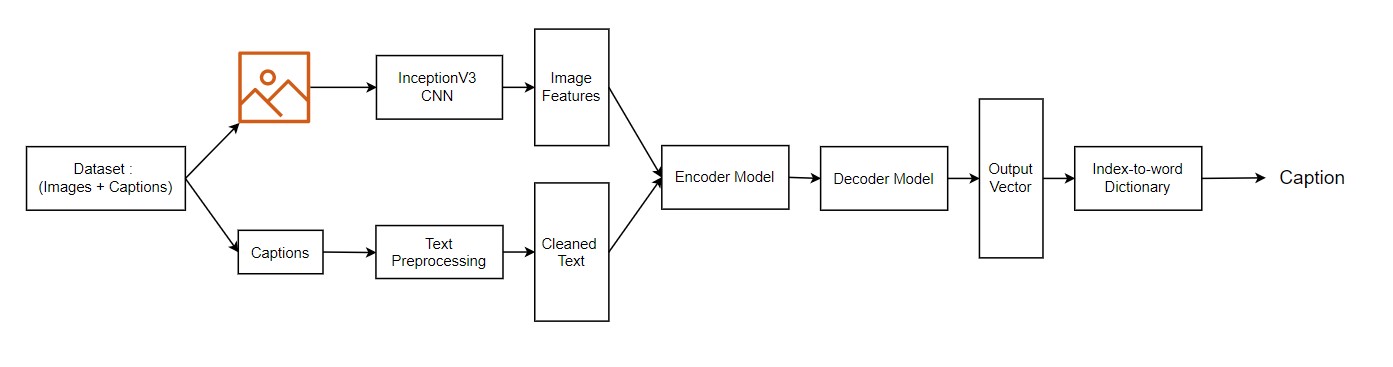
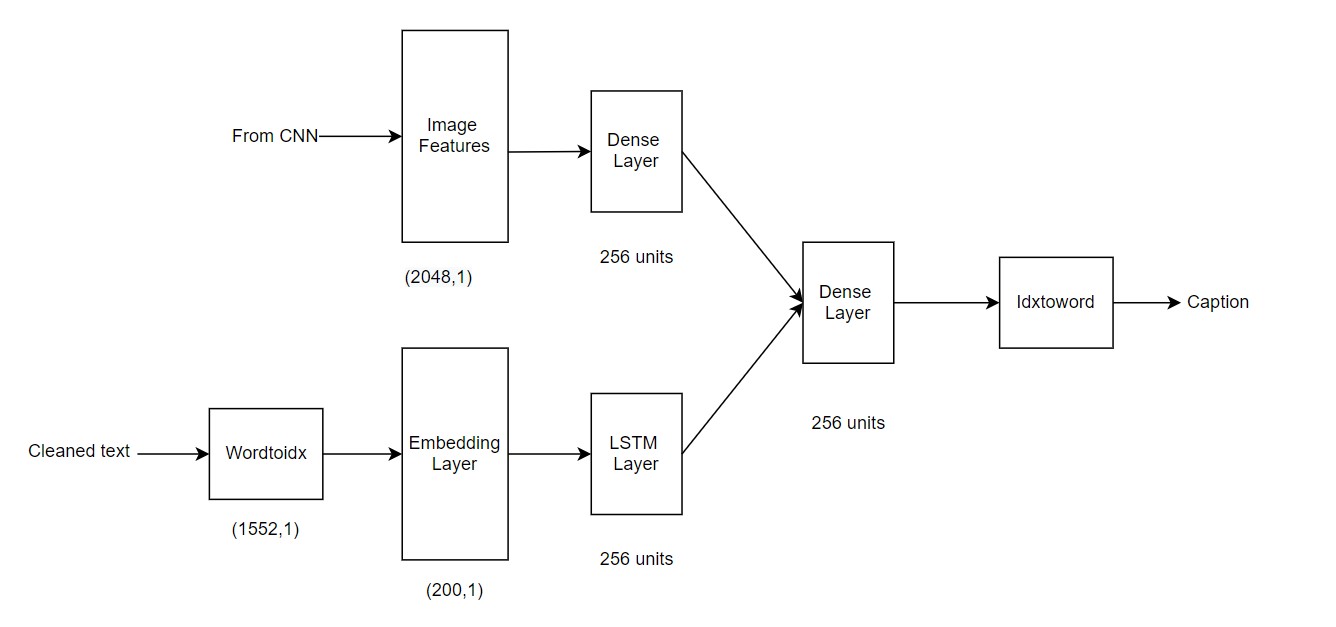
IMAGE CAPTIONER

Name: Appasamudram Naga Mohith

Register No: 122003032

Logic:



Dataset used:

<https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_Dataset.zip>

<https://github.com/jbrownlee/Datasets/releases/download/Flickr8k/Flickr8k_text.zip>

Environment: Google Colab

Python code:

from google.colab import drive

drive.mount('/content/drive')

!unzip "/content/drive/MyDrive/Image Captioning/Flickr8k\_Dataset.zip"

!unzip "/content/drive/MyDrive/Image Captioning/Flickr8k\_text.zip"

# Importing libraries

import numpy as np

import pandas as pd

from time import time

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.keras.layers import (Input,Flatten,TimeDistributed,RepeatVector,Reshape,

Dense,LSTM,Activation,Dropout,BatchNormalization,

concatenate,Embedding,add)

from tensorflow.keras.models import Model,Sequential

from tensorflow.keras.preprocessing.image import load\_img,img\_to\_array

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.optimizers import Adam,RMSprop

from tensorflow.keras.applications import MobileNet,InceptionV3,VGG16

from tensorflow.keras.applications.mobilenet import preprocess\_input as preMobile

from tensorflow.keras.applications.inception\_v3 import preprocess\_input as preInception

from tensorflow.keras.applications.vgg16 import preprocess\_input as preVgg

from tensorflow.keras.utils import to\_categorical

import os

import string

import glob

import pickle

from PIL import Image

from tqdm import tqdm

START = "startseq"

STOP = "stopseq"

EPOCHS = 20

def hms\_string(sec\_elapsed):

h = int(sec\_elapsed / (60 \* 60))

m = int((sec\_elapsed % (60 \* 60)) / 60)

s = sec\_elapsed % 60

return f"{h}:{m:>02}:{s:>05.2f}"

# Data Preprocessing

null\_punct = str.maketrans('','',string.punctuation)

look\_up = dict()

file = open("/content/Flickr8k.token.txt",'r')

maxlength = 0

for line in file.read().split('\n'):

if len(line)<2:

continue

words = line.split()

photoid,caption=words[0],words[1:]

photoid = photoid.split('#')[0]

# Cleaning the captions

caption = [word.lower() for word in caption]

caption = [word.translate(null\_punct) for word in caption]

caption = [word for word in caption if len(word)>1]

caption = [word for word in caption if word.isalpha()]

maxlength = max(len(caption),maxlength)

caption = ' '.join(caption)

if photoid not in look\_up:

look\_up[photoid] = list()

look\_up[photoid].append(caption)

print(f'LookUp Table i.e photo-to-caption mapping size : {len(look\_up)}')

vocabulary = set() # a new vocabulary

for key in look\_up:

[vocabulary.update(d.split()) for d in look\_up[key]]

print(f'Vocabulary size : {len(vocabulary)}')

print(f'Max Caption Length : {maxlength}')

data = pd.DataFrame(look\_up.items())

data.iloc[0,1]

img = glob.glob(os.path.join('/content/Flicker8k\_Dataset','\*.jpg'))

len(img)

# Preparing Training Dataset

train\_images = []

file = open('/content/Flickr\_8k.trainImages.txt')

for line in file.read().strip().split('\n'):

if len(line)<2:

continue

train\_images.append(line)

file.close()

test\_images = []

file = open('/content/Flickr\_8k.testImages.txt')

for line in file.read().strip().split('\n'):

if len(line)<2:

continue

test\_images.append(line)

file.close()

dev\_images = []

file = open('/content/Flickr\_8k.devImages.txt')

for line in file.read().strip().split('\n'):

if len(line)<2:

continue

dev\_images.append(line)

file.close()

print(f'#Train Images = {len(train\_images)}')

print(f'#Test Images = {len(test\_images)}')

print(f'#Dev Images = {len(dev\_images)}')

train\_dataset = dict()

for photoid in train\_images:

if len(photoid)<2:

continue

train\_dataset[photoid] = list()

captions = look\_up[photoid]

for caption in captions:

caption = START + " " + caption +" "+STOP

train\_dataset[photoid].append(caption)

print(f'Train Dataset Ready !! {len(train\_dataset)} images')

len(train\_dataset)

for k,v in train\_dataset.items():

print(k)

for val in v:

print(val)

break

traindata = pd.DataFrame(train\_dataset.items())

traindata.head()

# Encoding train images

#encode\_model = MobileNet(weights='imagenet',include\_top=False)

#WIDTH,HEIGHT,OUTPUT\_DIM = 224,224,50176

encode\_model = InceptionV3(weights='imagenet')

encode\_model = Model(encode\_model.input,encode\_model.layers[-2].output)

WIDTH = 299

HEIGHT = 299

OUTPUT\_DIM = 2048

encode\_model.summary()

def encodeimage(img):

img = img.resize((WIDTH,HEIGHT),Image.ANTIALIAS)

X = img\_to\_array(img)

X = np.expand\_dims(img,axis=0)

#X = preMobile(X)

X = preInception(X)

X = encode\_model.predict(X)

X = np.reshape(X,OUTPUT\_DIM)

return X

start = time()

train\_encodings = dict()

for photoid in train\_images:

if len(photoid)<2:

continue

path = '/content/Flicker8k\_Dataset/' + photoid

train\_image = load\_img(path,target\_size=(HEIGHT,WIDTH))

train\_encodings[photoid] = encodeimage(train\_image)

with open('Train\_Images\_encoding\_inception.pkl','wb') as fp:

pickle.dump(train\_encodings,fp)

print(f'\nGenerating training set took: {hms\_string(time()-start)}')

with open('/content/drive/MyDrive/Image Captioning/Train\_Images\_encoding\_inception.pkl','wb') as fp:

pickle.dump(train\_encodings,fp)

len(train\_encodings)

train\_enc = pd.DataFrame(train\_encodings.items())

train\_enc.iloc[0,1].shape

start = time()

test\_encodings = dict()

for photoid in test\_images:

if len(photoid)<2:

continue

path = '/content/Flicker8k\_Dataset/' + photoid

test\_image = load\_img(path,target\_size=(HEIGHT,WIDTH))

test\_encodings[photoid] = encodeimage(test\_image)

with open('Test\_Images\_encoding\_inception.pkl','wb') as fp:

pickle.dump(test\_encodings,fp)

print(f'\nGenerating test set took: {hms\_string(time()-start)}')

with open('/content/drive/MyDrive/Image Captioning/Test\_Images\_encoding\_inception.pkl','wb') as fp:

pickle.dump(test\_encodings,fp)

all\_train\_captions = list()

for photoid in train\_dataset:

for cap in train\_dataset[photoid]:

all\_train\_captions.append(cap)

len(all\_train\_captions)

threshold = 10

word\_count = dict()

for cap in all\_train\_captions:

for w in cap.split(' '):

word\_count[w] = word\_count.get(w,0) + 1

vocab = [w for w,count in word\_count.items() if count>threshold]

print('preprocessed words %d ==> %d' % (len(word\_count), len(vocab)))

idxtoword = {}

wordtoidx = {}

ix = 1

for w in vocab:

wordtoidx[w] = ix

idxtoword[ix] = w

ix += 1

vocabsize = len(idxtoword) + 1

vocabsize

maxlength = 34 #(32+2)

"""# Data Generator"""

def data\_generator(descriptions, photos, wordtoidx,max\_length, num\_photos\_per\_batch):

# x1 - Training data for photos

# x2 - The caption that goes with each photo

# y - The predicted rest of the caption

x1, x2, y = [], [], []

n=0

while True:

for key, desc\_list in descriptions.items():

n+=1

photo = photos[key]

# Each photo has 5 descriptions

for desc in desc\_list:

# Convert each word into a list of sequences.

seq = [wordtoidx[word] for word in desc.split(' ') \

if word in wordtoidx]

# Generate a training case for every possible sequence and outcome

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=vocabsize)[0]

x1.append(photo)

x2.append(in\_seq)

y.append(out\_seq)

if n==num\_photos\_per\_batch:

yield ([np.array(x1), np.array(x2)], np.array(y))

x1, x2, y = [], [], []

n=0

"""# Loading Glove Vectors"""

f = open('/content/glove.6B.200d.txt',encoding="utf-8")

embeddings\_index = {}

for line in tqdm(f):

values = line.split()

word = values[0]

coefs = np.asarray(values[1:], dtype='float32')

embeddings\_index[word] = coefs

f.close()

print(f'Found {len(embeddings\_index)} word vectors.')

"""# Building the Neural Network

"""

embedding\_dim = 200

# Get 200-dim dense vector for each of the 10000 words in out vocabulary

embedding\_matrix = np.zeros((vocabsize,embedding\_dim))

for word,i in wordtoidx.items():

embedding\_vector = embeddings\_index.get(word)

if embedding\_vector is not None:

embedding\_matrix[i] = embedding\_vector

embedding\_matrix.shape

inputs1 = Input(shape=(OUTPUT\_DIM,))

fe1 = Dropout(0.5)(inputs1)

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

inputs2 = Input(shape=(maxlength,))

se1 = Embedding(vocabsize,embedding\_dim,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(vocabsize,activation='softmax')(decoder2)

caption\_model = Model(inputs=[inputs1,inputs2],outputs=outputs)

caption\_model.summary()

caption\_model.layers[2]

caption\_model.layers[2].set\_weights([embedding\_matrix])

caption\_model.layers[2].trainable=False

caption\_model.compile(loss='categorical\_crossentropy',optimizer='adam',metrics=[('accuracy')])

caption\_model.summary()

num\_pics\_per\_batch = 3

steps = len(train\_dataset)//num\_pics\_per\_batch

start = time()

for i in tqdm(range(EPOCHS\*2)):

generator = data\_generator(train\_dataset,train\_encodings,wordtoidx,maxlength,num\_pics\_per\_batch)

caption\_model.fit\_generator(generator,epochs=1,steps\_per\_epoch=steps,verbose=1)

caption\_model.optimizer.lr = 1e-4

num\_pics\_per\_batch = 6

steps = len(train\_dataset)//num\_pics\_per\_batch

for i in range(EPOCHS):

generator = data\_generator(train\_dataset,train\_encodings,wordtoidx,maxlength,num\_pics\_per\_batch)

caption\_model.fit\_generator(generator,epochs=1,steps\_per\_epoch=steps,verbose=1)

caption\_model.save('cpmodel.h5')

caption\_model.save\_weights('cpmodelweights.h5')

print(f"\Training took: {hms\_string(time()-start)}")

def generateCaption(photo):

in\_text = START

for i in range(maxlength):

sequence = [wordtoidx[w] for w in in\_text.split() if w in wordtoidx]

sequence = pad\_sequences([sequence], maxlen=maxlength)

yhat = caption\_model.predict([photo,sequence], verbose=0)

yhat = np.argmax(yhat)

word = idxtoword[yhat]

in\_text += ' ' + word

if word == STOP:

break

final = in\_text.split()

final = final[1:-1]

final = ' '.join(final)

return final

caption\_model.save('/content/drive/MyDrive/Image Captioning/cpmodel.h5')

caption\_model.save\_weights('/content/drive/MyDrive/Image Captioning/cpmodelweights.h5')

for i in range(20):

pic = list(test\_encodings.keys())[100+i]

image = test\_encodings[pic].reshape((1,OUTPUT\_DIM))

path = '/content/Flicker8k\_Dataset/' + pic

print(path)

X = plt.imread(path)

plt.imshow(X)

plt.show()

print("Caption: ",generateCaption(image))

print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

OUTPUT



