caption for an image

[26]:
```
# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text
```

Evaluation:

Calculate BLEU score

[40]:
```
from nltk.translate.bleu_score import corpus_bleu

# validate with test data
actual, predicted = list(), list()

for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

# calcuate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(1.5, 1, 0, 0)))
```

Error displaying widget: model not found

```
BLEU-1: 0.426269
BLEU-2: 0.038656
```

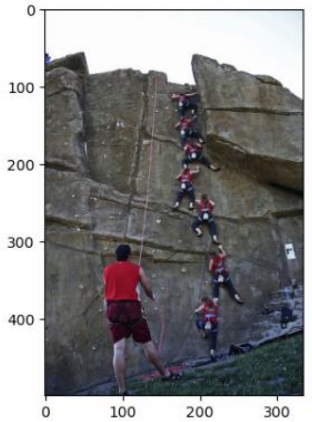
Deployment:

Caption Generate

[27]:
```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)
    image = Image.open(img_path)
    captions = mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
```

[28]:
```
generate_caption("1016887272_03199f49c4.jpg")
```

```
---------------------Actual--------------------
startseq collage of one person climbing cliff endseq
startseq group of people are rock climbing on rock climbing wall endseq
startseq group of people climbing rock while one man belays endseq
startseq seven climbers are ascending rock face whilst another man stands holding the rope endseq
startseq several climbers in row are climbing the rock while the man in red watches and holds the line endseq
--------------------Predicted-------------------
startseq collage of one person climbing cliff endseq
```



Feature Extraction with VGG16 Model:

[47]:
```
vgg_model = VGG16()
# restructure the model
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)
```

Predict new image caption:

Predict the new image caption

[48]:
```
from keras.preprocessing.image import load_img, img_to_array

image_p = input("Enter path : ").strip()  # Strip any leading or trailing whitespace
image_path = image_p.strip('"')  # Remove double quotes from the path
image = load_img(image_path, target_size=(224, 224))
# convert image pixels to numpy array
image = img_to_array(image)
# reshape data for model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# preprocess image for vgg
image = preprocess_input(image)
# extract features
feature = vgg_model.predict(image, verbose=0)


# predict from the trained model
predicted_caption=predict_caption(model, feature, tokenizer, max_length)
print("-" * 50)
# Print the predicted caption
print("Predicted Caption:", '\033[1m' + predicted_caption + '\033[0m')


# Display the image
plt.imshow(load_img(image_path))
plt.axis('on')
plt.show()
```

```
Enter path :  "C:\Users\mades\Downloads\590314-snow-winter-dog-nature-animals-running-jumping.jpg"
--------------------------------------------------
Predicted Caption: startseq brown dog chases the pool endseq
```

[44]:
```
from nltk.translate.meteor_score import meteor_score
meteor_score = meteor_score(references, predictions)
print("METEOR Score:", meteor_score)
```

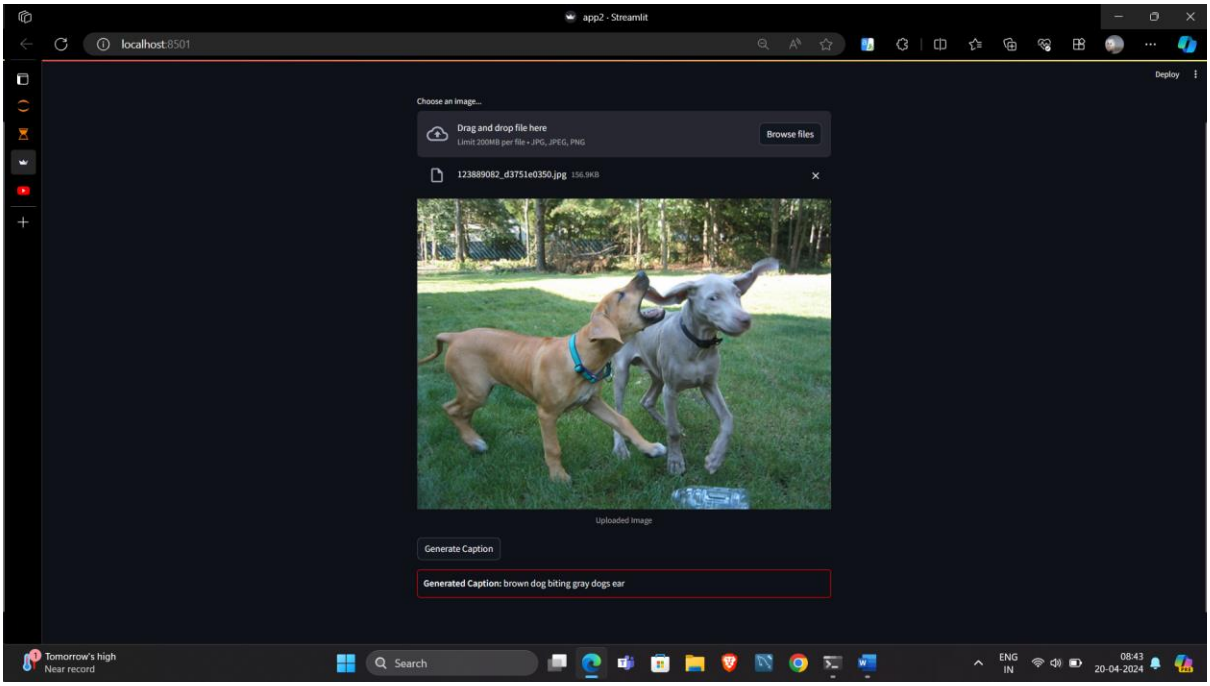
```
METEOR Score: 0.0058479532163742695
```

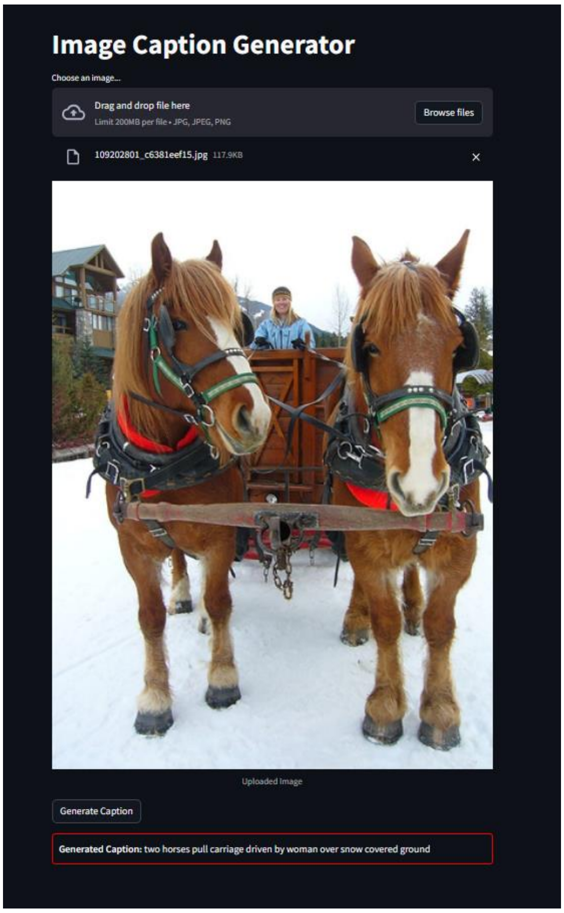
Deployment:

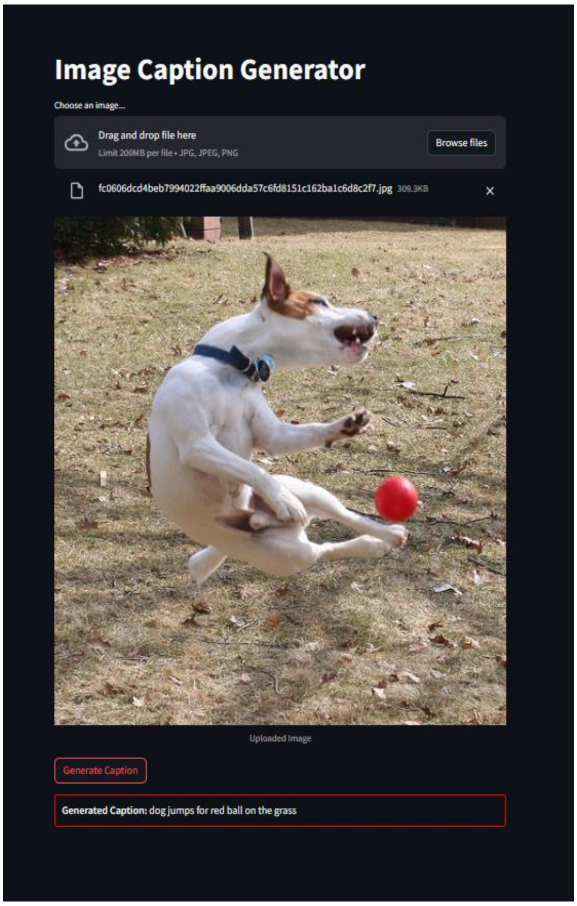
[1]:
```
!streamlit run app2.py
```

```
^C
```

Create web application:







CHAPTER - 6

SCORES

BLEU (Bilingual Evaluation Understudy):

BLEU measures the n-gram overlap between the candidate translation and reference translations, yielding scores from 0 to 1, where higher scores indicate better translations.

METEOR (Metric for Evaluation of Translation with Explicit Ordering):

METEOR considers factors like exact word matches, stemming, synonymy, and word order, also yielding scores from 0 to 1, with higher scores indicating better translations.

Metrics	Score
BLEU-1	0.423060
BLEU-2	0.036435
METEOR Score	0.0058479532163742695

CHAPTER – 7

SOURCE CODE

Libraries

```
import sklearn

import os

import pickle

from tqdm.notebook import tqdm

import cv2

import matplotlib.pyplot as plt

import matplotlib.image as mping

from tensorflow.keras.applications.vgg16 import VGG16,preprocess_input

from tensorflow.keras.preprocessing.image import load_img, img_to_array

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad_sequences

from tensorflow.keras.models import Model

from tensorflow.keras.utils import to_categorical, plot_model

from tensorflow.keras.layers import Input,Dense, LSTM, Embedding,Dropout,add
```

Directory:

```
BASE_DIR ="D:\\Data_science\\Mini Project\\image caption generater"

# %pwd

WORKING_DIR ='C:\\Users\\mades\\Documents\\Image Caption Generater'
```

Create Model:

```
# load vgg16 model

model = VGG16()

# restructure the model

model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

# summarize

print(model.summary())
```

Extract features from image:

```
# extract features from image
features = {}
directory = os.path.join(BASE_DIR, 'Images')
for img_name in tqdm(os.listdir(directory)):
    # load the image from file
    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, verbose=0)
    # get image ID
    image_id = img_name.split('.')[0]
    # store feature
    features[image_id] = feature
```

Store the features in pickle:

```
# store features in pickle
pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'), 'wb'))
```

load features from pickle:

```
# load features from pickle
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
len(features)
```

Load captions_doc:

```
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
len(captions_doc)
```

create mapping of image to captions:

```
# create mapping of image to captions
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)
```

Cleaning Function:

```
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
```

```
        caption = captions[i]
        # preprocessing steps
        # convert to lowercase
        caption = caption.lower()
        caption = caption.replace(' +', ' ')
        # delete digits, special chars, etc.,
        caption = caption.replace('[^A-Za-z]', '')
        # delete additional spaces
        caption = caption.replace('\s+', ' ')
        # add start and end tags to the caption
        caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + ' endseq'
        captions[i] = caption
```

Test to cleaning sample caption:

```
mapping['1000268201_693b08cb0e']
clean(mapping)
mapping['1000268201_693b08cb0e']
```

All_Captions:

```
all_captions = []
for key in mapping:
    for caption in mapping[key]:
        all_captions.append(caption)
len(all_captions)
```

Visualization len(Caption):

```
import matplotlib.pyplot as plt
# Assuming caption_lengths is a list containing the lengths of all captions
caption_lengths = [len(caption.split()) for caption in all_captions]
plt.hist(caption_lengths, bins=20, color='skyblue', edgecolor='black')
```

```
plt.xlabel('Caption Length (in words)')
plt.ylabel('Frequency')
plt.title('Distribution of Caption Lengths')
plt.grid(True)
plt.show()
```

Tokenize the text:

```
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions)
vocab_size = len(tokenizer.word_index) + 1
vocab_size
max_length = max(len(caption.split()) for caption in all_captions)
max_length
```

split data - train, test:

```
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
len(train)
```

create data generator to get data in batch (avoids session crash):

```
import tensorflow as tf
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size,
batch_size):
    def generator():
        while True:
            for key in data_keys:
                captions = mapping[key]
```

```
            for caption in captions:
                seq = tokenizer.texts_to_sequences([caption])[0]
                for i in range(1, len(seq)):
                    in_seq, out_seq = seq[:i], seq[i]
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
                    yield (features[key][0], in_seq), out_seq
    dataset = tf.data.Dataset.from_generator(
        generator,
        output_types=((tf.float32, tf.int32), tf.float32),
        output_shapes=(((4096,), (max_length,)), (vocab_size,))
    ).batch(batch_size)

    return dataset
```

Encoder Model:

```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3 = LSTM(256)(se2)
# decoder model
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)
```

```
model = Model(inputs=[inputs1, inputs2], outputs=outputs)
model.compile(loss='categorical_crossentropy', optimizer='adam')
# plot the model
plot_model(model, show_shapes=True)
```

Train the model:

```
# train the model
epochs = 150
batch_size = 64
for i in range(epochs):
    # create data generator
    generator = data_generator(train, mapping, features, tokenizer, max_length, vocab_size,
batch_size)
    # calculate steps per epoch
    steps_per_epoch = len(train) // batch_size
    # fit for one epoch
    model.fit(generator, epochs=1, steps_per_epoch=steps_per_epoch, verbose=1)
```

Save Model:

```
# Save the model in the native Keras format
model.save(os.path.join(WORKING_DIR, "best200_model.keras"))
```

Load saved model:

```
from keras.models import load_model
# Load the model
model = load_model(os.path.join(WORKING_DIR, "best200_model.keras"))
# Print model summary
print(model.summary())
```

Word Index to Word Conversion Function:

```
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

Generate caption for an image:

```
# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
```

```
        if word == 'endseq':
            break
    return in_text
```

Calculate BLEU score:

```
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = list(), list()
for key in tqdm(test):
    # get actual caption
    captions = mapping[key]
    # predict the caption for image
    y_pred = predict_caption(model, features[key], tokenizer, max_length)
    # split into words
    actual_captions = [caption.split() for caption in captions]
    y_pred = y_pred.split()
    # append to the list
    actual.append(actual_captions)
    predicted.append(y_pred)

# calcuate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(1.5, 1, 0, 0)))
```

Caption Generate:

```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
```

```
    img_path = os.path.join(BASE_DIR, "Images", image_name)
    image = Image.open(img_path)
    captions = mapping[image_id]
    print('---------------------Actual---------------------')
    for caption in captions:
        print(caption)
    # predict the caption
    y_pred = predict_caption(model, features[image_id], tokenizer, max_length)
    print('--------------------Predicted--------------------')
    print(y_pred)
    plt.imshow(image)
```

sample image:

```
generate_caption("1016887272_03199f49c4.jpg")
```

Feature Extraction with VGG16 Model:

```
vgg_model = VGG16()
# restructure the model
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)
```

Create Streamlit web Application Image Caption Generation:

```
# Libraries
import pandas as pd
import numpy as np
import os
import pickle
from tqdm.notebook import tqdm
from PIL import Image
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import img_to_array
```

```
from tensorflow.keras.models import Model
import streamlit as st
import unicodedata
from tensorflow.keras.models import load_model
# Libraries
import cv2
import matplotlib.pyplot as plt
import matplotlib.image as mping
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input,Dense, LSTM, Embedding,Dropout,add
# Load the model and tokenizer
BASE_DIR = "D:\\Data_science\\Mini Project\\image caption generater"
WORKING_DIR = 'C:\\Users\\mades\\Documents\\Image Caption Generater'

# Load features from pickle
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
# create mapping of image to captions
mapping = {}
# process lines
for line in tqdm(captions_doc.split('\n')):
    # split the line by comma(,)
    tokens = line.split(',')
```

```
    if len(line) < 2:
        continue
    image_id, caption = tokens[0], tokens[1:]
    # remove extension from image ID
    image_id = image_id.split('.')[0]
    # convert caption list to string
    caption = " ".join(caption)
    # create list if needed
    if image_id not in mapping:
        mapping[image_id] = []
    # store the caption
    mapping[image_id].append(caption)

def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

 # generate caption for an image

```
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
```

```
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break
    return in_text
#model
model = load_model(os.path.join(WORKING_DIR, "best3_model.keras"))
#tokenizer
with open("tokenizer.pkl", "rb") as f:
    tokenizer = pickle.load(f)
```

#streamlit application main page:

```
def main():
    st.title("Image Caption Generator")
    uploaded_file = st.file_uploader("Choose an image...", type=["jpg", "jpeg", "png"])
    if uploaded_file is not None:
        # Display the uploaded image
        image = Image.open(uploaded_file)
        st.image(image, caption='Uploaded Image', use_column_width=True)
        # Generate and display caption on button click
        if st.button('Generate Caption'):
```

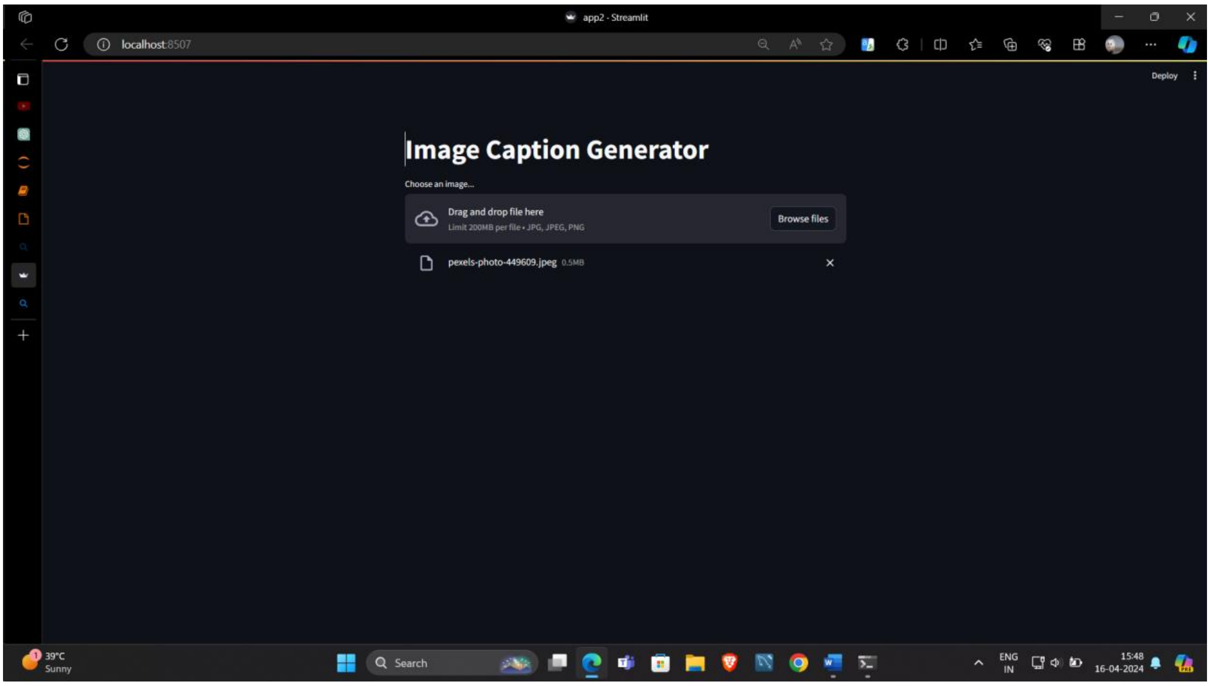
```
        # Resize the image to match the input shape of the VGG16 model
        image = image.resize((224, 224))
        # Convert the image to array and preprocess it
        image_array = img_to_array(image)
        image_array = image_array.reshape((1, image_array.shape[0], image_array.shape[1],
image_array.shape[2]))
        image_array = preprocess_input(image_array)
        # Extract features from the image using the VGG16 model
        vgg_model = VGG16()
        vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)
        feature = vgg_model.predict(image_array, verbose=0)
        # Predict caption using the captioning model
        caption = predict_caption(model, feature, tokenizer, max_length=35)
        # Print the caption without startseq and endseq
        # Print the caption
        st.markdown(f"<div style='border: 2px solid red; border-radius: 5px; padding:
10px;'><b>Generated Caption:</b> {' '.join(caption.split()[1:-1])}</div>",
unsafe_allow_html=True)
```

 # Run the app

```
if __name__ == "__main__":
    main()
```

#Jupyter Notebook commend:

!streamlit run app2.py



CHAPTER - 8

CONCLUSION

In conclusion, the Image Caption Generator project represents a significant advancement in leveraging deep learning for automating the generation of descriptive captions for images. Through the utilization of frameworks like TensorFlow or PyTorch and libraries such as OpenCV and NLTK or spaCy, we have constructed a robust system capable of analyzing images and producing human-like captions. By adhering to the specified system requirements and prioritizing user-friendly interfaces, including the development of a Streamlit application, we have ensured accessibility and ease of use. This project holds promise for various applications, from enhancing image retrieval systems to aiding individuals with visual impairments. With continued refinement and deployment, the Image Caption Generator stands poised to make meaningful contributions across diverse domains, demonstrating the efficacy of deep learning in addressing real-world challenges.

Web link: streamlit app2.py

BIBLIOGRAPHY

Reference:

1. Elgendy, Mohamed. (2018). Deep Learning for Vision Systems. Packt Publishing.[Link to Book]

2. Richard Szeliski. (2010). Computer Vision: Algorithms and Applications. Springer. [Link to Book]

3. Foster, David. (2019). Generative Deep Learning: Teaching Machines to Paint, Write, Compose, and Play. O'Reilly Media. [Link to Book]

4. Simard, P., Vezhnevets, A., & Netzer, Y. (2018). Image Captioning with Deep Learning. Microsoft Research. [Link to Book].

Dataset:

https://www.kaggle.com/datasets/adityajn105/flickr8k.