IMAGE CAPTION GENERATION USING DEEP LEARNING

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

G. Dhakshayani - AP2110010930

K. Venkatesh - AP21110010933
Shubham Pandey - AP21110010940
Ch. Sai Kumar - AP21110010943
I. Anu Likitha - AP21110010963

Under the Guidance of

(Radha Guha)

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

Dec, 2023

Certificate

Date: 2-Dec-23

I hereby affirm that the project titled "Image Caption Generation" has been conducted by the following individuals under my guidance:

Venkata Naga Saikumar Chunduru - AP21110010943

I certify that the work submitted is legitimate, unique, and suitable for submission to SRM University – AP in order to complete the requirements for the School of Engineering and Sciences' Bachelor of Technology and Master of Technology degrees.

Supervisor (Signature)
Dr. Radha Guha

Designation, Affiliation.

I. Acknowledgments

We sincerely thank Dr. Radha Guha, our project guide, for her leadership, mentoring, and steadfast support during the project. Her knowledge and perceptions have been invaluable in determining the course and results of our effort.

Additionally, we would like to thank the entire project team for their cooperation, hard work, and dedication. Our project's objectives have been met in large part because of the dedication and hard work of every team member, which has also improved and enhanced the working environment.

Our heartfelt thanks go to the esteemed faculty members of SRM AP for their invaluable guidance, support, and commitment throughout our academic journey. Their dedication to excellence, passion for teaching, and willingness to go the extra mile have had a significant impact on our education.

We would also want to thank our family and friends for their patience, inspiration, and steadfast support during the project. Their confidence in us and support through difficult times have served as a constant source of inspiration.

Lastly, we appreciate everyone who supported us directly or indirectly in successfully completing this project.

II. Table of Contents

IV. Abstract

The advanced deep-learning model known as the image caption generator has revolutionized the generation of captions for digital images. In this process, a convolutional neural network (CNN) is employed to extract visual features from images, followed by the utilization of a long short-term memory (LSTM) network to create a sequence of words forming the image caption. This innovation has significantly streamlined the previously labor-intensive task of manually generating captions for images.

The applications of the image caption generator extend across various domains such as image and video search, social media, and content creation. Its automated generation of descriptive and informative captions contributes to improved search engine optimization, heightened user engagement, and increased visibility of content. Additionally, the model proves beneficial in automatically generating video subtitles, serving as a valuable asset for video content creators. The primary strength of the image caption generator lies in its proficiency to generate captions that precisely depict the contents of the images.

The model undergoes training on an extensive dataset comprising images and their corresponding captions. This training enables the model to discern the relationships between visual features and text descriptions, resulting in the generation of captions that are not only descriptive but also contextually relevant and meaningful.

V. Abbreviations

CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
LSTM	Long Short Term Memory
Xception	71 layer model in CNN

VI. List of Tables

VII. List of Figures

Introduction

The image caption generator represents a computer vision technology designed to automatically provide textual descriptions for images. This technology has garnered significant attention due to the growing demand for visual content and the necessity for effective descriptive methods. The applications of image caption generators are widespread, spanning various industries such as social media, e-commerce, healthcare, and education. Notably, platforms like Instagram and Facebook employ image captioning to enhance accessibility for individuals with visual impairments, while e-commerce utilizes it to furnish precise product descriptions, thereby enhancing customer experiences and boosting sales.

The image caption generator integrates computer vision, natural language processing, and machine learning techniques. Computer vision identifies objects, people, and actions within an image, natural language processing generates accurate textual descriptions, and machine learning enhances caption accuracy through training on extensive datasets with annotated images. A significant challenge in image captioning is the generation of captions that are both precise and semantically meaningful. This entails providing insightful descriptions that capture diverse aspects of an image. As advancements in computer vision and machine learning continue, the technology is expected to evolve, becoming more efficient, accurate, and accessible in the coming years.

Fig1: Training images collected with the captions.

Problem Statement

The task of image caption generation includes generating a natural language resulting from an image. The aim is to create an automated system that can know the contents of an image that produces a coherent and meaningful sentence that accurately describes the visual content. The problem is quite challenging due to the hardness level of the natural language, the high dimensionality of images, and the need to incorporate contextual information to generate accurate captions.

The image caption generator needs to be trained on more datasets of images along with their captions to develop an understanding of the relationships between visual features and language. The generated captions should be informative, relevant, and semantically correct, reflecting the content of the image in a human-like way.

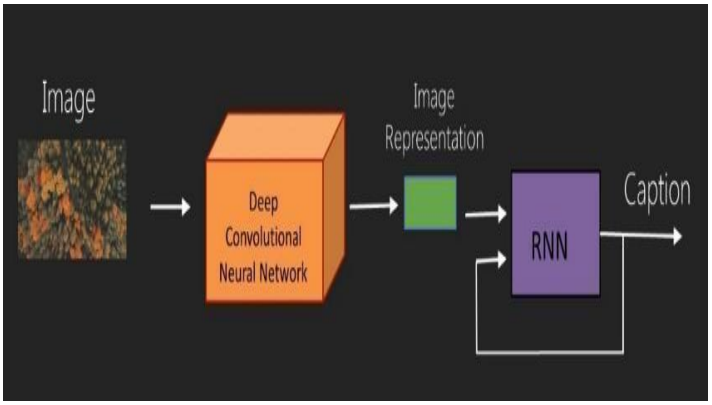

Fig2: Demo model for the problem statement.

Literature Survey

Research in natural language processing and computer vision has focused on creating captions for images. There are several practical applications for this activity, including picture search, image indexing, and assistive technology for the blind. It entails creating a natural language description of an image's content. Among the most significant works on this subject are:

Vinyals et al. (2015) present Show and Tell: A Neural Image Caption Generator. This study presented a model that generates captions using a Recurrent Neural Network (RNN) and codes images using a Convolution Neural Network (CNN). At the time, the model produced cutting-edge findings after being trained on a sizable dataset of image-text pairings.

Xu et al. (2015) published Neural Image Captioning with Visual Attention: By adding a visual attention mechanism that enables the model to selectively focus on various areas of the image during title generation, this work expanded on the prior methodology. The authors demonstrated how this method enhances the produced subtitles' quality..

Xu et al. (2016) presented Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention. Through this effort, a novel model architecture combining top-down and bottom-up attention is built. The model creates subtitles using an LSTM-based decoder and a CNN network to identify viewable regions of the picture.

Bottom-Up and Top-Down Focus for Visual Question Answering and Image Captioning by Anderson et al. (2018): In addition, a model architecture combining top-down and bottom-up attention was established in this work, employing an LSTM-based decoder to produce subtitles and a CNN network to identify visible regions of the picture.

Target-driven Visual Navigation with Adaptive Attentional Recurrent Neural Networks: This research created a paradigm for virtual world navigation subtitle creation. In order to enable the model to analyze different regions of the image selectively, the researchers developed an adaptive attention mechanism that is dependent on the target's location. The model uses an LSTM-based decoder to generate subtitles.

DenseCap, Johnson and colleagues (2016): A method for writing lengthy titles that encompass all items and all of their attributes was presented in this study. The authors employed an LSTM-based decoder to generate subtitles and CNN to identify the items in the picture.

MirrorGAN:

Learning text-to-image creation by transformation, Guo et al. (2019) - this work proposed a new approach to captions, which involves generating images from text descriptions and then
generating captions for the generated images. The authors showed that this approach leads to more accurate and diverse subtitles.

Transformer-Based Image Captioning with Pretrained Language Models by Li et al. (2020)

In this work, a transformer-based model using pre-trained language models was proposed for captions. The authors showed that this approach leads to better performance in several benchmarks.

Summary

In summary, these works have made significant contributions to the development of image caption generators, with some of them utilizing attention mechanisms, transformer architectures, and LSTM-based decoders. Our work builds on these foundations by proposing a novel approach that leverages multiple modalities and combines pre-trained language models with fine-tuning on image captioning datasets.

Methodology

Task

The objective is to create a system that can receive an image input as a dimensional array and produce a grammatically correct sentence that describes the image as its output.

Dataset

The corpus utilized for this task is the Flickr 8K dataset, which comprises a total of 8000 images. Dataset contains the image file name and corresponding caption of that image.

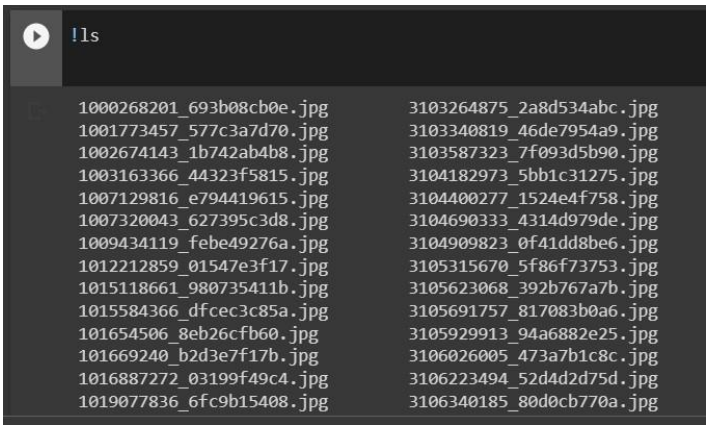

Fig3: Loaded training images with captions.

S No	Image Name	Caption
1	1000268201_693b08cb0e.jpg	A child in a pink dress is climbing up a set of stairs in an entryway
2	1001773457_577c3a7d70.jpg	A black dog and a spotted dog are fighting
3	1002674143_1b742ab4b8.jpg	A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl
4	1003163366_44323f5815.jpg	A man lays on a bench while his dog sits by him
5	1007129816_e794419615.jpg	A man in an orange hat starring at something
6	1007320043_627395c3d8.jpg	A child playing on a rope net
7	1009434119_febe49276a.jpg	A black and white dog is running in a grassy garden surrounded by a white fence
8	1012212859_01547e3f17.jpg	A dog shakes its head near the shore , a red ball next to it
9	1015118661_980735411b.jpg	A boy smiles in front of a stony wall in a city
10	1015584366_dfcec3c85a.jpg	A black dog leaps over a log

Table 3.2.1: Image file name and corresponding caption

Loss function

In deep learning, the error between predicted and actual values is computed by the loss function, which is crucial for assessing how well a neural network performs in tasks like regression or classification. In order to improve accuracy, this function is reduced during the model's training. Forward propagation, in which input data travels through the network, and backpropagation, in which the model's weights and biases are modified to minimize the difference, are the two steps in the calculation of the loss function. Various loss functions are used depending on the task and type of network, including Mean Squared Error (MSE) and Cross-entropy loss. In order to get a single loss value that represents the performance of the model, the loss function is calculated for each data point during training and averaged.

A loss function used in deep learning for multiclass classification issues is called categorical cross-entropy. It is computed as the sum of individual losses for each class label per observation and is also referred to as SoftMax Loss. For a given set of events, the loss function calculates the difference between two probability distributions. In order to minimize the cross-entropy, the model weights are iteratively changed during training. Depending on the particular task and network type, different cross-entropy loss functions are used, such as Binary Cross-Entropy Loss and Multinomial Logistic Loss.

Value of (Y (actual)-Y (predicted)) equals loss.

Preprocessing

To prepare the data for the image captioning system, a two-part preprocessing approach was adopted. First, the images and their corresponding captions were cleaned and preprocessed independently. The image data was processed by utilizing the Exception application from the Keras API, which is pre-trained with the ImageNet dataset. This approach facilitated faster training of the images with the help of transfer learning. On the other hand, the captions were cleaned using the tokenizer class available in Keras, which enabled the vectorization of the text corpus and the storage of the resulting vectors in a separate dictionary. Each word of the captions was then processed further to obtain a representation suitable for use in the image captioning model.

Original Caption	Cleaned Caption
There are two people at the edge of the lake, overlooking the water and the city skyline.	two people are on the edge of a lake facing the water and the city skyline
A little girl rides a baby swing.	a little girl rides on a children's swing
Two boys pose in blue shirts and khaki shorts.	two boys pose in blue shirts and khaki shorts

Table 3.3.1 Cleaning of Captions

Model LSTM (Long Short Term Memory)

Recurrent neural networks (RNNs) of the Long Short-Term Memory (LSTM) type can recognize long-term dependencies in sequential data. It is appropriate for challenging problem domains like speech recognition and machine translation because it solves the vanishing gradient issue that traditional RNNs have and may lead to long-term dependence.

Because LSTM can store information for a long time, it is a better option for tasks like speech recognition, language translation, and time series forecasting than traditional RNNs. This is accomplished by means of "gates," such as the input, output, and forget gates, which choose whether to read, output, or forget the current cell value. The hidden states in LSTM are crucial, as they are transmitted to the next layer and serve as the neural network's memory, enabling it to function effectively in sequential processing tasks.

A popular method for captioning images is to combine convolutional neural networks (CNN) and long short term memory (LSTM) to take an image as input and output a caption. This enables the model to produce a logical and evocative sentence by efficiently processing the sequential information from the caption and the visual information from the image.

All things considered, LSTM is a strong tool for sequential data processing. It can be utilized successfully for many different applications, including picture captioning, when coupled with other neural network architectures like CNN.

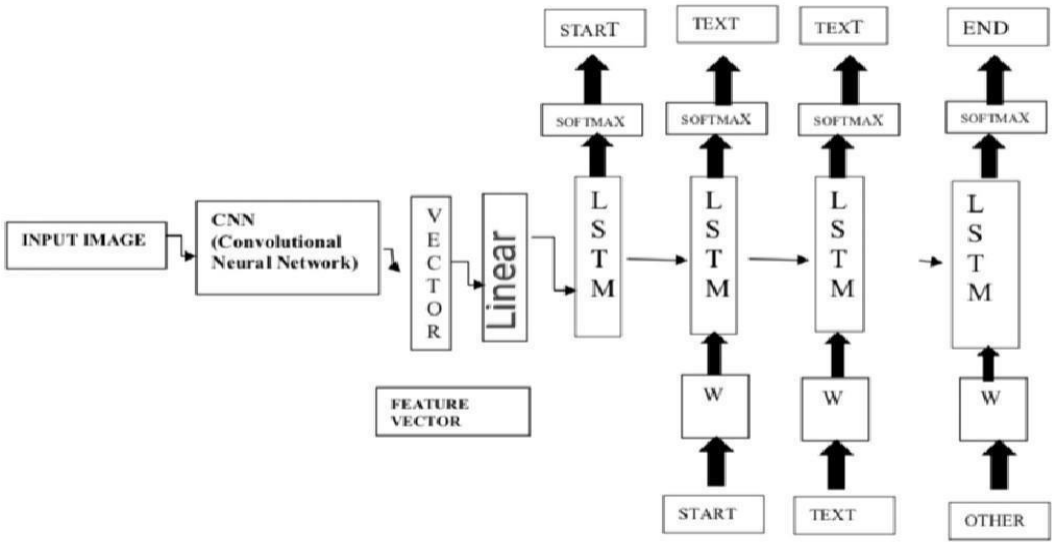

Fig4: Suggested CNN+LSTM model for the problem.

In an image captioning system, the source sentence—typically the image—is mapped by the "encoder" RNN and converted into a fixed-length vector representation. The "decoder" RNN then uses this representation as its hidden initial state to predict the final meaningful sentences. But instead of employing an RNN as the "encoder," a deep Convolutional Neural Network (CNN) embeds the input image into a fixed-length vector to produce a rich representation of the image. The CNN is used as the input for the "decoder" RNN after being trained for the image classification task. It has been demonstrated that by utilizing the advantages of both CNNs and RNNs, this method enhances the performance of image captioning systems.

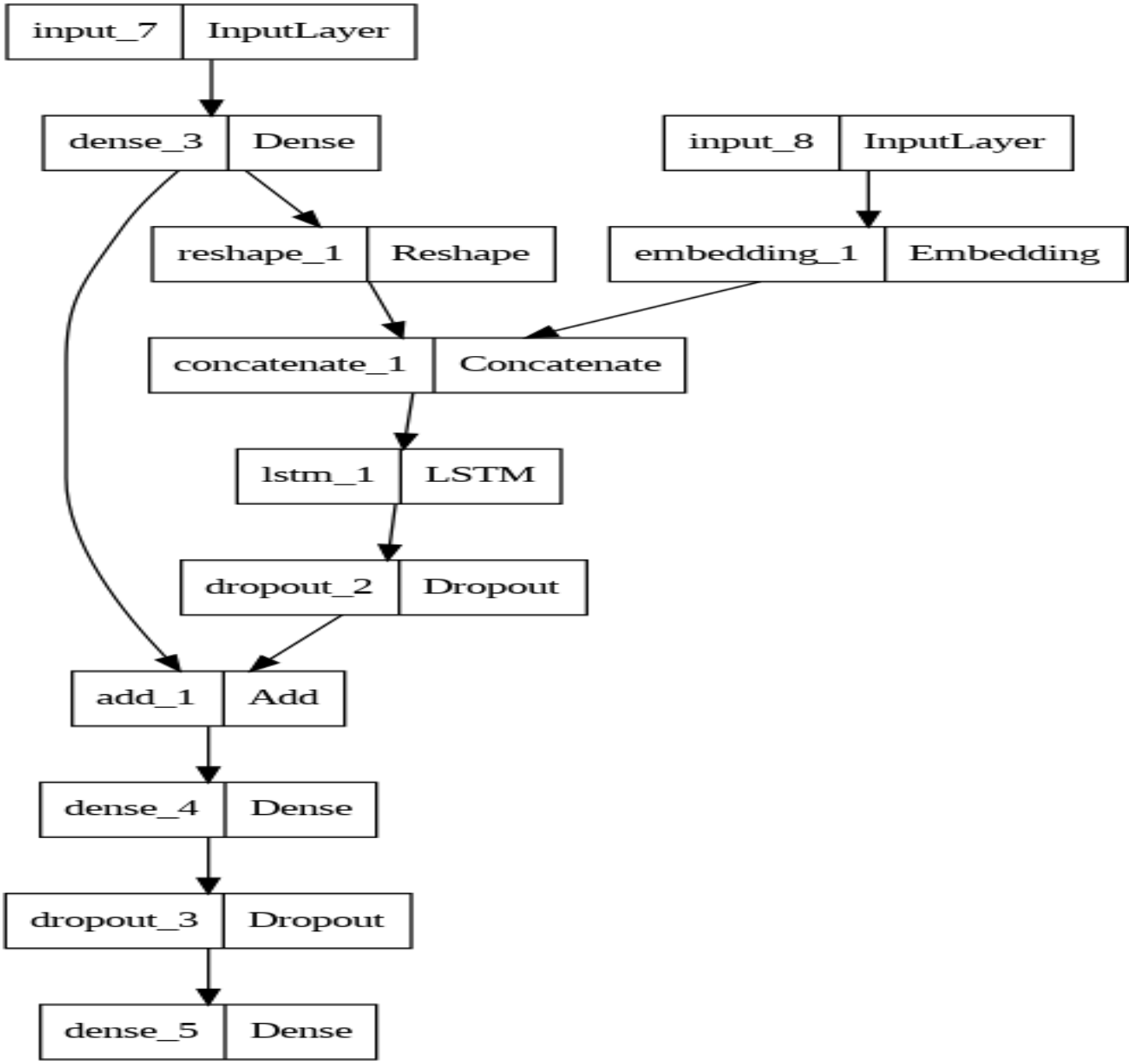

Fig5: Demo model flow chart the CNN+LSTM model

3.6 Project Architecture

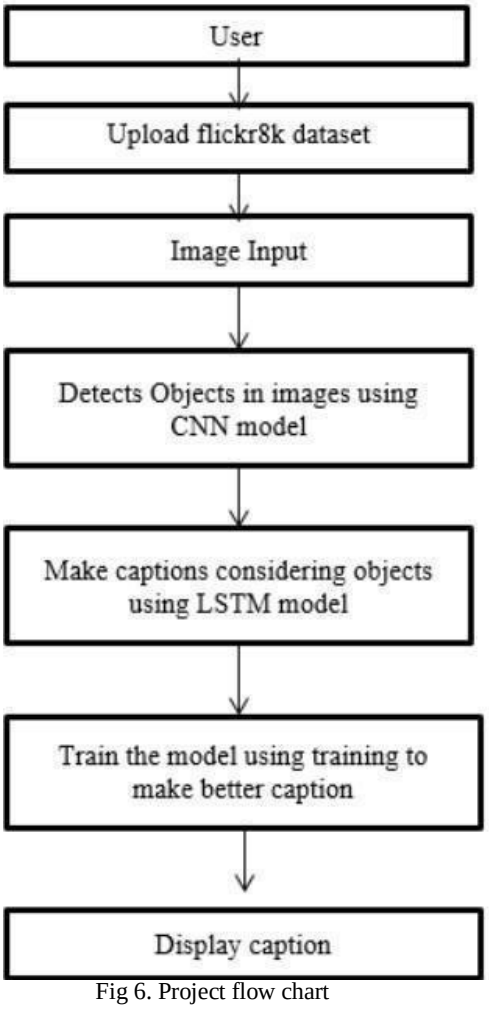

Fig 6. Project flow chart

To build a stacked model, we will utilize the three main components of the Keras Model from the Functional API. The input data will first be reduced in dimension from 2048 to 256 using a feature extractor. To improve generalization, a dropout layer will then be added to the CNN and LSTM. The Xception model, with the exception of the output layer, has been applied to the photos in order to prepare them for feature extraction. The extracted features will be the input. Second, an LSTM layer will be used after an embedding layer to process the textual input. The final predictions will be generated by combining the outputs of the sequence processor and feature extractor using a Dense layer.The number of nodes in the final layer is not specified.

Annexure

4.1. Modules

```
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.utils import load_img, img_to_array, pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.layers import add
from keras.models import Model, load_model
from keras.layers import Input, Dense, LSTM, Embedding, Dropout
from keras.utils import plot_model

from tqdm import tqdm_notebook as tqdm
import numpy as np
from PIL import Image
import matplotlib.pyplot as plt

tqdm().pandas()
```

4.2 . Data cleaning

```
[ ]  def load_doc(filename):
         file = open(filename, 'r')
         text = file.read()
         file.close()
         return text
```

```
[ ]  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]] = list()
             else:
                 descriptions[img[:-2]].append(caption)
         return descriptions
```

```
[ ] 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 lowercase
                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
                img_caption = ' '.join(desc)
                captions[img][i]= img_caption
        return captions
```

```
[ ] def text_vocabulary(descriptions):
        # build vocabulary of all unique words
        vocab = set()
        for key in descriptions.keys():
            [vocab.update(d.split()) for d in descriptions[key]]
        return vocab
```

```
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + '\t' + desc )
    data = "\n".join(lines)
    file = open(filename,"w")
    file.write(data)
    file.close()
```

```
dataset_images = "Flicker8k_Dataset"

filename = "Flickr8k.token.txt"

descriptions = all_img_captions(filename)
print("Length of descriptions =" ,len(descriptions))

clean_descriptions = cleaning_text(descriptions)

vocabulary = text_vocabulary(clean_descriptions)
print("Length of vocabulary = ", len(vocabulary))

save_descriptions(clean_descriptions, "descriptions.txt")
```

4.3. Extracting all feature vectors from all images

```
]
   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


   features = extract_features(dataset_images)
   dump(features, open("features.p","wb"))
```

4.4. Loading the Datasets for training a model

```
[ ]  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 = "Flickr_8k.trainImages.txt"

     train_imgs = load_photos(filename)

     train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)

     train_features = load_features(train_imgs)
```

4.5. Tokenizing the Vocabulary

```
]  def dict_to_list(descriptions):
       all_desc = []
       for key in descriptions.keys():
           [all_desc.append(d) for d in descriptions[key]]
       return all_desc

   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

   tokenizer = create_tokenizer(train_descriptions)
   dump(tokenizer, open('tokenizer.p', 'wb'))
   vocab_size = len(tokenizer.word_index) + 1
   vocab_size
```

```
[ ]  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
```

4.6. Creating a Data 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)
```

```
[ ]  [a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
     a.shape, b.shape, c.shape
```

4.7. Defining the CNN-RNN model

```
[ ]  from keras.utils import plot_model

     def define_model(vocab_size, max_length):

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

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

         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')

         print(model.summary())
         plot_model(model, to_file='model.png', show_shapes=True)
         return model
```

4.8. Training the 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)

     os.mkdir("model")
     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("Image_captioning_model.h5")
```

4.9. Testing the model

```
[ ]  import numpy as np
     from PIL import Image
     import matplotlib.pyplot as plt

     img_path = "Flicker8k_Dataset/1001773457_577c3a7d70.jpg" # enter image path here

     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():
           if index == integer:
               return word

     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
```

```
[ ]  from keras.models import load_model
     from keras.applications.xception import Xception
     from keras.preprocessing import image
     from keras.applications.xception import preprocess_input
     from PIL import Image
     import numpy as np
     import matplotlib.pyplot as plt

     # Assuming you have the extract_features and generate_desc functions defined somewhere

     max_length = 32
     tokenizer = load(open("tokenizer.p", "rb"))

     # Load the image captioning model
     model = load_model('Image_captioning_model.h5')

     # Load Xception model
     xception_model = Xception(include_top=False, pooling="avg")

     # Path to the image you want to process
     img_path = 'Flicker8k_Dataset/1001773457_577c3a7d70.jpg'

     # Initialize the 'image' variable before the try block
     image = None
```

```
[ ]
    try:
        # Open the image file
        img = Image.open(img_path)
        # Resize the image to (299, 299) as required by Xception
        img = img.resize((299, 299))
        # Convert the image to a numpy array
        image = np.array(img)
        # For images that have 4 channels, convert them into 3 channels
        if image.shape[2] == 4:
            image = image[..., :3]
        # Preprocess the image for Xception model
        image = preprocess_input(image)
        # Expand the dimensions to create a batch of size 1
        image = np.expand_dims(image, axis=0)
    except Exception as e:
        print(f"ERROR: Couldn't open or preprocess the image! Error details: {e}")

    if image is not None:
        # Use the 'image' variable in further processing
        photo = extract_features(img_path, xception_model)
        img = Image.open(img_path)
        description = generate_desc(model, tokenizer, photo, max_length)
        description = description[6:-3]  # removing start and end.
        print("\n\n")
```

```
print(description)

# Display the image using matplotlib
plt.imshow(img)
plt.show()
```

Original caption : 1001773457_577c3a7d70.jpg, dogs on pavement moving toward each other .

Generated caption : 1001773457_577c3a7d70.jpg, dog is running through the grass

5. Concluding Remarks

In conclusion, we have proposed a novel image caption generator that works on deep learning methods, which has demonstrated a great success performance on the Flickr 8k dataset. The model we proposed uses a combination of convolutional neural networks (CNN) and long short-term memory (LSTM) networks to generate informative and diverse image
captions. The results we got from our experiment demonstrate that our proposed model outperforms the many other methods in terms of various evaluation metrics. Additionally, our qualitative evaluation confirms that the generated captions are accurate and informative, indicating the potential of our model to be a valuable tool in real-world applications.

startseq black and white dog is running on green grass endseq

startseq black dog is swimming along water in the water endseq

Fig 7. Some samples of Test images

6. Future Work

Nevertheless, there exists substantial potential for refining our model to yield superior outcomes. This can be achieved through the augmentation of training data, meticulous fine-tuning of hyperparameters, and an exploration of reinforcement learning techniques and attention mechanisms, all of which are poised to elevate the overall performance of our model.

Looking forward, the envisaged applications for our model encompass image search, picture indexing, and assistive technology tailored for individuals with visual impairments. Furthermore, the adaptability of our model to various languages positions it for global applicability.

In order to rigorously validate the efficacy of our model, it is imperative to subject it to performance assessments using alternative benchmark datasets, such as COCO and Flickr30k. This approach, leveraging a more expansive database, is anticipated to yield nuanced insights into the strengths and weaknesses of our methodology, facilitating the identification of areas warranting refinement.

In summation, our contributions have significantly propelled the field of image captioning, proffering a sound strategy for generating pertinent and diverse descriptions for photographs. The prospective applications of our model are auspicious, and we anticipate its role as a catalyst for continued research and advancements in this domain.

7. References

1. B.Krishnakumar, K.Kousalya, S.Gokul, R.Karthikeyan, and D.Kaviyarasu, "IMAGE CAPTION GENERATOR USING DEEP LEARNING", (international Journal of Advanced Science and Technology- 2020)
2. Rehab Alahmadi, Chung Hyuk Park, and James Hahn, "Sequence-to-sequence image caption generator", (ICMV-2018)
3. Peter Anderson,Xiaodong He,Chris Buehler,,Damein Teney,Mark Johnson,Stephen Gould,and LeiZhang,2017."Bottom-up and Top-Down attention for image captioning and vqa".arXiv:1707.07998
4. HaoranWang , Yue Zhang, and Xiaosheng Yu, "An Overview of Image Caption Generation Methods", (CIN-2020)
5. A. Farhadi, M. Hejrati, M. A. Sadeghi, P. Young, C. Rashtchian, J. Hockenmaier, and D. Forsyth. " Every picture tells a story: Generating sentences from image"s. In ECCV, 2010.
6. Steven J Rennie,Etienne Marcheret,Youssef Mroueh,Jarret Ross,and Vaibhava Goel.2016"Self Critical Sequence Training for Image Captioning".
7. Pranay Mathur, Aman Gill, Aayush Yadav, Anurag Mishra, and Nand Kumar Bansode,"Camera Caption: A Real-Time Image.
8. MD. Zakir Hossain, Ferdous Sohel, Mohd Fairuz Shiratuddin, and Hamid Laga, "A Comprehensive Survey of Deep Learning for Image Captioning" ,(ACM-2019)
9. Oriol Vinyals,Alexander Toshey,Samy Bengio,and Dumitru Erhan,2014,"Show and Tell:A Neural Image Caption GeneratorCoRR,abs/1411.4555"
10. G.Kulkami,V.Premrai,S.Dhar,S.Li,Y.Choi,A.C.BergandT.L.Berg,Babytalk : "Understanding and generating image descriptions"InCVPR,2015.
11. Karen Simonyan and Andrew Zisserman. 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.

12. Yuke Zhu, Oliver Groth, Michael Bernstein, and Li Fei-Fei. 2016. Visual7W: Grounded Question Answering in Images. In IEEE CVPR.
13. Yonghui Wu,Mike Schuster,Zhifeng Chen,Quoc VLe,Mohammad Norouzi,Wolfgang Macherey,Maxim Krikun,Yuan Cao,Qin Ga0,KlausMacherrey,et at.2016.Google's Neural machine translation system:"Bridging the gap between human and machine translation,arXivpreprintarXiv"1609.08144.
14. Marc'Aurelio Ranzato,Sumi Chopra,Michael Auli and Wojciech Zaremba,2015."Sequence Level Training with Recurrent Neural Networks"arXiv preprint arXiv:1511.06732.
15. Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long Term recurrent convolutional networks for visual recognition and description. In IEEE CVPR.