Project Report Format TABLE OF CONTENTS:

1. INTRODUCTION

1.1 Project Overview

The creation of a system that can automatically provide insightful descriptions for photos is the primary objective of this research. In order to comprehend an image's content and provide logical and contextually appropriate captions, this requires utilizing machine learning and computer vision algorithms.

This is why caption generation has always been considered a challenging subject. Machine learning algorithms have a significant problem in that it essentially involves imitating the extraordinary human capacity to condense vast quantities of visually significant information into evocative language.

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

It might be difficult to come up with captions that effectively capture the relationships between things and the larger context of an image.

Captions for images with complicated or ambiguous content may be erroneous or imprecise.

It is possible that certain picture caption generators are not designed with real-time processing in mind, particularly when working with huge datasets or high-resolution photos.

2.2 References

Hairan Wang, A Synopsis of Image Captioning Techniques, (CIN2020)

"IMAGE CAPTION GENERATOR USING DEEP LEARNING," B. Krishnakumar, K. Kochalya, S. Gokul, R. Karthikeyan, and D. Kavithayarasu, International Journal of Advanced Science and Technology, 2020

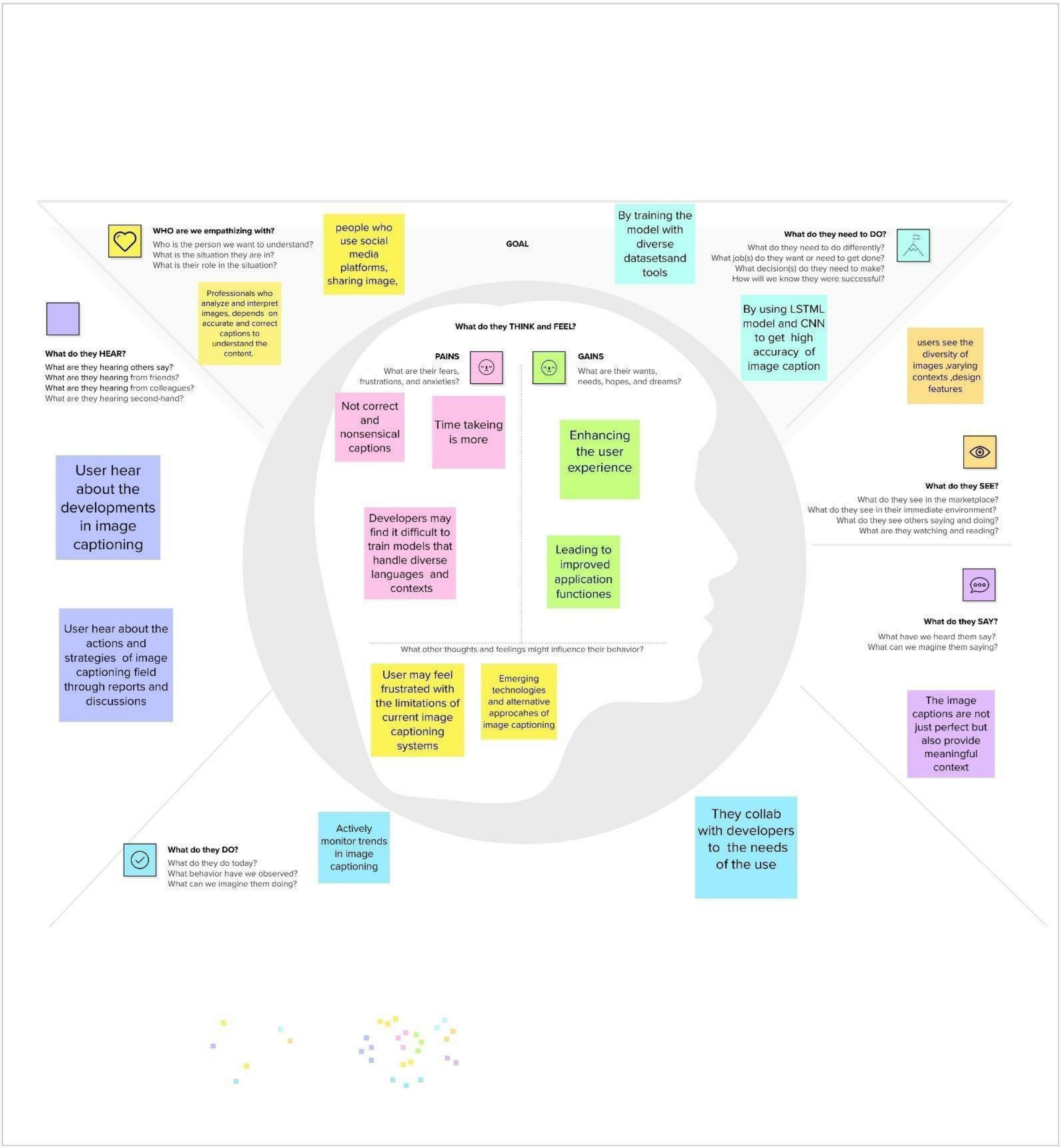
A thorough analysis of deep learning for image captioning by MD. Zakir Hossain and Hamid Laga (ACM-2019)

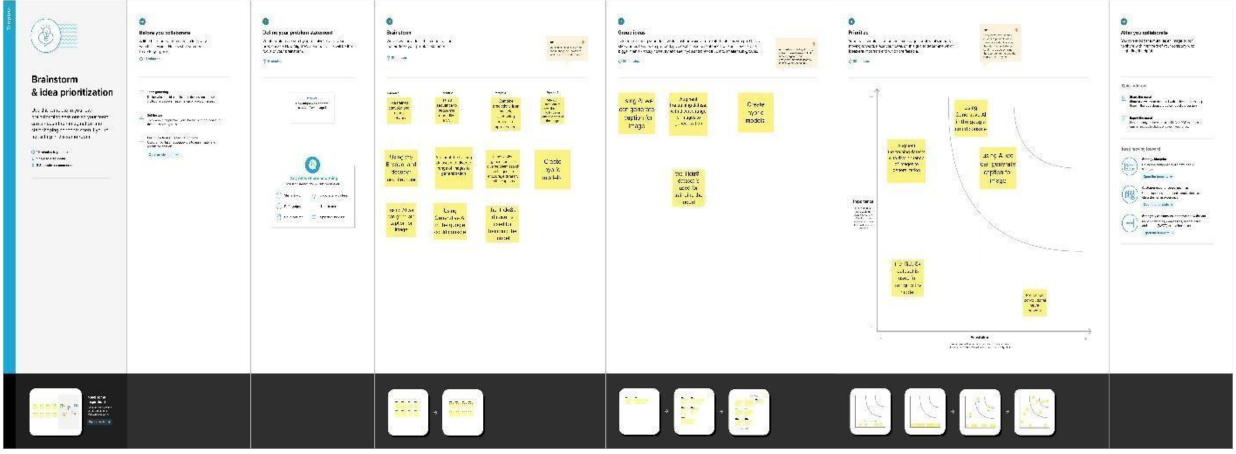
2.3 Problem Statement Definition

Generating correct and contextually relevant image captions for a wide range of visual content is a challenging endeavour when done automatically. Present-day picture caption generators frequently struggle to deal with ambiguity, comprehend intricate relationships among images, and correct biases in training data. The inability of current models to adapt to various domains, creative constraints, and difficulties with real-time processing further reduce their efficacy.

3. IDEATION & PROPOSED SOLUTION 3.1

Empathy Map Canvas





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

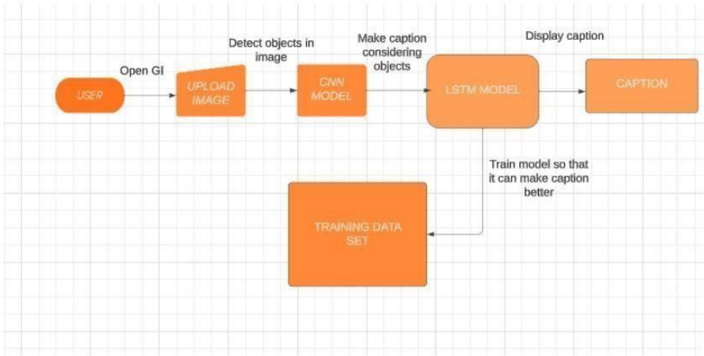
4. Usability

5. Security

5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

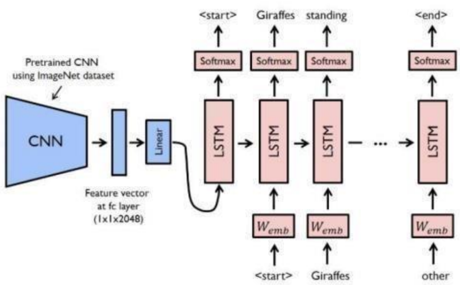
Data Flow Diagrams:



User Type	Functional Requirement (Epic)	User Story Number	User Story / Task	Acceptance criteria	Priority	Release
Security and surveillance	Analysing and understanding	USN-1	Security systems and surveillance applications can use image caption generation to automatically describe scenes captured by cameras.	Trained Dataset	High	Sprint-1
HEALTH CARE	Well-designed automotive AI	USN-2	Healthcare applications may use image caption generation to assist in automatically generating descriptions for medical images, aiding healthcare professionals in documentation and analysis.	We Could prepare these models CNN and LSTM	High	Sprint-1
Social Media Posts	Matter Recognition	USN-3	Social media platforms like Facebook, Instagram, and Twitter use image caption generation to automatically generate descriptive captions for photos uploaded by users.	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

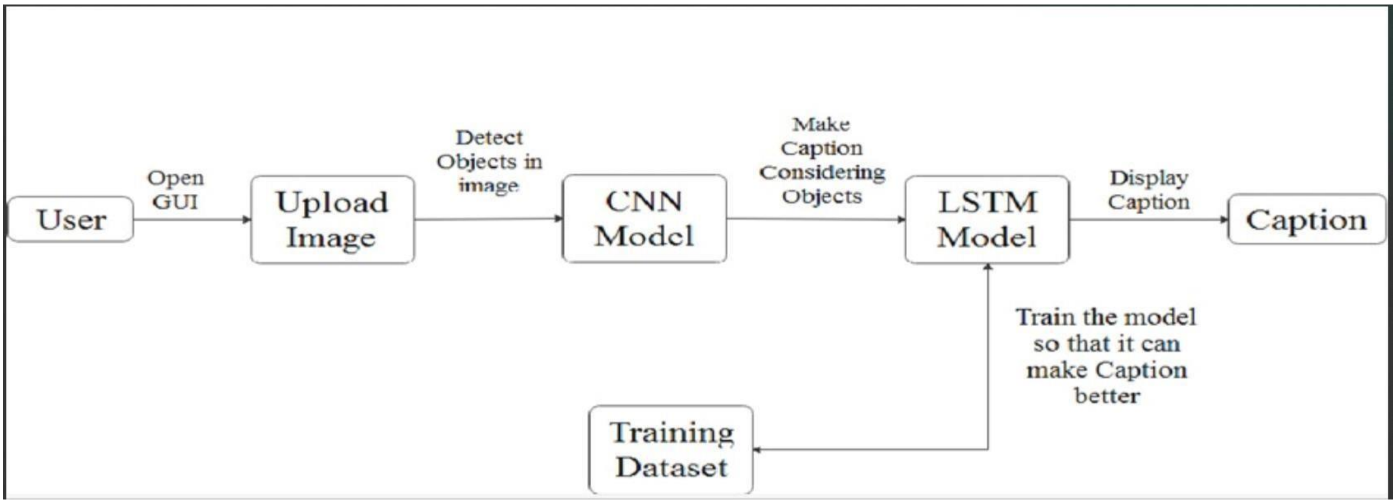
5.2 Solution Architecture

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	Project setup &infrastructure	USN-1	Set up the development environment with the required tools and frameworks	1	High	ROSHAAN
Sprint-1	Development environment	USN-2	Gather a diverse dataset		High	SRIHAS
Sprint-2	Data Collection	USN-3	Preprocess the collected dataset	2	High	VARSHITH
Sprint-2	Data Preprocessing	USN-4	Investigate and assess different machine learning techniques.	3	High	SHANMUKH

6.3 Sprint Delivery Schedule

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

Project Structure:
Project Structure:

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

Sprint-3	Model Deployment	USN-5	Train the machine learning model on the pre-processed data	5	Medium	ROSHAAN
Sprint-3	Training	USN-6	Incorporate data augmentation techniques	5	Medium	SRIHAS
Sprint-4	Model Deployment & Integration	USN-7	Deploy the trained machine learning model ad an API or web. Service to enable detect	1	Medium	VARSHITH
Sprint-5	Testing &quality assurance	USN-8	Test the model and web interface to uncover and repot any problems	1	Medium	SHANMUKH

Sprint - 1	3	2 days	28 Oct 2023	30 Oct 2023	3	30 Oct 2023
Sprint – 2	5	2 days	31 Oct 2023	2 Nov 2023	8	2 Nov 2023
Sprint – 3	10	5 days	3 Nov 2023	8 Nov 2023	18	8 Nov 2023
Sprint – 4	1	4 days	9 Nov 2023	13 Nov 2023	19	13 Nov 2023
Sprint - 5	1	2 days	14 Nov 2023	16 Nov 2023	20	16 Nov 2023

.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:

The data is available for download on the Kaggle website; it contains almost 8000 photos with five captions each. There are 40,000 excellent, readable text captions in the provided dataset. Once the datasets have been downloaded, you should make a folder and add the datasets as Flickr_8k_text and Flickr_8k_dataset.

Step 2:

Data pre-processing:

Organize the text captions and group them collectively. Next, we will construct our Vgg16 model, which consists of an LSTM model and an input layer CNN model.

Task-1

First we have to import all the necessary packages

```python
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

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 cleans the data after receiving all descriptions. This is a crucial stage in the process of working with textual data because it allows us to choose the kind of text cleaning that will best serve our objectives. In this instance, all text will be converted to lowercase, punctuation will be eliminated, and words containing numbers will be eliminated. With this in mind, a caption such as "A man riding on a three-wheeled wheelchair" will become "man riding on three-wheeled wheelchair."

This

Task 3

Task-3

Extracting the features

```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
```

```python
##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
```

technique is also known as transfer learning because we don't have to do everything ourselves; instead, we use pre-trained models that have already been trained on large datasets and extract the features from these models to use for our tasks. We're employing the Exception model, which was trained on an ImageNet dataset with 1000 different classes to classify. We can import this model directly from keras.applications. Make sure you're connected to the internet because the weights will be downloaded automatically. We will make few changes to the Xception model because it was originally designed for imagenet. One thing to keep in mind is that the Xception model requires an image size of 299*299*3. 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

Task-4

Loading dataset for training the model

Loading dataset for Train ing 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.

```
#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
```

```
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)
```

```
#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

Task-5

Tokinizing the vocabulary

Tokenizing the vocabulary

Since computers cannot understand English words, we must represent them for them using numbers. As a result, we will assign a distinct index value to every word in the vocabulary. The tokenizer function in the Keras library is what we'll use to generate tokens from our vocabulary and store them in a pickle file called "tokenizer.p."

```
# 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
```

```
#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

Task-6
Task-6
Data Generation
Data Generation

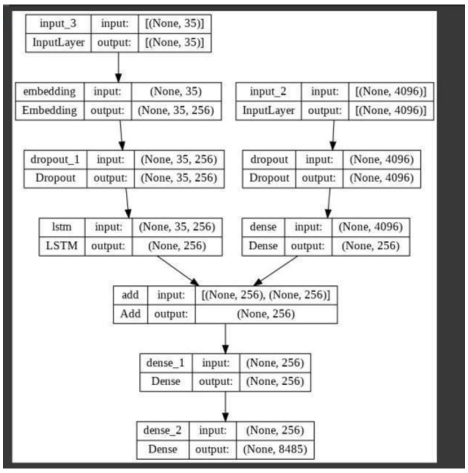
Data generation

We are developing a data generation where the images can be stored into memory in order to run our model using the images and description captions in the Flicker8k-datset, which contains roughly 8000 images.

```python
#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)
```

To arrive at the final prediction, we will process by the dense layer by combining the output from the two layers above. The number of nodes in the final layer will match the size of our vocabulary.



```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
Step-3:

Model Building
Model Building

Now we can train our image data set

Train the model
Train the model

By creating the input and output sequences in batches and fitting them to the model using the model.fit_generator() function, we will be able to train the model with the 6000 training images. The model is additionally saved to our models folder. Depending on the capabilities of your system, this may take some time.

```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
```

tep 4 S

Step 4:

Step 4:

Testing the model

TTesting the modelesting the model

It is important to determine whether the data accurately fits the model as we test it. Now that the model has been trained, testing_caption_generator.py, a separate file, will be created and loaded to enable the model to produce 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']

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

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


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)
```

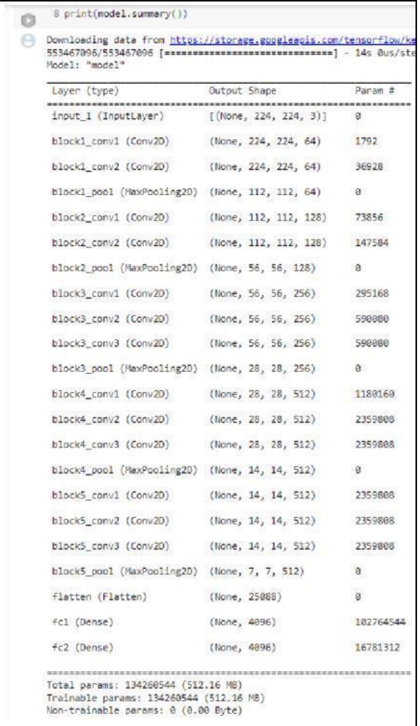
8. PERFORMANCE TESTING

8.1 Performance Metrics

S.No.	Parameter	Values	Screenshot

<table>
<tr><td>1.</td><td>Model Summary</td><td>Total params: 134260544

Trainable params: 134260544

Non-trainable params: 0</td><td>
</td></tr>
<tr><td>2.</td><td>Accuracy</td><td>Training Accuracy - 97.81%

Validation Accuracy – 98.17%</td><td></td></tr>
<tr><td>3.</td><td>Confidence Score (Only Yolo Projects)</td><td>Class Detected - NA

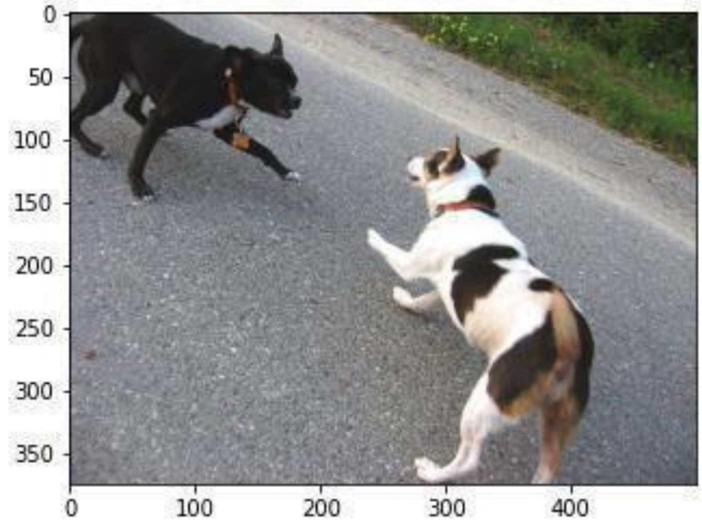
Confidence Score - NA</td><td>NOT APPLICABLE</td></tr>
</table>

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

- By offering textual information related to images, captions support search engine optimization efforts and may enhance the discoverability and placement of visual content in search results.

- By offering more details and context, image captions are an invaluable educational resource that facilitate the comprehension and learning of visual content.

- You can use image caption generators for a variety of creative projects, like making interactive experiences, image-based narratives, or captions for artwork.

- By providing text descriptions for images, image captions enable people with visual impairments to interact and understand visual content.

DISADVANTAGES

- Complex or nuanced images may not always be accurately described by caption generators, resulting in inaccurate or misleading captions.

- Complex image caption generators can require a lot of processing and storage power to train and implement, which limits their use for smaller-scale applications.

- Image caption generators may find it difficult to deal with ambiguity in visual scenes, which can result in generic or ambiguous captions—particularly for images that can be interpreted in different ways.

- Models may exhibit poor generalization to novel or diverse images due to overfitting to particular patterns found in the training data.

- The caliber and variety of the training data have a significant impact on caption generator performance. Insufficient or skewed datasets may result in subpar generalization.

11. CONCLUSION

In summary, image caption generators offer a significant combination of advantages and challenges at the cutting edge of computer vision and natural language processing. There are many uses for the capability of automatically creating insightful captions for photos, from improving user experience and accessibility to supporting search engine optimization and creating instructional materials.

Image caption generators could be instrumental in influencing how we engage with visual content online, promoting inclusivity, and advancing AI-driven narrative generation as technology develops. The quest continues for more precise, flexible, and imaginative image captioning systems that are genuinely capable of comprehending and describing the rich and varied realm of visual data.

12. FUTURE SCOPE

The potential for image caption generators is bright, as continued research and development should solve present issues and create new opportunities. Subsequent studies could investigate multimodal strategies that integrate data from both text and images to enhance caption creation. For outputs that are more logical and sensitive to context, this might entail adding textual context to the image captioning procedure. It's likely that efforts will continue to optimize image caption generators for real-time processing. This is especially crucial for apps like augmented reality and live streaming that call for instant communication

To consistently evaluate image caption generator performance, it will probably be necessary to develop standardized benchmarks and evaluation metrics. This will make it easier for researchers to compare models and monitor field advancements. Image caption generators have a bright future ahead of them, full of opportunities for innovations that could greatly expand their capabilities and make them more useful in a variety of contexts. Collaboration and ongoing interdisciplinary research will be essential in forming this exciting future.

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]:

```
# 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]:

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

In [34]:

```
# 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 e ndseq 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
```

<u>Project Demo</u>

<u>Link</u>

https://vitapacin-my.sharepoint.com/:v:/g/personal/srihas_21bce7362_vitapstudent_ac_in/Ee5uuETfY2lDhKQPoe31uZEBtxBa-7SyqEqJ-W61Su_A0Q