Project Report Format

TABLE OF CONTENTS:

13.	APPENDIX Source Code GitHub & Project Demo Link

1. INTRODUCTION

1.1 Project Overview

The main goal of this project is to develop a system that can automatically generate descriptive captions for images. This involves leveraging machine learning and computer vision techniques to understand the content of an image and generate coherent and contextually relevant captions.

For this reason caption generation has long been viewed as a difficult problem. It is very important challenge for machine learning algorithms, as it amounts to mimicking the remarkable human ability to compress huge amounts of salient visual information into descriptive language.

1.2 Purpose

Image captions make visual content accessible to individuals with visual impairments, allowing them to understand and engage with images through text descriptions. Image captions contribute to user engagement on social media platforms by providing context and encouraging discussions around shared images.

2. LITERATURE SURVEY

2.1 Existing problem

1. Generating captions that accurately reflect the broader context of an image, including relationships between objects, is challenging.

2. Images with ambiguous or complex content may lead to inaccurate or unclear captions.

3. Some image caption generators may not be optimized for real-time processing, especially when dealing with large datasets or high-resolution images.

2.2 References

1. HaoranWang An Overview of Image Caption Generation Methods, (CIN-2020)

2. B.Krishnakumar, K.Kousalya, S.Gokul, R.Karthikeyan, and D.Kaviyarasu, IMAGE CAPTION GENERATOR USING DEEP LEARNING, (international Journal of Advanced Science and Technology- 2020)

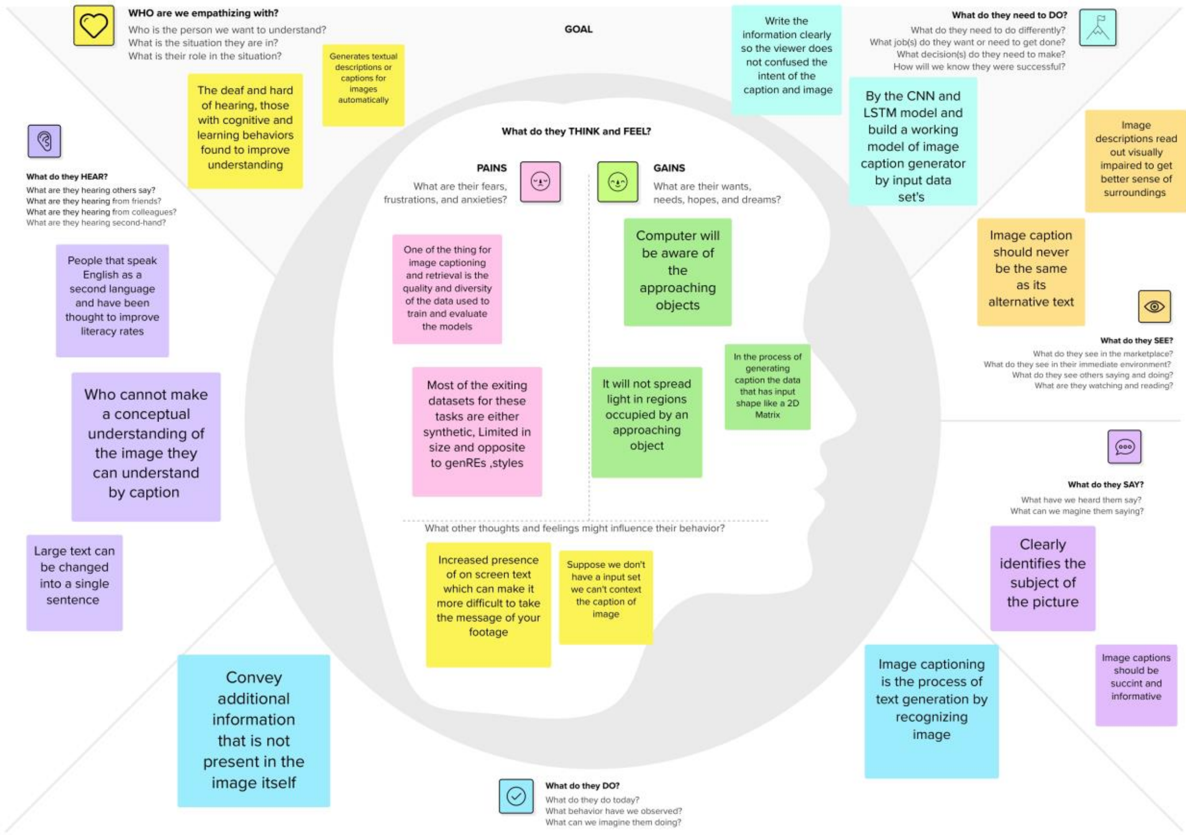
3. MD. Zakir Hossain and Hamid Laga, A Comprehensive Survey of Deep Learning for Image Captioning,(ACM-2019)

2.3 Problem Statement Definition

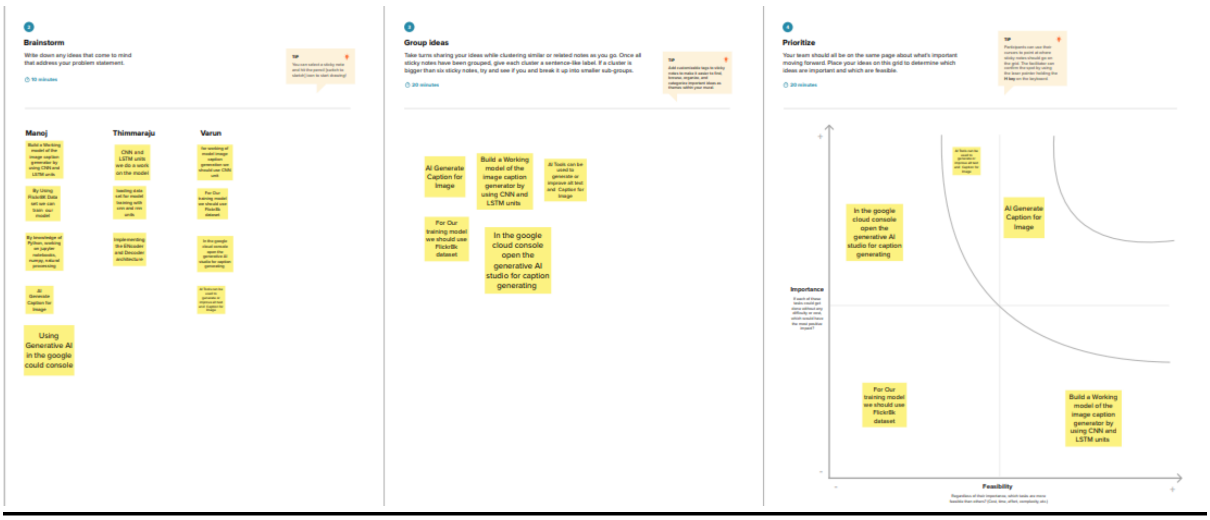
The task of automatic image caption generation poses significant challenges in achieving accurate and contextually relevant descriptions for diverse visual content. Current image caption generators often face issues in handling ambiguity, understanding complex relationships within images, and addressing biases present in training data. Additionally, the lack of adaptability to different domains, limitations in creativity, and real-time processing challenges hinder the effectiveness of existing models.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



4. REQUIREMENT ANALYSIS

4.1 Functional requirements

1. Image Pre-processing
2. Feature Extraction
3. Sequence-to-Sequence Model
4. Natural Language Processing (NLP)
5. User Interface
6. Training and Evaluation

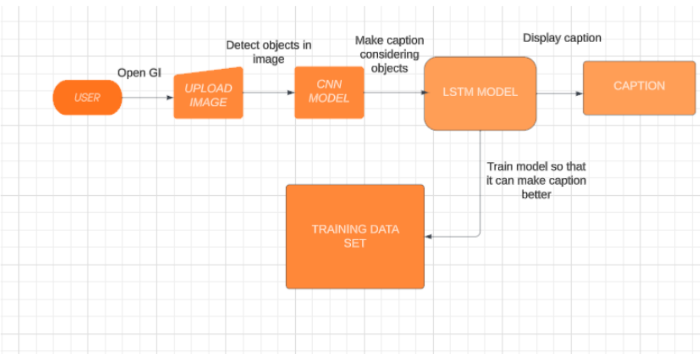
4.2 Non-Functional requirements

1. Performance
2. Scalability
3. Accuracy
4. Usability
5. Security

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

Data Flow Diagrams:

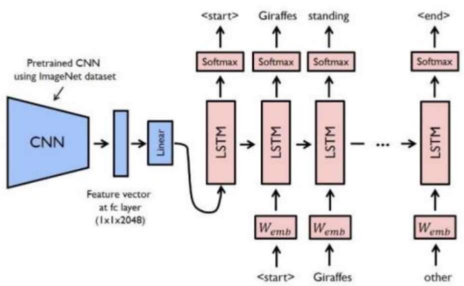


User Stories

User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Assistance for Visually Impaired	Well-designed automotive AI	USN-1	When we input the image the AI will Generate the caption	Trained Dataset	High	Sprint-1
Deaf People	Model development	USN-2	While seeing the image they can understand the image without asking others	We Could prepare these models CNN and LSTM	High	Sprint-1
Social Media Posts	Matter Recognition	USN-3	This free AI powered social media caption generator will create the perfect caption for your photo.	Importing into Social Media	Low	Sprint-2
authenticate your image	Testing and quality Assurance	USN-4	The user or computer has to prove its identity to the server or client	Exploring the input Machine models	Medium	Sprint-3
Reduce road accidents	Well-designed automotive AI	USN-5	By installing an image caption generator in the vehicles, vehicles can stop by applying the automatic brake when an object in the surrounding is detected	Testing the Model with packages	High	Sprint-4

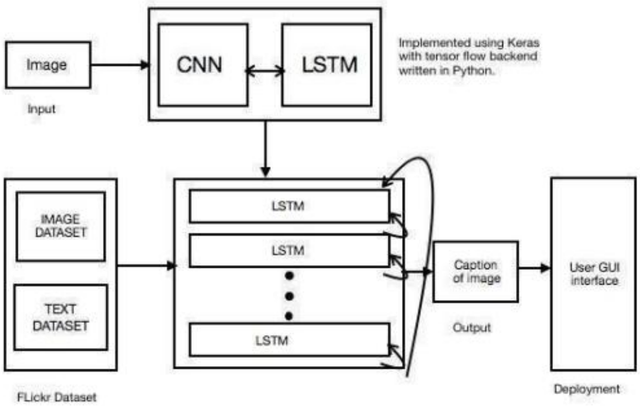
5.2 Solution Architecture

Example - Solution Architecture Diagram:



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture



6.2 Sprint Planning & Estimation

Sprint	Functional Requirement (Epic)	User Story Number	User Story / Task	Story Points	Priority	Team Members
Sprint-1	Well-designed automotive AI	USN-1	When we input the image the AI will Generate the caption	1	High	Raju
Sprint-1	Model development	USN-2	train the selected deep learning model using the preprocessed Dataset and monitor its performance on the validation set.	2	High	Manoj
Sprint-2	Matter Recognition	USN-3	This free AI powered social media caption Generator will create the perfect caption for your photo.	2	Low	Manoj,Varun
Sprint-3	Data collection	USN-4	Preprocess the collected dataset by resizing images, normalizing pixel values, and splitting it into training and validation sets	3	Medium	Varun
Sprint-4	Testing and quality Assurance	USN-5	. Explore and evaluate different deep learning architectures (e.g., CNNs)	4	High	Raju

6.3 Sprint Delivery Schedule

Sprint	Total Story Points	Duration	Sprint Start Date	Sprint End Date (Planned)	Story Points Completed (as on Planned End Date)	Sprint Release Date (Actual)
Sprint-1	3	3 Days	10 Oct 2023	13 Oct 2023	12	13 Oct 2023
Sprint-2	2	2 Days	15 Oct 2023	17 Oct 2023		
Sprint-3	3	7 Days	18 Oct 2023	27Oct 2023		
Sprint-4	4	6 Days	29 Oct 2023	5 Nov 2023		

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

Project Structure:

1. Create a Project folder which contains files as shown below

.ipynb_checkpoints	06-11-2023 14:47	File folder	
Flicker8k_Datadet	06-11-2023 15:11	File folder	
Flickr_8k_text	06-11-2023 15:01	File folder	
models	06-11-2023 14:49	File folder	
descriptions	06-11-2023 14:51	Text Document	3,072 KB
features.p	06-11-2023 14:52	P File	65,477 KB
model	06-11-2023 14:53	PNG File	48 KB
testing_caption_generator	06-11-2023 14:53	Python Source File	3 KB
tokenizer.p	06-11-2023 14:53	P File	286 KB
training_caption_generator	06-11-2023 14:53	Jupyter Source File	33 KB

2. The Dataset folder contains the training and testing images for training our model.
3. We need the model which is saved as model.h5 and the captions as tokenizer.pkl the templates folder contains index.html and prediction.html pages.

Step-1:

Data Collection

We can download the data from kaggle website, there are nearly 8000 images associated with the 5 captions for each image. The given dataset has 40000 high quality human readable text captions. After downloading datasets you should create a folder and insert datasets into folder as Flicker8k_Dataset and Flickr_8k_text.

Step-2:

Data Pre-processing

Clean the text captions and mapping each together. Then it's time to build our Vgg16 model which contains an input layer CNN model and the LSTM model.

Task-1

First we have to import all the necessary packages

```
import string
import numpy as np
from PIL import Image
import os
from pickle import dump, load
import numpy as np

from keras.applications.xception import Xception, preprocess_input
from keras.preprocessing.image import load_img, img_to_array
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.layers.merge import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout

# small library for seeing the progress of loops.
from tqdm import tqdm_notebook as tqdm
tqdm().pandas()
```

Task-2

Getting and performing data cleaning

File Edit Format Run Options Window Help


```
{
'3461437556_cc5e97f3ac.jpg': ['dogs on grass',
                              'three dogs are running on the grass',
                              'three dogs one white and two brown are running together
                              'three dogs run along grassy yard',
                              'three dogs run together in the grass'
                              ],

'3461583471_2b8b6b4d73.jpg': ['buy is grinding rail on snowboard',
                              'person is jumping ramp on snowboard',
                              'snowboarder goes down ramp',
                              'snowboarder going over ramp',
                              'snowboarder performs jump on the clean white snow'
                              ],
'997722733_0cb5439472.jpg' : ['man in pink shirt climbs rock face',
                              'man is rock climbing high in the air',
                              'person in red shirt climbing up rock face covered in as
                              'rock climber in red shirt',
                              'rock climber practices on rock climbing wall'
                              ]

}
```


This function takes all descriptions and performs data cleaning. This is an important step when we work with textual data, according to our goal, we decide what type of cleaning we want to perform on the text. In our case, we will be removing punctuations it will converting all text to lowercase and removing words that contain numbers. So, a caption like “A man riding on a three-wheeled wheelchair” will be transformed into “man riding on three wheeled wheelchair”.

```python
# get all imgs with their captions
def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    descriptions ={}
    for caption in captions[:-1]:
        img, caption = caption.split('\t')
        if img[:-2] not in descriptions:
            descriptions[img[:-2]] = [caption]
        else:
            descriptions[img[:-2]].append(caption)
    return descriptions


##Data cleaning- lower casing, removing puntuations and words containing numbers
def cleaning_text(captions):
    table = str.maketrans('','',string.punctuation)
    for img,caps in captions.items():
        for i,img_caption in enumerate(caps):

            img_caption.replace("-"," ")
            desc = img_caption.split()

            #converts to lower case
            desc = [word.lower() for word in desc]
            #remove punctuation from each token
            desc = [word.translate(table) for word in desc]
            #remove hanging 's and a
            desc = [word for word in desc if(len(word)>1)]
            #remove tokens with numbers in them
            desc = [word for word in desc if(word.isalpha())]
            #convert back to string
```

Task-3

Extracting the features

This technique is also called transfer learning, we don't have to do everything on our own, and we use the pre-trained model that have been already trained on large datasets and extract the features from these models and use them for our tasks. We are using the Xception model which has been trained on imagenet dataset that had 1000 different classes to classify. We can directly import this model from the keras.applications . Make sure you are connected to the internet as the weights get automatically downloaded. Since the Xception model was originally built for imagenet, we will do little changes for integrating with our model. One thing to notice is that the Xception model takes 299*299*3 image size as input. We will remove the last classification layer and get the 2048 feature vector.

```python
def extract_features(directory):
        model = Xception( include_top=False, pooling='avg' )
        features = {}
        for img in tqdm(os.listdir(directory)):
            filename = directory + "/" + img
            image = Image.open(filename)
            image = image.resize((299,299))
            image = np.expand_dims(image, axis=0)
            #image = preprocess_input(image)
            image = image/127.5
            image = image - 1.0

            feature = model.predict(image)
            features[img] = feature
        return features
```

```python
#2048 feature vector
features = extract_features(dataset_images)
dump(features, open("features.p","wb"))
```

Task-4

Loading dataset for Training the model

In our Flickr_8k_test folder, we have Flickr_8k.trainImages.txt file that contains a list of 6000 image names that we will use for training. This function will create a dictionary that contains captions for each photo from the list of photos.

```python
#load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos


def load_clean_descriptions(filename, photos):
    #loading clean_descriptions
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):

        words = line.split()
        if len(words)<1 :
            continue

        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions


def load_features(photos):
    #loading all features
    all_features = load(open("features.p","rb"))
    #selecting only needed features
    features = {k:all_features[k] for k in photos}
    return features
```

```python
filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"

#train = loading_data(filename)
train_imgs = load_photos(filename)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)
```

```python
#converting dictionary to clean list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

#creating tokenizer class
#this will vectorise text corpus
#each integer will represent token in dictionary

from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer
```

Task-5

Tokenizing the vocabulary

Computers don't understand English words, for computers, we will have to represent them with numbers. So, we will map each word of the vocabulary with a unique index value. Keras library provides us with the tokenizer function that we will use to create tokens from our vocabulary and save them to a "tokenizer.p" pickle file.

```python
# give each word a index, and store that into tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.p', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

```
7577
```

```python
#calculate maximum length of descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
max_length
```

```
32
```

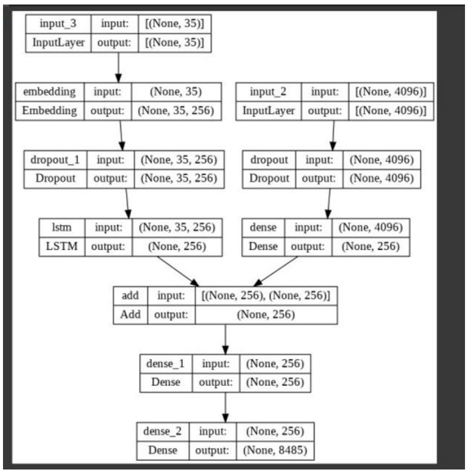
Task-6

Data Generation

In Flicker8k-datset we are having lot of images approximately 8000 so we are creating a data generation where the images can store into memory it so we can run our model by images and description captions.

```
#data generator, used by model.fit_generator()
def data_generator(descriptions, features, tokenizer, max_length):
    while 1:
        for key, description_list in descriptions.items():
            #retrieve photo features
            feature = features[key][0]
            input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
            yield [[input_image, input_sequence], output_word]

def create_sequences(tokenizer, max_length, desc_list, feature):
    X1, X2, y = list(), list(), list()
    # walk through each description for the image
    for desc in desc_list:
        # encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # split one sequence into multiple X,y pairs
        for i in range(1, len(seq)):
            # split into input and output pair
            in_seq, out_seq = seq[:i], seq[i]
            # pad input sequence
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            # encode output sequence
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            # store
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
```

By merging the output from the above two layers, we will process by the dense layer to make the final prediction. The final layer will contain the number of nodes equal to our vocabulary size.



```python
from keras.utils import plot_model

# define the captioning model
def define_model(vocab_size, max_length):

    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

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

    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)

    return model
```

Step-3:

Model Building

Now we can train our image data set

Train the model

To train the model, we will be using the 6000 training images by generating the input and output sequences in batches and fitting them to the model using model.fit_generator() method. We also save the model to our models folder. This will take some time depending on your system capability.

```python
# train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train=', len(train_descriptions))
print('Photos: train=', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)

model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
    model.fit_generator(generator, epochs=1, steps_per_epoch= steps, verbose=1)
    model.save("models/model_" + str(i) + ".h5")
```

```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32

Layer (type)                    Output Shape         Param #     Connected to
==================================================================================
input_2 (InputLayer)            (None, 32)           0

input_1 (InputLayer)            (None, 2048)         0

embedding_1 (Embedding)         (None, 32, 256)      1939712     input_2[0][0]

dropout_1 (Dropout)             (None, 2048)         0           input_1[0][0]

dropout_2 (Dropout)             (None, 32, 256)      0           embedding_1[0][0]

dense_1 (Dense)                 (None, 256)          524544      dropout_1[0][0]

lstm_1 (LSTM)                   (None, 256)          525312      dropout_2[0][0]

add_1 (Add)                     (None, 256)          0           dense_1[0][0]
                                                                 lstm_1[0][0]

...
Trainable params: 5,002,649
Non-trainable params: 0
```

Step-4:

Testing the model

While we are testing the model we should check whether it is perfectly fit into to the model or not. The model has been trained, now, we will make a separate file testing_caption_generator.py which will load the model and can generate the predictions.

D: > Project- image caption generator > testing_caption_generator.py > extract_features

```python
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import Xception
from keras.models import load_model
from pickle import load
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt
import argparse


ap = argparse.ArgumentParser()
ap.add_argument('-i', '--image', required=True, help="Image Path")
args = vars(ap.parse_args())
img_path = args['image']

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def extract_features(filename, model):
        try:
            image = Image.open(filename)

        except:
            print("ERROR: Couldn't open image! Make sure the image path and extension is correct")
        image = image.resize((299,299))
        image = np.array(image)
        # for images that has 4 channels, we convert them into 3 channels
        if image.shape[2] == 4:
            image = image[..., :3]
        image = np.expand_dims(image, axis=0)
        image = image/127.5
        image = image - 1.0
        feature = model.predict(image)
        return feature

Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def word_for_id(integer, tokenizer):
 for word, index in tokenizer.word_index.items():
```

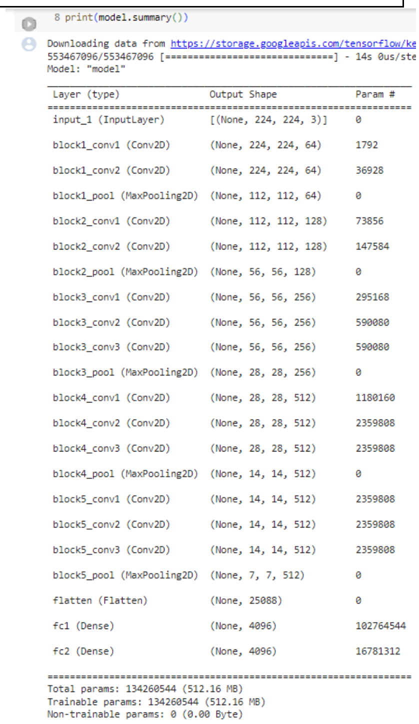
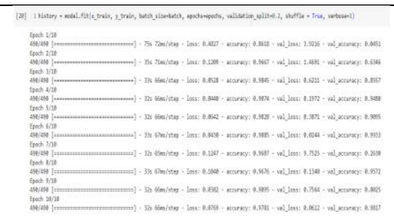
training_caption_generator.ipynb ● testing_caption_generator.py 7 ✕

D: > Project- image caption generator > testing_caption_generator.py > extract_features

```python
     if index == integer:
         return word
 return None


Explain Code | Generate Tests | Generate Docstrings | Ask Sourcery
def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo,sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text


#path = 'Flicker8k_Dataset/111537222_07e56d5a30.jpg'
max_length = 32
tokenizer = load(open("tokenizer.p","rb"))
model = load_model('models/model_9.h5')
xception_model = Xception(include_top=False, pooling="avg")

photo = extract_features(img_path, xception_model)
img = Image.open(img_path)

description = generate_desc(model, tokenizer, photo, max_length)
print("\n\n")
print(description)
plt.imshow(img)
```

8. PERFORMANCE TESTING

8.1 Performance Metrics

S.No.	Parameter	Values	Screenshot
1.	Model Summary	Total params: 134260544 Trainable params: 134260544 Non-trainable params: 0	
2.	Accuracy	Training Accuracy - 97.81% Validation Accuracy – 98.17%	
3.	Confidence Score (Only Yolo Projects)	Class Detected - NA Confidence Score - NA	NOT APPLICABLE

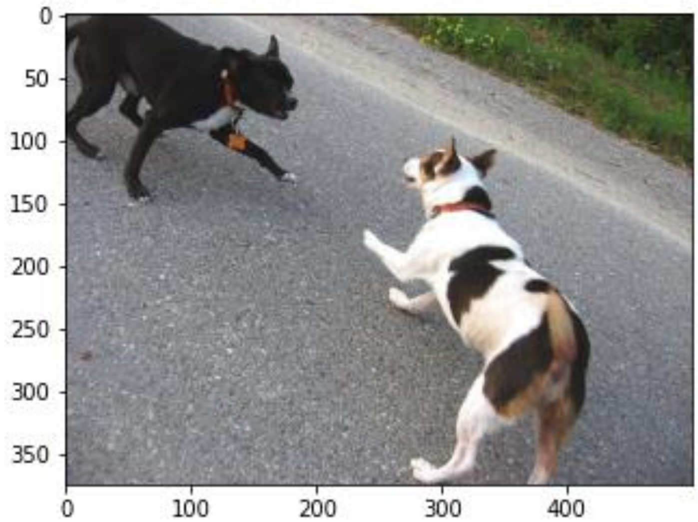
```
8 print(model.summary())

Downloading data from https://storage.googleapis.com/tensorflow/ke
553467096/553467096 [==============================] - 14s 0us/ste
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: 134260544 (512.16 MB)
Trainable params: 134260544 (512.16 MB)
Non-trainable params: 0 (0.00 Byte)
```

9. RESULTS

9.1 Output Screenshots

startseq two children are sitting in the middle of the rainbow endseq

startseq two dogs play with each other on the sidewalk endseq



startseq man displaying paintings in the snow endseq

10. ADVANTAGES & DISADVANTAGES

ADVANTAGES

- Captions contribute to search engine optimization efforts by providing textual content associated with images, potentially improving the discoverability and ranking of visual content in search results.
- Image captions serve as valuable educational tools by providing additional information and context, aiding in understanding and learning about visual content.
- Image caption generators can be used in various creative projects, such as generating captions for art pieces, creating image-based stories, or developing interactive experiences.
- Image captions make visual content accessible to individuals with visual impairments, allowing them to comprehend and engage with images through text descriptions.

DISADVANTAGES

- Caption generators may not always accurately describe complex or nuanced images, leading to misleading or incorrect captions.
- Training and deploying sophisticated image caption generators can be resource-intensive in terms of computational power and storage, making them less accessible for smaller-scale applications.
- Image caption generators may struggle to handle ambiguity in visual scenes, leading to generic or unclear captions, especially in images with multiple interpretations.
- Models may over fit to specific patterns present in the training data, resulting in poor generalization to unseen or diverse images.
- The performance of caption generators heavily relies on the quality and diversity of the training data. Inadequate or biased datasets can lead to poor generalization.

11. CONCLUSION

In conclusion, image caption generators represent a powerful intersection of computer vision and natural language processing, offering both significant benefits and challenges. The ability to automatically generate descriptive captions for images has wide-ranging applications, from enhancing accessibility and user experience to contributing to SEO and educational tools.

As technology advances, image caption generators have the potential to play a crucial role in shaping how we interact with visual content online, fostering inclusivity, and contributing to the evolution of AI-driven narrative generation. The journey continues towards more accurate, adaptable, and creative image captioning systems that can truly understand and describe the diverse and intricate world of visual information.

12. FUTURE SCOPE

Future scope for image caption generators is promising, with ongoing research and advancements likely to address current challenges and open up new possibilities. Future research may explore multimodal approaches that combine information from both images and text to improve caption generation. This could involve incorporating textual context into the image captioning process for more coherent and context-aware outputs. Efforts to optimize image caption generators for real-time processing will likely continue. This is particularly important for applications that require immediate interaction, such as live streaming or augmented reality.

The development of standardized benchmarks and evaluation metrics will likely continue to be important for assessing the performance of image caption generators consistently. This will help researchers compare models and track progress in the field. The future of image caption generators is dynamic and holds the potential for breakthroughs that could significantly enhance their capabilities, making them more accurate, adaptable, and valuable across various applications. Ongoing interdisciplinary research and collaboration will play a crucial role in shaping this exciting future.

13. APPENDIX

Source Code GitHub

```
!mkdir ~/.kaggle
! cp kaggle.json ~/.kaggle/
! chmod 600 ~/.kaggle/kaggle.json
```

```
mkdir: cannot create directory '/root/.kaggle': File exists
```

In [3]:

```
!kaggle datasets download -d adityajn105/flickr8k
```

```
Downloading flickr8k.zip to /content
 99% 1.02G/1.04G [00:05<00:00, 288MB/s]
100% 1.04G/1.04G [00:05<00:00, 199MB/s]
```

In [4]:

```
!unzip flickr8k.zip -d flickr8k
```

Streaming output truncated to the last 5000 lines.

In [5]:

```
import os    # handling the files
import pickle # storing numpy features
import numpy as np
from tqdm.notebook import tqdm # how much data is process till now
```

```
from tensorflow.keras.applications.vgg16 import VGG16 , preprocess_input #
extract features from image data.
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
```

In [12]:

```
BASE_DIR = '/content/flickr8k'
WORKING_DIR = '/content/sample_data/working'
```

In [8]:

```
# Load vgg16 Model
model = VGG16()

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

# Summerize
print(model.summary())
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-appli
cations/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 2s 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: 134260544 (512.16 MB)
Trainable params: 134260544 (512.16 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

In [9]:

```
# 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
```

```
  0%|          | 0/8091 [00:00<?, ?it/s]
```

In [31]:

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

```
---------------------------------------------------------------------------
EOFError                                  Traceback (most recent call last)
<ipython-input-31-7e165beb69cf> in <cell line: 2>()
      1 # load features from pickle
      2 with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
----> 3     features = pickle.load(f)
```

```
EOFError: Ran out of input
```

In [14]:

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

In [15]:

```
# 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)
```

```
  0%|          | 0/40456 [00:00<?, ?it/s]
```

In [16]:

```
len(mapping)
```

Out[16]:

```
8091
```

Preprocess Text Data

In [17]:

```
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()
            # 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
```

In [18]:

```
# before preprocess of text
mapping['1000268201_693b08cb0e']
```

Out[18]:

```
['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 .']
```

In [19]:

```
# preprocess the text
clean(mapping)
```

In [20]:

```
# after preprocess of text
mapping['1000268201_693b08cb0e']
```

Out[20]:

```
['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']
```

Next we will store the preprocessed captions into a list

In [21]:

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

In [22]:

```
len(all_captions)
```

Out[22]:

```
40455
```

In [23]:

```
all_captions[:10]
```

Out[23]:

```
['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 roa
d endseq',
 'startseq black dog and white dog with brown spots are staring at each oth
er in the street endseq',
 'startseq two dogs of different breeds looking at each other on the road e
ndseq',
 'startseq two dogs on pavement moving toward each other endseq']
```

Processing of Text Data

Now we start processing the text data

In [24]:

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

In [25]:

```
vocab_size
```

Out[25]:

```
8485
```

In [26]:

```
# get maximum length of the caption available
max_length = max(len(caption.split()) for caption in all_captions)
max_length
```

Out[26]:

```
35
```

Train Test Split

In [27]:

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

In [28]:

```
# create data generator to get data in batch (avoids session crash)
def data_generator(data_keys, mapping, features, tokenizer, max_length, 
vocab_size, batch_size):
    # loop over images
    X1, X2, y = list(), list(), list()
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence
                seq = tokenizer.texts_to_sequences([caption])[0]
                # split the sequence into X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pairs
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = 
to_categorical([out_seq],num_classes=vocab_size)[0]
                    # store the sequences
                    X1.append(features[key][0])
                    X2.append(in_seq)
                    y.append(out_seq)
```

```
            if n == batch_size:
                X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                yield [X1, X2], y
                X1, X2, y = list(), list(), list()
                n = 0
```

Padding sequence normalizes the size of all captions to the max size filling them with zeros for better results.

Model Creation

In [29]:

```
# 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)
```

Out[29]:

Train Model

Now let us train the model

In [30]:

```
# train the model
epochs = 20
batch_size = 32
steps = len(train) // batch_size

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

```
227/227 [==============================] - 75s 293ms/step - loss: 5.2128
```

```
227/227 [==============================] - 61s 269ms/step - loss: 4.0026
227/227 [==============================] - 62s 271ms/step - loss: 3.5828
227/227 [==============================] - 63s 275ms/step - loss: 3.3219
227/227 [==============================] - 61s 270ms/step - loss: 3.1231
227/227 [==============================] - 61s 269ms/step - loss: 2.9730
227/227 [==============================] - 60s 265ms/step - loss: 2.8588
227/227 [==============================] - 61s 267ms/step - loss: 2.7637
227/227 [==============================] - 60s 262ms/step - loss: 2.6807
227/227 [==============================] - 60s 265ms/step - loss: 2.6081
227/227 [==============================] - 59s 261ms/step - loss: 2.5452
227/227 [==============================] - 61s 270ms/step - loss: 2.4951
227/227 [==============================] - 62s 272ms/step - loss: 2.4450
227/227 [==============================] - 58s 253ms/step - loss: 2.3986
227/227 [==============================] - 60s 264ms/step - loss: 2.3576
227/227 [==============================] - 62s 272ms/step - loss: 2.3198
227/227 [==============================] - 61s 270ms/step - loss: 2.2823
227/227 [==============================] - 61s 266ms/step - loss: 2.2496
227/227 [==============================] - 63s 275ms/step - loss: 2.2198
227/227 [==============================] - 64s 283ms/step - loss: 2.1910
```

In [32]:

```python
# save the model
model.save(WORKING_DIR+'/best_model.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3079:
UserWarning: You are saving your model as an HDF5 file via `model.save()`.
This file format is considered legacy. We recommend using instead the nativ
e Keras format, e.g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Generate Captions for the Image

In [33]:

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

In [34]:

```python
# 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
```

Visualize the Results

In [35]:

```
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)
```

- Image caption generator defined
- First prints the actual captions of the image then prints a predicted caption of the image

In [36]:

```
generate_caption("1001773457_577c3a7d70.jpg")
```

```
---------------------Actual---------------------
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 end
seq
startseq two dogs on pavement moving toward each other endseq
--------------------Predicted--------------------
startseq two dogs are playing with toy in the grass endseq
```

In [37]:

```
generate_caption("1002674143_1b742ab4b8.jpg")
```

```
---------------------Actual---------------------
startseq little girl covered in paint sits in front of painted rainbow with
her hands in bowl endseq
startseq little girl is sitting in front of large painted rainbow endseq
```

```
startseq small girl in the grass plays with fingerpaints in front of white canvas with rainbow on it endseq
startseq there is girl with pigtails sitting in front of rainbow painting endseq
startseq young girl with pigtails painting outside in the grass endseq
--------------------Predicted--------------------
startseq little girl in red dress pulls fingerpaints endseq
```

In [38]:

```
generate_caption("101669240_b2d3e7f17b.jpg")
```

```
---------------------Actual---------------------
startseq man in hat is displaying pictures next to skier in blue hat endseq
startseq man skis past another man displaying paintings in the snow endseq
startseq person wearing skis looking at framed pictures set up in the snow endseq
startseq skier looks at framed pictures in the snow next to trees endseq
startseq man on skis looking at artwork for sale in the snow endseq
--------------------Predicted--------------------
startseq man displaying paintings in the snow endseq
```

Project Demo Link