**Automatic Image Caption Generator for Visually Impaired**

**(Major Project-4th Year)**

**Mentor- Ms. Sandhya Avasthi**

Tanu Singh, Ashwani, Dheeraj Maurya

Department of Computer Science and Engineering

ABES Engineering College, Ghaziabad, Uttar Pradesh, 201009

**PROJECT CODE**

**ML code:**

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

BASE\_DIR = 'C:/Users/ashmy/OneDrive/Desktop/Image\_caption\_generator/Dataset'

WORKING\_DIR = 'C:/Users/ashmy/OneDrive/Desktop/Image\_caption\_generator'

# load vgg16 model

model = VGG16()

# restructure the model

model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

# summarize

print(model.summary())

features = {}

directory = os.path.join(BASE\_DIR, 'Images')

for img\_name in tqdm(os.listdir(directory)):

# load the image from file

img\_path = directory + '/' + img\_name

image = load\_img(img\_path, target\_size=(224, 224))

# convert image pixels to numpy array

image = img\_to\_array(image)

# reshape data for model

image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))

# preprocess image for vgg

image = preprocess\_input(image)

# extract features

feature = model.predict(image, verbose=0)

# get image ID

image\_id = img\_name.split('.')[0]

# store feature

features[image\_id] = feature

pickle.dump(features, open(os.path.join(WORKING\_DIR, 'features.pkl'), 'wb'))

with open(os.path.join(WORKING\_DIR, 'features.pkl'), 'rb') as f:

features = pickle.load(f)

with open(os.path.join(BASE\_DIR, 'captions.txt'), 'r') as f:

next(f)

captions\_doc = f.read()

# 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)

# Preprocess Text Data

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(r'\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

# before preprocess of text

mapping['1000268201\_693b08cb0e']

clean(mapping)

mapping['1000268201\_693b08cb0e']

all\_captions = []

for key in mapping:

for caption in mapping[key]:

all\_captions.append(caption)

len(all\_captions)

all\_captions[:10]

tokenizer = Tokenizer()

tokenizer.fit\_on\_texts(all\_captions)

vocab\_size = len(tokenizer.word\_index) + 1

vocab\_size

max\_length = max(len(caption.split()) for caption in all\_captions)

max\_length

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, 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 {"image": X1, "text": X2}, y

X1, X2, y = list(), list(), list()

n = 0

# encoder model

# image feature layers

inputs1 = Input(shape=(4096,), name="image")

fe1 = Dropout(0.4)(inputs1)

fe2 = Dense(256, activation='relu')(fe1)

# sequence feature layers

inputs2 = Input(shape=(max\_length,), name="text")

se1 = Embedding(vocab\_size, 256, mask\_zero=True)(inputs2)

se2 = Dropout(0.4)(se1)

se3 = LSTM(256)(se2)

# decoder model

decoder1 = add([fe2, se3])

decoder2 = Dense(256, activation='relu')(decoder1)

outputs = Dense(vocab\_size, activation='softmax')(decoder2)

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

model.compile(loss='categorical\_crossentropy', optimizer='adam')

# plot the model

#plot\_model(model, show\_shapes=True)

# train the model

epochs = 1

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)

# save the model

model.save(WORKING\_DIR+'/best\_model.h5')

def idx\_to\_word(integer, tokenizer):

for word, index in tokenizer.word\_index.items():

if index == integer:

return word

return None

# 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

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\_caption("44856031\_0d82c2c7d1.jpg")

generate\_caption("54501196\_a9ac9d66f2.jpg")

generate\_caption("86542183\_5e312ae4d4.jpg")

generate\_caption("103106960\_e8a41d64f8.jpg")

generate\_caption("110595925\_f3395c8bd6.jpg")

import matplotlib.pyplot as plt

# Assume 'history' is the variable that stores the history of training

history = model.fit(generator, epochs=epochs, steps\_per\_epoch=steps, verbose=1)

# Plot training loss

plt.plot(history.history['loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train'], loc='upper right')

plt.show()

**App.py:**

from flask import Flask, rendertemplate, request

importcv2

fromkeras.modelsimportloadmodel

importnumpyasnp

fromkeras.applications.resnetimportResNet50

fromkeras.layersimportDense, LSTM, T imeDistributed, Embedding, Activation, RepeatV ector, Concatenate

fromkeras.modelsimportSequential, Model

fromkeras.preprocessingimportimage, sequence

importcv2

fromkeras.utilsimportpadsequences

fromtqdmimporttqdm

app = Flask(name)

app.config[’SENDF ILEMAXAGEDEF AULT′] = 1

vocab = np.load(’minevocab.npy′, allowpickle = True)

vocab = vocab.item()

invvocab = v: kfork, vinvocab.items()

print("+" \* 50)

print("vocabulary loaded")

embeddingsize = 128

vocabsize = len(vocab)

maxlen = 40

imagemodel = Sequential()

imagemodel.add(Dense(embeddingsize, inputshape = (2048,), activation =′relu′))

imagemodel.add(RepeatV ector(maxlen))

languagemodel = Sequential()

languagemodel.add(Embedding(inputdim=vocabsize,outputdim = embeddingsize, inputlength =maxlen))

languagemodel.add(LSTM(256, returnsequences = True))

languagemodel.add(T imeDistributed(Dense(embeddingsize)))

conca = Concatenate()([imagemodel.output, languagemodel.output])

x = LSTM(128, returnsequences = T rue)(conca)

x = LSTM(512, returnsequences = F alse)(x)

x = Dense(vocabsize)(x)

out = Activation(’softmax’)(x)

model = Model(inputs=[imagemodel.input, languagemodel.input], outputs = out)

model.compile(loss=’categoricalcrossentropy′, optimizer =′ RMSprop′, metrics = [′accuracy′])

model.loadweights(′minemodelweights.h5′)

print("=" \* 150)

print("MODEL LOADED")

resnet = ResNet50(includetop = F alse, weights =′imagenet′, inputshape = (224, 224, 3), pooling =′avg′)

app = Flask(name)

app.config[’SENDFILEMAXAGEDEFAULT′] = 1

**Index.html:**

<div class="main-background">

<div class="main-head">

<h1 data-shadow=’dang!’ class="main-caption">Image Caption Generator</h1>

</div>

<div class="main-form">

<form action="urlf or(′after′)”method =′ POST′

enctype =′ multipart/form − data′

style =”margin − top : 150px; ”class = ”main − form” >

< inputtype = ”f ile”name =′file1′

class = ”main − img” >

< inputtype = ”submit”name = ”btn”value =′ Generate′

class =′button′ >

< /form >

< /div >

< /div >

**Base.html:**

<!doctype html>

<html lang="en">

<head>

<!– Required meta tags –>

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!– Bootstrap CSS –>

<linkrel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css"

integrity="sha384JcKb8q3iqJ61gNV9KGb8thSsNjpSL0n8PARn9HuZOnIxN0hoP+VmmDGMN5t9UJ0Z"

crossorigin="anonymous">

<link rel="stylesheet" href="urlf or(′static′, filename =′ xyz.css′)” >

<title>Image Captioning</title>

</head>

<body class="backgroundmainP age”style = ”height : 100vh; ” >

<!– Optional JavaScript –>

<!– jQuery first, then Popper.js, then Bootstrap JS –>

<script src="https://code.jquery.com/jquery-3.5.1.slim.min.js" integrity="sha384-DfXdz2htPH0lsSSs5nCTpuj/zy4C+OGpamoFVy38MVBnE+IbbVYUew+OrC”

crossorigin="anonymous"></script>

<scriptsrc="https://cdn.jsdelivr.net/npm/popper.js@1.16.1/dist/umd/popper.min.j" integrity="sha3849/reFTGAW83EW2RDu2S0VKaIzap3H66lZH81PoYlFhbGU+6BZp6G7niu735Sk7lN" crossori gin="anonymous"></script>

<scriptsrc="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"integrity="sha384B4gt1jrGC7Jh4AgTPSdUtOBvfO8shuf57BaghqFfPlYxofvL8/KUEfYiJOMMV+rV" crossorigin="anonymous"></script>

</body>

</html>

**After.html:**

<center>

<h1 class=’display-2’ style="color: black;">Generated Description </h1>

<div>

</div>

<img src="urlf or(′static′, filename =′

file.jpg′)”alt = ”image”height = 400pxclass = ”img” >

< h2class =′ head2′ >

< span > ” < /span > data < span > ” < /span >

< /h2 >

< br >

< AHREF =′ http : //127.0.0.1 : 5000/′

class =′buttonbtn′ > T rywithanewimage < /A >

< /center >