```
import torch
import torch.nn as nn
import torch.optim as optim
import os
from collections import Counter

import numpy as np
import torch.utils.data
from torchvision import transforms
from torchvision import models
from torch.utils.data import DataLoader
from torch.utils.data.sampler import SubsetRandomSampler, BatchSampler
from torchtext.data.metrics import bleu_score
from pycocotools.coco import COCO

import math
import time
import pickle
import json
import os
import urllib
import zipfile
import random
from tqdm import tqdm
from copy import deepcopy


import matplotlib.pyplot as plt
from PIL import Image
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /home/jdalal_umass_edu/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
class EncoderCNN(nn.Module):
    def __init__(self, embed_size):
        super(EncoderCNN, self).__init__()
        # load the pre-trained ResNet
        resnet = models.resnet50(pretrained=True)
        # freeze the weights
        for param in resnet.parameters():
            param.requires_grad_(False)
        # grab all CNN layers except the last one
        modules = list(resnet.children())[:-1]
        self.resnet = nn.Sequential(*modules)
        # embedding layers
        self.embedding = nn.Linear(resnet.fc.in_features, embed_size)

    def forward(self, images):
        # resnet stage
        features = self.resnet(images)
        # flatten to 1 dim
        features = features.view(features.size(0), -1)
        # embedding to final feature
        features = self.embedding(features)
        return features
```

```
class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1):
        super(DecoderRNN, self).__init__()
        # embedding layer
        self.embedding = nn.Embedding(vocab_size, embed_size)
        # LSTM layer(s)
        self.lstm = nn.LSTM(input_size=embed_size, hidden_size=hidden_size, num_layers=num_layers, batch_first=True)
        # dense layer from hidden states to vocab dimension
        self.fc = nn.Linear(hidden_size, vocab_size)

    def forward(self, features, captions):
        # batch size
        batch_size = features.shape[0]
        # embedding dimension
        embed_size = features.shape[1]
        # caption length
        seq_len = captions.shape[1]
        # remove the <end> token
        captions = captions[:, :-1]
        # pass the tokenized captions into the embedding layer
        embedded_captions = self.embedding(captions)  # (batch_size, seq_len-1, embed_size)
        # convert features as the very first tokens
        features = torch.unsqueeze(features, dim=1)  # (batch_size, 1, embed_size)
        # concatenate to obtain lstm_input
        lstm_input = torch.cat((features, embedded_captions), dim=1)  # (batch_size, seq_len, embed_size)
        # LSTM layer
        lstm_output, lstm_hidden = self.lstm(lstm_input)
        # dense layer
        fc_output = self.fc(lstm_output)
        return fc_output


    def sample(self, inputs, states=None, max_len=20):
        tokens = []
        x = inputs
        # output tokens one by one
        for _ in range(max_len):
            # lstm layer
            x, states = self.lstm(x, states)  # (batch_size=1, 1, hidden_size)
            # dense layer
            x = self.fc(x)  # (batch_size=1, 1, vocab_size)
            # token
            tok = torch.argmax(x, dim=-1)  # (batch_size=1, 1)
            # append to the output
            tokens.append(int(tok[0, 0]))
            # early stop (token == 1)
            if tok[0, 0] == 1:
                break
            # embedding
            x = self.embedding(tok)  # (batch_size, 1, embed_size)
        return tokens


def image_captioning_custom_image(img_path, encoder, decoder):
    encoder.eval()
    decoder.eval()
    encoder = encoder.to(device)
    decoder = decoder.to(device)
    # image preprocessing
    orig_image = np.array(Image.open(img_path).convert('RGB'))
    # plot the original image
    plt.imshow(orig_image)
    plt.axis('off')

    # caption prediction
    image_t = transform_eval(Image.open(img_path).convert('RGB'))
    image_t = torch.unsqueeze(image_t, 0)
    image_t = image_t.to(device)
    with torch.no_grad():
        features_t = encoder(image_t).unsqueeze(1)
        token_list = decoder.sample(features_t)
    decoded_word_list, decoded_sentence = get_word_list_and_sentence(token_list)
    print(decoded_sentence)
```

```
EMBED_SIZE = 512
HIDDEN_SIZE = 512
VOCAB_SIZE = 8852
encoder = EncoderCNN(embed_size=EMBED_SIZE)
decoder = DecoderRNN(embed_size=EMBED_SIZE, hidden_size=HIDDEN_SIZE, vocab_size=VOCAB_SIZE)
```

```
os.chdir('..')
```

```
model_name = "020422"
best_epoch = 1

encoder.load_state_dict(torch.load(os.path.join("encoder_" + model_name + "_ep" + str(best_epoch) + ".pth")))
decoder.load_state_dict(torch.load(os.path.join("decoder_" + model_name + "_ep" + str(best_epoch) + ".pth")))
```

```
<All keys matched successfully>
```

```
class Vocabulary(object):
    def __init__(self, vocab_threshold, vocab_file='/content/vocab.pkl',
                 start_word="<start>", end_word="<end>", unk_word="<unk>",
                 annotations_file="cocoapi/annotations/captions_train2014.json",
                 vocab_from_file=False):
        self.vocab_threshold = vocab_threshold
        self.vocab_file = vocab_file
        self.start_word = start_word
        self.end_word = end_word
        self.unk_word = unk_word
        self.annotations_file = annotations_file
        self.vocab_from_file = vocab_from_file
        self.get_vocab()

    def get_vocab(self):
        # load and use the existing vocab file
        if os.path.exists(self.vocab_file) & self.vocab_from_file:
            with open(self.vocab_file, 'rb') as f:
                vocab = pickle.load(f)
                self.word2idx = vocab.word2idx
                self.idx2word = vocab.idx2word
            print('Vocabulary successfully loaded from vocab.pkl file!')

        # build a new vocab file
        else:
            self.build_vocab()
            with open(self.vocab_file, 'wb') as f:
                pickle.dump(self, f)

    def build_vocab(self):
        self.init_vocab()
        self.add_word(self.start_word)
        self.add_word(self.end_word)
        self.add_word(self.unk_word)
        self.add_captions()

    def init_vocab(self):
        self.word2idx = {}
        self.idx2word = {}
        self.idx = 0

    def add_word(self, word):
        if not word in self.word2idx:
            self.word2idx[word] = self.idx
            self.idx2word[self.idx] = word
            self.idx += 1

    def add_captions(self):
        coco = COCO(self.annotations_file)
        counter = Counter()
        ids = coco.anns.keys()
        for i, id in enumerate(ids):
            caption = str(coco.anns[id]['caption'])
            tokens = nltk.tokenize.word_tokenize(caption.lower())
            counter.update(tokens)

            if i % 100000 == 0:
                print("[%d/%d] Tokenizing captions..." % (i, len(ids)))

        words = [word for word, cnt in counter.items() if cnt >= self.vocab_threshold]

        for i, word in enumerate(words):
            self.add_word(word)

    def __call__(self, word):
        if not word in self.word2idx:
            return self.word2idx[self.unk_word]
        return self.word2idx[word]

    def __len__(self):
        return len(self.word2idx)


VOCAB_THRESHOLD = 5

# build vocab file from training data
train_vocab = Vocabulary(vocab_threshold=VOCAB_THRESHOLD,
                         vocab_file="./vocab.pkl",
                         start_word="<start>",
                         end_word="<end>",
                         unk_word="<unk>",
                         annotations_file="cocoapi/annotations/captions_train2014.json",
                         vocab_from_file=False)
```

```
loading annotations into memory...
Done (t=0.58s)
creating index...
index created!
[0/414113] Tokenizing captions...
[100000/414113] Tokenizing captions...
[200000/414113] Tokenizing captions...
[300000/414113] Tokenizing captions...
[400000/414113] Tokenizing captions...
```

```
def get_word_list_and_sentence(token_list):
    word_list = []

    for tok in token_list:
        # skip the <start> token
        if tok == 0:
            continue

        # break if it's an <end> token
        if tok == 1:
            break

        # look up the word
        word = train_vocab.idx2word[tok]
        word_list.append(word)

    sentence = " ".join(word_list)

    return word_list, sentence


# validation/test data transform
transform_eval = transforms.Compose([transforms.Resize(256),
                                     transforms.CenterCrop(224),
                                     transforms.ToTensor(),
                                     transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))])

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
image_captioning_custom_image("./custom_test_images/animal_grass.png", encoder, decoder)
```

```
a large bear is standing in the grass .
```

Double-click (or enter) to edit