DATS 6203: Machine Learning 2
Spring 2022
Instructor: A. Jafari, Ph.D

Image Captioning using Visual Attention in TensorFlow

Group 1: Adel Hassen, Lydia Teinfalt, Xingyu (Alice) Yang
Initials: AH, LT, XY
Last updated: April 26, 2022

Table of Contents

1 Abstract

An image caption is a concise sentence describing what is happening in a picture. A good image caption provides context to an image and is useful for retrieving the image at a later point. For the visual impaired population, an image caption is a vital tool for being able to understand a picture without the ability to directly view it. This project demonstrates machine auto-generated captions for pictures in natural language sentence format by implementing in TensorFlow a machine learning model that leverages visual attention mechanism. We use a subset of a large-scale dataset called MS-COCO (Common Objects in Context) to train the model and then generate test captions for New Yorker cartoons in English as well as German. We evaluate the model using BLEU (Bilingual Evaluation Study) score that compares the official caption with the automated generated caption.

2 Introduction

Humans can easily create a caption that describes the content of a picture using a few words. An image caption captures the facts and helps the reader understand what they see in the picture. Image captions are useful not only to people but also to machines. Images with captions are optimized for search engines because they can be indexed by a few words and thus, increasing the accuracy and speed of information retrieval. As machine learning models become more accurate and descriptive in captioning images, they can be used catalog digital archives that are missing descriptions. A task that might take an individual person several years to complete.

Image captions increase accessibility for the visually impaired. The text describing an image is critical to ensure that users with disabilities who cannot see an image can use a tool like a screen reader on a web page to have it read to them instead. Section 508 of the Rehabilitation Act requires that Federal Government must provide U.S. government agencies provide accessibility to electronic information to the public so a machine learning model that can captions for images would be beneficial in staying compliant with the regulation.

This project demonstrates that machine learning models can be trained to generate captions to images. We are implementing a model architecture in TensorFlow based on the paper published by Xu, Kelvin et al called Show, Attend and Tell: Neural Image CaptionGeneration with Visual Attention (Xu, 2016) published in April 2006. The model takes in a raw image uses convolutional neural networks (CNN) to identify objects within an image, translate the image to text, use visual attention mechanism to determine what part of the image the model should focus on and then creates a caption in natural language.

To write a good image caption is a science as well as an art. We use BLEU (Bilingual Evaluation Study) to evaluate captions generated by the model. Instead of measuring accuracy, we can use BLEU score to evaluate quality of machine-generated text against reference text. We compare the results of the model we built against the results reported by the Show, Attend and Tell paper.

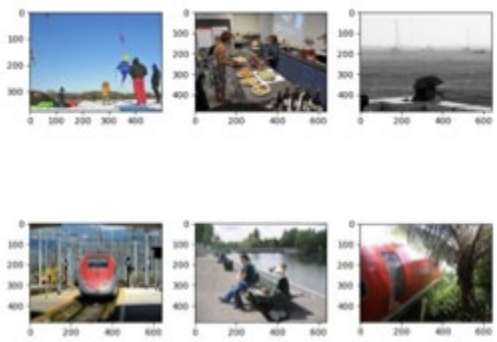
The COCO dataset captions are exclusively in English, we added value to the model by translating the captions to a second language. Initially, we were going to investigate using Neural Machine Translation (NMT) as part of our pipeline but with our already slow performing model, we went with using Google Translation API that takes a single sentence at a time and translates it to any of the hundreds of languages supported. Google Translation API would not make sense on a large-scale implementation like creating a French version COCO dataset. As proof of concept, the translation API worked well.

3 Dataset

The MS-COCO (Common Objects in Context) dataset is a classic deep learning dataset that includes both images and their corresponding captions. The dataset was created by Microsoft. It is used in many competitions, and it is also used as a benchmark dataset for state-of-the-art models. This computer vision dataset can be used for image

captioning, object detection, image segmentation, and key-point detection. The initial release was in 2014 with around 165,000 images split into training (83,000), validation (41,000), and test (41,000) sets. In 2017, additional images and minor changes were made to the dataset.

We selected the 2014 dataset for our model-building purposes. MS-COCO contains 91 object categories with 82 of the categories containing more than 5,000 labeled images. Compared to the popular ImageNet dataset, MS-COCO has fewer categories but more observations per category which makes it better for computer vision models looking to localize objects in 2D and 3D. The total number of images we used for our project was more than 50,000 images. Each image includes 5 captions resulting in a total of over 250,000 captions. The training set includes 40,018 images and 200,090 captions. The test set includes 10,003 images and 50,015 captions. The data contains an image folder and an annotation folder containing “id”, “image_id”, and “caption”. Below is an example of six images and their captions. One thing to note is that each caption begins with <start> and finishes with <end>.



```
<start> a group of people in a snowy field with kites above <end>
<start> A table with lots of plates of food with wines and a slideshow in the background. <end>
<start> A couple with an umbrella overlooking boats in the water. <end>
<start> a red bullet train is coming towards us <end>
<start> People sitting on a bench near the water.  <end>
<start> A red and white plane in heavy forest area. <end>
```

4 Model

The model uses a Convolutional Neural Network as an encoder to extract features from the input images and feed them into a Long-Short-Term-Memory (LSMT) Network to calculate a context vector, which is then fed into the RNN decoder, generating a predicted word according to the input convolutional feature at each time.

The CNN encoder transforms a raw image data into a 14*14*512 feature map, as the feature map goes through the flatten layer, it is then transformed into a sequence of annotation vectors. Finally, by concatenating the annotation vectors, we get the image feature, which is passed to the LSMT as a input.

The LSMT includes a function calculating attention score, a calculation of attention weights as well as a calculation regarding the context vector.

The formula for generating attention score is denoted as

$$score_{t,j} = v_a^T * tanh(U_a * h_{t-1} + W_a * h_j)$$

which contains a hyperbolic tangent transformation of the sum of the multiplication of hidden state ht-1 from the decoder of the previous step with its weight and the output feature map from the CNN encoder with its weight. This score implies how important the jth pixel located in the input image is.

After this, the scores are transferred into an attention weight for the jth pixel at time t with a SoftMax function:

$$\alpha_{ti} = \frac{\exp(e_{ti})}{\sum_{k=1}^{L} \exp(e_{tk})}.$$

Indicating the probability distribution for each feature map, where the sum of attention weight for all pixels within a single prediction step equal to 1.

Finally, the LSTM network calculates a context vector Ct, by simply summing up the multiplication of each feature and its attention weight calculated in the previous step, formulated as

$$c_t = \sum_{j=1}^{T_x} \alpha_{tj} h_j$$

This step combines the extracted features back into an entire image with different weights applied on the blocks.

The last state of the model is an RNN decoder. This decoder is a GRU layer followed by two dense layers. It first takes the embedding dimension through an embedding layer and concatenates it with the context vector calculated by the LSTM to get the input batch size, timesteps and features. Then, this concatenated vector is passed to the GRU to get the output and state. As the output went through two dense layers, it can be finally translated into a predicted word.

5 Experiments

We assume that training with 10000 values versus 6000 from COCO 2014 dataset would improve our model. Using the BLEU scores, we saw minimal improvements with the larger dataset. It does not seem to be worthwhile to increase the dataset because it does not significantly improve BLEU scores. The cost of larger dataset slower performance of training the model.

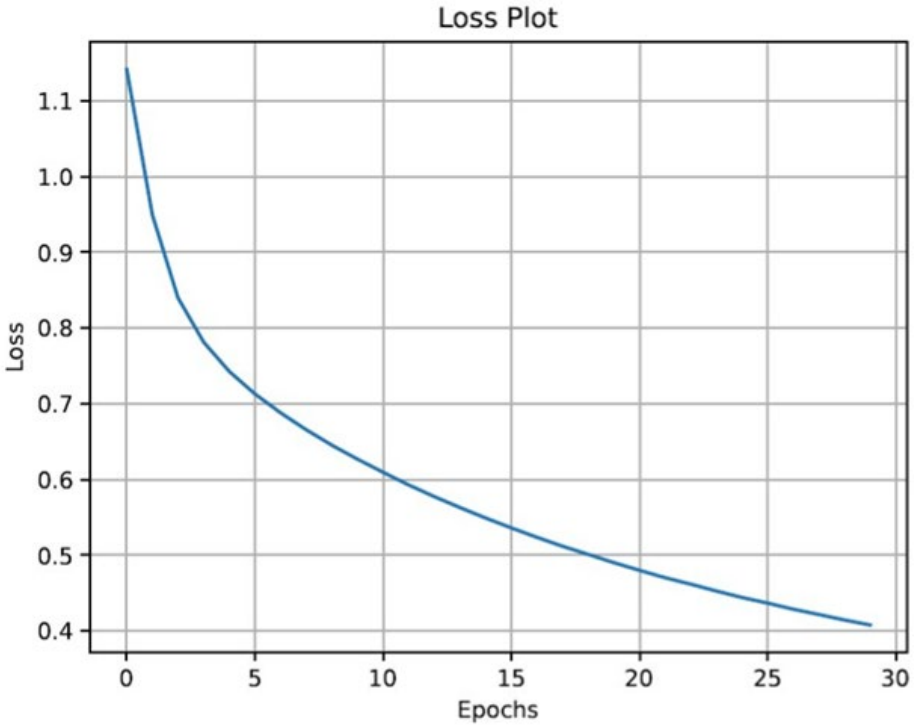
Dataset and Batch Size	BLEU-1	BLEU-2	BLEU-3	BLEU-4
~40,000 Train Images Batch size = 64	44.5	23.3	12.2	6.5
~24,000 Train Images Batch size = 64	44.3	22.0	10.8	5.7
~ 24,000 Train Images Batch size = 128	43.2	21.7	10.6	5.4

We trained the model by varying the number of epochs. The lowest number of epochs being 20 and the highest being 100. By incrementally increasing the number of epochs to train the model, the loss plot shows a steady decrease towards 0. The BLEU score seems to worsen with the increased number of epochs. The best score was a result of training on 10 epochs.

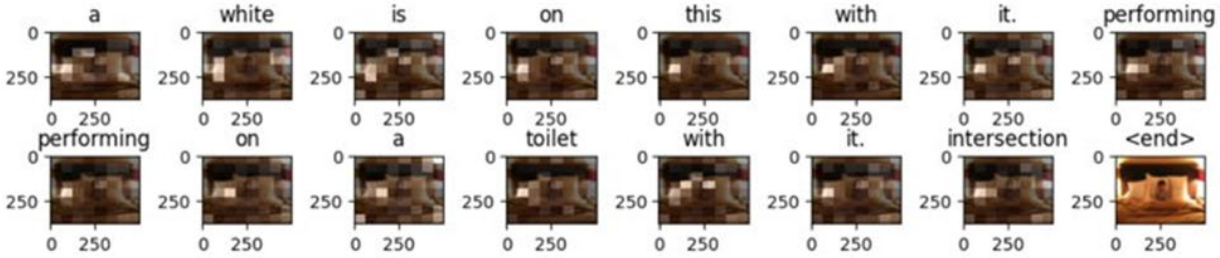
The model's performance is slow to train. On average, an EC2 server on AWS cloud took approximately 160 seconds (about 2 and a half minutes) per epoch to train the model. Assuming that is the average time to train the model, it would take approximately 266.7 hours (about 1 and a half weeks) to train the entire model if we had 100 epochs. The model performed better on a GPU deployed in Google Cloud Platform – approximately 90 seconds (about 1 and a half minutes) per epoch. On a GCP GPU server, it would take us only 150 hours (about 6 and a half days) to train the model if it had 100 epochs.

6 Results

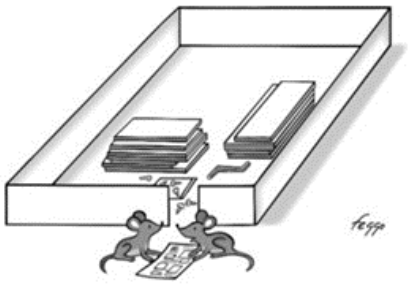
We trained the model on a subset of the COCO 2014 dataset. This resulted in 40,018 training images and 200,090 training captions because there are five captions per image. For the test set, we have 10,003 images and 50,015 captions. We trained using 20, 35, 50, 65, 85, and 100 epochs. Here is the loss plot when we trained using 25 epochs using Adam optimizer. The plot shows a steady decline in loss approaching 0.4.



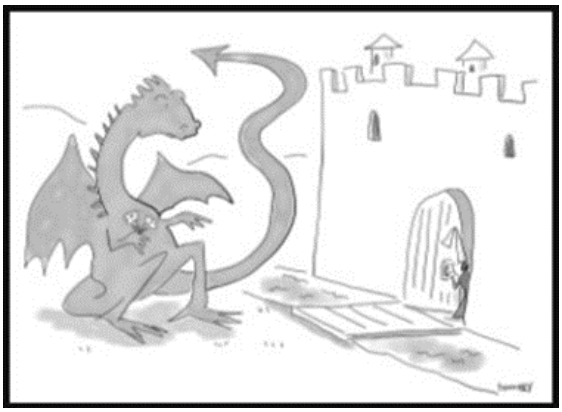
As part of the evaluation, we ran the model against test and created a single output for validation. The model generated the following caption for the image below which does not make sense in terms of the object identification nor as a sentence.



Then we used the trained model to predict captions for New Yorker cartoons. To be fair, we have been training on pictures so to switch to cartoons it has to be acknowledged that cartoons and pictures are not equivalent.



Generated Caption: A table [UNK] a table and a box placed on it. <end>
German Translation: Ein Tisch [UNK] ein Tisch und eine darauf gestellte Kiste. <Ende>
The model correctly identified the rectangular shape as a box but the table detection is not accurate. The two mice were missed by the model entirely. An illustrated mouse is much more an abstraction than a picture of a mouse. The German translation is perfect even if the English sentence does not make sense. Note that [UNK] is for "unknown".



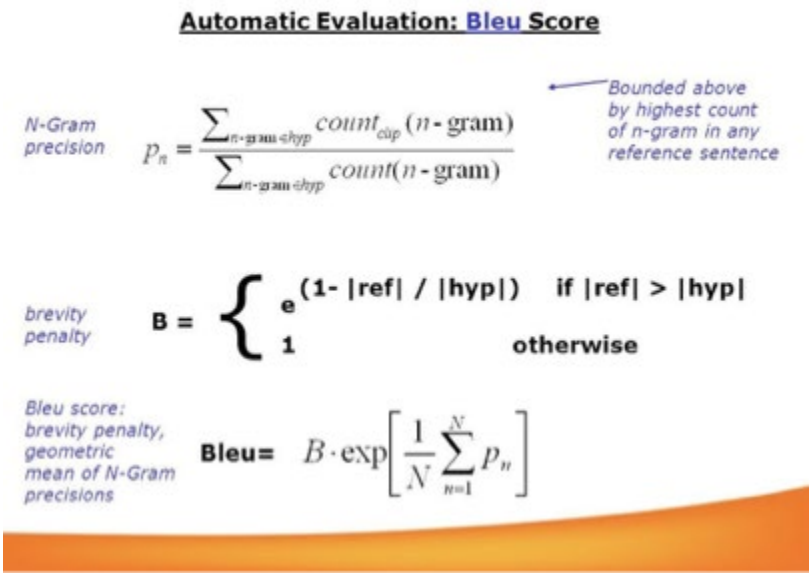
The model identified an animal in the image but incorrectly classified it as a cow. The dragon above also was misclassified as a sheep goat. In terms of the caption the word lounging would make more sense to by the word "lunging" especially following with the word "at." The combination of words sheep goat could be interpreted as the model thinking that a sheep or goat is in the picture. The English caption translated to German is well done except for the sheep goat which is an unknown breed.

BLEU (Bilingual Evaluation Understudy) score is an evaluation metric used to assess the quality of a machine-generated text against reference text (Brownlee, 2019). It was initially developed for translation, but it has been adopted to evaluate other text generating tasks such as image captioning. It ranges from 0 to 1, with 1 being a perfect match of the reference text and 0 being a complete mismatch of the reference. The BLEU score is a simple metric calculation that offers its own strengths and weaknesses. For its strengths, it is inexpensive to calculate, it is widely adopted, and most importantly it correlates highly with human evaluation (Brownlee, 2019). We will be able to understand the flaws of BLEU when we describe how it is calculated.

The BLEU score calculation is simple; count matching n-grams of the candidate text to n-grams in the reference text. Matches are independent of word order. Then, you calculate the precision using the following equation (Khandelwal, 2021).

$$\text{Precision} = \frac{\text{No. of candidate translation words occuring in any reference translation}}{\text{Total no. of words in the candidate translation}}$$

This basic approach falls prey to candidate texts that contain common words like "the" since they will create matches but do not necessarily do a good job of reflecting the reference text (Khandelwal, 2021). Modified n-gram precision clips the counts of n-grams to the number of occurrences in the reference. In addition to clipping, a brevity penalty was also introduced by the authors of BLEU to prevent machine-generated text that is too short compared to the reference text. The brevity penalty and the modified n-gram precision together make up the BLEU score. Cumulative n-gram scores are used in our project; this score is the weighted geometric mean of individual n-gram scores from 1 to n (in our case, n=4). Below is the complete BLEU score formula (Vashee, 2017).



The Show, Attend, and Tell paper includes BLEU scores for several different models. Below, we compare our model to the Soft-attention and Hard-attention models featured in the paper.

	BLEU-1	BLEU-2	BLEU-3	BLEU-4
Our model	46.4	23.9	12.3	6.3
Soft-attention	70.7	49.2	34.4	24.3
Hard-attention	71.8	50.4	35.7	25.0

7 Summary/Conclusions

We built a working image captioning model in TensorFlow using attention mechanism. Using BLEU score, our model performed poorly compared to the results reported in the Show, Attend, and Tell Paper (Xu, 2016). The visual attention mechanism is a good discovery that is worth exploring because it resembles what humans do on an intuitive level. There are a lot of areas of improvement for the model. To improve, we could the model using

the entire COCO2014 dataset instead of only a subset. In addition, we could parameterize the language selection for translating the image caption so that the output can be in any language that the user specifies. We could have used another image dataset like Flicker8k and created a loop to iterate over the images and generate captions. We could have applied the Hard attention Method to the model since the paper implies that the prediction from hard attention has a higher BLEU score. However, it requires sampling and averaging the result with the Monte Carlo simulation. In contrast, the entire Soft Attention model is differentiable, making it computationally easier to compute the gradient. In this case, we prefer using a soft attention model.

8 References

Brownlee, J. (2019, December 18). *A Gentle Introduction to Calculating the BLEU Score for Text in Python*. Machine Learning Mastery. Retrieved April 26, 2022, from https://machinelearningmastery.com/calculate-bleu-score-for-text-python/

COCO - Common Objects in Context. (n.d.). COCODataset. Retrieved April 11, 2022, from https://cocodataset.org/

Image captioning with visual attention | TensorFlow Core. (2022, January 26). TensorFlow. Retrieved April 15, 2022, from https://www.tensorflow.org/tutorials/text/image_captioning

Khandelwal, R. (2021, December 13). *BLEU — Bilingual Evaluation Understudy - Towards Data Science*. Medium. Retrieved April 26, 2022, from https://towardsdatascience.com/bleu-bilingual-evaluation-understudy-2b4eab9bcfd1

Makarov, Artyom. "Image Captioning with Attention: Part 1." *Medium*, Analytics Vidhya, 14 Dec. 2020, https://medium.com/analytics-vidhya/image-captioning-with-attention-part-1-e8a5f783f6d3.

Papineni, Roukos, Ward, Zhu (2022, July 1). BLEU: a Method for Automatic Evaluation of Machine Translation. Dl.acm.org. Retrieved April 26, 2022, from https://aclanthology.org/P02-1040.pdf

Sarkar, Subham. "Image Captioning Using Attention Mechanism." *Medium*, The Startup, 15 June 2021, https://medium.com/swlh/image-captioning-using-attention-mechanism-f3d7fc96eb0e.

Section 508 (Federal Electronic and Information Technology). (n.d.). U.S. Access Board. Retrieved April 25, 2022, from https://www.access-board.gov/law/ra.html

Show, Attend And Tell - Paper Explained. (2021, May 3). YouTube Halfling Wizard. Retrieved April 16, 2022, from https://www.youtube.com/watch?v=y1S3Ri7myMg&t=6s

Vashee, Kirti. (2017, April 11). *The Problem with BLEU and Neural Machine Translation*. Blogger.Com. Retrieved April 26, 2022, from http://kv-emptypages.blogspot.com/2017/04/the-problem-with-bleu-and-neural.html

Xu, Ba, Kiros, K. J. R. (2016, April 19). *Show, Attend and Tell: Neural Image Caption Generation with Visual Attention*. Arxiv.Org. Retrieved April 4, 2022, from https://arxiv.org/pdf/1502.03044.pdf?msclkid=97863852c49311ecb15903abb2f078b2

9 Appendix

```
# -*- coding: utf-8 -*-
"""image_captioning.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/
tutorials/text/image_captioning.ipynb

##### Copyright 2018 The TensorFlow Authors.
"""

#@title Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
```

```
# https://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

"""# Image captioning with visual attention

<table class="tfo-notebook-buttons" align="left">
  <td>
    <a target="_blank"
href="https://www.tensorflow.org/tutorials/text/image_captioning">
    <img src="https://www.tensorflow.org/images/tf_logo_32px.png" />
    View on TensorFlow.org</a>
  </td>
  <td>
    <a target="_blank"
href="https://colab.research.google.com/github/tensorflow/docs/blob/master/si
te/en/tutorials/text/image_captioning.ipynb">
    <img src="https://www.tensorflow.org/images/colab_logo_32px.png" />
    Run in Google Colab</a>
  </td>
  <td>
    <a target="_blank"
href="https://github.com/tensorflow/docs/blob/master/site/en/tutorials/text/i
mage_captioning.ipynb">
    <img src="https://www.tensorflow.org/images/GitHub-Mark-32px.png" />
    View source on GitHub</a>
  </td>
  <td>
    <a
href="https://storage.googleapis.com/tensorflow_docs/docs/site/en/tutorials/t
ext/image_captioning.ipynb"><img
src="https://www.tensorflow.org/images/download_logo_32px.png" />Download
notebook</a>
  </td>
</table>

Given an image like the example below, your goal is to generate a caption
such as "a surfer riding on a wave".

![Man Surfing](https://tensorflow.org/images/surf.jpg)

*[Image
Source](https://commons.wikimedia.org/wiki/Surfing#/media/File:Surfing_in_Haw
aii.jpg); License: Public Domain*

To accomplish this, you'll use an attention-based model, which enables us to
see what parts of the image the model focuses on as it generates a caption.

![Prediction](https://tensorflow.org/images/imcap_prediction.png)

The model architecture is similar to [Show, Attend and Tell: Neural Image
Caption Generation with Visual Attention](https://arxiv.org/abs/1502.03044).
```

```
This notebook is an end-to-end example. When you run the notebook, it
downloads the [MS-COCO](http://cocodataset.org/#home) dataset, preprocesses
and caches a subset of images using Inception V3, trains an encoder-decoder
model, and generates captions on new images using the trained model.

In this example, you will train a model on a relatively small amount of data—
the first 30,000 captions  for about 20,000 images (because there are
multiple captions per image in the dataset).
"""

import tensorflow as tf

# You'll generate plots of attention in order to see which parts of an image
# your model focuses on during captioning

import matplotlib.pyplot as plt
import collections
import random
import numpy as np
import os

os.system("sudo pip install 'tensorflow-text==2.8.*'")
os.system("sudo pip install googletrans==3.1.0a0")
os.system("sudo pip install --user -U nltk")

import time
import json
from PIL import Image
import tensorflow_text
from googletrans import Translator
from nltk.translate import bleu_score as blue


# Google Translator API, specify language output by setting the variable lang
to translate the image captions
# Full list of languages supported available here
https://cloud.google.com/translate/docs/languages?msclkid=c1b4b783c49511ec992
480e415bfc258
translator = Translator()
translated_caption = ""
lang = 'fr'

"""## Download and prepare the MS-COCO dataset

You will use the [MS-COCO dataset](http://cocodataset.org/#home) to train
your model.
The dataset contains over 82,000 images, each of which has at least 5
different caption annotations.
The code below downloads and extracts the dataset automatically.

**Caution: large download ahead**. You'll use the training set, which is a
13GB file.
"""

# Download caption annotation files
annotation_folder = '/annotations/'
annotation_zip = tf.keras.utils.get_file('captions.zip',
```

```
                                          cache_subdir=os.path.abspath('.'),
origin='http://images.cocodataset.org/annotations/annotations_trainval2014.zip',
                                          extract=True)
annotation_file = os.path.dirname(annotation_zip) +
'/annotations/captions_train2014.json'
os.remove(annotation_zip)

# Download image files
image_folder = '/train2014/'
if not os.path.exists(os.path.abspath('.') + image_folder):
    image_zip = tf.keras.utils.get_file('train2014.zip',
                                        cache_subdir=os.path.abspath('.'),
origin='http://images.cocodataset.org/zips/train2014.zip',
                                        extract=True)
    PATH = os.path.dirname(image_zip) + image_folder
    os.remove(image_zip)
else:
    PATH = os.path.abspath('.') + image_folder

"""## Optional: limit the size of the training set
To speed up training for this tutorial, you'll use a subset of 30,000
captions and their corresponding images to train your model. Choosing to use
more data would result in improved captioning quality.
"""

with open(annotation_file, 'r') as f:
    annotations = json.load(f)

# Group all captions together having the same image ID.
image_path_to_caption = collections.defaultdict(list)
for val in annotations['annotations']:
    caption = f"<start> {val['caption']} <end>"
    image_path = PATH + 'COCO_train2014_' + '%012d.jpg' % (val['image_id'])
    image_path_to_caption[image_path].append(caption)

image_paths = list(image_path_to_caption.keys())
random.shuffle(image_paths)

# Select the first 10000 image_paths from the shuffled set.
# Approximately each image id has 5 captions associated with it, so that will
# lead to 30,000 examples.
train_image_paths = image_paths[:10000]
print(len(train_image_paths))

train_captions = []
img_name_vector = []

for image_path in train_image_paths:
    caption_list = image_path_to_caption[image_path]
    train_captions.extend(caption_list)
    img_name_vector.extend([image_path] * len(caption_list))

print(train_captions[0])
Image.open(img_name_vector[0])
```

```
# Added this to plot examples from the dataset
fig = plt.figure(figsize=(8,8))
i = 1
for j in range(6):
  n = np.random.randint(0, 100)
  ax1 = fig.add_subplot(2,3,i)
  plt.imshow(Image.open(img_name_vector[n]))
  print(train_captions[n])
  i += 1
plt.savefig('sample.pdf')
plt.close()

"""## Preprocess the images using InceptionV3
Next, you will use VGG16 (which is pretrained on Imagenet) to classify each
image. You will extract features from the last convolutional layer.

* [Preprocess the images](https://cloud.google.com/tpu/docs/inception-v3-
advanced#preprocessing_stage) using the
[preprocess_input](https://www.tensorflow.org/api_docs/python/tf/keras/applic
ations/inception_v3/preprocess_input) method to normalize the image so that
it contains pixels in the range of -1 to 1, which matches the format of the
images used to train InceptionV3.
"""


def load_image(image_path):
    img = tf.io.read_file(image_path)
    img = tf.io.decode_jpeg(img, channels=3)
    img = tf.keras.layers.Resizing(224, 224)(img)
    img = tf.keras.applications.vgg16.preprocess_input(img)
    return img, image_path


"""## Initialize VGG16 and load the pretrained Imagenet weights

Now you'll create a tf.keras model where the output layer is the last
convolutional layer in the InceptionV3 architecture.
You use the last convolutional layer because you are using attention in this
example. You don't perform this initialization during training because it
could become a bottleneck.

* You forward each image through the network and store the resulting vector
in a dictionary (image_name --> feature_vector).
* After all the images are passed through the network, you save the
dictionary to disk.

"""

image_model = tf.keras.applications.VGG16(include_top=False,
                                          weights='imagenet')
new_input = image_model.input
hidden_layer = image_model.layers[-1].output

image_features_extract_model = tf.keras.Model(new_input, hidden_layer)
image_features_extract_model.summary()
```

```
"""## Caching the features extracted

You will pre-process each image with VGG16 and cache the output to disk.
Caching the output in RAM would be faster but also memory.
At the time of writing, this exceeds the memory limitations of Colab
(currently 12GB of memory).

Performance could be improved with a more sophisticated caching strategy (for
example, by sharding the images to reduce random access disk I/O), but that
would require more code.

"""

# Get unique images
encode_train = sorted(set(img_name_vector))

# Feel free to change batch_size according to your system configuration
image_dataset = tf.data.Dataset.from_tensor_slices(encode_train)
image_dataset = image_dataset.map(
    load_image, num_parallel_calls=tf.data.AUTOTUNE).batch(16)

for img, path in image_dataset:
    batch_features = image_features_extract_model(img)
    batch_features = tf.reshape(batch_features,
                                (batch_features.shape[0], -1,
batch_features.shape[3]))

    for bf, p in zip(batch_features, path):
        path_of_feature = p.numpy().decode("utf-8")
        np.save(path_of_feature, bf.numpy())

"""## Preprocess and tokenize the captions

You will transform the text captions into integer sequences using the
[TextVectorization](https://www.tensorflow.org/api_docs/python/tf/keras/layer
s/TextVectorization) layer, with the following steps:

* Use
[adapt](https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVector
ization#adapt) to iterate over all captions, split the captions into words,
and compute a vocabulary of the top 5,000 words (to save memory).
* Tokenize all captions by mapping each word to it's index in the vocabulary.
All output sequences will be padded to length 50.
* Create word-to-index and index-to-word mappings to display results.
"""

caption_dataset = tf.data.Dataset.from_tensor_slices(train_captions)


# We will override the default standardization of TextVectorization to
preserve
# "<>" characters, so we preserve the tokens for the <start> and <end>.
def standardize(inputs):
    # inputs = tf.strings.lower(inputs, encoding='utf-8')
    return tf.strings.regex_replace(inputs,
                                    r"!\"#$%&\(\)\*\+.,-/:;=?@\[\\\]^_`{|}~",
"")
```

```
# Max word count for a caption.
max_length = 50
# Use the top 5000 words for a vocabulary.
vocabulary_size = 5000
tokenizer = tf.keras.layers.TextVectorization(
    max_tokens=vocabulary_size,
    standardize=standardize,
    output_sequence_length=max_length)
# Learn the vocabulary from the caption data.
tokenizer.adapt(caption_dataset)

# Create the tokenized vectors
cap_vector = caption_dataset.map(lambda x: tokenizer(x))

# Create mappings for words to indices and indicies to words.
word_to_index = tf.keras.layers.StringLookup(
    mask_token="",
    vocabulary=tokenizer.get_vocabulary())
index_to_word = tf.keras.layers.StringLookup(
    mask_token="",
    vocabulary=tokenizer.get_vocabulary(),
    invert=True)

"""## Split the data into training and testing"""

img_to_cap_vector = collections.defaultdict(list)
for img, cap in zip(img_name_vector, cap_vector):
    img_to_cap_vector[img].append(cap)

# Changed to 80-20 Train - Test Split. Create training and validation sets
using an 70-30 split randomly.
img_keys = list(img_to_cap_vector.keys())
random.shuffle(img_keys)

slice_index = int(len(img_keys) * 0.8)
img_name_train_keys, img_name_val_keys = img_keys[:slice_index],
img_keys[slice_index:]

img_name_train = []
cap_train = []
for imgt in img_name_train_keys:
    capt_len = len(img_to_cap_vector[imgt])
    img_name_train.extend([imgt] * capt_len)
    cap_train.extend(img_to_cap_vector[imgt])

img_name_val = []
cap_val = []
for imgv in img_name_val_keys:
    capv_len = len(img_to_cap_vector[imgv])
    img_name_val.extend([imgv] * capv_len)
    cap_val.extend(img_to_cap_vector[imgv])

# Added print statement to get lengths
print(f'train images = {len(img_name_train)}, test images
{len(img_name_val)}')
```

```
"""## Create a tf.data dataset for training

Your images and captions are ready! Next, let's create a `tf.data` dataset to
use for training your model.
"""

# Feel free to change these parameters according to your system's
configuration

BATCH_SIZE = 64
BUFFER_SIZE = 1000
embedding_dim = 256
units = 512
num_steps = len(img_name_train) // BATCH_SIZE
# These two variables represent that vector shape
features_shape = 512
attention_features_shape = 49


# Load the numpy files
def map_func(img_name, cap):
    img_tensor = np.load(img_name.decode('utf-8') + '.npy')
    return img_tensor, cap


dataset = tf.data.Dataset.from_tensor_slices((img_name_train, cap_train))

# Use map to load the numpy files in parallel
dataset = dataset.map(lambda item1, item2: tf.numpy_function(
    map_func, [item1, item2], [tf.float32, tf.int64]),
                      num_parallel_calls=tf.data.AUTOTUNE)

# Shuffle and batch
dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
dataset = dataset.prefetch(buffer_size=tf.data.AUTOTUNE)

"""## Model

Fun fact: the decoder below is identical to the one in the example for
[Neural Machine Translation with
Attention](https://www.tensorflow.org/text/tutorials/nmt_with_attention).

The model architecture is inspired by the [Show, Attend and
Tell](https://arxiv.org/pdf/1502.03044.pdf) paper.

* This vector is then passed through the CNN Encoder (which consists of a
single Fully connected layer).
* The RNN (here GRU) attends over the image to predict the next word.
"""


class BahdanauAttention(tf.keras.Model):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
```

```
        self.V = tf.keras.layers.Dense(1)

    def call(self, features, hidden):
        # features(CNN_encoder output) shape == (batch_size, 64,
embedding_dim)

        # hidden shape == (batch_size, hidden_size)
        # hidden_with_time_axis shape == (batch_size, 1, hidden_size)
        hidden_with_time_axis = tf.expand_dims(hidden, 1)

        # attention_hidden_layer shape == (batch_size, 64, units)
        attention_hidden_layer = (tf.nn.tanh(self.W1(features) +
                                             self.W2(hidden_with_time_axis)))

        # score shape == (batch_size, 64, 1)
        # This gives you an unnormalized score for each image feature.
        score = self.V(attention_hidden_layer)

        # attention_weights shape == (batch_size, 64, 1)
        attention_weights = tf.nn.softmax(score, axis=1)

        # context_vector shape after sum == (batch_size, hidden_size)
        context_vector = attention_weights * features
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights


class CNN_Encoder(tf.keras.Model):
    # Since you have already extracted the features and dumped it
    # This encoder passes those features through a Fully connected layer
    def __init__(self, embedding_dim):
        super(CNN_Encoder, self).__init__()
        # shape after fc == (batch_size, 49, embedding_dim)
        self.fc = tf.keras.layers.Dense(embedding_dim)

    def call(self, x):
        x = self.fc(x)
        x = tf.nn.relu(x)
        return x


class RNN_Decoder(tf.keras.Model):
    def __init__(self, embedding_dim, units, vocab_size):
        super(RNN_Decoder, self).__init__()
        self.units = units

        self.embedding = tf.keras.layers.Embedding(vocab_size, embedding_dim)
        self.gru = tf.keras.layers.GRU(self.units,
                                       return_sequences=True,
                                       return_state=True,

recurrent_initializer='glorot_uniform')
        self.fc1 = tf.keras.layers.Dense(self.units)
        self.fc2 = tf.keras.layers.Dense(vocab_size)

        self.attention = BahdanauAttention(self.units)
```

```
    def call(self, x, features, hidden):
        # defining attention as a separate model
        context_vector, attention_weights = self.attention(features, hidden)

        # x shape after passing through embedding == (batch_size, 1,
embedding_dim)
        x = self.embedding(x)

        # x shape after concatenation == (batch_size, 1, embedding_dim +
hidden_size)
        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        # passing the concatenated vector to the GRU
        output, state = self.gru(x)

        # shape == (batch_size, max_length, hidden_size)
        x = self.fc1(output)

        # x shape == (batch_size * max_length, hidden_size)
        x = tf.reshape(x, (-1, x.shape[2]))

        # output shape == (batch_size * max_length, vocab)
        x = self.fc2(x)

        return x, state, attention_weights

    def reset_state(self, batch_size):
        return tf.zeros((batch_size, self.units))


encoder = CNN_Encoder(embedding_dim)
decoder = RNN_Decoder(embedding_dim, units, tokenizer.vocabulary_size())

optimizer = tf.keras.optimizers.Adam()
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')


def loss_function(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)


"""## Checkpoint"""

checkpoint_path = "./checkpoints/train"
ckpt = tf.train.Checkpoint(encoder=encoder,
                           decoder=decoder,
                           optimizer=optimizer)
ckpt_manager = tf.train.CheckpointManager(ckpt, checkpoint_path,
max_to_keep=5)
```

```
start_epoch = 0
if ckpt_manager.latest_checkpoint:
    start_epoch = int(ckpt_manager.latest_checkpoint.split('-')[-1])
    # restoring the latest checkpoint in checkpoint_path
    ckpt.restore(ckpt_manager.latest_checkpoint)

"""## Training

* You extract the features stored in the respective `.npy` files and then
pass those features through the encoder.
* The encoder output, hidden state(initialized to 0) and the decoder input
(which is the start token) is passed to the decoder.
* The decoder returns the predictions and the decoder hidden state.
* The decoder hidden state is then passed back into the model and the
predictions are used to calculate the loss.
* Use teacher forcing to decide the next input to the decoder.
* Teacher forcing is the technique where the target word is passed as the
next input to the decoder.
* The final step is to calculate the gradients and apply it to the optimizer
and backpropagate.

"""

# adding this in a separate cell because if you run the training cell
# many times, the loss_plot array will be reset
loss_plot = []


@tf.function
def train_step(img_tensor, target):
    loss = 0

    # initializing the hidden state for each batch
    # because the captions are not related from image to image
    hidden = decoder.reset_state(batch_size=target.shape[0])

    dec_input = tf.expand_dims([word_to_index('<start>')] * target.shape[0],
1)

    with tf.GradientTape() as tape:
        features = encoder(img_tensor)

        for i in range(1, target.shape[1]):
            # passing the features through the decoder
            predictions, hidden, _ = decoder(dec_input, features, hidden)

            loss += loss_function(target[:, i], predictions)

            # using teacher forcing
            dec_input = tf.expand_dims(target[:, i], 1)

    total_loss = (loss / int(target.shape[1]))

    trainable_variables = encoder.trainable_variables +
decoder.trainable_variables
```

```
    gradients = tape.gradient(loss, trainable_variables)

    optimizer.apply_gradients(zip(gradients, trainable_variables))

    return loss, total_loss


EPOCHS = 50

for epoch in range(start_epoch, EPOCHS):
    start = time.time()
    total_loss = 0

    for (batch, (img_tensor, target)) in enumerate(dataset):
        batch_loss, t_loss = train_step(img_tensor, target)
        total_loss += t_loss

        if batch % 100 == 0:
            average_batch_loss = batch_loss.numpy() / int(target.shape[1])
            print(f'Epoch {epoch + 1} Batch {batch} Loss
{average_batch_loss:.4f}')
    # storing the epoch end loss value to plot later
    loss_plot.append(total_loss / num_steps)

    if epoch % 5 == 0:
        ckpt_manager.save()

    print(f'Epoch {epoch + 1} Loss {total_loss / num_steps:.6f}')
    print(f'Time taken for 1 epoch {time.time() - start:.2f} sec\n')

plt.grid()
plt.plot(loss_plot)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss Plot')
plt.savefig('loss_plot.pdf')
plt.close()

"""## Caption!

* The evaluate function is similar to the training loop, except you don't use
teacher forcing here. The input to the decoder at each time step is its
previous predictions along with the hidden state and the encoder output.
* Stop predicting when the model predicts the end token.
* And store the attention weights for every time step.
"""


def evaluate(image):
    attention_plot = np.zeros((max_length, attention_features_shape))

    hidden = decoder.reset_state(batch_size=1)

    temp_input = tf.expand_dims(load_image(image)[0], 0)
    img_tensor_val = image_features_extract_model(temp_input)
    img_tensor_val = tf.reshape(img_tensor_val, (img_tensor_val.shape[0],
                                                 -1,
```

```
                                                    img_tensor_val.shape[3]))

    features = encoder(img_tensor_val)

    dec_input = tf.expand_dims([word_to_index('<start>')], 0)
    result = []

    for i in range(max_length):
        predictions, hidden, attention_weights = decoder(dec_input,
                                                         features,
                                                         hidden)

        attention_plot[i] = tf.reshape(attention_weights, (-1,)).numpy()

        predicted_id = tf.random.categorical(predictions, 1)[0][0].numpy()
        predicted_word =
tf.compat.as_text(index_to_word(predicted_id).numpy())
        result.append(predicted_word)

        if predicted_word == '<end>':
            return result, attention_plot

        dec_input = tf.expand_dims([predicted_id], 0)

    attention_plot = attention_plot[:len(result), :]
    return result, attention_plot


def plot_attention(image, result, attention_plot):
    temp_image = np.array(Image.open(image))

    fig = plt.figure(figsize=(10, 10))

    len_result = len(result)
    for i in range(len_result):
        temp_att = np.resize(attention_plot[i], (8, 8))
        grid_size = max(int(np.ceil(len_result / 2)), 2)
        ax = fig.add_subplot(grid_size, grid_size, i + 1)
        ax.set_title(result[i])
        img = ax.imshow(temp_image)
        ax.imshow(temp_att, cmap='gray', alpha=0.6, extent=img.get_extent())

    # plt.tight_layout()
    plt.savefig('figure.pdf')
    plt.close()

# captions on the validation set
rid = np.random.randint(0, len(img_name_val))
image = img_name_val[rid]
real_caption = ' '.join([tf.compat.as_text(index_to_word(i).numpy())
                         for i in cap_val[rid] if i not in [0]])
result, attention_plot = evaluate(image)

print('Real Caption:', real_caption)
print('Prediction Caption:', ' '.join(result))
sentence = ' '.join(result)
translated_captions = translator.translate(sentence, dest= lang)
```

```
print(sentence, ' -> ', translated_captions.text)
plot_attention(image, result, attention_plot)

ref_dict = {}

for i in img_name_val:
  ref_dict[i] = []

# remove <>
references = [[] for i in range(len(img_name_val))]
hypotheses = []
for j in range(len(img_name_val)):
    image = img_name_val[j]

    ref_dict[image]
    real_caption = ' '.join([tf.compat.as_text(index_to_word(i).numpy())
                             for i in cap_val[j] if i not in [0]])
    real_caption_split = real_caption.split()
    ref_dict[image].append(real_caption_split)

for image in ref_dict.keys():
  result, attention_plot = evaluate(image)
  hypotheses.append(result)

references = list(map(list, (ref_dict.values())))
len(references)

len(hypotheses)

blue1 = blue.corpus_bleu(references, hypotheses, weights=(1,))
blue2 = blue.corpus_bleu(references, hypotheses, weights=(.5,.5))
blue3 = blue.corpus_bleu(references, hypotheses, weights=(1/3, 1/3, 1/3,))
blue4 = blue.corpus_bleu(references, hypotheses)

print(f'blue1 (weights = 1) = {blue1}')
print(f'blue2 (weights = 0.5) = {blue2}')
print(f'blue3 (weights = 0.333) = {blue3}')
print(f'blue4 = {blue4}')

"""## Try it on your own images

For fun, below you're provided a method you can use to caption your own
images with the model you've just trained. Keep in mind, it was trained on a
relatively small amount of data, and your images may be different from the
training data (so be prepared for weird results!)
Source for New Yorker cartoon images: https://github.com/nextml/caption-
contest-data/tree/gh-pages/cartoons
"""
image_url0 = 'https://raw.githubusercontent.com/nextml/caption-contest-
data/gh-pages/cartoons/667.jpg'
image_url1 = 'https://raw.githubusercontent.com/nextml/caption-contest-
data/gh-pages/cartoons/671.jpg'
image_url2 = 'https://raw.githubusercontent.com/nextml/caption-contest-
data/gh-pages/cartoons/670.jpg'
image_list = [image_url0, image_url1, image_url2]

for image_url in image_list:
```

```
    image_extension = image_url[-4:]
    image_path = tf.keras.utils.get_file('image'+image_extension,
origin=image_url)
    result, attention_plot = evaluate(image_path)
    plot_attention(image_path, result, attention_plot)
    # opening the image
    Image.open(image_path)
```