INSTITUTE FOR ADVANCED
COMPUTING AND
SOFTWARE
DEVELOPMENT
AKURDI, PUNE

Documentation On

“Image Caption Generation using Deep Learning”

PG-DBDA SEP 2022

Submitted By:

Group No: 03

Ajit Yadav

Roll no. 229302

Mr. Rohit Puranik
Centre Coordinator

Dr. Shantanu Pathak
Project Guide

Contents:

1. INTRODUCTION

1.1Problem Statement:

To develop a system for users, that can automatically generate a textual description of an image. This involves teaching a machine learning model to understand the visual content of an image and use that understanding to produce a coherent sentence that describes the objects, people, and actions depicted in the image.

1.2 Abstract:

Image captioning is an interesting and challenging task with applications in diverse domains such as image retrieval, organizing and locating images of users' interest etc. It has huge potential for replacing manual caption generation for images and is especially suitable for large scale image data. Recently, deep neural network based methods have achieved great success in the field of computer vision, machine translation and language generation. In this project, we propose an encoder-decoder based model that is capable of generating grammatically correct captions for images. This model makes use of VGG16 (Pre-Trained Model) as encoder and LSTM as decoder. To ensure the complete ground truth accuracy, the model is trained on the labelled Flickr8k dataset.

1.3 Use Case:

An image caption generator has several use cases across various fields.

Social Media: Social media platforms such as Facebook, Instagram, and Twitter can use image caption generators to help users with visual impairments access the content shared on the platform. The image caption generator can automatically generate a description of the images that are shared on the platform.

E-commerce: E-commerce websites can use image caption generators to improve the search ability and accessibility of their product catalogs. The generator can automatically generate descriptions of the products and help users find what they are looking for.

Healthcare: Medical imaging such as X-rays, CT scans, and MRIs can be described using an image caption generator. This can help doctors and healthcare professionals better understand and diagnose medical conditions.

Entertainment: Image caption generators can be used to enhance the user experience of online media platforms such as YouTube and Netflix. The generator can automatically generate captions for the videos and make them more accessible to a wider audience.

Education: Image caption generators can be used in educational settings to help students with visual impairments access visual content such as charts, graphs, and diagrams. The generator can automatically generate descriptions of the visual content and help students understand the material.

Autonomous Vehicles: Image caption generators can be used in autonomous vehicles to help them better understand and navigate their surroundings. The generator can automatically generate descriptions of the objects and scenes that the vehicle encounters and help it make better decisions.

2. Overall Description

2.1 Workflow of Project:

The diagram below shows the workflow of this project.

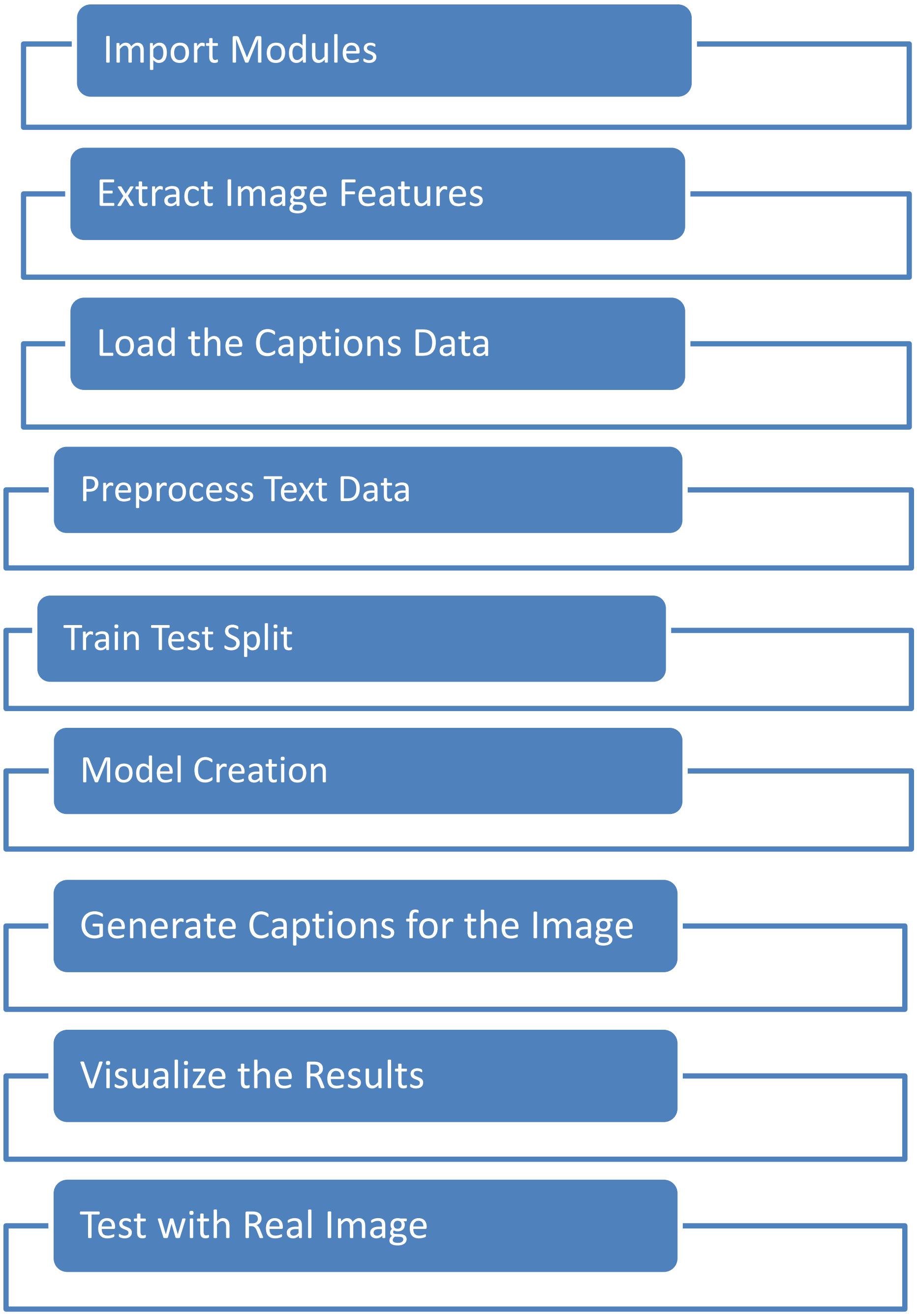


Figure 2.1 Workflow Diagram

2.2 About Datasets:

The Flickr8K dataset is a publicly available dataset consisting of 8,091 images that have been annotated with textual descriptions. The dataset was created by gathering images from the photo-sharing website Flickr and then annotating each image with five different captions. The captions were written by different people to capture a range of perspectives and interpretations of the same image. The dataset contains a wide variety of images, including landscapes, animals, people, and objects. The images have been resized to a resolution of 500 pixels on the longest side, and each image is accompanied by five captions in plain text format.

The Flickr8K dataset has been widely used for research in computer vision and natural language processing, particularly for tasks such as image captioning and multimodal machine learning. Text file contains 5 different caption for each image.

```
image,caption
1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg,A girl going into a wooden building .
1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg,A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg,A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg,A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg,A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg,A black dog and a white dog with brown spots are staring at each other in the street .
1001773457_577c3a7d70.jpg,Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg,Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg,A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .
1002674143_1b742ab4b8.jpg,A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg,A small girl in the grass plays with fingerpaints in front of a white canvas with a rainbow on it .
1002674143_1b742ab4b8.jpg,There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg,Young girl with pigtails painting outside in the grass .
```

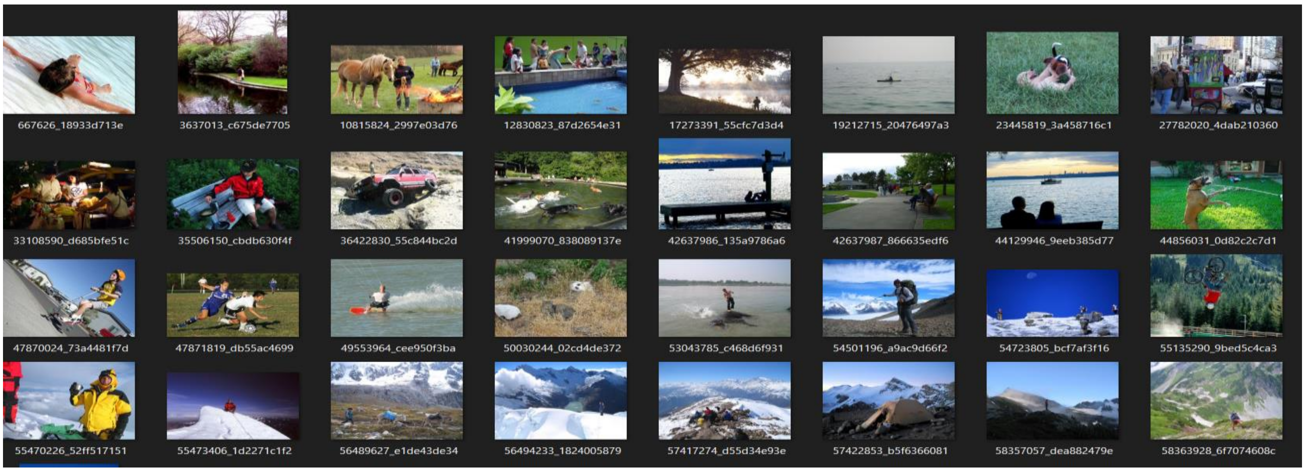


Figure 2.2 About Dataset

2.2.1 VGG16 Model:

The VGG16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford in 2014. It achieved state-of-the-art performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014, and has since been widely used as a pre-trained model for various computer vision tasks.

The VGG16 architecture consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. It has a fixed input size of 224x224 RGB images and produces a 1000-dimensional output vector that represents the probabilities of the input image belonging to each of the 1000 classes in the ImageNet dataset.

The key innovation of the VGG16 architecture is the use of small 3x3 convolutional filters throughout the network, which allows the model to learn more complex and non-linear features while keeping the number of parameters manageable. VGG16 also uses max pooling and dropout layers to prevent overfitting.

Because of its strong performance on the ILSVRC, the pre-trained VGG16 model has been used as a feature extractor for various computer vision tasks, such as image classification, object detection, and image segmentation. The model is available in the TensorFlow and Keras libraries, and can be easily fine-tuned on a new dataset for a specific task.

VGG16 Architecture:

The 16 in VGG16 refers to 16 layers that have weights. In VGG16 there are thirteen convolutional layers, five Max Pooling layers, and three Dense layers which sum up to 21 layers but it has only sixteen weight layers i.e., learnable parameters layer. VGG16 takes input tensor size as 224, 244 with 3 RGB channel.

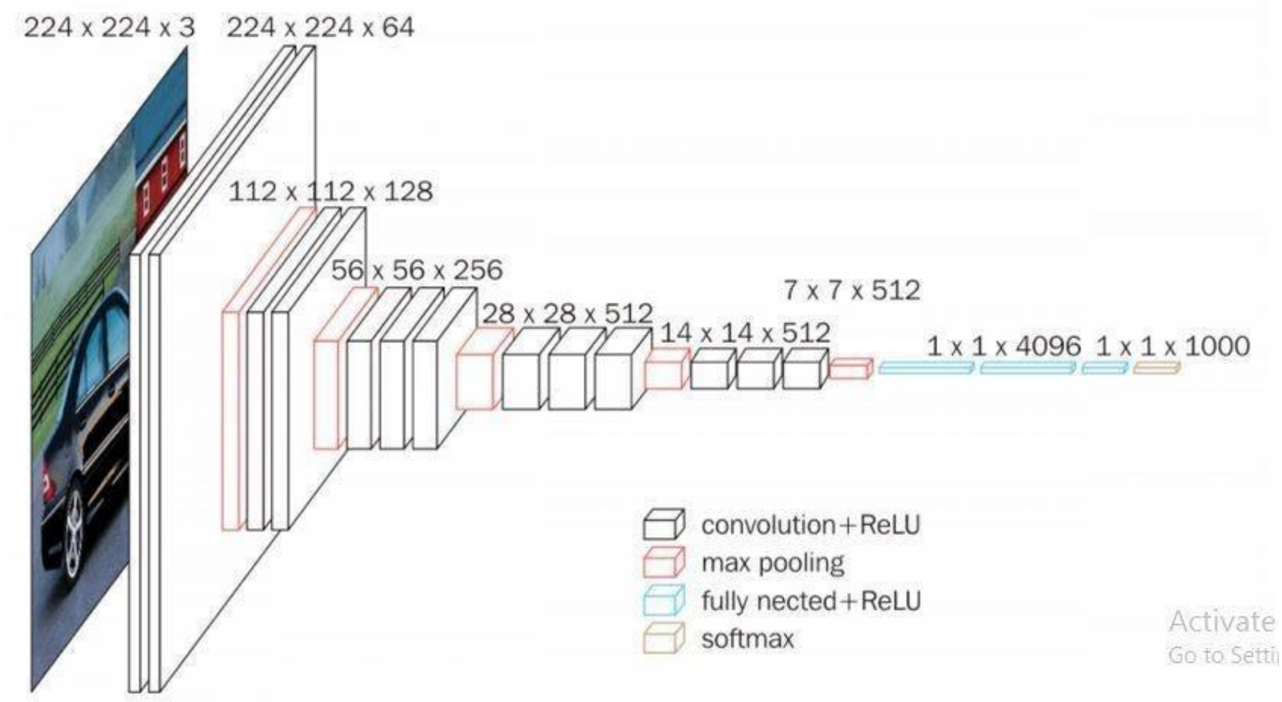


Figure 2.3 Architecture of VGG16 Model

Most unique thing about VGG16 is that instead of having a large number of hyper-parameters they focused on having convolution layers of 3x3 filter with stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2.

The convolution and max pool layers are consistently arranged throughout the whole architecture. Conv-1 Layer has 64 number of filters, Conv-2 has 128 filters, Conv-3 has 256 filters, Conv 4 and Conv 5 has 512 filters.

Three Fully-Connected (FC) layers follow a stack of convolutional layers: the first two have 4096 channels each, the third performs 1000-way ILSVRC classification and thus contains 1000 channels (one for each class). The final layer is the soft-max layer.

2.3.1 Import Modules:

Import Modules

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

from tensorflow.keras.applications.vgg16 import VGG16, preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.utils import to_categorical, plot_model
from tensorflow.keras.layers import Input, Dense, LSTM, Embedding, Dropout, add
```

OS Module:

OS provides a way of using operating system dependent functionality like reading or writing to the file system, working with environment variables, and executing system commands.

`os.path.join(path1, path2): Joins two paths together to form a complete path`

Pickle Module:

The pickle module is used for object serialization and deserialization. It allows you to convert a Python object hierarchy into a byte stream that can be stored or transmitted, and then reconstruct the object hierarchy from the byte stream. This process is known as "pickling" and "unpickling".

The pickle module is commonly used for tasks such as:

Saving and loading machine learning models, so that they can be reused later Caching expensive computations, so that they can be reused instead of recalculated every time Saving and loading program state, so that a program can be resumed from where it left off after a crash or restart. The pickle module provides two main functions: **dump and load.**

```
with open('/content/drive/MyDrive/Colab Notebooks/Project/working/features.pk
l', 'rb') as f:
    features = pickle.load(f)


pickle.dump(features, open(os.path.join(WORKING_DIR, '/content/drive/MyDrive/
Colab Notebooks/Project/working/features_ver3.pkl'), 'wb'))
```

Numpy module:

NumPy is a powerful library for numerical computing in Python and is widely used for tasks such as scientific computing, data analysis, and machine learning.

Tensorflow.keras module:

TensorFlow is a popular open-source library for numerical computation and machine learning, which provides a flexible platform for building and deploying machine learning models. TensorFlow includes a wide range of tools and libraries for building and training deep learning models, including the Keras API. Keras is a high-level deep learning API written in Python, which provides a user-friendly interface for building, training, and evaluating deep learning models. Once VGG16 and the preprocess_input function are imported, you can use them to classify images or fine-tune the pre-trained model on a new dataset.

img_to_array, load_img :

img_to_array is used to import two functions from the Keras preprocessing module in TensorFlow: load_img and img_to_array.

load_img is a function that loads an image file from disk and returns a PIL (Python Imaging Library) image object. It takes two arguments: path is the path to the image file, and target_size is a tuple specifying the size to which the image should be resized. If target_size is not specified, the function returns the original size of the image.

img_to_array is a function that converts a PIL image object into a NumPy array. It takes a single argument: img is the PIL image object to be converted.

These two functions are often used together to load and pre-process image data for deep learning models.

2.3.2 Extract Image Features:

In image caption generation, one common approach for extracting image features is to use a pre-trained convolutional neural network (CNN) such as VGG16. These CNNs are trained on large datasets (such as ImageNet) to classify images into different categories.

To extract image features, the pre-trained CNN is typically used as a feature extractor. The input image is passed through the CNN, and the output of one of the intermediate layers (before the fully connected layers) is extracted as the image feature representation. This feature representation can then be used as input to a separate natural language processing (NLP) model, such as a recurrent neural network (RNN), to generate captions

Extract Image Features

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

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467096/553467096 [==============================] - 28s 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
```

```python
# extract features from image
features = {}
directory = os.path.join(BASE_DIR, 'Images')

for img_name in tqdm(os.listdir(directory)):
    # load the image from file
    img_path = directory + '/' + img_name
    image = load_img(img_path, target_size=(224, 224))
    # convert image pixels to numpy array
    image = img_to_array(image)
    # reshape data for model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2])) # rgb image
    # preprocess image for vgg
    image = preprocess_input(image)
    # extract features
    feature = model.predict(image, verbose=0)
    # get image ID
    image_id = img_name.split('.')[0]
    # store feature
    features[image_id] = feature
```

2.3.3 Load the Captions Data

We reads the contents of a file named 'captions.txt' located in the directory specified by the variable BASE_DIR. It then skips the first line of the file using the next(f) function call, and reads the rest of the file into a string variable named captions_doc. And we are Mapping each captions to it's image ID.

Load the Captions Data

```python
[10]
with open(os.path.join(BASE_DIR, 'captions.txt'), 'r') as f:
    next(f)
    captions_doc = f.read()
```

```python
captions_doc
```

```
'1000268201_693b08cb0e.jpg,A child in a pink dress is climbing up a set of stairs in an entry way .\n1000268201_693b08cb0e.jpg,A girl going int
o a wooden building .\n1000268201_693b08cb0e.jpg,A little girl climbing into a wooden playhouse .\n1000268201_693b08cb0e.jpg,A little girl clim
bing the stairs to her playhouse .\n1000268201_693b08cb0e.jpg,A little girl in a pink dress going into a wooden cabin .\n1001773457_577c3a7d70.
jpg,A black dog and a spotted dog are fighting\n1001773457_577c3a7d70.jpg,A black dog and a tri-colored dog playing with each other on the road
.\n1001773457_577c3a7d70.jpg,A black dog and a white dog with brown spots are staring at each other in the street .\n1001773457_577c3a7d70.jpg,
Two dogs of different breeds looking at each other on the road .\n1001773457_577c3a7d70.jpg,Two dogs on pavement moving toward each other .\n10
02674143_1b742ab4b8.jpg,A little girl covered in paint sits in front of a painted rainbow with her hands in a bowl .\n1002674143_1b742ab4b8.jp...
'
```

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

```
100% 40456/40456 [00:00<00:00, 259385.00it/s]
```

2.3.4 Preprocess Text Data:

We defined a function named clean that takes a single argument mapping, which is assumed to be a dictionary where each key corresponds to an image and the associated value is a list of captions describing that image. The function cleans and preprocesses each caption in the mapping by performing the following steps:

1. Converts each caption to lowercase using the .lower() method.

2. Removes any non-alphabetic characters (e.g. digits, special characters) from the caption using the .replace() method with a regular expression pattern [^\w].

3. Removes any extra whitespace characters (e.g. multiple spaces in a row) from the caption using the .replace() method with the pattern \s+.

4. Adds special "start" and "end" tags to the caption to indicate the beginning and end of the caption using the str.join() method with a list comprehension.

The cleaned captions are stored back in the original mapping dictionary, overwriting the original values for each key.

Preprocess Text Data

```
[13]
def clean(mapping):
    for key, captions in mapping.items():
        for i in range(len(captions)):
            # take one caption at a time
            caption = captions[i]
            # preprocessing steps
            # convert to lowercase
            caption = caption.lower()
            # delete digits, special chars, etc.,
            caption = caption.replace('[^A-Za-z]', '')
            # delete additional spaces
            caption = caption.replace('\s+', ' ')
            # add start and end tags to the caption
            caption = 'startseq ' + " ".join([word for word in caption.split() if len(word)>1]) + ' endseq'
            captions[i] = caption
```

We have displayed before and after Preprocessing of text data:

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

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

```
[15]
# preprocess the text
clean(mapping)
```

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

```
['startseq child in pink dress is climbing up set of stairs in an entry way endseq',
 'startseq girl going into wooden building endseq',
 'startseq little girl climbing into wooden playhouse endseq',
 'startseq little girl climbing the stairs to her playhouse endseq',
 'startseq little girl in pink dress going into wooden cabin endseq']
```

Tokenize the text data:

We are creating Keras Tokenizer class by calling the constructor with no arguments. This creates a new Tokenizer object that can be used to convert sequences of text into sequences of integers suitable for use as input to a neural network.

The fit_on_texts method is then called on the Tokenizer object, passing in a list of all the captions in the dataset as its argument. This method updates the internal state of the Tokenizer object to create a vocabulary of all the unique words in the text, and assigns a unique integer index to each word.

Finally, the vocab_size variable is set to the size of the vocabulary created by the Tokenizer, which is the total number of unique words plus one (to account for the special "out of vocabulary" token).

To prepare the vocabulary for the captions in the dataset, which will be used to represent each caption as a sequence of integers for input into a neural network. The Tokenizer object provides a convenient way to perform this transformation, and the resulting vocabulary size is used to set the size of the embedding layer in the neural network.

```
[ ]
# tokenize the text
tokenizer = Tokenizer()
tokenizer.fit_on_texts(all_captions) #fit_on_texts is used to produces the one-hot encoding for the original set of texts.
vocab_size = len(tokenizer.word_index) + 1

# Because Tokenizer.word_index is a python dictionary that contains token keys (string) and token ID values (integer),
# and where the first token ID is 1 (not zero) and where the token IDs are assigned incrementally. Therefore,
# the greatest token ID in word_index is len(word_index).
# Therefore, we need vocabulary of size len(word_index) + 1 to be able to index up to the greatest token ID.
```

```
[ ]
tokenizer.word_index
```

```
'him': 148,
'it': 149,
"'s": 150,
'road': 151,
'area': 152,
'that': 153,
'basketball': 154,
'tan': 155,
'back': 156,
'trick': 157,
'race': 158,
'swing': 159,
'head': 160,
'shorts': 161,
```

2.3.5 Train Test Split:

We are splitting the image ids into training and testing sets for use in a machine learning model. The split is based on a percentage of the total number of image ids, with 85% of the image ids assigned to the training set and 15% assigned to the testing set.

Train Test Split

```python
image_ids = list(mapping.keys())
split = int(len(image_ids) * 0.85)
train = image_ids[:split]
test = image_ids[split:]
```

data_generator():

We have created data generator function to get data in batch (avoids session crash).

```python
# create data generator to get data in batch (avoids session crash)
def data_generator(data_keys, mapping, features, tokenizer, max_length, vocab_size, batch_size):
    # loop over images
    X1, X2, y = list(), list(), list()
    n = 0
    while 1:
        for key in data_keys:
            n += 1
            captions = mapping[key]
            # process each caption
            for caption in captions:
                # encode the sequence
                seq = tokenizer.texts_to_sequences([caption])[0]
                # split the sequence into X, y pairs
                for i in range(1, len(seq)):
                    # split into input and output pairs
                    in_seq, out_seq = seq[:i], seq[i]
                    # pad input sequence
                    in_seq = pad_sequences([in_seq], maxlen=max_length)[0]
                    # encode output sequence
                    out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

                    # store the sequences
                    X1.append(features[key][0])
                    X2.append(in_seq)
                    y.append(out_seq)
            if n == batch_size:
                X1, X2, y = np.array(X1), np.array(X2), np.array(y)
                yield [X1, X2], y
                X1, X2, y = list(), list(), list()
                n = 0
```

2.3.6 Model Creation:

We have defined an image captioning model with an encoder-decoder architecture. The encoder takes in the image features as input and passes them through a series of layers, including dropout and a fully connected layer, to extract a feature vector. The decoder takes in the captions as input and passes them through an embedding layer and an LSTM layer to extract a sequence of feature vectors. The decoder then combines the image feature vector with the sequence of feature vectors from the LSTM layer using an element-wise addition and passes the resulting vector through a series of fully connected layers to generate the final output. The model is compiled using the Adam optimizer and categorical cross-entropy loss function.

Model Creation

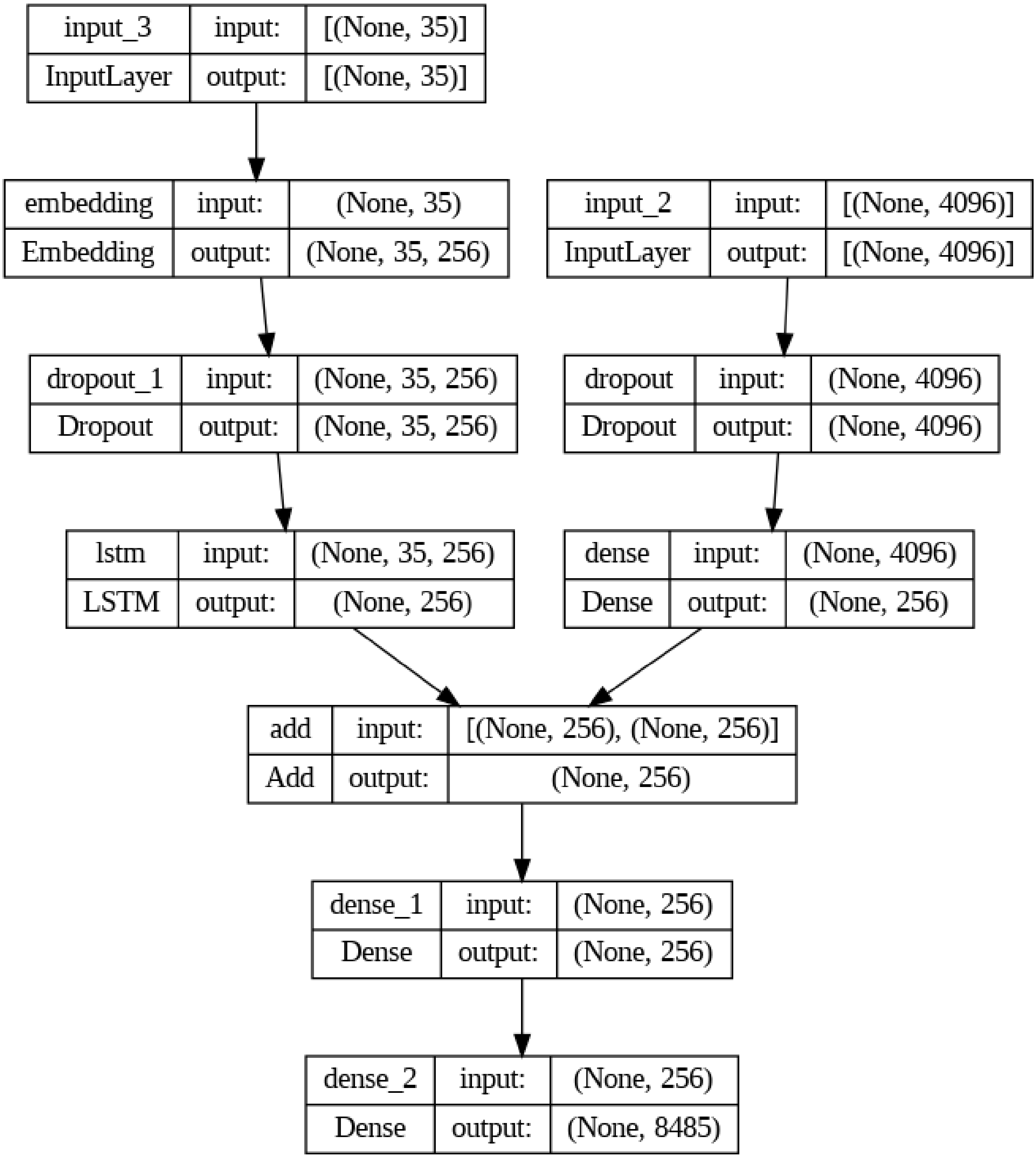
```
# encoder model
# image feature layers
inputs1 = Input(shape=(4096,))
fe1 = Dropout(0.30)(inputs1)
fe2 = Dense(256, activation='relu')(fe1)
# sequence feature layers
inputs2 = Input(shape=(max_length,))
se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
se2 = Dropout(0.30)(se1)
se3 = LSTM(256)(se2)

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

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

# Unlike maintaining a single learning rate through training in SGD, Adam optimizer updates the learning rate for each network weight indivi
# Categorical cross entropy loss function is used to compute the quantity that the the model should seek to minimize during training
model.compile(loss='categorical_crossentropy', optimizer='adam')

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


input_3
InputLayer
input:
output:
[(None, 35)]
[(None, 35)]
embedding
Embedding
input:
output:
(None, 35)
(None, 35, 256)
input_2
InputLayer
input:
output:
[(None, 4096)]
[(None, 4096)]
dropout_1
Dropout
input:
output:
(None, 35, 256)
(None, 35, 256)
dropout
Dropout
input:
output:
(None, 4096)
(None, 4096)
lstm
LSTM
input:
output:
(None, 35, 256)
(None, 256)
dense
Dense
input:
output:
(None, 4096)
(None, 256)
add
Add
input:
output:
[(None, 256), (None, 256)]
(None, 256)
dense_1
Dense
input:
output:
(None, 256)
(None, 256)
dense_2
Dense
input:
output:
(None, 256)
(None, 8485)

```
# train the model
epochs = 25
batch_size = 30
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)
```

```
229/229 [==============================] - 76s 331ms/step - loss: 3.7133
229/229 [==============================] - 67s 292ms/step - loss: 3.4110
229/229 [==============================] - 69s 299ms/step - loss: 3.1835
229/229 [==============================] - 67s 293ms/step - loss: 3.0095
229/229 [==============================] - 68s 298ms/step - loss: 2.8785
229/229 [==============================] - 69s 301ms/step - loss: 2.7816
229/229 [==============================] - 69s 301ms/step - loss: 2.6947
229/229 [==============================] - 68s 298ms/step - loss: 2.6118
229/229 [==============================] - 67s 294ms/step - loss: 2.5419
229/229 [==============================] - 67s 293ms/step - loss: 2.4759
229/229 [==============================] - 68s 296ms/step - loss: 2.4119
229/229 [==============================] - 71s 309ms/step - loss: 2.3539
229/229 [==============================] - 69s 302ms/step - loss: 2.3063
229/229 [==============================] - 69s 302ms/step - loss: 2.2603
229/229 [==============================] - 67s 291ms/step - loss: 2.2233
229/229 [==============================] - 68s 297ms/step - loss: 2.1877
229/229 [==============================] - 67s 294ms/step - loss: 2.1497
229/229 [==============================] - 71s 308ms/step - loss: 2.1164
229/229 [==============================] - 70s 306ms/step - loss: 2.0884
```

We defined the maximum number of epochs to 25 and the batch size to 30. We calculated the number of steps needed in each epoch for training and validation data.

And we use a data generator to generate arrays of these sequence of the size of the batch progressively. We used this approach to avoid reaching RAM and GPU limits. Otherwise, we weren’t able to train the model with the RAM available in Google Collaboratory. The data generator receives the training data shuffled and works with image ids to take the images that correspond to the captions.

Rnn Model1: It has two inputs. In the text submodel it has an Embedding, Dropout and LSTM layers. In the image submodel, it has a Dropout and a Dense layer. Then it adds these two submodels and finally there are two Dense layers, the last one having vocabulary size with softmax to train the model with the RAM available in Google Colaboratoty. The data generator receives the training data shuffled and works with image ids to take the images that correspond to the captions.

2.3.7 Generate Captions for the Image:

The **idx_to_word** function takes an integer representing a word index and a tokenizer object as inputs, and returns the corresponding word. It iterates over the word_index dictionary of the tokenizer, which maps words to integer indices, and returns the word that has the given integer index. If the integer index is not found in the word_index, it returns None. This function can be used to convert the output of the model, which is a sequence of integer indices representing the predicted words, back into a sequence of words.

The function **predict_caption()** takes as input the trained model, an image, a tokenizer, and the maximum length of the sequence, and returns the predicted caption for the image. The function starts by adding a special start tag startseq to the input sequence. It then iterates over the maximum length of the sequence, encoding the input sequence, padding it, and using the model to predict the next word. The predicted word is then converted to its corresponding string form using the idx_to_word() function. If the predicted word is not found, the loop is stopped. Otherwise, the predicted word is appended to the input sequence for generating the next word. If the predicted word is the end tag endseq, the loop is stopped.

Generate Captions for the Image

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

```python
# generate caption for an image
def predict_caption(model, image, tokenizer, max_length):
    # add start tag for generation process
    in_text = 'startseq'
    # iterate over the max length of sequence
    for i in range(max_length):
        # encode input sequence
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        # pad the sequence
        sequence = pad_sequences([sequence], max_length)
        # predict next word
        yhat = model.predict([image, sequence], verbose=0)
        # get index with high probability
        yhat = np.argmax(yhat)
        # convert index to word
        word = idx_to_word(yhat, tokenizer)
        # stop if word not found
        if word is None:
            break
        # append word as input for generating next word
        in_text += " " + word
        # stop if we reach end tag
        if word == 'endseq':
            break

    return in_text
```

The BLEU score is a metric used to evaluate the quality of generated text, such as machine translation or image captioning. It measures how similar the generated text is to a set of reference texts. The score ranges between 0 and 1, where a higher score indicates a better match with the reference texts.The BLEU score is computed using n-grams, which are sequences of n words. The score is a weighted average of the n-gram precision, which is the percentage of n-grams in the generated text that appear in the reference texts. The weights give more importance to longer n-grams.

In this code, we are computing the BLEU-1 and BLEU-2 scores using the corpus_bleu function from the nltk.translate.bleu_score module. The actual list contains the reference texts, which are the captions associated with the test images. The predicted list contains the generated texts, which are the captions predicted by the model for the test images.

The BLEU-1 score is computed using only unigrams (n=1), while the BLEU-2 score is computed using both unigrams and bigrams (n=2). The weights used in the computation are (1,0,0,0) for BLEU-1 and (0.5,0.5,0,0) for BLEU-2, which give equal importance to unigrams and bigrams.

```
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)))
```

```
100% 1214/1214 [13:40<00:00, 1.62it/s]
BLEU-1: 0.534889
BLEU-2: 0.313522
```

2.3.8 Visualize the Results:

We are using generate_caption() function for caption generation , and we are displaying actual vs predicted caption .

▾ 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)
```

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

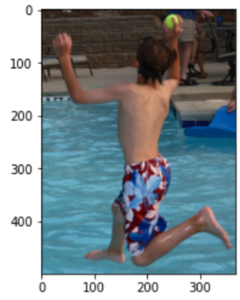
```
---------------------Actual---------------------
startseq black dog and spotted dog are fighting endseq
startseq black dog and tri-colored dog playing with each other on the road endseq
startseq black dog and white dog with brown spots are staring at each other in the street endseq
startseq two dogs of different breeds looking at each other on the road endseq
startseq two dogs on pavement moving toward each other endseq
--------------------Predicted--------------------
startseq black dog and tri colored dog are playing with each other on the grass endseq
```

```
1 generate_caption("161669933_3e7d8c7e2c.jpg")
```

```
---------------------Actual---------------------
startseq competitive motorcycle racer prepares to make left turn along paved road endseq
startseq man rides motorcycle with the number on it endseq
startseq motorcycle driver swerves to the left endseq
startseq motorcycle with number five on front is being ridden by rider wearing red helmet endseq
startseq professional motorcycle racer turning corner endseq
--------------------Predicted--------------------
startseq motorcycle rider rides his bike on track endseq
```

```
generate_caption("524036004_6747cf909b.jpg")
```

```
--------------------Actual--------------------
startseq "a boy in blue and red trunks with green ball in his hand is jumping into pool ." endseq
startseq boy is jumping into the water with ball in his hand endseq
startseq boy jumps into pool while holding ball endseq
startseq boy in floral swim trunks jumps into the pool endseq
startseq boy in red and blue flowered trunks jumps into pool holding green ball endseq
--------------------Predicted-------------------
startseq boy in bathing suit is jumping into pool while another boy watches endseq
```



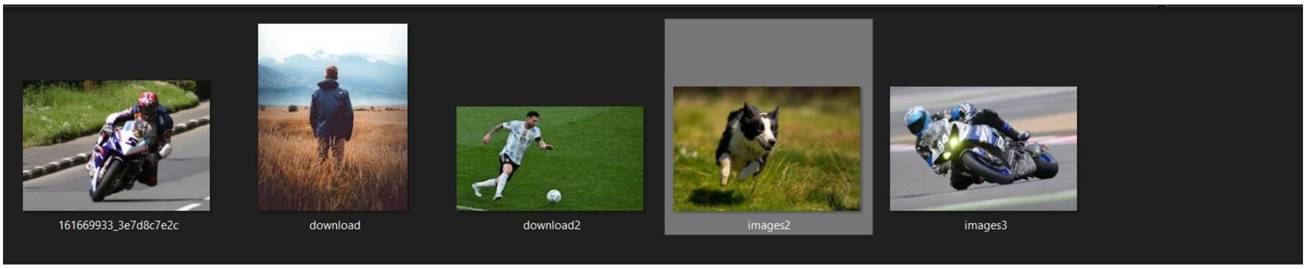
2.3.9 Test with Real Image:

Test with Real Image

```
vgg_model = VGG16()
# restructure the model
vgg_model = Model(inputs=vgg_model.inputs, outputs=vgg_model.layers[-2].output)
```

```
image_path = '/content/images2.jpeg'
# load image
image = load_img(image_path, target_size=(224, 224))
# convert image pixels to numpy array
image = img_to_array(image)
# reshape data for model
image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
# preprocess image for vgg
image = preprocess_input(image)
# extract features
feature = vgg_model.predict(image, verbose=0)
# predict from the trained model
predict_caption(model, feature, tokenizer, max_length)
```

```
'startseq black and white dog is running through the grass endseq'
```



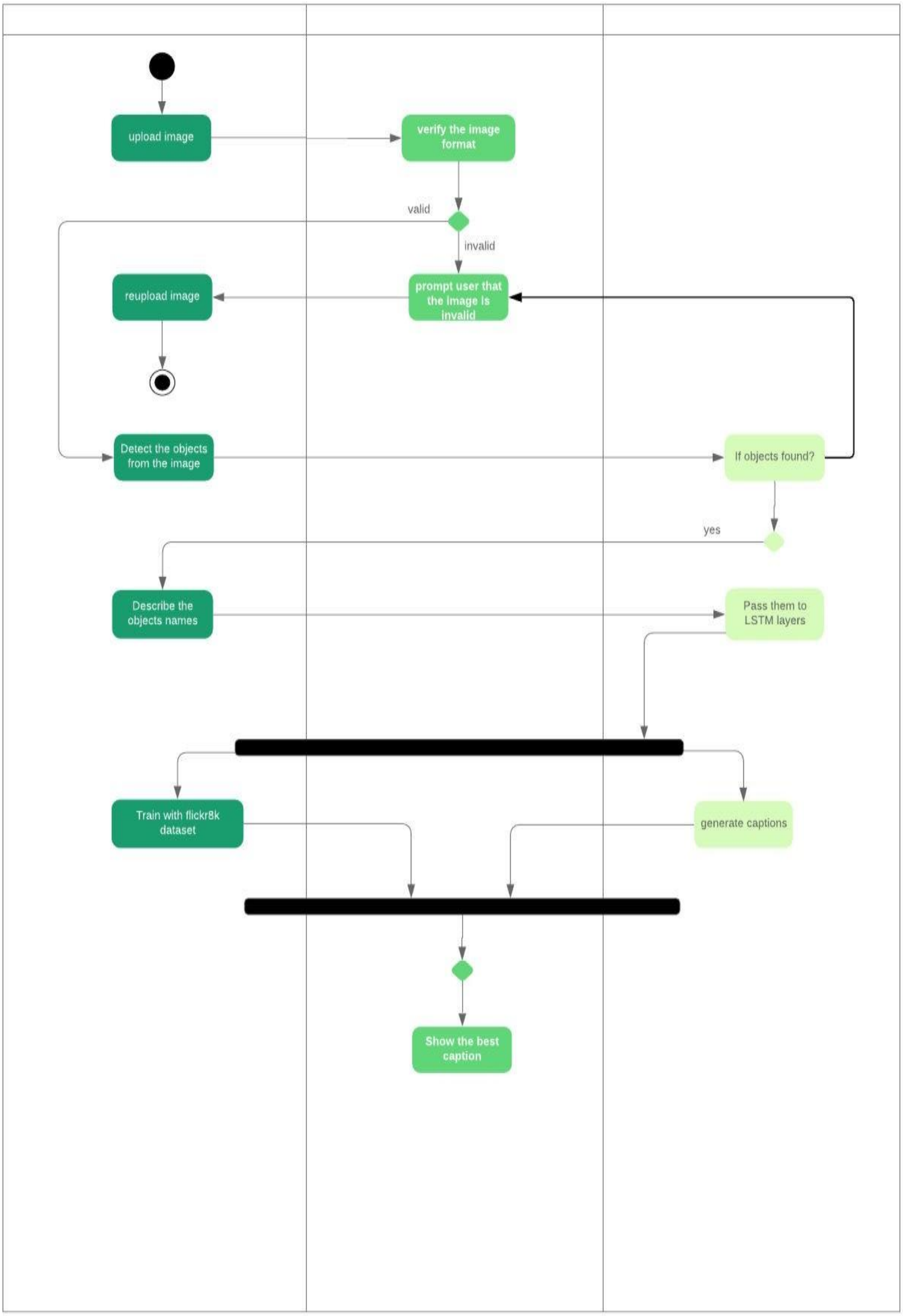


Fig : Flow of the project

Applications and Future scope of Image Caption Generator:

- It can be used to assist the blind using text-to-speech.
- It can be used to convert captions for images in social feed as well as messages to speech which will enhance social medial experience of users.
- It can also be used for educational purposes as young children can be assisted about recognition of objects and learning the English language.
- It is also helpful in field of robotics as environmental insights can be provided through natural language representation.
- It can also be used for image searches and indexing purposes on internet if images present on the internet have captions

Conclusion

In conclusion, an image caption generator is a powerful tool that uses deep learning algorithms to analyze an image and generate a descriptive and accurate caption for it. This technology has various applications, including improving accessibility for the visually impaired, enhancing social media engagement, and supporting image retrieval in large databases.

Image caption generators have been developed using various deep learning architectures, including Convolutional Neural Networks (CNNs) and Long Short Term Memory (LSTM), and have been trained on large datasets of images with corresponding captions.

While image caption generators have shown impressive results, there is still room for improvement in terms of the accuracy and diversity of generated captions. Researchers are continuing to work on developing more advanced models that can capture more nuanced and contextual information from images, as well as incorporating more knowledge about language and grammar.

Overall, image caption generators have the potential to revolutionize the way we interact with and understand visual content, and will likely continue to evolve and improve in the years to come.

References

https://www.kaggle.com/code/aswintechguy/image-caption-generator-tutorial-flickr-dataset

https://www.youtube.com/watch?v=fUSTbGrL1tc

https://www.kaggle.com/datasets/shadabhussain/flickr8k

https://machinelearningmastery.com/develop-a-deep-learning-caption-generation-model-in-python/

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