МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(ФГБОУ ВО «КубГУ»)

Факультет компьютерных технологий и прикладной математики
Кафедра анализа данных и искусственного интеллекта

Допустить к защите
заведующий кафедрой
д-р тех. наук, доцент
_____ А.В. Коваленко
_____2024 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
(БАКАЛАВРСКАЯ РАБОТА)

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РАСПОЗНАВАНИЯ 2D ОБЪЕКТОВ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ

Работу выполнил(а) _____ К.И. Варламов

Направление подготовки 09.03.03 Прикладная информатика

Направленность (профиль) Прикладная информатика в экономике

Научный руководитель
канд. пед. наук, доцент _____ В.А. Акиньшина

Нормоконтролер
канд. физ.-мат. наук, доцент_____ Г.В. Калайдина
(подпись)

Краснодар
2024

СОДЕРЖАНИЕ

РЕФЕРАТ

Выпускная квалификационная работа 36 с., 11 рис., 6 источн., 1 прил.

РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РАСПОЗНАВАНИЯ 2D ОБЪЕКТОВ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ

В работе исследуются сверточные нейронные сети (далее – англ. convolutional neural network, CNN), предмет исследования – их применение в обработке изображений.

Целью работы является написание приложения для распознавания различных природных объектов и памятников архитектуры.

Цель работы определяет следующие задачи:

– Изучить структуру сверточных нейронных сетей и способы их построения;

– Собрать и подготовить исходные данные;

– Построить нейронную сеть для распознавания известных объектов архитектуры и памятников природы, используя язык программирования Python;

– Написать десктопное приложение для распознавания достопримечательностей;

– Протестировать написанное приложение.

ВВЕДЕНИЕ

В современном мире ежедневно генерируется и накапливается огромное количество данных, в том числе визуальных. Изображения являются одним из основных источников информации, однако извлекать ценные сведения из них вручную не представляется возможным. Для решения этой задачи требуется эффективный метод распознавания объектов в изображениях.

Сверточные нейронные сети (Convolutional Neural Networks, CNN) показали высокую эффективность в задачах распознавания объектов. Они способны извлекать иерархические признаки из изображений с учетом их пространственной структуры, что обеспечивает точность распознавания. Поэтому CNN стали ключевым инструментом в области компьютерного зрения и широко применяются во многих сферах жизни современного человека.

Разработка приложения для распознавания 2D объектов с использованием сверточных нейронных сетей является актуальной и перспективной задачей. Такое приложение позволит автоматизировать процесс распознавания объектов на изображениях, сделав его более точным и эффективным. Практическая значимость подобного решения трудно переоценить, поскольку оно найдет применение во множестве областей.

1 Нейронные сети и с чем их едят

1.1 Основные понятия и определения

Нейронные сети – это виртуальные математические модели, вдохновленные биологической организацией нейронов в мозге живых организмов. Они используются для обработки информации и выполнения различных задач машинного обучения.

Архитектура нейронных сетей включает множество связанных искусственных нейронов, организованных в слои, такие как входной слой, скрытые слои и выходной слой. Каждый нейрон обладает своими весами (параметрами) и функцией активации.

Эти модели способствуют разработке компьютерных систем, способных принимать обоснованные решения с ограниченным участием человека. Они моделируют функции и деятельность нейронов в биологическом мозге.

Нейронные сети включают в себя множество типов, каждый из которых предназначен для решения определенных задач.

Многослойные перцептроны – наиболее простые нейронные сети, которые используются для классификации изображений, распознавания речи и других задач.

Сверточные нейронные сети – такие нейронные сети, которые специализированы на обработке изображений и видео. Используя фильтры для выделения локальных признаков, они достигли значительных успехов в области компьютерного зрения. Именно их мы будем использовать в дальнейшей разработке приложения.

Еще один тип нейронных сетей – рекуррентные нейронные сети. Они работают с последовательными данными, такими как текст и речь, запоминая информацию из прошлого для прогнозирования будущего. Они применяются в машинном переводе и генерации текста.

Кроме этих основных типов, существуют и другие виды нейронных сетей:

– Самоорганизующиеся карты Кохонена;

– Генеративно-состязательные сети;

– Нейросети прямого распространения;

– Сети с радиальными базисными функциями;

– Нейросети с экстремальным обучением машин.

Каждый из этих видов разработан для решения специализированных задач и имеет свои уникальные особенности и области применения.

Нейронные сети широко применяются в различных отраслях, таких как:

– Использование классификации медицинских изображений для диагностики заболеваний;

– Фильтрация социальных сетей и анализ поведенческих данных для третированного маркетинга;

– экономические прогнозы на основе временных рядов;

– прогнозирование нагрузки электростанций в больших городах и энергопотребления;

– контроль соответствия стандартам и обеспечение качества на производстве;

– определение наилучших химических соединений лекарственных препаратов.

Стоит отметить, что нейронные сети решают четыре основные задачи, каждая из которых играет важную роль в различных областях.

Первая задача – это машинное зрение. Машинное зрение позволяет компьютерам извлекать информацию из изображений или видео, распознавая и классифицируя их, подобно тому, как это делают люди. Машинное зрение имеет широкое применение в различных предметных областях:

– В беспилотных автомобилях оно используется для визуального распознавания дорожных знаков и препятствий на пути движения автомобиля;

– Автоматическое удаление небезопасного или нежелательного контента при модерации архивов изображений и видео;

– Машинное обучение также используется для распознавания различных изображений, таких как: растения, животные, человек, для идентификации сотрудников в некоторых организациях, помимо этого, можно детально изучать мимику лиц людей, определять цвет глаз, присутствие или отсутствие очков, наличие шляпы и других элементов одежды;

– Другой его задачей является распознавание звуков. Нейросетевые модели достаточно эффективно анализируют человеческую речь независимо от языка и тембра голоса. Сегодня виртуальные помощники, например, Яндекс.Станция или Маруся, могут использовать распознавание голоса человека для выполнения ряда определенных команд и задач.

– Помимо этого, распознавание речи используется для автоматической классификация звонков, преобразования клинических рекомендаций в поликлиниках в документацию режиме реального времени.

– Достаточно трудной является обработка естественного языка. Произведения, созданные человеком, могут быть написаны на разных языках, в связи с этим трудно выделить основную мысль и смысл текста с помощью машинного обучения. Наиболее часто используемой и выполнимой задачей для обработки языка сегодня являются автоматизированные виртуальные агенты и чат-боты, они отвечают на примитивные запросы пользователя, классифицируют введенные человеком данные, осуществляют анализ статей по выбранной пользователем тематике, генерируют тексты и статьи по указанной теме.

Кроме того, с помощью нейронных сетей можно анализировать действия пользователей для формирования персонализированных рекомендаций. Они обнаруживают новые продукты или услуги, которые могут заинтересовать конкретного пользователя.

Например, стартап Curalate из Филадельфии помогает брендам превращать социальные медиа-сообщения в продажи. Curalate использует нейронные сети для автоматического поиска и рекомендации продуктов на основе активности пользователя в социальных сетях. Потребителям не нужно будет искать конкретный продукт или товар в онлайн-каталогах, основываясь на личных изображениях или из социальных сетей. Вместо этого они могут легко приобрести продукт с помощью автоматической маркировки Curalate.

1.2 Как работают нейронные сети?

Для понимания принципов работы нейронных сетей необходимо рассмотреть их основные составляющие и параметры.

Нейрон — это вычислительная единица, которая принимает информацию, выполняет простые вычисления и передает результат дальше. Нейроны делятся на входные, скрытые и выходные. Когда нейронная сеть состоит из множества нейронов, их объединяют в слои. Входной слой получает информацию, n скрытых слоев (обычно не более трех), обрабатывают ее, а выходной слой выводит результат (Рисунок 1).

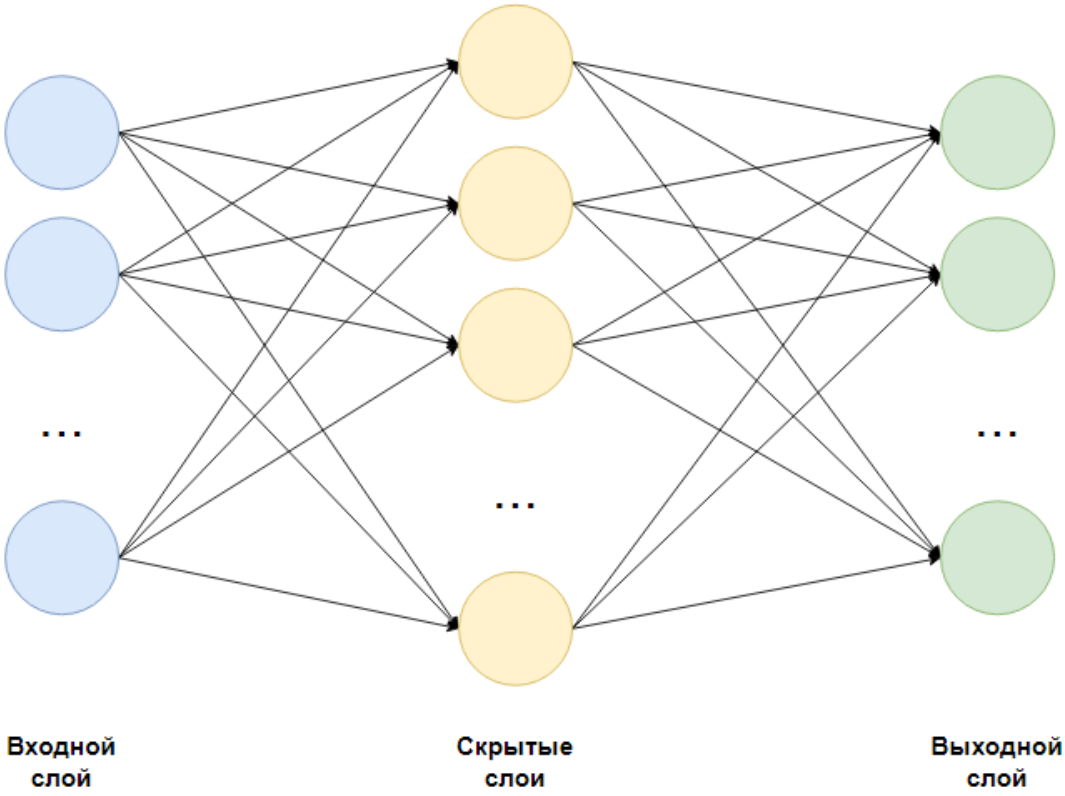

Рисунок 1 – Структура нейронной сети (Нейроны и слои)

Каждый нейрон имеет два главных параметра: входные (input) и выходные (output) данные (Рисунок 2). В случае входного нейрона, входные данные равны выходным: input = output. В других нейронах входные данные представляют собой сумму данных от всех нейронов предыдущего слоя. Эти данные нормализуются с помощью функции активации и записываются в выходное поле.

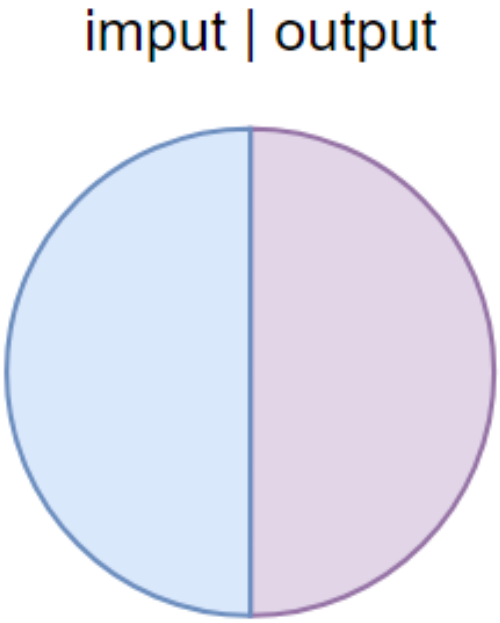

Рисунок 2 – Параметры нейрона

Важно отметить, что нейроны работают с числами в диапазоне [0,1] или [-1,1]. Вопрос, как обрабатывать числа за пределами этого диапазона, решается с помощью нормализации, которая достигается делением 1 на это число. Нормализация широко используется в нейронных сетях.

Для связи двух нейронов используется синапс — соединение между ними, которое имеет один параметр — вес. Вес изменяет входную информацию при передаче от одного нейрона к другому. Если три нейрона передают информацию следующему, то тогда у нас будет три веса, по одному для каждого нейрона. Нейрон с более высоким весом окажет уже большее влияние на следующий, и тогда его информация будет доминировать (например, при смешении цветов, Рисунок 3).

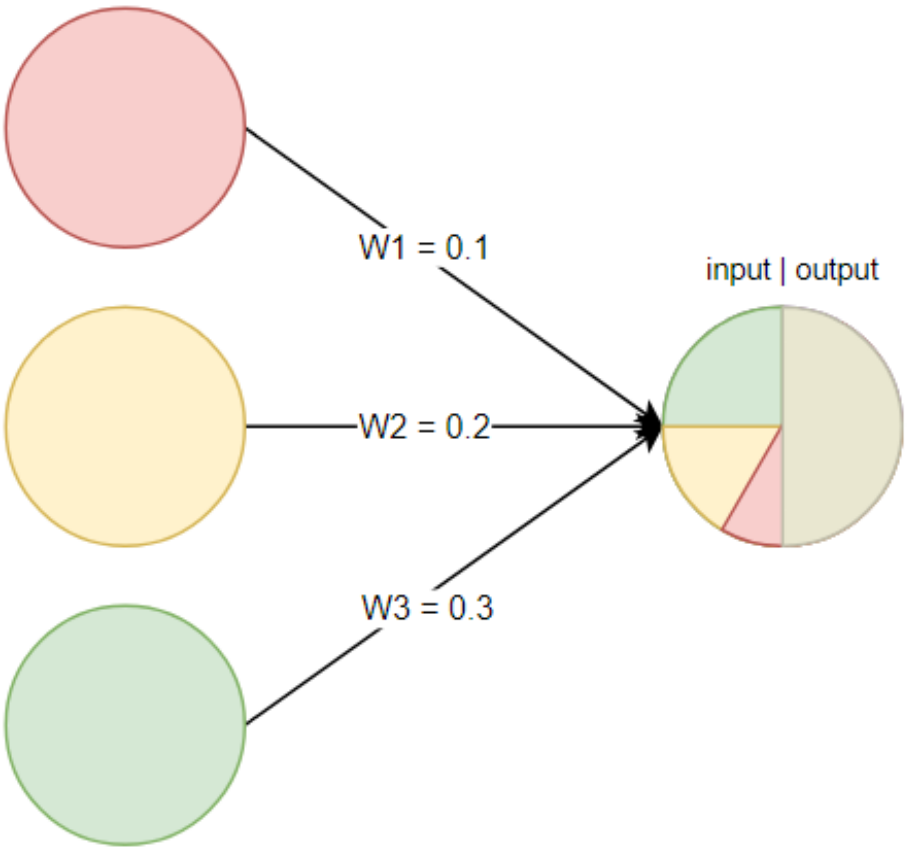

Рисунок 3 – Пример смешения цветов (синапс)

Совокупность матрицы весов, играет ключевую роль в ее работе, выступая в роли «мозга» системы. Именно благодаря ним поступающая информация обрабатывается и преобразуется в конечный результат. Стоит отметить, что при инициализации нейронной сети веса задаются случайным образом.

Рассмотрим несложную модель нейронной сети, состоящую из трёх слоев (см. Рисунок 3). Первый входной слой состоит из нескольких нейронов, на представленном рисунки их три. Далее задаются весовые коэффициенты для каждого нейрона и с помощью функции активации передаются в промежуточный скрытый слой, количество промежуточных слоев зависит от модели сети. Основная обработка информации, полученная с входного слоя нейронов, обрабатывается в промежуточных слоях и посредством функции активации передается в выходные результирующие нейроны, на рисунке 3

представлен один выходной нейрон. Информация, представленная на выходе модели может быть представлена в различных форматах.

1.3 Проблемы использования нейронных сетей

Развитие нейросетевых технологий происходит стремительно и с увеличением количества решаемых ими задач возникает ряд ограничений и проблем, требующих реализации. Основной проблемой является объем поступающих объектов данных, используемых для обучения. С одной стороны их малое количество приводит к недостаточному обучению модели, так как свойства используемых объектов не всегда будут описывать генеральную совокупность. Сегодня существуют специальные техники для работы с небольшими выборками. Это трансферное обучение и генеративные модели.

С другой стороны, большие объемы данных, особенно графических, требуют мощных вычислительных ресурсов, необходимых для обучения сети. Поэтому необходимо использование высокопроизводительных компьютеров для обработки графических объектов. Решением этой проблемы является использование облачных сервисов Яндекс.Облако или Mail.ru.Cloud.

Одной из основных и значимых проблем при использовании искусственных нейронных сетей является то, что несмотря на достаточно глубокий анализ больших числовых данных, они совершенно не улавливают суть текстов.

Нейронные сети достаточно чувствительны к выбросам. Поэтому важно правильно провести сбор данных и первичную их обработку, удаление повторяющихся данных и выбросов с помощью их сжатия и фильтрации.

Помимо этого, можно говорить об уязвимости нейронных сетей к атакам, таким как внедрение шума или изменение получаемых параметров. Дополнительные слои обнаружения аномалий помогают решть данную проблему.

2 Сверточные нейронные сети

2.1 Общие понятия

Для распознавания графических и видеоизображений служат свёрточные нейронные сети. Это одно из значимых достижений в компьютерном зрении.

Основным преимуществом свёрточных нейронных сетей, которое позволяет им эффективно классифицировать объекты - это использование промежуточных слоев свёртки. Чем больше таких слоев и вернее подобраны функции активации, то есть сложнее их архитектура, тем лучше качество распознаваемых объектов.

Как правило, при выполнении свёртки чаще всего используются нелинейные функции активаи. У изначальных образцов подвыборки уменьшают разрешение изображений и выделяют в них наиболее значимые признаки. Полносвязные нейронные сети объединяют эти черты для выполнения классификации объектов. Функции активации являются нелинейными, что помогает при обучении на изображениях с большим количеством мелких деталей.

Именно благодаря своей способности автоматически извлекать свойства из входных данных, свёрточные нейросети демонстрируют высокую производительность в задачах распознавания классификации изображений. Обучение сверточных нейронных сетей для распознавания объектов и/или их классификации, как правило, происходит на больших наборах данных, которые чаще всего делятся на тестовую и обучающую выборки, взятые 80% к 20% соответственно. . Одной из наиболее известных сверточных нейронных сетей, разработанной в 2012, являлась AlexNet. Она продемонстрировала выдающиеся результаты на конкурсе LSVRC по классификации изображений.

Сегодня отечественные сервисы, такие как Yandex DataSphere, предоставляют доступ к готовым средам разработки и облачным ресурсам.

2.2 Структура и принцип работы CNN

Сверточные нейронные сети работают на основе принципов, аналогичных стандартным, но с отличием: они применяют операцию свертки наряду с обычным матричным умножением. Она является линейной, используемой для двух функций с вещественными аргументами, итогом которой является третья функция, которая уже показывает степень сходства одной с отраженной и сдвинутой копией другой.

Одним из уникальных свойств CNN является их разреженная связность. В отличие от простых многослойных персептронов они используют небольшую матрицу весов. Она, известная как ядро свертки, перемещается по слою на каждом шаге обработки. Их может быть несколько, каждое кодирует наличие определенных отличительных свойств на изображении, таких как горизонтальные и вертикальные линии, дуги и сложные формы (например, эллипсы, треугольники, квадраты и т.д.).

Результатом операции с каждым ядром является карта признаков (КП), которая представляет собой массив матриц, указывающих на наличие определенных черт в обрабатываемом слое. Каждая из них содержит фильтр и при переборе каждого набора весов создается уникальная КП. Таким образом, нейронная сеть становится многоканальной.

В процессе свертки окно ядра сканируется с фиксированным шагом по всей области изображения. Оно умножается на ядро, итоги суммируются и сохраняются в матрице результатов, которая становится следующей картой.

Для уменьшения масштаба КП используются операции подвыборки. Наиболее часто используемой является MaxPooling, которая выбирает наибольшие значения из нескольких соседних нейронов и записывает их как один в новую, меньшую по размеру карту признаков. Это значительно сокращает объем памяти и ускоряет последующие вычисления.

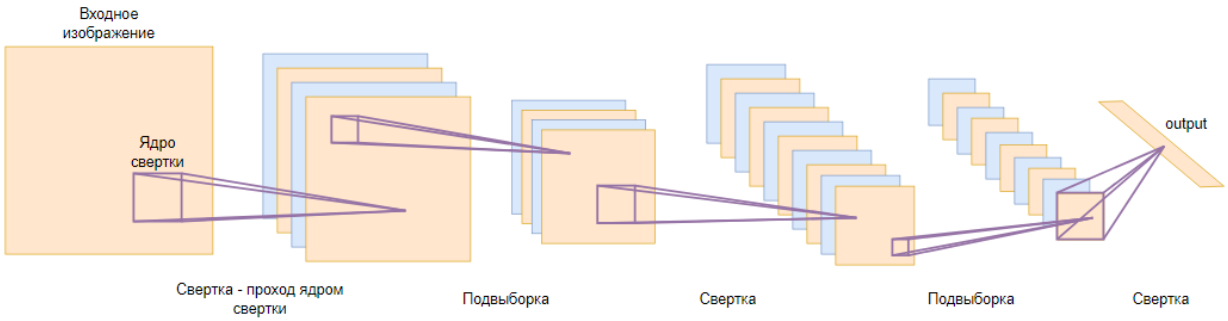

Рисунок 4 – Схема работы сверточной нейронной сети

Алгоритм обработки сигнала в нейросети последовательно пропускает данные через несколько слоев. На схеме (Рисунок 4) показаны входной, четыре скрытых и выходной. В каждом выполняются операции свертки и подвыборки для создания КП. По мере продвижения по ним количество карт увеличивается, а их разрешение уменьшается.

Результатом этих реализаций является набор каналов с уменьшенными данными, отражающими особенности исходной картинки. Эти сведения передаются в полносвязную НС, которая формирует конечный выходной сигнал. Таким образом, изображение проходит через несколько фильтров для извлечения ключевых свойств.

Параметры ядра оптимизируются с помощью метода обратного распространения ошибки, используемого для обучения. Сверточные операции выполняются параллельно для каждой карты признаков, что ускоряет работу сети.

Количество карт признаков выбирается в зависимости от требований задачи. Увеличение числа карт улучшает качество распознавания, но увеличивает вычислительную сложность. Обычно применяется соотношение 1:2, при котором на каждую карту предыдущего слоя приходится две нового.

2.3 Использование CNN

Одной из основных задач является распознавание и классификация информации на изображениях. Эти сети могут обнаружить объекты и разделить картинки на группы.

Так, например, автоматизированные системы вождения активно используют CNN для выявления дорожных знаков, пешеходов, автомобилей и препятствий, чтобы повысить безопасность и эффективность вождения. В медицине же они используются для анализа рентгеновских снимков, МРТ и КТ. Это помогает обнаруживать и классифицировать заболевания и повышает точность диагностики.

Еще одно важное применение – распознавание рукописного текста. Это полезно для систем оцифровки и ввода данных.

CNN также используются в исследовании видеоданных для классификации действий на видео. Они могут обнаруживать и отслеживать движущиеся объекты, что важно для видеонаблюдения.

В робототехнике CNN помогают роботам анализировать информацию с камер и датчиков, что позволяет им распознавать объекты и эффективно взаимодействовать с окружающей средой.

3 Обучение CNN

3.1 Что требуется для обучения?

Изначально нейронная сеть не настраивается. Процесс тренировки заключается в последовательной подаче изображений из набора на вход и сравнении полученных ответов с желаемыми. Например, при классификации лиц и фонов желаемый вывод устанавливается равным 1 для лиц и -1 для фонов. Разница между ожидаемым и фактическим откликом называется дельта ошибкой, которая уже передается всем связанным нейронам.

Основная задача заключается в минимизации функции ошибки путем изменения весов связей между узлами. Функция ошибки – это разница между полученным и желаемым ответами. Например, если на вход подается изображение и выход сети составляет 0,73, а предполагаемый результат – 1, то функция ошибки равна 0,27. Для узлов выходного слоя настройка относительно проста, поскольку известны как фактические, так и желаемые значения. Однако настройка весов предыдущего более сложна. Долгое время не существовало алгоритма, позволяющего распространять ошибку по скрытым слоям.

3.2 Алгоритм обратного распространения ошибки (ОРО)

Данный процесс минимизирует ошибку выходных нейронов сети. Впервые он был реализован Румельхартом, МакКлелландом и Уильямсом в 1986 году и оживил интерес к машинному обучению после спада в начале 1970-х годов. Этот алгоритм стал первым практическим методом, использованным для обучения многослойных нейронных сетей.

Шаги алгоритма:

1) Определить весовые коэффициенты связей, где начальные значения могут быть случайными или предопределенными;

2) Выполнить проход данных через сеть;

3) Вычислить ошибки на выходных нейронах, где она формируется путем сравнения полученных с ожидаемыми значениями;

4) Распространить ее обратно по сети, где на каждом слое будет вычисляться на основе другой на следующем слое;

5) Скорректировать весовые коэффициенты связей;

6) Повторить шаги 2-5 для всех обучающих примеров до достижения заданной точности или максимального количества эпох.

Данный метод основан на вычислении вклада каждого нейрона в общую ошибку и использовании этой информации для настройки весов с помощью градиентного спуска. Он был применен к нейронным сетям различной архитектуры и сложности и широко используется в распознавании образов, прогнозировании временных рядов и машинном обучении. Понимание этого алгоритма помогает лучше разбираться в принципах работы нейросетей и эффективно использовать их для решения сложных задач.

Обучение с использованием этого процесса занимает много времени и требует больших вычислительных затрат. Однако прогресс в области аппаратного обеспечения и совершенствование алгоритмов привели к широкому использованию этой техники в таких областях, как распознавание образов и обработка естественного языка.

4 Анализ предметной области и метода решения задачи

4.1 Постановка задачи

Целью данной дипломной работы является разработка приложения, использующего CNN для распознавания двумерных объектов, а именно распознавания достопримечательностей. Данное приложение очень важно и обладает множеством полезных функций. Например, оно может распознавать объекты, которые встречаются туристам во время их путешествий, и получать информацию о них. Вместо того чтобы вручную искать информацию, пользователь может просто сделать фотографию и получить мгновенную сведения о месте, тем самым сэкономив силы, особенно когда время на посещение различных мест ограничено.

В целом, планируемая разработка делают путешествия более интересными и познавательными для пользователей, обеспечивая удобство и предоставляя полезный материал.

По итогу ожидается, что программа сможет эффективно обнаруживать и классифицировать окружающие монументы на фотографиях и найдет практическое применение в туризме и общем культурном развитии.

Конкретные задачи, которые необходимо решить, включают в себя:

– Подготовку набора данных достопримечательностей для обучения сверточной нейронной сети;

– Разработки и обучение CNN для распознавания достопримечательностей;

– Оценки качества работы CNN на тестовом наборе данных;

– Написания десктопного приложения.

Чтобы построить сверточную нейронную сеть, будем использовать возможности языка Python и ее библиотеки, такие как cv2, torch, albumentation и flet.

4.2 Данные для обучения и их хранение

Для тренировки НС необходимо выбрать подходящие данные. В исследовании были найдены конкурса по распознаванию достопримечательностей, проводимого компанией Google. Однако размер набора данных - около 100 Гб, 200 000 классов и 5 миллионов изображений - представлял собой проблему. Учитывая, что возможности использовать собственные вычислительные ресурсы для обучения нет, было принято решение отказаться от этого варианта. В итоге, после дальнейших исследований, был выбран датасет Landmarks (Рисунок 5). Он содержит 210 классов и примерно 50 изображений для каждого класса. Картинки различаются по размеру, а также сняты с разных ракурсов и расстояний.

Рисунок 5 – Пример изображения, взятого из датасета Landmarks

В первую очередь, мы проведем проверку фотографий на количество каналов. Обычно предпочтительным является работа с таким, которые содержат три канала (RGB), и выбранный нами датасет в основном имеет картинки такого формата. Однако, помимо трехканальных изображений, в этом датасете мы также встречаем черно-белые и формата RGBA. Поскольку таких объектов немного (19), их удаление не должно существенно повлиять на точность конечной модели.

Для хранения данных было разработано несколько классов, которые наследуются от torch.utils.data.Dataset. При их реализации необходимо переопределить методы __getitem__ и __len__, то есть добавить возможность получения элемента по индексу и возврата длины экземпляра. Было принято решение заняться этим, так как существует несколько способов реализации этой функциональности.

Затем возникла идея: почему бы просто не хранить список путей к нашим файлам? Таким образом, расход памяти будет минимальным. Однако, при таком подходе мы жертвуем временем, необходимым для обхода данных, ведь при их хранении в виде списка путей мы должны загружать экземпляры по указанному пути каждый раз при обращении к нему, выполнять операции изменения размера и преобразования в тензоры, а затем уже работать с полученным объектом. Это может занять много времени, но у нас нет другого выбора.

4.3 Обучение CNN

Теперь перейдем к обучению. Для этого нам нужно написать цикл, в котором мы также будем вычислять метрики на валидационном наборе данных для тщательной настройки гиперпараметров модели. Однако, для того чтобы иметь такой набор, нам необходимо разделить наши материалы на обучающую и валидационную выборки. Тогда был написан метод в классе, который позволяет разделить исходные объекты на две непересекающиеся

части. Следует отметить, что внутри них каждый класс обрабатывался отдельно, поэтому мы получили сбалансированное разделение.

Для тренинга использовался оптимизатор Adam из библиотеки PyTorch, а в качестве функции потерь выбран nn.CrossEntropyLoss. Размерность изображения преобразуем в 100x100 пикселей. Сначала попробовали создавать и обучать простые модели, состоящие из двух частей:

– Сверточной части, включающей сверточные слои и пулинг;

– Полносвязанной части, состоящей из линейных слоев и небольшого количества слоев Dropout.

Очевидно, что архитектура должна была быть более сложной. Добавление слоя батч-нормализации значительно улучшило качество. Изменив параметры обучения и архитектуру сети, удалось достичь значения метрики F1 в 91% на валидационной выборке. Чтобы проверить этот результат, метрика была проанализирована более подробно. Было обнаружено, что результаты рассчитываются неверно из-за различного применения метрики F1. В существующем коде метрика возвращала тонкие значения. После исправления этой ошибки значение метрики уменьшилось до 30 % в валидационном примере и 95% в обучающем примере. Модель была явно переобучена.

4.4 Увеличение данных для обучения

Вероятно, сеть не сможет эффективно обучить 45 изображений. Что можно сделать в этом случае? Одно из решений – увеличить объем данных, применив различные преобразования к имеющимся изображениям. К каждому изображению можно применить такие преобразования, как поворот на 180 градусов и наложение фильтров. Это увеличит их разнообразие и позволит сети обучаться на более широком диапазоне материала.

Чем больше данных, тем более универсальной становится модель и тем меньше переобучения происходит на небольших наборах данных. Это

позволяет сети лучше распознавать и классифицировать новые изображения, отличающиеся от обучающего набора.

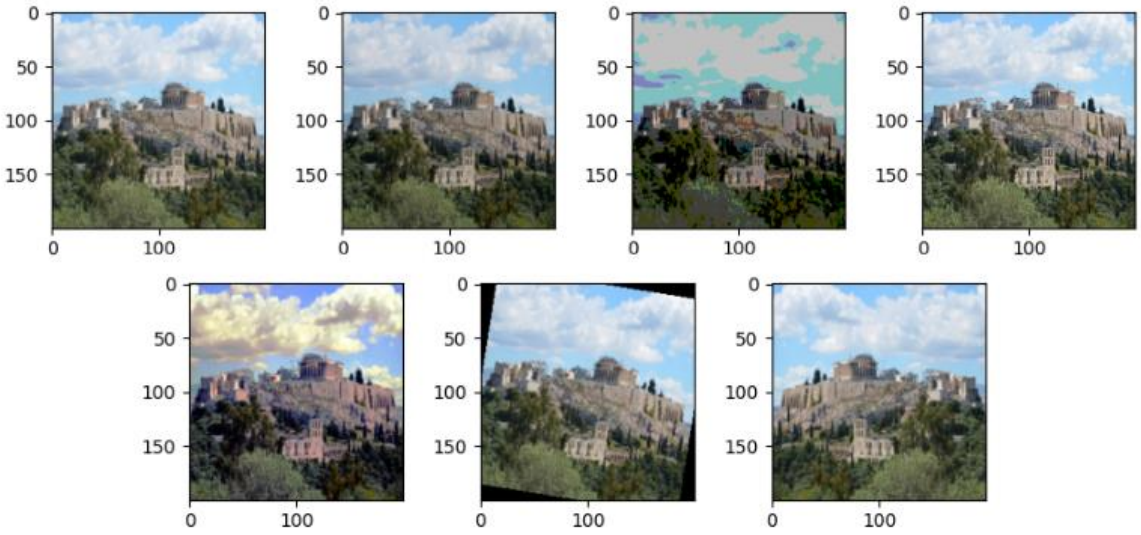

Рисунок 6 – Обработанные изображения картинки датасета Landmarks

Набор данных был увеличен в семь раз (Рисунок 6). Давайте используем его для обучения модели.

Однако модель все еще переобучается, нужно искать новый подход. Тогда появилась мысль: зачем применять только одно преобразование за раз? Если применять их последовательно, можно создавать различные комбинации и расширять набор данных.

Мы передаем аргументы конструктору класса, указывая, сколько экземпляров мы хотим получить из каждого. Затем для них к случайному изображению применяется случайная последовательность преобразований, пока не будет достигнуто желаемое количество.

Таким образом, генерируется больше вариантов изображений для обучения модели. Таким образом, улучшается обобщение модели и уменьшается проблема переобучения (Рисунок 7).

Рисунок 7 – Обработанные изображения картинки датасета Landmarks

В код были внесены некоторые изменения, а функции были заменены аналогичными из других библиотек. Причина проста: при увеличении размера набора данных доступ к элементам замедляется, и обработка путей занимает слишком много времени.

Ранее для открытия изображений использовалась библиотека PIL. Однако сравнительные тесты показали, что библиотека cv2 справляется с этой задачей гораздо быстрее. Поэтому при последнем обновлении набора данных было решено использовать cv2 вместо PIL.

Кроме того, для преобразования изображений первоначально использовался модуль torchvision.transforms. Позже выяснилось, что аналогичная функция в модуле albumentations работает быстрее. Поэтому мы перешли на ее использование для преобразования изображений.

Теперь, когда мы расширили датасет и настроили метрики, пришло время приступить к обучению. Так как теперь у нас была возможность выбрать, сколько изображений мы хотим для каждого класса, было установлено значение 2000. После длительного процесса обучения и

валидации мы получили модель нейронной сети с F1-мерой 66% на валидационной выборке. Это уже достаточно хороший результат.

Итак, что у нас в итоге: у нас есть модель нейронной сети, которая работает хорошо и качественно распознает изображения. Она состоит из входного слоя, которая принимает цветные изображения размером 3 канала, шести сверточных, имеющих определенную размерность и количество фильтров, сглаживания и трех полносвязных. Для активации используется функция ReLU. Есть даже предположения, что 30% ошибка может быть связана с некачественными фотографиями в датасете.

Для дальнейшего использования нейронной сети ее сохранили в отдельный файл cnn.pth.

4.5 Первоначальный код для распознавания достопримечательностей

Перед тем, как приступить к разработке оконного приложения, необходимо создать основу для него – код для распознавания достопримечательностей.

Для итерации по различным архитектурам не будем создавать отдельный класс. Вместо этого напишем шаблонную модель Net. Модель Net состоит из сверточных слоев, слоев пулинга, слоев активации ReLU, слоев нормализации пакета (Batch Normalization) и полносвязных слоев. Конструктор Net принимает необязательный аргумент n_classes, определяющий количество классов (достопримечательностей), которые модель может распознавать. По умолчанию n_classes равно 210.

Затем необходимо установить библиотеки albumentations и opencv-python, а также импортировать соответствующие модули: albumentations, cv2, torch и ToTensorV2 из albumentations.pytorch. Также импортируем модель Net.

Зададим переменные image_path, указывающую путь к изображению, которое требуется распознать, и device, указывающую устройство, на котором будет выполняться распознавание (в данном случае, CPU).

Создадим экземпляр модели model с помощью Net(). Затем загрузим веса из файла cnn.pth с помощью метода load_state_dict(). Важно отметить, что мы указываем аргумент map_location = torch.device('cpu'), что означает, что веса модели будут загружены на CPU.

Далее создадим экземпляр преобразования tf с использованием функции A.Resize(), изменяющая размер изображения до 100x100 пикселей. Загрузим изображение с помощью функции cv2.imread(), а затем преобразуем его из цветового пространства BGR в RGB с помощью функции cv2.cvtColor(). Применим преобразование к изображению с использованием экземпляра tf, где произойдет изменение размера изображения.

Затем преобразуем изображение в тензор и нормализуем его, разделив на 255. Значения ширины и высоты картинки сохраняются в переменные width и height.

Продолжим преобразование с использованием экземпляра ToTensorV2(). Затем изменим размерность на [1, 3, width, height] с помощью метода view().

С помощью метода to() перенесем модель my_model на указанное устройство, а затем также перенесем написанный многомерный массив на указанное устройство.

Выполним распознавание объекта на изображении, передав тензор в модель и получив выходной ответ answer.

Наконец, выведем распознанный объект. Давайте проверим код на практике. У нас есть изображение Русского музея в Санкт-Петербурке. Сохраним его в отдельный файл с именем test_image.jpg (Рисунок 8).

Рисунок 8 – Картинка для распознавания (test_image.jpg)

При компиляции программы в консоли мы получим результат (Рисунок 9):

```
Распознанный объект: Russian_Museum_in_Saint_Petersburg

Process finished with exit code 0
```

Рисунок 9 – Результат работы программы

Мы успешно получили правильный ответ на поставленную задачу! Теперь мы можем перейти к реализации программы, которое будет предоставлять функциональность по распознаванию достопримечательностей.

4.6 Написание приложения

Перед началом разработки рекомендуется создать макет, чтобы иметь представление о его внешнем виде: расположении текста, кнопок и месте вывода изображения. Ниже приведен пример прототипа программы, который был разработан в Photoshop (Рисунок 10):

Рисунок 10 – Макет приложения

Этот образец поможет нам визуализировать структуру и компоненты, что облегчит процесс реализации.

Ориентируясь на макет, приступим к написанию кода для десктопного приложения. Для его создания мы воспользуемся библиотекой Flet.

При отображении текста будем использовать простую функцию ft.Text(). Параллельно будем применять форматирование к нему, однако этому аспекту мы не будем уделять особого внимания.

В нашем приложении для распознавания достопримечательностей сначала нужно выбрать файл, содержащий изображение памятника архитектуры или природы. Создадим кнопку “Выбрать файл”, для этого будем использовать функцию, которая позволит открыть картинку с ее носителя и одновременно отобразить ее. Назовем ее pick_files_result и рассмотрим более подробно.

Эта функция содержит цикл for x in e.files, который перебирает выбранные объекты. В переменной e.files хранится список файлов для распознавания.

Внутри цикла выполняются следующие действия:

1) Копирование фотографии с сохранением имени в указанную папку;

2) Запись пути к изображению в параметр image_path;

3) Указание источника (src) картинки на путь /images/{x.name};

4) Установка видимости (visible) фото в значение True;

5) Обновление интерфейса после выполнения всех действий.

Сначала создадим функцию pick_files_dialog с помощью ft.FilePicker и поместим ее в одноименную переменную. Затем добавим кнопку btn1, используя класс ft.ElevatedButton. Зададим следующие параметры класса:

– Текст «Выберите файл», который будет отображаться в приложении;

– Обработчик события, щелчок мыши по ней, который вызывает функция pick_files_dialog.pick_files и открывается диалоговое окно для выбора файла. Пользователь приложения может выбрать его только с расширениями .jpg и .jpeg;

– Форматирование и стиль объекта.

Таким образом, нами была создана форма для выбора картинки. Следующим шагом в создании приложения будет добавление кнопки, которая на основе свёрточной нейронной сети будет выводить результат распознавания загруженной картинки, а именно, название выбранной пользователем и отображенной достопримечательности.

В предыдущей главе нами была создана нейросеть для обнаружения объектов. Теперь добавим ее в наше приложение с помощью процесса recognition(img), где параметр img будет указывать путь к фотографии. Помимо этого, необходимо создать новый документ classes_ru.txt, который будет содержать информацию о классах на русском языке, в конце функции добавим алгоритм для замены английских названий достопримечательностей на русские, и выведем результат выполнения работы приложения – название представленной достопримечательности.

Вывод будет происходить с помощью функции result, которая будет выдавать ответ recognition и сохранять его в предварительно созданной переменной answer. Затем создадим кнопку btn2, используя класс FilledButton, которая будет распознавать загруженное изображение.

Интерфейсное окно приложения для распознавания достопримечательностей представлено на следующем рисунке.

Рассмотрим работу полученного приложения после добавления всех необходимых элементов. После запуска приложения рассмотрим его можем запустить приложение и проверитм его работу (Рисунок 11).

Рисунок 11 – Окно приложения

После загрузки фотографии с достопримечательностью типа .jpg/.jpeg нажимаем на кнопку «РАСПОЗНАТЬ» и с помощью нейронной сети приложение определяет, какой объект представлен на картинке

ЗАКЛЮЧЕНИЕ

Нейронные сети и методы машинного обучения представляют собой наиболее перспективное направление развития методов обработки данных.

Использование сверточных нейронных сетей для распознавания изображений применяется в различных областях человеческой деятельности, таких как: компьютерное зрение, автоматическое распознавание лиц, медицинская диагностика и другие. Дальнейшие исследования в этой области могут привести к улучшению точности и скорости работы моделей, а также расширению возможностей их применения.

В данной работе были изучены теоретические основы нейросетей и способы обработки изображений, написано приложение для распознавания культурных достопримечательностей.

В работе проведены следующие действия:

– изучена структура сверточных нейронных сетей и способы их построения;

– собраны и подготовлены исходные данные;

– построена сверточная сеть для распознавания известных объектов архитектуры и памятников природы в среде программирования Python;

– создано десктопное приложение для распознавания достопримечательностей;

– приложение протестировано и готово к эксплуатации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Что такое нейронная сеть? // AWS Amazon : [сайт]. – 2023. – URL: https://aws.amazon.com/ru/what-is/neural-network/ (дата обращения: 22.04.2024).

2) Нейронные сети для начинающих // Хабр : [сайт]. – 2006. – URL: https://habr.com/ru/articles/312450/ (дата обращения: 22.04.2024).

3) Нейросеть // Unisender : [сайт]. – 2008. – URL: https://www.unisender.com/ru/glossary/chto-takoe-neiroseti/#anchor-2 (дата обращения: 23.04.2024).

4) Недостатки нейронных сетей: какие ограничения и проблемы есть у этой технологии и как их можно решить // vc.ru : [сайт]. – 2018. – URL: https://vc.ru/u/22269-aleksandr-shulepov/682524-nedostatki-neyronnyh-setey-kakie-ogranicheniya-i-problemy-est-u-etoy-tehnologii-i-kak-ih-mozhno-reshit (дата обращения: 23.04.2024).

5) Сверточные нейронные сети: основы и принцип работы // GeekBrains : [сайт]. – 2019. – URL: https://gb.ru/blog/svertochnye-nejronnye-seti/ (дата обращения: 24.04.2024).

6) Метод обратного распространения ошибки: полное руководство и примеры // braincatalog.ru : [сайт]. – 2023. – URL: https://braincatalog.ru/osnovy-nejronnyh-setej/metod_obratnogo_rasprostranenija_oshibki/ (дата обращения: 24.04.2024).

ПРИЛОЖЕНИЕ А

Распознавание достопримечательностей

```
import shutil
import flet as ft
import albumentations as A
import cv2
import torch
from albumentations.pytorch import ToTensorV2
from my_model import Net


def recognition(img):
    image_path = img
    device = 'cpu'

    my_model = Net()
    my_model.load_state_dict(torch.load('./my_cnn.pth',
map_location=torch.device('cpu')))
    tf = A.Resize(100, 100)

    img = cv2.imread(image_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    tensor = tf(image=img)['image']
    tensor = tensor / 255
    width = tf.width
    height = tf.height
    tensor = ToTensorV2()(image=tensor)['image'].float()
    tensor = tensor.view(1, 3, width, height)

    my_model.to(device)
    tensor = tensor.to(device)
    my_model.eval()

    answer = my_model(tensor)

    idx = torch.argmax(answer)
    with open('./classes.txt', 'r', encoding='utf-8') as f1:
        labels = f1.read()
    labels = labels.split('\n')
    with open('./classes_ru.txt', 'r', encoding='utf-8') as f2:
        labels_ru = f2.read().splitlines()
    answer = labels[idx]
    index = labels.index(answer)
    if index < len(labels_ru):
        translation = labels_ru[index]
    answer = translation

    return answer
def main(page: ft.Page):
    page.title = "Распознавание достопримечательностей"
```

```
    page.vertical_alignment = "center"
    page.horizontal_alignment = ft.CrossAxisAlignment.CENTER

    def pick_files_result(e: ft.FilePickerResultEvent):
        for x in e.files:
            shutil.copy(x.name,f"venv/Lib/site-
packages/flet/web/images/{x.name}")
            image_path.value = f"./venv/Lib/site-
packages/flet/web/images/{x.name}"
            img_display.src = f"/images/{x.name}"
            img_display.visible = True
        page.update()

    image_path = ft.Text('', visible=False)
    img_display = ft.Image(src = "", visible=False, width=450,
height=450, fit=ft.ImageFit.COVER)
    title = ft.Text(
        "РАСПОЗНАВАНИЕ ДОСТОПРИМЕЧАТЕЛЬНОСТЕЙ",
        size=60,
        theme_style = ft.TextThemeStyle.TITLE_LARGE,
        color='#665AAF'
    )
    text = ft.Text(
        "Выберите файл для распознавания: ",
        size=20,
        theme_style = ft.TextThemeStyle.LABEL_MEDIUM,
        color = '#7076c4'
    )
    pick_files_dialog =
ft.FilePicker(on_result=pick_files_result)

    page.overlay.append(pick_files_dialog)
    page.overlay.append(image_path)

    def result(e):
        answer.value = str(recognition(image_path.value))
        page.update()

    answer = ft.Text(
        size=40,
        theme_style=ft.TextThemeStyle.LABEL_MEDIUM,
        color='#665AAF'
    )

    btn1 = ft.ElevatedButton(
        "Выбрать файл",
        icon=ft.icons.UPLOAD_FILE,
        on_click=lambda _: pick_files_dialog.pick_files(
            allow_multiple=False, allowed_extensions=['jpg,
jpeg']
        ),
        style=ft.ButtonStyle(
            shape=ft.RoundedRectangleBorder(radius=10),
```

```
            color='white',
            bgcolor='#7076c4'
        )
    )

    btn2 = ft.FilledButton(
        text="РАСПОЗНАТЬ",
        on_click=result,
        style=ft.ButtonStyle(
            shape=ft.RoundedRectangleBorder(radius=10),
            color = 'white',
            bgcolor = '#665AAF'
        ),
        height=48,
        width=300
    )

    row1 = ft.Row(controls=[title],
alignment=ft.MainAxisAlignment.CENTER)
    row2 = ft.Row(controls=[text, btn1],
alignment=ft.MainAxisAlignment.CENTER)
    row3 = ft.Row(controls=[img_display],
alignment=ft.MainAxisAlignment.CENTER)
    row4 = ft.Row(controls=[btn2],
alignment=ft.MainAxisAlignment.CENTER)
    row5 = ft.Row(controls=[answer],
alignment=ft.MainAxisAlignment.CENTER)
    column1 = ft.Column(controls=[row1, row2, row3, row4, row5],
alignment=ft.MainAxisAlignment.START)

    page.add(
        column1
    )

ft.app(target=main, view=ft.WEB_BROWSER, upload_dir="images")
```

Рисунок А. 1 – Текст кода