Image Captioning using Deep Learning

Introduction

Caption generation is an artificial intelligence problem where a textual description must be generated for a given photograph. It requires both methods from computer vision and natural language processing. Computer vision is used to understand the content of the image while natural language processing is used to turn the understanding of the image into words in the right order.

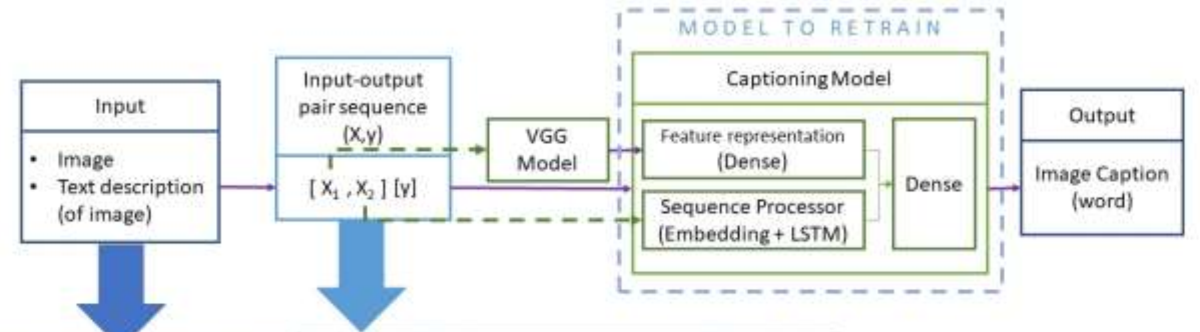
In this lab, you will be guided to create a deep learning model for photo captioning using pre-trained model and smaller dataset for faster training. In the last part of this guide, the homework is explained.

Objective

The students are expected to learn how to prepare photo and text data for training a deep learning model, how to retrain pre-trained model of caption generation model, how to evaluate a trained caption captioning model and use it to caption entirely new photographs.

Background

The task of image captioning can be divided into two modules: (1) image-based model, which extracts the features of the image, and (2) captioning model, which handles the text description of each image, merges with the image feature extracted and makes prediction over the entire output vocabulary for the next word in the sequence which describe the input image, as shown in the figure below.



X_1	X_2 (text sequence)	y (word)
Photo1	startseq,	little
Photo1	startseq, little,	girl
Photo1	startseq, little, girl,	running
Photo1	startseq, little, girl, running,	endseq
Photo1	startseq,	girl
Photo1	startseq, girl,	in
Photo1	startseq, girl, in,	field
Photo1	startseq, girl, in, field,	endseq

The input image will be fed to the VGG model (without the output layer) and use the extracted features predicted as input to a Dense layer to produce a 256-element representation of the photo. Whereas the text descriptions which are split into words and encoded as integers are fed to the sequence processor which compose of Embedding layer that handles text input and then followed by LSTM layer with 256 memory units. Both the input models (Feature Extractor and Sequence Processor) produce a 256-element vector which are then merge using an add operation and fed to a 256-neuron Dense layer. Then to a final output Dense layer that makes the *softmax* prediction over the entire output vocabulary for the next word in the sequence.

However, creating this model from scratch takes time in training, thus, in this Lab exercise, we will use pre-trained model and retrain it on smaller dataset. By retraining means, we use pre-trained model as our starting point. To do this, we will follow the steps below:

1. Caption testing on pre-trained model

2. Retraining pre-trained model using smaller dataset (transfer learning) by following the steps below:

    - Prepare photo dataset by extracting its features using the VGG16 model
    - Prepare the text dataset by cleaning each text descriptions of each photo and by mapping the words to unique integer values using the *tokenizer*
    - Transform data into input-output pair for training the model
    - Load the pre-trained model and fit on the prepared data.

Implementation

Before training, we can examine the pre-trained model by testing it to caption a sample photo.

Caption testing on pre-trained model

In this section, we can test the pre-trained model by following the steps below.

Step 1: Download the model. You can download the model by running the script below. Doing so, it will download and unzip a compressed file (.zip file) which contains the pre-trained model and the dataset we will be using on this Lab.

```
# to download the dataset and pretrained model
!gdown --id 1D79dNcLXu6mV1Uueo7EJJXtQm2AO9yUb
# to unzip the file
!unzip dataset.zip
```

Step 2: Import all necessary libraries for data and model loading.

```
from pickle import load
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from keras.applications.vgg16 import VGG16
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.models import load_model

import IPython.display as display
from PIL import Image
```

Step 3: Load the pre-trained model, the image to caption and the tokenizer. The tokenizer is use to encode generated words for the model while generating the sequence. This tokenizer is fitted on the pre-trained model which contains the dictionary of descriptions of the images used.

```
# load the tokenizer
tokenizer = load(open('/content/tokenizer.pkl', 'rb'))
# pre-define the max sequence length (from pre-training)
max_length = 34
# load the pre-trained model
model = load_model('/content/model_4.h5')
# load and prepare the photograph
photo = extract_features('/content/example.jpg')
# display the image
display.display(Image.open('/content/example.jpg'))
# generate description
description = generate_desc(model, tokenizer, photo, max_length)
print(description)
```

Running above script will render the following output. You can access the full code in here.

startseq man in red shirt is climbing rock endseq

Training using pre-trained model and smaller dataset

Before training, let's prepare the photo dataset and text data first.

Step 1: Extract features on photo images. We will use the VGG model to extract features on our images. We can pre-compute the "photo features" of our training and testing dataset and save to a file. Then, we can load these features later and feed into the model.

```
# extract features from each photo in the directory
def extract_features(directory, dataset):
  # load the model
  model = VGG16()
  # re-structure the model
  model = Model(inputs=model.inputs, outputs=model.layers[-2].output)
  # summarize
  print(model.summary())
  # extract features from each photo
  features = dict()
  for name in dataset:
    # load an image from file
    filename = directory + '/' + name + '.jpg'
    image = load_img(filename, target_size=(224,224))
    # convert the image pixels to a numpy array
    image = img_to_array(image)
    # reshape data for the model
    image = image.reshape((1, image.shape[0], image.shape[1], image.shape[2]))
    # prepare the image for the VGG model
    image = preprocess_input(image)
    # get features
    feature = model.predict(image, verbose=0)
    # get image id
    image_id = name.split('.')[0]
    # store feature
    features[image_id] = feature
    print('>%s' % name)
  return features
```

```
# extract features from training images
filename = '/content/Flickr8k_text/Flickr_100.trainImages.txt'
train = load_set(filename)
train_features = extract_features('/content/Flickr8k_Dataset', train)
print('Extracted Features of training data: %d' % len(train_features))
# save to file
dump(train_features, open('/content/features_train.pkl', 'wb'))

# extract features from testing images
filename = '/content/Flickr8k_text/Flickr_50.testImages.txt'
test = load_set(filename)
test_features = extract_features('/content/Flickr8k_Dataset', test)
print('Extracted Features of test data: %d' % len(test_features))
# save to file
dump(test_features, open('/content/features_test.pkl', 'wb'))
```

Step 2: Prepare text data. The dataset contains multiple descriptions for each photograph and each photo has a unique identifier. This identifier is used on the photo filename and in the text file of descriptions.

Step 2.1: Load the file containing all of the descriptions.

```python
import string

# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, 'r')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

```

```python
filename = 'dataset/Flickr8k_text/Flickr8k.token.txt'
# load descriptions
doc = load_doc(filename)
```

Step 2.2: Map the descriptions to corresponding photo, each photo identifier maps to a list of one or more textual descriptions.

```python
# extract descriptions for images
def load_descriptions(doc):
    mapping = dict()
    # process lines
    for line in doc.split('\n'):
        # split line by white space
        tokens = line.split()
        if len(line) < 2:
            continue
        # take the first token as the image id, the rest as the description
        image_id, image_desc = tokens[0], tokens[1:]
        # remove filename from image id
        image_id = image_id.split('.')[0]
        # convert description tokens back to string
        image_desc = ' '.join(image_desc)
        # create the list if needed:
        if image_id not in mapping:
            mapping[image_id] = list()
        # store description
        mapping[image_id].append(image_desc)
    return mapping

```

```python
# parse descriptions
descriptions = load_descriptions(doc)
print('Loaded: %d ' % len(descriptions))
```

Step 2.3: The text of the descriptions requires some minimal cleaning to reduce the size of the vocabulary of words. The following cleaning will be applied:

- Convert all words to lowercase
- Remove all punctuations
- Remove all words that are one character or less in length (e.g. ‘a’)
- Remove all words that contain numbers

```python
def clean_descriptions(descriptions):
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for key, desc_list in descriptions.items():
        for i in range(len(desc_list)):
            desc = desc_list[i]
            # tokenize
            desc = desc.split()
            # convert to lower case
            desc = [word.lower() for word in desc]
            # remove punctuation from each token
            desc = [w.translate(table) for w in desc]
            # remove hanging 's' and 'a'
            desc = [word for word in desc if len(word)>1]
            # remove tokens with numbers
            desc = [word for word in desc if word.isalpha()]
            # store as string
            desc_list[i] = ' '.join(desc)
```

```python
# clean descriptions
clean_descriptions(descriptions)
```

<u>Step 2.4</u>: Summarize the size of the vocabulary. Ideally, we want a vocabulary that is both expressive and as small as possible. A smaller vocabulary will result in a smaller that will train faster.

```python
# convert the loaded descriptions into a vocabulary of words
def to_vocabulary(descriptions):
    # build a list of all description strings
    all_desc = set()
    for key in descriptions.keys():
        [all_desc.update(d.split()) for d in descriptions[key]]
    return all_desc
```

```python
# summarize vocabulary
vocabulary = to_vocabulary(descriptions)
print('Vocabulary size: %d' % len(vocabulary))
```

<u>Step 2.5</u>: Finally, save the dictionary of image identifiers and descriptions to a new file named "*descriptions.txt*", with one image identifier and description per line.

```python
# save descriptions to file with one per line
def save_descriptions(descriptions, filename):
    lines = list()
    for key, desc_list in descriptions.items():
        for desc in desc_list:
            lines.append(key + ' ' + desc)
    data = '\n'.join(lines)
    file = open(filename, 'w')
    file.write(data)
    file.close()
```

```python
# save to file
save_descriptions(descriptions, 'descriptions.txt')
```

Running the scripts above, will display the following output and save a file which look like as below. (The order of descriptions in your file may vary)

```
Loaded: 8092
Vocabulary size: 8763
```

```
1000268201_693b08cb0e child in pink dress is climbing up set of stairs in an entry way
1000268201_693b08cb0e girl going into wooden building
1000268201_693b08cb0e little girl climbing into wooden playhouse
1000268201_693b08cb0e little girl climbing the stairs to her playhouse
1000268201_693b08cb0e little girl in pink dress going into wooden cabin
1001773457_577c3a7d70 black dog and spotted dog are fighting
1001773457_577c3a7d70 black dog and tricolored dog playing with each other on the road
1001773457_577c3a7d70 black dog and white dog with brown spots are staring at each other in the street
1001773457_577c3a7d70 two dogs of different breeds looking at each other on the road
```

We can now retrain the model using the prepared data.

Step 1: Load the training data. The train dataset has been predefined in the *Flickr_100.trainImages.txt* that contains lists of photo file names.

```python
from numpy import array
from pickle import load
from numpy import argmax
from keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.applications.vgg16 import preprocess_input
from keras.models import Model
from keras.models import load_model

# load doc into memory
def load_doc(filename):
  # open the file as read only
  file = open(filename, 'r')
  # read all text
  text = file.read()
  # close the file
  file.close()
  return text

# load a pre-defined list of photo identifiers
def load_set(filename):
  doc = load_doc(filename)
  dataset = list()
  # process line by line
  for line in doc.split('\n'):
      # skip empty lines
      if len(line) < 1:
          continue
      # get the image identifier
      identifier = line.split('.')[0]
      dataset.append(identifier)
  return set(dataset)
```

```python
# load training dataset (100)
filename = '/content/Flickr8k_text/Flickr_100.trainImages.txt'
train = load_set(filename)
print('Dataset: %d' % len(train))
```

Step 2: Load the prepared descriptions, saved as "*descriptions.txt*", which contains a set of identifiers and text descriptions. The model will generate a caption given a photo, and the caption will be generated one word at a time. The sequence of previously generated will be provided as input, therefore, we will need of “*first word*” to kick-off the generation process and a “*last word”* to signal the end of the caption. With this, we will use the strings “*startseq”* and “*endseq”* (line 54). These tokens are added to the loaded descriptions as they are loaded. It is important to add these tokens first before encoding the text so that it will be encoded correctly.

```
38 # load clean descriptions into memory
39 def load_clean_descriptions(filename, dataset):
40   # load document
41   doc = load_doc(filename)
42   descriptions = dict()
43   for line in doc.split('\n'):
44       # split line by white space
45       tokens = line.split()
46       # split id from description
47       image_id, image_desc = tokens[0], tokens[1:]
48       # skip images not in the set
49       if image_id in dataset:
50           # create list
51           if image_id not in descriptions:
52               descriptions[image_id] = list()
53           # wrap description in tokens
54           desc = 'startseq ' + ' '.join(image_desc) + ' endseq'
55           # store
56           descriptions[image_id].append(desc)
57   return descriptions
```

```
# descriptions
train_descriptions = load_clean_descriptions('/content/descriptions.txt', train)
print('Descriptions: train=%d' % len(train_descriptions))
```

Step 3: Load the prepared photos.

```
# load photo features
def load_photo_features(filename, dataset):
  # load all features
  all_features = load(open(filename, 'rb'))
  # filter features
  features = {k: all_features[k] for k in dataset}
  return features
```

```
# photo features
train_features = load_photo_features('/content/features_train.pkl', train)
print('Photos: train=%d' % len(train_features))
```

Step 4: Load the tokenizer.

```
# load the tokenizer
tokenizer = load(open('/content/tokenizer.pkl', 'rb'))
vocab_size = len(tokenizer.word_index) + 1
print('Vocabulary Size: %d' % vocab_size)
# pre-define the max sequence length (from pre-training)
max_length = 34
print('Description Length: %d' % max_length)
```

Step 5: Transform data to input-output pairs for training the model. There are two input arrays to the model: (1) one for the photos and (2) one for the encoded text. And there is one output for the model which is the encoded next word in the text sequence. The input text is encoded as integers, which will be fed to a word embedding layer. The photo features will be fed directly to other part of the model. The model will output a prediction, which will be a probability distribution of all words in the vocabulary. The output data will therefore be a one-hot encoded version of each word, representing an idealized probability distribution with 0 values at all word positions except the actual word position, which has a value of 1.

```
# create sequences of images, input sequences and output word for an image
def create_sequences(tokenizer, max_length, desc_list, photo, vocab_size):
  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(photo)
          X2.append(in_seq)
          y.append(out_seq)
  return array(X1), array(X2), array(y)
```

Step 6: Define the model. Since we're retraining from pre-trained model, we can just load the pre-trained model.

```
# define captioning model
def define_model():
  model = load_model('/content/model_4.h5')
  # summarize model
  print(model.summary())
  return model
```

```
# define the model
model = define_model()
```

Step 7: Fit the model and the save model after each epoch.

```
# intended to be used in a call to model.fit_generator()
def data_generator(descriptions, photos, tokenizer, max_length, vocab_size):
  # loop forever over images
  while 1:
      for key, desc_list in descriptions.items():
          # retrieve the photo feature
          photo = photos[key][0]
          in_img, in_seq, out_word = create_sequences(tokenizer, max_length, desc_list, photo, vocab_size)
          yield [in_img, in_seq], out_word
```

```python
# train the model, run epochs and save model after each epoch
epochs = 5
steps = len(train_descriptions)
for i in range(epochs):
  # create data generator
  generator = data_generator(train_descriptions, train_features, tokenizer, max_length, vocab_size)
  # fit for one epoch
  model.fit_generator(generator, epochs=1, steps_per_epoch=steps, verbose=1)
  # save model
  model.save('/content/lab_model_' + str(i) + '.h5')
```

After running the above scripts, it will train model, save the models and display the following.

```
Dataset: 100
Descriptions: train=100
Photos: train=100
Vocabulary Size: 7579
Description Length: 34
Model: "model_7"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 input_17 (InputLayer)          [(None, 34)]         0           []

 input_16 (InputLayer)          [(None, 4096)]       0           []

 embedding_7 (Embedding)        (None, 34, 256)      1940224     ['input_17[0][0]']

 dropout_14 (Dropout)           (None, 4096)         0           ['input_16[0][0]']

 dropout_15 (Dropout)           (None, 34, 256)      0           ['embedding_7[0][0]']

 dense_20 (Dense)               (None, 256)          1048832     ['dropout_14[0][0]']

 lstm_7 (LSTM)                  (None, 256)          525312      ['dropout_15[0][0]']

 add_7 (Add)                    (None, 256)          0           ['dense_20[0][0]',
                                                                  'lstm_7[0][0]']

 dense_21 (Dense)               (None, 256)          65792       ['add_7[0][0]']

 dense_22 (Dense)               (None, 7579)         1947803     ['dense_21[0][0]']

==================================================================================================
Total params: 5,527,963
Trainable params: 5,527,963
Non-trainable params: 0
__________________________________________________________________________________________________
None
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:136: UserWarning: `Model.fit_generator`
100/100 [==============================] - 31s 281ms/step - loss: 3.7674
/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410: CustomMaskWarning: Custom ma
  layer_config = serialize_layer_fn(layer)
100/100 [==============================] - 28s 282ms/step - loss: 3.3883
100/100 [==============================] - 29s 289ms/step - loss: 3.1597
100/100 [==============================] - 30s 301ms/step - loss: 2.9971
100/100 [==============================] - 28s 284ms/step - loss: 2.8479
```

Once the model is fit, it can be evaluated on the holdout test dataset. We will evaluate a model by generating descriptions for all photos in the test dataset and evaluating those predictions with a standard cost function.

<u>Step 1</u>: Load the test dataset, its features and descriptions, and the tokenizer.

```python
# load test set
filename = '/content/Flickr8k_text/Flickr_50.testImages.txt'
test = load_set(filename)
print('Dataset: %d' % len(test))
# descriptions
test_descriptions = load_clean_descriptions('/content/descriptions.txt', test)
print('Descriptions: test=%d' % len(test_descriptions))
# photo features
test_features = load_photo_features('/content/features_test.pkl', test)
print('Photos: test=%d' % len(test_features))
# load the tokenizer
tokenizer = load(open('/content/tokenizer.pkl', 'rb'))
# pre-define the max sequence length (from pre-training)
max_length = 34
```

<u>Step 2</u>: Define a function that can generate a description for a photo using the trained model. This involves passing in the start description token “*startseq”*, generating one word, then calling the model recursively with generated words as input until the end of sequence token is reach “*endseq*” or the maximum length is reached.

```
# map an integer to a word
def word_for_id(integer, tokenizer):
  for word, index in tokenizer.word_index.items():
      if index == integer:
          return word
  return None

# generate a description for an image
def generate_desc(model, tokenizer, photo, max_length):
  # seed the generation process
  in_text = 'startseq'
  # iterate over the whole length of the sequence
  for i in range(max_length):
      # integer encode input sequence
      sequence = tokenizer.texts_to_sequences([in_text])[0]
      # pad input
      sequence = pad_sequences([sequence], maxlen=max_length)
      # predict next word
      yhat = model.predict([photo, sequence], verbose=0)
      # convert probability to integer
      yhat = argmax(yhat)
      # map integer to word
      word = word_for_id(yhat, tokenizer)
      # stop if cannot map the word
      if word is None:
          break
      # append as input for generating the next word
      in_text += ' ' + word
      # stop if the end of the sequence is predicted
      if word == 'endseq':
          break
  return in_text
```

<u>Step 3</u>: Evaluate a trained model against a given test dataset. The actual and predicted descriptions are collected and evaluated collectively using the corpus BLEU score that summarizes how close the generated text is to the expected text. BLEU scores are used in text translation for evaluating translated text against one or more reference translations. The NLTK Python library implements the BLEU score calculation in the *corpus_bleu()* function. A higher score close to 1 is better, a score closer to zero is worse. Then, we display the last photo with its generated caption.

```
# evaluate the skill of the model
def evaluate_model(model, descriptions, photos, tokenizer, max_length):
  actual, predicted = list(), list()
  # step over the whole set
  for key, desc_list in descriptions.items():
      # generate description
      yhat = generate_desc(model, tokenizer, photos[key], max_length)
      # store actual and predicted
      references = [d.split() for d in desc_list]
      actual.append(references)
      predicted.append(yhat.split())
  # calculate BLEU score
  print('BLEU-1: %f' % corpus_bleu(actual, predicted, weights=(1.0, 0, 0, 0)))
  print('BLEU-2: %f' % corpus_bleu(actual, predicted, weights=(0.5, 0.5, 0, 0)))
  print('BLEU-3: %f' % corpus_bleu(actual, predicted, weights=(0.3, 0.3, 0.3, 0)))
  print('BLEU-4: %f' % corpus_bleu(actual, predicted, weights=(0.25, 0.25, 0.25, 0.25)))

  # display the predicted caption of the last photo
  display.display(Image.open('/content/Flickr8k_Dataset/' + key +'.jpg'))
  print(yhat)
```

```
# load the model
filename = '/content/lab_model_4.h5'
model = load_model(filename)
# evaluate model
evaluate_model(model, test_descriptions, test_features, tokenizer, max_length)
```

Running the scripts above, render the following results.

```
Dataset: 50
Descriptions: test=50
Photos: test=50
BLEU-1: 0.562609
BLEU-2: 0.304598
BLEU-3: 0.212766
BLEU-4: 0.104708
```

```
startseq two people are playing on the beach endseq
```

Homework

Now that you know how to develop and evaluate a caption generation model, you can now use it to generate captions for entirely new photographs. Write a code that can generate caption for entirely new data (You can download new photos or used the image below).

For this homework, you are required to turn in the following:

a. Code for generating a description for a new photograph.

b. Generated caption result/s of the new photograph/s.