СОДЕРЖАНИЕ

1 Постановка задачи

Описание изображения компьютером – задача искусственного интеллекта, которая объединяет компьютерное зрение и обработку естественного языка.

Эта задача значительно сложнее, например, чем задачи классификации изображений или распознавания объектов, поскольку описание должно охватывать не только объекты, представленные на изображении, но также должно выражать как эти объекты связаны друг с другом, их признаки и может быть действия.

В настоящее работе представлена модель, которая принимает изображение I в качестве входных данных и обучена максимизировать вероятность $p(S|I)$ получения целевой последовательности слов $S = \{S_1, S_2, ...\}$, где каждое слово S_t происходит из словаря, который адекватно описывает изображение.

Естественно [1] использовать CHC в качестве кодировщика изображения, предварительно обучив его задаче классификации изображений и используя последний скрытый слой в качестве входных данных для раскодировщика РНС, который порождает предложения [1].

В настоящей работе объединены глубокие свёрточные сети для классификации изображений [2] с рекуррентными сетями для моделирования последовательностей [3], чтобы создать единую сеть, которая порождает описания изображений [1].

При обучении такой сети необходимо максимизировать вероятность правильного описания данного изображения, используя следующую формулу [1]:

$$\theta^* = \arg\max \sum_{(I,S)} \log p(S \mid I; \theta),$$

где θ – параметры модели, I – изображение, S – предложение, длина которого не ограничена.

Применим правило цепочки для моделирования совместной вероятности S_0, ..., S_N, где N – длина конкретного примера:

$$\log p(S \mid I) = \sum_{t=0}^{N} \log p(S_t \mid I, S_0, ..., S_{t-1})$$

(S, I) является парой примеров обучения, в течение обучения оптимизируется сумма логарифмических вероятностей по всему набору обучения, используя стохастический градиентный спуск.

Вероятность $p(S_t \mid \mathrm{I}, \mathrm{S}_0, ..., S_{t-1})$ моделируется РНС, для которой переменное количество слов ограничивается до *t-1* скрытым состоянием фиксированной длины или памятью h_t. Эта память обновляется с помощью нелинейной функции f после просмотра нового ввода x_t:

$$h_{t+1} = f(h_t, x_t).$$

Для построения РНС нужно выбрать нелинейную функцию f и как изображения и предложения подаются в качестве входных данных x_t: в качестве f используется так называемая долгая краткосрочная память [3] (long short-term memory, LSTM, ДКП) а для представления изображений используется СНС.

Выбор ДКП в качестве f обусловлен её способностью справляться с исчезающими и взрывающимися градиентами [3] – распространённой проблемой при проектировании и обучении РНС [1].

Цель настоящего курсового проекта – разработка искусственной нейронной сети, порождающей к изображениям подписи на русском языке.

2 Долгая краткосрочная память

Любая РНС имеет форму цепочки повторяющихся ячеек (или модулей) нейронной сети. В обычной РНС структура одной такой ячейки проста, например, она может представлять собой один слой с функцией активации гиперболический тангенс.

Структура ДКП также напоминает цепочку, но её ячейки выглядят

иначе. Вместо одного слоя нейронной сети она содержат четыре, и эти слои взаимодействуют особенным образом. На рисунке 1 представлена схема ДКП.

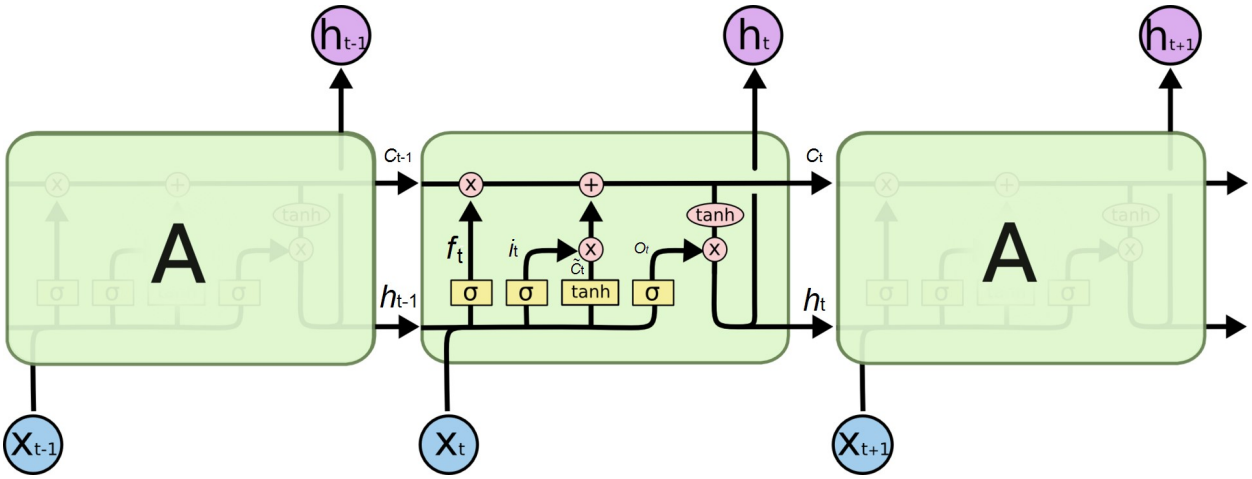


Рисунок 1 – Схема ДКП [4]

На схеме каждая линия переносит вектор от выхода одного узла ко входу другого. Розовыми кружками обозначены поэлементные операции сложения или умножения, а жёлтыми прямоугольниками –обученные слои нейронной сети. Сливающиеся линии означают объединение, а разветвляющиеся стрелки говорят о том, что данные копируются и копии передаются в разные компоненты сети.

Ключевой компонент ДКП – это состояние ячейки – горизонтальная линия, проходящая по верхней части схемы. Состояние ячейки проходит через всю цепочку.

ДКП может изменять информацию состояния ячейки; этот происходит с помощью фильтров (gates). Фильтры состоят из слоя сигмоидальной нейронной сети и операции поточечного умножения. Сигмоидальный слой возвращает числа от нуля до единицы, которые обозначают, какую долю каждого блока информации следует пропустить дальше по сети.

Первый шаг ДКП – определить, какую информацию можно выбросить из состояния ячейки. Это решение принимает сигмоидальный слой фильтра забывания (forget gate layer). Слой возвращает число от 0 до 1 для каждого числа из состояния ячейки C_{t-1}, руководствуясь значениями h_{t-1} и x_t (1 означает полное сохранение, а 0 – полностью забыть):

$$f_t = \sigma\left(W_f \cdot \left[h_{t-1}, x_t\right] + b_f\right).$$

Следующий шаг – решить, какая новая информация будет храниться в состоянии ячейки. Этот этап состоит из двух частей:

– сигмоидальный слой входного фильтра (input layer gate) определяет, какие значения следует обновить;

– слой гиперболического тангенса строит вектор новых значений $\tilde{C}_t$, которые можно добавить в состояние ячейки.

$$i_t = \sigma\left(W_i \cdot \left[h_{t-1}, x_t\right] + b_i\right),$$

$$\tilde{C}_t = \tanh\left(W_C \cdot \left[h_{t-1}, x_t\right] + b_C\right).$$

После этих двух шагов необходимо заменить состояние ячейки C_{t-1} на новое состояние C_t. Сеть забывает информацию умножением старого состояния C_{t-1} на f_t, затем запоминает новую информацию суммируя это изменённое состояние и $i_t \cdot \tilde{C}_t$:

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t.$$

Выходные данные будут основаны на нашем состоянии ячейки, к ним будут применены некоторые фильтры. Сначала применится сигмоидальный слой, который решает, какую информацию из состояния ячейки вывести, затем значения состояния ячейки проходят через слой гиперболического тангенса, чтобы получить на выходе значения из диапазона от -1 до 1, и перемножаются с выходными значениями сигмоидального слоя, что позволяет выводить только требуемую информацию:

$$o_t = \sigma\left(W_o \cdot \left[h_{t-1}, x_t\right] + b_o\right),$$

$$h_t = o_t \cdot \tanh(C_t).$$

3 Метод обучения

Модель ДКП обучается предсказывать каждое слово предложения после того, как она увидела изображение и все предыдущие слова: $p(S_t \mid \mathrm{I}, \mathrm{S}_0, ..., S_{t-1})$. Представим ДКП в развёрнутом виде – все рекуррентные соединения преобразуем в соединения прямой передач: имеем копии ДКП для каждого слова предложения и изображения, все копии имеют одни и те же параметры, и выходное значение h_{t-1} в момент времени t-1 подаётся на вход сети в момент t [1]. Схема такого представления изображена на рисунке 2.

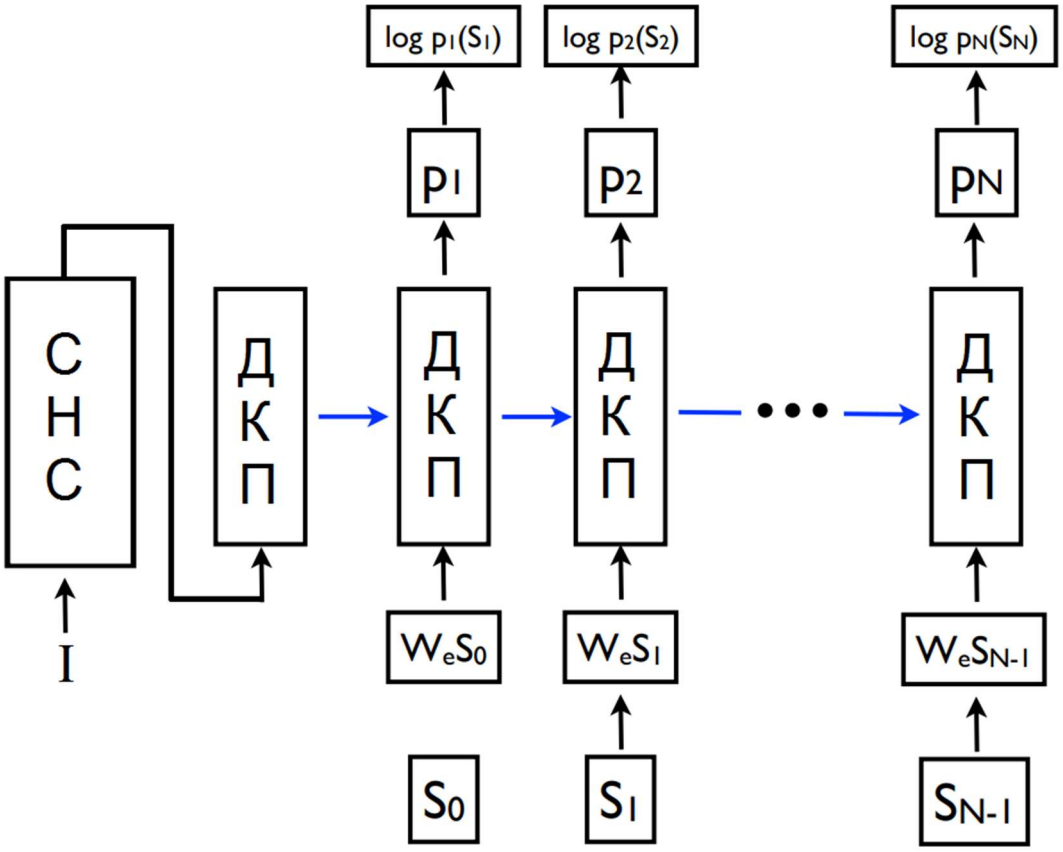


Рисунок 2 – Схема ДКП (упразднена рекурсия)

Обозначим через I входное изображение и через $S = (S_0, ..., S_N)$ предложение, описывающее это изображение, имеем следующую последовательность вводов и вывода:

$$x_{-1} = CHC(I)$$
$$x_t = W_e S_t, \; t \in \{0, ..., N-1\}$$
$$p_{t+1} = ДКП(x_t), \; t \in \{0, ..., N-1\}$$

Каждое слово представлено в виде одномерного вектора S_t размерности, равной размеру словаря. Слова S_0 и S_N – метки, обозначающие начало и конец. Так, выдавая конечное слово S_N, ДКП сигнализирует о том, что создано полное предложение.

И изображение, и слова отображаются в одно и то же пространство, изображение с помощью СНС, слова с помощью слоя встраивания слов (embedding) W_e. Изображение I вводится только один раз, при t = -1, чтобы сообщить ДКП о содержимом изображения.

Функция потерь обучения – сумма отрицательной логарифмической вероятности (перекрёстная энтропия) правильного слова на каждом шаге:

$$L(I,S) = -\sum_{t=1}^{N} \log p_t(S_t) .$$

Потери сведены к минимуму без учёта всех параметров ДКП, верхнего слоя встраивания изображений СНС и встраиваний слов W_e.

4 Метод оценки BLEU

BLEU (Bilingual Evaluation Understudy) – это измерение различий между сгенерированном предложением и одним или несколькими предложениями [5].

Алгоритм BLEU подсчитывает количество совпадающих n-грамм (цепочка из n слов) в сгенерированном продолжении с n-граммами в эталонным предложением и подсчитывает количество совпадений. Эти совпадения не зависят от позиции, внятность и грамматика не учитываются.

Идеальное совпадение приводит к баллу 1.0, а идеальное несоответствие – к баллу 0.0. Идеальный результат на практике маловероятен, так как сгенерированное предложение должно точно соответствовать эталонному – это маловероятно даже в том случае, если предложение “генерирует” человек.

Пример: имеем два предложения “"Карл у Клары украл кораллы"” и “"Карл у мамы украл кораллы”. Сравним по 1-граммам (словам) последнее

предложение с первым по методу BLEU: значение BLUE в этом случае 0.8, т.к. совпало 4 слова из 5.

5 Архитектура нейронной сети

Искусственная нейронная сеть (нейросеть, сеть, ИНС) – это способ собрать нейроны в сеть так, чтобы она решала определённую задачу, например, задачу порождения подписей.

Общая архитектура сети была описана ранее (в пунктах 1 и 3): вкратце, сеть состоит из двух частей – СНС (свёрточная часть) и ДКП (рекуррентная часть) – на вход ДКП подаётся выход СНС, из СНС убран последний слой – слой классификации.

Для порождения подписей к изображению был построенные две сети с разными архитектурами свёрточной части: одна с VGG16, другая с Inception.

Далее, рассмотрим послойно архитектуры VGG16, Inception и рекуррентную часть.

5.1 VGG16

VGG16 – свёрточная нейронная сеть, особенностью которой являются слои свёртки с фильтром 3 × 3 с шагом 1 [6]. В таблице 1 описана архитектура сети VGG16.

Таблица 1 – Архитектура VGG16 [6]

Слой	Размерность выходных данных	Особенности слоя
1	2	3
Свёртка (convolution)	224 × 224	Размер ядра – 3 × 3, длина шага – 1
Свёртка	224 × 224	

Продолжение таблицы 1

1	2	3
Подвыборка (pooling)	112 × 112	Подвыборка максимальных значений (max pooling) с ядром 2 × 2 и шагом – 2
Свёртка	112 × 112	Размер ядра – 3 × 3, длина шага – 1
Свёртка	112 × 112	
Подвыборка	56 × 56	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Свёртка	56 × 56	Размер ядра – 3 × 3, длина шага – 1
Свёртка	56 × 56	
Свёртка	56 × 56	
Подвыборка	28 × 28	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Свёртка	28 × 28	Размер ядра – 3 × 3, длина шага – 1
Свёртка	28 × 28	
Свёртка	28 × 28	
Подвыборка	14 × 14	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Свёртка	14 × 14	Размер ядра – 3 × 3, длина шага – 1
Свёртка	14 × 14	
Свёртка	14 × 14	
Подвыборка	7 × 7	Подвыборка максимальных значений с ядром 2 × 2 и шагом – 2
Выход	4096	Полносвязанный слой
	4096	Полносвязанный слой

На вход слоя свёртки подаётся RGB изображение размером 224 × 224 пикселей. Затем изображение проходят через свёрточные слои, в которых используются фильтры с ядром размера 3x3.

После всех свёрточных слоёв идут три полносвязных слоя: первые два

имеют по 4096 каналов, третий – число каналов соответствует числу классов. Последним идёт софтмакс-слой.

Все скрытые слои снабжены функцией активации ReLU.

5.2 Inception

Inception – это свёрточная нейронная сеть для помощи в анализе изображений и обнаружении объектов [7]. Ключевой особенностью сети являются блоки слоёв (module) – параллельные комбинации свёрточные слоёв – эти блоки уменьшают количество признаков, а значит и вычислений. Схемы таких блоков трёх типов представлены на рисунках 3-5.

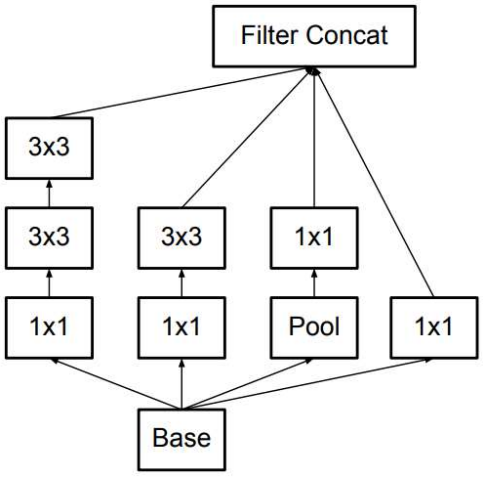


Рисунок 3 – Блок Inception типа 1

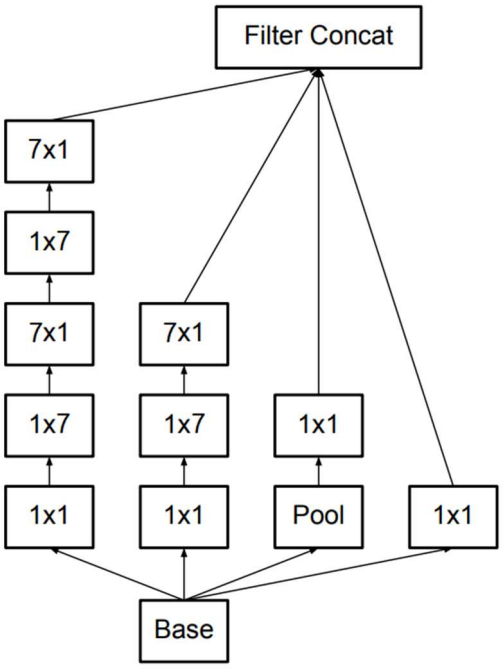


Рисунок 4 – Блок Inception типа 2

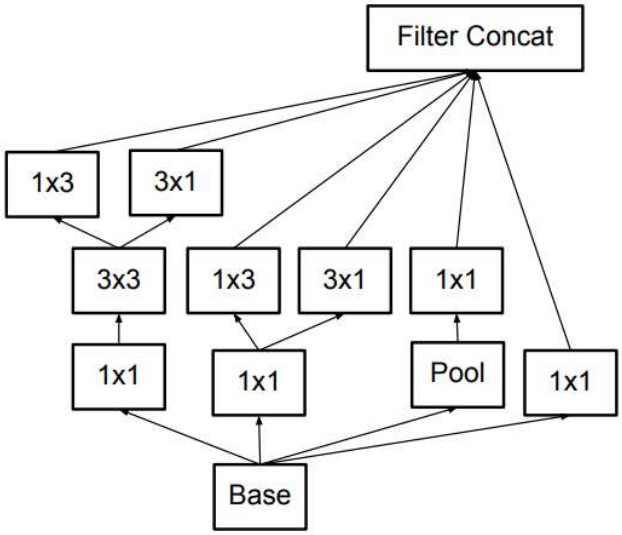


Рисунок 5 – Блок Inception типа 3

Следующие подписи на рисунках означают:

– вида “n x n” означают свёрточный слой размером ядра свёртки n x n;

– “Pool” – подвыборку максимальных значений;

– “Base” – предшествующий слой;

– “Filter Concat” – слой объединения результатов параллельных комбинаций слоёв.

В таблице 2 описана архитектура сети Inception.

Таблица 2 – Архитектура Inception [7]

Слой	Размерность выходных данных	Особенности слоя
1	2	3
Свёртка	229 × 229 × 3	Размер ядра – 3 × 3, длина шага – 2
Свёртка	149 × 149 × 32	Размер ядра – 3 × 3, длина шага – 1
Свёртка	147 × 147 × 32	Размер ядра – 3 × 3, длина шага – 1
Подвыборка	147 × 147 × 64	Подвыборка максимальных значений с ядром 3 × 3 и шагом – 2
Свёртка	73 × 73 × 64	Размер ядра – 3 × 3, длина шага – 1
Свёртка	71 × 71 × 80	Размер ядра – 3 × 3, длина шага – 2

Продолжение таблицы 2

1	2	3
Свёртка	35 × 35 × 192	Размер ядра – 3 × 3, длина шага – 1
3 x Inception	35 × 35 × 288	Слой Inception типа 1
5 x Inception	17 × 17 × 768	Слой Inception типа 2
2 x Inception	8 × 8 × 1280	Слой Inception типа 3
Подвыборка	8 × 8 × 2048	Подвыборка максимальных значений с ядром 8 × 82
Выход	1 × 1 × 2048	Выходной вектор

На вход слоя свёртки подаётся RGB изображение размером 229 × 229 пикселей. Все скрытые слои снабжены функцией активации ReLU.

5.3 Рекуррентная часть

На рисунке 6 представлена схема архитектуры рекуррентный части сети. На выходе этой части – вектор, представляющие вероятности появления слова в предложении, размер вектора соответствует размеру словарю (21391 слово). У этой части два входа:

– один вход (правый) принимает представление изображения, полученное от свёрточной нейронное сети (на схеме – 4096, представление получено от VGG16);

– другой вход (левый) имеет размерность максимального количества слов в предложении (на схеме – 22).

Значение подписей на схеме:

– input и output – размерность входа и выхода

– embedding – преобразование индексов слов в словаре в плотные векторы фиксированного размера (текст предложения представлен в виде списка чисел, где число – индекс слова в словаре), поскольку предложение не имеет фиксированное количество слов, но максимальное число слов – 22;

– dropout – применение исключения на входе в слой (“выключает”

перцептроны);

– lstm – слой ДКП;

– add – слой, который складывает список, поступающий на вход, принимает на вход список тензоров, все одинаковой формы, и возвращает один тензор (также одинаковой формы);

– dense – полносвязанный слой.

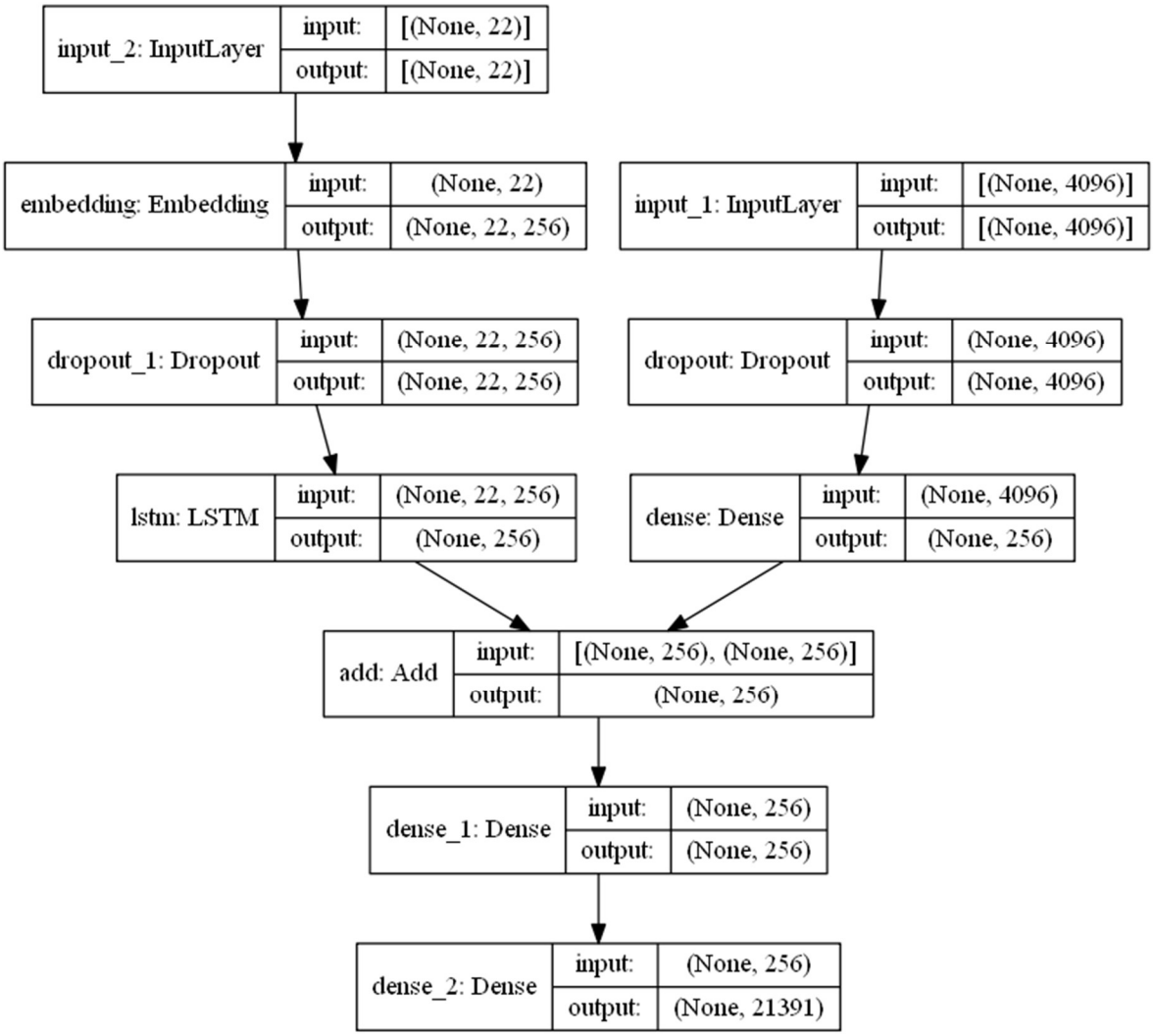


Рисунок 6 – Схема рекуррентной части

6 Обучение

При обучении моделей использовался набор данных, полученные путём преобразований набора Flickr 8k [8].

Особенности метода обучения:

– оптимизатор градиентного спуска – Adam;

– функция потерь градиентного спуска – перекрёстная энтропия;

– количество эпох – 20.

Обучение заканчивалось, после просчёт всех 20 эпох. Для оценки полученных результатов обучения, на каждой эпохе веса моделей сохранялись в отдельный файл.

6.1 Набор данных

Flickr 8k – набор, состоящий из 8000 фотографий, каждая из которых связана с пятью различными подписями на английском языке, описывающие фотографию. Фотографии не содержат каких-либо известных людей или мест и были выбраны вручную для изображения различных ситуаций [8].

Flickr 8k Dataset распространяется для некоммерческого использованиях в целях исследования и/или обучения, набор можно получить, заполнив форму по следующий ссылке – https://forms.illinois.edu/sec/1713398.

Поскольку задача настоящей работы – порождение подписей на русском языке, а в выбранном наборе подписи на английском, подписи набора были переведены на русский язык с помощью машинного перевода Yandex Translate API.

Для предотвращений недообучения и переобучения, переведённый набор данных (далее, набор данных), был обработан:

– все буквы приведены к нижнему регистру;

– из каждого предложения исключены любые знаки препинания;

– числа от 0 до 9, набранные цифрами, заменены именами числительными.

Из набора были удалены изображения и подписи к ним, если хотя бы в одном предложении:

– были кавычки (в наборе означало имя собственное);

– было число больше 9;

– было английское слово (имя собственное);

– число знаков больше 100.

Обработанный набор из 59020 предложений был разделён на три части: для обучения (70% от общего числа предложений), для валидации (20%) и для оценки (10%).

6.2 Оценка

На рисунке 7 представлен график зависимости измерения BLEU от эпохи на валидационном наборе, поскольку модель никогда не “видела” валидационный набор, можно принять за тестовый. В этом наборе 11805 предложений и 2361 изображений, по 5 предложений на каждое изображение.

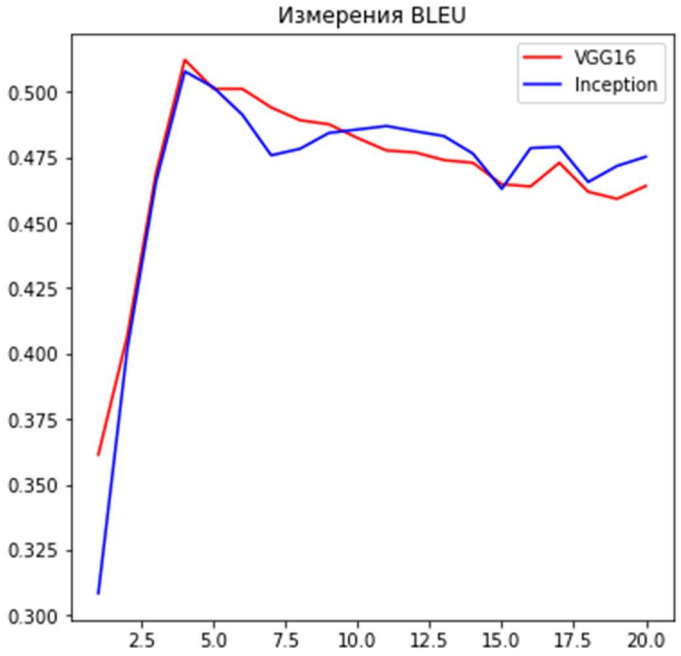


Рисунок 7 – График измерений BLEU

Красная линия изображает измерения BLEU для модели с VGG16 в качестве CNC, синяя – для модели с Inception.

Во время обучения веса модели сохранялись на каждой эпохе в отдельный файл. Для каждой эпохи (для каждого такого файла) было рассчитано измерение BLEU для всего корпуса: сеть генерировал подпись для каждого определённого изображения, эта подпись сравнивалась с пятью эталонным подписями этого изображения, т.е. на каждой эпохе сеть генерировала 2361 подписей для 2361 изображений, и сравнивала каждую с соответствующими ей эталонными (всего 11805 сравнений на эпоху).

Вместо усреднения оценок уровня предложения – макросредняя точность, измерение BLEU учитывает микросреднюю точность – суммирование числителей и знаменателей для каждого сгенерированного предложения и его эталонов до деления.

Время обучения модели с VGG16 – 1 час 28 минут, модели с Inception – 1 час 39 минут.

7 Результаты работы

Выберем две модели с большими значениями измерения BLEU. Лучшее измерение BLEU модели с VGG16 – 0.512229, модели с Inception – 0.50781.

Применим обученные модели на практике. Для этого передадим каждой нейросети пять изображений. Эти изображения не принимали участия в обучении и оценке BLEU. На рисунках 8-12 представлены изображения и подписи к ним, первая строчка – подпись, порождённая моделью с VGG16, вторая – подпись модели с Inception.


```
мужчина в синей куртке держит в руках
мужчина в синей рубашке и очках держит на руках маленького ребенка
```

Рисунок 8 – Подписи к изображению 1

```
мужчина в черном шлеме бежит по грунтовой дорожке
мужчина в красной рубашке и черной шляпе едет на велосипеде по грунтовой дороге
```

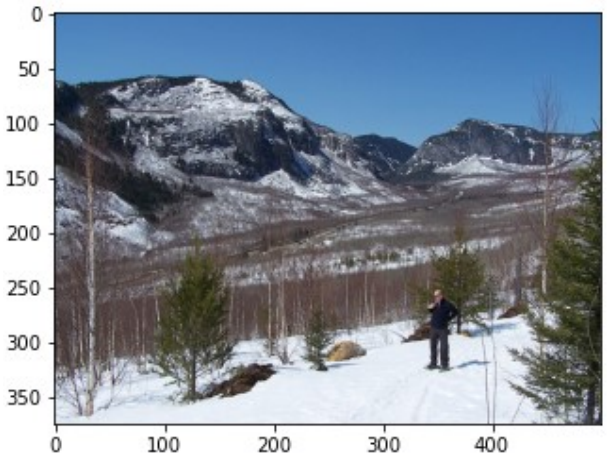
Рисунок 9 – Подписи к изображению 2

```
собака бежит по траве
две собаки бегут по полю
```

Рисунок 10 – Подписи к изображению 3


```
группа людей в зимних пальто и зимних пальто
группа людей идет по улице
```

Рисунок 11 – Подписи к изображению 4



человек в зимней куртке стоит на лыжах с заснеженного холма
человек в воздухе на вершине горы

Рисунок 12 – Подписи к изображению 5

8 Вывод

В рамках настоящей работы были созданы нейронные сети, которые порождают подпись к изображению на русском языке.

Для оценки созданных моделей использовалась метрика BLEU, которая используется для оценки машинного перевода [5]. Сети обучались в течение 20 эпох, обе имеют наибольшее значение BLEU на тестовом наборе данных на четвёртой эпохе – модели, полученные на этих эпоха мы приняли за лучшие. После четвёртой эпохи обе модели имеют меньшее значение BLEU – отметим тенденцию к уменьшению значения с увеличением количества эпох обучения.

Лучшее измерение BLEU модели с VGG16 – 0.512229, модели с Inception – 0.50781, время обучения – 1 час 28 минут и 1 час 39 минут соответственно, размер полученных моделей – 144 МБ и 138 МБ соответственно. Разница между значениями двух моделей – 0.004419 – что меньше одного процента.

Аналогичная сеть, построенная группой исследователей из Google, имеет значение BLEU на наборе данных Flickr 8k– 0.66 [1].

Очевидно, что задача порождения подписей на русском языке сложнее порождения на английском, поскольку, к примеру, в русском языке слово может изменяться по падежам – с точки зрения нейронной сети одно и тоже

слово, изменённое по падежам – это шесть разных слов, в английском же языке нет падежей. Обучение на русском языке требует больше наблюдений и больший словарь, нежели чем на английском.

В целом обе сети справляются удовлетворительно со своей задачей. Возможно результат удастся улучить, если увеличить набор данных и использовать набор, полученный путём разметки людьми, а не полученный машинным переводом Flick 8k.

Исходный код курсового проекта представлен в приложении А и доступен в публичном репозитории по следующей ссылке – https://github.com/algorithm-ssau/image-caption-generator/tree/main/lab3-image-caption-generator.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Show and Tell: A Neural Image Caption Generator [Электронный ресурс]. URL: https://arxiv.org/pdf/1411.4555.pdf (дата обращения: 01.05.2021).

2 Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Электронный ресурс]. URL: https://arxiv.org/pdf/1502.03167.pdf (дата обращения: 01.05.2021).

3 Long Short-term Memory [Электронный ресурс]. URL: https://www.researchgate.net/publication/13853244_Long_Short-term_Memory (дата обращения: 01.05.2021).

4 Understanding LSTM Networks [Электронный ресурс]. URL: http://colah.github.io/posts/2015-08-Understanding-LSTMs/ (дата обращения: 01.05.2021).

5 BLEU: a Method for Automatic Evaluation of Machine Translation [Электронный ресурс]. URL: https://www.aclweb.org/anthology/P02-1040.pdf (дата обращения: 01.05.2021).

6 Very Deep Convolutional Networks for Large-Scale Image Recognition [Электронный ресурс]. URL: https://arxiv.org/pdf/1805.03716.pdf (дата обращения: 01.05.2021).

7 Rethinking the Inception Architecture for Computer Vision [Электронный ресурс]. URL: https://arxiv.org/pdf/1512.00567.pdf (дата обращения: 01.05.2021).

8 Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics [Электронный ресурс]. URL: https://www.jair.org/index.php/jair/article/view/10833/25854 (дата обращения: 01.05.2021).

Приложение А

Исходный код

Файл 11-clear-data.ipynb:

```
import pandas as pd
#%% md
## Набор данных
#%%
path_dataset = "D:/datasets/flickr-30k-images"
path_captions = "D:/YandexDisk/datasets/flickr-30k-images-metadata.csv"
path_prepared_captions = "D:/flickr-30k-images-metadata.csv"
#%%
df = pd.read_csv(path_captions, delimiter='|')
df.head(5)
#%%
df.iat[1, 2]
#%% md
# Подготовка к переводу

### Исправить некоторые знаки препинания

- Первая буква предложения - заглавная.
- Удалить пробелы в начале и в конце.
- Удалить точку в конце.
- Удалить точку.
- Удалить пробел перед запятой.
- Удалить пробел перед апострофом.
- Удалить многоточие.
- Удалить вопросительный знак.
- Удалить восклицательный знак.
- Заменить двоеточие на запятую.
- Заменить точку с запятой на запятую.
- Заменить латинский апостроф на английский.
#%%
def fix_punctuation_marks(a_sentence):
    return a_sentence\
        .strip()\
        .capitalize()\
        .replace(" .", "")\
        .replace(".", "")\
        .replace(" ,", ",")\
        .replace(" '", "'")\
        .replace(" ...", "")\
        .replace("...", "")\
        .replace(" ?", "")\
        .replace("?", "")\
        .replace(" !", "")\
        .replace("!", "")\
        .replace(" : ", ", ")\
        .replace(" :", ", ")\
        .replace(" ; ", ", ")\
        .replace("’", "'")
#%% md
### Заменить некоторые числа на текст
#%%
def replace_some_numbers_with_text(a_sentence):
    return a_sentence\
        .replace(" 0 ", "zero")\
        .replace(" 1 ", "one")\
```

```
        .replace(" 2 ", "two")\
        .replace(" 3 ", "three")\
        .replace(" 4 ", "four")\
        .replace(" 5 ", "five")\
        .replace(" 6 ", "six")\
        .replace(" 7 ", "seven")\
        .replace(" 8 ", "eight")\
        .replace(" 9 ", "nine")
#%% md
### Заменить некоторые числа в начале предложения на текст
#%%
def replace_some_start_numbers_with_text(a_sentence):
    if a_sentence[1] != ' ':
        return a_sentence

    text = ""
    if a_sentence[0] == '0':
        text = "Zero"
    elif a_sentence[0] == '1':
        text = "One"
    elif a_sentence[0] == '2':
        text = "Two"
    elif a_sentence[0] == '3':
        text = "Three"
    elif a_sentence[0] == '4':
        text = "Four"
    elif a_sentence[0] == '5':
        text = "Five"
    elif a_sentence[0] == '6':
        text = "Six"
    elif a_sentence[0] == '7':
        text = "Seven"
    elif a_sentence[0] == '8':
        text = "Eight"
    elif a_sentence[0] == '9':
        text = "Nine"

    a_sentence = text + a_sentence[1:]
    return a_sentence
#%% md
# ***

- Найти кавычки.
- Найти числа.
- Найти двоеточие.
#%%
def is_there_number(sentence):
    for i in sentence.split():
        if i.isdigit():
            return True

    return False
#%%
def fix_dataframe(data, out_idxs_to_delete):
    count_chars = 0

    for idx in data.index:
        sentence = fix_punctuation_marks(data.iat[idx, 2])
        sentence = replace_some_numbers_with_text(sentence)
        sentence = replace_some_start_numbers_with_text(sentence)
        data.iat[idx, 2] = sentence

        if sentence.find('"') != -1:
            out_idxs_to_delete.append(idx)
        elif is_there_number(sentence):
```

```
            out_idxs_to_delete.append(idx)
        elif sentence.find(':') != -1:
            out_idxs_to_delete.append(idx)

        count_chars += len(sentence)

    return count_chars
#%%
idxs_to_delete = []
chars_num = fix_dataframe(df, idxs_to_delete)
print(chars_num)
#%% md
# Удалить изображения

- у которых в описании есть кавычки
- у которых в описании есть число
- у которых в описании есть двоеточие
#%%
df_to_delete = df.loc[idxs_to_delete, :]
images_to_delete = df_to_delete["image_name"].unique()
#%%
df = df[~df.image_name.isin(images_to_delete)]
df.shape
#%%
df.head(5)
#%% md
## Сохранить dataframe в файл
#%%
df.to_csv(path_prepared_captions, sep='|', encoding='utf-8', index=False)
```

Файл 12-translate-data.ipynb:

```
import json
import requests
import time
import pandas as pd
#%%
path_dataset = "D:/datasets/flickr-30k-images"
path_captions = "D:/YandexDisk/datasets/flickr-30k-images-metadata.csv"
path_prepared_captions = "D:/flickr-30k-images-metadata.csv"
#%%
df = pd.read_csv(path_captions, delimiter='|')
df.head(5)
#%% md
## Перевод

Квоты — организационные ограничения, которые можно изменить по запросу в
техническую поддержку.
Лимиты — технические ограничения, обусловленные особенностями архитектуры
Yandex.Cloud. Изменение лимитов невозможно.

Yandex Translate. Квоты (4 апреля 2021):
Вызовов одного метода API в секунду - 20
Символов, отправленных на перевод, в час - 1 млн
#%%
IAM_TOKEN = "_____"
FOLDER_ID = "_____"

BEARER = "Bearer " + IAM_TOKEN
HEADERS = {
    "Content-type": "application/json",
    "Authorization": BEARER
}
```

```
def translate_with_yandex(text):
    data = {
        "folder_id": FOLDER_ID,
        "texts": text,
        "targetLanguageCode": "ru"
    }

    response =
requests.post('https://translate.api.cloud.yandex.net/translate/v2/translate'
, headers=HEADERS, data=str(data).encode('utf-8'))
    json_text = json.loads(response.text)

    return json_text["translations"][0]["text"]
#%%
translate_with_yandex("cloud")
#%%
def request_yandex_translate(data, col_name, start_index, end_index):
    start = start_index
    count iters = 0

    for idx in range(start_index, end_index+1, 20):
        start_time = time.time()

        df_twenty = data[start:idx]
        start = idx

        for idx_twenty in df_twenty.index:
            df_twenty.at[idx_twenty, col_name] =
translate_with_yandex(df_twenty.at[idx_twenty, col_name])

        count_iters += 20
        if count_iters > 999:
            data.to_csv("D:/temp.csv", sep='|', encoding='utf-8',
index=False)
            count_iters = 0

        time_difference = time.time() - start_time

        if time_difference < 1.0:
            time.sleep(1.01 - time_difference)
#%% md
## Подсчёт количества знаков
#%%
def count_chars_num(data, start_index, end_index, max_count):
    chars_counter = 0
    last_idx = -1

    for idx in range(start_index, end_index+1):
        chars_counter += len(data.iloc[idx, 2])
        if chars_counter > max_count:
            last_idx = idx
            break

    return chars_counter, last_idx
#%%
counter, idx = count_chars_num(df, 0, 16460, 1000000)

print(idx)
print(counter)
#%% md
## Выбрать по миллиону символов

(1) 0 -- 16460
(2) 16461 -- 33234
(3) 33235 -- 50124
```

```
(4) 50125 -- 67337
(5) 67338 -- 80713
#%%
first_idx = 16461
second_idx = 33234

df_to_trans = df[first_idx:second_idx]
df_to_trans.shape
#%%
df_to_trans.to_csv("D:/million-flickr-30k-images-metadata.csv", sep='|',
encoding='utf-8', index=False)
#%% md
## Перевести миллион символов
#%%
df_to_trans = pd.read_csv("D:/YandexDisk/datasets/million-flickr-30k-images-
metadata.csv.csv", delimiter='|')
df_to_trans.tail(5)
#%%
%%time
request_yandex_translate(df_to_trans, " comment", 0, 16772)
#%%
df_to_trans.to_csv("D:/russian-flickr-30k-images-metadata-0.csv", sep='|',
encoding='utf-8', index=False)
#%% md
## Объединить переведённые данные
#%%
df1 = pd.read_csv("D:/YandexDisk/datasets/russian-flickr-30k-images-metadata-
1.csv", delimiter='|')
df2 = pd.read_csv("D:/YandexDisk/datasets/russian-flickr-30k-images-metadata-
2.csv", delimiter='|')
df3 = pd.read_csv("D:/YandexDisk/datasets/russian-flickr-30k-images-metadata-
3.csv", delimiter='|')
df4 = pd.read_csv("D:/YandexDisk/datasets/russian-flickr-30k-images-metadata-
4.csv", delimiter='|')
df5 = pd.read_csv("D:/YandexDisk/datasets/russian-flickr-30k-images-metadata-
5.csv", delimiter='|')

df_concated = pd.concat([df1, df2, df3, df4, df5])
#%%
df_concated.shape
#%%
df_concated.to_csv("D:/russian-flickr-30k-images-metadata.csv", sep='|',
encoding='utf-8', index=False)
```

Файл 13-clear-translated-data.ipynb:

```
path_captions = "D:/YandexDisk/datasets/russian-flickr-30k-images-
metadata.csv"
df = pd.read_csv(path_captions, delimiter='|')
df.rename(columns={'image_name': 'image_name', ' comment_number':
'comment_number', ' comment': 'comment'}, inplace=True)
df.head(5)
#%% md
# Проверка

- числа
- кавычки
- апострофы
- удалить точки
#%%
def is_there_number(sentence):
    for i in sentence.split():
        if i.isdigit():
            return True
```

```
    return False
#%%
def is_there_english(sentence):
    sentence = sentence.lower()

    alphabet = set('abcdefghijklmnopqrstuvwxyz')
    if any((char in alphabet) for char in sentence):
        return True
    else:
        return False
#%%
idxs_to_del = []

for idx in df.index:
    curr_str = df.iat[idx, 2]

    if is_there_english(curr_str):
        idxs to del.append(idx)
    elif is_there_number(curr_str):
        idxs_to_del.append(idx)
    elif curr_str.find('"') != -1:
        idxs_to_del.append(idx)
    elif curr_str.find("'") != -1:
        idxs_to_del.append(idx)

    if curr_str.find('.') != -1:
        df.iat[idx, 2] = curr_str.replace(".", "")
#%% md
## Удаление

- числа
- кавычки
- апострофы (время)
- английский буквы (имена собственные)
#%%
df_to_delete = df.loc[idxs_to_del, :]
images_to_delete = df_to_delete["image_name"].unique()

print(len(images_to_delete))
print(df.shape)
#%%
df = df[~df.image_name.isin(images_to_delete)]
df.shape
#%% md
## Сохранить
#%%
df.to_csv("D:/temp-russian-flickr-30k-images-metadata.csv", sep='|',
encoding='utf-8', index=False)
```

Файл 14-prepare-data.ipynb:

```
import os
import shutil
import pickle
import string
import pandas as pd
#%%
curr_folder = "D:/YandexDisk/datasets/"

start_dir = "D:/datasets/flickr-images-30k"
end_dir = "D:/datasets/flickr-images-12k"

path_captions = curr_folder + "captions-ru-12k.csv"
```

```
path_captions_no_puncts = curr_folder + "captions-ru-12k-no-puncts.csv"

path_train = curr_folder + "captions-ru-12k-train.csv"
path_val = curr_folder + "captions-ru-12k-val.csv"
path_test = curr_folder + "captions-ru-12k-test.csv"

path_features = curr_folder + "ru-12k-features.pkl"
path_vocab = curr_folder + "ru-12k-vocab.pkl"
path_sentences = curr_folder + "ru-12k-sentences-train.pkl"

path_train_dict = curr_folder + "captions-ru-12k-train.pkl"
path_val_dict = curr_folder + "captions-ru-12k-val.pkl"
#%% md
# Удалить длинные предложения
#%%
df = pd.read_csv(path_captions, sep='|')
#%%
idxs_to_del = []

for idx in df.index:
    curr_len = len(df.iat[idx, 2])
    if curr_len > 100:
        idxs_to_del.append(idx)

df_to_delete = df.loc[idxs_to_del, :]
images_to_delete = df_to_delete["image_name"].unique()
#%%
print(df.shape)
df = df[~df.image_name.isin(images_to_delete)]
print(df.shape)
#%% md
# Подготовка данных к обучению

- Каждое слово с маленькой буквы
- Удалить знаки препинания
- Только буквы
#%%
def clean_captions(data):
    table = str.maketrans('', '', string.punctuation)

    for idx in data.index:
        curr_capt = data.iat[idx, 2]
        curr_capt = curr_capt.split()
        curr_capt = [word.lower() for word in curr_capt]
        curr_capt = [word.translate(table) for word in curr_capt]
        curr_capt = [word for word in curr_capt if word.isalpha()]

        data.iat[idx, 2] = ' '.join(curr_capt)
#%%
df = pd.read_csv(path_captions, delimiter='|')
df.head(5)
#%%
clean_captions(df)
df.head()
#%% md
# Словарь
#%%
def to_vocab(data):
    vocab = set()

    for idx in data.index:
        vocab.update(data.iat[idx, 2].split())

    return vocab
#%%
```

```
df = pd.read_csv(path_captions_no_puncts, sep='|')
vocab = to_vocab(df)
print('размер словаря ... %d' % len(vocab))
#%%
with open(path_vocab,'wb') as f:
    pickle.dump(vocab, f)
#%%
with open(path_vocab,'rb') as f:
    loaded_vocab = pickle.load(f)
#%% md
# Разбить набор на три части
#%%
df = pd.read_csv(path_captions_no_puncts, delimiter='|')
df.head(5)
#%%
n = len(df)

train_df = df[0:int(n*0.7)]
val_df = df[int(n*0.7):int(n*0.9)]
test_df = df[int(n*0.9):]

train_df.to_csv(path_train, sep='|', encoding='utf-8', index=False)
val_df.to_csv(path_val, sep='|', encoding='utf-8', index=False)
test_df.to_csv(path_test, sep='|', encoding='utf-8', index=False)
#%% md
# Добавить начальные и конечные строки startseq и endseq
#%%
def add_start_end_tags(data):
    for idx in data.index:
        curr_str = data.iat[idx, 2]
        data.iat[idx, 2] = 'startseq ' + curr_str + ' endseq'
#%%
train_df = pd.read_csv(path_train, sep='|')
val_df = pd.read_csv(path_val, sep='|')
test_df = pd.read_csv(path_test, sep='|')

add_start_end_tags(train_df)
add_start_end_tags(val_df)
add_start_end_tags(test_df)

train_df.to_csv(path_train, sep='|', encoding='utf-8', index=False)
val_df.to_csv(path_val, sep='|', encoding='utf-8', index=False)
test_df.to_csv(path_test, sep='|', encoding='utf-8', index=False)
#%%
test_df.iat[0, 2]
#%% md
# Конвертация pandas.dataframe в dict
#%%
def to_dict(data):
    out_dict = dict()

    start_index = 0
    end_index = len(data) - 1 - 5

    for idx in range(start_index, end_index+1, 5):
        image_name = data.iat[idx, 0][:-4]
        curr_list = list()

        curr_list.append(data.iat[idx, 2])
        curr_list.append(data.iat[idx+1, 2])
        curr_list.append(data.iat[idx+2, 2])
        curr_list.append(data.iat[idx+3, 2])
        curr_list.append(data.iat[idx+4, 2])

        out_dict[image_name] = curr_list
```

```
    return out_dict
#%%
df_to_convert = pd.read_csv(path_val, sep='|')
df_to_convert.head()
#%%
new_dict = to_dict(df_to_convert)
#%%
with open(path_val_dict,'wb') as f:
    pickle.dump(new_dict, f)
#%%
with open (path_val_dict, 'rb') as f:
    test_dict = pickle.load(f)
#%% md
# Предложения для обучения
#%%
def to_sentences(data):
    all_sentences = list()

    for idx in data.index:
        all_sentences.append(data.iat[idx, 2])

    return all_sentences
#%%
train_df = pd.read_csv(path_train, delimiter='|')
sentences = to_sentences(train_df)
print(len(sentences))
#%%
with open(path_sentences, 'wb') as f:
    pickle.dump(sentences, f)
#%%
with open (path_sentences, 'rb') as f:
    list_sentences = pickle.load(f)
#%% md
# Выбрать и скопировать изображения
#%%
df = pd.read_csv(path_captions, sep='|')
images_unique = df["image_name"].unique()
print(images_unique.shape)
print(df.shape)
#%%
for image_name in images_unique:
    curr_image = start_dir + '/' + image_name
    copied_image = end_dir + '/' + image_name
    shutil.copy2(curr_image, copied_image )

counter = len(os.listdir(path=end_dir))
print("скопировано изображений ... " + str(counter))
```

Файл 21-build-models.ipynb:

```
import os
import pickle
import random
import time
import numpy as np

from tensorflow.python.client import device_lib
from tensorflow.python.keras.layers import RepeatVector, TimeDistributed,
Bidirectional

from keras_preprocessing.text import Tokenizer
from keras_preprocessing.sequence import pad_sequences
from keras_preprocessing.image import load_img, img_to_array
```

```
from keras.utils import to_categorical
from keras.models import Model
from keras.layers import Input
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Embedding
from keras.layers import Dropout
from keras.layers.merge import add
#%% md
# Расположение файлов
#%%
curr_folder = "D:/YandexDisk/datasets/"
end_dir = "D:/datasets/flickr-images-12k"

path_tokenizer = curr_folder + "ru-12k-tokenizer-train.pkl"
path_train_dict = curr_folder + "captions-ru-12k-train.pkl"
path_val_dict = curr_folder + "captions-ru-12k-val.pkl"

path_features_vgg16 = curr_folder + "ru-12k-features.pkl"
path_features_inception = "D:/features-inception"
#%% md
# Проверка видеокарты
#%%
print(device_lib.list_local_devices())
#%% md
# Загрузка данных
```

Мы собираемся обучить данные по всем фотографиям и подписям в обучающем наборе данных. Во время обучения мы будем отслеживать производительность модели в наборе данных разработки и использовать эту производительность, чтобы решить, когда сохранять модели в файл.

Модель, которую мы разработаем, будет генерировать подпись к фотографии, и подпись будет генерироваться по одному слову за раз.

Последовательность ранее сгенерированных слов будет предоставлена в качестве входных данных. Поэтому нам понадобится "первое слово", чтобы начать процесс генерации, и "последнее слово", чтобы сигнализировать об окончании подписи. Для этой цели мы будем использовать строки "startseq" и "endseq". Эти маркеры добавляются к загруженным описаниям по мере их загрузки. Важно сделать это сейчас, прежде чем мы закодируем текст, чтобы токены также были закодированы правильно.

```
#%%
def image_names_set(data):
    vals = set()

    for idx in data.index:
        vals.add(data.iat[idx, 0][:-4])

    return vals

def load_image_features(filename, data):
    all_features = pickle.load(open(filename, 'rb'))
    features = {k: all_features[k] for k in data}

    return features
#%% md
# Закодировать знаки в числа
```

Текст описания необходимо будет закодировать в числа, прежде чем его можно будет представить модели в качестве входных данных или сравнить с предсказаниями модели.

Первым шагом в кодировании данных является создание согласованного

сопоставления слов с уникальными целочисленными значениями. Keras предоставляет класс Tokenizer, который может изучить это сопоставление из загруженных данных описания.

Каждое описание будет разделено на слова. Модель будет предоставлена одним словом и фотографией и сгенерирует следующее слово. Затем первые два слова описания будут предоставлены модели в качестве входных данных вместе с изображением для создания следующего слова. Именно так будет обучаться модель.

```
#%%
def to_lines(data):
    all_vals = list()
    for key in data.keys():
        [all_vals.append(d) for d in data[key]]

    return all_vals

def create_tokenizer(data):
    lines = to_lines(data)
    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(lines)

    return tokenizer

def find_max_words(data):
    lines = to_lines(data)
    return max(len(l.split()) for l in lines)
#%% md
# Создание последовательности
```

Приведенная ниже функция с именем create_sequences(), учитывая токенизатор, максимальную длину последовательности и словарь всех описаний и фотографий, преобразует данные в пары ввода-вывода данных для обучения модели.

В модели есть два входных массива: один для признаков фотографии и один для закодированного текста. Существует один вывод для модели, который представляет собой закодированное следующее слово в текстовой последовательности.

Входной текст кодируется в виде целых чисел, которые будут передаваться на слой встраивания слов. Признаки изображения будут передаваться непосредственно в другую часть модели. Модель выдаст прогноз, который будет представлять собой распределение вероятностей по всем словам в словаре.

Таким образом, выходные данные будут представлять собой однократно закодированную версию каждого слова, представляющую идеализированное распределение вероятностей со значениями 0 во всех позициях слов, кроме фактической позиции слова, которая имеет значение 1.

```
#%%
def create_sequences(tokenizer, max_words, captions_list, image_name):
    X_image, X_text, y_word = list(), list(), list()
    vocab_size = len(tokenizer.word_index) + 1

    for caption in captions_list:
        seq = tokenizer.texts_to_sequences([caption])[0]

        for i in range(1, len(seq)):
            in_seq, out_seq = seq[:i], seq[i]
            in_seq = pad_sequences([in_seq], maxlen=max_words)[0]
            out_seq = to_categorical([out_seq], num_classes=vocab_size)[0]

            X_image.append(image_name)
            X_text.append(in_seq)
            y_word.append(out_seq)
```

```
    return X_image, X_text, y_word
#%% md
# Генератор данных
```

Генератор данных будет выдавать данные на одно изображении в каждой партии. Это будут все последовательности, сгенерированные для изображения и её набора описаний.

Функция data_generator() будет генератором данных и будет принимать загруженные текстовые описания, признаки изображений, токенизатор и максимальную длину.8 ГБ оперативной памяти должно быть более чем достаточно.

Вы можете видеть, что мы вызываем функцию create_sequence (), чтобы создать пакет данных для одного изображения, а не для всего набора данных. Это означает, что мы должны обновить функцию create_sequences (), чтобы удалить “итерацию по всем описаниям” для цикла.

Генератор данных, предназначен для использования в вызове model.fit generator().

Обратите внимание, что это очень простой генератор данных. Большая экономия памяти, которую он предлагает, заключается в том, чтобы не иметь развернутых последовательностей обучающих и тестовых данных в памяти до подгонки модели, чтобы эти образцы (например, результаты create_sequences()) создавались по мере необходимости для каждого изображения.

Некоторые нестандартные идеи для дальнейшего совершенствования этого генератора данных включают в себя:
- Рандомизируйте порядок фотографий каждой эпохи.
- Работайте со списком идентификаторов изображений и загружайте текст и данные изображений по мере необходимости, чтобы ещё больше сократить объём памяти.
- Получите более одного изображения в партии.

```
#%%
def data_generator(tokenizer, max_words, data, images, batch_size,
random_seed):
    count = 0
    random.seed(random_seed)

    img_names = list(data.keys())
    assert batch_size <= len(img_names), 'batch size must be less than or
equal to {}'.format(len(img_names))

    while True:
        input_img_batch, input_seq_batch, output_word_batch = list(), list(),
list()

        if count >= len(img_names):
            count = 0
        start_i = count
        end_i = min(len(img_names), count + batch_size)

        for i in range(start_i, end_i):
            curr_img = img_names[i]
            image = images[curr_img][0]
            captions_list = data[curr_img]
            random.shuffle(captions_list)

            input_img, input_seq, output_word = create_sequences(tokenizer,
max_words, captions_list, image)

            for j in range(len(input_img)):
                input_img_batch.append(input_img[j])
                input_seq_batch.append(input_seq[j])
                output_word_batch.append(output_word[j])
```

```
        count = count + batch_size
        yield [np.array(input_img_batch), np.array(input_seq_batch)],
np.array(output_word_batch)
#%% md
# Построение модели

Мы опишем модель в трёх частях:

1 - Извлечение признаков изображения. Это 16-слойная модель VGG,
предварительно обученная на наборе данных ImageNet. Мы предварительно
обработали изображения с помощью модели VGG (без выходного слоя) и будем
использовать извлечённые признаки, предсказанные этой моделью, в качестве
входных данных.

2 - Обработка последовательностей. Это слой встраивания слов для обработки
ввода текста, за которым следует слой рекуррентной нейронной сети с
длительной кратковременной памятью (LSTM).

3 - Расшифровка. (1) и (2) выводят вектор фиксированной длины. Они
объединяются вместе и обрабатываются плотным слоем, чтобы сделать
окончательный прогноз.

Модель (1) ожидает, что входные признаки изображений будут вектором из 4096
элементов. Они обрабатываются плотным слоем для получения 256-элементного
представления изображения.

Модель (2) ожидает входные последовательности с заранее определённой длиной,
которые подаются в слой встраивания, использующий маску для игнорирования
дополненных значений. За этим следует слой LSTM с 256 единицами памяти.

Обе входные модели создают вектор из 256 элементов. Кроме того, обе входные
модели используют регуляризацию в виде 50% отсева (dropout). Это делается для
того, чтобы уменьшить переобучение модели на текущем наборе данных, так как
эта конфигурация модели обучается очень быстро.

Модель (3) объединяет векторы из обеих входных моделей с помощью операции
сложения. Затем этот вектор подаётся на плотный слой из 256 нейронов, а затем
на конечный выходной плотный слой, который делает прогноз softmax по всему
выходному словарю для следующего слова в последовательности.
#%%
def build_rnn(input_size, vocab_size, max_words):
    inputs1 = Input(shape=(input_size,))
    fe1 = Dropout(0.5)(inputs1)
    fe2 = Dense(256, activation='relu')(fe1)

    inputs2 = Input(shape=(max_words,))
    se1 = Embedding(vocab_size, 256, mask_zero=True)(inputs2)
    se2 = Dropout(0.5)(se1)
    se3 = LSTM(256)(se2)

    de1 = add([fe2, se3])
    de2 = Dense(256, activation='relu')(de1)
    outputs = Dense(vocab_size, activation='softmax')(de2)

    model = Model(inputs=[inputs1, inputs2], outputs=outputs)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    return model
#%%
def build_alt_rnn(input_size, vocab_size, max_words):
    image_input = Input(shape=(input_size,))
    fe1 = Dense(256, activation='relu')(image_input)
    image_model = RepeatVector(max_words)(fe1)
```

```
    caption_input = Input(shape=(max_words,))
    # we zero pad inputs to the same length, the zero mask ignores those
inputs
    se1 = Embedding(vocab_size, 256, mask_zero=True)(caption_input)
    # since we are going to predict the next word using the previous words
    # (length of previous words changes with every iteration over the
caption), we have to set return_sequences = True
    se2 = LSTM(256, return_sequences=True)(se1)
    caption_model = TimeDistributed(Dense(256))(se2)

    # merging the models and creating a softmax classifier
    de1 = add([image_model, caption_model])
    de2 = Bidirectional(LSTM(256, return_sequences=False))(de1)
    final_model = Dense(vocab_size, activation='softmax')(de2)

    model = Model(inputs=[image_input, caption_input], outputs=final_model)
    model.compile(loss='categorical_crossentropy', optimizer='adam')

    return model
#%% md
# Обучение

В этом примере мы отбросим загрузку тестового набора данных и контрольные
точки модели и просто сохраним модель после каждой эпохи обучения. Затем мы
сможем вернуться и загрузить/оценить каждую сохраненную модель после
обучения, чтобы найти ту, которая имеет наименьшие потери.
#%%
def load_train_data(train_dict_path, tokenizer_path, features_path):
    with open (train_dict_path, 'rb') as f:
        out_train_dict = pickle.load(f)
    print('кол-во подписей ..... %d' % len(out_train_dict))

    out_train_features = load_image_features(features_path, out_train_dict)

    with open (tokenizer_path, 'rb') as f:
        out_tokenizer = pickle.load(f)
    out_vocab_size = len(out_tokenizer.word_index) + 1
    print('размер словаря ..... %d' % out_vocab_size)

    out_max_words = find_max_words(out_train_dict)
    print('длина предложения в словах ... %d' % out_max_words)

    return out_train_dict, out_tokenizer, out_vocab_size, out_max_words,
out_train_features

def train_model(model, train_dict, tokenizer, max_words, train_features,
batch_size, epochs_num):
    steps = len(train_dict)/batch_size
    if len(train_dict) % batch_size != 0:
        steps = steps + 1

    start_time = time.time()
    for i in range(epochs_num):
        generator = data_generator(tokenizer, max_words, train_dict,
train_features, batch_size, 42)
        model.fit(generator,
                  epochs=1, steps_per_epoch=steps,
                  verbose=1)
        model.save('model-' + str(i) + '.h5')

    time_difference = time.time() - start_time
    minutes = time_difference/60
    print('время обучения в минутах ..... %d' % minutes)
#%% md
### Набор для обучения
```

```
#%%
batch_size = 16
epochs_num = 20
train_dict, tokenizer, vocab_size, max_words, train_features =
load_train_data(path_train_dict, path_tokenizer, path_features_inception)
#%% md
# Обучение VGG16
#%%
model = build_rnn(4096, vocab_size, max_words)
train_model(model, train_dict, tokenizer, max_words, train_features,
batch_size, epochs_num)
#%%
model = build_alt_rnn(4096, vocab_size, max_words)
train_model(model, train_dict, tokenizer, max_words, train_features,
batch_size, epochs_num)
#%% md
# Обучение Inception
#%%
from keras.applications.inception v3 import InceptionV3
from keras.applications.inception_v3 import preprocess_input as
preprocess_input_i

def extract_features_inception(directory):
    model = InceptionV3(weights="imagenet")
    model = Model(inputs=model.inputs, outputs=model.layers[-2].output)

    features = dict()
    for name in os.listdir(directory):
        filename = directory + '/' + name

        image = load_img(filename, target_size=(299, 299))
        image = img_to_array(image)
        image = image.reshape((1, image.shape[0], image.shape[1],
image.shape[2]))
        image = preprocess_input_i(image)

        feature = model.predict(image, verbose=0)
        image_id = name.split('.')[0]
        features[image_id] = feature

    return features
#%%
%%time
features = extract_features_inception(end_dir)
print('выделенные признаки ... %d' % len(features))
pickle.dump(features, open(path_features_inception, 'wb'))
#%%
model = build_rnn(2048, vocab_size, max_words)
train_model(model, train_dict, tokenizer, max_words, train_features,
batch_size, epochs_num)
#%%
model = build_alt_rnn(2048, vocab_size, max_words)
train_model(model, train_dict, tokenizer, max_words, train_features,
batch_size, epochs_num)
```

Файл 23-plot-evaluations.ipynb:

```
import matplotlib.pyplot as plt
import operator
#%%
plt.rcParams["figure.figsize"] = (6,6)
plt.title("Измерения BLEU")

y_bleu_vgg16 = [0.361409, 0.406490, 0.468829, 0.512229, 0.501154,
```

```
                0.501107, 0.494001, 0.489138, 0.487587, 0.482318,
                0.477577, 0.476834, 0.473901, 0.472866, 0.464693,
                0.463840, 0.472879, 0.461818, 0.459094, 0.464013]

y_bleu_inception = [0.308462, 0.401386, 0.465643, 0.507810, 0.501669,
                    0.491208, 0.475721, 0.478230, 0.484276, 0.485589,
                    0.486937, 0.484933, 0.483060, 0.476378, 0.462925,
                    0.478496, 0.478996, 0.465538, 0.471662, 0.475187]

x_epochs = range(1, 21)
plt.plot(x_epochs, y_bleu_vgg16, "red", label="VGG16")
plt.plot(x_epochs, y_bleu_inception, "blue", label="Inception")

plt.legend()
plt.show()
#%%
index, value = max(enumerate(y_bleu_vgg16), key=operator.itemgetter(1))
index +=1
print("VGG16 ..... эпоха ... " + str(index) + " ... максимальное измерение
BLUE ... " + str(value))

index, value = max(enumerate(y_bleu_inception), key=operator.itemgetter(1))
index +=1
print("Inception ... эпоха ... " + str(index) + " ... максимальное измерение
BLUE ... " + str(value))
```

Файл 24-check-models.ipynb:

```
from pickle import load
from numpy import argmax

from keras.models import Model
from tensorflow.python.keras.models import load_model
from keras_preprocessing.image import load_img, img_to_array
from keras_preprocessing.sequence import pad_sequences

from keras.applications.inception_v3 import InceptionV3
from keras.applications.vgg16 import VGG16
from keras.applications.inception_v3 import preprocess_input as
preprocess_input_i
from keras.applications.vgg16 import preprocess input as preprocess input v

import matplotlib.pyplot as plt
import matplotlib.image as mpimg

import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'
import tensorflow as tf
#%%
def map_int_to_word(integer, tokenizer):
    for word, idx in tokenizer.word_index.items():
        if idx == integer:
            return word

    return None

def generate_caption(model, tokenizer, max_words, image):
    in_text = 'startseq'

    for i in range(max_words):
        seq = tokenizer.texts_to_sequences([in_text])[0]
        seq = pad_sequences([seq], maxlen=max_words)

        y_hat = model.predict([image,seq], verbose=0)
```

```
        y_hat = argmax(y_hat)

        word = map_int_to_word(y_hat, tokenizer)
        if word is None:
            break

        in_text += ' ' + word

        if word == 'endseq':
            break

    return in_text

def print_image(path_image):
    image = mpimg.imread(path_image)
    plt.imshow(image)
    plt.show()

def extract_features(path_image, extractor, target_size, model_type):
    image = load_img(path_image, target_size=(target_size, target_size))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1],
image.shape[2]))
    if model_type == "vgg16":
        image = preprocess_input_v(image)
    elif model_type == "inception":
        image = preprocess_input_i(image)

    feature = extractor.predict(image, verbose=0)

    return feature

def generate_and_print_captions(path_image, extractor, target_size,
models_paths, model_type):
    image_features = extract_features(path_image, extractor, target_size,
model_type)

    for i, path in enumerate(models_paths):
        model = load_model(path)
        caption = generate_caption(model, tokenizer, max_words,
image_features)
        print(str(i) + ' ' + caption)
#%%
curr_folder = "D:/YandexDisk/datasets/"
path_tokenizer = curr_folder + "ru-12k-tokenizer-train.pkl"
tokenizer = load(open(path_tokenizer, 'rb'))
max_words = 22

vgg16_extractor = VGG16()
vgg16_extractor = Model(inputs=vgg16_extractor.inputs,
outputs=vgg16_extractor.layers[-2].output)
vgg16_target_size = 224

inception_extractor = InceptionV3(weights="imagenet")
inception_extractor = Model(inputs=inception_extractor.inputs,
outputs=inception_extractor.layers[-2].output)
inception_target_size = 299

v0 = "D:/models/vgg16/model-0.h5"
v1 = "D:/models/vgg16/model-1.h5"
v2 = "D:/models/vgg16/model-2.h5"
v3 = "D:/models/vgg16/model-3.h5"
v4 = "D:/models/vgg16/model-4.h5"
v5 = "D:/models/vgg16/model-5.h5"
v6 = "D:/models/vgg16/model-6.h5"
```

```
v7 = "D:/models/vgg16/model-7.h5"
v8 = "D:/models/vgg16/model-8.h5"
v9 = "D:/models/vgg16/model-9.h5"
v10 = "D:/models/vgg16/model-10.h5"
v11 = "D:/models/vgg16/model-11.h5"
v12 = "D:/models/vgg16/model-12.h5"
v13 = "D:/models/vgg16//model-13.h5"
v14 = "D:/models/vgg16/model-14.h5"
v15 = "D:/models/vgg16/model-15.h5"
v16 = "D:/models/vgg16/model-16.h5"
v17 = "D:/models/vgg16/model-17.h5"
v18 = "D:/models/vgg16/model-18.h5"
v19 = "D:/models/vgg16/model-19.h5"

v0_2 = "D:/models/vgg16-2/model-0.h5"
v1_2 = "D:/models/vgg16-2/model-1.h5"
v2_2 = "D:/models/vgg16-2/model-2.h5"
v3_2 = "D:/models/vgg16-2/model-3.h5"
v4_2 = "D:/models/vgg16-2/model-4.h5"
v5_2 = "D:/models/vgg16-2/model-5.h5"
v6_2 = "D:/models/vgg16-2/model-6.h5"
v7_2 = "D:/models/vgg16-2/model-7.h5"
v8_2 = "D:/models/vgg16-2/model-8.h5"
v9_2 = "D:/models/vgg16-2/model-9.h5"
v10_2 = "D:/models/vgg16-2/model-10.h5"
v11_2 = "D:/models/vgg16-2/model-11.h5"
v12_2 = "D:/models/vgg16-2/model-12.h5"
v13_2 = "D:/models/vgg16-2//model-13.h5"
v14_2 = "D:/models/vgg16-2/model-14.h5"
v15_2 = "D:/models/vgg16-2/model-15.h5"
v16_2 = "D:/models/vgg16-2/model-16.h5"
v17_2 = "D:/models/vgg16-2/model-17.h5"
v18_2 = "D:/models/vgg16-2/model-18.h5"
v19_2 = "D:/models/vgg16-2/model-19.h5"

i0 = "D:/models/inception/model-0.h5"
i1 = "D:/models/inception/model-1.h5"
i2 = "D:/models/inception/model-2.h5"
i3 = "D:/models/inception/model-3.h5"
i4 = "D:/models/inception/model-4.h5"
i5 = "D:/models/inception/model-5.h5"
i6 = "D:/models/inception/model-6.h5"
i7 = "D:/models/inception/model-7.h5"
i8 = "D:/models/inception/model-8.h5"
i9 = "D:/models/inception/model-9.h5"
i10 = "D:/models/inception/model-10.h5"
i11 = "D:/models/inception/model-11.h5"
i12 = "D:/models/inception/model-12.h5"
i13 = "D:/models/inception/model-13.h5"
i14 = "D:/models/inception/model-14.h5"
i15 = "D:/models/inception/model-15.h5"
i16 = "D:/models/inception/model-16.h5"
i17 = "D:/models/inception/model-17.h5"
i18 = "D:/models/inception/model-18.h5"
i19 = "D:/models/inception/model-19.h5"

i0_2 = "D:/models/inception-2/model-0.h5"
i1_2 = "D:/models/inception-2/model-1.h5"
i2_2 = "D:/models/inception-2/model-2.h5"
i3_2 = "D:/models/inception-2/model-3.h5"
i4_2 = "D:/models/inception-2/model-4.h5"
i5_2 = "D:/models/inception-2/model-5.h5"
i6_2 = "D:/models/inception-2/model-6.h5"
i7_2 = "D:/models/inception-2/model-7.h5"
i8_2 = "D:/models/inception-2/model-8.h5"
```

```
i9_2 = "D:/models/inception-2/model-9.h5"
i10_2 = "D:/models/inception-2/model-10.h5"
i11_2 = "D:/models/inception-2/model-11.h5"
i12_2 = "D:/models/inception-2/model-12.h5"
i13_2 = "D:/models/inception-2/model-13.h5"
i14_2 = "D:/models/inception-2/model-14.h5"
i15_2 = "D:/models/inception-2/model-15.h5"
i16_2 = "D:/models/inception-2/model-16.h5"
i17_2 = "D:/models/inception-2/model-17.h5"
i18_2 = "D:/models/inception-2/model-18.h5"
i19_2 = "D:/models/inception-2/model-19.h5"

v_models = [v0, v1, v2, v3, v4, v5, v6, v7, v8, v9,
            v10, v11, v12, v13, v14, v15, v16, v17, v18, v19]

v_2_models = [v0_2, v1_2, v2_2, v3_2, v4_2, v5_2, v6_2, v7_2, v8_2, v9_2,
              v10_2, v11_2, v12_2, v13_2, v14_2, v15_2, v16_2, v17_2, v18_2,
v19_2]

i_models = [i0, i1, i2, i3, i4, i5, i6, i7, i8, i9,
            i10, i11, i12, i13, i14, i15, i16, i17, i18, i19]

i_2_models = [i0_2, i1_2, i2_2, i3_2, i4_2, i5_2, i6_2, i7_2, i8_2, i9_2,
            i10_2, i11_2, i12_2, i13, i14, i15, i16, i17, i18, i19]
#%% md
## Изображение 1
#%%
print_image('D:/downloads/00.jpg')
#%%
print("VGG16:")
generate_and_print_captions('D:/downloads/00.jpg', vgg16_extractor,
vgg16_target_size, v_models, "vgg16")
#%%
print("alt VGG16:")
generate_and_print_captions('D:/downloads/00.jpg', vgg16_extractor,
vgg16_target_size, v_2_models, "vgg16")
#%%
print("Inception:")
generate_and_print_captions('D:/downloads/00.jpg', inception_extractor,
inception_target_size, i_models, "inception")
#%%
print("alt Inception:")
generate_and_print_captions('D:/downloads/00.jpg', inception_extractor,
inception_target_size, i_2_models, "inception")
#%% md
## Изображение 2
#%%
print_image('D:/downloads/01.jpg')
#%%
print("VGG16:")
generate_and_print_captions('D:/downloads/01.jpg', vgg16_extractor,
vgg16_target_size, v_models, "vgg16")
#%%
print("alt VGG16:")
generate_and_print_captions('D:/downloads/01.jpg', vgg16_extractor,
vgg16_target_size, v_2_models, "vgg16")
#%%
print("Inception:")
generate_and_print_captions('D:/downloads/01.jpg', inception_extractor,
inception_target_size, i_models, "inception")
#%%
print("alt Inception:")
generate_and_print_captions('D:/downloads/01.jpg', inception_extractor,
inception_target_size, i_2_models, "inception")
#%% md
```

```
## Изображение 3
#%%
print_image('D:/downloads/02.jpg')
#%%
print("VGG16:")
generate_and_print_captions('D:/downloads/02.jpg', vgg16_extractor,
vgg16_target_size, v_models, "vgg16")
#%%
print("alt VGG16:")
generate_and_print_captions('D:/downloads/02.jpg', vgg16_extractor,
vgg16_target_size, v_2_models, "vgg16")
#%%
print("Inception:")
generate_and_print_captions('D:/downloads/02.jpg', inception_extractor,
inception_target_size, i_models, "inception")
#%%
print("alt Inception:")
generate_and_print_captions('D:/downloads/02.jpg', inception_extractor,
inception target size, i 2 models, "inception")
#%% md
## Изображение 4
#%%
print_image('D:/downloads/03.jpg')
#%%
print("VGG16:")
generate_and_print_captions('D:/downloads/03.jpg', vgg16_extractor,
vgg16_target_size, v_models, "vgg16")
#%%
print("alt VGG16:")
generate_and_print_captions('D:/downloads/03.jpg', vgg16_extractor,
vgg16_target_size, v_2_models, "vgg16")
#%%
print("Inception:")
generate_and_print_captions('D:/downloads/03.jpg', inception_extractor,
inception_target_size, i_models, "inception")
#%%
print("alt Inception:")
generate_and_print_captions('D:/downloads/03.jpg', inception_extractor,
inception_target_size, i_2_models, "inception")
#%% md
## Изображение 5
#%%
print_image('D:/downloads/04.jpg')
#%%
print("VGG16:")
generate_and_print_captions('D:/downloads/04.jpg', vgg16_extractor,
vgg16_target_size, v_models, "vgg16")
#%%
print("alt VGG16:")
generate_and_print_captions('D:/downloads/04.jpg', vgg16_extractor,
vgg16_target_size, v_2_models, "vgg16")
#%%
print("Inception:")
generate_and_print_captions('D:/downloads/04.jpg', inception_extractor,
inception_target_size, i_models, "inception")
#%%
print("alt Inception:")
generate_and_print_captions('D:/downloads/04.jpg', inception_extractor,
inception_target_size, i_2_models, "inception")
```

Файл 31-use-models.ipynb:

```
from pickle import load
```

```
from tensorflow.python.keras.models import load_model
from tensorflow.python.keras.models import Model
from tensorflow.python.keras.applications.vgg16 import VGG16
from tensorflow.python.keras.applications.inception_v3 import InceptionV3

from keras_preprocessing.image import load_img, img_to_array
from keras.applications.vgg16 import preprocess_input as vgg16_preprocess
from keras.applications.inception_v3 import preprocess_input as
inception_preprocess

from keras_preprocessing.sequence import pad_sequences
from numpy import argmax

from matplotlib import image as mpimg, pyplot as plt

from os import system, environ
environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

PATH_TOKENIZER = "tokenizer.pkl"
PATH_VGG16 = "bleu-0512229-vgg16.h5"
PATH_INCEPTION = "bleu-0507810-inception.h5"

TOKENIZER = load(open(PATH_TOKENIZER, 'rb'))
MAX_WORDS = 22

VGG16_EXTRACTOR = VGG16(weights="imagenet")
VGG16_EXTRACTOR = Model(inputs=VGG16_EXTRACTOR.inputs,
outputs=VGG16_EXTRACTOR.layers[-2].output)
VGG16_TARGET_SIZE = 224
VGG16_MODEL = load_model(PATH_VGG16)

INCEPTION_EXTRACTOR = InceptionV3(weights="imagenet")
INCEPTION_EXTRACTOR = Model(inputs=INCEPTION_EXTRACTOR.inputs,
outputs=INCEPTION_EXTRACTOR.layers[-2].output)
INCEPTION_TARGET_SIZE = 299
INCEPTION_MODEL = load_model(PATH_INCEPTION)

def vgg16_extract(path_image):
    image = load_img(path_image, target_size=(VGG16_TARGET_SIZE,
VGG16_TARGET_SIZE))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1],
image.shape[2]))
    image = vgg16_preprocess(image)

    feature = VGG16_EXTRACTOR.predict(image, verbose=0)
    return feature

def inception_extract(path_image):
    image = load_img(path_image, target_size=(INCEPTION_TARGET_SIZE,
INCEPTION_TARGET_SIZE))
    image = img_to_array(image)
    image = image.reshape((1, image.shape[0], image.shape[1],
image.shape[2]))
    image = inception_preprocess(image)

    feature = INCEPTION_EXTRACTOR.predict(image, verbose=0)
    return feature

def map_int_to_word(integer):
    for word, idx in TOKENIZER.word_index.items():
        if idx == integer:
            return word

    return None
```

```
def generate_caption(model, image_features):
    in_text = 'startseq'

    for i in range(MAX_WORDS):
        seq = TOKENIZER.texts_to_sequences([in_text])[0]
        seq = pad_sequences([seq], maxlen=MAX_WORDS)

        y_hat = model.predict([image_features,seq], verbose=0)
        y_hat = argmax(y_hat)

        word = map_int_to_word(y_hat)
        if word is None:
            break

        in_text += ' ' + word

        if word == 'endseq':
            break

    return in_text

def vgg16_generate(path_image):
    image_features = vgg16_extract(path_image)
    caption = generate_caption(VGG16_MODEL, image_features)

    return caption[9:-7]

def inception_generate(path_image):
    image_features = inception_extract(path_image)
    caption = generate_caption(INCEPTION_MODEL, image_features)

    return caption[9:-7]

def print_image(path_image):
    image = mpimg.imread(path_image)
    plt.imshow(image)
    plt.show()

def captions_from_file(path_image):
    print_image(path_image)
    print(vgg16_generate(path_image))
    print(inception_generate(path_image))

def captions_from_url(url):
    system("curl -s {} -o {}".format(url, "temp.jpg"))
    captions_from_file("temp.jpg")
#%%
captions_from_url('https://i.imgur.com/QzxTOG1.jpg')
#%%
captions_from_file('D:/downloads/1.jpg')
#%%
captions_from_file('D:/downloads/2.jpg')
#%%
captions_from_file('D:/downloads/3.jpg')
#%%
captions_from_file('D:/downloads/4.jpg')
#%%
captions_from_file('D:/downloads/5.jpg')
```