part-1

July 1, 2024

[ ]:
```
import os
import pandas as pd
import torch
from torch.utils.data import Dataset
from skimage import io
from nltk.tokenize import word_tokenize
from collections import Counter
from itertools import chain

class Vocabulary:
    def __init__(self):
        self.StrToIndx = {"<START>": 1, "<END>": 2, "<PAD>": 0, "<UNK>": 3}
        self.IndxToStr = {1: "<START>", 2: "<END>", 0: "<PAD>", 3: "<UNK>"}
        self.stoi = self.StrToIndx  # Map from word to index
        self.itos = self.IndxToStr  # Map from index to word

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

    def build_vocabulary(self, sentence_list):
        idx = 4
        self.token = [word_tokenize(sentence) for sentence in sentence_list]
        self.words_list = list(chain.from_iterable(self.token))
        self.word_counts = Counter(self.words_list)
        self.words = sorted(self.word_counts, key=self.word_counts.get,␣
 ↪reverse=True)

        for w in self.words:
            self.StrToIndx[w] = idx
            self.IndxToStr[idx] = w
            idx += 1

        self.stoi = self.StrToIndx  # Update stoi after building vocabulary
        self.itos = self.IndxToStr  # Update itos after building vocabulary

    def caption_to_number(self, caption_text):
        self.caption_words = word_tokenize(caption_text)
```

```
        return [self.StrToIndx[token.lower()]
                if token in self.caption_words else self.StrToIndx['<UNK>']
                for token in self.caption_words]


class CustomCaptionDataset(Dataset):
    def __init__(self, root_dir, annotation_file, transform=None):
        self.root_dir = root_dir
        self.annotations = pd.read_csv(annotation_file)
        self.transform = transform
        self.vocab = Vocabulary()
        self.vocab.build_vocabulary(self.annotations['caption'])

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

    def __getitem__(self, index):
        img_path = os.path.join(self.root_dir, str(self.annotations.iloc[index,␣
 ↪0]))

        image = io.imread(img_path)
        caption = self.vocab.caption_to_number(self.annotations.iloc[index, 1])

        if self.transform:
            image = self.transform(image)

        return image, caption




import torch
import torch.nn as nn
import torchvision.models as models


class EncoderCNN(nn.Module):
    def __init__(self, embed_size, train_CNN=False):
        super(EncoderCNN, self).__init__()
        self.train_CNN = train_CNN
        self.inception = models.inception_v3(pretrained=True, aux_logits=True) ␣
 ↪# Set aux_logits to True
        self.inception.fc = nn.Linear(self.inception.fc.in_features, embed_size)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)
```

```
    def forward(self, images):
        features = self.inception(images)

        for name, param in self.inception.named_parameters():
            if "fc.weight" in name or "fc.bias" in name:
                param.requires_grad = True
            else:
                param.requires_grad = self.train_CNN

        return self.dropout(self.relu(features))


class DecoderRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers):
        super(DecoderRNN, self).__init__()
        self.embed = nn.Embedding(vocab_size, embed_size)
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers)
        self.linear = nn.Linear(hidden_size, vocab_size)
        self.dropout = nn.Dropout(0.5)

    def forward(self, features, captions):
        embeddings = self.dropout(self.embed(captions))
        embeddings = torch.cat((features.unsqueeze(0), embeddings), dim=0)
        hiddens, _ = self.lstm(embeddings)
        outputs = self.linear(hiddens)
        return outputs

class CNNtoRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers):
        super(CNNtoRNN, self).__init__()
        self.encoderCNN = EncoderCNN(embed_size)
        self.decoderRNN = DecoderRNN(embed_size, hidden_size, vocab_size,␣
 ↪num_layers)

    def forward(self, images, captions):
        features = self.encoderCNN(images)
        outputs = self.decoderRNN(features, captions)
        return outputs

    def caption_image(self, image, vocabulary, max_length=50):
        result_caption = []

        with torch.no_grad():
            x = self.encoderCNN(image).unsqueeze(0)
            states = None

            for _ in range(max_length):
```

```
                hiddens, states = self.decoderRNN.lstm(x, states)
                output = self.decoderRNN.linear(hiddens.squeeze(0))
                predicted = output.argmax(1)
                result_caption.append(predicted.item())
                x = self.decoderRNN.embed(predicted).unsqueeze(0)

                if vocabulary.itos[predicted.item()] == "<EOS>":
                    break

        return [vocabulary.itos[idx] for idx in result_caption]


import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
from torch.utils.tensorboard import SummaryWriter
import nltk
nltk.download('punkt')
import nltk
from nltk.translate.bleu_score import corpus_bleu
from nltk.translate.bleu_score import SmoothingFunction
from nltk.translate.meteor_score import meteor_score
from pycocoevalcap.bleu.bleu import Bleu
from pycocoevalcap.rouge.rouge import Rouge
from pycocoevalcap.cider.cider import Cider
from pycocoevalcap.spice.spice import Spice

# from get_loader import get_loader
# from model import CNNtoRNN
# from customDataset import CustomCaptionDataset
from torch.utils.data import DataLoader

def save_checkpoint(state, filename="my_checkpoint.pth.tar"):
    print("=> Saving checkpoint")
    torch.save(state, filename)

def load_checkpoint(checkpoint, model, optimizer):
    print("=> Loading checkpoint")
    model.load_state_dict(checkpoint["state_dict"])
    optimizer.load_state_dict(checkpoint["optimizer"])
    step = checkpoint["step"]
    return step

import torch
import torchvision.transforms as transforms
from PIL import Image
```

```
def print_examples(model, device, dataset):
    transform = transforms.Compose(
        [
            transforms.Resize((299, 299)),
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ]
    )

    model.eval()

    # loop over the 5714 images in the train directory

    for i in range(5714):
        x = i + 1
        print(f"Training image-{x}")
        img = transform(Image.open(f"./custom_captions_dataset/train/train_{x}.
 ↪jpg").convert("RGB")).unsqueeze(0)
        ␣
 ↪print("-----------------------------------------------------------------------------------------
        print(f"Example {x} CORRECT: " + dataset.annotations.iloc[i,2])
        print("\n\n")
        print(f"Example {x} OUTPUT: " + " ".join(model.caption_image(img.
 ↪to(device), dataset.vocab)))
        ␣
 ↪print("-----------------------------------------------------------------------------------------
        )

def print_test(model, device, dataset):
    transform = transforms.Compose(
        [
            transforms.Resize((299, 299)),
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ]
    )

    for i in range(928):
        x = i+1

        print(f"Test image-{x}")
        img = transform(Image.open(f"./custom_captions_dataset/test/test_{x}.
 ↪jpg").convert("RGB")).unsqueeze(0)
        ␣
 ↪print("-----------------------------------------------------------------------------------------
```

```
        print(f"Example {x} CORRECT: " + dataset.annotations.iloc[i,2])
        print("\n\n")
        print(f"Example {x} OUTPUT: " + " ".join(model.caption_image(img.
 ↪to(device), dataset.vocab)))
        ␣
 ↪print("--------------------------------------------------------------------------------
        )

def train():
    transform = transforms.Compose(
        [
            transforms.Resize((356, 356)),
            transforms.RandomCrop((299, 299)),
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ]
    )

    TRAIN_DATASET = CustomCaptionDataset(
        root_dir="custom_captions_dataset/train",
        annotation_file="custom_captions_dataset/train.csv",
        transform=transform,
    )

    train_loader = DataLoader(dataset = TRAIN_DATASET, batch_size=32,␣
 ↪shuffle=True)
    torch.backends.cudnn.benchmark = True
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    load_model = False
    save_model = True

    # Hyperparameters
    embed_size = 256
    hidden_size = 256
    vocab_size = len(TRAIN_DATASET.vocab)
    num_layers = 1
    learning_rate = 3e-4
    num_epochs = 100


    # for tensorboard
    writer = SummaryWriter("runs/flickr")
    step = 0


    # initialize model, loss etc
    model = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers).to(device)
```

```
    criterion = nn.CrossEntropyLoss(ignore_index=TRAIN_DATASET.vocab.
↪stoi["<PAD>"])
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)


    # my_checkpoint.pth.tar is the model checkpoint file name to save the model
    if load_model:
        step = load_checkpoint(torch.load("my_checkpoint.pth.tar"), model,␣
↪optimizer)

    model.train()

    # _ = enumerate(train_loader)

    for epoch in range(num_epochs):

        print_examples(model, device, TRAIN_DATASET)

        if save_model:
            checkpoint = {
                "state_dict": model.state_dict(),
                "optimizer": optimizer.state_dict(),
                "step": step,
            }
            save_checkpoint(checkpoint)

        # _ = enumerate(train_loader)
        #  print all the images and captions in the train_loader


        for idx, (imgs, captions) in enumerate(train_loader):

            imgs = imgs.to(device)
            captions = captions.to(device)

            outputs = model(imgs, captions[:-1])
            loss = criterion(outputs.reshape(-1, outputs.shape[2]), captions.
↪reshape(-1))

            writer.add_scalar("Training loss", loss.item(), global_step=step)
            step += 1

            optimizer.zero_grad()
            loss.backward(loss)
            optimizer.step()
            pass
        print("for loop done")
```

```
    TEST_DATASET = CustomCaptionDataset(
        root_dir="custom_captions_dataset/test",
        annotation_file="custom_captions_dataset/test.csv",
        transform=transform,
    )

    test_loader = DataLoader(dataset = TEST_DATASET, batch_size=1, shuffle=True)

    print_test(model, device, TEST_DATASET)




def calculate_metrics(references, hypotheses):
    # Calculate BLEU score
    smoothing = SmoothingFunction().method4
    bleu_score = corpus_bleu(references, hypotheses,␣
 ↪smoothing_function=smoothing)

    # Calculate METEOR score
    meteor_score_val = meteor_score(references, hypotheses)

    # Initialize evaluation metrics objects
    bleu_eval = Bleu(n=4)
    rouge_eval = Rouge()
    cider_eval = Cider()
    spice_eval = Spice()

    # Prepare data for evaluation
    refs = {i: [refs] for i, refs in enumerate(references)}
    hyps = {i: [hyp] for i, hyp in enumerate(hypotheses)}

    # Calculate ROUGE-L score
    rouge_score = rouge_eval.compute_score(refs, hyps)['ROUGE_L']

    # Calculate CIDEr score
    cider_score = cider_eval.compute_score(refs, hyps)

    # Calculate SPICE score
    spice_score = spice_eval.compute_score(refs, hyps)

    return bleu_score, meteor_score_val, rouge_score, cider_score, spice_score
```

```python
if __name__ == "__main__":
    train()
```