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, 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 компьютерное зрение и обработка 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].

Особенностью сверточной нейронной сети является наличие в ней 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). Результат работы свертки в простейшем случае (определяемый как $(N, C_{out}, H_{out}, W_{out})$), с входными данными в формате (N, C_{in}, H, W) можно выразить в виде следующей формулы [4]:

$$out(N_i, C_{out_j}) = bias(C_{out_j}) + \sum_{k=0}^{C_{in}-1} weight(C_{out_j}, k) \star input(N_i, k),$$

где $\star$ — оператор взаимнокорреляционной функции, N — размер тренировочной выборки, используемой в одной итерации (англ. batch size), C определяет количество каналов, H это высота входных плоскостей в пикселях, а W — ширина этих плоскостей в пикселях. H_{out} и W_{out} определяются следующим образом:

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor;$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor,$$

где $stride$ определяет шаг для взаимной корреляции, является одним числом или списком чисел; $padding$ управляет количеством ”отступов”, применяемых к входным данным. Это может быть либо строка {’новое значение’, ’старое значение’}, либо список целых чисел, указывающий количество неявного заполнения, применяемого с обеих сторон; $dilation$, также известный как алгоритм à trous, задает расстояние между точками ядра в процессе свертки; $kernel\_size$ — размер ядра.

Визуальная интерпретация операции свертки представлена на изображении ниже.

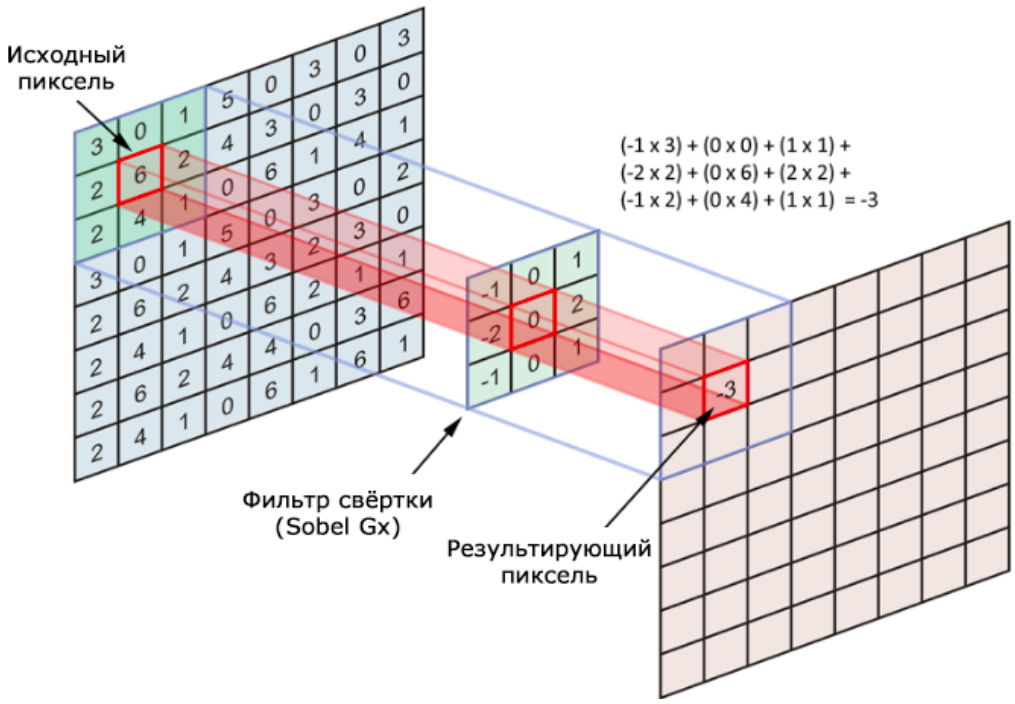

Рисунок 1 – Операция свёртки [5]

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 [6].

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 пространственным деформациям:

1. локальные рецепторные поля (обеспечивают локальную двумерную связ-

ность 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 [7].

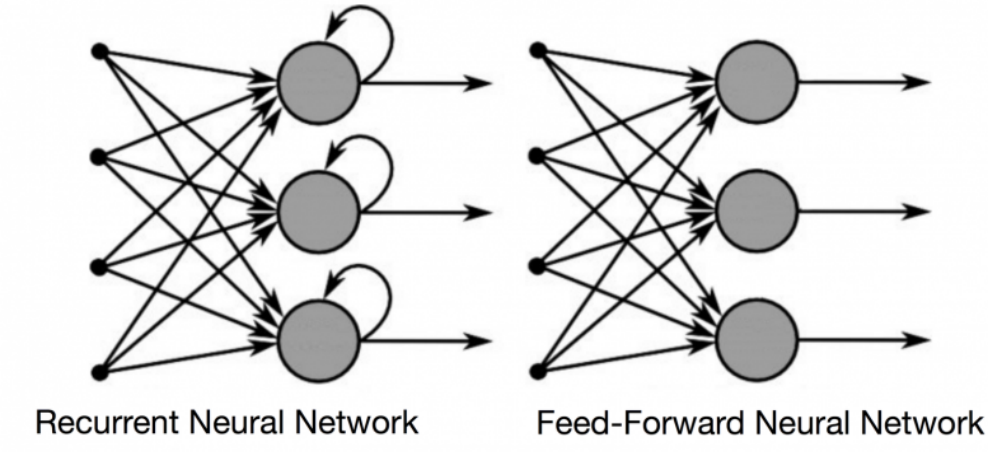

Рисунок 2 – Абстрактное сравнение архитектур FNN и RNN [8]

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) [10].

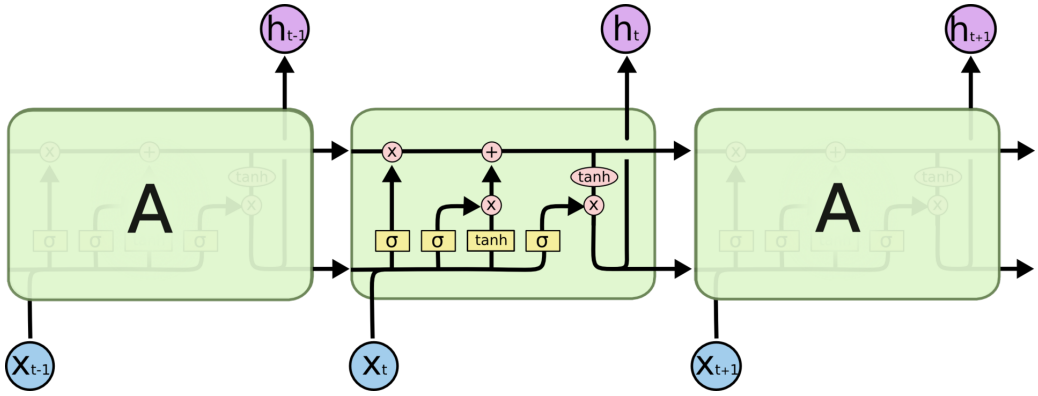

Рисунок 3 – Схема содержимого модуля LSTM-сети [11]

Таким образом, для каждого элемента входной последовательности каждый t-ый слой LSTM модели осуществляет следующие вычисления [12]:

$$i_t = \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi});$$

$$f_t = \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf});$$

$$g_t = \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg});$$

$$o_t = \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho});$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t;$$

$$h_t = o_t \odot \tanh(c_t),$$

где h_t, c_t, x_t — скрытое состояние слоя модели, состояние клетки и входной параметр в момент времени t соответственно; h_{t-1} определяет скрытое состояние слоя в момент времени $t-1$ или начальное скрытое состояние в момент времени o. Элементы i_t, f_t, g_t, o_t являются входными, забывающими, клеточными и выходными воротами соответственно. Символ σ определяет функцию сигмоиды, $\odot$ — поэлементное произведение, также называемое произведением Адамара.

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) [9].

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 [11].

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 мо-

дель? Правильный ответ на этот вопрос зависит от совокупности факторов, которые для каждой определенной модели могут определяться по-своему [13].

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 определенного 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.

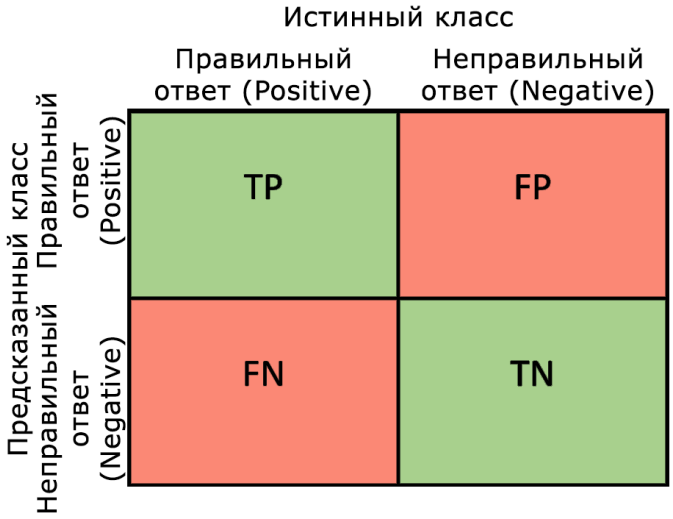

Рисунок 4 – Матрица неточностей [14]

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 [15]:

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 приме-

нения 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 [16].

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 [17].

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 [18]. Определить перекрестную энтропийную функцию потерь, используемую в практической части данной работы, можно следующей формулой [19]:

$$l(x, y) = \sum_{n=1}^{N} \frac{1}{\sum_{n=1}^{N} w_{y_n} \cdot 1\{y_n \neq ignore\_index\}} l_n,$$

где x и y — это входные и целевые значения, веса заданы как w, а N определяет размер подмножества тренировочной выборки, используемой в одной итерации (англ. minibatch size). Обозначение $ignore\_index$ является целевым значением, которое игнорируется и не влияет на входной градиент.

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 значениях 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 [21].

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 [20].

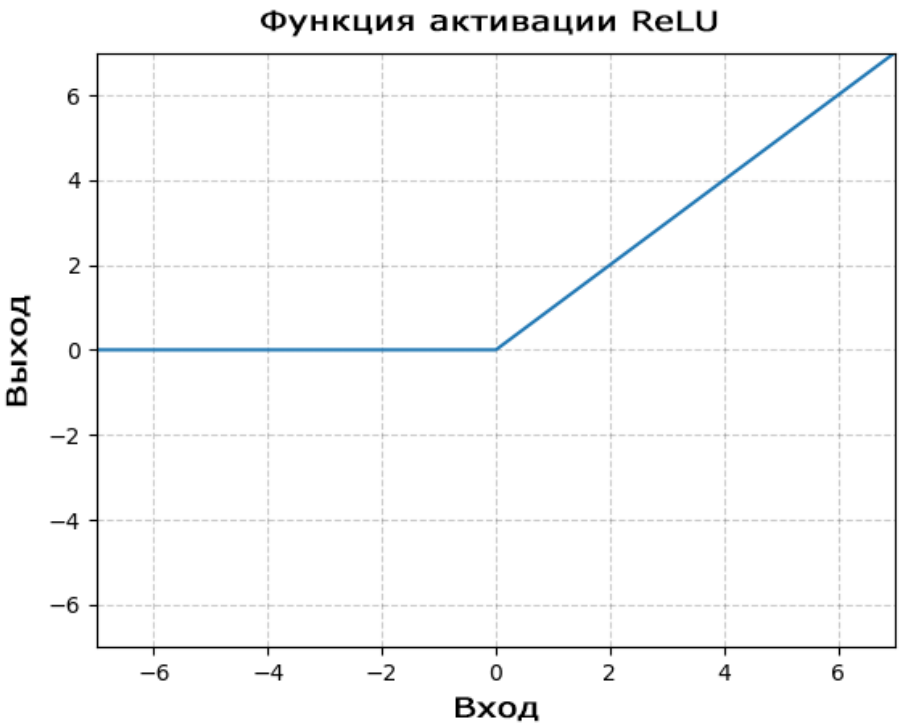

Рисунок 5 – Выпрямленная линейная единица (ReLU) [22]

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 соци-

альных 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 [23]:

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. Пример применения модели генерации текстового описания для магазина одежды представлен на изображении ниже.
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.

Рисунок 6 – Создание описания стиля одежды для магазина с помощью нейросети [24]

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 TensorFlow, PyTorch, MxNet/TVM и JAX, которые в основном предоставляют интерфейсы Python [38], [39], [40], [41], [42]. 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 [25], [43], [44], [45].

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) [26]. 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.

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 таблицы 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 [27].

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 (англ. pipeline) NLP, принимая объектно-ориентированное представление текста [28]. 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.

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 [29]:

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 обес-

печения этой производительности. Кроме того, предоставление инструментов, 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

Осуществление предобработки и принятие решения о выборе того или иного набора данных для конкретной задачи машинного или глубокого обучения является столь же важным этапом деятельности создания ИИ, как и проектирование архитектуры модели и сопровождение процесса обучения.

В данной работе осуществляется решение задачи генерации текстового описания к изображению — следовательно, рациональнее и эффективнее следует взять в качестве основного ресурса для обучения нейросети один из таких наборов данных, как ”Microsoft COCO”, ”Visual Genome”, ”TextCaps”, ”Flickr 8k” и т.п. В качестве основополагающих данных необязательно брать большой объем изображений с их текстовым описанием — для проверки качества реализации нейросети достаточно выбрать умеренное количество информации, и в связи с этим было принято решение выбрать набор данных ”Flickr 8k” (то есть исходя из других различных исследовательских работ, связанных с генерацией текстового описания к изображению) [30], [31].

Набор данных находится в открытом доступе и его можно загрузить как с официального сайта фотохостинга, так и с других различных форумов и сайтов для машинного и глубокого обучения. В частности, для обучения модели, описывающейся в данной работе, такая коллекция совместно с файлом формата CSV, в котором осуществляется сопоставление названия каждого изображения с одним из его текстовых описаний, была взята с платформы Kaggle [32].

Набор представляет собой эталонную коллекцию для генерации текстового описания и поиска изображений на основе предложений на естественном

языке, которая состоит из более чем 8000 изображений, каждое из которых имеет пять разных текстовых подписей, которые обеспечивают четкое описание фигурирующих на конкретном изображении объектов и событий. Изображения были выбраны из шести разных групп Flickr и, как правило, не содержат каких-либо известных людей или мест, а были отобраны вручную для отображения различных ситуаций.

Эти данные будут в дальнейшем поделены на тренировочную и тестовую выборку, где первая выборка будет представлять собой 80% от всего набора данных, а тестовая — всё остальное.

Пример части содержимого CSV-файла:

```
image,caption
1000268201_693b08cb0e.jpg,A girl going into a wooden building
1001773457_577c3a7d70.jpg,A black dog and a spotted dog are fighting
```

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 сведены к 356 пикселя в ширину и 356 пикселя в высоту). В качестве CNN будет использоваться модель GoogleNetv3, предобученная на датасете 2015-го года ImageNet.

Предобученная модель основана на исследовании способов масштабирования сетей таким образом, чтобы максимально эффективно использовать дополнительные вычисления за счет подходящей факторизованной свертки и агрессивной регуляризации. В официальной документации библиотеки PyTorch можно посмотреть статистику работы данной нейросети с набором тестовых задач классификации ILSVRC 2012, которые демонстрируют хорошее качество модели [34].

Получая на вход изображение в виде тензора, она будет пропускать его через все свои слои, и из последнего полностью соединенного слоя нейросети будет выдавать 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.

LSTM-модель обучена предсказывать каждое слово предложения после получения входного изображения таким же образом как и предшествующие слова (это можно определить как $P(S_t|I, S_0, S_1, ..., S_{t-1})$). Для этого следует определить работу LSTM в развернутом виде. Копия памяти LSTM для изображения, а также каждое слово предложения (такое, которое определяет для всех LSTM-модулей одни и те же параметры) и выход m_{t-1} модуля LSTM в момент времени $t-1$ подаются на вход к модулю LSTM в момент времени t. В развернутом виде все рекуррентные соединения преобразуются в соединения прямой связи. Если рассматривать более детально, то если определить I как входное изображение, для которого задано исходное текстовое описание в виде $S = S_0, ..., S_N$, то процедура развертывания выглядит следующим образом:

$$X_{-1} = CNN(I);$$

$$x_t = W_e S_t, t \in \{0...N-1\};$$

$$p_{t+1} = LSTM(x_t), t \in \{0...N-1\},$$

где каждое слово представляется унитарно закодированным вектором (англ. one-hot vector) размерности, равной размеру словаря. Следует отметить, что в качестве S_0 задается токен, определяющий начало предложения, а в качестве S_N — токен, определяющий конец предложения. В частности, с помощью токена конца предложения LSTM сигнализирует о том, что текстовое описание было полностью сгенерировано. И изображение, и слова сопоставляются к одному пространству: изображение с помощью CNN, слова с помощью эмбэддинга слов W_e. Изображение I подается на вход только один раз, в момент времени $t = -1$, чтобы передать информацию модулю LSTM о своём содержании [30].

Это делает LSTM-модель своего рода декодировщиком, результатом деятельности которого будет подпись к изображению.

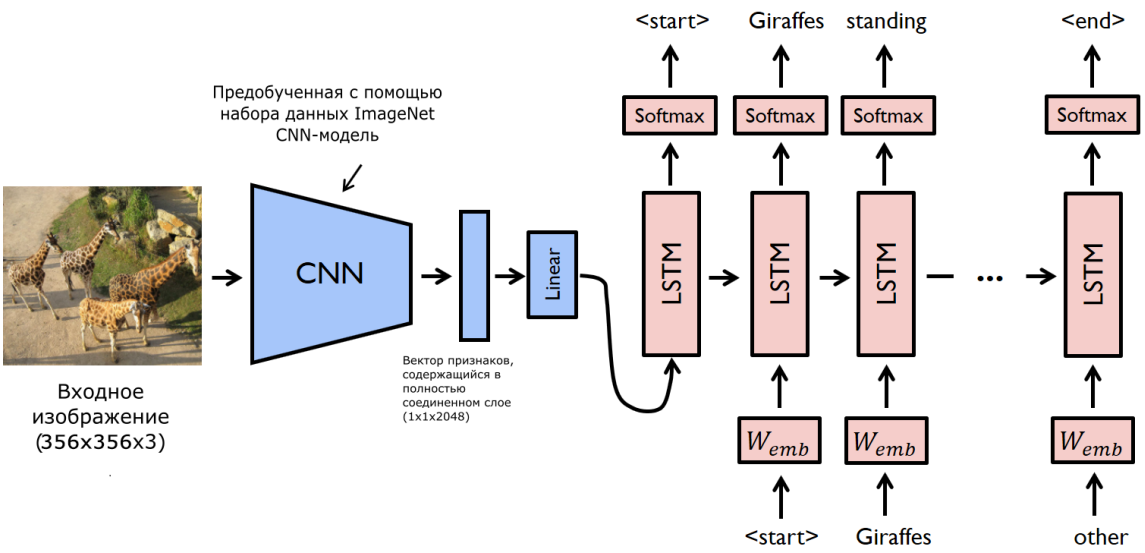

Рисунок 7 – Архитектура программы [33]

2.4 Приведение характеристик обучения и гиперпараметров

В большинстве случаев применения, размер эмбэддинга (англ. embedding size) определяется эмпирическим путем, через ошибки и в процессе улучшения качества обучения модели. Старые работы на тему NLP использовали значение 300. Более свежие работы используют размер, равный 512, 768 и 1024. Одним из факторов, влияющих на выбор размера является то, каким образом предпочтительнее осуществлять корреляцию различных векторов друг с другом. В пространстве высокой размерности с вероятностью 1, выбранные случайным образом векторы будут приблизительно взаимно ортогональны. В то время как при малых размерностях и в случае множества различных классов, многие векторы будут иметь скалярное произведение, значение которого значительно отличается от нуля. Если ожидать, что многие векторы должны быть коррелированными, размерность не должна быть очень высокой. В противном случае, если ожидается, что каждый из возможных ключей эмбэддинга будет создавать другой, несвязанный вектор, тогда размерность следует выбрать большой. В качестве изначального значения размера эмбэддинга было выбрано число 256, аналогично количеству выходных векторов LSTM [35].

Размер словаря, с помощью которого будет формироваться текстовое описание для каждого изображения, определяется напрямую количеством уникальных слов, найденных в CSV-файле набора данных, а также NLP токенами типа ”начало предложения”, ”конец предложения” и т.д.

Начальное количество слоев LSTM (то есть количество идущих подряд модулей LSTM) задано числом 2, и при необходимости может установиться с иным значением.

Говоря о темпе обучения (англ. learning rate), разработчики Tensorflow, Pytorch и другие в своей документации рекомендуют установить значение темпа обучения равным 0.001. Однако в решении университета Станфорд задачи генерации текстового описания начальное значение этого параметра для LSTM-сети было выбрано 0.0004, что также послужило основной для определения темпа обучения сети-кодировщика в данной работе [36].

Количество эпох обучения определено значением 100.

2.5 Результаты обучения

Для того, чтобы проверить качество обучения модели (то есть проконтролировать отсутствие переобучения и недообучения, вследствие которых модель может работать плохо, согласно оцениваемой метрике), в процессе обучения сохранялись значения функции ошибки согласно конкретному моменту времени. Используя эти параметры, можно построить график зависимости перекрестной энтропийной потери от времени (называемой кривой обучения), чтобы проследить, уменьшается ли значение ошибки с каждой итерацией или нет. Ниже представлено изображение, где вдоль оси OY обозначены значения функции потерь, а вдоль оси OX — номера итерации. На этом графике бледной линией показано изменение значения функции потерь с увеличением количества итераций, а жирной линией представлено изменение экспоненциального скользящего среднего этого значения, сглаживающая константа которого ≈ 0.94. В конце графика отображено несколько жирных линий — это связано с тем, что модель на последних эпохах обучения была переучена, и из-за этого значения номеров итераций при различных показателях функции потерь совпадают. Исходя из представленного изображения, на качество работы модели процесс переучивания не привел к существенному различию между результатами.

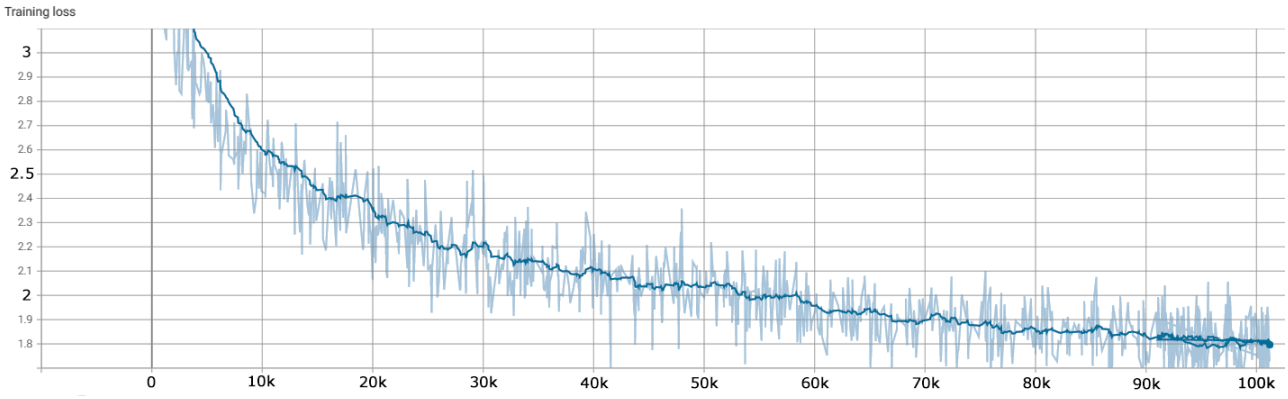

Рисунок 8 – График кривой обучения

Согласно графику можно сказать, что процесс обучения проходил без каких-либо сложностей, и признаков недообучения и переобучения на данном графике не обнаружено [37].

Для осуществления оценки работы модели с помощью выбранной ранее метрики, предварительно следует подчеркнуть создание функции, которая получает на вход изображение, и генерирует с помощью созданной нейросети текстовое описание на английском языке.

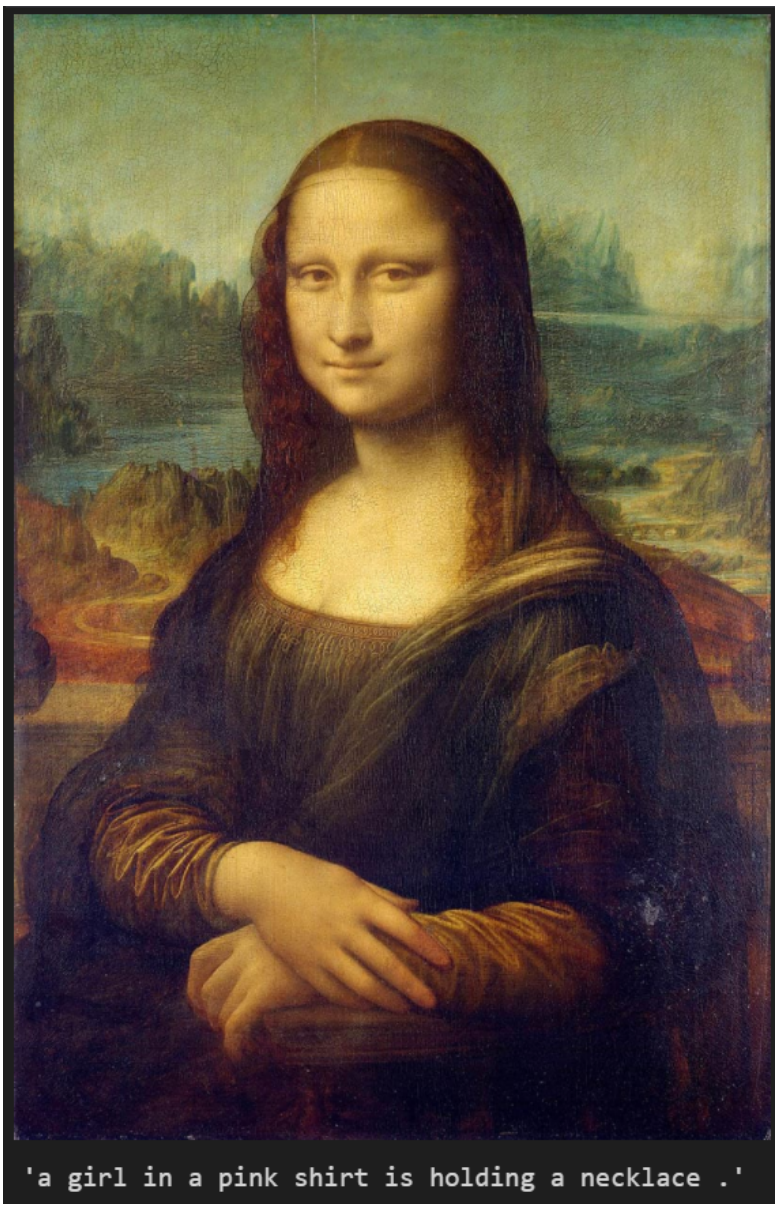

Рисунок 9 – Пример генерации текстового описания с помощью нейросети, созданной согласно данной работе

Используя методы библиотеки NumPy, можно создать метрику, упомянутую ранее как косинусный коэффициент, с помощью которой можно осуществить проверку качества модели. Для этого была написана функция, которая будет получать в качестве аргумента тестовую выборку, и с помощью модели генерировать предсказания, которые затем будут использоваться метрикой для оценки посредством применения косинусного коэффициента к предсказанию и эталонному текстовому описанию. Затем следует взять среднее значение от результата работы метрики для всех изображений, которое и будет являться средним значением качества данной модели.

Эта средняя оценка будет представлена действительным числом, которое

находится в диапазоне значений от 0 до 1 (чем ближе значение к 1, тем лучше качество). Однако у человека, который не понимает принципов работы данной оценки, будут возникать проблемы понимания того, хороша ли модель или нет. Поэтому функция выше была дополнена возможностью, которая интерпретирует значение оценки в более понятный вид разделением диапазона значений на 4 промежутка. Это означает, что:

1. Значение метрики от 0 до 0.25 приведет к возврату данной функцией строки ”Модель работает плохо, не способна даже в общих чертах описать то, что присутствует на изображении.”;
2. Значение от 0.25 до 0.5 — ”Модель работает удовлетворительно — способна различать и корректно интерпретировать основные детали изображения, но может допускать некоторые логические и смысловые ошибки в описании.”;
3. от 0.5 до 0.75 — ”Модель работает хорошо — почти всегда корректно описывает основные детали изображения, но способна допускать незначительные логические ошибки.”;
4. от 0.75 — ”Модель работает отлично, количество ошибок при генерации текстового описания к входному изображению сведено к минимуму.”

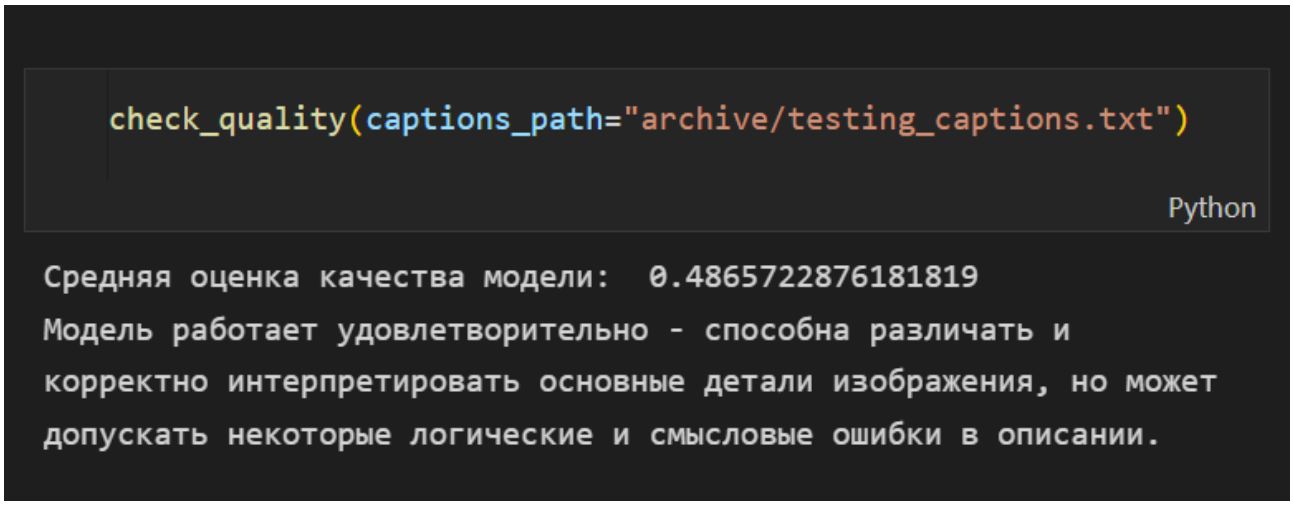

Рисунок 10 – Вывод действительного и интерпретированного значения метрики

ЗАКЛЮЧЕНИЕ

В данной работе был рассмотрен метод решения задачи генерации текстового описания к изображению путём построения ансамбля из двух видов нейросетей. В связи с этим, предварительно были рассмотрены теоретические понятия искусственных, сверточных, рекуррентных нейросетей, описаны основные принципы работы этих видов сетей и их роль в решении. Также были приведены описания таких терминов, как функции потерь, функции активации, метрики оценки качества обучения, перечислены их виды (в том числе описаны те, что применяются конкретно в этой задаче) и указана их важная роль в науке о машинном и глубоком обучении.

В качестве практической части работы была описана программная реализация алгоритма генерации текстового описания к изображению и рассмотрена архитектура ансамбля используемых нейросетей. Согласно обозначенным гиперпараметрам, было осуществлено обучение модели, результаты которого также были представлены посредством графиков кривой обучения, демонстрации генерации текстового описания к изображению и вычислению среднего значения выбранной метрики. Качество работы ансамбля, оцененное метрикой, можно считать средним, но вместе с тем доступным к улучшению с помощью подбора гиперпараметров.

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 O’Shea K., Nash R., ”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 ”Conv2D”, [Электронный ресурс] : [статья] / URL https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html#torch.nn.Conv2d (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

5 Du S., ”Understanding Deep Self-attention Mechanism in Convolution Neural Networks”, [Электронный ресурс] : [статья] / URL https://medium.com/ai-salon/understanding-deep-self-attention-mechanism-in-convolution-neural-networks-e8f9c01cb251 (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

6 Cornelisse D., ”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.

7 Schmidt R. M., ”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.

8 Donges N., ’A Guide to RNN: Understanding Recurrent Neural Networks and LSTM Networks”, [Электронный ресурс] : [статья] / URL https://builtin.com/data-science/recurrent-neural-networks-and-lstm (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

9 ”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.

10 Staudemeyer R.C., Morris E.R., ”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.

11 Fund W., ”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.

12 ”LSTM”, [Электронный ресурс] : [статья] / URL https://pytorch.org/docs/stable/generated/torch.nn.LSTM.html#torch.nn.LSTM (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

13 Botchkarev A., ”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.

14 Tandale S., ”Importance of Confusion matrix in Machine learning and Cybersecurity”, [Электронный ресурс] : [статья] / URL https://shravantandale456.medium.com/importance-of-confusion-matrix-in-machine-learning-and-cybersecurity-80e67f5858fb (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

15 Naser M.Z., Alavi A.H., ”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.

16 Enright A. J., ”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.

17 Gordon-Rodriguez E., Loaiza-Ganem G. 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.

18 Jadon S., ”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.

19 ”CrossEntropyLoss”, [Электронный ресурс] : [статья] / URL https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html #torch.nn.CrossEntropyLoss (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

20 Agarap A. M., ”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.

21 Stock P., Gribonval R., ”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.

22 ”ReLU”, [Электронный ресурс] : [статья] / URL https://pytorch.org/docs/stable/generated/torch.nn.ReLU.html (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

23 Malyk D., ”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.

24 Yang X., Zhang H. и т.д., ”Fashion Captioning: Towards Generating Accurate Descriptions with Semantic Rewards”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/2008.02693.pdf (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

25 DeVito Z., Ansel J. 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.

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

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

28 Eyre H., Chapman A. B. 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.

29 Paszke A., Gross S. 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.

30 Mullachery V., Motwani V., ”Image Captioning”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1805.09137.pdf (дата обращения 8.05.2022) Загл. с экрана. Яз. англ.

31 Xu K., Kiros R. и т.д., ”Show, Attend and Tell: Neural Image Caption Generation with Visual Attention”, [Электронный ресурс] : [статья] / URL https://arxiv.org/pdf/1502.03044.pdf (дата обращения 8.05.2022) Загл. с экрана. Яз. англ.

32 ADITYAJN105, ”Flickr 8k Dataset ”, [Электронный ресурс] : [статья] / URL https://www.kaggle.com/datasets/adityajn105/flickr8k (дата обращения 8.05.2022) Загл. с экрана. Яз. англ.

33 Mehri S., ”Image Captioning”, [Электронный ресурс] : [статья] / URL http://shikib.com/captioning.html (дата обращения 14.04.2022) Загл. с экрана. Яз. англ.

34 ”Inception_v3”, [Электронный ресурс] : [статья] / URL https://pytorch.org/hub/pytorch_vision_inception_v3/ (дата обращения 8.05.2022) Загл. с экрана. Яз. англ.

35 Schwab A., ”Embeddings: A Matrix of Meaning”, [Электронный ресурс] : [статья] / URL https://petuum.medium.com/embeddings-a-matrix-of-meaning-4de877c9aa27 (дата обращения 8.05.2022) Загл. с экрана. Яз. англ.

36 Luo Z., Peng B. и т.д., ”Show, Discriminate, and Tell: A Discriminatory Image Captioning Model with Deep Neural Networks”, [Электронный ресурс] : [статья] / URL https://web.stanford.edu/class/cs231a/prev_projects_-2016/cs231a.pdf (дата обращения 8.05.2022) Загл. с экрана. Яз. англ.

37 Ritter F. E., Schooler L. J., ”The learning curve”, [Электронный ресурс] : [статья] / URL http://ritter.ist.psu.edu/papers/ritterS01.pdf (дата обращения 12.05.2022) Загл. с экрана. Яз. англ.

38 ”An open-source software library for Machine Intelligence” [Электронный ресурс] : [сайт] / URL: https://www.tensorflow.org (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

39 ”An open source machine learning framework that accelerates the path from research prototyping to production deployment.” [Электронный ресурс] : [сайт] / URL: https://pytorch.org/ (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

40 ”A flexible and efficient library for deep learning” [Электронный ресурс] : [сайт] / URL: https://mxnet.incubator.apache.org/versions/1.9.0/ (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

41 ”Apache TVM” [Электронный ресурс] : [сайт] / URL: https://tvm.apache.org/ (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

42 ”JAX reference documentation” [Электронный ресурс] : [сайт] / URL: https://jax.readthedocs.io/en/latest/index.html (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

43 ”NumPy. The fundamental package for scientific computing with Python” [Электронный ресурс] : [сайт] / URL: https://numpy.org/ (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

44 ”pandas” [Электронный ресурс] : [сайт] / URL: https://pandas.pydata.org/ (дата обращения 17.05.2022) Загл. с экрана. Яз. англ.

45 ”Jupyter. Free software, open standards, and web services for interactive computing across all programming languages” [Электронный ресурс] : [сайт] / URL: https://jupyter.org/ (дата обращения 17.05.2022) 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)
```

```
39          """
40          return len(self.itos)
41
42      @staticmethod
43      def tokenizer_eng(text):
44          """
45              Функция токенизирует слова некоторого текста.
46
47          :param text: входной текст на английском языке
48          :return: список токенов для каждого слова в тексте
49          """
50
51          # пример: "I love bananas" -> ["i", "love", "bananas"]
52          return [tok.text.lower() for tok in SPACY_ENG.tokenizer(text)]
53
54      def build_vocabulary(self, sentence_list):
55          """
56              Функция осуществляет заполнение словаря (добавление слов в него и
57              определение для каждого из них индекса).
58
59          :param sentence_list: список предложений, являющихся подписям к
60              изображениям
61          """
62
63          frequencies = {}
64          idx = 4
65
66          for sentence in sentence_list:
67              for word in self.tokenizer_eng(sentence):
68                  if word not in frequencies:
69                      frequencies[word] = 1
70                  else:
71                      frequencies[word] += 1
72
73                  if frequencies[word] == self.freq_threshold:
74                      self.stoi[word] = idx
75                      self.itos[idx] = word
76                      idx += 1
77
78      def numericalize(self, text):
79          """
```

```
80              Функция заменяет слова токенизированного текста соответствующими
81              этим словам индексами.
82
83          :param text: некоторый текст, подлежащий токенизированию
84          :return: список индексов
85          """
86          tokenized_text = self.tokenizer_eng(text)
87
88          return [
89              self.stoi[token] if token in self.stoi else self.stoi["<UNK>"]
90              for token in tokenized_text
91          ]
92
93
94  class FlickrDataset(Dataset):
95      """
96          Определение класса датасета для набора данных Flickr.
97      """
98
99      def __init__(self, root_dir, caption_file, transform=None,
        ↪  freq_threshold=5):
100         """
101             Инициализирует объект датасета, определяя списки путей к
102             изображениям и соответствующим им подписям, а также формирует
103             словарь.
104
105         :param root_dir: корневая директория, откуда будут браться изображения
106         :param caption_file: адрес файла с подписями к изображениям
107         :param transform: некоторый объект, преобразовывающий изображения
108             заданным образом
109         :param freq_threshold: максимальное количество повторяющихся слов
110         """
111         self.root_dir = root_dir
112         self.df = pd.read_csv(caption_file)
113         self.transform = transform
114
115         self.imgs = self.df["image"]
116         self.captions = self.df["caption"]
117
118         self.vocabulary = Vocabulary(freq_threshold)
119         self.vocabulary.build_vocabulary(self.captions.tolist())
```

```
120
121     def __len__(self):
122         """
123             Функция возвращает длину датафрейма (т.е. количество различных
124             подписей к изображениям).
125
126         :return: длина датафрейма
127         """
128         return len(self.df)
129
130     def __getitem__(self, index):
131         """
132             Функция определяет возможность взятия одной сущности из датасета
    ↪  (в
133             частности, изображения и соответствующей ему подписи).
134
135         :param index: индекс сущности, которую необходимо считать
136         :return: кортеж из трансформированного изображения и тензора
137                  преобразованной в индексы слов подписи
138         """
139         caption = self.captions[index]
140         img_id = self.imgs[index]
141         img = Image.open(os.path.join(self.root_dir, img_id)).convert("RGB")
142
143         if self.transform is not None:
144             img = self.transform(img)
145
146         numericalized_caption = [self.vocabulary.stoi["<SOS>"]]
147         numericalized_caption += self.vocabulary.numericalize(caption)
148         numericalized_caption.append(self.vocabulary.stoi["<EOS>"])
149
150         return img, torch.tensor(numericalized_caption)
151
152
153 class MyCollate:
154     """
155         Класс MyCollate для помещения в batch значений датасета.
156
157     """
158     def __init__(self, pad_idx):
159         self.pad_idx = pad_idx
```

```
160
161     def __call__(self, batch):
162         imgs = [item[0].unsqueeze(0) for item in batch]
163         imgs = torch.cat(imgs, dim=0)
164         targets = [item[1] for item in batch]
165         targets = pad_sequence(targets, batch_first=False,
            ↪  padding_value=self.pad_idx)
166
167         return imgs, targets
168
169
170 def get_loader(
171         root_folder,
172         annotation_file,
173         transform,
174         batch_size=32,
175         num_workers=8,
176         shuffle=True,
177         pin_memory=True
178 ):
179     """
180     Функция get_loader осуществляет загрузку данных для работы модели
    ↪  машинного
181     обучения с этими данными.
182
183     :param root_folder: корневая директория, откуда будут браться изображения
184     :param annotation_file: путь к файлу с подписями для изображений
185     :param transform: преобразование, которое необходимо сделать с
    ↪  изображениями
186     :param batch_size: размер batch
187     :param num_workers: определяет количество некоторого ресурса компьютера
    ↪  для
188         ускорения работы
189     :param shuffle: параметр перемешивания сущностей
190     :param pin_memory: определяет количество некоторого ресурса компьютера
191
192     :return: кортеж из объекта загрузки данных и объекта датасета
193     """
194     dataset = FlickrDataset(root_folder, annotation_file, transform)
195     pad_idx = dataset.vocabulary.stoi["<PAD>"]
196
```

```
197     loader = DataLoader(
198         dataset=dataset,
199         batch_size=batch_size,
200         num_workers=num_workers,
201         shuffle=shuffle,
202         pin_memory=pin_memory,
203         collate_fn=MyCollate(pad_idx)
204     )
205 
206     return loader, dataset
207 
208 
209 def main():
210     """
211         Запуск функцию осуществляет проверку корректности созданных классов.
212     """
213     transform = transforms.Compose(
214         [
215             transforms.Resize((224, 224)),
216             transforms.ToTensor(),
217         ]
218     )
219     dataloader = get_loader("archive/images",
        ↪  annotation_file="archive/captions.txt",
220                             transform=transform)
221 
222     for idx, (imgs, captions) in enumerate(dataloader):
223         print(imgs.shape)
224         print(captions.shape)
225 
226 
227 if __name__ == "__main__":
228     main()
```

ПРИЛОЖЕНИЕ Б

Код `model.py`

```
1  import torch
2  import torch.nn as nn
3  import torchvision.models as models
4
5
6  class EncoderCNN(nn.Module):
7      """
8          Кодировщик, определяющийся с помощью предобученной GoogleNetv3.
9      """
10
11     def __init__(self, embedding_size, train_CNN=False):
12         """
13             Инициализация модели с определением её слоев.
14
15         :param embedding_size: размер эмбеддинга
16         :param train_CNN: задается ли CNN для обучения
17         """
18         super(EncoderCNN, self).__init__()
19         self.train_CNN = train_CNN
20         self.inception = models.inception_v3(pretrained=True,
           ↪ aux_logits=False)
21         self.inception.fc = nn.Linear(self.inception.fc.in_features,
           ↪ embedding_size)
22         self.relu = nn.ReLU()
23         self.dropout = nn.Dropout(0.5)
24
25     def forward(self, images):
26         """
27             Функция, определяющая процесс передачи весов внутри кодировщика.
28
29         :param images: список тензоров, полученных преобразованием изображений
30         :return: веса модели, пропущенные через relu с осуществлением сброса
   ↪ весов
31         """
32         features = self.inception(images)
33
34         for name, param in self.inception.named_parameters():
35             if "fc.weight" in name or "fc.bias" in name:
36                 param.requires_grad = True
```

```
37              else:
38                  param.requires_grad = self.train_CNN
39
40          return self.dropout(self.relu(features))
41
42
43  class DecoderRNN(nn.Module):
44      """
45          Декодировщик, определяемый с помощью LSTM-модели.
46      """
47
48      def __init__(self, embedding_size, hidden_size, vocabulary_size,
        ↪ layers_num):
49          """
50              Инициализация модели с определением её слоев.
51
52          :param embedding_size: размер эмбеддинга
53          :param hidden_size: выходной вектор значений LSTM
54          :param vocabulary_size: размер словаря
55          :param layers_num: количество слоев, составляющих LSTM
56          """
57          super(DecoderRNN, self).__init__()
58          self.embedding = nn.Embedding(vocabulary_size, embedding_size)
59          self.lstm = nn.LSTM(embedding_size, hidden_size, layers_num)
60          self.linear = nn.Linear(hidden_size, vocabulary_size)
61          self.dropout = nn.Dropout(0.5)
62
63      def forward(self, features, captions):
64          """
65              Функция определяющая процесс передачи весов внутри декодировщика
66
67          :param features: получаемые вектора, являющиеся результатом работы
    ↪ кодировщика
68          :param captions: подписи к изображениям
69          :return: подпись к входному изображению
70          """
71          embeddings = self.dropout(self.embedding(captions))
72          embeddings = torch.cat((features.unsqueeze(0), embeddings), dim=0)
73          hiddens, _ = self.lstm(embeddings)
74          outputs = self.linear(hiddens)
75          return outputs
```

```
76
77
78  class CNNtoRNNTranslator(nn.Module):
79      """
80          Класс, описывающий ансамбль, составленный из кодировщика и
    ↪   декодировщика
81      """
82
83      def __init__(self, embedding_size, hidden_size, vocabulary_size,
        ↪   layers_num):
84          """
85              Инициализация элементов ансамбля.
86
87          :param embedding_size: размер эмбеддинга
88          :param hidden_size: количество выходных векторов LSTM
89          :param vocabulary_size: размер словаря
90          :param layers_num: количество слоев, составляющих LSTM
91          """
92          super(CNNtoRNNTranslator, self).__init__()
93          self.encoderCNN = EncoderCNN(embedding_size)
94          self.decoderRNN = DecoderRNN(embedding_size, hidden_size,
            ↪   vocabulary_size, layers_num)
95
96      def forward(self, images, captions):
97          """
98              Процесс передачи весов между элементами ансамбля
99
100         :param images: тензоры изображений
101         :param captions: подписи к изображениям
102         :return: сгенерированные подписи к изображениям
103         """
104         features = self.encoderCNN(images)
105         outputs = self.decoderRNN(features, captions)
106         return outputs
107
108     def image_caption(self, image, vocabulary, max_length=50):
109         """
110             Функция генерирует подпись (максимальная длина - 50 символов) к
    ↪   изображению
111             на основе словаря, предварительно обработав входное изображение.
112
```

```
113         :param image: изображение
114         :param vocabulary: словарь
115         :param max_length: максимальная длина генерируемой подписи
116         :return: вектор слов, определяющий подпись к изображению
117         """
118         result_caption = []
119 
120         with torch.no_grad():
121             x = self.encoderCNN(image).unsqueeze(0)
122             states = None
123 
124             for _ in range(max_length):
125                 hiddens, states = self.decoderRNN.lstm(x, states)
126                 output = self.decoderRNN.linear(hiddens.squeeze(0))
127                 predicted = output.argmax(1)
128 
129                 result_caption.append(predicted.item())
130                 x = self.decoderRNN.embedding(predicted).unsqueeze(0)
131 
132                 if vocabulary.itos[predicted.item()] == "<EOS>":
133                     break
134 
135             return [vocabulary.itos[idx] for idx in result_caption]
```

ПРИЛОЖЕНИЕ В

Код train.py

```
1  import torch
2  import torch.nn as nn
3  import torch.optim as optim
4  import torchvision.transforms as transforms
5  import tensorboard
6  from torch.utils.tensorboard import SummaryWriter
7  from model import CNNtoRNNTranslator
8  from getloader import get_loader
9
10
11 def train():
12     """
13         Функция осуществляет процесс обучения модели машинного обучения.
14
15     """
16
17     # преобразования, которым подлежат изображения
18     transform = transforms.Compose(
19         [
20             transforms.Resize((356, 356)),
21             transforms.RandomCrop((299, 299)),
22             transforms.ToTensor(),
23             transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
24         ]
25     )
26
27     # определение объектов загрузчика данных с объектом датасета с учетом пути
28     # к нужным файлам и заданными преобразованиями изображений
29     train_loader, dataset = get_loader(
30         root_folder="archive/images",
31         annotation_file="archive/training_captions.txt",
32         transform=transform,
33         num_workers=6,
34     )
35
36     # обучение будет происходит с помощью технологии Cuda
37     torch.backends.cudnn.benchmark = True
38     print("cuda: ", torch.cuda.is_available())
39     device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
40      load_model = False
41      save_model = True
42
43      # задача гиперпараметров
44      embedding_size = 256
45      hidden_size = 256
46      vocabulary_size = len(dataset.vocabulary)
47      num_layers = 4
48      learning_rate = 4e-4
49      num_epochs = 100
50
51      # в процессе обучения будут сохраняться характеристики,
52      # отображаемые с помощью tensorboard
53      writer =
        ↪  SummaryWriter(f"runs/flickr/nl_{num_layers}_lr_{learning_rate}")
54      step = 0
55
56      # инициализация модели, функции потерь и оптимизатора
57      model = CNNtoRNNTranslator(embedding_size,
58                                 hidden_size,
59                                 vocabulary_size,
60                                 num_layers).to(device)
61      criterion =
        ↪  nn.CrossEntropyLoss(ignore_index=dataset.vocabulary.stoi["<PAD>"])
62      optimizer = optim.Adam(model.parameters(), lr=learning_rate)
63
64      # подразумевается возможность загрузки весов модели
65      if load_model:
66          checkpoint = torch.load("my_checkpoint.pth.tar")
67          model.load_state_dict(checkpoint["state_dict"])
68          optimizer.load_state_dict(checkpoint["optimizer"])
69          step = checkpoint["step"]
70
71      # включение режима обучения модели
72      model.train()
73
74      for epoch in range(num_epochs + 1):
75          print("Epoch: ", epoch)
76
77          # сохранение параметров модели каждые 10 эпох с учетом булевой
            ↪  переменной
```

```
78          if save_model and (epoch % 10 == 0):
79              checkpoint = {
80                  "state_dict": model.state_dict(),
81                  "optimizer": optimizer.state_dict(),
82                  "step": step,
83              }
84              torch.save(checkpoint, f"my_checkpoint"
85                                     f"_epoc_{epoch}"
86                                     f"_nl_{num_layers}"
87                                     f"_lr_{learning_rate}.pth.tar")
88
89          for idx, (imgs, captions) in enumerate(train_loader):
90              imgs = imgs.to(device)
91              captions = captions.to(device)
92
93              outputs = model(imgs, captions[:-1])
94              loss = criterion(outputs.reshape(-1, outputs.shape[2]),
                ↪  captions.reshape(-1))
95
96              # сохранение значения функции потерь для формирования графика
97              # с помощью tensorboard
98              writer.add_scalar("Training loss", loss.item(), global_step=step)
99              step += 1
100
101             optimizer.zero_grad()
102             loss.backward(loss)
103             optimizer.step()
104
105
106 if __name__ == "__main__":
107     train()
```

ПРИЛОЖЕНИЕ Г

Код checkmetric.py

```
import torch
from model import *
from getloader import get_loader
import torchvision.transforms as transforms
from PIL import Image
from IPython import display
import pandas as pd
from sentence_transformers import SentenceTransformer
import numpy as np


def load_model_to_captioning(image_path="archive/images",
                             caption_path="archive/captions.txt",
                             checkpoint_path="my_checkpoint.pth.tar"):
    """
        Функция загружает веса модели согласно указанным в аргументах
        директориям, создает объекты загрузчика данных.

    :param image_path: путь к изображениям
    :param caption_path: путь к csv-файлу с текстовыми описаниями для
        изображений
    :param checkpoint_path: путь к файлу весов модели

    :return: кортеж из объекта модели, словаря набора данных, объекта
        преобразований 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)),
        ]
    )

    test_loader, dataset = get_loader(
        root_folder=image_path,
        annotation_file=caption_path,
        transform=transform,
```

```
40          num_workers=6,
41      )
42
43      device = "cuda"
44      embedding_size = 256
45      hidden_size = 256
46      vocabulary_size = len(dataset.vocabulary)
47      num_layers = 2
48      model = CNNtoRNNTranslator(embedding_size, hidden_size,
49                                 vocabulary_size, num_layers).to(device)
50      checkpoint = torch.load(checkpoint_path)
51      model.load_state_dict(checkpoint["state_dict"])
52
53      return model, dataset.vocabulary, transform
54
55
56  def create_caption(image_path, model=None, vocabulary=None,
57                     transform=None, device="cuda", show_image=False):
58      """
59          Функция генерирует текстовое описание для изображения.
60
61      :param image_path: путь к изображению
62      :param model: файл модели, которая будет генерировать текстовое описание
63      :param vocabulary: словарь, слова из которого будут использоваться в
64          описании
65      :param transform: объект преобразований изображения
66      :param device: указание строкой ресурса, с помощью которого будет работать
67          модель (cuda или cpu)
68      :param show_image: выводить ли изображение на экран перед возвратом
    ↪ функцией
69          текстового описания
70
71      :return: кортеж из объекта модели, словаря набора данных, объекта
72              преобразований изображения
73      """
74      if not model:
75          capt_path = "archive/training_captions.txt"
76          checkp_path = "my_checkpoint_100.pth.tar"
77          model, vocabulary, transform =
            ↪ load_model_to_captioning(caption_path=capt_path,
78
                                                                ↪ checkpoint_path=cl
```

```
79      model.eval()
80      image = transform(Image.open(image_path).convert("RGB")).unsqueeze(0)
81      word_list = model.image_caption(image.to(device), vocabulary)
82      word_list = [word for word in word_list if not (word in
        ↪ [*vocabulary.stoi][:4])]
83      caption = ' '.join([str(elem) for elem in word_list])
84      if show_image:
85          display.display(Image.open(image_path))
86      return caption
87
88
89  def cosine_similarity(sentence_embeddings, ind_a, ind_b):
90      """
91          Функция реализует косинусный коэффициент.
92
93      :param sentence_embeddings: список из эмбеддингов сгенерированного и
94          истинного текстового описания
95      :param ind_a: индекс в этом списке сгенерированного описания
96      :param ind_b: индекс в этом списке истинного описания
97
98      :return: действительное значение косинусного коэффициента
99      """
100     s = sentence_embeddings
101     return np.dot(s[ind_a], s[ind_b]) / (np.linalg.norm(s[ind_a])
102                                          * np.linalg.norm(s[ind_b]))
103
104
105 def check_quality(images_path="archive/images",
    ↪ captions_path="archive/captions.txt"):
106     """
107         Функция находит среднее значение косинусного коэффициента относительно
108         сгенерированных с помощью модели текстовых описаний и истинных
    ↪ описаний
109         тестовой выборки. Выводит это значение, затем выводит
    ↪ интерпретированное
110         описание смысла этого значения.
111
112     :param images_path: путь к изображениям
113     :param captions_path: путь к csv-файлу с истинными текстовыми описаниями к
114         изображениям
115     """
```

```
116
117     df = pd.read_csv(captions_path)
118     quality = 0
119     sent_trans_model = SentenceTransformer('bert-base-nli-mean-tokens')
120     capt_path = "archive/training_captions.txt"
121     checkp_path = "my_checkpoint_100.pth.tar"
122     model, vocab, transform = load_model_to_captioning(caption_path=capt_path,
        ↪ checkpoint_path=checkp_path)
123
124     for elem in df.iterrows():
125         image_name = elem[1][1]
126         ans_caption = elem[1][2]
127         out_caption = create_caption(images_path + "/" + str(image_name),
128                                      model,
129                                      vocab,
130                                      transform)
131
132         sentences = [out_caption, ans_caption]
133         sentence_embeddings = sent_trans_model.encode(sentences)
134         quality += cosine_similarity(sentence_embeddings, 0, 1)
135
136     mean_quality = quality / df.shape[0]
137     print("Средняя оценка качества модели: ", mean_quality)
138
139     if mean_quality <= 0.25:
140         print("Модель работает плохо, не способна даже в общих чертах описать
            ↪ то,"
141               " что присутствует на изображении.")
142     elif 0.25 < mean_quality <= 0.5:
143         print("Модель работает удовлетворительно - способна различать и
            ↪ корректно"
144               " интерпретировать основные детали изображения, но может
                  ↪ допускать"
145               " некоторые логические и смысловые ошибки в описании.")
146     elif 0.5 < mean_quality <= 0.75:
147         print("Модель работает хорошо - почти всегда корректно описывает
            ↪ основные"
148               " детали изображения, но способна допускать незначительные
                  ↪ логические"
149               " ошибки.")
150     else:
```

```
151         print("Модель работает отлично, количество ошибок при генерации
            ↪ текстового"
152               " описания к входному изображению сведено к минимуму.")
```