VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

IMAGE DESCRIPTION GENERATOR

A report submitted for the Final Review of

SOCIAL AND INFORMATION NETWORKS
(CSE3021)

By
Abhiram Sreekantham - 19BCE2294

Submitted to
Prof. Govinda K

DECLARATION

I hereby declare that the report entitled "Image Description Generator" submitted by me, for the CSE3021 Social and Information Networks (EPJ) to VIT is a record of bonafide work carried out by me under the supervision of Prof. Govinda K.

I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for any other courses in this institute or any other institute or university.

Date : 26th August 2022

Candidate Signature

Abhiram Sreekantham

CONTENT

AIM

A daunting challenge in the field of deep learning is creating a description for a picture. In this project, I will discuss several Natural Language Processing and Computer Vision strategies for identifying an image's context and describing it in a language like English.
Using different Convolutional Neural Network architectures like the VGG16, ResNet, MobileNet, Xception and Long Short-Term Memory (LSTM) components, I will be creating a functioning model of the picture description generator. I'm leveraging The Flickr8K dataset to train the model. It has 8000 distinct photos, each of which will be linked to five alternative descriptions of the image.

ABSTRACT

It was formerly thought inconceivable for a computer to describe an image. We can now create models that can provide descriptions summarizing a picture because of advancements in deep learning techniques and the availability of enormous amounts of data.
The most actively explored and developing topics of research that is permeating every aspect of our everyday life is deep learning. It is a branch of machine learning that focuses on algorithms and is motivated by the design and operation of the brain.
With the rapid development of Artificial Intelligence in recent years, Image description has steadily drawn the attention of the scientific community in the field of Deep learning and has grown to be both an exciting and challenging issue. Image descriptions, which automatically generate natural language descriptions based on the material shown in an image, are a crucial component of scene comprehension, which integrates the expertise of computer vision with natural language processing. It is widely used and important to realise human-computer interaction, for instance, through the use of picture descriptions.
In this paper, I will be creating a working prototype of the image description generator. The AI-powered image description generating model is an automated method that effectively produces succinct and insightful descriptions for enormous quantities of photographs. The model uses computer vision and Natural Language Processing (NLP) techniques to extract thorough textual data about the provided images. The benefits and drawbacks of these methodologies are also examined, along with the most popular assessment standards in this domain. This research concludes by highlighting several unresolved issues with the picture description task.

INTRODCUTION

When you see an image, your brain immediately understands what it is about, however, will a computer be able to do the same? Computer vision researchers spent a lot of time working on this and thought it was unattainable until today. We can create models that can generate descriptions for a picture thanks to advancements in deep learning techniques, the availability of massive datasets, and computing power.
In this project, I will employ a combination of deep learning methods using convolutional neural networks and a particular kind of recurrent neural network (LSTM). One to several RNNs are used in the process of describing images. Using the vocabulary of training data, the model suggests the description for a particular input image.

The reason I chose the Flickr8K dataset is:

- It is not too big in size. Thus, the model may be trained with ease on domestic computers and laptops.
- Data has been labelled correctly. There are 5 captions available for each image.
- The dataset is freely accessible.

The dataset has 2 folders:

- 'Flickr8k_Dataset': There are 8092 JPEG photos altogether, all of different sizes and forms. 1000 are used for testing, 1000 are used for development, and 6000 are used for training.
- 'Flickr8k_text': Includes text files explaining the train set and test set. Each image in the Flickr8k.token.txt file has five captions, for a total of 40460 captions.

Convolutional Neural Network:

Convolutional Neural Network, abbreviated as CNN is a Deep Learning method that takes in an input image and gives priority for components such as learnable weights and biases to different characteristics and objects in the image to help it distinguish between distinct images.
Image categorization is one of this architecture's most well-liked uses. The neural network combines nonlinear, pooling, and multiple convolutional layers. The output of the first convolution layer has now become the input for the second layer after the picture has gone through one convolution layer. For each layer after that, this process is repeated.
It is important to attach a completely connected layer following a sequence of convolutional, nonlinear, and pooling layers. The output data from convolutional networks is used in this layer. An N-dimensional vector, where N is the number of classes wherein the model chooses the desired class, is produced by attaching a completely linked layer to the end of the network.

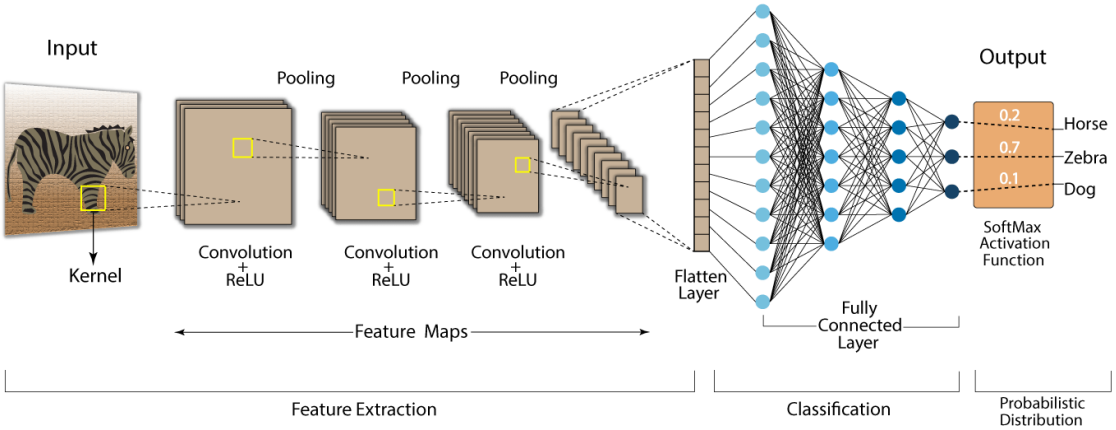


Fig 1. Working of CNN model

Long Short-Term Memory (LSTM):

A type of recurrent neural network (RNN) models known as Long Short-Term Memory (LSTM) networks have the capacity to learn order dependency in sequence prediction issues. The majority of the time, this is used to difficult issues like speech recognition, machine translation, and many others.
The development of LSTM was motivated by the discovery that while training conventional RNNs, poor predicting performance resulted from gradients that were very tiny or zero as we descended farther into a neural network. Since there may be delays of uncertain length between significant occurrences in a time series, LSTM networks are well-suited for categorizing, processing, and generating predictions based on time series data.
A memory cell known as a "cell state" that preserves its state over time plays a key function in an LSTM model. The horizontal line that passes across the top of the figure below shows cell state. It may be seen as a conveyor belt across which data simply and unaltered passes. Gates in LSTMs control the addition and removal of information from the cell state. These gates may allow information to enter and exit the cell. It has a sigmoid neural network layer and a pointwise multiplication operation that help the mechanism. Between zero and one, the sigmoid layer outputs a number, with zero denoting "nothing should be let through" and one denoting "everything should be let through."

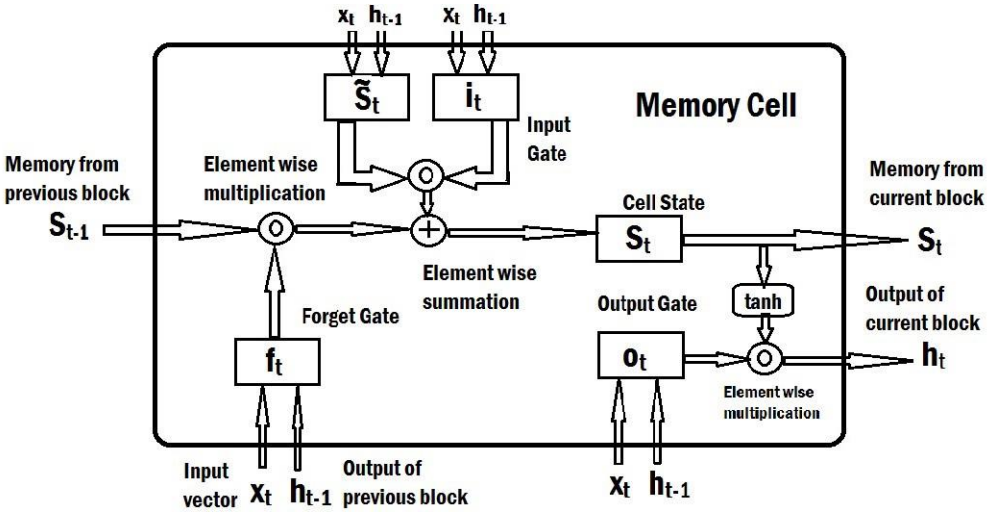


Fig 2. Working of LSTM model

CNN-LSTM Architecture:

The CNN-LSTM architecture combines LSTMs to facilitate sequence prediction with CNN layers for feature extraction on input data. This approach is especially made for issues involving the sequence prediction of spatial inputs, such as photos or videos. They are rather frequently utilized in a lot of activities which includes activity recognition, image-video description, and many others.
CNN-LSTMs are typically used when the inputs have both spatial and temporal structure, including the sequence of pictures in a video or the words in text, or when the output must have temporal structure, such as phrases in a text content. Examples of inputs with spatial and temporal structure include the 2D structure of pixels in the region or the 1-dimensional structure of words and sentences, paragraph, or document.
The process of captioning images may be logically separated into two modules:

- The characteristics of our image are extracted using an image-based model.
- a language-based model that converts the traits and objects our image-based model has retrieved into a natural phrase.

We employ CNN for our image-based model and LSTM for our language-based model.

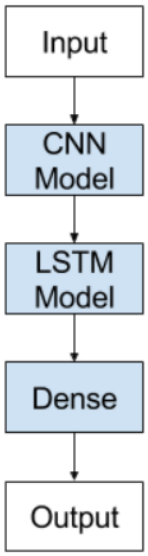


Fig 3. CNN-LSTM Architecture

LITERATURE REVIEW

[1] Image Caption Generator Using CNN and LSTM

For this work, CNN and LSTM are used to get aware of the image caption. A technique called image caption generation uses computer vision and natural language processing to understand the relationship between the image and the English caption. In this study work, we cautiously explore some key ideas about image captioning and its well-known procedures. In order to create this article, we discuss the Keras library, numpy, and Jupyter notebooks. Additionally, we discuss how CNN and the flickr dataset are utilised to categorise photos.
Every picture on the internet can have captions, which makes browsing and indexing more quickly and thoroughly. There are several applications for image captioning, including those in biology, business, the military, and online search. Social media platforms like Instagram and Facebook may automatically create captions from photographs. This study paper's main objective is to get some understanding of deep learning techniques. For image classification, we specifically employ two strategies: CNN and LSTM.
The concept of the CNN-LSTM model was to provide captions for the input images. There are several uses for this approach. We researched the CNN model, RNN model, and LSTM model in this case, and we ultimately verified that the model is producing captions for the input photographs.

[2] Show and Tell: A Neural Image Caption Generator

We have introduced NIC, a complete neural network system that can automatically view a picture and produce a logical plain English description. NIC relies on a convolution neural network to compress an image's representation into a little amount of data, then a recurrent neural network to produce the text that goes with it. The model is trained to increase the sentence's probability given the picture. Using ranking metrics or BLEU, a measure used in machine translation to assess the quality of generated phrases, experiments on various datasets demonstrate the resilience of NIC in terms of qualitative outcomes (the generated sentences are extremely logical), as well as quantitative assessments. These tests demonstrate that the performance of methods like NIC will improve as the size of the datasets available for picture description grows. It will also be intriguing to observe how picture description

methods may be improved with unsupervised data, both from photos alone and text alone. We show that our model is frequently fairly accurate and provide qualitative and quantitative evidence. For example, our method produces a score of 59, which may be compared to human performance of roughly 69, whereas the current state-of-the-art BLEU-1 score (the higher the better) on the Pascal dataset is 25, according to the Pascal dataset. Additionally, we demonstrate BLEU-1 score increases from 56 to 66 on Flickr30k and from 19 to 28 on SBU. Last but not least, we reach the state-of-the-art BLEU-4 on the recently published COCO dataset, which is 27.7.

[3] Learning CNN-LSTM Architectures for Image Caption Generation

Recent developments in computer vision and natural language processing are combined in automatic picture caption production. This study creates a generative CNN-LSTM model that outperforms human baselines by 2.7 BLEU-4 points and is almost on par with the state of the art (3.8 CIDEr points lower). Research on the MSCOCO dataset set demonstrates that it typically produces correct and comprehensible captions, and hyperparameter optimization with dropout and the number of LSTM layers enables us to reduce the consequences of overfitting. Using a maintain probability of 75% for dropout and two layers for our decoder LSTM network, we ran a thorough hyperparameter search across the CNN-LSTM model architecture, yielding a best model that yields results that are 3.3 BLEU-4 points and 3.8 CIDEr points below the state-of-the-art. The model appears to be capable of intelligently captioning a wide range of photos from the MSCOCO dataset, according to a detailed quantitative and qualitative review of the output metrics. Partial mistakes sometimes happen as a result of failing to pay attention to certain aspects in photos, such as mislabeling an image of elephants wandering in a pen as "elephants in a field" because of backdrop trees.
We also show that, despite having different prior contexts, words that are semantically close to one another when they are emitted—like "plate" and "bowl"—move the LSTM hidden state similarly, and that divergences in hidden state only occur when semantically distant words—like "vase" and "food"—are emitted. This gives the interplay between learnt word embeddings and LSTM hidden states a semantic significance. To our knowledge, this is a fresh literary contribution.

[4] What is the Role of Recurrent Neural Networks (RNNs) in an Image Caption Generator?

Recurrent neural networks (RNNs) are frequently seen as the main "generation" component in neural image captioning systems. According to this perspective, the RNN should be "injected" with the image characteristics. In fact, this is the most prevalent viewpoint in the literature. The RNN may also be thought of as just encoding the words that have already been created. According to this perspective, the RNN should only be used to encode linguistic features, and the picture features should only be "merged" into the final representation at a later time. These two architectures are contrasted in this essay. We discover that late merging performs better than injection generally, indicating that RNNs are more appropriately thought of as encoders than generators.
The studies detailed in this article used data from two different datasets. One issue is that, despite the evident superiority of merge over inject in both, results on Flickr8k and Flickr30k are not totally consistent. Our studies are now being expanded to include the bigger MSCOCO dataset. Merging designs has further practical benefits, such as transfer learning. The RNN used for captioning might theoretically be moved from a neural language model that has been trained on general text since merging maintains the picture distinct from the RNN. Since the RNN would need to be trained to merge picture and text in the input, this cannot be done with an inject architecture. Future research will examine how the initialization of the RNN's weights from those of a broad neural language model affects the performance of a caption generator in a manner similar to that of neural machine translation.

[5] Review Networks for Caption Generation

The review network is a brand-new addition to the encoder-decoder system that we suggest. In this study, we take into consideration RNN decoders using both CNN and RNN encoders. The review network is flexible and may improve any current encoder decoder model. After each review step, the review network outputs a thought vector, which is utilised as the input of the attention mechanism in the decoder. The review network executes a series of review steps with an attention mechanism on the encoder hidden states. We demonstrate that traditional encoder-decoders are an exception to our approach. Empirically, we demonstrate that our approach outperforms cutting-edge encoder-decoder systems in source code and picture captioning.
In Table 3, we present the log-likelihood and top-k character savings of several model versions. The LSTM decoder that makes up the baseline model's "Language Model" produces output that is insensitive to the input code sequence. We chose the LSTM decoder as a benchmark since early testing revealed that it performs much better than the Ngram models used in [11] (+3% in CS-2). We also contrast several encoder-decoder variations, such as those that include attention mechanisms and bidirectional RNN encoders. Table 3 shows that bidirectional encoders and attention mechanisms can both outperform standard encoder-decoders. The review network regularly beats attentive encoder-decoders in all parameters, demonstrating its effectiveness in learning valuable representation.
[6] An Overview of Image Caption Generation Methods
The entire image caption generation task has been covered in this overview, along with the model framework that has been recently suggested to address the description task. We have also concentrated on the algorithmic core of various attention mechanisms and provided a brief overview of how the attention mechanism is used. We provide an overview of the sizable datasets and standard practise rating standards. Despite the fact that image caption can be used for image retrieval, video caption, and video movement, and despite the wide range of image caption systems that are currently available, experimental results demonstrate that this task still has room for improvement and better performance systems. It primarily struggles with the following three issues: first, how to produce complete sentences in natural language that are grammatically correct; second, how to ensure that the generated sentences are correct; and third, how to make the caption semantics as transparent and consistent with the provided image content as possible. We suggest the following four potential advancements for further work:

(1) An image frequently has a lot of substance. Instead of merely describing one target item, the 0e model should be able to provide description sentences that correspond to many primary objects in pictures with multiple target objects.
(2) A universal image description system that can handle numerous languages should be designed for corpus description languages of various languages.
(3) It can be challenging to assess the output of natural language generating systems. The greatest technique to measure the calibre of texts produced automatically is through subjective linguistic analysis, which is challenging to accomplish. The evaluation indicators should be improved to bring them closer to assessments made by human specialists in order to increase system performance.
(4) A very serious issue is that the model's performance can be improved by speeding up the training, testing, and sentence generation processes.

[7] Image Caption Generator

It is a very difficult challenge to automatically generate the description or caption of an image using any natural language words. It takes both computer vision techniques to comprehend the image's content and a language model from the discipline of natural language processing to translate that knowledge into the appropriate sequence of words. Additionally, we have spoken about how to apply this model online so that it is also user-accessible. The goal of our project is to put into practise an

image caption generator that responds to user input to produce captions for a picture. By creating automatic descriptions, the image caption generator ultimately aims to improve user experience. This has applications in social networking, picture indexing, assistive technology for the blind, and various other areas of natural language processing. Deep learning techniques have produced cutting-edge outcomes for caption generating issues. The most amazing aspect of these techniques is that, rather than requiring complex data preparation or a pipeline of specially created models, a single end-to-end model can be developed to predict a caption given a photo. Based on the image we supply; this tool will automatically produce a caption using a trained model. When we utilize or apply it on social media or on other applications, the main premise is that consumers will receive automatic captions.

[8] Visual Image Caption Generator Using Deep Learning

With the intention of not only explaining the surrounding world but also assisting those who are visually impaired in understanding their settings, we have introduced a deep learning model that tends to automatically produce image captions. The foundation of the model we've discussed is a CNN feature extraction model that converts an image into a vector representation, followed by an RNN decoder model that creates related sentences using the image characteristics that have been trained. In order to evaluate different encoder decoder models and show various use cases on our system, we looked at how each component affects the production of captions. The findings indicate that, while needing more time for training and sentence creation due to its complexity, the LSTM model performs somewhat better than GRU overall. By training on more photos from a larger dataset, performance is also anticipated to improve. The text-to-speech technology that we have also implemented can be quite helpful to visually impaired persons and help them obtain a better feel of their surroundings because of the significant accuracy of the generated image captions. Our model is not flawless and occasionally produces inaccurate captions. In the following step, we will create models that employ Inceptionv3 as the feature extractor rather than VGG. The 4 models we have thus far obtained—VGG+GRU, VGG+LSTM, Inceptionv3+GRU, and Inceptionv3+LSTM—will next be compared. This will aid in our analysis of the CNN component's impact on the overall network. Currently, we are choosing the word with the highest probability as the next word in the sequence using a greedy method. Instead, beam search chooses a set of terms with the greatest likelihood and conducts simultaneous searches over every sequence. We could improve our forecast accuracy by using this strategy. Our model was developed using the relatively modest and homogeneous Flickr 8K dataset. The Flickr30K and MSCOCO datasets will be used to train our algorithm, which will enable us to make more accurate predictions. Other improvements involve adjusting the hyperparameters and learning how each one affects our model, such as batch size, number of epochs, learning rate, etc.

[9] Image Caption Generator

We can see from the data that the deep learning technology employed here produced fruitful outcomes. Together, the CNN and LSTM were able to determine the relationship between objects in pictures by synchronizing their operations. The goal is to create a system that can accept an image input in the form of a dimensional array and produce an output that is a syntactically and grammatically accurate sentence that describes the picture. The corpus that we utilized was the Flickr 8K dataset. There are 5 captions for each of the 8000 photos in the collection. Understanding all the different circumstances is made easier by the one image's 5 captions. The dataset includes three prepared datasets: Flickr 8k.trainImages.txt (6,000 photos), Flickr 8k.devImages.txt (1,000 images), and Flickr 8k.testImages.txt (1,000 images) (1,000 images). The chosen photos are from six different Flickr groups and don't feature any famous people or locations. But they are chosen by hand to display a range of scenes. Using the BLEU (Bilingual Evaluation Understudy) score, we can compare the projected captions to the target captions in our Flickr8k test dataset to assess how accurate they are [5, 8]. Text translation uses BLEU ratings to compare translated text to one or more reference translations. Similar to the one described

here; hybrid picture caption generators have been made throughout the years using a variety of other neural network approaches. such as using the GRU model instead of the STM model or the VGG16 model in place of the Xception model. Additionally, BLEU Score may be used to compare different models and determine which one has the highest level of accuracy. The scope of the subject of machine learning and artificial intelligence was revealed in this article, along with a number of recent significant breakthroughs. While this work attempts to address the fundamental requirements needed to produce an image caption generator, some subjects within it are open to additional research and improvement.

[10] Image Caption Generator Based On Deep Neural Networks

In this study, we thoroughly examine an image caption creation technique based on deep neural networks. The approach may produce an English sentence that describes the content of a picture given as input. Convolutional neural networks (CNN), recurrent neural networks (RNN), and sentence generation are the three parts of the approach that we examine. We discover that the VGGNet outperforms the CNN component when it is substituted with three cutting-edge architectures, as measured by the BLEU score. We also suggest the Gated Recurrent Units (GRU) as a new recurrent layer, with MATLAB and C++ implementation in Caffe. When the simplified GRU is put up against the long short-term memory (LSTM) approach, similar results are obtained. However, it just has a few parameters, which speeds up training and conserves memory. Finally, we use Beam Search to produce several phrases. The results demonstrate that, with less training memory, the improved approach may provide captions that are equivalent to those produced by state-of-the-art systems. We examined and improved the LRCN approach for captioning images. We broke down the process into CNN, RNN, and sentence creation in order to fully grasp it. We changed or replaced each component for each portion to observe how it affected the end outcome. The COCO caption corpus is used to assess the improved approach. The results of the experiment demonstrate that: first, the VGGNet performs better in measuring BLEU scores than AlexNet and GoogleLeNet; second, the simplified GRU model achieves results comparable to those of the more complex LSTM model; and third, increasing the beam size generally increases the BLEU score but does not always improve the quality of the description as determined by humans. We want to investigate ways to produce many sentences with distinct content in the future. Combining picture captioning with interesting region recognition is one approach that may be used. The VSA approach, developed by Karpathy and Fei-Fei, provides guidance for our future work. We anticipate the result will be a succinct paragraph using Fig. 2 as an example: Jack has a delicious brunch on a Sunday. He is holding a bunch of red flowers while seated at a table. He has a cup of coffee and a platter of fruits while using his new iPad Air on the left. The brief paragraph organically and in a way that tells a tale makes the visual material more appealing to readers.

PROPOSED METHOD

Algorithm description:

- A difficult task in the field of deep learning is creating a caption for a picture. Here, we'll employ a variety of computer vision and natural language processing (NLP) approaches to identify the context of a picture and explain it in an English-like natural language. Using CNN (Convolutional Neural Networks) and LSTM (Long Short-Term Memory) components, we will create a functioning model of the picture caption generator.
- First Import the required libraries
- We extract the image feature by using the VGG16 model.
- Load the descriptions in step two. Our file is formatted with a picture followed by a space, a caption, and the image description in CSV format. The image and caption are separated by a

newline ("n"). Here, we must save the picture's descriptors in a dictionary and map them to the image.

- Then comes Text cleaning, to enable the machine to quickly identify patterns in the text, one of the major NLP procedures is to eliminate noise. Special characters like hashtags, punctuation, and numerals will be used as noise. If any of these are present in the text, computers will have trouble understanding it. For improved outcomes, we must thus get rid of them. Using the NLTK library, you can also eliminate stop words, carry out stemming, and carry out lemmatization.
- Create the vocabulary in next step. A set of distinct terms that are found in our text corpus make up our vocabulary. Everything that is done while processing raw text for NLP revolves around the vocabulary.
- Add the pictures. Here, we must connect the training set's photos to the descriptions that go with them, which are stored in our descriptions variable. Make a list of all the names of the training pictures, then make an empty dictionary and use the names of the images as the keys and a list of descriptions as the values to map the images to their descriptions. When mapping the descriptions, add distinguishing terms at the beginning and end of each phrase to indicate where the sentence begins and ends.
- We will now provide an image as input to our model, however robots, unlike humans, cannot comprehend the image just by looking at it. So that the machine can recognize the patterns in the image, we must encode the original image. I'm utilizing transfer learning for this assignment, which means that we take the characteristics from a pre-trained model that has previously been trained on huge datasets and apply them in our work. I'm using the InceptionV3 model in this instance, which was trained using the Imagenet dataset with its 1000 distinct classifications. This model may be easily imported from the Keras.applications module. To obtain the (2048,) dimensional feature vector from the InceptionV3 model, we must first remove the last classification layer.
- Tokenizing the vocabulary is step seven. We need to tokenize each term in our vocabulary in this stage. As an alternative, we may complete this work using Keras' tokenizer.
- Global vectors for word representation is referred to as GloVe. It is a Stanford-developed unsupervised learning technique that creates word embeddings by combining the global word-word co-occurrence matrix from a corpus. Additionally, there are 8000 photographs on our site, and each one includes five captions. This indicates that we have 30.000 samples to train our model on. As there are more instances, you can also utilise a data generator to feed our model input in batches as opposed to all at once. Additionally, in order to store the relationships between the terms in our lexicon, we will employ an embedding matrix. A linear mapping of the original space to a real-valued space, where things would have significant connections, is known as an embedding matrix.
- We will use the Keras Model from Functional API to specify the structure of our model. There are three main steps: Processing the text's sequence and removing the image's feature vector Concatenating the top two layers will allow you to decode the output.
- Teaching the model, I'm using categorical cross-entropy as the loss function and Adam's optimizer to train our model. I'm giving the model 50 training epochs, which should be sufficient to predict the results. You may train it by reducing batch size and increasing number of epochs if you have greater computing capacity (number of GPUs).
- Estimating the result

Procedure

- We must import all of the fundamental modules we will use for this project first.
  - os - utilized for system command-based file management.
  - Pickle is used to store the extracted numpy features.
  - arrays may be processed using a wide range of mathematical operations using numpy.
  - Progress bar decoration for iterators using tqdm. comes with a preset range iterator that prints to stderr.
  - Imported modules for extracting the features from the picture data are found in VGG16, preprocess input.
  - Use the functions load img and img to array to load the picture and make it into a numpy array.
  - Tokenizers are used to load text and turn it into tokens.
  - Pad sequences is a tool for distributing words evenly throughout sentences by replacing any empty spaces with zeros.
  - plot model is a tool for displaying the model's architecture as a series of pictures.

Import Modules

In [1]:
```
import numpy as np
from tqdm.notebook import tqdm
import os
import pickle

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
from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

- The directories must now be configured to use the data.

In [2]:
```
BASE_DIR = '/kaggle/input/flickr8k'
WORKING_DIR = '/kaggle/working'
```

- The model has to be loaded and reorganized. Here, just the earlier layers of the VGG16 model are required to extract feature findings; the fully linked layer is not required. If you'd like, you may add more layers, but for speedier results, stay away from doing so.

Extract Image Features

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 3s 0us/step
553476096/553467096 [==============================] - 3s 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
```

- We now import the data for preprocessing and extract the picture characteristics.
  - The dictionary "features" is built and filled with the features of the picture data.
  - Img path, target size=(224, 224), load img - a custom dimension that will resize the picture when it is loaded into the array.
  - Reshaping the image data to preprocess in an RGB type picture using image.reshape((1, image.shape[0], image.shape[1], image.shape[2])).
  - Extrapolation of features from the picture using model.predict(image, verbose=0)
  - img name.split('.')[0] - separating the picture name from the extension so that it loads alone.
  - Using pickle:
    Re-extraction of characteristics may increase running time because they are not kept on the disc. To save time, empty and store your lexicon in a pickle before reloading it. For a faster runtime, load all of your feature data that is currently stored.

In [4]:
```
# 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 [5]:
```
# store features in pickle
pickle.dump(features, open(os.path.join(WORKING_DIR, 'features.pkl'), 'wb'))
```

In [6]:
```
# load features from pickle
with open(os.path.join(WORKING_DIR, 'features.pkl'), 'rb') as f:
    features = pickle.load(f)
```

- Let's save the text file's captions data.

Load the Captions Data

In [7]:
```
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
```

- We now separate the captions data and append it to the image. With the key being the image id and the value being the associated caption text, a dictionary "mapping" is established. If image id is not in the mapping, the same image may have different captions: image id mapping = [] produces a list so that captions may be added to the associated image.

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

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

- Let's check the number of loaded photos now.

In [9]:
```
len(mapping)
```

Out[9]: 8091

- To clean and transform the text for a speedier procedure and better outcomes, preprocess text data.

Preprocess Text Data

In [10]:
```
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
```

- Observe the text both before and after cleaning.

```
In [11]: # before preprocess of text
         mapping['1000268201_693b08cb0e']
```

```
Out[11]: ['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 [12]: # preprocess the text
         clean(mapping)
```

```
In [13]: # after preprocess of text
         mapping['1000268201_693b08cb0e']
```

```
Out[13]: ['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']
```

- The preprocessed captions will then be kept in a list. The first ten captions will be displayed.

```
In [14]: all_captions = []
         for key in mapping:
             for caption in mapping[key]:
                 all_captions.append(caption)
```

```
In [15]: len(all_captions)
```

```
Out[15]: 40455
```

```
In [16]: all_captions[:10]
```

```
Out[16]: ['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
          'startseq girl going into wooden building endseq',
          'startseq little girl climbing into wooden playhouse endseq',
          'startseq little girl climbing the stairs to her playhouse endseq',
          'startseq little girl in pink dress going into wooden cabin endseq',
          'startseq black dog and spotted dog are fighting endseq',
          'startseq black dog and tri-colored dog playing with each other on the road endseq',
          'startseq black dog and white dog with brown spots are staring at each other in the street endseq',
          'startseq two dogs of different breeds looking at each other on the road endseq',
          'startseq two dogs on pavement moving toward each other endseq']
```

- We now begin analysing the text input, determining the maximum caption length to be utilised as a guide for the padding sequence.

```
In [17]: # tokenize the text
         tokenizer = Tokenizer()
         tokenizer.fit_on_texts(all_captions)
         vocab_size = len(tokenizer.word_index) + 1

In [18]: vocab_size

Out[18]: 8485

In [19]: # get maximum length of the caption available
         max_length = max(len(caption.split()) for caption in all_captions)
         max_length

Out[19]: 35
```

- After preparing the data, we will now train, test, and break the sequence into pairs, as explained by an example. The padding sequence will then be included in a batch that we designate later. For improved outcomes, padding sequence normalizes all caption sizes to the maximum size and fills them with zeros.

Train Test Split

In [20]:
```
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.90)
train = image_ids[:split]
test = image_ids[split:]
```

In [21]:
```
# startseq girl going into wooden building endseq
#        X                  y
# startseq                  girl
# startseq girl             going
# startseq girl going       into
# ..........
# startseq girl going into wooden building      endseq
```

In [22]:
```
# 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
```

- Creating the model
  - The output length of the features from the VGG model shape=(4096,)
  - Single-dimensional linear layer array that is dense
  - Dropout() is a function that regularises the data, prevents overfitting, and removes a portion of the data from the layers.
  - Model compilation is performed through model.compile().
  - loss=’sparse categorical crossentropy’ - loss function for outputs by categories
  - optimizer='adam' - automatically modify the model's learning rate throughout the number of epochs
  - The inputs and outputs are concatenated into a single layer on the model visualization.
  - No CNN model was required in this stage because feature extraction from the picture was previously done using VGG.

Model Creation

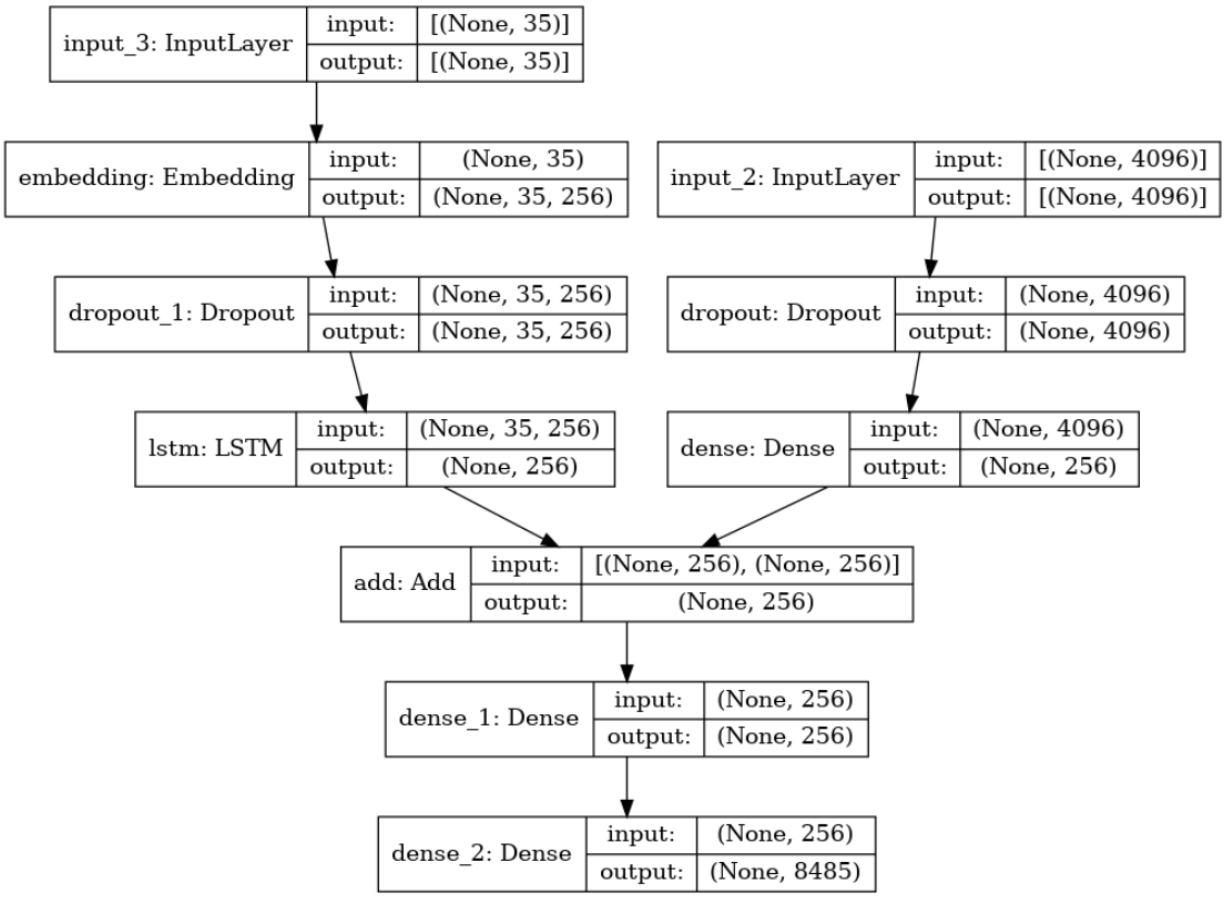
In [23]:
```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.4)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.4)(se1)
se3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

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

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

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

Out[23]:



- Let's train the model now, and you may reuse it by saving it in the working directory.

In [24]:
```
# 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 [==============================] - 68s 285ms/step - loss: 5.2210
227/227 [==============================] - 66s 291ms/step - loss: 4.0199
227/227 [==============================] - 66s 292ms/step - loss: 3.5781
227/227 [==============================] - 65s 287ms/step - loss: 3.3090
227/227 [==============================] - 66s 292ms/step - loss: 3.1080
227/227 [==============================] - 65s 286ms/step - loss: 2.9619
227/227 [==============================] - 63s 276ms/step - loss: 2.8491
227/227 [==============================] - 64s 282ms/step - loss: 2.7516
227/227 [==============================] - 64s 282ms/step - loss: 2.6670
227/227 [==============================] - 65s 286ms/step - loss: 2.5966
227/227 [==============================] - 66s 290ms/step - loss: 2.5327
227/227 [==============================] - 61s 270ms/step - loss: 2.4774
227/227 [==============================] - 65s 288ms/step - loss: 2.4307
227/227 [==============================] - 66s 289ms/step - loss: 2.3873
227/227 [==============================] - 62s 274ms/step - loss: 2.3451
227/227 [==============================] - 65s 285ms/step - loss: 2.3081
227/227 [==============================] - 65s 288ms/step - loss: 2.2678
227/227 [==============================] - 66s 292ms/step - loss: 2.2323
227/227 [==============================] - 65s 285ms/step - loss: 2.1992
227/227 [==============================] - 66s 291ms/step - loss: 2.1702
```

In [25]:
```
# save the model
model.save(WORKING_DIR+'/best_model.h5')
```

- Make captions for the picture. Transform the model's predicted index into a word. Appending all the words to a caption generator for an image The caption begins with "startseq," and the model keeps predicting it until "endseq" shows up.

Generate Descriptions for the Image

In [26]:
```
def idx_to_word(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
```

In [27]:
```
# 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
```

- The data is now verified using BLEU Score, which compares the projected text to a reference text in a list of tokens. Every word added to the reference text from the captions data (actual captions) is included. A BLEU Score of greater than 0.4 is regarded as a successful outcome; to get a higher score, increase the number of epochs correspondingly.

In [28]:
```
from nltk.translate.bleu_score import corpus_bleu
# validate with test data
actual, predicted = list(), list()

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

# calcuate BLEU score
print("BLEU-1: %f" % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
print("BLEU-2: %f" % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
```

```
  0%|          | 0/810 [00:00<?, ?it/s]

BLEU-1: 0.516880
BLEU-2: 0.293009
```

- Visualize the Results Using the Detailed Image Caption Generator. Printing the image's expected caption follows the printing of the image's actual caption.

Visualize the Results

In [29]:
```
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)
```

RESULTS

A few chosen images that are forming clusters; These are just used to display the dataset's associated photographs.

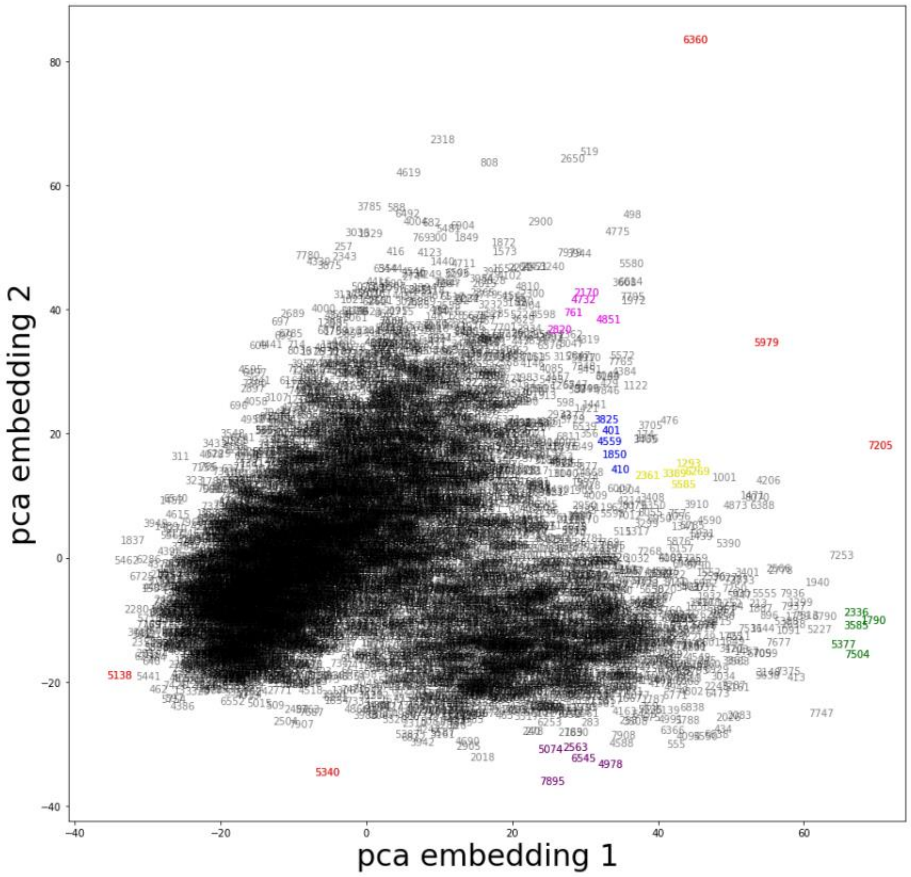


Fig 4. Dataset’s associated photographs

Fig 5. Plotting sample images

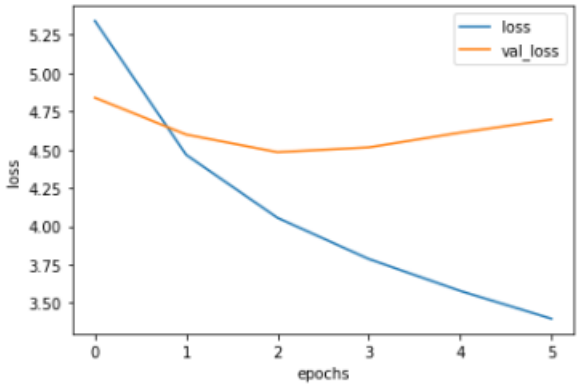


Fig 6. LSTM model – Loss vs Epochs

```
In [30]: 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 endseq
startseq two dogs on pavement moving toward each other endseq
--------------------Predicted--------------------
startseq two dogs play with each other in the grass endseq
```

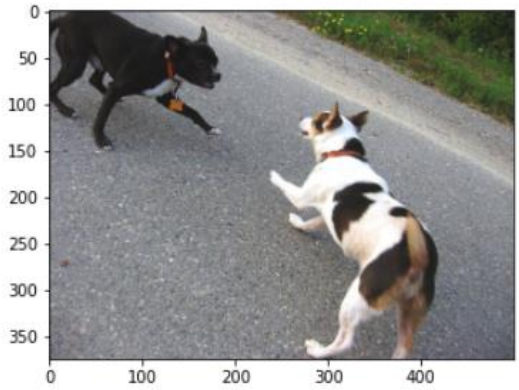


Fig 7. Example 1 of Image Description Generator model

```
In [31]: 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 pink dress is lying on the side of the grass endseq
```

Fig 8. Example 2 of Image Description Generator model

```
In [32]: 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 two people are hiking up snowy mountain endseq
```

Fig 9. Example 3 of Image Description Generator model

CONCLUSION

Deep learning and computer vision are used in the process of creating image description generators, which identify the context of a picture and annotate it with pertinent captions. In order to get the results for this complex deep learning project, more than one model must be employed for data analysis and preprocessing. In this paper, I go over how to use the flickr dataset to create an image caption generator in Python. The implementation of the project makes use of the Keras & Tensorflow technology. In the project, the model is built using both text and picture characteristics. This will help us learn how to use the model architecture of several domains for a particular application.

Predicting the captions for the supplied image is the project's goal. The collection includes 5 captions for each of the 8 000 photos. Both the image and the text captions are used to extract the characteristics for input. The attributes will be combined to forecast the caption's subsequent word. For images, CNN is utilised, and for text, LSTM. The performance of the trained model is measured using the BLEU Score.

Final BLUE Score achieved:

BLEU 1 is 0.544

BLEU 2 is 0.319

Increasing the number of epochs used while training the model can produce better, more accurate results. Processing a lot of data may be time- and resource-intensive. You may increase the model's layer count if you wish to handle a big dataset like flickr32k. We may also use CNN model to extract the picture characteristics rather than VGG16.

Applications of Image Description generator in Social and Information Networks:

There are several uses for image description, including suggestions in editing software, use in virtual assistants, image indexing, for people with visual impairments, on social media, and in a number of other natural language processing implementations.
In Social and Information Networks, this model may be used to estimate network centrality metrics such as degree, betweenness, proximity, etc.
This can also be used to detect communities in social networks depending on the images and captions used to train the model. With the right data, we can categorize certain network nodes into various communities and even determine which nodes are involved in overlapping the neighboring communities.
Most importantly we can use this model to represent the visualizations of a social network. It can extract all the necessary information for a given image and provide use with the Social and Information Network within seconds.

REFERENCES

Citations

[1] Swarnim Tripathi, Ravi Sharma. Image Caption Generator Using CNN and LSTM. IJCRT_196552.

[2] Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3156-3164).

[3] Soh, M. (2016). Learning CNN-LSTM architectures for image caption generation. *Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, Tech. Rep, 1*.

[4] Tanti, M., Gatt, A., & Camilleri, K. P. (2017). What is the role of recurrent neural networks (rnns) in an image caption generator?. *arXiv preprint arXiv:1708.02043*.

[5] Yang, Z., Yuan, Y., Wu, Y., Cohen, W. W., & Salakhutdinov, R. R. (2016). Review networks for caption generation. *Advances in neural information processing systems*, *29*.

[6] Wang, H., Zhang, Y., & Yu, X. (2020). An overview of image caption generation methods. *Computational intelligence and neuroscience*, *2020*.

[7] Parth Kotak , Prem Kotak, 2021, Image Caption Generator, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 10, Issue 11 (November 2021),

[8] Sharma, Grishma & Kalena, Priyanka & Malde, Nishi & Nair, Aromal & Parkar, Saurabh. (2019). Visual Image Caption Generator Using Deep Learning. SSRN Electronic Journal. 10.2139/ssrn.3368837.

[9] Panicker, Megha & Upadhayay, Vikas & Sethi, Gunjan & Mathur, Vrinda. (2021). Image Caption Generator. International Journal of Innovative Technology and Exploring Engineering. 10. 87-92. 10.35940/ijitee.C8383.0110321.

[10] Chen, J., Dong, W., & Li, M. (2014). Image caption generator based on deep neural networks.

Links

https://data-flair.training/blogs/python-based-project-image-caption-generator-cnn/

https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/

https://artificialintelligence.oodles.io/blogs/ai-powered-image-caption-generator/

https://www.clairvoyant.ai/blog/image-caption-generator

https://www.hindawi.com/journals/cin/2020/3062706/#dataset-and-evaluation

https://www.geeksforgeeks.org/image-caption-generator-using-deep-learning-on-flickr8k-dataset/

https://www.kaggle.com/code/kevinwaghela/image-caption-generator-deep-learning