МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра теоретических основ компьютерной безопасности и криптографии

ГЕНЕРАЦИЯ ТЕКСТОВОГО ОПИСАНИЯ К ИЗОБРАЖЕНИЮ С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ

КУРСОВАЯ РАБОТА

студента 3 курса 331 группы
направления 100501 — Компьютерная безопасность
факультета КНиИТ
Улитина Ивана Владимировича

Научный руководитель
доцент _____ Слеповичев И. И.

Заведующий кафедрой
_____ Абросимов М. Б.

Саратов 2022

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Здесь будет введение.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Здесь будет список определений, обозначений и сокращений

Здесь будет заключение

ПРИЛОЖЕНИЕ А
Код getloader.py

```
import os
import pandas as pd
import spacy
import torch
from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import DataLoader, Dataset
from PIL import Image
import torchvision.transforms as transforms

# будут определяться слова английского языка
SPACY_ENG = spacy.load('en_core_web_sm')


class Vocabulary:
    """
        Класс словаря, описывающий структуры, содержащие слова, которые запоминает модель в процессе
        основе которых формирует подпись к изображению.
    """

    def __init__(self, freq_threshold):
        """
            Функция инициализации, при котором генерируются объекты типа словарь (первый объект - оп
            индексу, второй - определяет индекс по слову).

            :param freq_threshold: максимальное количество повторяющихся слов
            """

        self.itos = {0: "<PAD>", 1: "<SOS>", 2: "<EOS>", 3: "<UNK>"}
        self.stoi = {"<PAD>": 0, "<SOS>": 1, "<EOS>": 2, "<UNK>": 3}
        self.freq_threshold = freq_threshold

    def __len__(self):
        """
            Функция возвращает длину словаря.

            :return: длина словаря (ключ - индекс, значение - слово в словаре)
            """
        return len(self.itos)

    @staticmethod
    def tokenizer_eng(text):
        """
            Функция токенизирует слова некоторого текста.

            :param text: входной текст на английском языке
            :return: список токенов для каждого слова в тексте
```

```
        """

        # пример: "I love bananas" -> ["i", "love", "bananas"]
        return [tok.text.lower() for tok in SPACY_ENG.tokenizer(text)]

    def build_vocabulary(self, sentence_list):
        """
            Функция осуществляет заполнение словаря (добавление слов в него и определение для каждог

        :param sentence_list: список предложений, являющихся подписям к изображениям
        """

        frequencies = {}
        idx = 4

        for sentence in sentence_list:
            for word in self.tokenizer_eng(sentence):
                if word not in frequencies:
                    frequencies[word] = 1
                else:
                    frequencies[word] += 1

                if frequencies[word] == self.freq_threshold:
                    self.stoi[word] = idx
                    self.itos[idx] = word
                    idx += 1

    def numericalize(self, text):
        """
            Функция заменяет слова токенизированного текста соответствующими этим словам индексами.

        :param text: некоторый текст, подлежащий токенизированию
        :return: список индексов
        """
        tokenized_text = self.tokenizer_eng(text)

        return [
            self.stoi[token] if token in self.stoi else self.stoi["<UNK>"]
            for token in tokenized_text
        ]


class FlickrDataset(Dataset):
    """
        Определение класса датасета для набора данных Flickr.
    """

    def __init__(self, root_dir, caption_file, transform=None, freq_threshold=5):
```

```
        """
            Инициализирует объект датасета, определяя списки путей к изображениям и соответствующим
            формирует словарь.

        :param root_dir: корневая директория, откуда будут браться изображения
        :param caption_file: адрес файла с подписями к изображениям
        :param transform: некоторый объект, преобразовывающий изображения заданным образом
        :param freq_threshold: максимальное количество повторяющихся слов
        """
        self.root_dir = root_dir
        self.df = pd.read_csv(caption_file)
        self.transform = transform

        self.imgs = self.df["image"]
        self.captions = self.df["caption"]

        self.vocabulary = Vocabulary(freq_threshold)
        self.vocabulary.build_vocabulary(self.captions.tolist())

    def __len__(self):
        """
            Функция возвращает длину датафрейма (т.е. количество различных подписей к изображениям).

        :return: длина датафрейма
        """
        return len(self.df)

    def __getitem__(self, index):
        """
            Функция определяет возможность взятия одной сущности из датасета (в частности, изображен
            ему подписи).

        :param index: индекс сущности, которую необходимо считать
        :return: кортеж из трансформированного изображения и тензора преобразованной в индексы слов
        """
        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)

        numericalized_caption = [self.vocabulary.stoi["<SOS>"]]
        numericalized_caption += self.vocabulary.numericalize(caption)
        numericalized_caption.append(self.vocabulary.stoi["<EOS>"])

        return img, torch.tensor(numericalized_caption)
```

```python
class MyCollate:
    """
        Класс MyCollate для помещения в batch значений датасета.

    """
    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 get_loader(
        root_folder,
        annotation_file,
        transform,
        batch_size=32,
        num_workers=8,
        shuffle=True,
        pin_memory=True
):
    """
    Функция get_loader осуществляет загрузку данных для работы модели машинного обучения с этими дан

    :param root_folder: корневая директория, откуда будут браться изображения
    :param annotation_file: путь к файлу с подписями для изображений
    :param transform: преобразование, которое необходимо сделать с изображениями
    :param batch_size: размер batch
    :param num_workers: определяет количество некоторого ресурса компьютера для ускорения работы
    :param shuffle: параметр перемешивания сущностей
    :param pin_memory: определяет количество некоторого ресурса компьютера

    :return: кортеж из объекта загрузки данных и объекта датасета
    """
    dataset = FlickrDataset(root_folder, annotation_file, transform)
    pad_idx = dataset.vocabulary.stoi["<PAD>"]

    loader = DataLoader(
        dataset=dataset,
        batch_size=batch_size,
        num_workers=num_workers,
        shuffle=shuffle,
```

```
        pin_memory=pin_memory,
        collate_fn=MyCollate(pad_idx)
    )

    return loader, dataset


def main():
    """
        Запуск функцию осуществляет проверку корректности созданных классов.
    """
    transform = transforms.Compose(
        [
            transforms.Resize((224, 224)),
            transforms.ToTensor(),
        ]
    )
    dataloader = get_loader("archive/images", annotation_file="archive/captions.txt",
                            transform=transform)

    for idx, (imgs, captions) in enumerate(dataloader):
        print(imgs.shape)
        print(captions.shape)


if __name__ == "__main__":
    main()
```

ПРИЛОЖЕНИЕ Б
Код model.py

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


class EncoderCNN(nn.Module):
    """
        Кодировщик, определяющийся с помощью предобученной GoogleNetv3.
    """

    def __init__(self, embedding_size, train_CNN=False):
        """
            Инициализация модели с определением её слоев.

        :param embedding_size: размер эмбеддинга
        :param train_CNN: задается ли CNN для обучения
        """
        super(EncoderCNN, self).__init__()
        self.train_CNN = train_CNN
        self.inception = models.inception_v3(pretrained=True, aux_logits=False)
        self.inception.fc = nn.Linear(self.inception.fc.in_features, embedding_size)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.5)

    def forward(self, images):
        """
            Функция, определяющая процесс передачи весов внутри кодировщика.

        :param images: список тензоров, полученных преобразованием изображений
        :return: веса модели, пропущенные через relu с осуществлением сброса весов
        """
        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):
    """
        Декодировщик, определяемый с помощью LSTM-модели.
    """
```

```
    def __init__(self, embedding_size, hidden_size, vocabulary_size, layers_num):
        """
            Инициализация модели с определением её слоев.

        :param embedding_size: размер эмбеддинга
        :param hidden_size: выходной вектор значений LSTM
        :param vocabulary_size: размер словаря
        :param layers_num: количество слоев, составляющих LSTM
        """
        super(DecoderRNN, self).__init__()
        self.embedding = nn.Embedding(vocabulary_size, embedding_size)
        self.lstm = nn.LSTM(embedding_size, hidden_size, layers_num)
        self.linear = nn.Linear(hidden_size, vocabulary_size)
        self.dropout = nn.Dropout(0.5)

    def forward(self, features, captions):
        """
            Функция определяющая процесс передачи весов внутри декодировщика

        :param features: получаемые вектора, являющиеся результатом работы кодировщика
        :param captions: подписи к изображениям
        :return: подпись к входному изображению
        """
        embeddings = self.dropout(self.embedding(captions))
        embeddings = torch.cat((features.unsqueeze(0), embeddings), dim=0)
        hiddens, _ = self.lstm(embeddings)
        outputs = self.linear(hiddens)
        return outputs


class CNNtoRNNTranslator(nn.Module):
    """
        Класс, описывающий ансамбль, составленный из кодировщика и декодировщика
    """

    def __init__(self, embedding_size, hidden_size, vocabulary_size, layers_num):
        """
            Инициализация элементов ансамбля.

        :param embedding_size: размер эмбеддинга
        :param hidden_size: количество выходных векторов LSTM
        :param vocabulary_size: размер словаря
        :param layers_num: количество слоев, составляющих LSTM
        """
        super(CNNtoRNNTranslator, self).__init__()
        self.encoderCNN = EncoderCNN(embedding_size)
        self.decoderRNN = DecoderRNN(embedding_size, hidden_size, vocabulary_size, layers_num)
```

```
def forward(self, images, captions):
    """
        Процесс передачи весов между элементами ансамбля

    :param images: тензоры изображений
    :param captions: подписи к изображениям
    :return: сгенерированные подписи к изображениям
    """
    features = self.encoderCNN(images)
    outputs = self.decoderRNN(features, captions)
    return outputs

def image_caption(self, image, vocabulary, max_length=50):
    """
        Функция генерирует подпись (максимальная длина - 50 символов) к изображению на основе сл
        предварительно обработав входное изображение.

    :param image: изображение
    :param vocabulary: словарь
    :param max_length: максимальная длина генерируемой подписи
    :return: вектор слов, определяющий подпись к изображению
    """
    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.embedding(predicted).unsqueeze(0)

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

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

ПРИЛОЖЕНИЕ В

Код train.py

```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision.transforms as transforms
import tensorboard
from torch.utils.tensorboard import SummaryWriter
# from utils import save_checkpoint, load_checkpoint, print_examples
from model import CNNtoRNNTranslator
from getloader import get_loader


def train():
    """
        Функция осуществляет процесс обучения модели машинного обучения.

    """

    # преобразования, которым подлежат изображения
    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)),
        ]
    )

    # определение объектов загрузчика данных с объектом датасета с учетом пути к нужным файлам и зад
    # преобразованиями изображений
    train_loader, dataset = get_loader(
        root_folder="archive/images",
        annotation_file="archive/training_captions.txt",
        transform=transform,
        num_workers=6,
    )

    # обучение будет происходит с помощью технологии Cuda
    torch.backends.cudnn.benchmark = True
    print("cuda: ", torch.cuda.is_available())
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    load_model = False
    save_model = True

    # задача гиперпараметров
    embedding_size = 256
    hidden_size = 256
```

```
vocabulary_size = len(dataset.vocabulary)
num_layers = 1
learning_rate = 3e-4
num_epochs = 100

# в процессе обучения будут сохраняться характеристики, отображаемые с помощью tensorboard
writer = SummaryWriter("runs/flickr")
step = 0

# инициализация модели, функции потерь и оптимизатора
model = CNNtoRNNTranslator(embedding_size, hidden_size, vocabulary_size, num_layers).to(device)
criterion = nn.CrossEntropyLoss(ignore_index=dataset.vocabulary.stoi["<PAD>"])
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# подразумевается возможность загрузки весов модели
if load_model:
    checkpoint = torch.load("my_checkpoint.pth.tar")
    model.load_state_dict(checkpoint["state_dict"])
    optimizer.load_state_dict(checkpoint["optimizer"])
    step = checkpoint["step"]

# включение режима обучения модели
model.train()

for epoch in range(num_epochs + 1):
    print("Epoch: ", epoch)

    # сохранение параметров модели каждые 10 эпох с учетом булевой переменной
    if save_model and (epoch % 10 == 0):
        checkpoint = {
            "state_dict": model.state_dict(),
            "optimizer": optimizer.state_dict(),
            "step": step,
        }
        torch.save(checkpoint, f"my_checkpoint_{epoch}.pth.tar")

    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))

        # сохранение значения функции потерь для формирования графика с помощью tensorboard
        writer.add_scalar("Training loss", loss.item(), global_step=step)
        step += 1

        optimizer.zero_grad()
```

```
            loss.backward(loss)
            optimizer.step()


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

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Короткий С., ”Нейронные сети: Основные положения”, [Электронный ресурс] : [статья] / URL: http://www.shestopaloff.ca/kyriako/Russian/Artificial_-Intelligence/Some_publications/Korotky_Neuron_network_Lectures.pdf (дата обращения 27.04.2021) Загл. с экрана. Яз. рус.