VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme : **M.Tech Int. Software Engineering**

Course Title : **Digital Image Processing**

Course Code : **SWE – 1010**

Slot : **F1 + TF1**

Title: Image Caption Generator

Team Members: **Karthikeyan.S 20MIS1065**

Karthikeyan.R 20MIS1122

Faculty: Dr. Geetha **Sign:**

Date:

ABSTRACT

In this project, we use CNN and LSTM to identify the caption of the image. As the deep learning techniques are growing, huge datasets and computer power are helpful to make models that can generate captions for an image. This is what we're going to apply in this Python based project where we will be using deep learning techniques like CNN and RNN. Image caption generator is a process which involves natural language processing and computer vision concepts to recognize the context of an image and present it in English. In this project, we carefully follow some of the core concepts of image captioning and its common approaches. We discuss Keras library, numpy and Kaggle notebook for the making of this project. We also discuss about flickr_dataset and CNN used for image classification.

TABLE OF CONTENTS

INTRODUCTION

Every day, we encounter many images from various sources such as the internet, news articles, document diagrams and advertisements. These sources contain images that viewers would have to interpret themselves. Most images do not have a description, but the human can largely understand them without their detailed captions. However, machine needs to interpret some form of image captions if humans need automatic image captions from it. Image captioning is important for many reasons. Captions for every image on the internet can lead to faster and descriptively accurate images searches and indexing.

Ever since researchers started working on object recognition in images, it became clear that only providing the names of the objects recognized does not make such a good impression as a full human-like description. As long as machines do not think, talk, and behave like humans, natural language descriptions will remain a challenge to be solved.

Image captioning has various applications in various fields such as biomedicine, commerce, web searching and military etc. Social media like Instagram, Facebook etc. can generate captions automatically from images.

1.1 Motivation:

Generating captions for images is a vital task relevant to the area of both Computer Vision and Natural Language Processing. Mimicking the human ability of providing descriptions for images by a machine is itself a remarkable step along the line of Artificial Intelligence. The main challenge of this task is to capture how objects relate to each other in the image and to express them in a natural language (like English). Traditionally, computer systems

have been using pre-defined templates for generating text descriptions for images. However, 1 this approach does not provide sufficient variety required for generating lexically rich text descriptions. This shortcoming has been suppressed with the increased efficiency of neural networks. Many states of art models use neural networks for generating captions by taking image as input and predicting next lexical unit in the output sentence.

1.2 Problem Statement:

To develop a system for users, which can automatically generate the description of an image with the use of CNN along with LSTM.

Automatically describing the content of images using natural language is a fundamental and challenging task. With the advancement in computing power along with the availability of huge datasets, building models that can generate captions for an image has become possible. On the other hand, humans are able to easily describe the environments they are in. Given a picture, it's natural for a person to explain an immense number of details about this image with a fast glance. Although great development has been made in computer vision, tasks such as recognizing an object, action classification, image classification, attribute classification and scene recognition are possible but it is a relatively new task to let a computer describe an image that is forwarded to it in the form of a human-like sentence.

For this goal of image captioning, based on semantics of images should be captured here and expressed in the desired form of natural languages. It has a great impact in the real world, for instance by helping visually impaired people better understand the content of images on the web.

So, to make our image caption generator model, we will be merging CNN-RNN architectures. Feature extraction from images is done using CNN. We have used the pre-trained model Exception. The information received from CNN is then used by LSTM for generating a description of the image.

However, sentences that are generated using these approaches are usually generic descriptions of the visual content and background information is ignored. Such generic descriptions do not satisfy in emergent situations as they, essentially replicate the information present in the images and detailed descriptions regarding events and entities present in the images are not provided, which is imperative to understanding emergent situations.

The objective of our project is to develop a web-based interface for users to get the description of the image and to make a classification system in order to differentiate images as per their description. It can also make the task of SEO easier which is complicated as they have to maintain and explore enormous amounts of data.

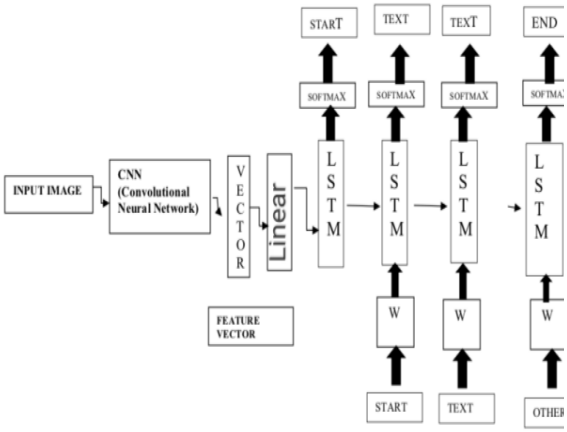
2.SOFTWARE REQUIREMENTS:

Operating system: win7/above
Data base: Flickr
Software: Kaggle Notebook
Language: Python

HARDWARE REQUIREMENTS:

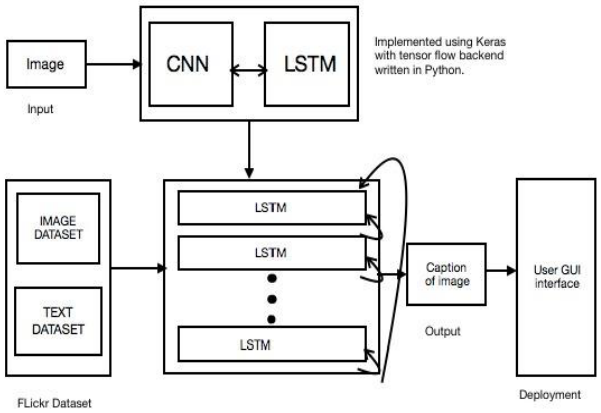
Processor: pentium-iv
Processor speed: 2.4ghz
Hard disk: 50GB
Ram: 2GB

3.SYSTEM ARCHITECTURE



The proposed model of Image Caption Generator is as shown in the above figure. Here in this model, input image is given & then A convolutional neural network is used to create a dense feature vector as shown in figure. This dense vector, also called an embedding, this vector can be used as input into other algorithms, and it generates suitable caption for given image as output. For an image caption generator, this embedding becomes a representation of the image and used as the initial state of the LSTM for generating meaningful captions, for the image.

This is how our proposed system architecture will look like,



4.ALGORITHMS

4.1 Convolutional Neural Network

Convolutional Neural networks are specialized deep neural networks which processes the data that has input shape like a 2D matrix. CNN works well with images and are easily represented as a 2D matrix. Image classification and identification can be easily done using CNN. It can determine whether an image is a bird, a plane or Superman,

etc. Important features of an image can be extracted by scanning the image from left to right and top to bottom and finally the features are combined together to classify images. It can deal with the images that have been translated, rotated, scaled and changes in perspective.

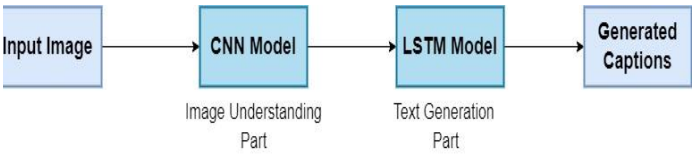
4.2 Long Short-term Memory

LSTM are type of RNN (Recurrent Neural Network) which is well suited for sequence prediction problems. We can predict what the next words will be based on the previous text. It has shown itself effective from the traditional RNN by overcoming the limitations of RNN. LSTM can carry out relevant information throughout the processing, it discards non-relevant information.

4.3 Data Exploration

For the image caption generator, we have used the Flickr_8K dataset. There are also other big datasets like Flickr_30K and MSCOCO dataset but it can take weeks for systems having only CPU support just to train the network, so we used a small Flickr8k dataset. Using a huge dataset helps in developing a better model.

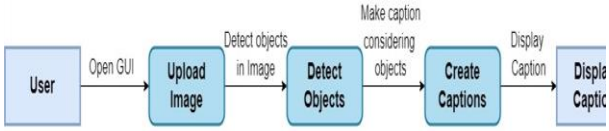
5.METHODOLOGY



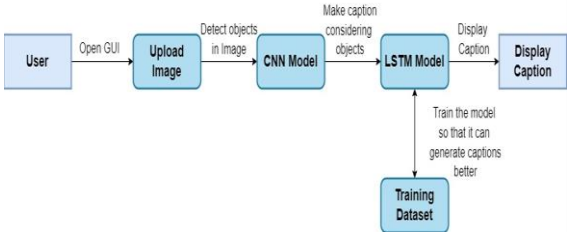
This Figure shows Block Diagram of the system. When we upload the image, CNN will identify the objects present in the image then LSTM will start preparing captions considering the objects present in the image using Training Dataset,

which comprises of Image Data set and Text Data Set, after the training a suitable caption will be generated and displayed top the user.

DFD Diagram Level 0

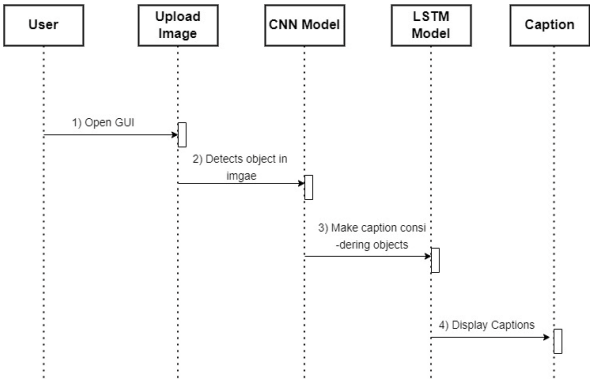


DFD Diagram Level 1



DFD Diagram Level 2

Here we have shown the DFD's of our system (i.e., Data Flow Diagrams). DFD's provide us the basic overview of the whole Image Caption Generator System or process being analysed or modelled.



The Figure shows Sequence Diagram of the system. First user will browse the site. Then he will upload the image, CNN will identify the objects present in the image then LSTM will start preparing captions considering the objects present in the image using Training Dataset, which comprises of Image Data set and Text Data Set, after the training a suitable caption will be generated and displayed top the user.

The proposed system of Image Caption Generator has the capabilities to Generate Captions for the Images, provided during the Training purpose & also for the new images as well. Our Model takes an Image as Input and by analysing the image it detects

objects present in an image and create a caption which describes the image well enough for any machine to understand what an image is trying to say.

6.IMPLEMENTATION

Pre-Requisites:

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

The necessary libraries that are required are:

- pip install tensorflow
- keras
- pillow
- numpy
- tqdm

Project File Structure:

- **Flickr8k_Dataset** – Dataset folder which contains 8091 images.
- **Flickr_8k_text** – Dataset folder which contains text files and captions of images.

1. First, we import all the necessary packages.

Code:

```
import os
import pickle
import numpy as np
from tqdm.notebook import tqdm

from tensorflow.keras.applications.v
gg16 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
from tensorflow.keras.models import
Model
from tensorflow.keras.utils import t
o_categorical, plot_model
from tensorflow.keras.layers import
Input, Dense, LSTM, Embedding, Dropo
ut, add
```

```
BASE_DIR = '/kaggle/input/flickr8k'
WORKING_DIR = '/kaggle/working'
```

2. Extract Image Features.

Load VGG16() Model:

```
model = VGG16()
# restructure the model
model = Model(inputs=model.inputs, o
utputs=model.layers[-2].output)
# summarize
print(model.summary())
```

Output:

```
2022-09-04 14:00:28.836414: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:28.936997: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:28.937990: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:28.939357: I tensor
flow/core/platform/cpu_feature_guard
.cc:142] This TensorFlow binary is o
ptimized with oneAPI Deep Neural Net
work Library (oneDNN) to use the fol
lowing CPU instructions in performan
ce-critical operations:  AVX2 AVX512
F FMA
To enable them in other operations,
rebuild TensorFlow with the appropri
ate compiler flags.
2022-09-04 14:00:28.940494: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:28.941223: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:28.941844: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:31.207292: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:31.208154: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:31.208806: I tensor
flow/stream_executor/cuda/cuda_gpu_e
xecutor.cc:937] successful NUMA node
read from SysFS had negative value (
-1), but there must be at least one
NUMA node, so returning NUMA node ze
ro
2022-09-04 14:00:31.209454: I tensor
flow/core/common_runtime/gpu/gpu_dev
ice.cc:1510] Created device /job:loc
alhost/replica:0/task:0/device:GPU:0
with 15403 MB memory:  -> device: 0,
name: Tesla P100-PCIE-16GB, pci bus
id: 0000:00:04.0, compute capability
: 6.0
Downloading data from https://storag
e.googleapis.com/tensorflow/keras-ap
plications/vgg16/vgg16_weights_tf_di
m_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
_________________________________
_________________________________
block2_conv2 (Conv2D)        (None,
112, 112, 128)     147584
_________________________________
_________________________________
block2_pool (MaxPooling2D)   (None,
56, 56, 128)       0
_________________________________
_________________________________
block3_conv1 (Conv2D)        (None,
56, 56, 256)       295168
_________________________________
_________________________________
block3_conv2 (Conv2D)        (None,
56, 56, 256)       590080
_________________________________
_________________________________
block3_conv3 (Conv2D)        (None,
56, 56, 256)       590080
_________________________________
_________________________________
block3_pool (MaxPooling2D)   (None,
28, 28, 256)       0
_________________________________
_________________________________
block4_conv1 (Conv2D)        (None,
28, 28, 512)       1180160
_________________________________
_________________________________
block4_conv2 (Conv2D)        (None,
28, 28, 512)       2359808
_________________________________
_________________________________
block4_conv3 (Conv2D)        (None,
28, 28, 512)       2359808
_________________________________
_________________________________
block4_pool (MaxPooling2D)   (None,
14, 14, 512)       0
_________________________________
_________________________________
block5_conv1 (Conv2D)        (None,
14, 14, 512)       2359808
_________________________________
_________________________________
block5_conv2 (Conv2D)        (None,
14, 14, 512)       2359808
_________________________________
_________________________________
block5_conv3 (Conv2D)        (None,
14, 14, 512)       2359808
_________________________________
_________________________________
block5_pool (MaxPooling2D)   (None,
7, 7, 512)         0
_________________________________
_________________________________
flatten (Flatten)            (None,
25088)             0
_________________________________
_________________________________
fc1 (Dense)                  (None,
4096)              102764544
_________________________________
_________________________________
fc2 (Dense)                  (None,
4096)              16781312
=================================
=============================
Total params: 134,260,544
Trainable params: 134,260,544
Non-trainable params: 0
_________________________________
_________________________________
None
```

Extract features from image:

```
features = {}
directory = os.path.join(BASE_DIR, '
Images')

for img_name in tqdm(os.listdir(dire
ctory)):
    # load the image from file
    img_path = directory + '/' + img
_name
    image = load_img(img_path, targe
t_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.shap
e[2]))
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, v
erbose=0)
    # get image ID
    image_id = img_name.split('.')[0
]
    # store feature
    features[image_id] = feature
```

Output:

```
  0%|          | 0/8091 [00:00<?, ?i
t/s]
2022-09-04 14:00:41.595003: I tensor
flow/compiler/mlir/mlir_graph_optimi
zation_pass.cc:185] None of the MLIR
Optimization Passes are enabled (reg
istered 2)
2022-09-04 14:00:42.612931: I tensor
flow/stream_executor/cuda/cuda_dnn.c
c:369] Loaded cuDNN version 8005
```

Store features in pickle:

```
pickle.dump(features, open(os.path.j
oin(WORKING_DIR, 'features.pkl'), 'w
b'))
```

Load Features from pickle:

```
with open(os.path.join(WORKING_DIR,
'features.pkl'), 'rb') as f:
    features = pickle.load(f)
```

3. Load the Captions Data.

```
with open(os.path.join(BASE_DIR, 'ca
ptions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
```

Create Mapping of image to caption:

```
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], t
okens[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
)
```

Output:

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

```
len(mapping)
```

Output:

```
8091
```

4. Pre-process Text Data.

```
def clean(mapping):
    for key, captions in mapping.ite
ms():
        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.replac
e('[^A-Za-z]', '')
            # delete additional spac
es
            caption = caption.replac
e('\s+', ' ')
            # add start and end tags
to the caption
            caption = 'startseq ' +
" ".join([word for word in caption.s
plit() if len(word)>1]) + ' endseq'
            captions[i] = caption
```

Before pre-process of text:

```
mapping['1000268201_693b08cb0e']
```

Output:

```
['A child in a pink dress is climbin
g up a set of stairs in an entry way
.',
 'A girl going into a wooden buildin
g .',
 'A little girl climbing into a wood
en playhouse .',
 'A little girl climbing the stairs
to her playhouse .',
 'A little girl in a pink dress goin
g into a wooden cabin .']
```

Pre-process the text:

```
clean(mapping)
```

After pre-process of text:

```
mapping['1000268201_693b08cb0e']
```

Output:

```
['startseq child in pink dress is cl
imbing up set of stairs in an entry
way endseq',
 'startseq girl going into wooden bu
ilding 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']
```

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

Output:

```
40455
```

```
all_captions[:10]
```

Output:

```
['startseq child in pink dress is cl
imbing up set of stairs in an entry
way endseq',
 'startseq girl going into wooden bu
ilding 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 r
oad endseq',
 'startseq black dog and white dog w
ith brown spots are staring at each
other in the street endseq',
 'startseq two dogs of different bre
eds looking at each other on the roa
d endseq',
 'startseq two dogs on pavement movi
ng toward each other endseq']
```

5. Train Test Split.

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

Create data generator to get data in batch (avoids session crash):

```
def data_generator(data_keys, mappin
g, 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 sequenc
e
                seq = tokenizer.text
s_to_sequences([caption])[0]
                # split the sequence
into X, y pairs
                for i in range(1, le
n(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
```

6. Model Creation.

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)
```

Decode Model:

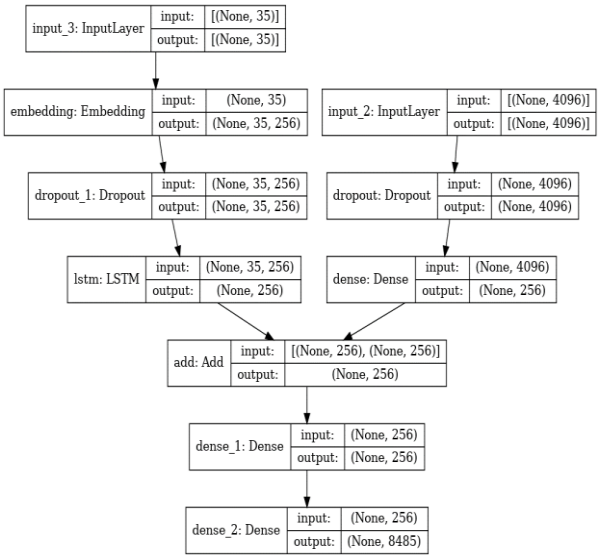
```
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation='relu')(decoder1)
outputs = Dense(vocab_size, activation='softmax')(decoder2)

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

Plot the model:

```
plot_model(model, show_shapes=True)
```

Output:



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)
```

Output:

```
227/227 [==============================] - 67s 280ms/step - loss: 5.2046
227/227 [==============================] - 62s 273ms/step - loss: 3.9878
227/227 [==============================] - 62s 274ms/step - loss: 3.5616
```

```
227/227 [==============================] - 62s 272ms/step - loss: 3.2967
227/227 [==============================] - 64s 280ms/step - loss: 3.1033
227/227 [==============================] - 64s 281ms/step - loss: 2.9574
227/227 [==============================] - 60s 264ms/step - loss: 2.8515
227/227 [==============================] - 65s 287ms/step - loss: 2.7627
227/227 [==============================] - 64s 281ms/step - loss: 2.6863
227/227 [==============================] - 63s 279ms/step - loss: 2.6168
227/227 [==============================] - 64s 279ms/step - loss: 2.5521
227/227 [==============================] - 65s 288ms/step - loss: 2.4960
227/227 [==============================] - 63s 276ms/step - loss: 2.4402
227/227 [==============================] - 62s 272ms/step - loss: 2.3939
227/227 [==============================] - 64s 280ms/step - loss: 2.3509
227/227 [==============================] - 65s 285ms/step - loss: 2.3123
227/227 [==============================] - 63s 279ms/step - loss: 2.2777
227/227 [==============================] - 63s 277ms/step - loss: 2.2467
227/227 [==============================] - 64s 283ms/step - loss: 2.2208
227/227 [==============================] - 63s 277ms/step - loss: 2.1964
```

Save the model:

```
model.save(WORKING_DIR+'/best_model.h5')
```

7. Generate Captions for the Image.

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

Generate Caption for the 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
```

```
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)))
```

Output:

```
  0%|          | 0/810 [00:00<?, ?it/s]
BLEU-1: 0.535628
BLEU-2: 0.308823
```

8. Visualize the Results.

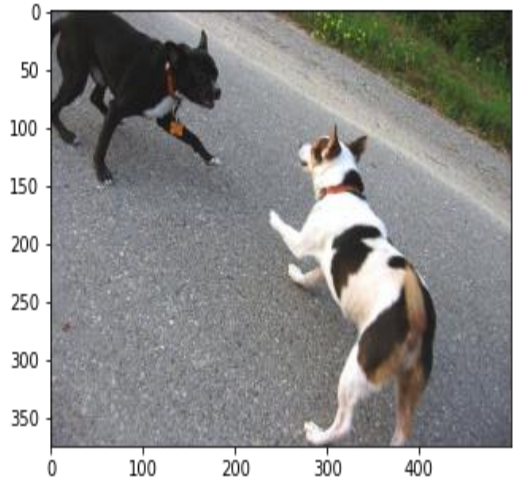
```
from PIL import Image
import matplotlib.pyplot as plt
def generate_caption(image_name):
    # load the image
    # image_name = "1001773457_577c3a7d70.jpg"
    image_id = image_name.split('.')[0]
    img_path = os.path.join(BASE_DIR, "Images", image_name)
    image = Image.open(img_path)
    captions = 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)
```

Generate the caption for an image:

```
generate_caption("1001773457_577c3a7d70.jpg")
```

Output:

```
---------------------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 on the sidewalk endseq
```




```
generate_caption("1002674143_1b742ab4b8.jpg")
```

Output:

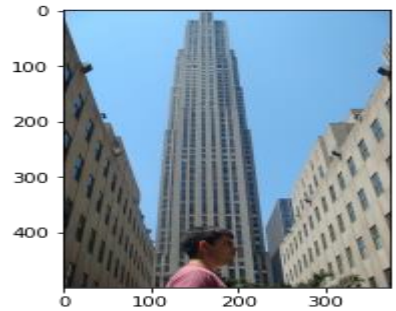
```
---------------------Actual---------------------
startseq little girl covered in paint sits in front of painted rainbow with her hands in bowl endseq
startseq little girl is sitting in front of large painted rainbow endseq
startseq small girl in the grass plays with fingerpaints in front of white 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 red dress an
d red dress is sitting on the grass
with fingerpaints in the background
endseq
```

```
generate_caption("1032460886_4a598ed
535.jpg")
```

Output:

```
---------------------Actual---------
------------
startseq man is standing in front of
skyscraper endseq
startseq man stands in front of skys
craper endseq
startseq man stands in front of very
tall building endseq
startseq behind the man in red shirt
stands large skyscraper endseq
startseq there is skyscraper in the
distance with man walking in front o
f the camera endseq
--------------------Predicted-------
-------------
startseq man in black jacket and whi
te vest is walking on sidewalk endse
q
```



7.CONCLUSION

In this advanced Python project, an image caption generator has been developed using a CNN model. Some key aspects about our project to note are that our model depends on the data, so, it cannot predict the words that are out of its vocabulary. A dataset consisting of 8000 images is used here. But for production-level models i.e., higher accuracy models, we need to train the model on larger than 100,000 images datasets so that better accuracy models can be developed.

8.REFERENCES

- https://www.kaggle.com/
- https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network
- https://www.analyticsvidhya.com/blog/2021/12/step-by-step-guide-to-build-image-caption-generator-using-deep-learning/
- https://www.geeksforgeeks.org/image-caption-

generator-using-deep-learning-on-flickr8k-dataset/
- https://intellipaat.com/blog/what-is-lstm/

9.FUTURE WORK

In the future, we would like to explore methods to generate multiple sentences with different content. One possible way is to combine interesting region detection and image captioning. The VSA method (Karpathy and Fei-Fei,) gives a direction of our future work. Taking Fig. 2 as an example, we hope the output will be a short paragraph: ”Jack has a wonderful breakfast in a Sunday morning. He is sitting at a table with a bouquet of red flowers. With his new iPad air on the left, he enjoys a plate of fruits and a cup of coffee.” The short paragraph naturally describes the image content in a story-telling fashion which is more attractive to the human beings.

10.LITERATURE SURVEY

Title	Year	Objectives	Techniques	Dataset
Camera2Caption: A Real-Time Image Caption Generator	2017	Simplistic encoder and decoder to produce real-time captions for the image	CNN RNN LSTM	MSCOCO (2014) 82780 images
Building A Voice Based Image Caption Generator with Deep Learning	2021	Voice based captions for an input image	CNN LSTM	Flickr_8k
Explainable Image Caption Generator Using Attention and Bayesian Inference	2018	Explain the caption generated without using encoder and decoder model	CNN LSTM RNN	Flickr_8k Flickr_30k MSCOCO