cnn-rnn-image-caption

April 20, 2024

0.1 CNN-RNN Based Image Caption Generator

0.1.1 Submitted By:

0.1.2 Muhammad Waseem (2020-EE-111)

0.1.3 Muhammad Huzaifa (2020-EE-118)

0.1.4 Arhum Ahmed (2020-EE-123)

0.1.5 Uzair Ahmed (2020-EE-130)

[ ]:
```
# Checking if device has cuda
import torch
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
```

[ ]:
```
import torch
from tqdm import tqdm
import os  # when loading file paths
import pandas as pd  # for lookup in annotation file
import spacy  # for tokenizer
import torch
import torch.optim as optim
from torch.nn.utils.rnn import pad_sequence  # pad batch
from torch.utils.data import DataLoader, Dataset
from PIL import Image  # Load img
import torchvision.transforms as transforms
import torch.nn as nn
import torchvision.models as models
from PIL import Image
from torchtext.data.metrics import bleu_score # for bleu score
from torch.utils.tensorboard import SummaryWriter # For validation and training␣
 ↪error plots
```

[ ]:
```
"""
    Here are some classes to load data,
    make vocabolary and for dataloader
"""
```

```python
class WordMap:
    def __init__(self, thresh):
        self.spacy = spacy.load("en_core_web_sm")   # Loading library for tokenization
        self.itos = {0: "<PAD>", 1: "<SOS>", 2: "<EOS>", 3: "<UNK>"} # index to sentence
        self.stoi = {"<PAD>": 0, "<SOS>": 1, "<EOS>": 2, "<UNK>": 3} # sentence to index
        self.thresh = thresh  # frequency threshold for vocabulary

    def tokens(self, text):  # It converts the text to lowercase and returns a list of tokens.
        return [tok.text.lower() for tok in self.spacy.tokenizer(text)]

    def build_wordmap(self, captions_list): # It builds the Vocab, based on the frequency selected
        idx = 4 # starting index for start adding tokens in itos
        frequencies = {}
        for caption in captions_list:
            for token in self.tokens(caption):
                if token not in frequencies: # If token not in in wordmap add it
                    frequencies[token] = 1

                else: # if token already in wordmap then increase its frequency
                    frequencies[token] += 1

                if frequencies[token] == self.thresh: # if token passes a threshold then add it to the dict's
                    self.stoi[token] = idx
                    self.itos[idx] = token
                    idx += 1

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

    def text2idx(self, text): # Convert text into indices
        out = []
        tokenized_text = self.tokens(text)
        for token in tokenized_text:
            if token in self.stoi:
                  out.append(self.stoi[token])
            else:  # if token not in stoi then add this
                out.append(self.stoi["<UNK>"])
        return out
```

```python
class CustomDataSet(Dataset):
    def __init__(self, root_dir, captions_file, transform=None, thresh=5):
        self.root_dir = root_dir
        self.df = pd.read_csv(captions_file) # Reading the txt file as csv
        self.transform = transform
        self.imgs = self.df["image"]            # Taking out images names
        self.captions = self.df["caption"]      # Taking out captions
        self.vocab = WordMap(thresh) # Building Vocab
        self.vocab.build_wordmap(self.captions.tolist())

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

    def __getitem__(self, index):
        caption = self.captions[index]
        img_id = self.imgs[index]
        img = Image.open(os.path.join(self.root_dir, img_id)).convert("RGB")

        if self.transform is not None:
            img = self.transform(img)

        caption_vec = [self.vocab.stoi["<SOS>"]]        # At the start add <SOS>
        caption_vec += self.vocab.text2idx(caption) # Add the stoi
        caption_vec.append(self.vocab.stoi["<EOS>"])    # At the end add <EOS>

        return img, torch.tensor(caption_vec)

class Collate: # for padding
    def __init__(self, pad_idx):
        self.pad_idx = pad_idx

    def __call__(self, batch):
        imgs = [item[0].unsqueeze(0) for item in batch]
        imgs = torch.cat(imgs, dim=0)
        targets = [item[1] for item in batch]
        targets = pad_sequence(targets, batch_first=False, padding_value=self.
 ↪pad_idx)

        return imgs, targets

def call_dataloader( root_folder, annotation_file, transform, batch_size,␣
 ↪num_workers=0, shuffle=True, pin_memory=True):
    dataset = CustomDataSet(root_folder, annotation_file, transform=transform)
    pad_idx = dataset.vocab.stoi["<PAD>"]
    loader = DataLoader( dataset=dataset, batch_size=batch_size,␣
 ↪num_workers=num_workers, shuffle=shuffle, pin_memory=pin_memory,␣
 ↪collate_fn=Collate(pad_idx=pad_idx))
```

```
    return loader, dataset
```

[ ]:
```
"""
    Here are some classes for CNN and RNN
    initialization and their conjunction
"""

class CNN(nn.Module):
    def __init__(self, embed_size):
        super(CNN, self).__init__() # to assign module before module call
        resnet = models.resnet50(pretrained=True) # Using a pre-trained␣
 ↪resnet-50 model
        for param in resnet.parameters():
            param.requires_grad_(False)
        modules = list(resnet.children())[:-1]  # removing all of the FC layers
        self.resnet = nn.Sequential(*modules) # creating a new network
        self.embed = nn.Linear(resnet.fc.in_features, embed_size)
        self.batch= nn.BatchNorm1d(embed_size,momentum = 0.01) # Added an extra␣
 ↪batch normalization layer on top of conventional linear layer

    def forward(self, images):
        features = self.resnet(images)
        features = features.view(features.size(0), -1)
        features = self.batch(self.embed(features))  #  Applying the batch␣
 ↪normalization
        return features


class RNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers=1):
        super(RNN, self).__init__() # to assign module before module call
        self.embed = nn.Embedding(vocab_size, embed_size)  # Initializing an␣
 ↪embedded layer
        self.lstm = nn.LSTM(embed_size, hidden_size, num_layers) # Adding LSTM
        self.linear = nn.Linear(hidden_size, vocab_size) # Adding a linear␣
 ↪layer at output
        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) # taking out hidden states
        outputs = self.linear(hiddens) # output through the linear layer
        return outputs
```

```
class CNNtoRNN(nn.Module):
    def __init__(self, embed_size, hidden_size, vocab_size, num_layers):
        super(CNNtoRNN, self).__init__()
        self.cnn = CNN(embed_size)
        self.rnn = RNN(embed_size, hidden_size, vocab_size, num_layers)

    def forward(self, images, captions):
        features = self.cnn(images)
        outputs = self.rnn(features, captions)
        return outputs

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

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

            for _ in range(max_length):
                hiddens, states = self.rnn.lstm(x, states)
                output = self.rnn.linear(hiddens.squeeze(0))
                predicted = output.argmax(1)
                result_caption.append(predicted.item())
                x = self.rnn.embed(predicted).unsqueeze(0)

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

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

[ ]:
```
# Function to check the model's improvemnet during training
def print_examples(model, device, dataset):
    transform = transforms.Compose(
        [
            transforms.Resize((256, 256)),
            transforms.RandomCrop(224),
            transforms.ToTensor(),
            transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
        ]
    )

    predicted_captions = []

    model.eval()
```

```
    test_img1 = transform(Image.open("test_examples/dog.jpg").convert("RGB")).
↪unsqueeze(0)
    out = model.caption_image(test_img1.to(device), dataset.vocab)
    predicted_captions.append((out)[1:-1])
    print("Example 1 CORRECT: Dog on a beach by the ocean")
    print( "Example 1 OUTPUT: " + " ".join(out))

    test_img2 = transform(Image.open("test_examples/child.jpg").convert("RGB")).
↪unsqueeze(0)
    out = model.caption_image(test_img2.to(device), dataset.vocab)
    predicted_captions.append((out)[1:-1])
    print("Example 2 CORRECT: Child holding red frisbee outdoors")
    print("Example 2 OUTPUT: " + " ".join(out))

    test_img3 = transform(Image.open("test_examples/bus.png").convert("RGB")).
↪unsqueeze(0)
    out = model.caption_image(test_img3.to(device), dataset.vocab)
    predicted_captions.append((out)[1:-1])
    print("Example 3 CORRECT: Bus driving by parked cars")
    print("Example 3 OUTPUT: " + " ".join(out))

    test_img4 = transform(Image.open("test_examples/boat.png").convert("RGB")).
↪unsqueeze(0)
    out = model.caption_image(test_img4.to(device), dataset.vocab)
    predicted_captions.append((out)[1:-1])
    print("Example 4 CORRECT: A small boat in the ocean")
    print("Example 4 OUTPUT: "+ " ".join(out))

    test_img5 = transform(Image.open("test_examples/horse.png").convert("RGB")).
↪unsqueeze(0)
    out = model.caption_image(test_img5.to(device), dataset.vocab)
    predicted_captions.append((out)[1:-1])
    print("Example 5 CORRECT: A cowboy riding a horse in the desert")
    print("Example 5 OUTPUT: "+ " ".join(out))
    #model.train()

def save_checkpoint(state, filename="model/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
```

[ ]:
```
"""
    Training function. It loads data for training, validation and
    testing. Calculates the error and validates it using the validation set.
    Also calculates the bleu score
"""
def train():
    transform = transforms.Compose(
        [
            transforms.Resize((256, 256)),
            transforms.RandomCrop(224),
            transforms.ToTensor(),
            transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
        ]
    )

    train_loader, dataset = call_dataloader(
        root_folder="flickr8k/Training/images",
        annotation_file="flickr8k/Training/captions.txt",
        transform=transform,
        batch_size=32,
        num_workers=0,
    )

    validation_loader, validation_dataset = call_dataloader(
        root_folder="flickr8k/Validation/images",
        annotation_file="flickr8k/Validation/captions.txt",
        transform=transform,
        batch_size=32,
        num_workers=0,
    )

    _, test_dataset = call_dataloader(
            root_folder="flickr8k/Training/images",
            annotation_file="flickr8k/Training/captions.txt",
            transform=transform,
            num_workers=0,
            batch_size=1,
            shuffle=False
        )

    df = pd.read_csv("flickr8k/Validation/captions.txt")
    test_img = sorted(set(df['image']))
    test_caption = df['caption']

    best_bleu = 0

    torch.backends.cudnn.benchmark = True
```

```
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    load_model = True
    save_model = True
    train_CNN = False

    # Hyperparameters
    embed_size = 512
    hidden_size = 512
    vocab_size = len(dataset.vocab)
    num_layers = 1
    learning_rate = 4e-4
    num_epochs = 50

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

    # initialize model, loss etc
    model = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers).to(device)
    loss_fn = nn.CrossEntropyLoss(ignore_index=dataset.vocab.stoi["<PAD>"])
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    # Only finetune the CNN
    for name, param in model.cnn.resnet.named_parameters():
        if "fc.weight" in name or "fc.bias" in name:
            param.requires_grad = True
        else:
            param.requires_grad = train_CNN

    if load_model:
        step = load_checkpoint(torch.load("checkpoint.pth.tar"), model,␣
↪optimizer)

    model.train()

    bleu_val = 0

    for epoch in range(num_epochs):
        print(epoch)
        print_examples(model, device, dataset)
        print("Bleu Score:   ", bleu_val)
        model.train()

        if(bleu_val > best_bleu):
            best_bleu = bleu_val
            checkpoint = {"state_dict": model.state_dict(),"optimizer":␣
↪optimizer.state_dict(),"step": step,}
```

```
            save_checkpoint(checkpoint, filename="best_model/
↪"+str(best_bleu)+"checkpoint.pth.tar")

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

        total_train_loss = 0
        for imgs, captions in tqdm(train_loader, total=len(train_loader),␣
↪leave=False):
            imgs = imgs.to(device)
            captions = captions.to(device)

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

            writer.add_scalar("Training and Validation loss (Batch)", loss.
↪item(), global_step=step)
            writer.add_scalar("Training loss (Batch)", loss.item(),␣
↪global_step=step)
            step += 1

            optimizer.zero_grad()
            loss.backward(loss)
            optimizer.step()
            total_train_loss += loss.item()

        avg_train_loss = total_train_loss / len(train_loader)
        writer.add_scalar("Training and Validation loss (Epoch)",␣
↪avg_train_loss, global_step=epoch)
        writer.add_scalar("Training loss (Epoch)", avg_train_loss,␣
↪global_step=epoch)

        # Validation loop
        model.eval()
        total_val_loss = 0
        with torch.no_grad():
            for imgs, captions in tqdm(validation_loader,␣
↪total=len(validation_loader), leave=False):
                imgs = imgs.to(device)
                captions = captions.to(device)

                outputs = model(imgs, captions[:-1])
```

```
                val_loss = loss_fn(outputs.reshape(-1, outputs.shape[2]),␣
↪captions.reshape(-1))
                total_val_loss += val_loss.item()

        avg_val_loss = total_val_loss / len(validation_loader)
        writer.add_scalar("Validation loss (Epoch)", avg_val_loss,␣
↪global_step=epoch)
        writer.add_scalar("Training and Validation loss (Epoch)", avg_val_loss,␣
↪global_step=epoch)
        writer.add_scalar("Training and Validation loss (Batch)", avg_val_loss,␣
↪global_step=step)
        writer.add_scalar("Validation loss (Batch)", avg_val_loss,␣
↪global_step=step)

        # calculating bleu score
        pred_cptns = []
        ref_cptns = []
        start_idx = 0
        end_idx = 5

        for img in test_img:
            out = transform(Image.open("flickr8k/Validation/images/"+img).
↪convert("RGB")).unsqueeze(0)
            cptn = model.caption_image(out.to(device), dataset.vocab)
            pred_cptns.append(cptn[1:-1])
            out = list(test_caption[start_idx:end_idx])
            temp = []
            for i in out:
                temp.append(i.split())
            ref_cptns.append(temp)
            temp = []
            start_idx += 5
            end_idx += 5

        bleu_val = bleu_score(pred_cptns, ref_cptns, weights=[0.5, 0.3, 0.1, 0.
↪1])
```

[ ]:
```
# Calling the training function
train()
```

[ ]:
```
# Testing the model with Random Images
import random
from PIL import Image

transform = transforms.Compose(
        [
```

```
            transforms.Resize((256, 256)),
            transforms.RandomCrop(224),
            transforms.ToTensor(),
            transforms.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225))
        ]
    )

_, dataset = call_dataloader(
        root_folder="flickr8k/Training/images",
        annotation_file="flickr8k/Training/captions.txt",
        transform=transform,
        num_workers=0,
        batch_size=1
    )

df = pd.read_csv("flickr8k/Testing/captions.txt")
imgs = list(sorted(set(df["image"])))
captions = df["caption"]
rand_idx = random.randint(0, len(imgs)-1)
img_name = imgs[rand_idx]
caption = list(captions[0+5*rand_idx : 5+5*rand_idx])
print("Correct Captions: ", caption)

vocab_size = len(dataset.vocab)
device = torch.device("cuda")

embed_size = 512
hidden_size = 512
num_layers = 1
learning_rate = 4e-4
device = torch.device("cuda")
model = CNNtoRNN(embed_size, hidden_size, vocab_size, num_layers).to(device)
optimizer = optim.Adam(model.parameters(), lr=learning_rate)
step = load_checkpoint(torch.load("best_model/with_bn/50checkpoint.pth.tar",␣
 ↪map_location=device), model, optimizer)

model.eval()

test_img1 = transform(Image.open("flickr8k/Testing/images/"+str(img_name)).
 ↪convert("RGB")).unsqueeze(0)
out = model.caption_image(test_img1.to(device), dataset.vocab)
print("Image Name:",  img_name)
print( "Model's Output: " + " ".join(out))

image = Image.open("flickr8k/Testing/images/"+str(img_name))
image.show()
```

[ ]:
```
# Run this command in the terminal to get the training and validation error␣
↪plots
# tensorboard --logdir=\Users\Arhum\Desktop\Project\Demo\runs
```