NAAN MUDHALVAN

FINAL PROJECT

IMAGE CAPTIONING USING CNN

CAROLINA CHRISTOPHER
3RD YEAR
BTECH CSBS

Table of CONTENTS

01 Problem statement

02 Introduction

03 Overview

04 Scope of the project

05 Solution

06 Modelling

07 Results

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.

INTRODUCTION

Automatically describing the content ofimages 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. 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.

OVERVIEW

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 .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.

SCOPE OF IMAGE CAPTIONING

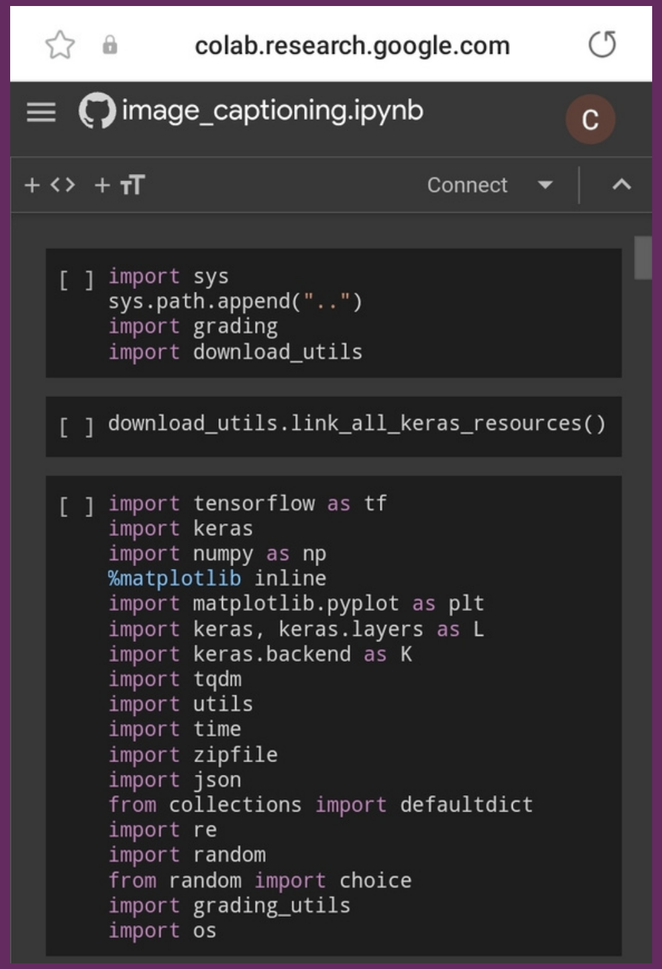
- Image captioning aims to automatically generate descriptions for a given image, i.e., capture the relationship between the objects present in the picture, generate natural language expressions, and judge the quality of the generated descriptions.

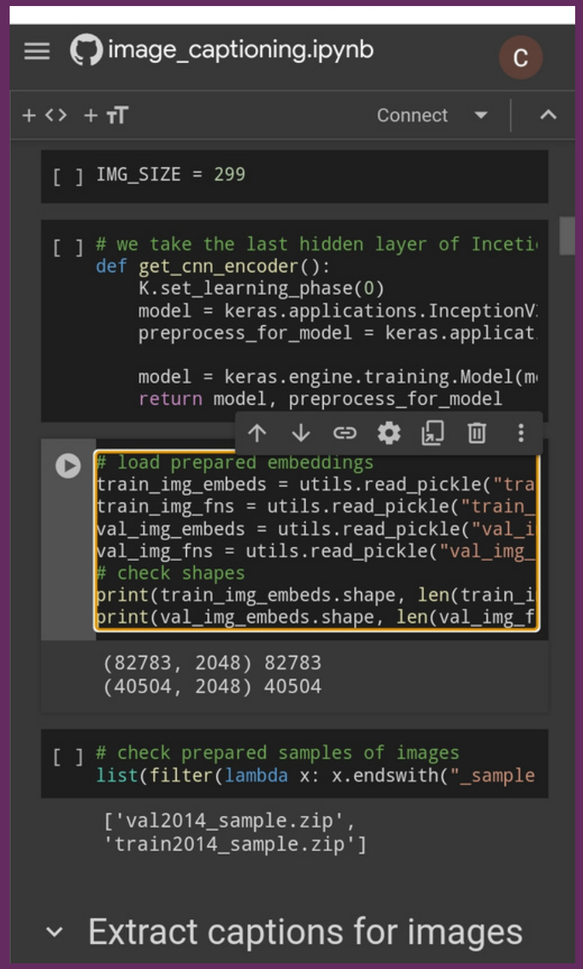
Recommendations in Editing Applications

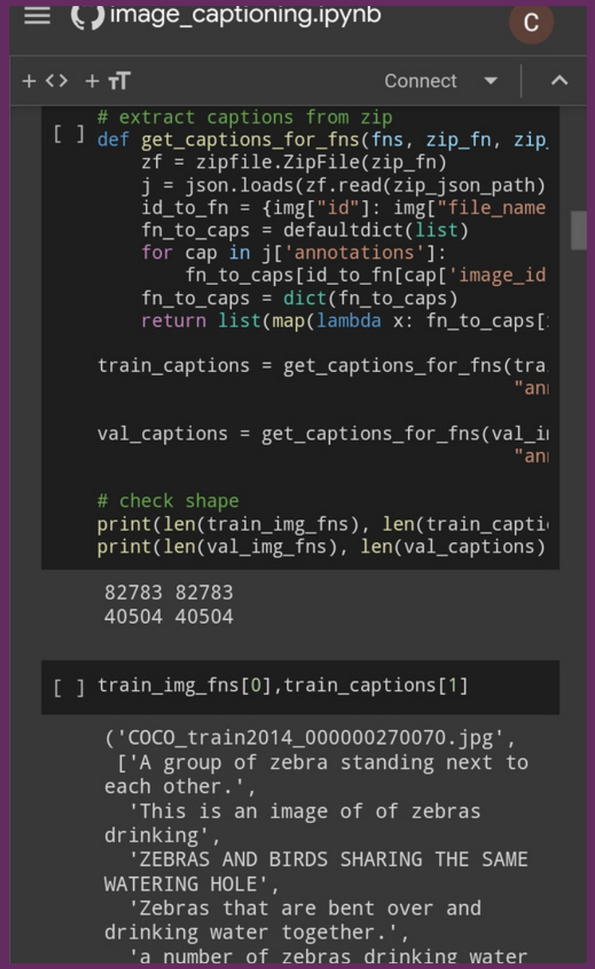
Assistance for Visually Impaired

Social media posts

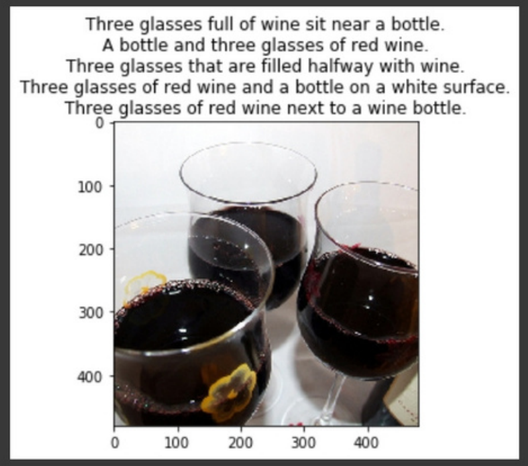
Media and publishing houses


colab.research.google.com
image_captioning.ipynb
C
Connect
[ ] import sys
sys.path.append("..")
import grading
import download_utils
[ ] download_utils.link_all_keras_resources()
[ ] import tensorflow as tf
import keras
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import keras, keras.layers as L
import keras.backend as K
import tqdm
import utils
import time
import zipfile
import json
from collections import defaultdict
import re
import random
from random import choice
import grading_utils
import os


image_captioning.ipynb
C
Connect
[ ] IMG_SIZE = 299
[ ] # we take the last hidden layer of Inceti
def get_cnn_encoder():
K.set_learning_phase(0)
model = keras.applications.InceptionV
preprocess_for_model = keras.applicat
model = keras.engine.training.Model(m
return model, preprocess_for_model
# load prepared embeddings
train_img_embeds = utils.read_pickle("tra
train_img_fns = utils.read_pickle("train_
val_img_embeds = utils.read_pickle("val_i
val_img_fns = utils.read_pickle("val_img_
# check shapes
print(train_img_embeds.shape, len(train_i
print(val_img_embeds.shape, len(val_img_f
(82783, 2048) 82783
(40504, 2048) 40504
[ ] # check prepared samples of images
list(filter(lambda x: x.endswith("_sample
['val2014_sample.zip',
'train2014_sample.zip']
Extract captions for images


image_captioning.ipynb
C
Connect
[ ] # extract captions from zip
def get_captions_for_fns(fns, zip_fn, zip
zf = zipfile.ZipFile(zip_fn)
j = json.loads(zf.read(zip_json_path)
id_to_fn = {img["id"]: img["file_name
fn_to_caps = defaultdict(list)
for cap in j['annotations']:
fn_to_caps[id_to_fn[cap['image_id
fn_to_caps = dict(fn_to_caps)
return list(map(lambda x: fn_to_caps[
train_captions = get_captions_for_fns(tra
"an
val_captions = get_captions_for_fns(val_i
"an
check shape
print(len(train_img_fns), len(train_capti
print(len(val_img_fns), len(val_captions)
82783 82783
40504 40504
[ ] train_img_fns[0],train_captions[1]
('COCO_train2014_000000270070.jpg',
['A group of zebra standing next to
each other.',
'This is an image of of zebras
drinking',
'ZEBRAS AND BIRDS SHARING THE SAME
WATERING HOLE',
'Zebras that are bent over and
drinking water together.',
'a number of zebras drinking water

```
[ ] # look at training example (each has 5 ca
    def show_trainig_example(train_img_fns, t
        """
        You can change example_idx and see di
        """
        zf = zipfile.ZipFile("train2014_sampl
        captions_by_file = dict(zip(train_img
        all_files = set(train_img_fns)
        found_files = list(filter(lambda x: x
        example = found_files[example_idx]
        img = utils.decode_image_from_buf(zf.
        plt.imshow(utils.image_center_crop(im
        plt.title("\n".join(captions_by_file[
        plt.show()

    show_trainig_example(train_img_fns, train
```



image_captioning.ipynb

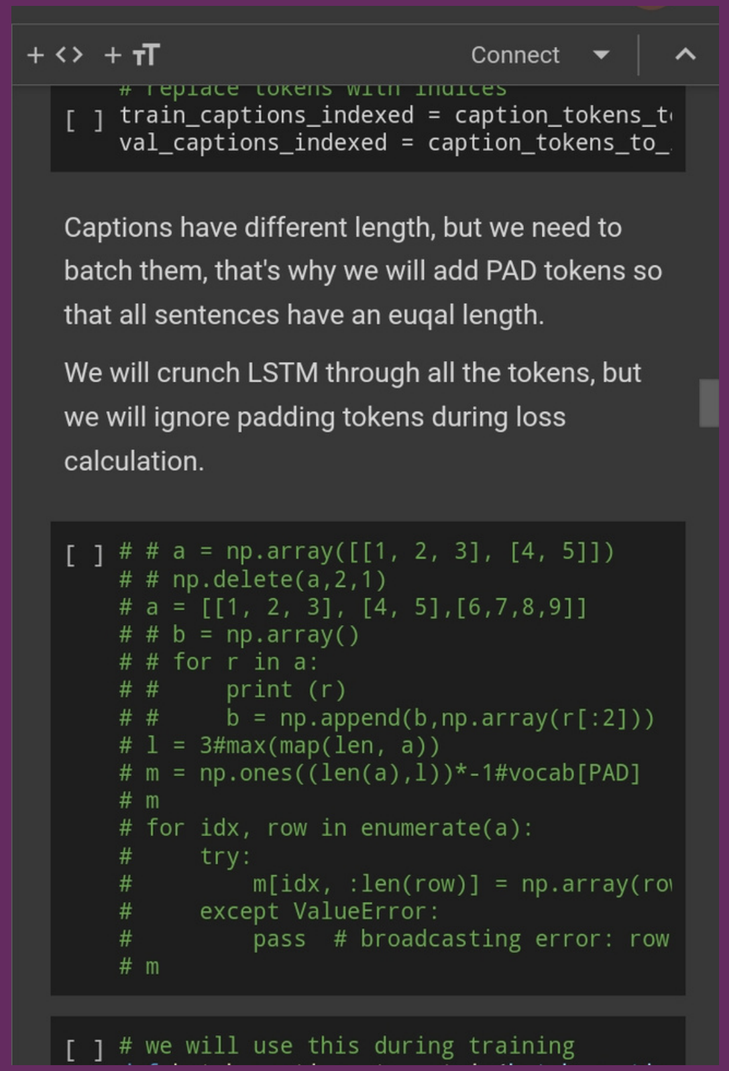
```
[ ] # special tokens
    PAD = "#PAD#"
    UNK = "#UNK#"
    START = "#START#"
    END = "#END#"

    # split sentence into tokens (split into
    def split_sentence(sentence):
        return list(filter(lambda x: len(x) >
```

```
[ ] # words = [word for caption_set in [train_
    # len(words), words[:4]
```

```
[ ] from collections import defaultdict
```

```
[ ] # split_sentence(train_captions[0][0])
```

```
[ ] def generate_vocabulary(train_captions):
        """
        Return {token: index} for all train t
            `index` should be from 0 to N, wh
        Also, add PAD (for batch padding), UN
            START (start of sentence) and END
        """
        vocab = {token: idx for idx,token in
        current_idx = len(vocab)
        frequency = defaultdict(int)
    #     vocab = ### YOUR CODE HERE ###
        for caption_set in [train_captions] :
            for sample in caption_set:
                for caption in sample:
                    for word in split_sentenc
                        frequency[word] +=1
                        if frequency[word] >=
                            vocab[word] = cur
        return {token: index for index, token
```



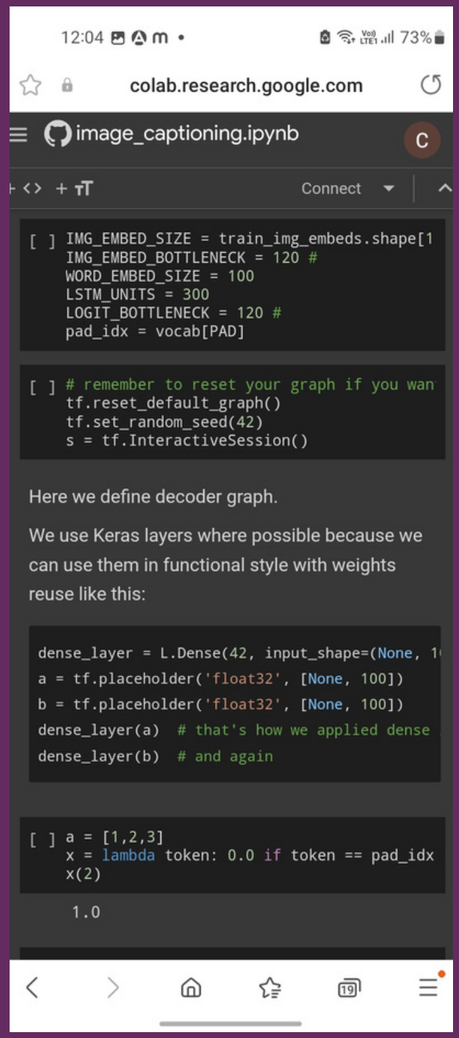
```
# replace tokens with indices
train_captions_indexed = caption_tokens_t
val_captions_indexed = caption_tokens_to_
```

Captions have different length, but we need to batch them, that's why we will add PAD tokens so that all sentences have an euqal length.

We will crunch LSTM through all the tokens, but we will ignore padding tokens during loss calculation.

```
# # a = np.array([[1, 2, 3], [4, 5]])
# # np.delete(a,2,1)
# a = [[1, 2, 3], [4, 5],[6,7,8,9]]
# # b = np.array()
# # for r in a:
# #     print (r)
# #     b = np.append(b,np.array(r[:2]))
# l = 3#max(map(len, a))
# m = np.ones((len(a),l))*-1#vocab[PAD]
# m
# for idx, row in enumerate(a):
#     try:
#         m[idx, :len(row)] = np.array(ro
#     except ValueError:
#         pass  # broadcasting error: row
# m
```

```
# we will use this during training
```



12:04 73%

colab.research.google.com

image_captioning.ipynb

Connect

```
IMG_EMBED_SIZE = train_img_embeds.shape[1
IMG_EMBED_BOTTLENECK = 120 #
WORD_EMBED_SIZE = 100
LSTM_UNITS = 300
LOGIT_BOTTLENECK = 120 #
pad_idx = vocab[PAD]
```

```
# remember to reset your graph if you wan
tf.reset_default_graph()
tf.set_random_seed(42)
s = tf.InteractiveSession()
```

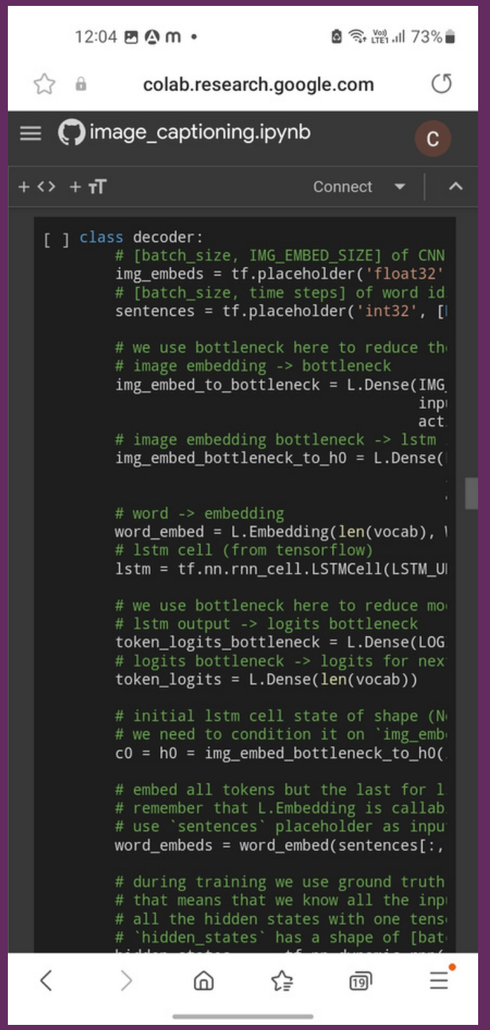
Here we define decoder graph.

We use Keras layers where possible because we can use them in functional style with weights reuse like this:

```
dense_layer = L.Dense(42, input_shape=(None, 1
a = tf.placeholder('float32', [None, 100])
b = tf.placeholder('float32', [None, 100])
dense_layer(a)  # that's how we applied dense
dense_layer(b)  # and again
```

```
a = [1,2,3]
x = lambda token: 0.0 if token == pad_idx
x(2)
```

```
1.0
```



12:04 73%

colab.research.google.com

image_captioning.ipynb

Connect

```
class decoder:
    # [batch_size, IMG_EMBED_SIZE] of CNN
    img_embeds = tf.placeholder('float32'
    # [batch_size, time steps] of word id
    sentences = tf.placeholder('int32', [

    # we use bottleneck here to reduce th
    # image embedding -> bottleneck
    img_embed_to_bottleneck = L.Dense(IMG
                                      inp
                                      act
    # image embedding bottleneck -> lstm
    img_embed_bottleneck_to_h0 = L.Dense(


    # word -> embedding
    word_embed = L.Embedding(len(vocab), 
    # lstm cell (from tensorflow)
    lstm = tf.nn.rnn_cell.LSTMCell(LSTM_U

    # we use bottleneck here to reduce mo
    # lstm output -> logits bottleneck
    token_logits_bottleneck = L.Dense(LOG
    # logits bottleneck -> logits for nex
    token_logits = L.Dense(len(vocab))

    # initial lstm cell state of shape (N
    # we need to condition it on `img_emb
    c0 = h0 = img_embed_bottleneck_to_h0(

    # embed all tokens but the last for l
    # remember that L.Embedding is callab
    # use `sentences` placeholder as inpu
    word_embeds = word_embed(sentences[:,

    # during training we use ground truth
    # that means that we know all the inp
    # all the hidden states with one tens
    # `hidden_states` has a shape of [bat
```

CONCLUSION

In this advanced Python project, an image caption generator has been developed using a CNN-RNN model. Some key aspects about our projectto 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.