# DAYANANDA SAGAR UNIVERSITY


MINOR PROJECT

REPORT

ON

IMAGE CAPTION GENERATOR

BACHELOR OF

TECHNOLOGY

IN

COMPUTERSCIENCE & ENGINEERING

Submitted by

NAVANEETH KRISHNAN NAIR (ENG19CS0201)

PAVITRA ANKESH (ENG19CS0221)

PRANAV AVINASH MAHAMUNI (ENG19CS0225)

PRANAV NAGARAJ GHATIGAR (ENG19CS0227)

5th SEMESTER, 2021

Under the Guidance of

Dr. Kiran B
Designation

# DAYANANDA SAGAR UNIVERSITY

## School of Engineering, Kudlu Gate ,Bangalore-560068


# CERTIFICATE

*This is to certify that the minor project report entitled "Image Caption Generator" being submitted by Mr/Ms Navaneeth Krishnan Nair, Pavitra Ankesh, Pranav Avinash Mahamuni, Pranav Nagaraj Ghatigar bearing USN ENG19CS0201, ENG19CS0221, ENG19CS0225, ENG19CS0227 has* satisfactorily completed her Minor Project as prescribed by the University for the 5th semester B.Tech Program in Computer Science & Engineering during the academic year 2021 – 22 at the School of Engineering, Dayananda Sagar University, Bangalore.

Date: Signature of the faculty in-charge

Signature of Chairman
Department of Computer Science & Engineering

# DECLARATION

We hereby declare that the work presented in this minor project entitled as “Image Caption Generator”, has been carried out by us and it has not been submitted for the award of any degree, diploma or the minor project of any other college or university.

NAVANEETH KRISHNAN NAIR (ENG19CS0201)
PAVITRA ANKESH (ENG19CS0221)
PRANAV AVINASH MAHAMUNI (ENG19CS0225)
PRANAV NAGARAJ GHATIGAR (ENG19CS0227)

# ACKNOWLEDGEMENT

The success and final outcome of this software requirement document required a lot of guidance and assistance from many people and we are extremely privileged to have got this all through the completion of the project. All that we have done is only due to such supervision and assistance and we would not forget to thank them.

We respect and thank our mentor, Professors, and the Chairman for providing us an opportunity to do the Software Requirement Document and giving us all support and guidance which made me complete the project duly. We are extremely thankful to all for providing such nice support and guidance.

We are especially thankful to our **Dean & Chairman, Dr. A Srinivas**, Department of Computer Science & Engineering who continuously helped us throughout the project and without his guidance, this project would have been an uphill task.

We are pleased to acknowledge Dr. Kiran B, Department of Computer Science & Engineering for her invaluable guidance, support, motivation and patience during the course of this mini-project work.

We are thankful for and fortunate enough to get constant encouragement, support and guidance from all Teaching staff of the Computer Science Engineering department, which helped us in completing our report. Also, we would like to extend our sincere esteems to all staff in the laboratory for their timely support.

NAVANEETH KRISHNAN NAIR (ENG19CS0201)
PAVITRA ANKESH (ENG19CS0221)
PRANAV AVINASH MAHAMUNI (ENG19CS0225)
PRANAV NAGARAJ GHATIGAR (ENG19CS0227)

# ABSTRACT

The objective of this project is to systematically analyse a deep neural network based on image caption generation. Image caption generator is a fundamental task that involves computer vision and natural language processing concepts to recognize the context of an image and describe them in a natural language like English. The goal of image caption generation is to convert a given input image into a natural language description.

This python project is implemented using two components: Convolutional Neural Network (CNN) model and Long Short Term Memory (LSTM) model. The CNN-LSTM architecture involves the Convolutional Neural Network (CNN) which is used to generate features describing the images and the Long Short Term Memory (LSTM) which is a type of Recurrent Neural Network (RNN) used to accurately structure meaningful sentences using the generated information.

Automatically describing the content of images has several applications such as usage in virtual assistants, help visually impaired people better understand the content of images, and provide more accurate and compact information of images for social media.

# TABLE OF CONTENTS

# TABLE OF FIGURES

# INTRODUCTION

Image caption generator is a task that involves computer vision and natural language processing concepts to recognize the context of an image and describe them in a natural language like English.

## Problem Statement:

The objective of our project is to learn the concepts of a CNN and LSTM model and build a working model of Image caption generator by implementing CNN with LSTM.

In this Python project, we will be implementing the caption generator using CNN (Convolutional Neural Networks) and LSTM (Long short term memory). The image features will be extracted from Xception which is a CNN model trained on the imagenet dataset and then we feed the features into the LSTM model which will be responsible for generating the image captions.

# Literature Survey

## Paper-1

- TITLE – Image Captioning With Semantic Attention
- AUTHOR - Quanzeng You, Hailin Jin, Zhaowen Wang, Chen Fang, and Jiebo Luo
- PUBLISHED IN – 2016
- SUMMARY – The two traditional approaches are top-down in which starts from a gist of the image and converts it into words, bottom-up first comes up with words describing various aspects of the image and then combines them. Here, both approaches are combined to come up with a semantic attention model which outperforms the other methods**.**

## Paper-2

- TITLE – Show, Attend and Tell: Neural Image Caption Generation with Visual Attention
- AUTHOR - Kelvin Xu, Jimmy Lei Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, Yoshua Bengio
- PUBLISHED IN – 2015
- SUMMARY – Describes how we can train this model in a deterministic manner using standard backpropagation techniques and stochastically by maximizing a variational lower bound.The two models which are considered are encoder and decoder.The varities of convolutional feature extractor available, how to split the result to get the most accurate predictions.

# Requirement Analysis

This project requires good knowledge of Deep learning, Python, Working on Jupyter notebooks, Keras library, Numpy and Natural language processing.

For the image caption generator, we will be using the “Flickr_8K” dataset. The advantage of a huge dataset is that we can build better models.

It uses two files:

- Flicker8k_Dataset - Contains Pictures
- Flickr_8k_text- Contains captions

# Design Model

## 8.1 Algorithm:

**CNN:** Convolutional Neural networks are specialized deep neural networks which can process the data that has input shape like a 2D matrix. Images are easily represented as a 2D matrix and CNN is very useful in working with images.

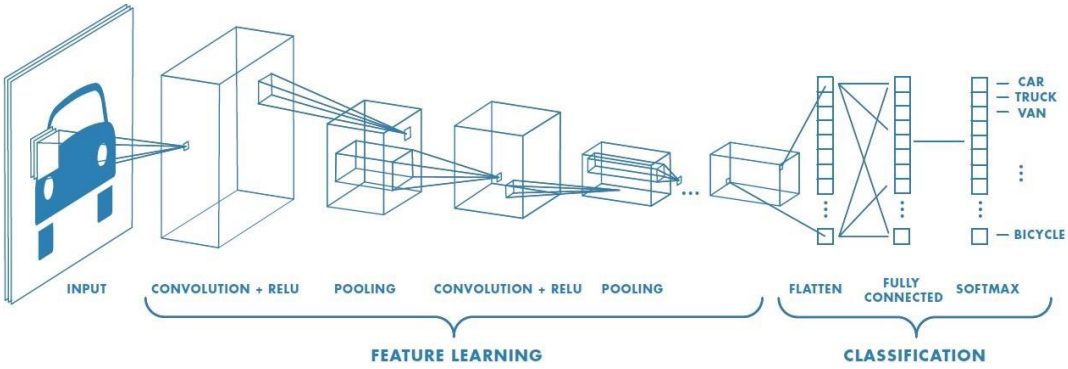


Figure 4.1.1 CNN

**LSTM**: LSTM stands for **Long short term memory**, they are a type of RNN **(**recurrent neural network**)** which is well suited for sequence prediction problems. Based on the previous text, we can predict what the next word will be. It has proven itself effective from the traditional RNN by overcoming the limitations of RNN which had short term memory. LSTM can carry out relevant information throughout the processing of inputs and with a forget gate, it discards non-relevant information.

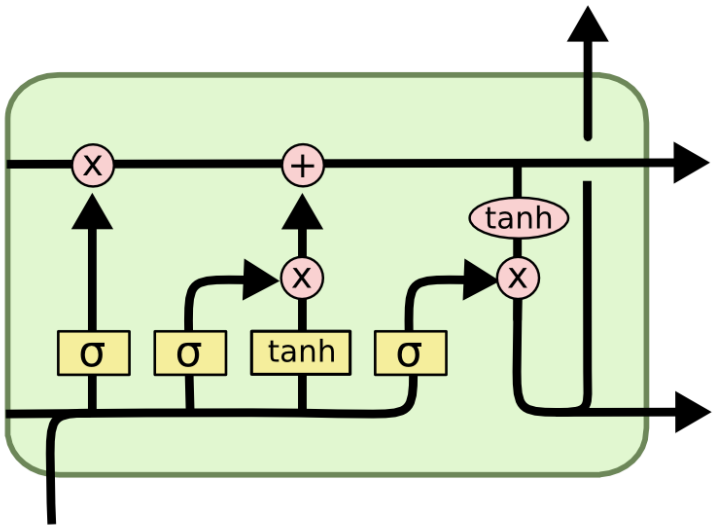


Figure 4.1.2 LSTM

## 8.2 Architecture Diagram:

To make our image caption generator model, we will be merging these architectures. It is also called a CNN-RNN model.

- CNN is used for extracting features from the image. We will use the pre-trained model Xception.
- LSTM will use the information from CNN to help generate a description of the image.

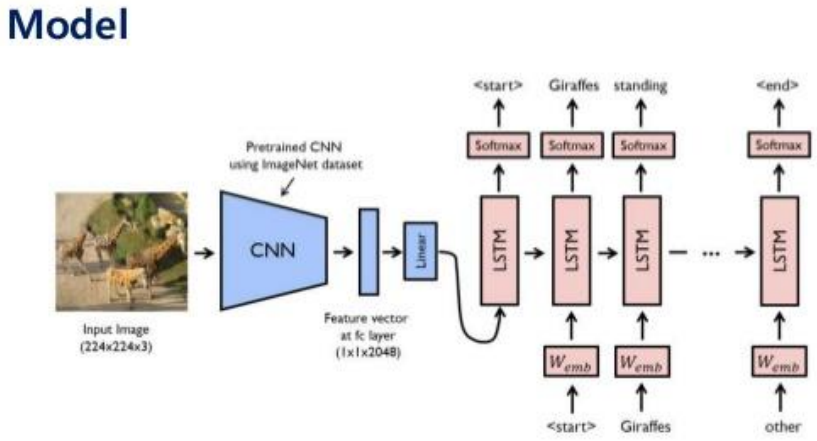


Figure 4.2.1 Architecture

## 8.3 Flowchart:

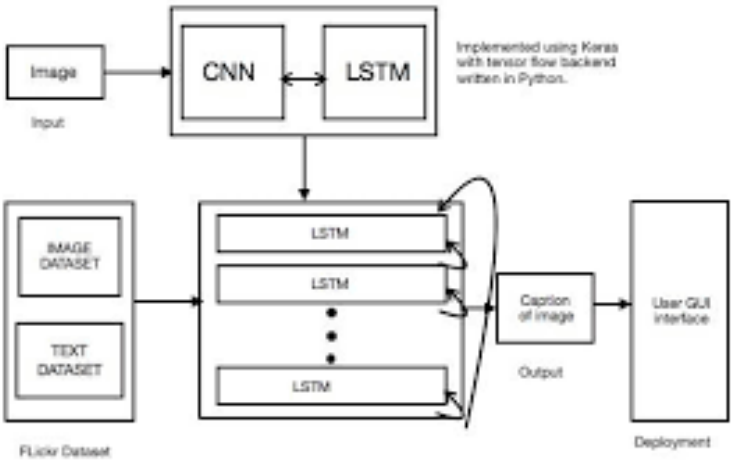


Figure 4.3.1 Flowchart

# Project Breakdown

**1) First, we import all the necessary packages**

**2) Getting and performing data cleaning**

**3) Extracting the feature vector from all images**

**4) Loading dataset for Training the model**

**5) Tokenizing the vocabulary**

**6) Create Data generator**

**7) Defining the CNN-RNN model**

**8) Training the model**

**9) Testing the model**

**10) Execution**

# IMPLEMENTATION

## Source Code:

## Image Data:

```
from os import listdir
from pickle import dump
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
# extract features from each photo in the directory
def extract_features(directory):
        model = Xception( include_top=False, pooling='avg' )
        features = {}
        for img in os.listdir(directory):
            filename = directory + "/" + img
            image = Image.open(filename)
            image = image.resize((299,299))
            image = np.expand_dims(image, axis=0)
            image = image/127.5
            image = image - 1.0
            feature = model.predict(image)
            features[img] = feature
        return features
# extract features from all images
directory = r'C:\Users\Pavitra\Documents\Image Caption
Generator\Flickr8k_Dataset\Flicker8k_Dataset'
features = extract_features(directory)
print('Extracted Features: %d' % len(features))
dump(features, open('features.pkl', 'wb'))
```

## Text Data:

```
import string
# load doc into memory
def load_doc(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
# extract descriptions for images
def load_descriptions(doc):
    mapping = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        if len(line) < 2:
            continue
```

```
        image_id, image_desc = tokens[0], tokens[1:]
        image_id = image_id.split('.')[0]
        image_desc = ' '.join(image_desc)
        if image_id not in mapping:
            mapping[image_id] = list()
        mapping[image_id].append(image_desc)
    return mapping
def clean_descriptions(descriptions):
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            desc = desc.split()
            desc = [word.lower() for word in desc]
            desc = [w.translate(table) for w in desc]
            desc = [word for word in desc if len(word)>1]
            desc = [word for word in desc if word.isalpha()]
            desc_list[i] =  ' '.join(desc)
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc
# save descriptions to file, one per line
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
filename = r'C:\Users\Pavitra\Documents\Image Caption
Generator\Flickr8k_text\Flickr8k.token.txt'
doc = load_doc(filename)
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
clean_descriptions(descriptions)
vocabulary = to_vocabulary(descriptions)
print('Vocabulary Size: %d' % len(vocabulary))
save_descriptions(descriptions, 'descriptions.txt')
```

## Training:

```
from numpy import array
from pickle import load,dump
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
from keras.utils import to_categorical
from keras.utils import plot_model
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
from keras.callbacks import ModelCheckpoint
import os
# load doc into memory
def load_doc(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
#load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        if len(tokens)<1 :
            continue
        image_id, image_desc = tokens[0], tokens[1:]
        if image_id in dataset:
            if image_id not in descriptions:
                descriptions[image_id] = list()
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            descriptions[image_id].append(desc)
    return descriptions
# load photo features
def load_photo_features(filename,dataset):
    all_features = load(open(filename, 'rb'))
    features = {k:all_features[k] for k in dataset}
    return features
# covert a dictionary of clean descriptions to a list of descriptions
def to_lines(descriptions):
    all_desc = list()
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc
# fit a tokenizer given caption descriptions
def create_tokenizer(descriptions):
```

```
    lines = to_lines(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)
    return tokenizer
# calculate the length of the description with the most words
def max_length(descriptions):
    lines = to_lines(descriptions)
    return max(len(d.split()) for d in lines)
def create_sequences(tokenizer, max_length, desc_list, photo, vocab_size):
    X1, X2, y = list(), list(), list()
    for desc in desc_list:
        seq = tokenizer.texts_to_sequences([desc])[0]
        for i in range(1, len(seq)):
            in_seq, out_seq = seq[:i], seq[i]
            in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]
            X1.append(photo)
            X2.append(in_seq)
            y.append(out_seq)
    return array(X1), array(X2), array(y)
# define the captioning 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
# data generator, intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, tokenizer, max_length, vocab_size):
    while 1:
        for key, desc_list in descriptions.items():
            photo = photos[key][0]
            in_img, in_seq, out_word = create_sequences(tokenizer, max_length,
desc_list, photo, vocab_size)
            yield [in_img, in_seq], out_word
# load training dataset (6K)
filename = r'C:\Users\Pavitra\Documents\Image Caption
Generator\Flickr8k_text\Flickr_8k.trainImages.txt'
train = load_photos(filename)
print('Dataset: %d' % len(train))
```

```
train_descriptions = load_clean_descriptions('descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
train_features = load_photo_features('features.pkl', train)
print('Photos: train=%d' % len(train_features))
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.pkl', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
max_length = max_length(train_descriptions)
print('Description Length: %d' % max_length)
# define the model
model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
os.mkdir("models")
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer,
max_length, vocab_size)
    model.fit(generator, epochs=1, steps_per_epoch=steps, verbose=1)
    model.save('models/model_' + str(i) + '.h5')
```

# TESTING

## Evaluation:

```
from numpy import argmax
from pickle import load
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import load_model
from nltk.translate.bleu_score import corpus_bleu
# load doc into memory
def load_doc(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
#load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos
# load clean descriptions into memory
def load_clean_descriptions(filename, dataset):
    doc = load_doc(filename)
    descriptions = dict()
    for line in doc.split('\n'):
        tokens = line.split()
        if len(tokens)<1 :
            continue
        image_id, image_desc = tokens[0], tokens[1:]
        if image_id in dataset:
            if image_id not in descriptions:
                descriptions[image_id] = list()
            desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
            descriptions[image_id].append(desc)
    return descriptions
# load photo features
def load_photo_features(filename,dataset):
    all_features = load(open(filename, 'rb'))
    features = {k:all_features[k] for k in dataset}
    return features
# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
```

```
    in_text = 'startseq'
    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 = argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'endseq':
            break
    return in_text
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
    actual, predicted = list(), list()
    for key, desc_list in descriptions.items():
        yhat = generate_desc(model, tokenizer, photos[key], max_length)
 filename = r'C:\Users\Pavitra\Documents\Image Caption
Generator\Flickr8k_text\Flickr_8k.testImages.txt'
test = load_photos(filename)
print('Dataset: %d' % len(test))
test_descriptions = load_clean_descriptions('descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
test_features = load_photo_features('features.pkl', test)
print('Photos: test=%d' % len(test_features))
tokenizer = load(open('tokenizer.pkl', 'rb'))
max_length=34
filename = 'models\model_9.h5'
model = load_model(filename)
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
references = [d.split() for d in desc_list]
        actual.append(references)
        predicted.append(yhat.split())
    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)))
    print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3,
0)))
    print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25,
0.25, 0.25)))
```

## Testing:

```
from pickle import load
import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.applications.xception import Xception
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import Model
from keras.models import load_model
```

```
from PIL import Image
import matplotlib.pyplot as plt
# load doc into memory
def load_doc(filename):
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
#load the data
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos
# extract features from each photo in the directory
def extract_features(filename):
    try:
        image = Image.open(filename)
    except:
        print("ERROR: Couldn't open image! Make sure the image path and
extension is correct")
    model = Xception(include_top=False, pooling="avg")
    image = image.resize((299,299))
    image = np.array(image)
    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
# map an integer to a word
def word_for_id(integer, tokenizer):
    for word, index in tokenizer.word_index.items():
        if index == integer:
            return word
    return None
# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'startseq'
    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 == 'endseq':
            break
```

```
    return in_text
# load the tokenizer
tokenizer = load(open('tokenizer.pkl', 'rb'))
max_length = 34
model = load_model('models\model_3.h5')
figure = plt.figure(figsize=(20, 20))
test=load_photos(r'C:\Users\Pavitra\Documents\Image Caption
Generator\Flickr8k_text\Flickr_8k.testImages.txt')
p=0
for i in range(464,470):
    sub_img = figure.add_subplot(3, 2, p+1)
    img_path=r'C:\Users\Pavitra\Documents\Image Caption
Generator\Flickr8k_Dataset\Flicker8k_Dataset'
    img_path=img_path+'\\'+test[i]
    photo = extract_features(img_path)
    img = Image.open(img_path)
    description = generate_desc(model, tokenizer, photo, max_length)
    sub_img.imshow(img)
    sub_img.set_title("\n"+description)
    p+=1
plt.show()
```

# RESULTS

## BLEU Scores:

BLEU-1: 0.556233
BLEU-2: 0.311220
BLEU-3: 0.215607
BLEU-4: 0.104051

## Sample Outputs:


Figure 8.1: Sample output 1


Figure 8.2: Sample output 2


Figure 8.3: Sample output 3


Figure 8.4: Sample output 4

# CONCLUSION AND FUTURE WORK

In this project, we have developed all aspects of the image captioning task. We have implemented a simple neural network with a predefined Convolutional Neural Network (CNN) and a Long Short Term Memory (LSTM) architecture. The CNN model generates features describing the images and the LSTM structures meaningful sentences. The model is trained over a vast dataset with over 8,000 images. The Bilingual Evaluation Understudy (BLEU) Score is used to evaluate the generated caption. Hence, the goals of this project has been achieved.

Further, we would like to improve on the following: The speed of training, testing, and generating sentences for the model should be optimized to improve performance. The model should be able to generate description sentences corresponding to multiple main objects for images with multiple target objects, instead of just describing a single target object. Creation of an interactive an efficient way to generate captions for new images.

# REFERENCES


- https://www.youtube.com/watch?v=UtlnGkqk1VU&list=PL12YWfULs0pnL_9Pj6udM6PE2-SWZbTlX
- https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/
- https://www.analyticsvidhya.com/blog/2018/04/solving-an-image-captioning-task-using-deep-learning/
- https://towardsdatascience.com/a-guide-to-image-captioning-e9fd5517f350
- https://www.hindawi.com/journals/cin/2020/3062706/
- https://iq.opengenus.org/automatic-image-annotation/