EXPERIMENT 3

IMAGE CAPTIONING USING FLICKR8K DATASET

AIM:

To develop and implement a deep learning based image captioning model using the Flickr8k dataset, where a Convolutional Neural Network (CNN) extracts image features and a Recurrent Neural Network (RNN) with LSTM units generates natural language descriptions for the given images.

PRE-REQUISITES:

1. Basics of Machine Learning
2. Python Programming
3. Knowledge on Numpy, Pandas, Matplotlib, TensorFlow/ Keras
4. Jupyter Notebook
5. Data Pre-Processing Techniques
6. Knowledge on Neural Networks

FLICKR8K Dataset

- Flikr8k Dataset contains 8,000 images, each showing people or animals doing various activities.
- Each image has five different captions written in plain English to describe it.
- The dataset is mainly used for training and testing image captioning models.
- Download the Dataset - https://www.kaggle.com/datasets/adityajn105/flickr8k/data

1. Importing the Basic Libraries

In [53]:
```
# Basic data handling and analysis
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os
import string
import re
from PIL import Image
```

2. Load the Dataset

In [54]:
```
images_path = "/Users/srinutupakula/Desktop/Deep Learning Lab/Experiment 3/archive/Images"
captions_path = '/Users/srinutupakula/Desktop/Deep Learning Lab/Experiment 3/archive/captions.txt'
```

In [55]:
```
# Create a Funtion to load the Captions File (File Handling)

def load_doc(filename):
    with open(filename, 'r') as file:
        text = file.read()
    return text

# Load captions text
captions_text = load_doc(captions_path)
```

In [56]:
```
# Reading the captions text
captions_text[0:500]
```

Out[56]:
```
'1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .\n1000
268201_693b08cb0e.jpg,A girl going into a wooden building .\n1000268201_693b08cb0e.jpg,A little girl clim
bing into a wooden playhouse .\n1000268201_693b08cb0e.jpg,A little girl climbing the stairs to her playho
use .\n1000268201_693b08cb0e.jpg,A little girl in a pink dress going into a wooden cabin .\n1001773457_57
7c3a7d70.jpg,A black dog and a spotted dog are fighting\n1001773457_577c3a7d70.jpg,A bl'
```

3. Pre-Process the Captions Data

a) Load captions file and organize in a dictionary

- It reads the captions file, then groups captions by their image name in a dictionary. Each image name becomes a key, and its value is a list of all captions for that image. It splits each line at the first comma, trims spaces, and stores the captions clearly for quick access later.

```
In [57]: def load_captions(captions_path):
             captions_dict = {}
             with open(captions_path, 'r') as f:
                 for line in f:
                     line = line.strip()
                     if not line:
                         continue

                     # Split only on the first comma --> filename, caption
                     img_id, caption = line.split(',', 1)

                     img_id = img_id.strip()
                     caption = caption.strip()

                     captions_dict.setdefault(img_id, []).append(caption)
             return captions_dict

         captions = load_captions(captions_path)
         len(captions)
```

```
Out[57]: 8091
```

```
In [58]: captions['1000268201_693b08cb0e.jpg']
```

```
Out[58]: ['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 .']
```

b) Clean captions (lowercase, remove punctuation/numbers)

- Cleans the captions for each image by making all letters lowercase, removing punctuation and numbers, and taking care of extra spaces. It processes every caption in the dictionary and returns a new dictionary with the cleaned captions, making the text neat and consistent for further use.

```
In [59]: def clean_captions(captions_dict):
             cleaned_dict = {}
             for img_id, captions in captions_dict.items():
                 cleaned_captions = []
                 for cap in captions:
                     # Lowercase
                     cap = cap.lower()
                     # Remove punctuation and numbers
                     cap = re.sub(r'[^a-z\s]', '', cap)
                     # Remove extra spaces
                     cap = re.sub(r'\s+', ' ', cap).strip()
                     cleaned_captions.append(cap)
                 cleaned_dict[img_id] = cleaned_captions
             return cleaned_dict

         captions = clean_captions(captions)
         len(captions)
```

```
Out[59]: 8091
```

```
In [60]: captions['1000268201_693b08cb0e.jpg']
```

```
Out[60]: ['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']
```

c) Add <start> and <end> tokens

- Adds special tokens *start* and *end* to every caption in the dictionary. For each image, it takes its captions and wraps them with these tokens, then returns a new dictionary. These tokens help mark the beginning and end of a caption, which is useful when training models for image captioning.

```
In [61]: def add_tokens(captions_dict):
             tokenized_dict = {}
             for img_id, captions in captions_dict.items():
                 tokenized_dict[img_id] = [f"<start> {cap} <end>" for cap in captions]
             return tokenized_dict

         captions = add_tokens(captions)
         len(captions)
```

```
Out[61]: 8091
```

```
In [62]: captions['1000268201_693b08cb0e.jpg']
```

```
Out[62]: ['<start> a child in a pink dress is climbing up a set of stairs in an entry way <end>',
          '<start> a girl going into a wooden building <end>',
          '<start> a little girl climbing into a wooden playhouse <end>',
          '<start> a little girl climbing the stairs to her playhouse <end>',
          '<start> a little girl in a pink dress going into a wooden cabin <end>']
```

d) Build vocabulary

- It reates a vocabulary set from all the captions in the dictionary. It goes through each caption, splits it into individual words, and adds them to a set (which automatically removes duplicates). The final result is a collection of all unique words used across the captions.

```
In [63]: def build_vocabulary(captions_dict):
             vocab = set()
             for captions in captions_dict.values():
                 for cap in captions:
                     vocab.update(cap.split())
             return vocab

         vocab = build_vocabulary(captions)
         len(vocab)
```

```
Out[63]: 8780
```

```
In [64]: # vocab
```

e) Create (image_id, caption) pairs for training

- It takes the captions dictionary and creates a list of (image_id, caption) pairs. It loops through each image ID and its associated captions, then for every caption, it makes a tuple with the image ID and that caption. The result is a flat list where each entry links a single image to one of its captions.

```
In [65]: def create_image_caption_pairs(captions_dict):
             pairs = []
             for img_id, captions in captions_dict.items():
                 for cap in captions:
                     pairs.append((img_id, cap))
             return pairs

         pairs = create_image_caption_pairs(captions)
         len(pairs)
```

```
Out[65]: 40455
```

```
In [66]: pairs[0:5]
```

```
Out[66]: [('1000268201_693b08cb0e.jpg',
           '<start> a child in a pink dress is climbing up a set of stairs in an entry way <end>'),
          ('1000268201_693b08cb0e.jpg',
           '<start> a girl going into a wooden building <end>'),
          ('1000268201_693b08cb0e.jpg',
           '<start> a little girl climbing into a wooden playhouse <end>'),
          ('1000268201_693b08cb0e.jpg',
           '<start> a little girl climbing the stairs to her playhouse <end>'),
          ('1000268201_693b08cb0e.jpg',
           '<start> a little girl in a pink dress going into a wooden cabin <end>')]
```

4. Extract Image Features

a) Choose a pre-trained CNN and create an encoder model

- It Imports InceptionV3 model pre-trained on ImageNet and builds an image encoder by removing the top classification layers (include_top=False). It uses global average pooling (pooling='avg') to produce a fixed-size feature vector for each input image. This encoder will be used to extract meaningful features from images for the captioning model

In [67]:
```
from tensorflow.keras.applications.inception_v3 import InceptionV3, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.models import Model

def build_image_encoder():
    base = InceptionV3(weights='imagenet', include_top=False, pooling='avg')
    return base

encoder = build_image_encoder()
```

In [68]:
```
# encoder.summary()
```

b) Image Pre-Processing

- It loads an image from the given images_path and resizes it to the target size (default 299×299 pixels, matching InceptionV3's input size). It converts the image to a NumPy array and removes the alpha channel if present (some PNGs have 4 channels). Then it adds a batch dimension (required for model input) and applies model-specific preprocessing (scaling and normalization) before returning the processed image array ready for feature extraction.

In [69]:
```
def load_and_preprocess_image(images_path, target_size=(299, 299)):
    img = load_img(images_path, target_size=target_size)
    arr = img_to_array(img)
    if arr.shape[-1] == 4:
        arr = arr[..., :3]
    arr = np.expand_dims(arr, axis=0)
    arr = preprocess_input(arr)
    return arr
```

c) Extract features for all images and save them

- It extracts feature vectors from all images in a given folder using a pre-trained CNN encoder (like InceptionV3). It processes only common image formats (jpg, jpeg, png), loads and preprocesses each image, obtains its feature vector by running it through the encoder, and stores these features in a dictionary keyed by filename. Optionally, it saves the extracted features as a pickle file for later use. Finally, it runs the function on your images folder and saves the features to "features.pkl

In [70]:
```
import pickle
from tqdm import tqdm

def extract_features_from_folder(images_path, encoder, save_path=None):
    features = {}

    # Only process common image types
    img_files = [f for f in os.listdir(images_path) if f.lower().endswith(('.jpg', '.jpeg', '.png'))]

    for fname in tqdm(img_files, desc='Extracting features'):
        full = os.path.join(images_path, fname)
        try:
            # Load and preprocess
            img_arr = load_and_preprocess_image(full)
            # Extract features
            feat = encoder.predict(img_arr, verbose=0)
            features[fname] = feat.flatten()
        except Exception as e:
            print(f"Error processing {fname}: {e}")

    # Save features if path provided
    if save_path:
        with open(save_path, 'wb') as f:
            pickle.dump(features, f)
    return features
```

In [71]:
```
#features = extract_features_from_folder(images_path, encoder, "features.pkl")
```

```
Extracting features: 100%|██████████████████| 8091/8091 [10:59<00:00, 12.27it/s]
```

In [72]:
```
print(f"Total images processed: {len(features)}")
first_key = list(features.keys())[0]
print(f"Sample image: {first_key}")
print(f"Feature vector shape: {features[first_key].shape}")
```

```
Total images processed: 8091
Sample image: 2387197355_237f6f41ee.jpg
Feature vector shape: (2048,)
```

d) Create Image Feature and Caption Pairs

- This code creates a list of training pairs, where each pair consists of an image feature vector and one of its corresponding captions. It iterates over each image ID in the captions dictionary, checks if the image's features exist, and if so, pairs each caption with the image features. If features for an image are missing, it prints a warning. Finally, it returns the list of all such (feature, caption) pairs and prints the total count of training samples.

In [73]:
```
def create_training_pairs(features, captions_dict):
    pairs = []
    for img_id, caps in captions_dict.items():
        if img_id in features:
            for cap in caps:
                pairs.append((features[img_id], cap))
        else:
            print(f"Warning: No features found for image {img_id}")
    return pairs

training_pairs = create_training_pairs(features, captions)
len(training_pairs)
```

Out[73]:
```
40455
```

e) Tokenize Captions and Prepare Sequences

- It processes all the captions by first gathering them into one list and then using a tokenizer to convert each word into a unique integer. The tokenizer is fitted on all captions to build a vocabulary, including a special token for unknown words. It calculates the total vocabulary size and converts every caption into a sequence of integer tokens. To ensure uniform input size for the model, it finds the longest caption length and pads all shorter sequences with zeros at the end, so every caption sequence has the same length. This prepares the text data in a numerical form suitable for training the deep learning model.

In [74]:
```
# Convert captions (strings) into sequences of integers that the model can process

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np

# Gather all captions in a list
all_captions = []
for caps in captions.values():
    all_captions.extend(caps)

# Initialize and fit tokenizer on all captions
tokenizer = Tokenizer(oov_token="<unk>")
tokenizer.fit_on_texts(all_captions)

# Vocabulary size (add 1 for padding token)
vocab_size = len(tokenizer.word_index) + 1
print(f"Vocabulary Size: {vocab_size}")

# Convert captions to sequences
sequences = tokenizer.texts_to_sequences(all_captions)

# Find max caption length for padding
max_length = max(len(seq) for seq in sequences)
print(f"Maximum caption length: {max_length}")
```

```
# Pad sequences to max_length
padded_sequences = pad_sequences(sequences, maxlen=max_length, padding='post')
```

```
Vocabulary Size: 8780
Maximum caption length: 37
```

f) Prepare Final Dataset for Training

- It prepares the training data by turning each caption into sequences of words where the model learns to predict the next word. For every caption, it creates multiple input-output pairs: the input is the image features plus a partial caption, and the output is the next word in the caption. The input sequences are padded to the same length, and the output words are one-hot encoded to indicate the correct prediction. All these inputs and outputs are collected into arrays that can be used to train the model.

In [75]:

```
# Separate image features and captions into arrays

# image features
X1 = []
# input sequences (captions shifted)
X2 = []
# output word (next word to predict)
y = []

for img_feat, cap in training_pairs:
    seq = tokenizer.texts_to_sequences([cap])[0]
    for i in range(1, len(seq)):
        in_seq, out_seq = seq[:i], seq[i]
        in_seq = pad_sequences([in_seq], maxlen=max_length, padding='post')[0]

        out_seq_onehot = np.zeros(vocab_size)
        out_seq_onehot[out_seq] = 1

        X1.append(img_feat)
        X2.append(in_seq)
        y.append(out_seq_onehot)


X1 = np.array(X1)
X2 = np.array(X2)
y = np.array(y)
```

In [76]:

```
X1.shape, X2.shape, y.shape
```

Out[76]:

```
((476793, 2048), (476793, 37), (476793, 8780))
```

5. Build the Decoder Model (Caption Generator)

- It builds an image captioning model that takes both image features and caption sequences as input. The image features are processed through dropout and a dense layer, while the captions pass through an embedding layer, dropout, and an LSTM to understand the sequence. These two outputs are combined and fed through additional dense layers to predict the next word in the caption. The model uses categorical cross-entropy loss and the Adam optimizer, and includes an early stopping mechanism to stop training if the loss stops improving.

In [77]:

```
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
from tensorflow.keras.models import Model
from tensorflow.keras.callbacks import EarlyStopping

# Image feature input
inputs1 = Input(shape=(2048,))
fe1 = Dropout(0.5)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)

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

# Decoder (combine)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
```

```python
outputs = Dense(vocab_size, activation='softmax')(decoder2)

# Define the Call back
early_stopping = EarlyStopping(monitor='val_loss', patience=3, restore_best_weights=True)


# Define the model
model = Model(inputs=[inputs1, inputs2], outputs=outputs)

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

In [78]:
```python
model.summary()
```

```
Model: "functional_1"

┏━━━━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┳━━━━━━━━━━━━┳━━━━━━━━━━━━━━━━━━━┓
┃ Layer (type)         ┃ Output Shape      ┃    Param # ┃ Connected to      ┃
┡━━━━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━╇━━━━━━━━━━━━╇━━━━━━━━━━━━━━━━━━━┩
│ input_layer_6        │ (None, 37)        │          0 │ -                 │
│ (InputLayer)         │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ input_layer_5        │ (None, 2048)      │          0 │ -                 │
│ (InputLayer)         │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ embedding_1          │ (None, 37, 256)   │  2,247,680 │ input_layer_6[0]… │
│ (Embedding)          │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_3 (Dropout)  │ (None, 2048)      │          0 │ input_layer_5[0]… │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dropout_4 (Dropout)  │ (None, 37, 256)   │          0 │ embedding_1[0][0] │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ not_equal_1          │ (None, 37)        │          0 │ input_layer_6[0]… │
│ (NotEqual)           │                   │            │                   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_4 (Dense)      │ (None, 256)       │    524,544 │ dropout_3[0][0]   │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ lstm_1 (LSTM)        │ (None, 256)       │    525,312 │ dropout_4[0][0],  │
│                      │                   │            │ not_equal_1[0][0] │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ add_1 (Add)          │ (None, 256)       │          0 │ dense_4[0][0],    │
│                      │                   │            │ lstm_1[0][0]      │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_5 (Dense)      │ (None, 256)       │     65,792 │ add_1[0][0]       │
├──────────────────────┼───────────────────┼────────────┼───────────────────┤
│ dense_6 (Dense)      │ (None, 8780)      │  2,256,460 │ dense_5[0][0]     │
└──────────────────────┴───────────────────┴────────────┴───────────────────┘

 Total params: 5,619,788 (21.44 MB)
 Trainable params: 5,619,788 (21.44 MB)
 Non-trainable params: 0 (0.00 B)
```

6. Train the Model

In [79]:
```python
# 10% of the training data will be set aside for validation
# The model doesn’t train on this data but uses it to check how well it’s generalizing after each epoch

model.fit([X1, X2], y, epochs=15, batch_size=64, verbose=1, validation_split=0.1, callbacks=[early_stoppin
```

```
Epoch 1/15
6705/6705 ━━━━━━━━━━━━━━━━━━━━ 605s 90ms/step - loss: 4.2663 - val_loss: 3.3908
Epoch 2/15
6705/6705 ━━━━━━━━━━━━━━━━━━━━ 497s 74ms/step - loss: 3.1816 - val_loss: 3.2879
Epoch 3/15
6705/6705 ━━━━━━━━━━━━━━━━━━━━ 1660s 248ms/step - loss: 2.9390 - val_loss: 3.2546
Epoch 4/15
6705/6705 ━━━━━━━━━━━━━━━━━━━━ 731s 109ms/step - loss: 2.8113 - val_loss: 3.2650
Epoch 5/15
6705/6705 ━━━━━━━━━━━━━━━━━━━━ 576s 86ms/step - loss: 2.7291 - val_loss: 3.3035
Epoch 6/15
6705/6705 ━━━━━━━━━━━━━━━━━━━━ 493s 73ms/step - loss: 2.6744 - val_loss: 3.3484
```

Out[79]:
```
<keras.src.callbacks.history.History at 0x1697588e0>
```

7. Save the Trained Model

In [84]:
```python
# Save the trained model to reuse it later without retraining
model.save('image_captioning_model.keras')
```

8. Define a function to generate captions for new Images

- This function generates a caption for a given image feature vector using the trained model and tokenizer. It starts with the special token and iteratively predicts the next word by converting the current text sequence into integers, padding it, and feeding it along with the image feature to the model. At each step, it selects the word with the highest predicted probability and adds it to the caption. The process continues until the model predicts the token or reaches the maximum caption length. Finally, it returns the complete generated caption as a string.

In [85]:
```
def generate_caption(model, tokenizer, photo_feature, max_length):
    in_text = '<start>'
    for _ in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length, padding='post')
        yhat = model.predict([photo_feature.reshape(1, -1), sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = tokenizer.index_word.get(yhat, None)
        if word is None or word == 'end':
            break
        in_text += ' ' + word
    return in_text.replace('<start> ', '')
```

9. Generate Captions for New Images

In [93]:
```
# Extract the Features for new images
features = extract_features_from_folder(images_path, encoder, "features.pkl")
```

```
Extracting features: 100%|██████████████████████████| 2/2 [00:00<00:00, 11.69it/s]
```

In [94]:
```
# Load image feature for a new image (extract features as before)
new_image_feature = features['kids.jpg']
```

In [95]:
```
caption = generate_caption(model, tokenizer, new_image_feature, max_length)
caption
```

Out[95]:
```
'a man in a red shirt is running on a field'
```

In [96]:
```
# Load image feature for a new image (extract features as before)
new_image_feature = features['dog.jpg']
```

In [97]:
```
caption = generate_caption(model, tokenizer, new_image_feature, max_length)
caption
```

Out[97]:
```
'a dog is running through a field'
```

RESULT:

The image captioning model was developed to generate captions for input images by combining visual features extracted from a pre-trained CNN with sequence modeling of captions using an LSTM network. Using the Flickr8k dataset, the model was able to produce descriptive sentences that capture the main elements and actions in images, demonstrating the effectiveness of deep learning and transfer learning techniques for this task.