A project report on

Image Caption Generator

Submitted by:

Rajus Nagwekar - 60002180080
Ritwik Khandelwal – 60002180081
Sarthak Mistry – 60002180098
Sarthak Dalal - 60002180099
Shrey Desai - 60002180106
Shrenik Ganguli – 60002180105

Guided by:

Dwarkadas J. Sanghvi College of Engineering

Mumbai University

2018-2022

Table of Contents

Background

Aim

Our brain is capable of identifying what an image is about, but can a computer tell what the image is representing? With the advancement in Deep learning techniques, availability of huge datasets and computer power, we can build models that can generate captions for an image. This is what we are going to implement in this **Python based project** where we will use deep learning techniques of Convolutional Neural Networks and a type of Recurrent Neural Network (LSTM) together.

Technologies:

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

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

Hardware Architecture:

The project is completely performed through software, and it did not require any hardware.

Software Architecture:

The following were used in this project :

- Tensorflow
- Keras
- Pillow
- Numpy
- Scikit Learn
- CV2

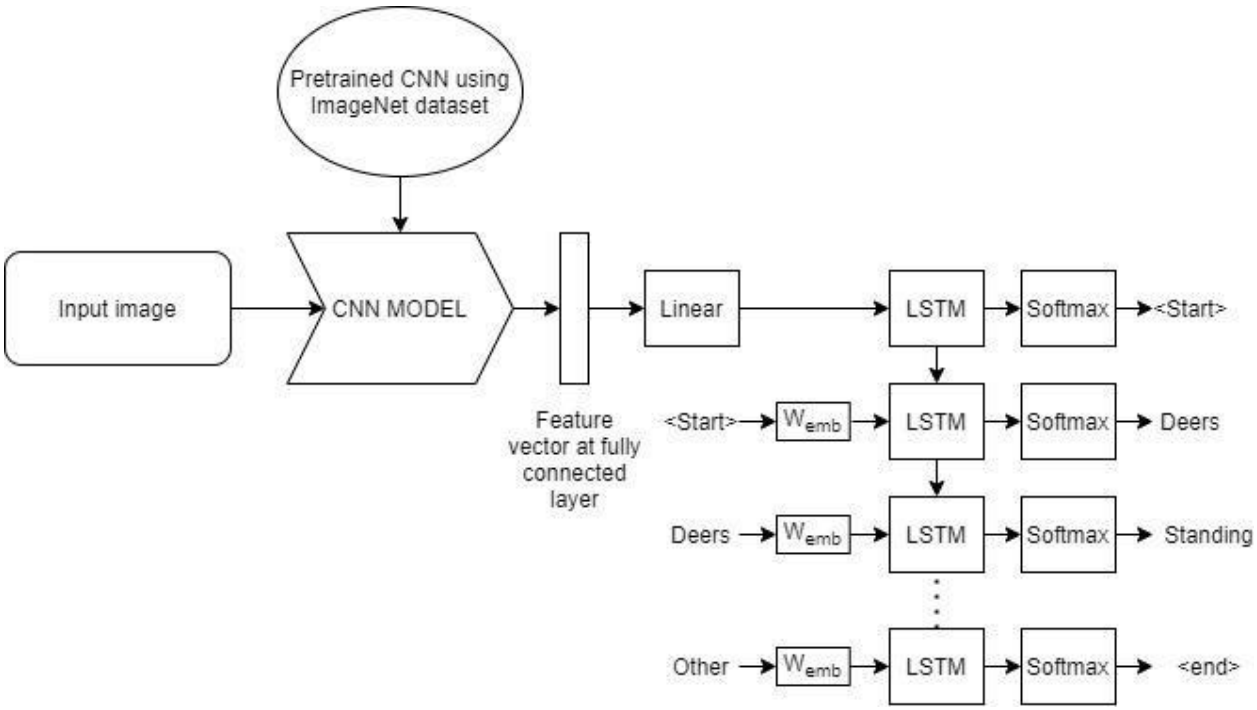
- tqdm
- jupyterlab

System

Architecture:

So, to make our image caption generator model, we merged CNN and RNN models. It is also called a CNN-RNN model.

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



Implementation:

1) Importing necessary packages:

```
In [1]: import string
        import numpy as np
        from PIL import Image
        import os
        from pickle import dump, load
        import numpy as np

        from keras.applications.xception import Xception, preprocess_input
        from keras.preprocessing.image import load_img, img_to_array
        from keras.preprocessing.text import Tokenizer
        from keras.preprocessing.sequence import pad_sequences
        from keras.utils import to_categorical
        from keras.layers.merge import add
        from keras.models import Model, load_model
        from keras.layers import Input, Dense, LSTM, Embedding, Dropout

        # small library for seeing the progress of loops.
        from tqdm import tqdm_notebook as tqdm
        tqdm().pandas()
```

2) Data Cleaning:
The main text file which contains all image captions is **Flickr 8k.token** in our **Flickr_8k_text** folder.

Have a look at the file –

File Edit Format Run Options Window Help

```
1000268201_693b08cb0e.jpg#0     A child in a pink dress is climbing up a set of stairs in an entry way .
1000268201_693b08cb0e.jpg#1     A girl going into a wooden building .
1000268201_693b08cb0e.jpg#2     A little girl climbing into a wooden playhouse .
1000268201_693b08cb0e.jpg#3     A little girl climbing the stairs to her playhouse .
1000268201_693b08cb0e.jpg#4     A little girl in a pink dress going into a wooden cabin .
1001773457_577c3a7d70.jpg#0     A black dog and a spotted dog are fighting
1001773457_577c3a7d70.jpg#1     A black dog and a tri-colored dog playing with each other on the road .
1001773457_577c3a7d70.jpg#2     A black dog and a white dog with brown spots are staring at each other in the
1001773457_577c3a7d70.jpg#3     Two dogs of different breeds looking at each other on the road .
1001773457_577c3a7d70.jpg#4     Two dogs on pavement moving toward each other .
1002674143_1b742ab4b8.jpg#0     A little girl covered in paint sits in front of a painted rainbow with her han
1002674143_1b742ab4b8.jpg#1     A little girl is sitting in front of a large painted rainbow .
1002674143_1b742ab4b8.jpg#2     A small girl in the grass plays with fingerpaints in front of a white canvas w
1002674143_1b742ab4b8.jpg#3     There is a girl with pigtails sitting in front of a rainbow painting .
1002674143_1b742ab4b8.jpg#4     Young girl with pigtails painting outside in the grass .
1003163366_44323f5815.jpg#0     A man lays on a bench while his dog sits by him .
1003163366_44323f5815.jpg#1     A man lays on the bench to which a white dog is also tied .
1003163366_44323f5815.jpg#2     a man sleeping on a bench outside with a white and black dog sitting next to h
1003163366_44323f5815.jpg#3     A shirtless man lies on a park bench with his dog .
1003163366_44323f5815.jpg#4     man laying on bench holding leash of dog sitting on ground
1007129816_e794419615.jpg#0     A man in an orange hat starring at something .
1007129816_e794419615.jpg#1     A man wears an orange hat and glasses .
1007129816_e794419615.jpg#2     A man with gauges and glasses is wearing a Blitz hat .
1007129816_e794419615.jpg#3     A man with glasses is wearing a beer can crocheted hat .
1007129816_e794419615.jpg#4     The man with pierced ears is wearing glasses and an orange hat .
1007320043_627395c3d8.jpg#0     A child playing on a rope net .
```

Each image has 5 captions and we can see that #(0 to 5)number is assigned for each caption.

We will define 5 functions:

- **load_doc(filename)** – For loading the document file and reading the contents inside the file into a string.
- **all_img_captions(filename)** – This function will create a description dictionary that maps images with a list of 5 captions.

- **cleaning_text(descriptions)** – This function takes all descriptions and performs data cleaning. This is an important step when we work with textual data. According to our goal, we decide what type of cleaning we want to perform on the text. In our case, we will be removing punctuations, converting all text to lowercase and removing words that contain numbers. So, a caption like “A man riding on a three-wheeled wheelchair” will be transformed into “man riding on three wheeled wheelchair”.
- **text_vocabulary(descriptions)** – This is a simple function that will separate all the unique words and create the vocabulary from all the descriptions.
- **save_descriptions(descriptions, filename**) – This function will create a list of all the descriptions that have been preprocessed and store them into a file. We will create a descriptions.txt file to store all the captions. It will look something like this:

In [2]:
```
# Loading a text file into memory
def load_doc(filename):
    # Opening the file as read only
    file = open(filename, 'r')
    text = file.read()
    file.close()
    return text
```

In [3]:
```
# get all imgs with their captions
def all_img_captions(filename):
    file = load_doc(filename)
    captions = file.split('\n')
    descriptions ={}
    for caption in captions[:-1]:
        img, caption = caption.split('\t')
        if img[:-2] not in descriptions:
            descriptions[img[:-2]] = [caption]
        else:
            descriptions[img[:-2]].append(caption)
    return descriptions
```

In [4]:
```
##Data cleaning- lower casing, removing puntuations and words containing numbers
def cleaning_text(captions):
    table = str.maketrans('','',string.punctuation)
    for img,caps in captions.items():
        for i,img_caption in enumerate(caps):

            img_caption.replace("-"," ")
            desc = img_caption.split()

            #converts to lower case
            desc = [word.lower() for word in desc]
            #remove punctuation from each token
            desc = [word.translate(table) for word in desc]
            #remove hanging 's and a
            desc = [word for word in desc if(len(word)>1)]
            #remove tokens with numbers in them
            desc = [word for word in desc if(word.isalpha())]
            #convert back to string

            img_caption = ' '.join(desc)
            captions[img][i]= img_caption
    return captions
```

In [5]:
```
def text_vocabulary(descriptions):
    # build vocabulary of all unique words
    vocab = set()

    for key in descriptions.keys():
        [vocab.update(d.split()) for d in descriptions[key]]

    return vocab
```

In [6]:
```
#All descriptions in one file
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + '\t' + desc )
    data = "\n".join(lines)
    file = open(filename,"w")
    file.write(data)
    file.close()
```

3) Extracting the feature vector from all images :

In [9]:
```
def extract_features(directory):
        model = Xception( include_top=False, pooling='avg' )
        features = {}
        for img in tqdm(os.listdir(directory)):
            filename = directory + "/" + img
            image = Image.open(filename)
            image = image.resize((299,299))
            image = np.expand_dims(image, axis=0)
            #image = preprocess_input(image)
            image = image/127.5
            image = image - 1.0

            feature = model.predict(image)
            features[img] = feature
        return features
```

4) Loading dataset for Training the model:

In [11]:
```
def load_photos(filename):
    file = load_doc(filename)
    photos = file.split("\n")[:-1]
    return photos


def load_clean_descriptions(filename, photos):
    #loading clean_descriptions
    file = load_doc(filename)
    descriptions = {}
    for line in file.split("\n"):

        words = line.split()
        if len(words)<1 :
            continue

        image, image_caption = words[0], words[1:]

        if image in photos:
            if image not in descriptions:
                descriptions[image] = []
            desc = '<start> ' + " ".join(image_caption) + ' <end>'
            descriptions[image].append(desc)

    return descriptions


def load_features(photos):
    #loading all features
    all_features = load(open("features.p","rb"))
    #selecting only needed features
    features = {k:all_features[k] for k in photos}
    return features
```

In [12]:
```
filename = dataset_text + "/" + "Flickr_8k.trainImages.txt"

#train = loading_data(filename)
train_imgs = load_photos(filename)
train_descriptions = load_clean_descriptions("descriptions.txt", train_imgs)
train_features = load_features(train_imgs)
```

5) Tokenizing the vocabulary:

In [13]:
```
#converting dictionary to clean list of descriptions
def dict_to_list(descriptions):
    all_desc = []
    for key in descriptions.keys():
        [all_desc.append(d) for d in descriptions[key]]
    return all_desc

#creating tokenizer class
#this will vectorise text corpus
#each integer will represent token in dictionary

from keras.preprocessing.text import Tokenizer

def create_tokenizer(descriptions):
    desc_list = dict_to_list(descriptions)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(desc_list)
    return tokenizer
```

In [14]:
```
# give each word a index, and store that into tokenizer.p pickle file
tokenizer = create_tokenizer(train_descriptions)
dump(tokenizer, open('tokenizer.p', 'wb'))
vocab_size = len(tokenizer.word_index) + 1
vocab_size
```

Out[14]:
```
7577
```

In [15]:
```
#calculate maximum length of descriptions
def max_length(descriptions):
    desc_list = dict_to_list(descriptions)
    return max(len(d.split()) for d in desc_list)

max_length = max_length(descriptions)
max_length
```

Out[15]:
```
32
```

In [16]:
```
features['1000268201_693b08cb0e.jpg'][0]
```

Out[16]:
```
array([0.36452794, 0.12713662, 0.0013574 , ..., 0.221817  , 0.01178991,
       0.24176797], dtype=float32)
```

6) Create Data generator:

In [17]:
```
#create input-output sequence pairs from the image description.

#data generator, used by model.fit_generator()
def data_generator(descriptions, features, tokenizer, max_length):
    while 1:
        for key, description_list in descriptions.items():
            #retrieve photo features
            feature = features[key][0]
            input_image, input_sequence, output_word = create_sequences(tokenizer, max_length, description_list, feature)
            yield [[input_image, input_sequence], output_word]

def create_sequences(tokenizer, max_length, desc_list, feature):
    X1, X2, y = list(), list(), list()
    # walk through each description for the image
    for desc in desc_list:
        # encode the sequence
        seq = tokenizer.texts_to_sequences([desc])[0]
        # split one sequence into multiple X,y pairs
        for i in range(1, len(seq)):
            # split into input and output pair
            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
            X1.append(feature)
            X2.append(in_seq)
            y.append(out_seq)
    return np.array(X1), np.array(X2), np.array(y)
```

In [18]:
```
[a,b],c = next(data_generator(train_descriptions, features, tokenizer, max_length))
a.shape, b.shape, c.shape
```

Out[18]:
```
((47, 2048), (47, 32), (47, 7577))
```

7) Defining the CNN-RNN model:

In [19]:
```
from keras.utils import plot_model

# define the captioning model
def define_model(vocab_size, max_length):

    # features from the CNN model squeezed from 2048 to 256 nodes
    inputs1 = Input(shape=(2048,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    # LSTM sequence model
    inputs2 = Input(shape=(max_length,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

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

    # tie it together [image, seq] [word]
    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    # summarize model
    print(model.summary())
    plot_model(model, to_file='model.png', show_shapes=True)

    return model
```

8) Training the model:

In [24]:
```
# train our model
print('Dataset: ', len(train_imgs))
print('Descriptions: train=', len(train_descriptions))
print('Photos: train=', len(train_features))
print('Vocabulary Size:', vocab_size)
print('Description Length: ', max_length)

model = define_model(vocab_size, max_length)
epochs = 10
steps = len(train_descriptions)
# making a directory models to save our models
os.mkdir("models")
```

```
Dataset:  6000
Descriptions: train= 6000
Photos: train= 6000
Vocabulary Size: 7577
Description Length:  32
Model: "model_4"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_10 (InputLayer)           [(None, 32)]         0
__________________________________________________________________________________________________
input_9 (InputLayer)            [(None, 2048)]       0
__________________________________________________________________________________________________
embedding_4 (Embedding)         (None, 32, 256)      1939712     input_10[0][0]
__________________________________________________________________________________________________
dropout_8 (Dropout)             (None, 2048)         0           input_9[0][0]
__________________________________________________________________________________________________
dropout_9 (Dropout)             (None, 32, 256)      0           embedding_4[0][0]
__________________________________________________________________________________________________
dense_12 (Dense)                (None, 256)          524544      dropout_8[0][0]
__________________________________________________________________________________________________
lstm_4 (LSTM)                   (None, 256)          525312      dropout_9[0][0]
__________________________________________________________________________________________________
add_4 (Add)                     (None, 256)          0           dense_12[0][0]
                                                                 lstm_4[0][0]
__________________________________________________________________________________________________
dense_13 (Dense)                (None, 256)          65792       add_4[0][0]
__________________________________________________________________________________________________
dense_14 (Dense)                (None, 7577)         1947289     dense_13[0][0]
==================================================================================================
Total params: 5,002,649
Trainable params: 5,002,649
Non-trainable params: 0
__________________________________________________________________________________________________
None
('Failed to import pydot. You must `pip install pydot` and install graphviz (https://graphviz.gitlab.io/download/), ', 'for `py
dotprint` to work.')
```

In [26]:
```
for i in range(epochs):
    generator = data_generator(train_descriptions, train_features, tokenizer, max_length)
    model.fit_generator(generator, epochs=1, steps_per_epoch = steps, verbose=1)
    model.save("models/model_" + str(i) + ".h5")
```

Testing

Dataset:

For the image caption generator, we will be using the Flickr_8K dataset.

- Flicker8k_Dataset – Dataset folder which contains 8091 images.
- Flickr_8k_text – Dataset folder which contains text files and captions of images.

Code:

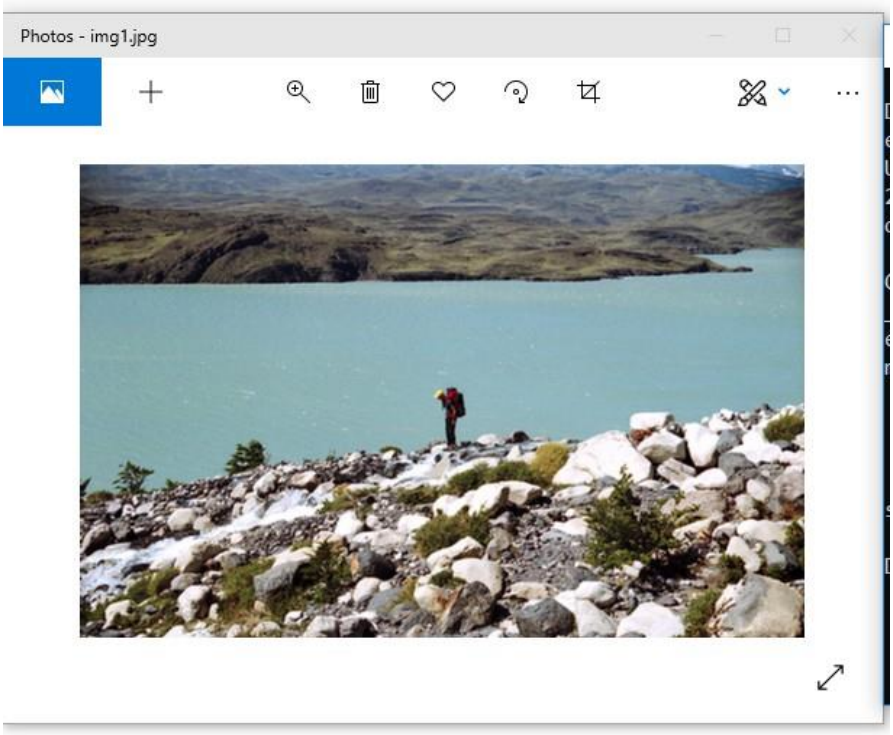
```
from keras.preprocessing.text import Tokenizer

from keras.preprocessing.sequence import pad_sequences

from keras.applications.xception import Xception

from keras.models import load_model

from pickle import load

import numpy as np

from PIL import Image

import matplotlib.pyplot as plt

import argparse




ap = argparse.ArgumentParser()

ap.add_argument('-i', '--image', required=True, help="Image Path")

args = vars(ap.parse_args())

img_path = args['image']
```

```
def extract_features(filename, model):
        try:
            image = Image.open(filename)



        except:
            print("ERROR: Couldn't open image! Make sure the image path and extension is correct")
        image = image.resize((299,299))
        image = np.array(image)
        # for images that has 4 channels, we convert them into 3 channels
        if image.shape[2] == 4:
            image = image[..., :3]
        image = np.expand_dims(image, axis=0)
        image = image/127.5
        image = image - 1.0
        feature = model.predict(image)
        return feature



def word_for_id(integer, tokenizer):
 for word, index in tokenizer.word_index.items():
     if index == integer:
```

```
        return word
    return None
def generate_desc(model, tokenizer, photo, max_length):
    in_text = 'start'
    for i in range(max_length):
        sequence = tokenizer.texts_to_sequences([in_text])[0]
        sequence = pad_sequences([sequence], maxlen=max_length)
        pred = model.predict([photo,sequence], verbose=0)
        pred = np.argmax(pred)
        word = word_for_id(pred, tokenizer)
        if word is None:
            break
        in_text += ' ' + word
        if word == 'end':
            break
    return in_text

#path = 'Flicker8k_Dataset/111537222_07e56d5a30.jpg'
max_length = 32
tokenizer = load(open("tokenizer.p","rb"))
model = load_model('models/model_9.h5')
```

```
xception_model = Xception(include_top=False, pooling="avg")

photo = extract_features(img_path, xception_model)

img = Image.open(img_path)



description = generate_desc(model, tokenizer, photo, max_length)

print("\n\n")

print(description)

plt.imshow(img)
```

Results:




```
start man is standing on rock overlooking the mountains end
```

Conclusion:

In this project, we have implemented a CNN-RNN model by building an image caption generator.
DRAWBACKS:
Some key points to note are that our model depends on the data, so it cannot predict the words that are out of its vocabulary. We used a small dataset consisting of 8000 images. For production-level models, we need to train on datasets larger than 100,000 images which can produce better accuracy models.

References:

1. Flickr8k Dataset- HTTPS://WWW.KAGGLE.COM/ADITYAJN105/FLICKR8K
2. CNN-LSTM Model- https://machinelearningmastery.com/cnn-long-short-term-memory-networks/