

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	3
ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ	4
1 Теоретическая часть	6
1.1 Сверточная нейронная сеть	6
1.2 Рекуррентная нейронная сеть и LSTM	8
1.3 Метрики оценки качества обучения	11
1.4 Функции потерь и функции активации	11
1.5 Задачи, решаемые генерацией текстового описания к изображе- нию с помощью нейронной сети	11
2 Практическая часть	11
2.1 Описание инструментов и библиотек программной реализации	11
2.2 Описание набора данных для обучения и теста	11
2.3 Программная реализация алгоритма	11
2.4 Приведение характеристик обучения и гиперпараметров	11
2.5 Результаты обучения	11
ЗАКЛЮЧЕНИЕ	11
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	12
Приложение А Код getloader.py	13
Приложение Б Код model.py	18
Приложение В Код train.py	21

ВВЕДЕНИЕ

Последнее десятилетие имплементации алгоритмов искусственного интеллекта раз за разом поражали людей своими способностями решать задачи, считавшиеся до этого выполнимыми исключительно человеком. И с каждым годом темп развития этой сферы информационных технологий многократно увеличивался, всё сильнее поражая людей обширностью потенциала и возможностей использования глубокого обучения в повседневной жизни, которая постепенно преображалась в силу движения прогресса. В современном мире так много приложений нейросетевых алгоритмов в самых различных отраслях человеческой жизнедеятельности, что всё чаще человек использует ИИ, иногда даже не подозревая об этом. От нахождения кратчайшего пути из дома до аэропорта и прогнозирования погоды по параметрам воздуха и вплоть до удовлетворения банковских услуг пользователя с помощью автоответчика — всё это напрямую говорит об удивительном множестве возможностей использования искусственного интеллекта, которое постоянно расширяется.

Среди разделов глубокого обучения, использование которых ввергает людей в восхищение перед возможностями ИИ, следует выделить **NLP** и **Computer Vision**. В данной работе пойдёт речь именно о такой совокупности нейросетей, которая будет осуществлять обработку изображения таким образом, чтобы ко входному для алгоритма рисунку генерировалось текстовое описание. Чаще всего, в англоязычных ресурсах данная задача называется **Image Captioning**, и реализации её решения являются прикладной технологией для самых различных нужд.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Перед анализом теоретической составляющей глубокого обучения и архитектур нейронных сетей, которые осуществляют генерацию текстового описания к входному для алгоритма изображению, стоит ввести ряд терминов и определений, являющихся основой для понимания работы алгоритма генерации.

Искусственный интеллект (англ. Artificial Intelligence) — технология создания алгоритмов, лежащих в основе проектирования интеллектуальных машин и программ, способных имитировать деятельность человека.

Нейронная сеть (нейросеть) (англ. Neural Network) — математическая модель, чаще всего имеющая программную интерпретацию, сутью которой является реализация деятельности, похожей на деятельность биологических нейронных сетей. Нейронная сеть используется при создании какого-либо из алгоритмов искусственного интеллекта и состоит из совокупности нейронов, соединенных между собой связями.

Признак (англ. Feature) — каждый отдельный элемент информации, включаемый в представление о каком-либо анализируемом объекте.

Машинное обучение (англ. Machine Learning) — область науки об искусственном интеллекте, которая изучает способы создания алгоритмов, которые могут обучаться (развиваться).

Глубокое обучение (англ. Deep Learning) — частный случай машинного обучения, который представляет из себя методы машинного обучения, основанные на обучении представлений. Осуществляет получение представлений путем их выражения через более простые представления, а формирование последних, в свою очередь, реализуется через ещё более простые представления, и так далее.

Компьютерное зрение (англ. Computer Vision) — область науки об искусственном интеллекте, использующая методы машинного и глубокого обучения для решения задач распознавания, классификации, мониторинга с помощью получения необходимой информации из изображения.

Обработка естественного языка (англ. Natural Language Processing, NLP) — направление развития ИИ и математической лингвистики, которое изучает проблемы синтеза и компьютерного анализа текстов на естественных языках. Говоря об искусственном интеллекте, под анализом подразумевается понимание

языка, а под синтезом — генерация грамотного с точки зрения языковых норм текста.

Текстовое описание к изображению (англ. Image Captioning) — это задача описания содержания изображения словами естественного языка. Принцип решения лежит на пересечении таких разделов глубокого обучения как компьютерного зрения и обработки естественного языка. В большинстве систем генерации текстовых описаний к изображениям используется структура кодировщик-декодер, в которой входное изображение кодируется в промежуточное представление информации о содержимом изображения, а затем декодируется в описательную текстовую последовательность.

1 Теоретическая часть

1.1 Сверточная нейронная сеть

Искусственные нейронные сети или ИНС (англ. Artificial Neural Network или ANN) — это системы вычислительной обработки, принцип работы которых в значительной степени основан на том, как работают биологические нервные системы (в частности, человеческий мозг). ИНС в основном состоят из большого количества взаимосвязанных вычислительных узлов (называемых нейронами), работа которых определяется соединениями со слоями, содержащими другие нейроны. В итоге совокупность вычислений, производимых этими нейронами, осуществляет процесс обучения, который заключается в минимизации ошибки окончательного вывода нейросети (т.е. результата её работы).

Сверточные нейронные сети (англ. Convolutional Neural Network, CNN) аналогичны традиционным ANN в том, что они состоят из нейронов, которые самооптимизируются посредством обучения. Принцип, при котором каждый нейрон получает входные данные и выполняет операцию (например, скалярное произведение, за которым следует применение нелинейной функции) — это основа бесчисленных вариантов реализаций ИНС. Организовывая процесс преобразования входных необработанных векторов изображений в некоторый окончательный результат (будь то обработка изображения, классификация изображения или генерация текстовой подписи), вся нейросеть по-прежнему будет представлять собой функции преобразования весов-коэффициентов. Последний слой будет содержать функцию потерь, определяемую классом решаемой задачи, и все обычные принципы организации архитектуры модели, разработанные для традиционных ИНС, также могут быть применимы и в работе со сверточными нейросетями.

Принципиальное отличие обычной нейронной сети от сверточной является наличие в последней специального ”сверточного” слоя, который применяет к входным весам математическую операцию свертки. Термин свертка относится к математической комбинации двух функций для получения третьей функции. В случае CNN свертка выполняется для входных данных с использованием фильтра или ядра (эти термины взаимозаменяемы), чтобы затем создать карту признаков (англ. feature map). Выполнение свертки осуществляется перемещением фильтра по минорам входной матрицы. Для каждого минора определенного размера выполняется матричное умножение с соответствующим ему фильтром

и суммируется результат на карте признаков.

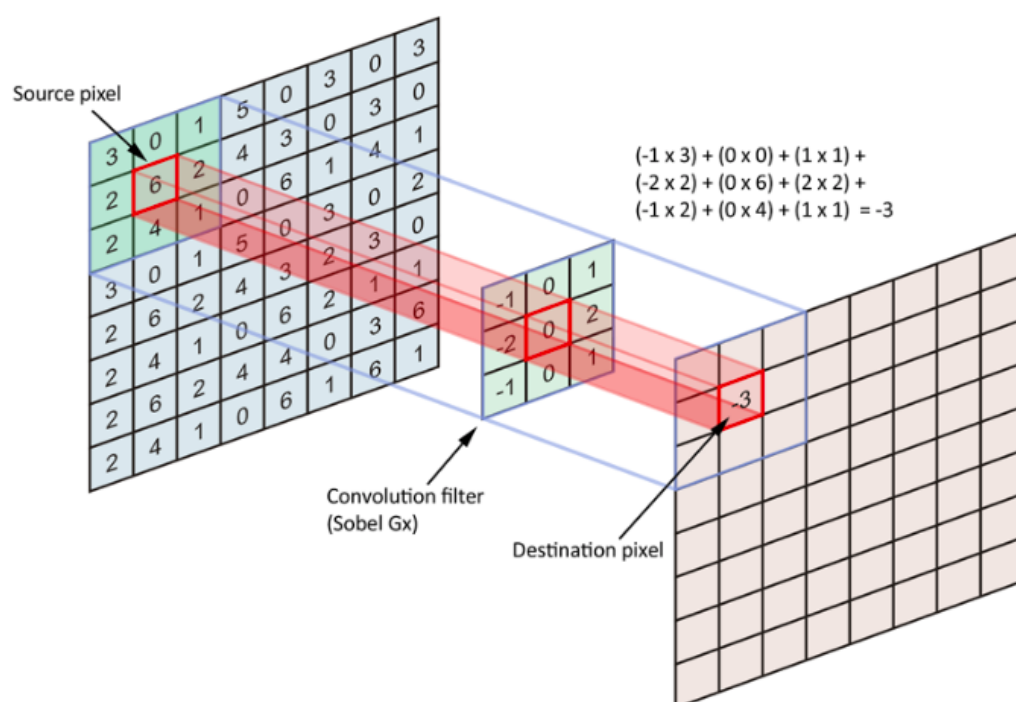


Рисунок 1 – Операция свёртки

Операции свертки выполняются на входной матрице, где каждая операция использует определенный фильтр. Результатом итераций математической процедуры являются различные карты признаков. В качестве шага, завершающего операцию, берутся все эти карты признаков и объединяются вместе в качестве конечного результата сверточного слоя.

Сверточная нейронная сеть — это алгоритм глубокого обучения, который применяется для обработки данных с сеточной топологией. Один из примеров такого вида данных — временные ряды, которые представимы в виде одномерной сетки примеров, выбираемых через регулярные промежутки времени (англ. timestamp). Вторым, более актуальным для этой работы примером, являются изображения, которые можно интерпретировать как двумерную сетку пикселей.

Популярность использования такого вида нейросетей при работе с изображениями обусловлена их отличительными положительными чертами. Сверточные нейронные сети обеспечивают частичную устойчивость к изменениям масштаба, смещениям, поворотам, смене ракурса и прочим искажениям картинки. Сверточные нейронные сети объединяют три архитектурных идеи, для обеспечения инвариантности к масштабируемости, вращению и другим про-

пространственным деформациям:

1. локальные рецепторные поля (обеспечивают локальную двумерную связность нейронов);
2. общие синаптические коэффициенты (обеспечивают детектирование некоторых черт в любом месте изображения и уменьшают общее число весовых коэффициентов);
3. иерархическая организация с пространственными подвыборками.

1.2 Рекуррентная нейронная сеть и LSTM

Рекуррентная нейронная сеть или РНС (англ. Recurrent Neural Network, RNN) — это тип искусственной нейронной сети, которая обрабатывает последовательности данных и временные ряды. Подобно нейронным сетям с прямой связью (англ. Feedforward Neural Network, FNN) и CNN, рекуррентные нейронные сети используют обучающие данные для изменения своих весов. Основным отличием от других видов сетей является ”память”, суть которой в том, что в процессе обработки входной информации особым текущим слоем в RNN, используются входные параметры к некоторым предыдущим слоям сети (таким образом влияя на результат работы текущего слоя сети). В то время как традиционные глубокие нейронные сети предполагают, что входные и выходные данные слоев независимы друг от друга, выходы рекуррентных нейронных сетей зависят от предшествующих элементов внутри последовательности этих слоев. Хотя будущие преобразования также могут быть полезны для определения результата данной последовательности, однонаправленные рекуррентные нейронные сети не могут учитывать эти преобразования в своих прогнозах.

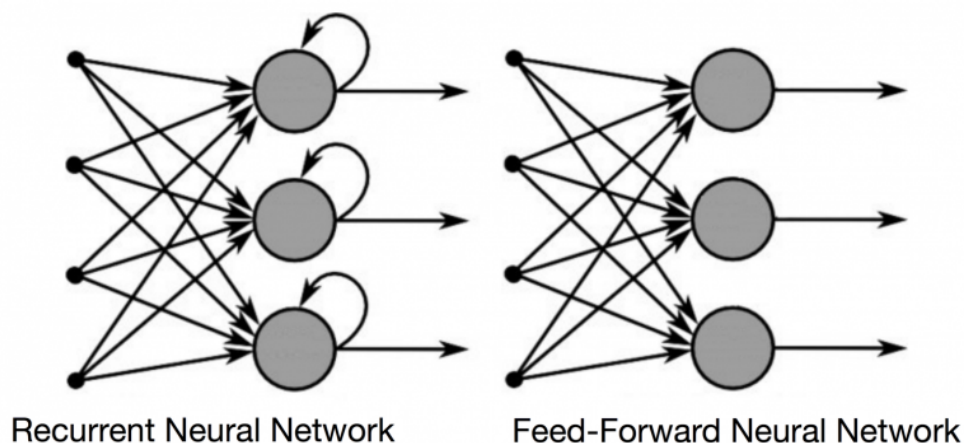


Рисунок 2 – Абстрактное сравнение архитектур FNN и RNN

Однако при использовании первых архитектур RNN возникала проблема потери способности связывать информацию в силу уменьшения влияния аргументов слоев сети на текущий обрабатываемый слой по мере увеличения "расстояния" между слоем, для которого были изначально предназначены аргументы, и текущим слоем. Уменьшение влияния выражается через проблему исчезающего градиента (англ. *Vanishing gradient problem*), которая возникает в процессе обучения ANN с помощью методов, основанных на градиентном спуске (англ. *Gradient Descent*) и методе обратного распространения ошибки (англ. *Backpropagation*). В этих способах обучения, на протяжении всей итерации обучения или эпохи, каждый из весов нейросети обновляется пропорционально частной производной функции ошибки от текущего веса. Времени значение градиента может становиться бесконечно малым, что препятствует обновлению значения веса. На практике, в силу отсутствия возможности сохранения качества передачи параметров между слоями, была представлена реализация модификации рекуррентной нейросети, которая способна к обучению долгосрочным зависимостям. Название такого подкласса RNN — сеть с долгой краткосрочной памятью (англ. *Long Short-Term Memory, LSTM*).

Решение с помощью LSTM использует карусель с постоянными ошибками (англ. *Constant Error Carousel, CEC*), которые обеспечивают постоянный поток ошибок (необходимый для хранения значений ошибки для дальнейшего обучения модели) в специальных ячейках. Доступ к ячейкам (и их применение) осуществляется мультипликативными блоками ворот (англ. *gate units*), которые

учатся своевременно предоставлять этот доступ. СЕС являются центральной функцией LSTM, где осуществляется хранение краткосрочной памяти в течение длительных периодов времени. В ходе выполнения обработки соединений между другими блоками сети LSTM может также возникнуть конфликт обновления веса. Входные соединения некоторого нейрона u могут конфликтовать в процессе обновления веса по причине того, что один и тот же вес может как использоваться для хранения некоторого входного значения, так и не использоваться. Для взвешенных выходов соединений, идущих от нейрона u , одинаковые веса могут вернуть содержимое u и сделать поток вывода ошибки в другие нейроны сети некорректным. Эту проблему решает расширение СЕС входными и выходными блоками ворот, которые соединены с входным слоем сети и с другими ячейками памяти, что ведет к формированию особого блока LSTM, называемого блоком памяти (англ. Memory Block).

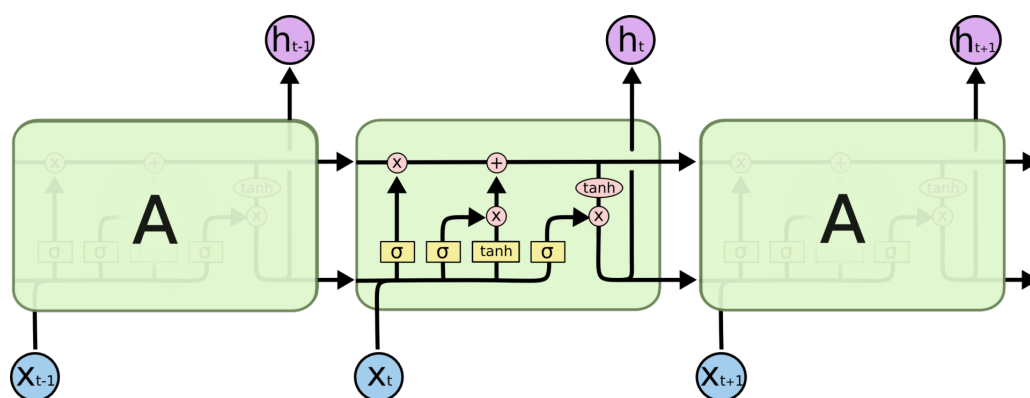


Рисунок 3 – Схема содержимого модуля LSTM-сети

РНС — это такой тип архитектуры нейронной сети, который преимущественно используется для нахождения закономерностей и шаблонов (англ. pattern) в последовательностях данных. Такими данными могут быть рукописи, представления геномов, текст или числовые временные ряды, часто используемые в рамках корпоративных задач машинного обучения (например, даны показатели курса некоторой акции или валюты, и требуется предсказать значение стоимости акции в следующий момент времени, или представлены значения сенсоров в течении некоторого временного промежутка и необходимо осуществить классификацию поведения этого сенсора). В общем смысле, RNN применяются в моделировании языка (англ. Language Modelling) и генерация текста, распознавании речи, генерации описания к изображениям (не только текстового, но и по возможным другим параметрам) или маркировке видео (англ. Video

Tagging).

В свою очередь, LSTM-сети (как подкласс сетей RNN) могут применяться в тех же сферах, что и обычные рекуррентные сети. В 2012 году модификация LSTM была применена для обнаружения ключевых слов и распознавания различных видов содержимого рукописных документов (такие как текст, формула, диаграмма и рисунок). Примерно в тот же период времени с помощью сетей с долгой краткосрочной памятью осуществлялась классификация изображений высокого разрешения из базы данных ImageNet, результаты которой были значительно лучше предыдущих (без использования LSTM). В 2015 году разновидность этого класса нейросети с использованием фреймворка Sequence-to-Sequence была успешно обучена для создания предложений на простом английском языке, описывающих изображения. Также в 2015 году LSTM была объединена с глубоким иерархическим экстрактором визуальных признаков и применена к решению задач интерпретации и классификации изображений, таких как определение некоторого активного действия на картинке и генерация описания изображения/видео.

1.3 Метрики оценки качества обучения

1.4 Функции потерь и функции активации

1.5 Задачи, решаемые генерацией текстового описания к изображению с помощью нейронной сети

2 Практическая часть

2.1 Описание инструментов и библиотек программной реализации

2.2 Описание набора данных для обучения и теста

2.3 Программная реализация алгоритма

2.4 Приведение характеристик обучения и гиперпараметров

2.5 Результаты обучения

ЗАКЛЮЧЕНИЕ

Здесь будет заключение

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.

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 обучения и на основе которых формирует подпись к изображению.
    """

    def __init__(self, freq_threshold):
        """
        Функция инициализации, при котором генерируются объекты типа словарь (первый
        объект - определяет слово по 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 каждого из них индекса).

        :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 им подписям, а также 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,
        изображения и соответствующей ему подписи).

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

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 данными.

        :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 эмбединга
        :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 эмбединга
        :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 CNNtoRNNTTranslator(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 эмбединга
            :param hidden_size: количество выходных векторов LSTM
            :param vocabulary_size: размер словаря
            :param layers_num: количество слоев, составляющих LSTM
        """
        super(CNNtoRNNTTranslator, 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 словаря, 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 заданными преобразованиями изображений
    train_loader, dataset = get_loader(
        root_folder="archive/images",
        annotation_file="archive/training_captions.txt",
        transform=transform,
        num_workers=6,
    )

    # обучение будет происходить 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()
```