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

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 [1].

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 [2].

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 [3].

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.

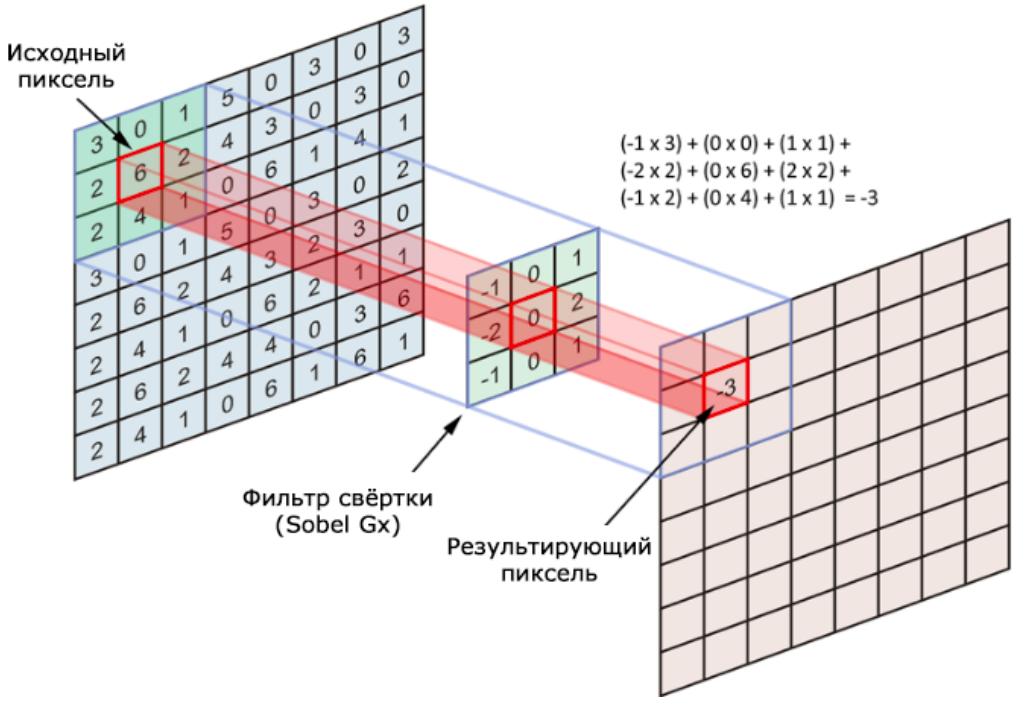

Рисунок 1 – Операция свёртки

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 [4].

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-

странственным 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 [5].

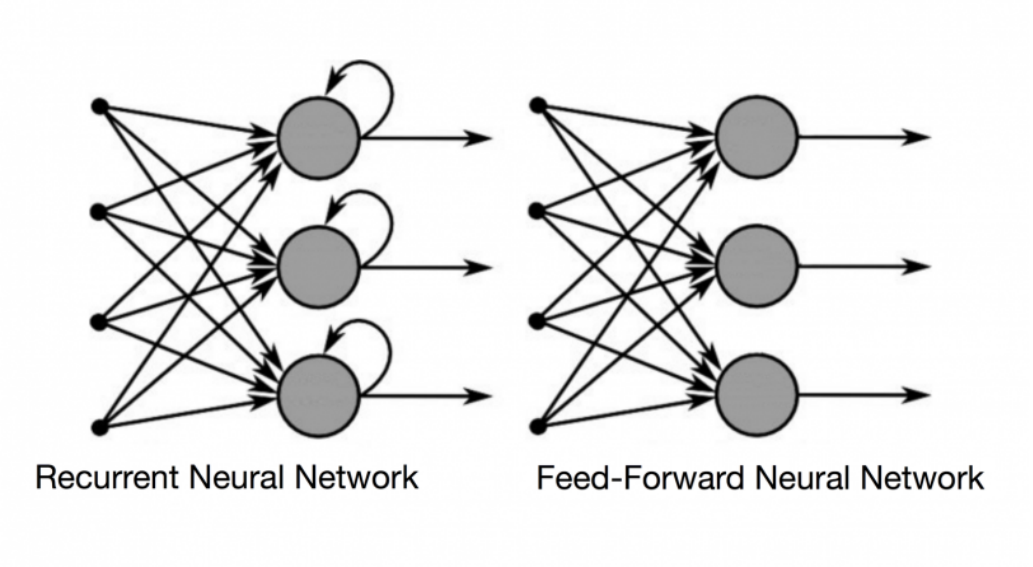

Рисунок 2 – Абстрактное сравнение архитектур FNN и RNN

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. Время от 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 долговременным 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. CEC являются центральной функцией LSTM, где осуществляется хранение краткосрочной памяти в течении 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 CEC 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) [7].

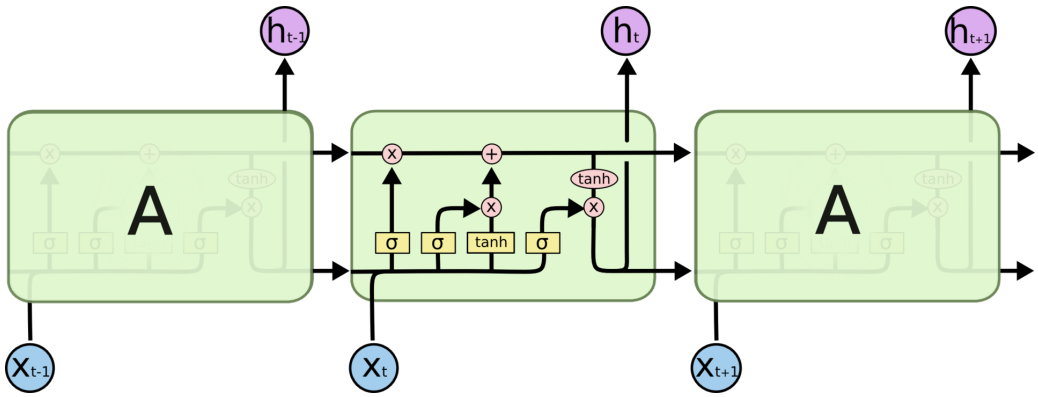

Рисунок 3 – Схема содержимого модуля LSTM-сети

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) [6].

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 [8].

1.3 Метрики оценки качества обучения

Как известно, машинное обучение — это такая область информационных технологий, которая рассматривает процессы обучения нейросетей в целях достижения высокоуровневого познания и проведения человекоподобного анализа различных задач. Поскольку ML подразумевает использование данных в процессе обучения, это прекрасно вписывается в повседневную жизнедеятельность человечества, так же как и в комплексные и междисциплинарные деятельности. С ростом популярности инструментов машинного обучения, подразумевающих коммерциализацию, открытый исходный код или ориентацию на конечного пользователя, при использовании ИИ в той или иной задаче всегда возникает закономерный вопрос — какие факторы определяют хорошую модель? Правильный ответ на этот вопрос зависит от совокупности факторов, которые для каждой определенной модели могут определяться по-своему [9].

Одним из факторов, определяющих, каким образом следует оценивать модель, является то, какой тип задач решается этой моделью, среди которых можно выделить самые популярные, а именно:

1. Регрессия — прогноз на основе выборке объектов с различными значениями конечного числа признаков. В качестве результата модель для опре-

деленного набора характеристик, определяющихся этими признаками, должна вернуть вещественное число (показатель целевого признака). В качестве примера можно взять прогноз на цену дома с учетом параметров самого дома (площадь, местоположение, средняя стоимость отапливания и т.д.)

2. Классификация — получение категориального ответа с учетом набора характеристик конкретной сущности в наборе данных (по сути соотнесение объекта выборки к тому или иному классу, где сами классы определяются целевой переменной). Примеры: есть ли на фотографии человек, какой породы собака с такими признаками.
3. Кластеризация — распределение данных на группы (разделение всех клиентов мобильного оператора по уровню платёжеспособности, отнесение космических объектов к той или иной категории: планета, звезда, чёрная дыра и т. п.).
4. Уменьшение размерности — преобразование числа признаков из большого в меньшее.
5. Выявление аномалий — отделение аномальных значений целевого признака сущностей от нормальных.

Простейшая метрика качества модели регрессии получается путем вычитания прогнозируемого значения из соответствующего ему фактического/наблюдаемого значения. Это прямолинейно, легко интерпретируемо и величина ошибки (или разницы) определяется в тех же единицах измерения, что сравниваемые величины целевого признака. Также данная метрика позволяет узнать, завышает или занижает модель свои наблюдения (путем рассмотрения знака результата оценки). Следует помнить, что может возникнуть проблема из-за противоположности знаков между прогнозом и истинным значением, заключающаяся в получении нуля в качестве результата оценки ошибки, что является некорректным и ведет к ложной интерпретации качества модели. Этого можно избежать, используя абсолютную ошибку (т. е. $|y - y'|$, где y - истинное значение, а y' - предсказание модели), которая дает только неотрицательные значения. По аналогии с традиционной ошибкой, абсолютная ошибка также поддерживает те же единицы измерения, что y и y'. Обобщая данную ошибку на всю выборку (находя её среднее значение по всему набору данных), а также модифицируя её для тех или иных задач по причине возникновения различных

частных проблем, можно выделить несколько основных метрик качества для оценки результата работы регрессионной модели:

1. Средняя абсолютная ошибка (англ. Mean Absolute Error) — показывает разницу между двумя переменными.

$$MAE = \frac{\sum_{i=1}^{n} |y - y'|}{n}$$

2. Средняя квадратичная ошибка (англ. Mean Squared Error) — показывает средний квадрат ошибки модели.

$$MSE = \frac{\sum_{i=1}^{n} (y - y')^2}{n}$$

3. Корень средней квадратичной ошибки (англ. Root Mean Squared Error) — позволяет оценить качество модели с помощью метрики MSE в тех же единицах измерения, в которых определены y и y'.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (y - y')^2}{n}}$$

Производительность классификаторов часто оценивается с помощью матрицы неточностей (англ. confusion matrix). Эта матрица содержит статистику о фактических и предсказанных классификациях и закладывает фундаментальные основы, необходимые для понимания оценки качества конкретного классификатора.

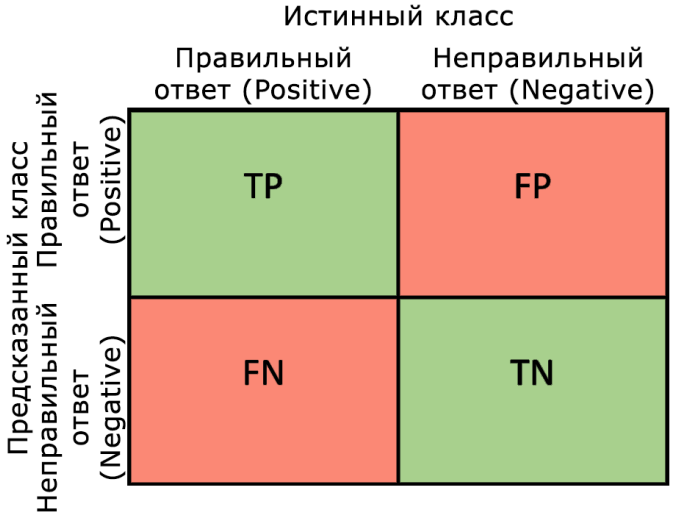

Рисунок 4 – Матрица неточностей

Каждый столбец в этой матрице означает предсказанные экземпляры, в то время как каждая строка представляет фактические экземпляры. Таким образом, можно сформировать список наиболее известных метрик для моделей, решаемых задачу классификации объектов [10]:

1. Полнота (англ. Recall, а также True Positive Rate или TPR) — это доля найденных классификатором сущностей, принадлежащих конкретному классу, относительно всех сущностей этого класса в выборке.

$$TPR = \frac{TP}{TP + FN}$$

2. Точность (англ. Precision, а также Positive Predictive Value или PPV) — это доля объектов, действительно принадлежащих данному классу относительно всех объектов, которые модель отнесла к этому классу.

$$PPV = \frac{TP}{TP + FP}$$

3. Accuracy (ACC) — представляет отношение количества верных предсказаний к общему количеству образцов выборки.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN}$$

Показатели производительности (англ. performance metrics) или же меры ошибки являются необходимыми компонентами в различных сферах применения машинного и глубокого обучения. Существует ряд исследований, направленных на анализ и классификацию метрик качества обучения модели. Наиболее важной для данной работы метрикой оценки модели является та, которая позволит оценить качество созданного текстового описания к входному изображению, сравнив полученное от нейросети описание с истинным, фактическим описанием, созданным человеком для конкретного изображения. Для того чтобы сравнивать две последовательности символов посредством некоторой метрики, необходимо предварительно каждому из символов сопоставить некоторое число, таким образом закодировав текста согласно некоторым правилам, после чего осуществить сравнение двух векторов, отождествленных с этими описаниями.

Коэффициент Отиаи или же косинусный коэффициент (англ. Cosine

similarity) осуществляет сравнение векторов, определяясь как косинус угла θ между этими двумя векторами **y** и **y'**:

$$sim(\mathbf{y}, \mathbf{y'}) = sim_{Cosine}(\mathbf{y}, \mathbf{y'}) = \frac{\langle \mathbf{y}, \mathbf{y'} \rangle}{\| \mathbf{y} \|_2 \cdot \| \mathbf{y'} \|_2} = \frac{\sum_i y_i y'_i}{\sqrt{\sum_i y_i^2} \cdot \sqrt{\sum_i y_i'^2}} = cos\theta$$

Косинусный коэффициент имеет интересные особенности, которые делают его популярным приложением в различных задачах, среди которых наиболее часто встречается анализ текста. В первую очередь, легко заметить, что $sim(x, y) = sim(\alpha x, y) = sim(x, \alpha y)$ для всех $\alpha > 0$, то есть выражение косинусного коэффициента инвариантно к масштабированию векторов с положительной скалярной величиной. В случае текстового анализа, это является наиболее часто востребованным свойством в силу того, что повторяемость содержимого документа несколько раз не меняет сущность информации этого текста [11].

1.4 Функции потерь

В математической оптимизации и теории принятия решений функция потерь или функция стоимости (иногда также называемая функцией ошибки) — это функция, которая отображает событие или значения одной или нескольких переменных в действительное число, интуитивно представляющее некоторую ”стоимость”, связанную с событием, или же ”погрешность относительно эталонного результата”. Задача оптимизации (ровно как и общий смысл обучения любой нейросети, модели машинного или глубокого обучения) направлена на минимизацию функции потерь. Искомая функция — это либо функция потерь, либо ее противоположность (в определенных областях она по-разному называется функцией вознаграждения, функцией прибыли, функцией полезности, функцией пригодности и т. д.), и в этом случае она должна быть максимизирована. В статистике обычно функция потерь используется для оценки параметров, а рассматриваемое в задаче событие является некоторой функцией разницы между оценочными и истинными значениями для экземпляра данных [12].

Алгоритмы глубокого обучения используют стохастический градиентный спуск для минимизации значения функции потерь и для того, чтобы сгенерировать значение целевого признака в качестве ответа. Чтобы узнать ответ точно и быстрее, нужно убедиться, что математическое представление ответа, также

известное как функции потерь, способно охватить даже крайние случаи. Введение функций потерь уходит корнями в традиционное машинное обучение, где эти функции потерь были получены на основе распределения меток. Например, бинарная перекрестная энтропия получена из распределения Бернулли, а категориальная перекрестная энтропия - из категориального распределения.

Перекрестная энтропийная потеря (англ. Cross-Entropy Loss) — одна из наиболее часто используемых функций потерь для обучения глубоких нейронных сетей, применение которой особенно популярно при решении задач небинарной классификации (то есть типа задачи, в которой в качестве результата необходимо получить вектор классов — предсказанных значений). Работая с категориальными данными, эта функция потерь соответствует вероятностному логарифмическому правдоподобию, что приводит к благоприятным свойствам оценки. С другой стороны, несколько известных методов современного машинного обучения занимаются подгонкой данных, которые не столько категориальны, сколько симплекснозначны — например, в такой технике регуляризации обучения, как сглаживание меток (англ. Label smoothing), и в работе с определением стратегии агента для достижения целей в обучении с подкреплением. В этих методах сообщество глубокого обучения по умолчанию заимствовало ту же кросс-энтропийную функцию потерь из задачи с категориальными данными, несмотря на то, что он больше не определяет целесообразную вероятностную модель [13].

1.5 Функции активации

В искусственных нейронных сетях каждый нейрон для своих входных данных формирует преобразованную согласно весам этого нейрона сумму и передает результирующее скалярное значение через функцию, упоминаемую как функцию активации или функцию преобразования. То есть, если нейрон имеет n входных значений вида $x_1, x_2, \ldots, x_n$, то выходные значения или активация нейрона будут выглядеть как $a = g(w_1 \cdot x_1, w_2 \cdot x_2, \ldots, w_n \cdot x_n + b)$. Функция g в данном примере является функцией активации.

Если в качестве функции g взять линейную функцию $g(z) = z$, тогда можно сказать, что нейрон выполняет задачу линейной регрессии или классификации. В общем случае g определяется как нелинейная функция для решения нелинейной регрессии и задач классификации, которые линейно неразделимы. Если функцией активации является сигмоида или s-образная функция в значе-

ниях от 0 до 1 или от -1 до 1, результат нейрона может быть интерпретирован как 'да' или 'нет' (или в виде выбора между двумя значениями).

Стоит отметить, что одной из популярных проблем, связанных с функциями активации, является проблема "насыщения". Суть её определяется состоянием, в котором нейрон преимущественно выдает в качестве результата значения, близкие к асимптотическим границам функции активации, и это может послужить причиной появления проблемы исчезающего градиента в глубоких нейросетях. Замена насыщенных сигмоидальных функций активации такими функциями, как ReLU, которые имеют большие производные значения, позволяет глубоким нейросетям быть обученными гораздо эффективнее. С тех пор появились немонотонные и осциллирующие функции активации, значительно превосходящие ReLU в качестве. В частности, осциллирующие улучшают градиентный поток, ускоряя обучение и позволяя одиночным нейронам учить XOR-функцию аналогично некоторым человеческим мозговым нейронам [15].

Функция активации "выпрямленная линейная единица", часто называемая просто ReLU (сокращение от Rectified Linear Units) имеет сильную биологическую и математическую подоплёку. В 2011 году эта функция была представлена как дальнейшее улучшение качества обучения глубоких нейронных сетей. Она работает с пороговыми значениями в нуле, т.е. $f(x) = max(0, x)$. Проще говоря, он выводит 0, когда $x < 0$, и наоборот, он выводит линейную функцию, когда $x \geq 0$ (см. рисунок ниже).

ReLU менее требовательно к вычислительным ресурсам, чем гиперболический тангенс или сигмоида, так как производит более простые математические операции. Поэтому имеет смысл использовать ReLu при создании глубоких нейронных сетей [14].

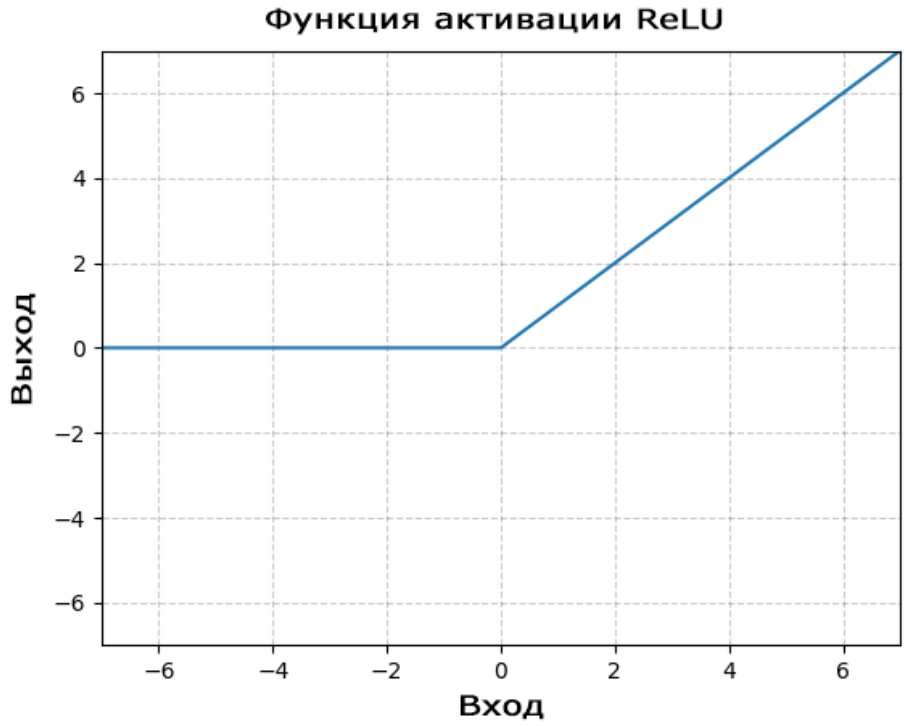

Рисунок 5 – Выпрямленная линейная единица (ReLU)

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

Технологии, применяемые для преобразования последовательности пикселей изображения в слова с помощью искусственного интеллекта, уже не такие безрезультатные, как пять или более лет назад. Более высокая производительность, точность и качество делают возможным плавную и эффективную генерацию текстового описания к изображениям в различных областях — от социальных сетей до электронной коммерции (покупки-продажи товаров с помощью цифровых ресурсов). Автоматическое создание тегов, соответствующих загруженной фотографии. Эта технология может помочь слепым людям познавать окружающий мир.

Как задача взаимодействия визуальных и языковых данных, генерация текстового описания к изображениям может быть решена с помощью компьютерного зрения и NLP. ИИ подразумевает использование совокупности CNN и RNN или любой другой подходящей модели для достижения цели.

Ниже будут рассматриваются варианты использования технологии генерации текстового описания к изображениям [16]:

1. Расстановка тегов для изображения в рамках электронной коммерции,

служб обмена фото и онлайн-каталогов. Автоматическое создание тегов для фото может упростить жизнь пользователям, когда они загружают изображение в онлайн-каталог. В этом случае ИИ распознает изображение и генерирует атрибуты — это могут быть подписи, категории товара или описания. Технология также может определять тип товара, материал, цвет, узор и посадку одежды для интернет-магазинов. В то же время текстовые описания изображений могут быть реализованы службой обмена фотографиями или любым онлайн-каталогом для создания автоматического осмысленного описания изображения в целях SEO или категоризации. Кроме того, подписи позволяют проверить, соответствует ли изображение правилам платформы, на которой оно опубликовано. Здесь они служат альтернативой категоризации с помощью CNN и помогают увеличить трафик и доход.

2. Автоматические аннотации изображений для слепых людей. Чтобы разработать такое решение, необходимо преобразовать картинку в текст, а затем в голос. Это два хорошо известных применения технологии глубокого обучения. Приложение под названием Seeing AI, разработанное Microsoft, позволяет людям с проблемами зрения видеть окружающий мир с помощью смартфонов. Программа умеет читать текст при наведении на него камеры и выдает звуковые подсказки. Он может распознавать как печатный, так и рукописный текст, а также идентифицировать предметы и людей. Google также представил инструмент, который может создавать текстовое описание изображения, позволяя слепым или людям с проблемами зрения понять контекст изображения или графики. Этот инструмент машинного обучения состоит из нескольких слоев. Первая модель распознает текст и рукописные цифры на изображении. Затем другая модель распознает простые объекты окружающего мира — машины, деревья, животных и т. д. И третий слой — это продвинутая модель, способная улавливать основную мысль в полноценном текстовом описании.
3. Подписи изображений искусственным интеллектом для социальных сетей. Подпись к изображению, созданная с помощью инструмента на основе ИИ, уже доступна для Facebook и Instagram. Кроме того, модель все время умнеет, учится распознавать новые объекты, действия и закономерности. Facebook создал систему, способную создавать альтернативные текстовые

описания почти пять лет назад. В настоящее время она стал более точным. Раньше она описывала изображение, используя общие слова, но теперь эта система может генерировать подробное описание.

Description:
long sleeve french terry sweatshirt in yellow. rib knit crew neck collar, cuff, and hem. logo printed in black at front. tonal stitching.

Ground truth
long sleeve cotton fleece sweatshirt in lime green . rib knit crew neck collar , cuff , and hem . multi color crystal cut logo at front . dropped shoulder . tonal stitching

Description:
long sleeve french terry hoodie in pink. draw string at hood. kangaroo pocket at waist. rib knit cuff and hem. tonal stitching.

Ground truth
long sleeve french terry hoodie in light pink . grey knit logo and tonal draw string at hood . dropped shoulder . rib knit cuff and hem . silver tone hardware . tonal stitching

Рисунок 6 – Пример применения модели генерации текстового описания для изображения — создание описания стиля одежды для магазина

2 Практическая часть

2.1 Описание инструментов и библиотек программной реализации

За последние несколько лет Python стал де-факто наиболее привлекательным языком для создания, обучения и работы с моделями глубокого обучения посредством таких инструментов (англ. framework), как PyTorch, TensorFlow, MxNet/TVM и JAX, которые в основном предоставляют интерфейсы Python. Простота применения Python для написания кода и возможность его распространения через такие пакетные менеджеры, как conda, упрощают обмен библиотеками и результатами исследований. Его экосистема библиотек для науки о данных, таких как NumPy, Pandas и Jupyter Notebook, упрощают анализ результатов обучения моделей и анализ нейросети и данных в целом [17].

Поскольку программная реализация напрямую подразумевает работу с изображениями, для экономии времени и гарантии качества обработки входных фото был поднят вопрос касательно выбора инструмента, позволяющего осуществлять интерпретацию и ту или иную пред- и постобработку картинки. В силу использования Python как основного языка для написания кода программной реализации, задача нахождения набора инструментов была сведена к поиску подходящей библиотеки для работы с изображениями. Для данной задачи и удовлетворения необходимых потребностей была выбрана библиотека PIL (сокращение от Python Imaging Library). PIL добавляет возможность обработки изображений в интерпретатор Python. Эта библиотека обеспечивает обширную поддержку форматов файлов, эффективное внутреннее представление и довольно мощные возможности процессинга изображений. Основная библиотека изображений предназначена для быстрого доступа к данным, хранящимся в нескольких основных форматах пикселей. Это должно обеспечить прочную основу для общего инструмента обработки изображений [18].

Так как каждому изображению в обучающей и тестовой выборке должно соответствовать как минимум одно текстовое описание, вопрос способа сопоставления текста и фото решается с помощью представления этой связи в виде таблицы, в которой в первой колонке указывается название фото или ссылка на путь к изображению, а во второй колонке записывается текстовое описание этой картинки (что является целевым признаком/переменной). Способ хранения таких данных непринципиален и основным требованием к нему является простота способа взятия данных, поэтому в качестве формата хранения таб-

лицы был выбран CSV (сокращение от Comma-Separated Values) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми. Наиболее популярным и в то же время самым эффективным способом работы с такого рода представлениями данных на языке Python является инструментарий библиотеки Pandas. Библиотека Pandas предоставляет структуры данных и функции, призванные сделать работу со структурированными данными простой, быстрой и выразительной. С момента появления в 2010 году она способствовала превращению Python в мощную и продуктивную среду анализа данных. Основные объекты Pandas — это DataFrame — двумерная таблица, в которой строки и столбцы имеют метки, и Series — объект одномерного массива с метками. В библиотеке Pandas сочетаются высокая производительность средств работы с массивами, присущая NumPy, и гибкие возможности манипулирования данными, свойственные электронным таблицам и реляционным базам данных (например, на основе SQL). Она предоставляет развитые средства индексирования, позволяющие без труда изменять форму наборов данных, формировать продольные и поперечные срезы, выполнять агрегирование и выбирать подмножества [19].

Помимо обработки изображений и табличных данных, необходимо осуществлять корректную обработку самих текстов, подготовленных для обучения и тестирования модели, и контролировать генерацию текстовых описаний посредством сильного инструмента. SpaCy обеспечивает надежную архитектуру для создания и совместного использования пользовательских высокопроизводительных конвейеров NLP (англ. pipeline), принимая объектно-ориентированное представление текста. Он не деструктивен, поддерживает плавную интеграцию статистических методов и методов машинного обучения с NLP, построенном на основе правил, и позволяет создавать пользовательские компоненты для специализированных задач. Благодаря возможностям Cython, оптимизирующего статического компилятора для Python, который генерирует очень эффективный код C или C++, spaCy позволяет достичь исключительной скорости работы. Подобно подходу UIMA в Java, spaCy предоставляет основу для модульного построения настраиваемых конвейеров NLP по принципу plug-and-play. С момента своего создания в 2015 году, библиотека spaCy приобрела сильное, очень активное и растущее сообщество разработчиков модулей с открытым исходным

кодом, современными моделями и комплексными системами, разработанными с помощью платформы или для неё самой [20].

Фреймворки глубокого обучения часто фокусируются либо на удобстве своего использования, либо на скорости работы. PyTorch — это библиотека машинного обучения, которая показывает, что эти две цели на самом деле совместимы: она обеспечивает императивный и Python-подобный стиль программирования, который поддерживает код как модель, упрощает отладку и согласуется с другими научно-популярными вычислительными библиотеками, оставаясь при этом эффективным и поддерживая аппаратные ускорители, такие как графические процессоры. Популярность PyTorch связана с объединением технических идей реализации библиотеки с дизайном, который сочетает в себе баланс скорости и лёгкости использования. В основе дизайна лежат четыре основных принципа [21]:

1. Быть Python-подобным. Подавляющее большинство специалистов по анализу данных знакомо с языком Python, его подходами к программированию и инструментами. PyTorch должен быть первоклассным членом этой экосистемы. Это ведет к общепринятым идеям дизайна — сделать интерфейс взаимодействия простым и согласованным, в идеале с одним идиоматическим способом создания решений задач машинного обучения. Он также естественным образом интегрируется со стандартными инструментами построения графиков, отладки и обработки данных.
2. Первостепенная важность исследователей. PyTorch стремится сделать написание моделей, загрузчиков данных и оптимизаторов как можно легче и продуктивнее. Сложность, присущая машинному обучению, должна быть решена внутренне библиотекой PyTorch и скрыта за интуитивно понятным API без побочных эффектов и неожиданных нюансов в производительности.
3. Обеспечение прагматичной производительности. Чтобы быть полезным, PyTorch должен обеспечивать существенную производительность, хотя и в небольшой ущерб простоте и удобству использования. Обменять 10% скорости на значительно упрощенную реализацию использования модели вполне приемлемо, но терять 100% быстродействия для этого — недопустимо. Поэтому библиотека допускает добавление сложности для обеспечения этой производительности. Кроме того, предоставление инстру-

ментов, позволяющих исследователям вручную управлять выполнением своего кода, позволит им найти собственную планку производительности, независимую от тех, которые библиотека предоставляет автоматически.

4. Чем хуже, тем лучше. При фиксированном количестве инженерных ресурсов и прочих равных условиях, время, сэкономленное за счет упрощения внутренней реализации PyTorch, можно использовать для реализации дополнительных функций, адаптирования к новым ситуациям, таким образом идя в ногу с быстрыми темпами прогресса в области ИИ. Поэтому лучше иметь простое, но немного неполное решение, чем всеобъемлющую, но сложную и неудобную в обслуживании конструкцию.

2.2 Описание набора данных для обучения и теста

2.3 Программная реализация алгоритма

Генератор подписей к входным изображениям будет представлять собой ансамбль из двух нейросетей - CNN и LSTM.

Идея работы ансамбля заключается в том, что на вход к CNN подается цветное изображение (размеры которого предварительно сделали 224 на 224 пикселя). В качестве CNN будет использоваться модель GoogleNetv3, предобученная на датасете 2015-го года ImageNet. Получая на вход изображение в виде тензора, она будет пропускать его через все свои слои, и из последнего полностью соединенного слоя нейросети будет браться некоторый вектор, который впоследствии преобразуется с помощью линейного слоя. Таким образом, CNN в данной архитектуре будет являться своего рода кодировщиком, выдавая для каждого конкретного изображения некоторый вектор.

Полученный обработанный вектор отправляется на вход нейросети LSTM, которая (с учетом определенного словаря, количества lstm-слоев и прочих гиперпараметров) будет создавать последовательные связанные друг с другом слова. Это делает LSTM-модель своего рода декодировщиком, результатом деятельности которого будет подпись к изображению.

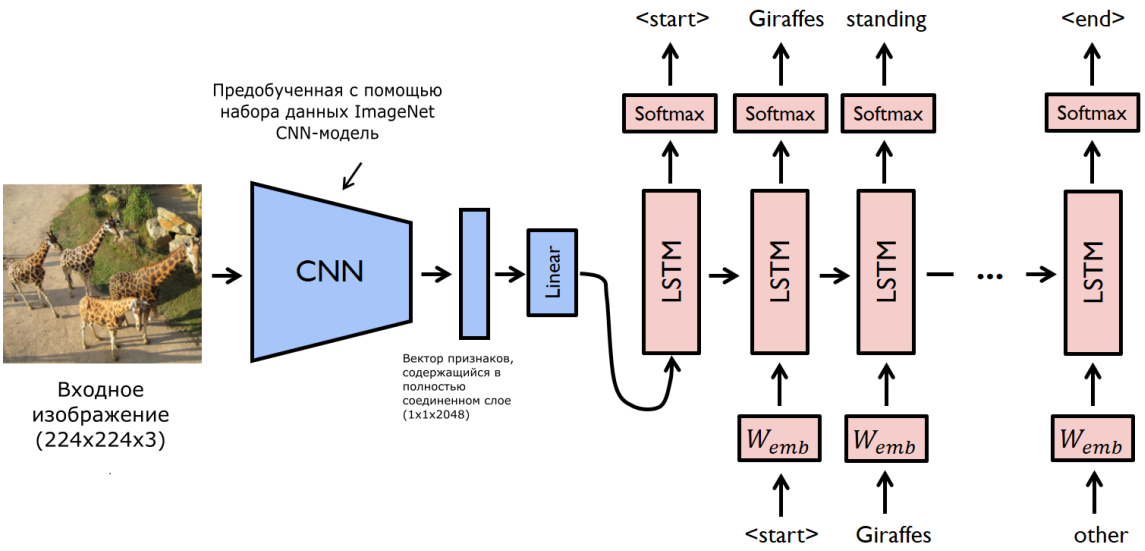

Рисунок 7 – Архитектура программы

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 27.04.2022) Загл. с экрана. Яз. рус.

2 Гудфеллоу Я., Бенджио И., Курвилль А., ”Глубокое обучение”, г. Москва, Издательство ДМК, 2018 г., Яз. рус.

3 Keiron O’Shea, Ryan Nash, ”An Introduction to Convolutional Neural Networks”, [Электронный ресурс] : [статья] / URL https://arxiv.org/abs/1511.08458 (дата обращения 14.04.2022) Загл. с экрана. Яз. англ.

4 Daphne Cornelisse, ”An intuitive guide to Convolutional Neural Networks”, [Электронный ресурс] : [статья] / URL https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/ (дата обращения 14.04.2022) Загл. с экрана. Яз. англ.

5 Robin M. Schmidt, ”Recurrent Neural Networks (RNNs): A gentle Introduction and Overview”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1912.05911.pdf (дата обращения 18.04.2022) Загл. с экрана. Яз. англ.

6 IBM Cloud Education, ”Recurrent Neural Networks”, [Электронный ресурс] : [статья] / URL https://www.ibm.com/cloud/learn/recurrent-neural-networks (дата обращения 18.04.2022) Загл. с экрана. Яз. англ.

7 Ralf C. Staudemeyer, Eric Rothstein Morris, ”Understanding LSTM — a tutorial into Long Short-Term Memory Recurrent Neural Networks ”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1909.09586.pdf (дата обращения 22.04.2022) Загл. с экрана. Яз. англ.

8 Wunder Fund, ”LSTM – сети долгой краткосрочной памяти”, [Электронный ресурс] : [статья] / URL https://habr.com/ru/company/wunderfund/blog/331310/ (дата обращения 22.04.2022) Загл. с экрана. Яз. рус.

9 Alexei Botchkarev, ”Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics:

Properties and Typology”, [Электронный ресурс] : [статья] / URL https://arxiv.org/ftp/arxiv/papers/1809/1809.03006.pdf (дата обращения 24.04.2022) Загл. с экрана. Яз. англ.

10 M.Z. Naser, Amir H. Alavi, ”Insights into Performance Fitness and Error Metrics for Machine Learning”, [Электронный ресурс] : [статья] / URL https://arxiv.org/ftp/arxiv/papers/2006/2006.00887.pdf (дата обращения 24.04.2022) Загл. с экрана. Яз. англ.

11 Stijn Van dongen, Anton J. Enright, ”METRIC DISTANCES DERIVED FROM COSINE SIMILARITY AND PEARSON AND SPEARMAN CORRELATIONS”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1208.3145.pdf (дата обращения 24.04.2022) Загл. с экрана. Яз. англ.

12 Elliott Gordon-Rodriguez, Gabriel Loaiza-Ganem и т.д., ”Uses and Abuses of the Cross-Entropy Loss: Case Studies in Modern Deep Learning”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/2011.05231.pdf (дата обращения 27.04.2022) Загл. с экрана. Яз. англ.

13 Shruti Jadon, ”A survey of loss functions for semantic segmentation”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/2006.14822.pdf (дата обращения 27.04.2022) Загл. с экрана. Яз. англ.

14 Abien Fred M. Agarap, ”Deep Learning using Rectified Linear Units (ReLU)”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1803.08375.pdf (дата обращения 28.04.2022) Загл. с экрана. Яз. англ.

15 Pierre Stock, Remi Gribonval, ”AN EMBEDDING OF RELU NETWORKS AND AN ANALYSIS OF THEIR IDENTIFIABILITY”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/2107.09370.pdf (дата обращения 28.04.2022) Загл. с экрана. Яз. англ.

16 Diana Malyk, ”Exploring Deep Learning Image Captioning”, [Электронный ресурс] : [статья] / URL https://mobidev.biz/blog/exploring-deep-learning-image-captioning (дата обращения 30.04.2022) Загл. с экрана. Яз. англ.

17 Zachary DeVito, Jason Ansel и т.д., ”USING PYTHON FOR MODEL INFERENCE IN DEEP LEARNING”, [Электронный ресурс] : [статья] / URL

https://arxiv.org/pdf/2104.00254.pdf (дата обращения 2.05.2022) Загл. с экрана. Яз. англ.

18 Alex Clark и т.д., ”Pillow”, [Электронный ресурс] : [статья] / URL https://pillow.readthedocs.io/en/stable/ (дата обращения 2.05.2022) Загл. с экрана. Яз. англ.

19 Маккинни У., ”Python и анализ данных”, ДМК Пресс, 2015. — 482 с. — ISBN 978-5-97060-315-4, 978-1-449-31979-3.

20 Hannah Eyre, Alec B Chapman и т.д., ”Launching into clinical space with medspaCy: a new clinical text processing toolkit in Python”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/2106.07799.pdf (дата обращения 2.05.2022) Загл. с экрана. Яз. англ.

21 Adam Paszke, Sam Gross и т.д., ”PyTorch: An Imperative Style, High-Performance Deep Learning Library”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1912.01703.pdf (дата обращения 2.05.2022) Загл. с экрана. Яз. англ.

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 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)
```

```python
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 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()
```