Аннотация

Эпоха цифрового развития открывает бесконечные возможности применения компьютерных технологий и искусственного интеллекта в нашей жизни. Это касается многих сфер деятельности, таких как транспорт, медицина, промышленное производство, искусство. Однако едва ли можно заметить использование современных технологий в такой области, как туризм. Почему бы это не исправить? Данная работа предлагает идею, как можно развивать сферу туризма и краеведения, вооружившись таким мощным инструментом, как нейронные сети.

Оглавление

Глава 1

Введение

Идея создания проекта, который послужил основой для данной дипломной работы, возникла во время моей прошлогодней поездки в Калининград. Там я познакомился с местным студентом, чей преподаватель увлекается историей и архитектурой Калининградской области. Во время нашего общения преподаватель поделился с нами огромным набором исторических фотографий региона, которые собирал на разных мероприятиях на протяжении нескольких лет, и предложил разработать на основе этих данных проект.

Мы часто представляем себе пешеходные экскурсии по городу таким мероприятием, в котором приходится верить на слова гиду, если он начинает словами описывать, как выглядело некоторое место в прошлом. Иногда гид может показать очень маленьких набор картинок, которые приносит с собой на прогулку. С одной стороны, это не так уж и плохо, ведь слушатели могут сами прорисовывать в своей голове образы. Это можно представить, будто мы читаем книгу, и мы можем самостоятельно представлять в своей голове происходящее, которое описывает словами автор. Однако, далеко не у каждого человека получится наладить такой контакт с описываемым местом. Кроме того, подобный формат экскурсий, как правило, носит локальный характер. Отсутствие централизованной системы приводит к тому, что у региональной сферы туризма крайне слабо развит единый фотоархив, если такой вообще имеется.

Представим себе следующую ситуацию. Турист гуляет по Калининграду и замечает необычное здание, сохранившееся со времён Восточной Пруссии. Ему становится интересно, что это за здание и как оно выглядело раньше. Он фотографирует здание, чтобы потом любым

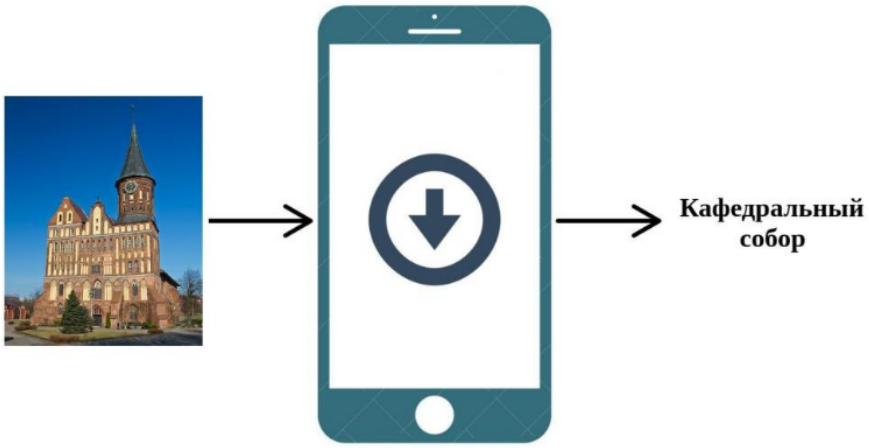

Рис. 1.1: Схема работы приложения.

способом получить нужную информацию.

А теперь представим, насколько было бы удобнее, если у туриста было бы приложение, которое способно предоставлять ему такие сведения, попросив лишь загрузить фотографию архитектурного объекта (рис. 1.1). В качестве таких сведений могла бы выступать как текстовая справка, так и наборы фотографий этого же объекта за разные исторические эпохи (например, Кёнигсберг 1930-х годов, Калининград советского периода или недавней современности). То есть у простого человека может наладиться полноценная межвременная связь с выбранным местом.

Важной особенностью подобного сервиса может оказаться постоянная пополняемость его базы. На начальном этапе существования у разработчиков может не оказаться фотографий некоторых архитектурных объектов города. Описанная ситуация вполне реалистична, поскольку в таком богатом архитектурным наследием городе, как Калининград, сложно перечислить все здания, которые сохранились со времён Восточной Пруссии. Поэтому крайне необходимо проводить регулярные обновления приложения с поступлением новой информации, чтобы оно расширяло свою историческую базу.

Хочется отметить ещё одно важное достоинство такого приложения. Это его географическая универсальность. Подобный сервис можно организовать в любом городе, в котором сохранилось достаточное количество исторического наследия. Единственной преградой для со-

здания полноценного приложения в произвольном городе является отсутствие достаточного количества исторических фотографий, которые необходимы для обучения сервиса.

Проект разрабатывается в команде из 4-х человек, включая меня в роли разработчика машинного обучения. На момент написания дипломной работы проект находится в стадии расширения и поиска потенциальных партнёров для помощи в развитии.

Глава 2

Методы глубокого обучения

Глубокое обучение является молодым разделом машинного обучения. Оно возникло в качестве альтернативы классическому машинному обучению ещё на заре данной науки в середине 20 века. Однако активное развитие новая область получила лишь в 21 веке, когда заметно выросла производительность компьютеров в задачах математических вычислений.

Глубокое обучение действительно является преемником классических методов. Как и предшественник, новый раздел машинного обучения точно так же подразумевает построение компьютерной *модели*, способной достаточно эффективно решать некоторую поставленную *задачу*. Однако стоит отметить основную особенность глубокого обучения, которая долгое время не позволяла проводить успешные исследования эффективности его методов. Новые модели подразумевалось строить таким образом, чтобы они преобразовывали каждый элемент из набора данных в некоторую сущность.

Таким образом, глубокое обучение представляет собой класс методов машинного обучения, работа которых заключается в построении **представлений**. Возникает вопрос, зачем вообще был нужен новый подход? Почему нельзя было обойтись более простыми и понятными методами классического машинного обучения?

На самом деле нетрудно понять, что базовые методы хорошо работают только для сильно ограниченного набора задач. Такая проблема существовала с самого рождения данной науки. Когда возникла

Рис. 2.1: Базовое представление модели машинного обучения.

потребность предсказывать ответы в задачах со сложными зависимостями, где не получается эффективно построить модели *линейной* или *логистической регрессии*, были предложены концепции *дерева* и *ансамблей*. В результате расширения данных идей классическое машинное обучение достигло вершины своего развития в лице *градиентного бустинга*. Этот метод до сих пор является самым популярным и эффективным среди тех, что используются в задачах с числовыми наборами данных.

Для ответа на вопрос о причинах возникновения совершенно нового подхода стоит в очередной раз вспомнить, что из себя представляет машинное обучение в своей основе. Оно было задумано как класс методов, каждый из которых способен с высокой точностью выявлять закономерности в некотором *наборе данных*. При этом в общем случае не требовалось иметь строгого понимания, как именно работает произвольный метод. То есть, в изначальной концепции модель машинного обучения представлялась в виде чёрного ящика, который принимает данные на вход и выдаёт предсказанные ответы на выход (рис. 2.1). Но здесь нигде не подразумевалось, что мы должны явно представлять, что находится внутри этого ящика. В качестве первого шага предлагались именно схемы, где модели будут легко представимы в математическом выражении, чтобы их было проще определять и изменять при обучении.

Появление интереса к применению концепции машинного обучения для работы со сложными типами данных, таких как тексты и изображения, показало необходимость менять подход. Ведь классические модели не смогли бы сделать предсказания, если на вход им подать объект, не являющийся числовым вектором. Тогда был предложена идея преобразовывать сложные объекты в представления, с которыми уже

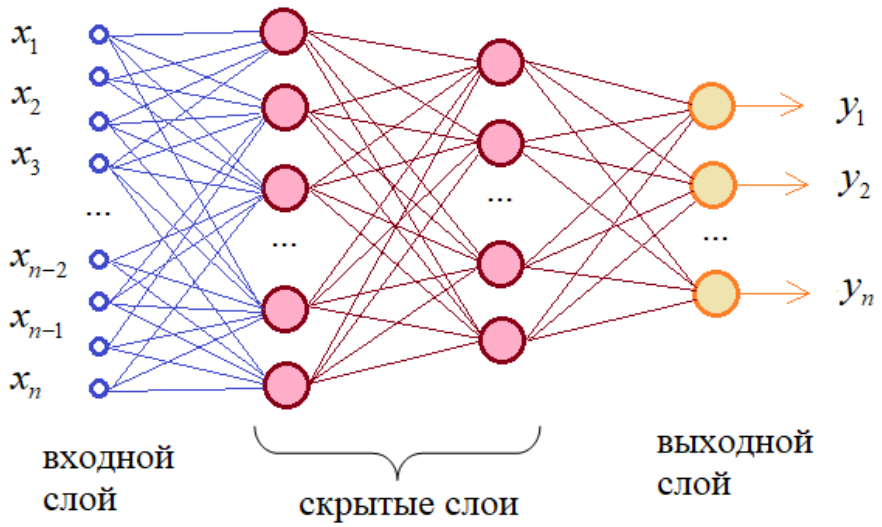

Рис. 2.2: Схема нейронной сети.

смогут работать стандартные методы. Но построить такое преобразование вручную человеку было бы не по силам. Требовалось предлагать большие схемы, которые работают с длинными текстами или крупными изображениями, ставя в соответствие их элементарным фрагментам некоторые числа. Поэтому в таком подходе пришлось полностью отказаться от чёткой интерпретации всех деталей модели глубокого обучения. При этом не стоило забывать, что схема модели должна быть представима в виде математической структуры, для которой можно было бы посчитать градиент и применить оптимизационный метод. Всё это привело к возникновению *нейронных сетей*.

2.1 Нейронная сеть

2.1.1 Структура

Основой глубокого обучения являются **нейронные сети** (рис. 2.2). Это структуры, которые созданы по принципу имитации работы человеческого мозга, представляя собой множеств связанных между собой искусственных элементов. Эти элементы называются **нейронами**. Они были придуманы как математические представления биологических нейронов, которые получают входные сигналы, обрабатывают их и передают выходные сигналы.

Нейроны объединяются в последовательные группы, называемые *слоями*. Слои нейронных сетей обычно делятся на три типа: входные,

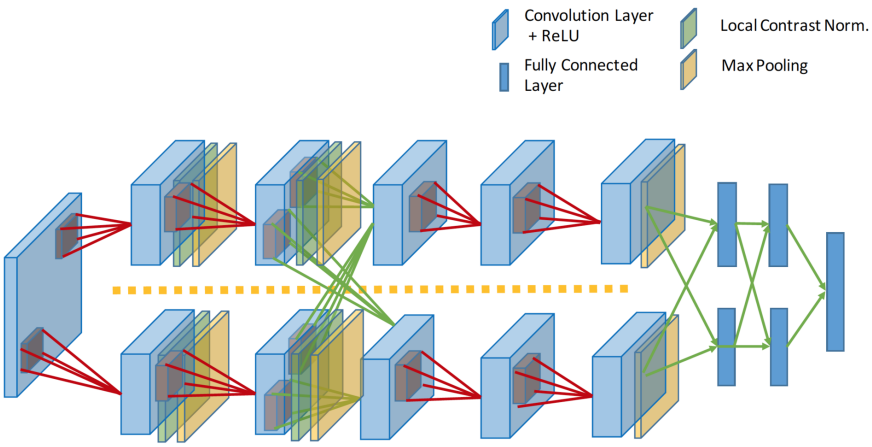

Рис. 2.3: Схема свёрточной нейронной сети.

скрытые и выходные. *Входной* слой отвечает за получение исходных данных. *Скрытые* слои, каковых обычно много, выполняют промежуточные вычисления и трансформации данных. А *выходной* слой выдает окончательный результат.

Каждый нейрон выполняет простую вычислительную операцию: он суммирует входные сигналы от нейронов предыдущего слоя, умноженные на соответствующие *веса*, добавляет *смещение*, применяет нелинейную *активационную функцию* и передаёт результат нейронам следующего слоя.

Для работы с изображениями, которые чаще всего представляются в виде двумерных массивов пикселей, используются **свёрточные нейронные сети** (CNN) (рис. 2.3). В состав свёрточных нейронных сетей входят *свёрточные слои* (convolutional layers), *слои пулинга* (pooling layers) и *полносвязанные слои* (fully connected layers).

Ключевым процессом обработки изображений с помощью CNN является операция **свёртки** (convolution). Она заключается в применении *ядер* (или *фильтров*) к массиву данных по принципу *многомерного скалярного произведения*. То есть во время операции на одном из слоёв массив ядра перемещается по входному массиву. При этом вычисляется сумма произведений элементов ядра и соответствующих элементов входного массива. Результат этих вычислений для каждого из расположений ядра записывается в соответствующую ячейку выходного массива, который передаётся на следующий слой.

Ядра имеются на каждом свёрточном слое. Причём может применяться как одно, так и сразу несколько. При этом разные ядра могут быть определены таким образом, что способны указывать на разные особенности фрагментов изображения. К примеру, одно может подмечать горизонтальные линии, а другое – вертикальные. Стоит также отметить, что ядра, как и изображения, зачастую представляют собой не двумерные, а трёхмерные массивы. Это позволяет работать не только с формой объектов, но и цветом.

Полносвязные слои похожи на свёрточные по своему определению. Однако их нейроны имеют соединения со всеми нейронами предыдущего слоя, из-за чего понятие ядра теряет смысл. Главным недостатком полносвязных слоёв является большое число параметров, что приводит к вычислительным затратам. Свёрточные слои при этом гораздо более легковесные, что и объясняет их частое использование.

2.1.2 Backpropagation

Для обучения нейронной сети используется метод **обратного распространения ошибки** (**backpropagation**). Он представляет собой алгоритм вычисления *градиента* произвольной *функции потерь* по всем праметрам нейронной сети. Подробнее о функциях потерь мы поговорим позже. Сейчас пока ограничимся тем, что у нас есть просто функция, которая оценивает результат работы нейронной сети. Исходя из определения и значения функции мы хотим научиться обновлять веса таким образом, чтобы нейронная сеть выдавала наилучшие результаты. Сделать это можно с помощью градиента функции потерь.

Как мы уже выяснили, нейронная сеть представляет собой последовательность слоёв. Каждый слой производит отображение одного массива в другой. То есть одному слою нейронной сети соответствует некоторая функция, аргументом и значением которой являются тензоры. Таким образом, нейронная сеть может быть описана одной глубокой *сложной функцией*. Это означает, что для нейронной сети можно вычислить градиент.

Как известно из курса математического анализа, градиент сложной тензорной функции равен последовательному умножению градиентов с конца каждой внутренней функции. То есть вычисляется градиент

функции последнего слоя по её входному тензору, затем получившийся тензор умножается на градиент функции предпоследнего слоя по её входному тензору, и т.д.

Теперь, когда чуть подробнее описан процесс вычисления градиента функции потерь, можно сформулировать алгоритм обновления весов нейронной сети с помощью backpropagation:

- *Прямое распространение.* Вычисление предсказания модели.
- *Вычисление ошибки.* Подсчёт разницы между предсказанным и истинным значениями.
- *Обратное распространение.* Вычисление градиентов и обновление весов.

Однако данный алгоритм может оказаться бесполезным, если не учитывать одну существенную проблему работы функции потерь.

2.1.3 Проблема затухания градиента

Нам уже известно, что изменение весов нейронной сети зависит от величины градиента функции потерь. Причём эта зависимость, как правило, линейная. То есть чем меньше значение градиента функции потерь, тем незаметнее будут обновляться веса нейронной сети. В случае с классическим машинным обучением эта ситуация не вызывала больших проблем, поскольку градиент относился к простой функции. Однако в глубоком обучении существует неприятная особенность, связанная с затуханием градиента функции потерь.

Проблема затухания градиента (vanishing gradient problem) особенно характерна для нейронных сетей в связи с их глубиной. Всё дело в том, что градиент функции потерь является произведением градиентов функций всех лоёв. А значит, если хотя бы на одном слое значение градиента оказывается очень малым, то это приводит к очень малому значению градиента всей сложной функции. А поскольку у нас нет строгой интерпретации, что происходит на каждом слое нейронной сети, то решить эту проблему, использовав аналоги подходов из классического машинного обучение, было бы сложно. В результате в данной концепции с высокой вероятностью обновление весов передних слоёв

будет практически нулевым, и найти оптимальное состояние не получится.

Стоит отметить, что проблема затухания градиента возникает не только при большом количестве слоёв нейронной сети. Подобная ситуация проявляется при использовании некоторых функций активации. Такими функциями являются сигмоида, гиперболический тангенс, чьи градиенты немалы лишь в очень узком диапазоне значений.

За долгое время развития глубового обучения было предложено множество способов решения проблемы затухания градиентов. К таковым можно отнести использование функции активации ReLU и применение нормализации слоёв. Но я хочу поговорить о подходе, связанном с гораздо более кардинальным изменением архитектуры нейронных сетей и возникновением того класса моделей, который используется в данной работе.

2.1.4 Архитектура ResNet

На данный момент уже существует огромное множество классов свёрточных нейронных сетей, связанных с особенностями их архитектуры. К таковым относятся, например, LeNet, AlexNet, VGG и ResNet. Поговорим подробнее про последнюю.

В качестве одного из примеров решения проблемы затухания градиента была предложена идея добавления в последовательность слоёв нейронной сети *обходных соединений* (skip connections). То есть выходные данные с одного слоя могут передаваться не только непосредственно на следующий, но и в обход него, сразу к одному из более глубоких слоёв. Развитие этой идеи привело к появлению такого понятия, как **Residual Block**, который представляет собой последовательность слоёв, в обход которой существует обходное соединение.

Применение skip connections действительно помогло решить проблему затухания градиента. Их добавление приводит к тому, что в формуле для градиента сложной функции возникают слагаемые, которые не зависят от градиентов некоторых слоёв. В результате произведение превращается в сумму нескольких произведений градиентов. Вероятность того, что все такие слагаемые будут затухающими, оказывается очень мала.

Данная концепция привела к появлению целого класса нейронный сетей **ResNet** [1]. Этот класс включает в себя модели разных размеров, то есть состоящих из разного количества слоёв. Благодаря использованию skip connections появилась возможность обучать нейронные сети огромной глубины, вплоть до нескольких сотен слоёв. Однако использование более крупных моделей может дать результаты, которые не существенно лучше тех, что показывают более простые.

Рассмотрим подробнее модель ResNet-18 (рис. 2.4). Данная архитектура состоит из 18 слоев. Входные данные передаются первому свёрточному слою с 64 ядрами 7x7, после чего применяется Max Pooling (взятие максимального элемента на подматрицах). Затем данные проходят 4 стадии из 4 свёрточных слоёв с ядрами 3x3. Каждая стадия разбита парами на два Residual блока. Начиная с 64 ядер на первой стадии, на каждой следующей число ядер увеличивается в 2 раза, вплоть до 512 на последней стадии. В финале данные проходят слой Average

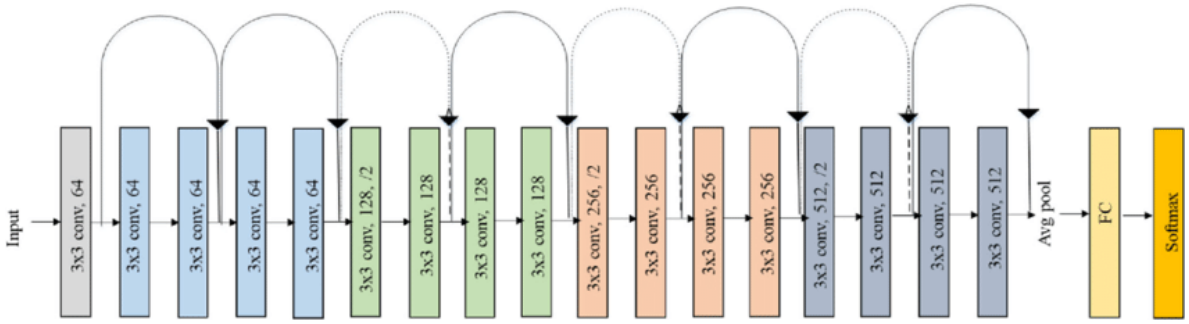

Рис. 2.4: Схема модели ResNet-18.

Pooling (взятие среднего арифметического всех элементов подматриц) и полносвязный слой с функцией активации Softmax.

Данная модель, несмотря на относительную простоту архитектуры, заявлена как один из лучших кандидатов для решения задачи классификации изображений. А её несомненным преимуществом является легковесность, позволяющая сократить затраты времени и объёмов памяти компьютера при её обучении.

2.2 Векторное представление изображений

2.2.1 Определение

Как мы уже знаем после знакомства с глубоким обучением, в последние годы одним из ключевых элементов многих методов машинного обучения и глубокого обучения стало **векторное представление** данных. Также этот термин имеет название **эмбеддинг**. Данный подход позволяет перевести задачу поиска закономерностей в наборах данных сложных типов в область более простых и понятных методов классического машинного обучения. Ведь если вместо текстов или изображений мы получаем числовые векторы, то к ним можно применять методы регрессии и классификации, поиск ближайших соседей и кластеризацию.

Изначально векторные представления возникли и были популярны в области обработки текста (например, word2vec). Но позднее аналогичный подход нашёл широкое применение и в других областях, включая компьютерное зрение. Ведь довольно очевидно, что если изображение представить в виде тензора (вообще говоря, 3-го ранга, чтобы учиывать цветовые слои), то можно некоторым отображением переве-

сти его в вектор произвольной размерности. Именно этим мы и занимаются нейронные сети в задачах классификации изображений.

2.2.2 Построение

Однако стоит понимать, что для качественного исследования эмбеддинги должны строиться не произвольным способом. Процесс отображения должен строиться так, чтобы в процессе не потерять различия между некоторыми свойствами разных изображений. Например, если на одной картинке содержится животное, а на другой – транспортное средство, то при построении эмбеддингов изображений ни на каком этапе даннные этих картинок не должны оказаться достаточно схожими. Тогда же финальные векторы-эмбеддинги для изображений с разными объектами не будут близки по значениям. Аналогичное условие нужно потребовать схожесть результатов построения эмбеддингов для изображений, на которых изображены похожие или совпадающие объекты.

Построение эмбеддингов можно произвести с помощью нейронных сетей. Ведь если мы вспомним, как устроен процесс преобразования данных внутри свёрточных нейронных сетей, то окажется, что именно этим они и занимаются. Для получения эмбеддинга нужной длины достаточно лишь заменить выходной слой сети таким образом, чтобы количество его нейронов совпадало с размерностью векторного пространства, с которым мы планируем дальше работать.

Таким образом, мы можем выделить следующие свойства эмбеддингов:

- *Компактность*. Позволяют уменьшить размер данных без потери важной информации, что делает работу с изображениями более эффективной.
- *Сравнимость*. Позволяют легко сравнивать изображения, вычисляя расстояние между их векторными представлениями по некоторым метрическим правилам.
- *Универсальность*. Их можно использовать в различных задачах, таких как классификация, кластеризация, поиск по изображениям и др.

- *Интерпретируемость*. Могут быть интерпретированы с точки зрения семантического сходства, что позволяет выявлять похожие изображения на основе их содержимого.

Стоит подробнее сказать про последнее свойство. Мы говорили, что эмбеддинги получаются на выходе нейронной сети. Однако было также обозначено, что нейронные сети не являются строго интепретируемыми моделями. Ведь за счёт количества параметров у человека нет возможности отследить, какая именно информация извлекается в процессе передачи данных через каждый слой. Однако если правильно настроить обучение модели, то после нахождения оптимального состояния нейронная сеть на каждом этапе будет выделять некоторую абстрактную информацию, которая хоть и не будет понятна разработчикам, но явно будет использоваться дальше. В результате, когда на выходе получется эмбеддинг, все свойства, которые извлекались при прохождении через модель, могут получить некоторую структуру, для которой будет возможность построить интерпретационное описание. Например, одни компоненты вектора будут отвечать за наличие на определённом участке окон, другие – людей, третьи – автомобилей, и т.д.

Размерность эмбеддингов может варьироваться в зависимости от используемой модели и задачи. Чаще всего размерность эмбеддингов варьируется от нескольких десятков до нескольких тысяч элементов. Например, модели ResNet-50 и ResNet-101 в своём обычном определении генерируют эмбеддинги размерностью 2048. Однако такая размерность больше подходит для задач классификации самых разных изображений. Если мы планируем использовать нейронную сеть для задачи классификации архитектурных объектов, то нам, очевидно, понадобится гораздо более короткие векторы.

Стоит отметить, что хоть высокая размерность и может позволить модели захватывать больше информации, она также может привести к переобучению модели и увеличению вычислительных затрат. С другой стороны, если исользовать слишком короткие векторы, то модель может попросту недоучиться из-за потери некоторой абстрактной информации. В задаче классификации изображений рекомендуется использовать эмбеддинги, длина которых примерно равна количеству классов.

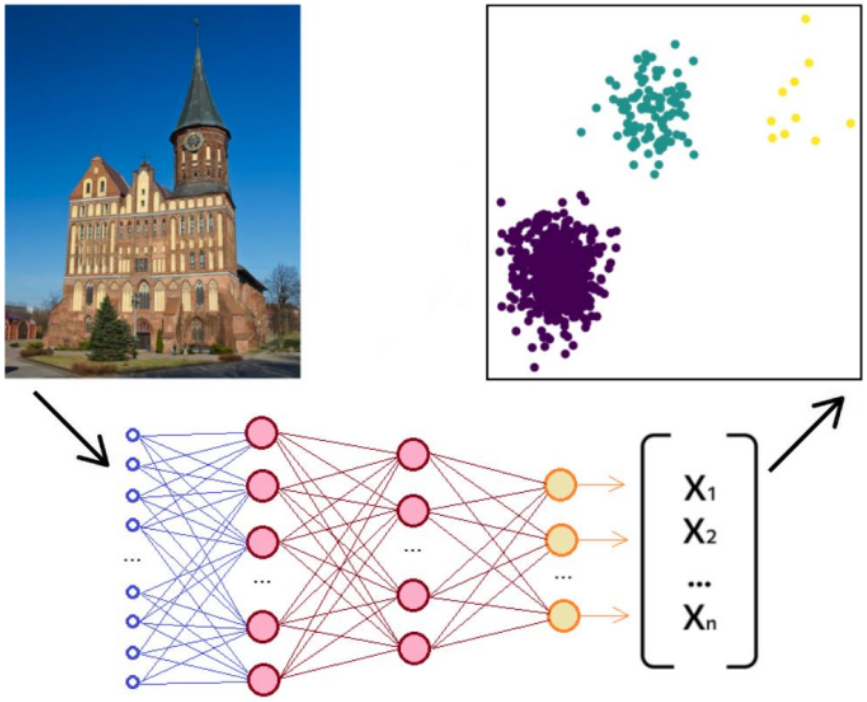

Рис. 2.5: Построение и расположение эмбеддингов в пространстве.

2.2.3 Интерпретация

Расположение эмбеддингов в пространстве, как было описано выше, отражает семантические отношения между объектами (рис. 2.5). Близкие друг к другу эмбеддинги векторно представляют изображения, которые имеют схожие характеристики или содержимое. К примеру, эмбеддинги изображений кошек и собак будут ближе друг к другу в векторном пространстве, чем эмбеддинги изображений кошек и автомобилей. Такое же свойство можно настроить и для изображений разных архитектурных объектов. Если хорошо подобрать веса нейронной сети, то она будет выдавать схожие эмбеддинги для изображений двух разных соборов, но сильно разные эмбеддинги для изображений собора и жилого дома.

Отдельной темой является определение так называемой *схожести* изображений. Подробнее об этом будет разговор дальше, в разделе про метрики. Однако уже сейчас можно сказать, что имеет смысл в дальнейшем анализе не смотреть на величину векторов. Гораздо более важной характеристикой является направление, которое этот вектор определяет. Дело в том, что если два разных эмбеддинга являются практически сонаправленными, но при этом из модули заметно отличаются,

то это говорит о том, что все их характеристики соотносятся пропорционально. К таковым характеристикам может относиться, например, контрастность или освещённость оригинального изображения. Но данные свойства не говорят о том, что на картинках были изображены разные объекты, а как раз наборот. Это показывает лишь различия форматов изображений, но не самих объектов.

Таким образом, если мы применим операцию нормировки для всех эмбеддингов, переместив их на многомерную единичную сферу, то мы с высокой вероятностью не потеряем той информации, которая определяет принадлежность изображения к произвольному классу.

2.3 Функция потерь

2.3.1 Определение

Модели глубокого обучения имеют аналогичную схему работы с классическими методами, включая использование так называемой **функции потерь** (loss function). Это центральный инструмент, используемый для оценивания *качества* работы модели машинного обучения.

Функция потерь, также известная как *функция стоимости* или *функция ошибки*, измеряет разницу между предсказанными значениями модели и фактическими значениями. Причём это измерение может производиться разными способами. Важно лишь понимать, что данная функция начисляет штрафы для модели, если она выдаёт неправильные или недостаточно точные предсказания. И чем больше разница между ответом модели и реальным значением, тем не меньше будет начисляться штраф.

Таким образом, функция потерь, как правило, представляется в виде неотрицательной суммы значений некоторой функции, которая принимает на вход вектор предсказаний и вектор правильных ответов. Для функции потерь можно попробовать вычислить градиент, чтобы иметь возможность подкорректировать параметры модели некоторым оптимизационным методом.

Примерами функций потерь в классическом машинном обучении

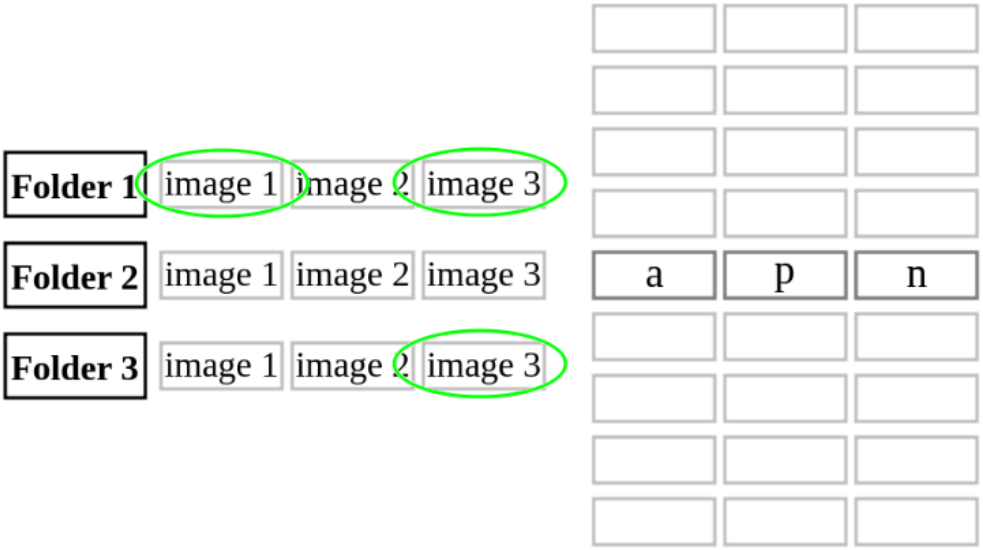

Рис. 2.6: Построение троек для Triplet Loss.

являются *средняя квадратичная ошибка* (MSE), *средняя абсолютная ошибка* (MAE), *кросс-энтропия* (cross-entropy). Эти функции работают именно по той схеме, что была описана выше, то есть принимают векторы предсказанных ответов. Однако для работы с нейронной сетью, которая выдаёт на выход векторы-эмбеддинги, эти функции не подходят. Требуется метод, который будет некоторым образом штрафовать модель за недостаточно хорошо вычисляемые эмбеддинги.

2.3.2 Triplet Loss

Triplet Loss (или Triplet Margin Loss) [2, 3] является функцией потерь, которая используется для обучения моделей, направленных на векторное представление данных, включая кластеризацию изображений. Основным принципом её работы является сравнение расстояний между векторами в пространстве эмбеддингов.

Triplet Loss работает с *тройками* изображений (рис. 2.6). Каждая тройка состоит из так называемых *якорного* (anchor), *положительного* (positive) и *отрицательного* (negative) изображений. Причём эти тройки выбираются таким образом, что изображения anchor и positive относятся к одному классу, а изображение negative – к другому. Цель состоит в том, чтобы минимизировать расстояние между anchor и positive и максимизировать расстояние между anchor и negative.

Формула Triplet Loss выглядит следующим образом:

$$TripletLoss(\alpha; A, P, N) = \sum max(0, Dist(a, p) - Dist(a, n) + \alpha).$$

Здесь $Dist(a, p)$ – некоторая *метрическая функция*, измеряющая расстояние между векторами a и p, α – некоторый положительный *сдвиг* (margin).

Мы видим, что функция действительно получается неотрицательная, за счёт использования подхода ReLU с максимумом. Также видно, что если эмбеддинг отрицательного изображения окажется ближе к якорю, чем эмбеддинг положительного, то функция получает штраф. Но в добавок в формуле используется margin, который добавляет штрафы для тех случаев, когда расстояние между anchor и positive незначительно меньше, чем расстояние между anchor и negative.

Данный подход провоцирует нейронную сеть двигаться к тому состоянию, в котором эмбеддинги одного класса будут оказываться рядом, а эмбеддинги разных классов – далеко друг от друга. В результате мы можем получить хорошо кластеризованный набор векторов для изображений из обучающей выборки.

2.4 Метрика

2.4.1 Определение

Понятие **метрики** пришло в машинное обучение из математики. Там под этим термином понималась произвольная неотрицательная функция двух, вообще говоря, векторных переменных, которая удовлетворяет аксиомам тождества, положительности и симметричности, а также неравенству треугольника. В контексте машинного обучения и метрика определяется точно так же.

Метрики используются для определения *схожести* векторов, которые являются числовыми представлениями данных. Сравнивая векторы, можно определить, насколько объекты, которые они представляют, похожи или отличаются друг от друга. При этом важно выбирать пра-

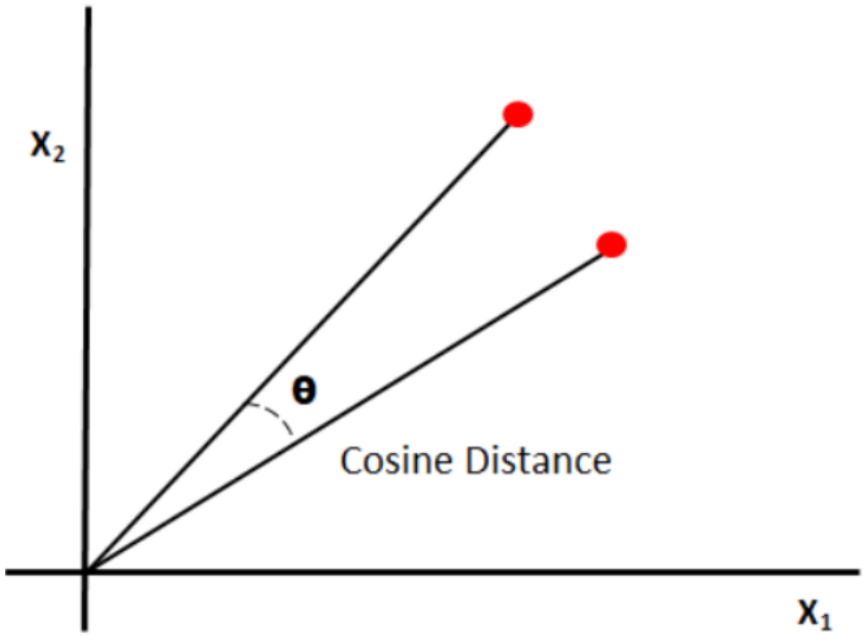

Рис. 2.7: Метрика косинусного расстояния.

вильную метрику в зависимости от задачи и особенностей данных.

В машинном обучении используется множество метрик для оценки схожести векторов. Таковыми являются, например, *евклидово расстояние*, *манхэттенское расстояние*, *расстояние Чебышёва*, *косинусное расстояние*, *расстояние Жаккара*.

2.4.2 Косинусное расстояние

Косинусное расстояние (cosine similarity) является метрикой, которая оценивает угловую разницу между двумя векторами (рис. 2.7). Это особенно полезно, когда важна не величина векторов, а направление. Косинусное расстояние используется в задачах, где важно оценить ориентацию векторов в пространстве.

Косинусное расстояние определяется как единица минус косинус угла между двумя векторами:

$$CosineSimilarity(x, y) = 1 - \frac{\langle x, y \rangle}{||x|| \cdot ||y||},$$

где $\langle x, y \rangle$ – скалярное произведение векторов.

Косинусное расстояние широко используется в задачах обработки текстов и изображений, где важно сравнение векторов высоких размерностей. В таких задачах векторы часто нормализованы, что делает косинусное расстояние особенно полезным.

Глава 3

Разработка решения

3.1 Алгоритм

Описанный ниже алгоритм проходил тестирование только для чёрно-белых фотографий предвоенного Кёнигсберга. Работа включала в себя несколько этапов.

1. **Подготовка данных**.
   В первую очередь, файлы с изображениями должны быть правильно распределены по папкам, соответствующих классам. Для более качественной работы классы, которым соответствует небольшое число картинок, были вынесены в отдельную папку и в работе не использовались. После этого, как в любой задаче машинного обучения, данные необходимо разделить на выборки: обучающую и тестовую. Разделение производилось в соотношении 80:20 для каждой папки, чтобы сохранить однородность данных.

2. **Предобработка изображений**.
   Для работы нейронной сети изображения должны пройти через последовательность преобразований. Для начала при чтении из файла изображение *конвертируется* в тензорный формат. Затем, поскольку размер входного слоя нейронной сети строго детерминирован (3x224x224), тензор должен быть сжат или растянут, чтобы иметь такой же размер. Чтобы сохранить пропорции сторон оригинальной картинки, но при этом получить квадратный тензор, было применено преобразование из двух шагов. Сперва тензор пропроционально *сжимался/растягивался*, после

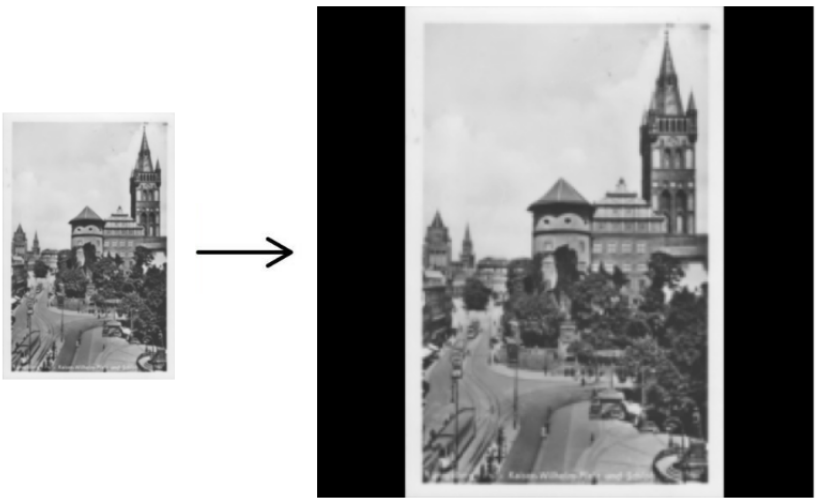

Рис. 3.1: Преобразование изображения.

чего к нему добавлялись две *чёрные полосы*: сверху и снизу (если картинка широкая) или по бокам (если картинка высокая) (рис. 3.1). Такое преобразование позволяет сохранить правильный наклон линий. В финале тензор *утраивается*, поскольку в используемой модели вход состоит из трёх каналов.

3. **Определение модели**.
   На данном этапе производится загрузка нейронной сети ResNet-18. В работе использовался подход, связанный с **дообучением модели** (fine tuning). Он подразумевает собой использование на первой итерации обучения готового набора весов, который был уже получен в результате обучения нейронной сети на большом датасете изображений **ImageNet**. Этот датасет включает в себя фотографии самых разных объектов, включая архитектурные. Для сокращения затрат времени и памяти, дополнительно применяется **заморозка слоёв** нейронной сети. Это означает, что веса на некоторых слоях фиксируются, и по ним не будет считаться градиент. Имеет смысл применить заморозку к большей части слоёв, оставив активными только несколько последних. Такое решение может оказаться эффективным, поскольку на первых слоях нейронной сети извлекается некоторая общая информация об изображениях, а уже на последних могут подмечаться признаки, характерные для архитектурных объектов. Напоследок не стоит забыть заменить последний слой. На выходе нужно получать векторы той же размерности, что и число классов.

4. **Определение оптимизатора и функции потерь**.
   Для обновления параметров модели используются методы оптимизации. Как правило, они основаны на градиентном спуске, то есть движении по направлению наискорейшего убывания функции. Универсальным оптимизационным методом в работе с нейронными сетями является **Adam** [4, 5], который использует сложные правила обновления *темпа обучения*, основанные на моментах градиентов. После синхронизации Adam с весами модели определяется функция потерь. В работе используется описанная выше функция Triplet Loss, способная достаточно эффективно разделить эмбеддинги обучающей выборки. В качестве метрики, которую используется функция потерь, выбирается косинусное расстояние, которое лучше всего показывает схожесть направлений векторов, расположенных на многомерной единичной сфере.

5. **Обучение модели**.
   Необходимо провести несколько циклов обучения, предварительно разбив датасет на *батчи*. На данном этапе модель по очереди получает батчи тензоров и считает по ним эмбеддинги. Затем полученные эрезультаты вместе с вектором ответов проходят через функцию потерь, на выходе которой считаются суммарные штрафы за тройки, сформированные из векторов данного батча. После этого считаются градиенты на каждом предыдущем слое по схеме backpropagation и производится обновление весов модели по оптимизационному методу.

6. **Сохранение эмбеддингов обучающей выборки**.
   После завершения стадии обучения необходимо вычислить и сохрнаить эмбеддинги для всех изображений обучающей выборки. Во-первых, это позволяет в дальнейшей работе не импортировать модель и не считать векторы заново. Во-вторых, это пригодится для поиска ближайших соседей на стадии тестирования.

7. **Тестирование модели**.
   Данный этап подразумевает задачу классификации эмбеддингов при помощи стандартного метода **k ближайших соседей** (kNN). Для этого импортируется специальная модель, способная быстро определять для любого вектора в пространстве, какие векторы обучающей выборки находятся к нему ближе всего. После этого методом простого голосования среди найденных ближайших соседей определяется самый популярный класс, который и выбирается в качестве ответа. В финале смотрится, какова доля совпадений между правильным ответом и ответом нашего алгоритма.

3.2 Результаты

В ходе работы над проектом были выполнены следующие действия:

- Использованы предложенные методы предобработки изображений.
- Импортирована нейронная сеть ResNet-18, предобученная на датасете ImageNet.
- Применена заморозка 13 из 18 слоёв модели.
- Подключена функция потерь Triplet Loss с косинусной метрикой схожести векторов.
- Проведено обучение модели на изображениях предвоенного Кёнигсберга.
- Проведено тестирование модели при помощи метода kNN.

Результаты работы продемонстрировали, что 85% изображений тестовой выборки оказались правильно классифицированы.

Глава 4

Планы на развитие

Работа, проделанная в рамках данного проекта, показала отличные результаты буквально после первого же эксперимента. Это говорит о том, что решение таких больших задач не требует огромных вычислительных затрат и применения слишком сложных подходов. Для этого достаточно лишь правильно построить алгоритм, используя самые подходящие инструменты.

Есть подозрения, что мы бы не получили таких впечатляющих рещультатов, если бы не упростили постановку задачи. Ведь нейронная сеть обучалась хоть и на внушительном, но всё же сильно ограниченном наборе фотографий. Среди изображений были только чёрно-белые картинки эпохи предвоенного Кёнигсберга. Также были целиком проигнорированы некоторые архитектурные объекты, которые запечатлены на слишком малом количестве фотографий, чтобы избежать недообучения модели под данные классы. Однако в случае дальнейшего роста проекта нужно уметь обрабатывать все такие случаи, не тратя большое количество времени на придумывание решения.

В данной главе хочу поделиться некоторыми идеями, которые могут развить проект и повысить его функциональность.

4.1 Расширение архива

4.1.1 Добавление эпох

Как уже было показано на примере Калининграда, у города можно выделить три больших исторических периода, на которые удобно разделить фотографии: немецкий, советский и современный. Имеется

возможность провести эксперимент, используя не только изображения немецкого периода, но и сразу весь архив по городу.

Главным преимуществом данного метода является рост информативности всего сервиса, поскольку клиенты смогут получать гораздо больше различных сведений.

В качестве основной преграды стоит отметить наличие архитектурных объектов, которые не сохранились до наших дней. Это требует применения новых методов разделения фотографий на классы и значительному росту размеров задачи, что в свою очередь потребует использования более сложных моделей. Помимо этого, если сервис планируется для расширения на другие регионы, то могут возникнуть затруднения, связанные с разделением фотографий новых городов на исторические эпохи.

4.1.2 Добавление новых классов

На начальном этапе проектный архив содержал изображения всего нескольких десятков классов, что несомненно мало. Для лучшего роста информативности требуется регулярно принимать в архив новые изображения. Это можно сделать например за счёт тех картинок, которые загружают пользователи в приложение.

Предложенный подход содержит очевидный недостаток. Присланные фотографии могут не принадлежать ни одному из известных нейронной сети классов. То есть нейронная сеть попросту не будет способна распознать архитектурный объект на изображении, и алгоритм гарантированно выдаст неправильный ответ.

Для решения данной проблемы есть возможность введения дополнительного критерия, который будет чётко определять, что на присланной фотографии содержится известный объект. В противном же случае можно отложить картинку в отдельную папку, чтобы потом провести дополнительное исследование. В результате из большого набора отложенных фотографий иногда будет возможность выделить новый класс, который можно передать на дообучение нейронной сети.

4.2 Вариация инструментов

В данном разделе предусматривается рассмотрение альтернативных вариантов построения решающего алгоритма. Очевидным примером является замена нейронной сети, что было бы актуально с ростом размеров задачи. Также имеет смысл провести поиск оптимальных значений множества гиперпараметров, которые определяют функции инструментов. Такими параметрами являются, например, margin в функции потерь Triplet Loss, размер передаваемых при обучении нейронной сети батчей, количество соседей в методе kNN или длина выходного слоя модели. На самом деле количество гиперпараметров в данной задаче настолько большое, что для поиска их оптимальных значений потребовались бы сотни экспериментов с обучением нейонной сети заново.

Главный критерий, которого я буду руководствоваться при выборе окончательного алгоритма решения – соотношение «цена-качество». Хочется найти золотую середину, в которой достигается баланс между качеством работы алгоритма и ценой, которая платится за его обучение и тестирование. То есть, если метод был очень эффективен с точки зрения затрат памяти и времени, но при этом показывает не очень точные результаты, то он не будет в приоритете. Аналогичное могу сказать и про алгоритм, который занимает большое количество ресурсов на обучение, хоть и выдающий после этого безупречные результаты.

4.3 Внедрение дополнительных функций

Проект приложения, основанный на работе с архивом исторических фотографий, может получить развитие не только с точки зрения собственной структуры, но и в качестве расширения на другие задачи. Иными словами, использование алгоритма по распознаванию объектов на изображении может применяться не только для обеспечения пользователей простой исторической информацией, но и для нечто большего.

Существуют варианты, как использовать данный метод в задаче воссоздания локаций в формате 3D-моделирования или виртуальной реальности. Аналогичные подходы уже применяются, например, в сфере выбора недвижимости, где клиентам предлагается посмотреть объёмные модели разных вариантов постоянного жилья или отелей в дру-

гих городах, находясь в любой точке планеты. Подобным образом у нашего проекта есть потенциал расшириться до многофункционального сервиса, способного предоставить любому человеку возможность путешествовать по разным эпохам и локациям.

4.4 Завершение

Напоследок хочется сказать, что проект планируется к дальнейшему продвижению в формате стартапа. Уже были предприняты попытки пройти с данной идеей в несколько акселлераторов, некоторые из которых были одобрены. Также команда уже наладила контакты с некоторыми инвесторами, которые проявили заинтересованность в развитии сервиса, предложив несколько вариантов повышения его стоимости.

Литература

[1] He, K., Zhang, X., Ren, S., Sun, J. Deep residual learning for image recognition //Microsoft Research – 2016. – С. 3-4.

[2] Schroff, F., Kalenichenko, D., Philbin, J. FaceNet: A unified embedding for face recognition and clustering //Google Inc. – 2015. – С. 3-4.

[3] Hoffer, E., Ailon, N. Deep metric learning using Triplet network. In International Workshop on Similarity-Based Pattern Recognition //Technion Israel Institute of Technology – 2015. – С. 1-3.

[4] Kingma, D. P., Ba, J. Adam: A method for stochastic optimization //University of Amsterdam, OpenAI. University of Toronto – 2014. – С. 1-3.

[5] Ruder, S. An overview of gradient descent optimization algorithms //Insight Centre for Data Analytics, NUI Galway Aylien Ltd., Dublin – 2016. – С. 8.