

ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(СПбГУ)

Направление: Прикладные математика и физика



Полуавтоматическое структурирование изображений
в социальной сети с помощью методов машинного
обучения

Выполнил: А. Р. Шабанов

Руководитель: к.ф.-м.н., А. С. Немнюгин

Рецензент: А. А. Дзюба

Санкт - Петербург
2019

Содержание

Введение	3
1 Постановка задачи	4
2 Используемые инструменты	6
3 Искусственные нейронные сети	7
3.1 Перцептрон	7
3.2 Функции активации	7
3.3 Полносвязная сеть	8
3.4 Функции потерь	9
3.5 Обучение нейронных сетей	10
3.6 Сверточные нейронные сети	11
3.7 Используемые сверточные архитектуры	12
4 Работа с данными	14
4.1 База семантических связей WordNet	14
4.2 Датасет SUN	14
4.3 Адоптация датасета SUN	15
5 Численные эксперименты	19
5.1 Архитектура программы	19
5.2 Выбор метрики	20
6 Ход экспериментов	21
6.1 Исследование модели	25
7 Валидация результатов	29
Выводы	30
Список литературы	31

Введение

С каждым днем пользователи социальных сетей создают и потребляют все большие и большие объемы информации, в том числе огромное количество фотографий. Построение системы для быстрой и точной навигации в миллионах изображений — не тривиальная задача. Одно из самых распространенных решений этой проблемы заключается в использовании тегов, в частности использование хештегов в социальных сетях.

Хештег — это любое слово или фраза без пробелов, перед которой стоит символ `#`, который называется *дизз* или *решетка*, а в англоязычном варианте — *hash*, отсюда и название. Приведем несколько примеров: `#masterwork`, `#spbu`, `#htaglovesport`. Обычно в браузерах или приложениях хештеги отображаются как гипертекст, кликнув по которому можно получить список публикаций, снабженных таким же или похожими хэштегами.

Кроме простоты и удобства использования теги обладают еще одним полезным свойством — они позволяют не думать об иерархии структурируемой информации. Например, набор изображений можно разложить по папкам, создав иерархию по датам, геолокациям или авторству. Причем в отдельных случаях подобрать наиболее подходящую иерархию бывает затруднительно. Проблему можно решить так: достаточно поставить по несколько тегов для всех изображений, которые могут храниться в плоской системе файлов. Благодаря этому свойству тегирование используются для рубрикации контента не только онлайн, но и в офлайн приложениях, например, просмоторщиках фотографий.

В настоящей работе будет описано построение и обучение модели, подсказывающей пользователям социальных сетей популярные хэштеги, релевантные к загружаемым изображениям. Для количественного измерения корректности работы системы кроме расчёта формальных метрик будет приведена точность, оценённая пользователями вручную.

Кроме того, с помощью разрабатываемой системы потенциально можно решать и обратную задачу — определять, уместно ли поставлены те или иные теги к заданным изображениям. Способность системы давать ответ на такой вопрос можно использовать для выявления злоупотреблений со стороны пользователей. Например, зачастую в рекламных целях продвигаемую публикацию снабжают множеством популярных тегов, не имеющих никакого отношения к её содержимому.

1 Постановка задачи

todo Рассказать про другие работы о тегах (например, на фликре).

Данная работа посвящена разработке интеллектуальной системы, подсказывающей пользователю релевантные и широко распространенные хештеги к загружаемым фотографиям. Можно выделить две разновидности популярных хэштегов в социальных сетях. Первые являются выражением чувств, эмоций и других абстрактных понятий, например `#love`, `#friendship`, `#happy`. Вторые непосредственно связаны с объектами, находящимися в кадре, например `#sport`, `#cafe`, `#nature`. Интуитивно понятно, что для второй разновидности тегов предсказательная система, построенная на машинном зрении, будет давать более точные результаты, чем для первой (но будет иметь относительно узкую специализацию). Чтобы построить такую систему нужен подходящий набор обучающих данных, но существующие датасеты, собранные из социальных сетей, не разделяют тренировочные примеры на те, где визуальный контент напрямую связан с целевыми классами, и те, где это не так. В связи с чем возникла идея для данной задачи адаптировать датасет, изначально не размеченный хэштегами, но содержащий аннотации, связанные с объектами в кадре. Кроме того, датасет должен быть относительно небольшим, чтобы не требовать значительных вычислительных ресурсов; но при этом достаточно разнообразным, чтобы быть релевантным как можно большему количеству загружаемого контента. В настоящей работе автор остановил свой выбор на датасете мест и локаций *SUN[2]* (*Scene Understanding Dataset*), разметка которого после некоторой предобработки может быть отображена в популярные хэштеги.

SUN – это набор фотографий, для каждой из которых выбрана одна из четырехсот названий локаций (сцен), вот несколько примеров:

- *baseball field*
- *basketball court*
- *ice shelf*
- *forest*
- *wind farm*

Большинство названий локаций сами по себе не являются популярными тегами из социальных сетей. Поэтому необходимо сопоставить их широко распространенным хэштегам (если это возможно):

- *baseball field*, *basketball court* → `#sport`
- *ice shelf*, *forest* → `#nature`

- *wind farm* → ×

Обученная на таких данных модель по окружению, обнаруженному на пользовательском фото, сможет подсказывать ему подходящий хэштег.

Ясно, что полученная модель будет корректно работать лишь для ограниченного (пусть и большого) домена классов. Следовательно, необходимо обучить её распознавать не входящие в этот домен изображения и не пытаться определить их категорию. Кроме того, в случае низкой уверенности в правильности предсказания так же лучше ничего не делать. По мнению автора, гораздо предпочтительнее не предложить пользователю подходящий хэштег, чем многократно предлагать не очень релевантные варианты.

В качестве моделей компьютерного зрения в работе используются глубокие сверточные нейронные сети. Данный выбор обосновывается тем, что в последние годы сверточные сети отлично зарекомендовали себя для решения задач, связанных с обработкой изображений. Можно привести в пример самый большой и известный конкурс по классификации изображений *ImageNet*[1], который проводится ежегодно с 2010 года, при этом, начиная с 2012 года, классические решения ни разу не побеждали, уступив место нейронным сетям.

Научная новизна работы определяется адаптацией датасета *SUN* для решения задачи о структурировании изображений в социальных сетях.

2 Используемые инструменты

Программные средства

В качестве языка программирования использовался *Python 3.6.7* (сборка *Anaconda*), в качестве среды разработки — *PyCharm Professional 2018.1*, операционная система *Ubuntu 16.04.4 LT*. Использованы следующие сторонние библиотеки для *python*:

- *pytorch, torchvision* — построение и обучение нейронных сетей
- *tensorboardX* — визуальное логирование процесса обучения
- *PIL, opencv, scikit-image, scipy* — обработка изображений
- *matplotlib* — отрисовка графиков
- *numpy* — матричные вычисления
- *pandas* — работа с таблицами
- *nltk* — работа с текстом, в том числе с базой *WordNet*
- *scikit-learn* — библиотека машинного обучения общего плана
- *pip* — пакетный менеджер

Вычислительные мощности

Обучение моделей производилось на удаленном сервере со следующей конфигурацией:

- видеокарта *GEFORCE GTX 1080 Ti*, (11 ГБ видеопамяти)
- процессор *AMD Ryzen Threadripper 1920X 12-Core*
- оперативная память объемом 62 ГБ

3 Искусственные нейронные сети

Настоящая часть работы предназначена для читателя, не знакомого с искусственными нейронными сетями и способами их обучения. Здесь будут приведены теоретические основы глубокого обучения и рассмотрены сверточные архитектуры, используемые в дальнейшей работе.

3.1 Перцептрон

Начнем с рассмотрения одиночного нейрона — перцептрана Розенблатта — базового элемента, содержащегося в большинстве современных нейросетевых архитектур. Перцептрон имеет несколько входов и один выход, значение на котором вычисляется как взвешенная сумма значений входов (рисунок 1). Кроме того, обычно к выходному значению применяется сдвиг и некоторая нелинейная функция, называемая функцией активации нейрона. Ее назначение мы обсудим позже.

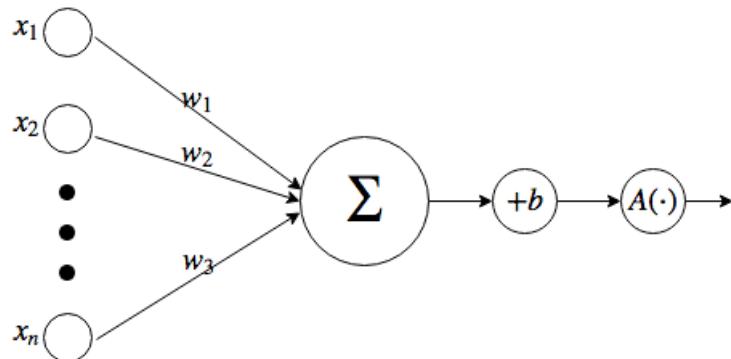


Рис. 1: Схема работы одного отдельного нейрона.

Таким образом, значение на выходе нейрона задается выражением 1.

$$f(\vec{x}) = A\left(\sum_{i=1}^n x_i w_i + b\right) \quad (1)$$

где $f(\vec{x})$ — выходное значение нейрона, посчитанное для входов x_i , w_i — весовые коэффициенты для каждого входа, b — параметр смещения, а A — нелинейная функция активации. Далее для упрощения повествования положим $b \equiv 0$.

3.2 Функции активации

Существуют множество различных функций активации, например, гиперболический тангенс, логистическая сигмоида или $ReLU$, рисунок 2.

$$\begin{aligned}
A_1(x) &= \text{th}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}, \quad \text{th}'(x) = 1 - \text{th}(x)^2 \\
A_2(x) &= \sigma(x) = \frac{1}{1 + e^{-2x}}, \quad \sigma'(x) = \sigma(x)(1 - \sigma(x)) \\
A_3(x) &= \text{ReLU} = \max(0, x), \quad \text{ReLU}'(x) = \theta(x)
\end{aligned} \tag{2}$$

где $\theta(x)$ – функция Хэвисайда.

Эти функции используются для добавления нелинейных зависимостей между слоями многослойной модели. Названные выше функции особенно популярны, так как значения их производных либо достаточно просты, либо легко выражаются через значения самих функций (выражение 2), что позволяет быстро вычислять значение производной.

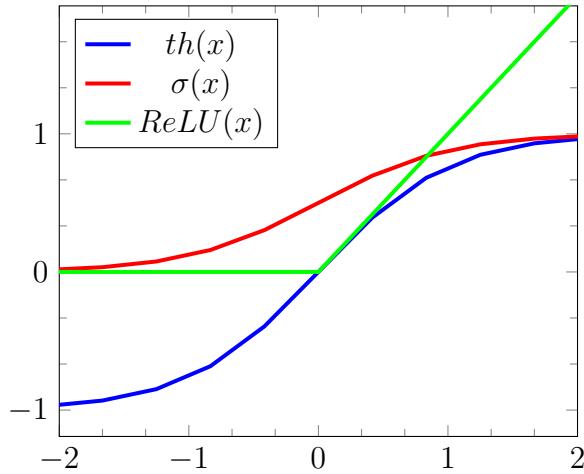


Рис. 2: Функции активации

3.3 Полносвязная сеть

Одиночный нейрон не способен выразить сложные в наборе признаков \vec{x} , поэтому нейроны объединяют в слои, а их, в свою очередь, в многослойные сети. Рассмотрим сеть, состоящую из двух слоев нейронов. Пусть количество входных признаков равно N , количество нейронов скрытого слоя P , а размер выхода – M , рисунок 3. Такая архитектура, состоящая из простых линейных слоев, называется полносвязной.

Рассмотрев выражение 1 можно увидеть, что совокупность значений нейронов на 1 слое может быть получена простым матричным умножением входов \vec{x} на матрицу весов W^1 размера $P \times N$, с последующим поэлементным применением функции активации к получившимся значениям. Аналогично, значения нейронов 2 слоя получаются умножением предыдущих значений на весовую матрицу W^2 размером $M \times P$. Таким образом, применение нейросети ко входу \vec{x} можно задать выражением 3.

$$\vec{y} = A(W^2 A(W^1 \vec{x})) \tag{3}$$

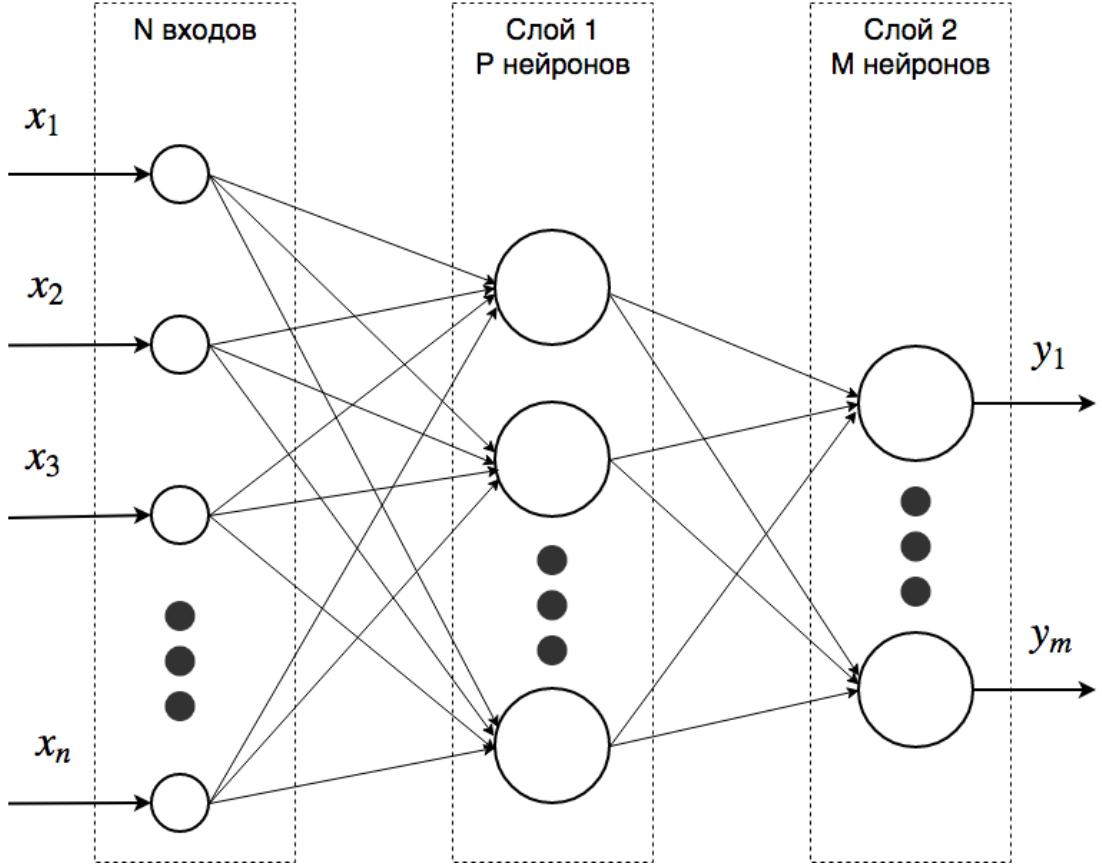


Рис. 3: Схема полносвязной нейронной сети.

где $A(\cdot)$ – применение нелинейности к каждому элементу входного вектора.

Если бы мы не добавляли после слоев нелинейность, то выражение 3 выродилось бы в простое умножение слева входного вектора \vec{x} на матрицу $W^{12} \equiv W^2W^1$. В этом случае модель не смогла бы выучить сложные нелинейные зависимости в данных, а добавление дополнительных слоев не имело бы смысла, т.к. все они могут быть описаны одной матрицей весов.

Отметим, что количество весов сети, состоящей из полносвязных слоев, растёт мультипликативно от их размеров. Поэтому из-за больших вычислительных затрат на практике обычно не встречаются по-настоящему глубокие полносвязные архитектуры. Один из способов выразить сложные зависимости меньшим количеством слоев — использование сверточных слоёв, о которых будет рассказано позднее.

3.4 Функции потерь

Близость предсказания сети к правильным ответам оценивается с помощью функции ошибки, так же называемой функцией потерь (*loss function*). Например, для задачи регрессии в качестве функции потерь может применяться сумма квадратов отклонений (выражение 5), или, в более простом случае – сумма разностей между выходами модели и правильными ответами (выражение 4); для задачи классификации обычно используют

перекрестную энтропию (*cross entropy*, выражение 6).

$$L_{diff}(\vec{y}_{gt}, \vec{y}) = \sum_{i=1}^M |y_i^{gt} - y_i| \quad (4)$$

$$L_{mse}(\vec{y}_{gt}, \vec{y}) = \frac{1}{2} \sum_{i=1}^M (y_i^{gt} - y_i)^2 \quad (5)$$

где \vec{y} – предсказанный моделью M -мерный целевой вектор признаков, \vec{y}_{gt} – правильные значения признаков; в общем случае в задаче регрессии признаки являются вещественными числами $\in (-\inf; +\inf)$.

$$L_{ce}(y_{gt}, \vec{y}) = - \sum_{i=1}^M \delta_{y_{gt,i}} \log y_i \quad (6)$$

где i – метка класса (классы пронумерованы натуральными числами от 1 до M), y_{gt} – правильная метка класса для рассматриваемого экземпляра данных; y_i – предсказанная вероятность того, что экземпляру соответствует i -й класс; δ_i – символ Кронекера.

Исходя из постановки решаемой задачи, можно составить и другие функции ошибок, но они обязательно должны быть дифференцируемыми. Это необходимо условие, чтобы использовать метод обратного распространения ошибки для обучения модели.

3.5 Обучение нейронных сетей

Обучение нейронной сети – это процесс изменения весовых коэффициентов между нейронами, направленный на уменьшение значения функции потерь на тренировочном наборе данных $\{\vec{x}|\vec{y}\}$. Мы рассмотрим базовую реализацию такого процесса – алгоритм стохастического градиентного спуска. Мы будем подавать на вход модели размеченные тренировочные данные, объединенные в небольшие партии (*batches*). Идея состоит в том, чтобы после каждой такой партии добавлять ко всем весам поправку, противоположную градиенту от функции потерь (выражение 7). Отметим, что метод называется стохастическим, так как веса модели обновляются после каждой партии данных. Из-за этого направление, в котором делается шаг по профилю функции потерь, не является оптимальным для обучающего набора данных в целом. Можно интерпретировать этот факт как добавление шума к вычисляемому градиенту, отсюда и стохастичность в названии.

$$w_{i,j} \leftarrow w_{i,j} - \Delta w_{i,j} = -lr \frac{\partial L}{\partial w_{i,j}} \quad (7)$$

где lr – параметр, отвечающий за скорость обучения (*learning rate*).

Для начала положим, что j является нейроном последнего слоя. Обозначим $S_j = \sum_i w_{i,j} x_i$ и распишем производную из выражения 7.

$$\frac{\partial L}{\partial w_{i,j}} = \frac{\partial L}{\partial S_j} \frac{\partial S_j}{\partial w_{i,j}} = x_i \frac{\partial L}{\partial S_j} = x_i \frac{\partial L}{\partial y_j} \frac{\partial y_j}{\partial S_j} = x_i \frac{\partial L}{\partial y_j} \frac{\partial A(S)}{\partial S}|_{S_j} \quad (8)$$

Чтобы сосчитать производные в выражении 8 осталось выбрать какие-то конкретные функции потерь и активации. Выберем в качестве функции активации A логистическую сигмоиду σ (формула 2), а в качестве функции потерь L – квадратичное отклонение (формула 5). После подстановки получим окончательную формулу для обновления весов последнего слоя (формула 9).

$$w_{i,j} \leftarrow w_{i,j} - 2vx_i y_j (1 - y_j)(y_j^{gt} - y_j) \quad (9)$$

Рассмотрим случай, когда j -й узел находится во внутренних слоях и у него есть выходы (обозначим их как $childrens(j)$).

$$\frac{\partial L}{\partial S_j} = \sum_{k \in childrens(j)} \frac{\partial L}{\partial S_k} \frac{\partial S_k}{\partial S_j} \quad (10)$$

В выражении 10 можно явно сосчитать $\frac{\partial S_k}{\partial S_j}$, в нашем случае мы получим $2w_{j,k}y_j(1 - y_j)$, а $\frac{\partial L}{\partial S_k}$ – это поправка для весов, вычисленная для узла следующего уровня (с точностью до коэффициента $-vx_i$). Таким образом, мы можем вычислить поправку для весов нейронов последнего уровня (выражение 9), и использовать её, чтобы вычислить поправки для остальных уровней (выражение 10). Из-за вычислений такого вида данный подход так же называют алгоритмом обратного распространения ошибки (*backpropagation*).

3.6 Сверточные нейронные сети

Сверточные нейронные сети (*Convolution neural network, CNN*) – это искусственные нейронные сети специального вида, изначально сконструированные для обработки изображений, хотя в настоящее время спектр их применения значительно расширился. Как следует из названия, основной таких сетей являются сверточные слои (*convolutional layers*).

Для начала на простом примере рассмотрим, как устроен результат G применения свертки с ядром h к матрице F , формула 11.

$$G = F \otimes h$$

$$G_{i,j} = \sum_{a=-n}^n \sum_{b=-n}^n h_{a,b} F_{s*i-n, s*j-n} \quad \forall i \in (1..I), \forall j \in (1..J) \quad (11)$$

где F – матрица размером $I \times J$; h – ядро свертки размером $2n + 1 \times 2n + 1$; s – шаг, с которым свертка "передвигается" по матрице F (*stride*); G – матрица размером $I \times J$, результат свертки.

Обычно ядро свертки h – это небольшая матрица весов размером 3×3 , 5×5 или 7×7 . В качестве матрицы F может выступать некоторое изображение (в приведенном примере – в оттенках серого), или карта признаков (*feature map*), полученная путём сворачивания изображения с ядрами других сверток. Графическая иллюстрация к операции свертки приводится на рис. 4.

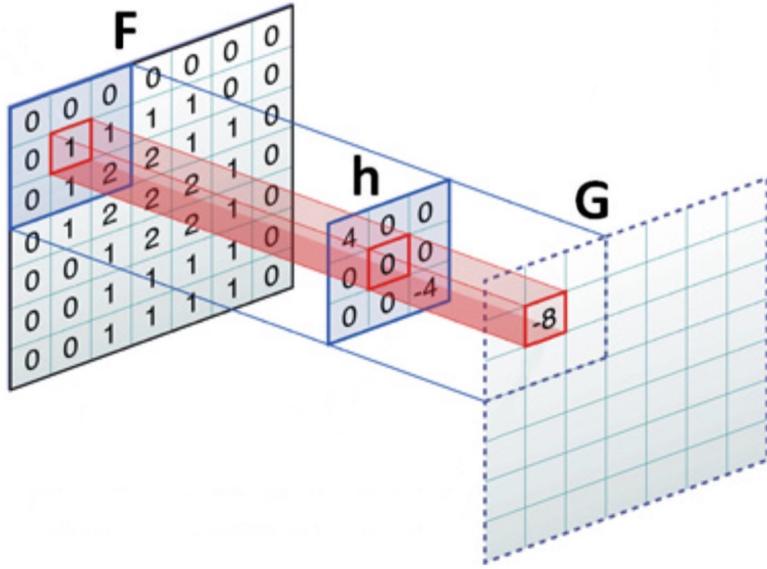


Рис. 4: Свертка матрицы F с ядром свертки h .

В классическом компьютерном зрении известны конкретные числовые значения элементов матрицы h для ряда сверточных ядер специального вида. Например, ядро (чаще называемое оператором) Собеля позволяет выделять границы на изображениях, а ядро (чаще называемое фильтром) Гаусса используется для устранения шума и размытия контуров на изображении. Идея добавления сверточных слоев в нейронные сети строится на том, что сеть сама в процессе обучения методом обратного распространения ошибки "выучит" подходящие для решения конкретной задачи весовые коэффициенты в обучаемых ядрах.

Кроме того, обычно вместе со сверточными слоями используются такие слои как *Pooling*, *Softmax*, *BatchNormalization*[3] и *Dropout*[4].

3.7 Используемые сверточные архитектуры

В данной работе используются следующие архитектуры нейронных сетей:

- Residual network (ResNet) — одна из самых популярных в настоящее время архитектур, предложенная в статье от Microsoft [5]. Основная идея состоит в добавлении конкатенации выходов слоя с номером i и слоя $i - 2$ (такая процедура получила название *skip connection*). Таким образом авторы успешно решают проблему обучения глубоких сетей — затухание градиентов.
- Inception — топология, предложенная исследователями из компании Google [6]. Основную идею можно выразить фразой "сеть внутри сети". Авторы конструируют сеть из блоков (*inception modules*) и используют свертки (1×1).
- VGG — относительно глубокая сеть, авторы которой предложили решение проблемы

обучения огромного количества параметров сверточных слоев [7]. Вместо сверток с ядрами (7×7) и (9×9) они используют несколько сверток (3×3) , уменьшив количество весов, при этом сохранив площадь восприятия (*reception field*).

Для перечисленных выше архитектур в открытом доступе находятся веса обученных моделей, тренированных на классификации 1000 категорий датасета *ImageNet*[1]. В настоящей работе эти веса используются для инициализации слоев обучаемых моделей (для этого требуется заменить размер выходного слоя на количество категорий, рассматриваемых в решаемой задаче). Такой подход имеет широкое распространение и называется *fine-tuning*, он позволяет значительно ускорить тренировочный процесс и, иногда, увеличить конечную точность модели.

4 Работа с данными

4.1 База семантических связей WordNet

Обсуждение работы с данными наиболее логично начать с описания базы знаний *WordNet'a*, который использовался и авторами датасета *SUN*, и автором настоящей работы. Составители датасета *SUN* использовали *WordNet* для создания иерархии названий сцен (локаций). А в настоящей работе он используется для объединения названий локаций в обобщающий домен и для поиска синонимов к предлагаемым пользователю тегам.

WordNet — это электронный словарь/семантическая сеть для английского языка. Он содержит 4 подсети: для глаголов, существительных, прилагательных и наречий. Узлами сети являются не отдельные слова, а синсеты (*synset*), объединяющие слова со схожим значением. Таким образом, слова, имеющие несколько значений могут быть включены в несколько синсетов. Например, слово *rose* соотносится и с синсетом *noun.rose* (роза, куст розы), и с синсетом *verb.rose* (прошедшая форма глагола *rise* — поднимался).

Синсеты связаны между собой различными отношениями. Например, один синсет может выступать по отношению к другому в роли гиперонима (лес → природа), гипонима (природа → лес), меронима (футбольное поле → ворота) и т.д.

Кроме того, между синсетами можно вычислять различные меры близости. Например, *path similarity*, показывающую, насколько похожи синсеты, основываясь на длине кратчайшего пути отношений гиперонимии/гипонимии, который их соединяет *WordNet'a*.

4.2 Датасет SUN

Набор данных *SUN* (*Scenes Understanding Dataset*) впервые был представлен исследовательскому сообществу в 2010 году на конференции CVPR, посвященной компьютерному зрению. Одновременно авторы опубликовали статью [2], в которой приводят различные статистики по датасету; описывают процесс сбора и разметки данных; применяют к задаче распознавания сцен лучшие из имеющихся на тот момент методов.

Датасет представляет собой набор фотографий, на каждой из которых запечатлена одна из 908 локаций, примеры приведены на рисунке 5. Причем часть локаций представлена в двух видах: снаружи и изнутри. Чтобы отличить эти два случая к названиям сцен добавляются слова *outdoor* или *exterier* и *indoor* или *interier* соответственно. Кроме того, в 2012 году авторы для части изображений представили разметку на уровне сегментации объектов: были предоставлены маски для 300 тыс. объектов, каждый из которых относился к одной из 5 тыс. категорий.

Наконец, авторы оставили только хорошо интерпретирующиеся классы сцен, содержащие хотя бы 100 примеров, после чего организовали на этом наборе данных соревнование по машинному обучению. Итоговый датасет для решения задачи классификации, который и будет использоваться в данной работе, содержит 108754 изображений (около 40 ГБ на

жестком диске), каждое из которых отнесено к одному из 397 классов.

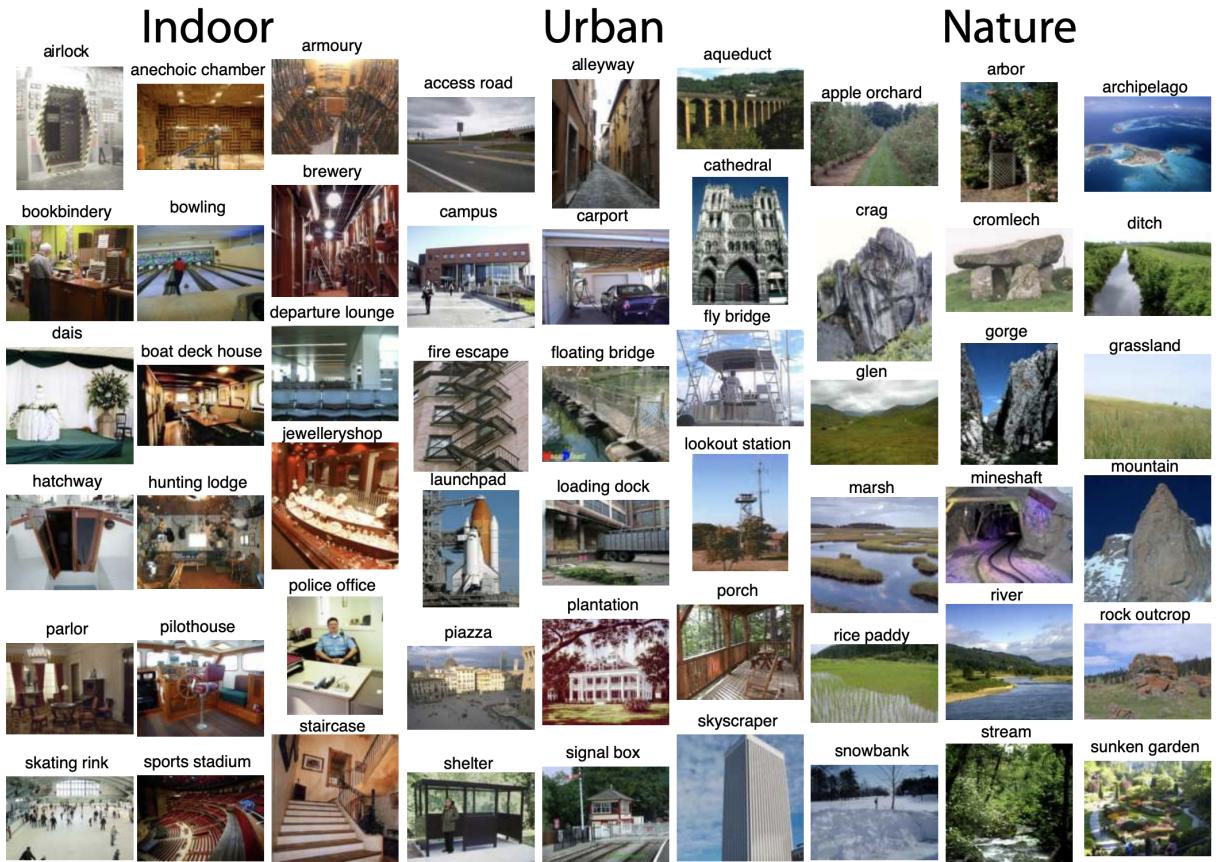


Рис. 5: Примеры размеченных фотографий из датасета *SUN*.

Ознакомиться с другими изображениями из датасета *SUN* можно через интерактивный веб-обозреватель¹, которым можно воспользоваться для просмотра упорядоченных изображений как по сценам, так и по объектам.

4.3 Адаптация датасета *SUN*

Как было сказано в главе, посвященной постановке задачи, названия локаций (сцен) из датасета *SUN* не являются сами по себе популярными тегами из социальных сетей. Поэтому прежде всего необходимо выполнить сопоставление. Условно можно разбить процедуру сопоставления на 2 части: объединение исходных классов датасета *SUN* в семантические домены и сопоставление полученных доменов с популярными хэштегами. В качестве источника хэштегов была выбрана социальная сеть *Instagram*², ориентированная на обмен фото и видео контентом между пользователями.

Итак, предварительно необходимо очистить названия классов *SUN* от служебных слов и символов, таких как *indoor*, *outdoor*, *exterier*, *interier*, знаков "/" и однобуквенных алфа-

¹groups.csail.mit.edu/vision/SUN/

²www.instagram.com

витных указателей. Затем для полученных слов или словосочетаний подбирается соответствующий синсет из базы знаний *Wordnet*. Далее для синсетов находились гиперонимы, которые либо уже были достаточно абстрактны, чтобы представлять собой часто встречающийся тег, либо автор работы находил для синсетов обобщающее понятие вручную. Несколько примеров приведено в таблице 1.

№	Исходное название	Синсет	Гипероним	Хэштег
1	/s/shoe_shop	shoe shop	shop	#shopping
2	/t/toyshop	toyshop	shop	#shopping
3	/v/volleyball_court/indoor	volleyball court	court	#sport
4	/w/wrestling_ring/indoor	wrestling ring	ring	#sport
5	/r/rainforest	rain forest	forest	#forest
6	/p/pantry	pantry	storeroom	?

Таблица 1: Сопоставление искомых классов и хэштегов.

Из таблицы 1 видно, что некоторые синсеты имеют общие гиперонимы. Кроме того, некоторые гиперонимы без каких-либо дополнительных изменений могли быть использованы пользователями в качестве тегов. Таким образом, использование гиперонимов позволило немного уменьшить количество ручной работы. Так же в таблице 1 приведен пример, когда для локации сложно подобрать какой-то подходящий и широко распространенный тег. В итоге использовалось около половины из 397 искомых классов датасета *SUN*, каждому из которых удалось поставить в соответствие один из 20 популярных хэштегов. В данной работе популярными считаются теги, использованные в сети *Instagram* более 10 млн. раз. При этом среднее число упоминаний отобранных тегов составило 100 млн. раз, а максимальное — 450 млн.. Полная информация о встречаемости хэштегов в сети *Instagram* приведена на рисунке 6, а о встречаемости в датасете *SUN* — на рисунке 7.

Отметим, что автор предпринял несколько попыток произвести процедуру адаптации разметки датасета *SUN* полностью автоматически, но они оказались неудачными.

Первая попытка — обобщить искомые классы, используя метод *topic_domains*, который доступен для синсетов в *python* реализации API к базе *WordNet*. Например, для синсета *basketball_court.n.01*, который определяется как *the court on which basketball is played*, вызов данного метода возвращает *basketball.n.02*, который определяется так: *a game played on a court by two opposing teams of 5 players; points are scored by throwing the ball through an elevated horizontal hoop*. К сожалению, проблема заключалась в том, что для подавляющего числа названий локаций *topic_domains* возвращал пустое значение, т.е. для этих синсетов авторами базы знаний не было назначено доменов.

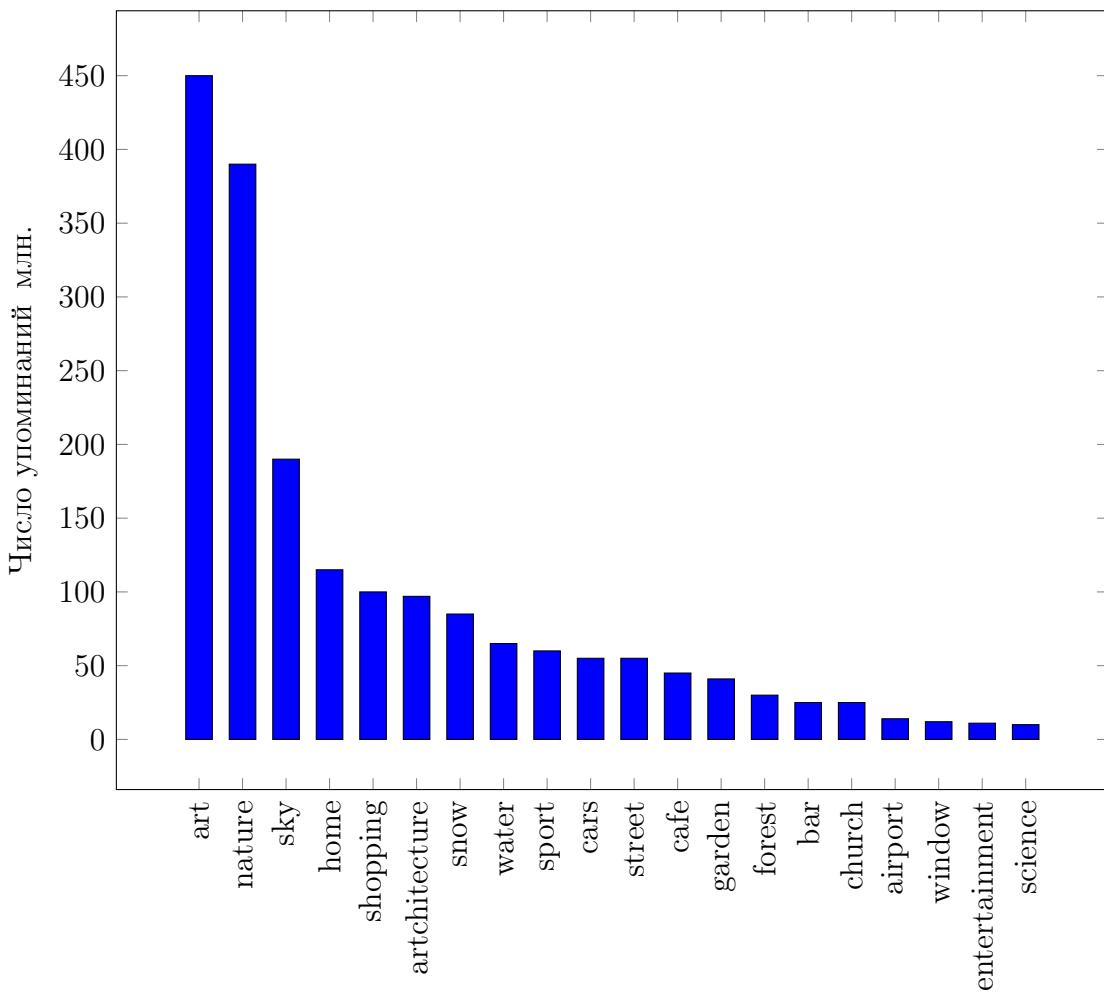


Рис. 6: Встречаемость отобранных хэштегов в социальной сети *Instagram*.

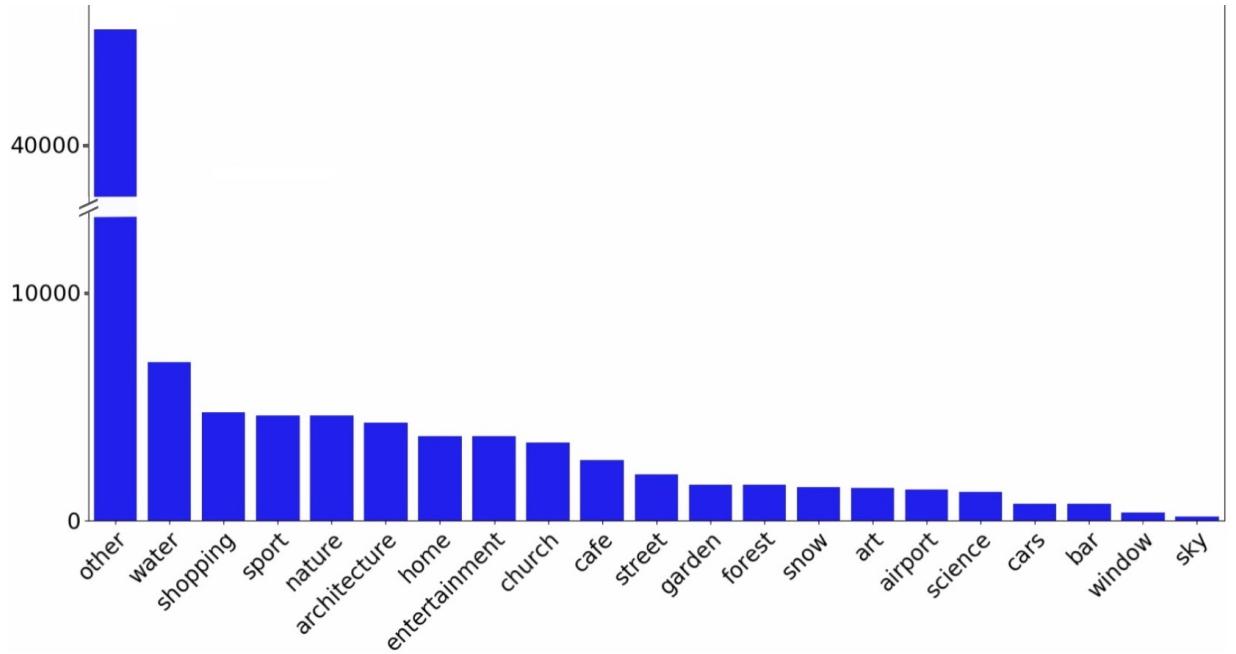


Рис. 7: Распределение хэштегов в датасете *SUN*.

Вторая попытка была аналогична первой, но использовалась сторонняя база знаний *WordNet Domains*³. К сожалению, такое расширение базы доменов не позволило решить проблему, описанную выше.

Третья идея заключалась в использовании информации о семантической близости синсетов, в частности, API *WordNet'a* позволяет для любых двух синсетов вычислить степень похожести несколькими способами: *jcn_similarity*, *lch_similarity*, *res_similarity*, *wup_similarity*. Для разных видов измерения расстояния были вычислены матрицы попарных дистанций, на основе которых производилась иерархическая кластеризация с различными гиперпараметрами (например, кластеризация по заданному максимальному расстоянию в кластере, по заданному количеству кластеров). К сожалению, автору не удалось получить кластеры, большую часть которых можно было бы без труда "озаглавить" каким-либо популярным хэштегом.

Таким образом, процедуру сопоставления всех 397 искомых классов датасета *SUN* и хэштегов из социальной сети *Instagram* пришлось произвести в полуручном режиме (опираясь только на гиперонимы), как это было описано выше в данном разделе.

Приведем объединения исходных классов в домен. В таблице 2 перечислены названия локаций, которым поставлен в соответствие хэштег *#sport*.

/w/wrestling_ring/indoor	/a/athletic_field/outdoor	/b/badminton_court/indoor
/b/ball_pit	/b/baseball_field	/b/basketball_court/outdoor
/b/boxing_ring	/b/bullring	/g/golf_course
/g/gymnasium/indoor	/m/martial_arts_gym	/r/racecourse
/r/riding_arena	/s/ski_lodge	/s/ski_resort
/s/ski_slope	/s/squash_court	/s/stadium/baseball
/s/stadium/football	/s/swimming_pool/indoor	/s/swimming_pool/outdoor
/t/tennis_court/indoor	/t/tennis_court/outdoor	/v/volleyball_court/indoor
/v/volleyball_court/outdoor		

Таблица 2: Список классов датасета *SUN*, которым сопоставлен хэштег *#sport*.

Файлы с полной информацией об итоговом сопоставлении искомых классов и хэштегов можно найти в репозитории автора⁴.

³<http://wndomains.fbk.eu/>

⁴github.com/AlekseySh/scenes

5 Численные эксперименты

В данной главе мы обсудим экспериментальную часть работы, начиная от архитектуры программы и заканчивая обсуждением полученных результатов.

5.1 Архитектура программы

Основная часть программы, выполняющая обучение и тестирование модели состоит из стандартного для фреймворка *pytorch* набора взаимосвязанных компонент (классов). Перечислим их:

- *Module* — описывает непосредственно вычислительный граф нейросети. Здесь указаны параметры и количество всех слоев, описаны связи между ними.
- *Dataset* — позволяет итерироваться по набору данных и объединять их в батчи для подачи на вход нейросети.
- *Loss* — вычисляет функцию ошибки/потери между предсказанными моделью и правильными значениями целевой переменной.
- *Optimizer* — совершает шаг градиентного спуска на заданное расстояние, которое определяется скоростью обучения (*learning rate*). А именно, изменяет веса модели так, чтобы уменьшить среднюю ошибку для очередного переданного на вход модели батча данных.
- *Scheduler* — изменяет скорость обучения модели (*learning rate*) с течением времени по заданному правилу.
- *Stopper* — останавливает тренировку при выполнении заданного условия, например, если в течение последних n эпох не произошло увеличения точности модели хотя бы на ϵ .
- *MetricsCalculator* — оценивает точность модели на некоторой размеченной подвыборке данных по заданным метрикам.
- *TensorboardX* — система для визуального логирования обучения модели; позволяет строить графики изменения функции ошибки, метрик и выводить любые другие пользовательские изображения.
- *Trainer* — объединяет воедино компоненты, названные выше. Обучает модель эпоху за эпохой, с заданной частотой проверяет текущую точность на тестовом подмножестве данных. Останавливает тренировку по достижению некоторого критерия. Сохраняет промежуточные веса модели. Визуализирует процесс обучения.

Взаимосвязь между компонентами программы можно проследить на рисунке 8.

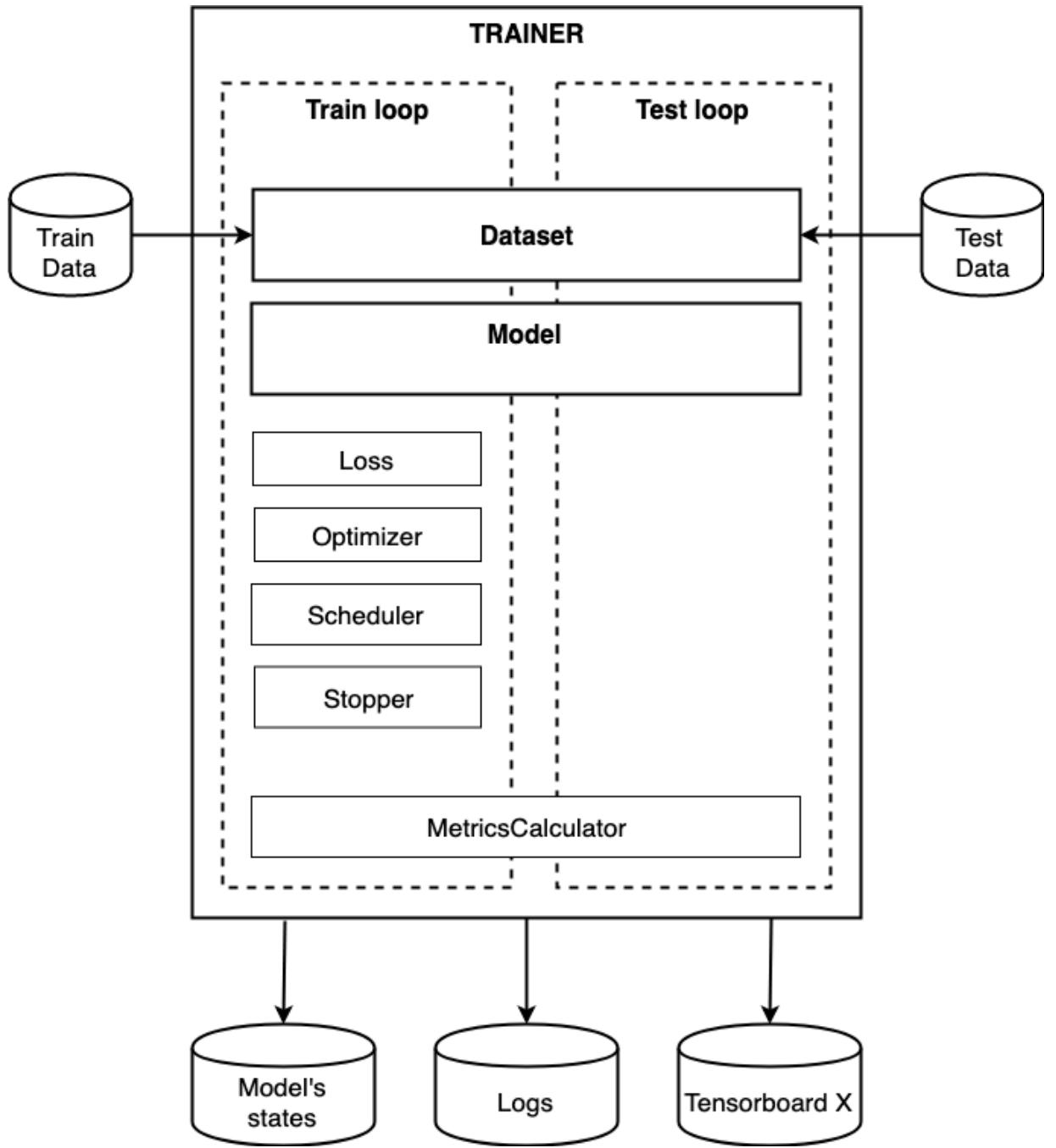


Рис. 8: Структура программы для тренировки и обучения модели.

5.2 Выбор метрики

Прежде всего необходимо определиться, как количественно будет оцениваться точность модели. Мы случайно выбрали 80% данных для тренировки моделей, и 20% для их тестирования. В качестве метрик будем использовать классические для задачи классификации точность (*accuracy*, не путать с *precision*) и взвешенную точность: выражения 12 и 13.

$$acc(\vec{y}^{gt}, \vec{y}) = \frac{1}{N} \sum_{i=1}^N \delta_{y_i^{gt}, y_i} \quad (12)$$

где \vec{y}^{gt} — вектор номеров классов длиной N , \vec{y} — вектор предсказанных номеров классов длиной N , N — количество рассматриваемых примеров, δ — символ Кронекера.

$$acc_w(\vec{y}^{gt}, \vec{y}) = \frac{1}{C} \sum_c \frac{1}{N_c} \sum_{\{i: y_i^{gt} \equiv c\}} \delta_{y_i^{gt}, y_i} \quad (13)$$

где N_c — количество примеров из класса c , C — количество классов.

Таким образом, чтобы посчитать точность достаточно просто разделить количество правильных ответов на общее количество ответов. Чтобы посчитать взвешенную точность, необходимо вычислить точность для каждого класса в отдельности, а затем усреднить полученные значения. В случае, если распределение по классам в значительной степени неравномерное, отсутствие такого усреднения приведет к тому, что значение метрики будет определяться точностью модели на нескольких наиболее представленных классах. В данной работе мы будем смотреть на значения обеих вариантов точности, но для принятия решений о выборе модели приоритетной является взвешенная точность, так как в нашем наборе данных присутствует большой по размеру класс *other*.

6 Ход экспериментов

Чтобы обучить модель оптимальным образом, сначала проведем серию легковесных (с вычислительной точки зрения) экспериментов, чтобы определиться со значениями основных гиперпараметров. Затем попробуем улучшить целевую метрику для лучшей модели, полученной на первом шаге.

Начнём с топологии нейронной сети. Будем выбирать из трех семейств архитектур: *ResNet*, *Inception* и *VGG*; подробнее см. в разделе 3.7. Чтобы ускорить эксперименты уменьшим размер всех изображений до 256×256 пикселей и не будем использовать аугментации во время тренировки. Прекратим обучение после 50-ти эпох⁵, либо остановимся досрочно, если в течение 5-ти последних эпох от текущей не будет улучшено значение метрики хотя бы на 0.5%. В качестве функции потерь используем перекрестную энтропию (см. раздел 3.4), а в качестве оптимизатора параметров нейронной сети — *Adam* [8], т.к. он не требует тонкой настройки параметров. Кроме того, отследим время, которое понадобилось на обучение модели до остановки эксперимента по названному выше критерию. В качестве финального состояния модели выберем то, которое соответствует моменту достижения максимального значения метрики на тестовой выборке (это совсем не обязательно происходит на последней эпохе).

Как видно из результатов предварительных экспериментов (таблица 3), наибольшую точность показала архитектура *Inception v3*. При этом *ResNet34* уступает в точности

⁵Эпохой в машинном обучении называется один полный проход по обучающему набору данных в процессе тренировки.

№	Архитектура	Точность	Взвешенная точность	Время [мин]
1	Inception v3	0.743	0.667	239
2	VGG 11	0.678	0.565	154
3	VGG 13	0.637	0.478	450
4	ResNet 18	0.692	0.602	35
5	ResNet 34	0.703	0.634	119
6	ResNet 50	0.659	0.593	114

Таблица 3: Сравнение метрик для различных архитектур.

незначительно, но требует в 2 раза меньше времени на обучение, поэтому мы остановим свой выбор на этой модели.

Теперь попытаемся улучшить точность базовой модели, используя различные техники тренировки и изменения основные гиперпараметры.

Оптимизатор

Используем другой оптимизатор, например, *SGD* с разными значениями скоростей обучения (*learning rate*). Кроме того, попробуем изменять скорость обучения в процессе тренировки: будем её постепенно понижать и повышать. Подразумевается, что такой подход может помочь вывести функцию потерь из локального минимума. Данный метод описан в статье 2017 года *Stochastic gradient descent with Warm Restarts* [9], хотя его основная идея возникла ещё в 1983 году в публикации *Optimization by simulated annealing* [10]. В соответствии со статьёй, скорость обучения изменяется по формуле 14.

$$lr_i = lr_{min} + \frac{1}{2}(lr_{max} - lr_{min})(1 + \cos(\frac{i}{T}\pi)) \quad (14)$$

где lr – скорость обучения, lr_{min} – минимальная скорость обучения, lr_{max} – максимальная скорость обучения, i – номер текущей эпохи, T – полупериод изменения скорости обучения.

В настоящей работе использованы два набора параметров: $lr_{min} = 0.001$, $lr_{max} = 0.1$ и $T = 3$ и $lr_{min} = 0.001$, $lr_{max} = 0.1$ и $T = 50$. В первом случае мы получаем так называемый косинусный отжиг, так как скорость обучения то увеличивается, то уменьшается. Второй набор параметров соответствует плавному уменьшению скорости обучения в течение всей тренировки: это может помочь быстро найти примерный регион, где находится глобальный минимум целевой функции, а затем уточнить его местоположение. Соответствующие графики изменения скорости обучения приведены на рисунке 9. Разумеется, понять, какая из эвристик окажется лучшей в конкретном случае, поможет только эксперимент.

Размер изображений

Попробуем уменьшать изображения не так сильно, до разрешения 512×512 вместо 256×256 . С одной стороны, это сохранит больше информации, но с другой, увеличит

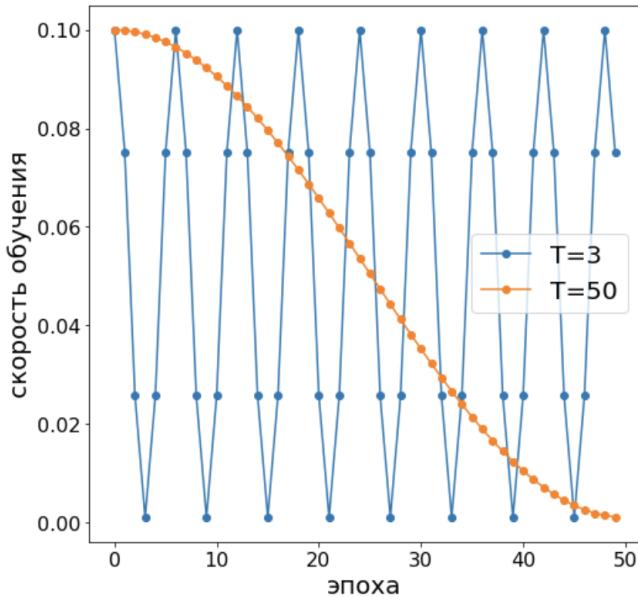


Рис. 9: Изменение скорости обучения в процессе тренировки.

количество пикселей в 4 раза, что уменьшит размер батча⁶ и замедлит тренировку. Поэтому необходимо соблюсти баланс между приростом точности, которого можно добиться использованием изображений большего размера, и увеличением времени тренировки.

Тренировочные аугментации

Применим технику *train time augmentations*, заключающуюся в намеренном случайном искажении изображений, поступающих на вход модели в процессе обучения. Такие преобразования позволяют искусственно раздуть размер тренировочного набора данных, увеличить его вариативность, что, в свою очередь, помогает бороться с переобучением модели⁷. Но возникает вопрос о силе применяемых искажений. Если трансформации слишком слабы, модель быстро адаптируется и её обобщающая способность не будет улучшена. Если слишком сильны, то визуальные паттерны, по которым можно было бы классифицировать изображение, исчезнут. Кроме того, чтобы было удобно подбирать подходящую степень искажений, необходимо каким-то образом количественно её охарактеризовать. Поступим следующим образом: для каждого применяемого преобразования выберем некоторое базовое значение параметра, соответствующее преобразованию средней силы. Затем, одновременно для всех преобразований, будем изменять значения параметров, в k раз увеличивая или уменьшая выбранные базовые значения. Таким образом, при $k = 1$ используются

⁶Батчом называется набор примеров, который передаётся модели на обработку за один раз. При работе с изображениями размер батча ограничивается размером памяти видеокарты, на которой происходят вычисления. Как правило, очередной шаг оптимизатора совершается после прохождения батча, а не отдельного примера.

⁷Переобучением называется ситуация, когда модель запомнила правильные ответы для примеров из тренировочной выборки, но не приобрела обобщающую способность. Другими словами, это ситуация, когда модель имеет хорошее значение метрики на тренировочном наборе, но очень плохое на тестовом.

трансформации с базовыми значениями параметров, при $k = 2$ с удвоенными значениями параметров и так далее. Кроме того, вероятность применить то или иное преобразование так же задаётся параметрически: $p = 0.4 + 0.1k$. Перечислим используемые трансформации и формулы для вычисления их параметров:

- Поворот на угол от $-10k$ до $+10k$ градусов.
- Из изображения вырезается прямоугольник, линейный размер которого выбирается от $1 - 0.1k$ до 1 размера изображения.
- Перенос на величину от 0 до $0.1k$ от линейного размера изображения (по вертикали и горизонтали).
- Сдвиг на величину от 0 до $0.05k$ от линейного размера изображения (по вертикали и горизонтали).
- Изменение отношения сторон в s раз, где s меняется от $1 - 0.1k$ до $1 + 0.1k$ от размера изображения. Причем большей стороной может оказаться как высота, так и ширина изображения.
- Случайное зеркальное отражение с вероятностью p (только по горизонтали).
- Изменения яркости, контраста, оттенка и насыщенности. Степень изменения задаётся числом от $\max(0, 1 - 0.1k)$ до $1 + 0.1k$. Чем дальше это число отстоит от 1 , тем значительнее изменения.

Тестовые аугментации

Применим технику *test time augmentations*. Её смысл заключается в том, что на стадии использования модели входное изображение несколько раз копируется, и к копиям применяются различные искажения из числа тех, которые использовались во время тренировки. После чего модель производит предсказание для каждой копии, а в качестве окончательного ответа вычисляется, например, среднее значение по всем выходам модели. Для многих задач таким образом удаётся улучшить точность предсказаний, но сложность вычислений линейно возрастает с количеством созданных копий. В данной работе изображение копируется 8 раз.

Используем техники, описанные выше, и проведём вторую серию экспериментов, направленных на усовершенствование выбранной ранее модели – *ResNet34*. Но на этот раз изменим критерий остановки: если раньше мы прекращали тренировку, когда значение метрики не увеличивалось хотя бы на 0.5% за последние 5 эпох, то теперь будем ждать 15 эпох. Это связано с тем, что для обучения с использованием аугментаций необходимо больше времени; мы так же будем проводить эксперименты с уменьшенной скоростью обучения, что тоже требует дополнительного времени. Кроме того, уменьшим максимальное

количество эпох для экспериментов с разрешением 512×512 с 50-ти до 30-ти, так одна эпоха для изображений такого размера занимает слишком много времени. Полученные результаты приведены в таблице 4.

№	Оптимизатор	Степень аугментаций (k)	Разрешение	Точность	Взвешенная точность	Время [мин]
1	Adam lr: 0.01	1	256×256	0.756	0.682	232
2	SGD, lr: 0.01	1	256×256	0.800	0.739	320
3	SGD, lr: 0.1	1	256×256	0.802	0.739	412
4	SGD, lr: 0.01	3	256×256	0.815	0.747	302
5	SGD, lr: 0.01	5	256×256	0.771	0.695	369
6	SGD, annealing $T = 3, lr_{max} = 0.1$	1	256×256	0.8113	0.752	275
7	Adam lr: 0.001	2	256×256	0.745	0.688	266
8	SGD, lr: 0.01	1	512×512	0.810	0.781	698
9	SGD, annealing $T = 3, lr_{max} = 0.1$	2	512×512	0.830	0.766	867
10	SGD, annealing $T = 50, lr_{max} = 0.1$	2	512×512	0.818	0.762	889
11	SGD, lr: 0.01	2	512×512	0.820	0.764	813

Таблица 4: Улучшение точности базовой модели (*ResNet34*).

Как видно из результатов второй серии экспериментов, лучшая модель была натренирована на изображениях разрешения 512×512 и используя оптимизатор SGD с постоянной скоростью обучения $lr : 0.01$ и применяя аугментации средней силы. Её точность достигла значения 0.810, а взвешенная точность – 0.781 (таблица 4, №8). Эту модель мы будем использовать для дальнейшего исследования. Отметим, что тренируясь на изображениях размера 256×256 и используя косинусный отжиг, нам удалось добиться почти такой же точности, затратив в три раза меньше времени (таблица 4, №6).

По завершении тренировки было сделано ещё одно предсказание классов для тестовой выборки с использованием описанной выше техники *test time augmentations*. Это позволило незначительно увеличить точность: примерно на 0.01.

6.1 Исследование модели

Итак, нам удалось улучшить базовый вариант тренировки модели *ResNet34*, увеличив значение взвешенной точности на 0.147 – с 0.634 до 0.781 (таблица 4), при этом продолжительность тренировки выросла на порядок.

Теперь попробуем понять, в каких случаях и почему ошибается модель. Построим матрицу ошибок для тестовых 20% данных (рисунок 10), чтобы увидеть, как распределены неправильные предсказания по различным классам (тегам).

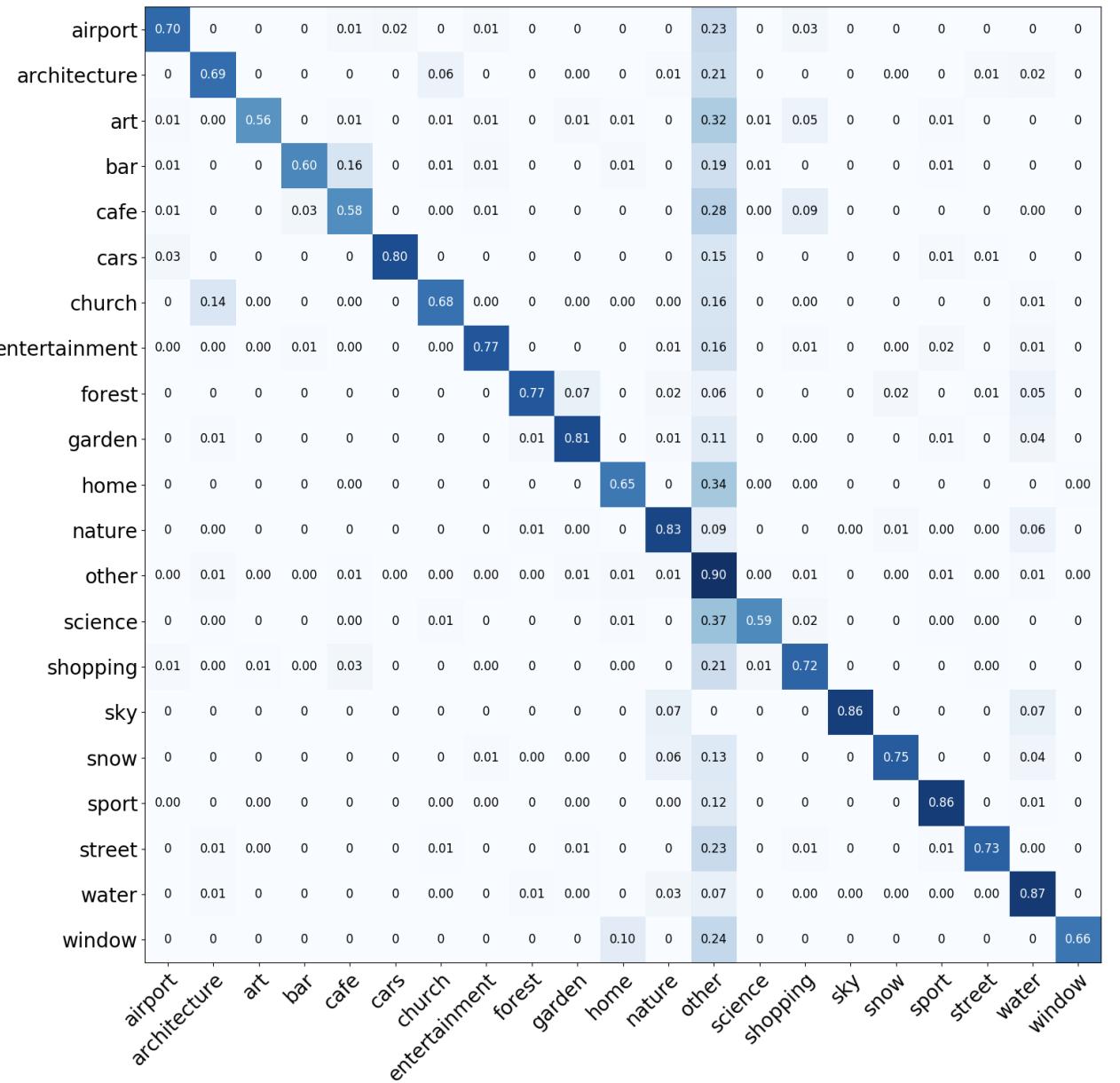


Рис. 10: Матрица ошибок (*confusion matrix*). Истинные метки классов отложены по горизонтали, предсказанные – по вертикали.

Проанализируем матрицу ошибок. Во-первых, её главная диагональ ярко выражена, что хорошо. Во-вторых, предсказания класса *other*, занимающего половину всего датасета, часто оказываются не точны (тёмная вертикальная полоса на рисунке 10). Это ожидаемое явление и оно не представляет большой проблемы, так как мы просто не предложим пользователю какой-то хэштег, хотя могли бы. Было бы хуже, если бы мы предлагали хэштег в случаях, где его быть не должно (такая ситуация соответствовала бы тёмной горизонтальной полосе на изображении матрицы ошибок). Как говорилось ранее,

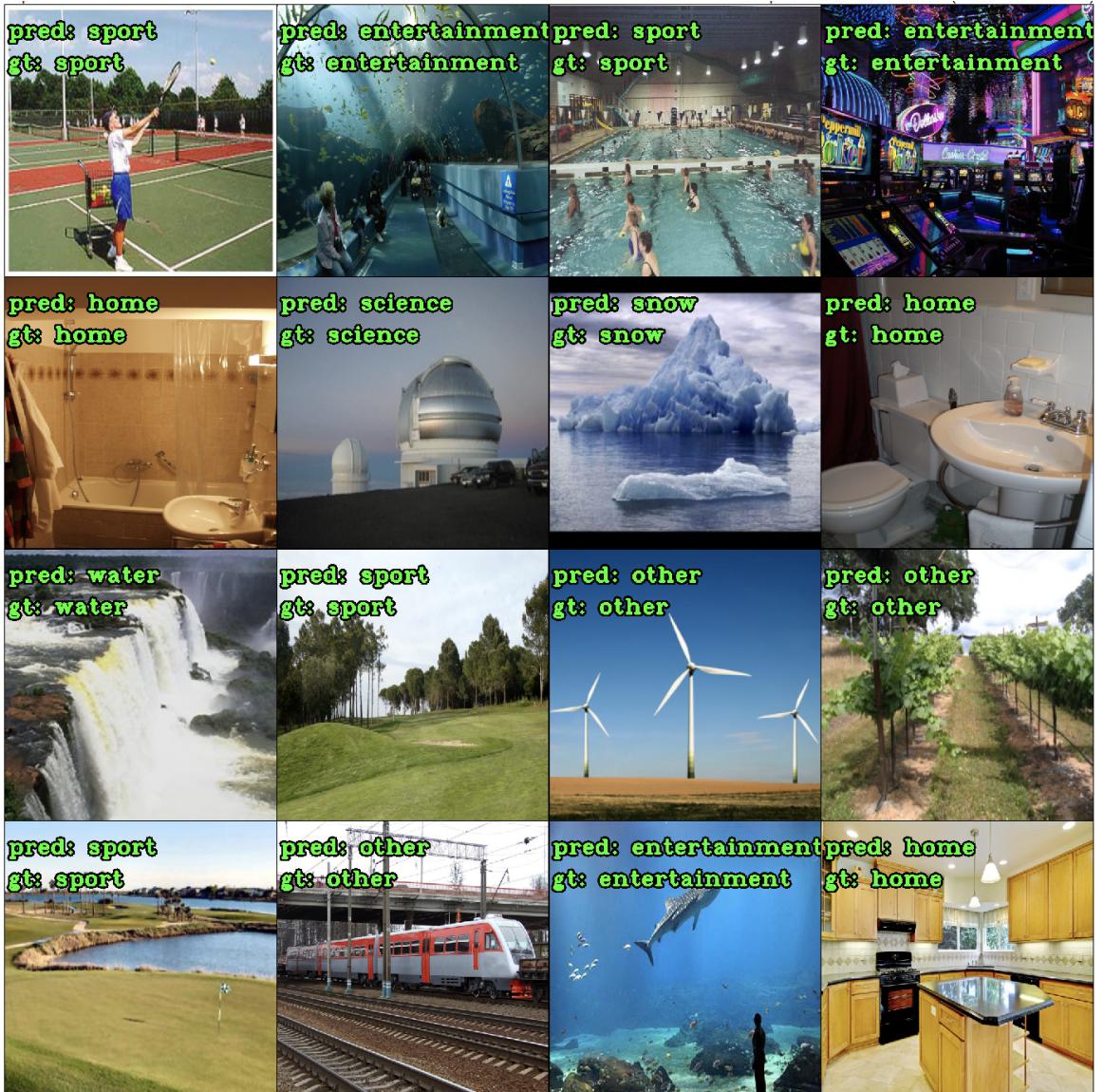


Рис. 11: Примеры, для которых предсказанные () и истинные () классы совпадают.

наша цель – построить модель, которая предсказывает точно, но редко. В-третьих, можно заметить, что путаются похожие классы, некоторые представители которых могут быть отнесены к нескольким классам одновременно. Например, *church* и *architecture*; *bar* и *cafe*; *nature* и *water*.

Наконец, посмотрим, какие теги предсказала модель для конкретных примеров из тестовой выборки. Корректные предсказания изображены на рисунке 11, а неправильные – на 12. Отобраны примеры, для которых сеть сделала предсказания с наибольшим значением уверенности. На рисунке 12 видна проблема для классов, упомянутых выше: модель ошибается на примерах, которые могли бы быть отнесены сразу к нескольким классам.



Рис. 12: Примеры, для которых предсказанные (*predicted*) и истинные (*ground truth*) классы не совпадают.



Рис. 13: Примеры, для которых предсказанные (*predicted*) и истинные (*ground truth*) классы совпадают.

7 Валидация результатов

Эта глава посвящена процедуре валидации предлагаемых хэштегов с привлечением пользователей социальной сети *Одноклассники*⁸.

todo 1. скачиваю изображения из insta 2. предсказываю для них тег моделью 3. отдаю пользователям, которые либо считают предсказанный тег подходящим, либо нет. 4. привожу долю тегов, которые пользователи посчитали подходящими.

⁸ok.ru/

Выводы

В ходе проделанной работы был реализован прототип системы, предлагающей пользователям социальных сетей популярные хэштеги к загружаемым изображениям.

Для тренировки модели, предлагающей хэштеги, был специальным образом адаптирован датасет для распознавания сцен и локаций *SUN*. Обученная на таких данных модель позволяет делать более точные предсказания, так как решение всегда связано с тем, какие объекты находятся в кадре. С другой стороны, такой подход не позволяет предлагать хэштеги, отражающие чувства, эмоции или другие абстрактные понятия.

Основным элементом разработанной системы является свёрточная нейронная сеть. По результатам множества численных экспериментов были выбраны гиперпараметры для обучения модели, соответствующие максимальному значению целевой метрики на тестовом подмножестве данных. Были исследованы паттерны возникновения ошибочных предсказаний.

Чтобы понять, насколько предложения модели адекватны с точки зрения человека, а не формальных метрик, был собран новый набор изображений. Для него с помощью модели были сделаны предсказания и переданы пользователям для проверки вручную. По их мнению в todo% случаев подсказанный хэштег является уместным.

Список литературы

- [1] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. [2015] *ImageNet Large Scale Visual Recognition Challenge*. International Journal of Computer Vision (IJCV).
- [2] J. Xiao, J. Hays, K. Ehinger, A. Oliva, and A. Torralba. [2010] *SUN Database: Large-scale Scene Recognition from Abbey to Zoo*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- [3] S. Ioffe, C. Szegedy. [2015] *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. International Conference on Machine Learning (ICML).
- [4] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. [2014] *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. Journal of Machine Learning Research.
- [5] K. He, X. Zhang, S. Ren, J. Sun. [2015] *Deep Residual Learning for Image Recognition*. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- [6] C. Szegedy et al. [2015] *Going deeper with convolutions* IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
- [7] K. Simonyan, Andrew Zisserman. [2015] *Very Deep Convolutional Networks for Large-Scale Image Recognition*. International Conference on Learning Representations (ICLR).
- [8] D. P. Kingma, J. L. Ba. [2015] *Adam: a method for stochastic optimization*. International Conference on Learning Representations (ICLR).
- [9] I. Loshchilov, F. Hutter. [2017] *SGDR: Stochastic gradient descent with Warm Restarts* International Conference on Learning Representations (ICLR).
- [10] S. Kirkpatrick, C. Gelett, M. Vecchi. [1983] Optimization by simulated annealing. Science.