МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра компьютерных технологий и систем

СТАВИЦКАЯ
Ксения Андреевна

РАЗРАБОТКА СИСТЕМЫ АНАЛИЗА ПОСЕЩАЕМОСТИ С ПРИМЕНЕНИЕМ МЕТОДОВ КОМПЬЮТЕРНОГО ЗРЕНИЯ

Дипломная работа

Научные руководители:
старший преподаватель,
С.В. Шолтанюк;
ассистент кафедры,
А.Г. Каркоцкий

Допущена к защите:
«___»_____ 2025г.
Зав. кафедрой компьютерных технологий и систем,
доктор педагогических наук,профессор В.В.Казаченок

Минск, 2025

СОДЕРЖАНИЕ

РЕФЕРАТ

Дипломная работа – 71 страница, 5 рисунков, 2 таблицы, 1 приложение, 20 источников.

Ключевые слова – КОМПЬЮТЕРНОЕ ЗРЕНИЕ, РАСПОЗНАВАНИЕ ЛИЦ, ЭМБЕДИНГИ, YOLOV5, MOBILENETV3, ARCFACE, АНАЛИЗ ПОСЕЩАЕМОСТИ, FASTAPI.

Объект исследования – методы и технологии компьютерного зрения, применяемые для автоматического распознавания лиц на фотографиях и регистрации посещаемости, включая архитектуры нейронных сетей и подходы к построению векторных представлений лиц.

Предмет исследования – технологии и методы автоматизированного распознавания лиц на изображениях для анализа посещаемости с применением алгоритмов компьютерного зрения и нейросетевых моделей.

Цель работы – разработка программного решения для автоматизированного учёта посещаемости на основе анализа фотоснимков с применением современных методов компьютерного зрения и нейронных сетей.

Методы исследования – теоретические: изучение литературы по теме распознавания лиц, нейросетевых архитектур и подходов к построению эмбедингов; анализ современных инструментов глубокого обучения и веб-разработки. Практические: реализация детекции лиц с помощью YOLOv5, генерация эмбедингов через MobileNetV3 с функцией потерь ArcFace, сравнение эмбедингов, построение API на FastAPI, тестирование точности распознавания в различных условиях.

Полученные результаты и их новизна – разработана и реализована система анализа посещаемости на основе собственных нейросетевых моделей детекции (YOLOv5) и распознавания лиц. Новизна заключается в полной самостоятельной сборке и обучении компонентов, их интеграции в единую архитектуру с веб-интерфейсом и базой данных, а также адаптации к реальным условиям съёмки.

Достоверность материалов и результатов дипломной работы – результаты подтверждены тестированием на открытых и собственных данных, валидацией модели по точности распознавания и устойчивостью к внешним условиям. Использованы современные методы и библиотеки, что обеспечивает воспроизводимость и объективность.

Область возможного практического применения – система может использоваться в вузах, школах, офисах и других организациях для автоматизированного учёта посещаемости по изображениям, обеспечивая точность, надёжность и снижение затрат на ручной контроль.

РЭФЕРАТ

Дыпломная работа – Дыпломная праца, 71 старонка, 5 малюнкаў, 2 табліцы, 1 дадатак, 20 крыніц.

Ключавыя словы – КАМП'ЮТАРНЫ ЗРОК, РАЗНАЗНАВАННЕ АСОБ, ЭМБЕДЫНГІ, YOLOV5, MOBILENETV3, ARCFACE, АНАЛІЗ НАВЕДНАСЦІ, FASTAPI.

Аб'ект даследавання – метады і тэхналогіі камп'ютэрнага зроку, якія прымяняюцца для аўтаматычнага распазнання асоб на фатаграфіях і рэгістрацыі наведвальнасці, уключаючы архітэктуры нейронавых сетак і падыходы да пабудовы вектарных уяўленняў асоб.

Прадмет даследавання – тэхналогіі і метады аўтаматызаванага распазнання асоб на малюнках для аналізу наведвальнасці з ужываннем алгарытмаў кампутарнага зроку і нейросетевых мадэляў.

Мэта работы – распрацоўка праграмнага рашэння для аўтаматызаванага ўліку наведвальнасці на аснове аналізу фотаздымкаў з прымяненнем сучасных метадаў камп'ютэрнага зроку і нейронавых сетак.

Метады даследавання – тэарэтычныя: вывучэнне літаратуры па тэме распазнання асоб, нейросетевых архітэктур і падыходаў да пабудовы эмбедынгаў; аналіз сучасных інструментаў глыбокага навучання і вэб-распрацоўкі. Практычныя: рэалізацыя дэтэкцыі асоб з дапамогай YOLOv5, генерацыя эмбедынгаў праз MobileNetV3 з функцыяй страт ArcFace, параўнанне эмбедынгаў, пабудова API на FastAPI, тэсціраванне дакладнасці распазнання ў розных умовах.

Атрыманыя вынікі і іх навізна – распрацавана і рэалізавана сістэма аналізу наведвальнасці на аснове ўласных нейросетевых мадэляў дэтэкцыі (YOLOv5) і распазнання асоб. Навізна заключаецца ў поўнай самастойнай зборцы і навучанні кампанентаў, іх інтэграцыі ў адзіную архітэктуру з вэб-інтэрфейсам і базай дадзеных, а таксама адаптацыі да рэальных умоў здымкі.

Дакладнасць матэрыялаў і вынікаў дыпломнай працы – вынікі пацверджаны тэсціраваннем на адкрытых і ўласных дадзеных, валідацыяй мадэлі па дакладнасці распазнання і ўстойлівасцю да знешніх умоў. Выкарыстаны сучасныя метады і бібліятэкі, што забяспечвае ўзнаўляльнасць і аб'ектыўнасць.

Вобласць магчымага практычнага прымянення – сістэма можа выкарыстоўвацца ў ВНУ, школах, офісах і іншых арганізацыях для аўтаматызаванага ўліку наведвальнасці па выявах, забяспечваючы дакладнасць, надзейнасць і зніжэнне затрат на ручны кантроль.

ABSTRACT

Graduate Work – 71 pages, 5 figures, 2 tables, 1 appendix, 20 references.

Keywords – COMPUTER VISION, FACE RECOGNITION, EMBEDDINGS, YOLOV5, MOBILENETV3, ARCFACE, VISITATION ANALYSIS, FASTAPI.

The object of the research – computer vision methods and technologies used for automatic face recognition in photographs and attendance registration, including neural network architectures and approaches to constructing vector representations of faces.

The subject of the research – technologies and methods of automated face recognition in images for attendance analysis using computer vision algorithms and neural network models.

The aim of the research – to develop a software solution for automated attendance registration based on photo analysis using modern computer vision and neural network techniques.

Research methods – Theoretical: study of literature on the topic of face recognition, neural network architectures and embedding approaches; analysis of modern deep learning and web development tools. Practical: implementation of face detection using YOLOv5, generation of embeddings via MobileNetV3 with ArcFace loss function, comparison of embeddings, construction of API on FastAPI, testing of recognition accuracy in different conditions.

The results of the work and their novelty – an attendance analysis system based on proprietary neural network models of detection (YOLOv5) and face recognition was developed and implemented. The novelty lies in the complete self-assembly and training of components, their integration into a unified architecture with web-interface and database, as well as adaptation to real shooting conditions.

Authenticity of the materials and results of the diploma work – the results are confirmed by testing on open and own data, validation of the model in terms of recognition accuracy and stability to external conditions. Modern methods and libraries were used, which ensures reproducibility and objectivity.

Recommendations on the usage – the system can be used in universities, schools, offices and other organizations for automated accounting of attendance by images, providing accuracy, reliability and reducing the cost of manual control.

ВВЕДЕНИЕ

Современный мир активно меняется благодаря стремительному развитию технологий. Нейронные сети, как одна из ключевых технологий искусственного интеллекта, становятся неотъемлемой частью нашей повседневной жизни. Они находят применение в самых различных сферах — от распознавания речи и изображений до прогнозирования поведения пользователей в интернете и автоматизации процессов на производстве. Компьютерное зрение, основанное на методах глубокого обучения, в частности, становится мощным инструментом для анализа визуальных данных, что открывает новые возможности в таких областях, как медицина, транспорт, безопасность и образование.

Одной из ключевых задач, решаемых с помощью методов компьютерного зрения, является распознавание лиц. Эта технология уже используется для разблокировки смартфонов, контроля доступа в здания, а также в системах видеонаблюдения. Распознавание лиц позволяет автоматизировать процессы идентификации и аутентификации, повышая их точность и снижая затраты на ручную проверку. В контексте образовательных учреждений данная технология имеет огромный потенциал. Например, она может быть применена для анализа посещаемости занятий, что актуально как для студентов, так и для преподавателей.

В университетах контроль посещаемости студентов зачастую остается рутинным процессом, который требует значительных временных затрат. Внедрение автоматизированной системы анализа посещаемости с использованием методов компьютерного зрения позволит решить эту проблему, сократив время на обработку данных и исключив возможность человеческой ошибки. Такая система может автоматически фиксировать присутствие студентов на занятиях, основываясь на данных, полученных с камер наблюдения, и предоставлять преподавателям и администраторам удобные отчеты.

Целью данной работы является разработка системы анализа посещаемости с применением методов компьютерного зрения, основанной на задаче распознавания лиц. Работа включает исследование основополагающих концепций нейронных сетей и компьютерного зрения, а также практическую реализацию задачи распознавания лиц с использованием датасета Labeled Faces in the Wild (LFW) [1]. Полученные результаты могут быть применены не только в образовательной среде, но и в других областях, где требуется автоматизация процессов идентификации людей, например, в бизнесе, здравоохранении и управлении доступом.

ГЛАВА 1
КОМПЬЮТЕРНОЕ ЗРЕНИЕ. ОБУЧЕНИЕ С УЧИТЕЛЕМ

1.1 Введение в компьютерное зрение

Компьютерное зрение – это область искусственного интеллекта, которая разрабатывает методы для автоматического извлечения, анализа и интерпретации визуальной информации из изображений и видео. Иными словами, цель компьютерного зрения заключается в том, чтобы дать компьютерам “зрение”, позволяя им обнаруживать, идентифицировать и понимать объекты и сцену на цифровых изображениях подобно тому, как это делает человеческий глаз. За последние десятилетия эта область прошла большой путь: от простых алгоритмов обработки изображений до современных глубоких нейронных сетей, достигающих превосходной точности распознавания. Сегодня компьютерное зрение находит применение во множестве сфер – от медицины и промышленности до автомобилей с автопилотом и системы видеонаблюдения, повсеместно изменяя нашу жизнь. В данном разделе представлены основные задачи и история развития компьютерного зрения, а также ключевые направления его практического применения.

1.1.1 Задачи компьютерного зрения

Системы компьютерного зрения решают широкий спектр задач, связанных с анализом визуальных данных. К основным задачам относятся:

- Классификация изображений. Задача классификации состоит в отнесении всего входного изображения к одному из заранее определенных классов . Например, на вход поступает фотография, и модель должна определить, что на ней изображено – кошка, собака или автомобиль. Формально, классификацию можно определить как поиск отображения $f : X \to Y$, где X – пространство изображений, а Y – дискретное множество меток классов. Модель, решающая задачу классификации, выдает метку $y \in Y$ для каждого входного изображения $x \in X$.

- Детекция объектов. В отличие от классификации, детекция (выделение объектов) предусматривает не только определение классов объектов, но и локализацию каждого объекта на изображении с помощью ограничивающей рамки. Алгоритм детекции обрабатывает изображение и выдает координаты рамок, обрамляющих обнаруженные объекты, а также класс (тип) каждого найденного объекта. Классическим примером

является обнаружение пешеходов или автомобилей на видеокадре системы автоматического вождения.

- Сегментация изображения. Сегментация разделяет изображение на смысловые области на уровне пикселей. Существуют две основные разновидности: семантическая сегментация, при которой каждому пикселю изображения присваивается метка класса (например, фон, дорога, автомобиль, пешеход), и сегментация экземпляров, при которой различаются отдельные объекты одного класса. Результатом сегментации является маска, выделяющая область каждого обнаруженного объекта с точностью до пикселя.

- Оценка позы и отслеживание. Оценка позы (pose estimation) заключается в нахождении ключевых точек объекта (например, суставов человека) и восстановлении конфигурации объекта в пространстве. Задача отслеживания (tracking) тесно связана с детекцией и решается на видеопоследовательностях – требуется проследить положение выбранного объекта (или множества объектов) на серии кадров видеоролика. К примеру, алгоритмы отслеживания позволяют в реальном времени следить за движением автомобиля на серии изображений с дорожных камер.

- Прочие задачи. К другим важным задачам компьютерного зрения относятся восстановление глубины и 3D-реконструкция сцены (по двум или более изображениям сцены), вычисление оптического потока (поля смещения между последовательными кадрами), распознавание действий на видео, генерация описаний к изображениям (image captioning) и др. Каждая из этих задач имеет свои алгоритмические особенности, но все они направлены на извлечение максимально полной информации из визуальных данных.[2]

Следует отметить, что современные алгоритмы часто решают сразу несколько связанных задач. Например, единая нейросетевая модель может выполнять и детекцию, и сегментацию объектов на изображении. Развитие комплексных моделей, способных одновременно классифицировать и локализовывать объекты, является одной из актуальных тенденций компьютерного зрения.

1.1.2 История развития компьютерного зрения

История компьютерного зрения насчитывает более 60 лет активных исследований. Ее условно можно разделить на несколько этапов:

1960-е годы – зарождение области. Одним из первых значимых трудов считается работа Лоуренса Робертса (Lawrence G. Roberts) 1963 года, в

которой была предложена концепция машинного восприятия трехмерных объектов. В этот период исследования были сосредоточены на базовых операциях обработки изображений. Разрабатывались методы фильтрации, детектирования простых признаков. В конце 1960-х были предложены первые алгоритмы выделения границ на изображениях, например оператор Собеля для детектирования границ посредством вычисления градиента яркости. Эти ранние достижения заложили фундамент для дальнейших экспериментов.

1970-е годы – распознавание образов. С увеличением вычислительных возможностей появились первые системы распознавания простых объектов. Исследователи разрабатывали алгоритмы шаблонного сопоставления и анализ основных признаков (форм, текстур) на изображении . Однако возможности таких систем были ограничены – методы оказались чувствительными к масштабированию, повороту объекта и шуму в данных. Тем не менее, в это десятилетие сформировалась сама постановка задачи распознавания образов, и появились первые датасеты для тестирования алгоритмов. Ограниченные ресурсы ЭВМ того времени не позволяли реализовать сложные модели, поэтому фокус был на алгоритмах, выполнимых на доступном оборудовании.

1980-е – 1990-е годы – развитие алгоритмов и признаковых методов. В 1980-е существенный вклад в теорию зрения внесла работа Д. Марра, предложившего многоуровневую модель зрительного восприятия (сырые примитивы – признаки – трёхмерное представление сцены). Появились классические методы извлечения признаков: оператор Канни для поиска границ (1986), методы анализа оптического потока (Horn Schunck, 1981) и др. В конце 1990-х были изобретены высокоуровневые дескрипторы изображений, например SIFT (Scale-Invariant Feature Transform) Д. Лоу (1999) – метод выделения ключевых точек и инвариантных локальных признаков, совершивший революцию в задаче сопоставления изображений. Параллельно росли объемы доступных данных: были сформированы базы изображений для обучения и оценки алгоритмов (например, MNIST для рукописных цифр в 1998 г.). Методы того времени во многом опирались на тщательно конструированные человеком признаки и классические алгоритмы машинного обучения (например, классификаторы на основе методома ближайших соседей или SVM).

2010-е годы – революция глубокого обучения. Кардинальный поворотным моментом стало применение глубоких нейронных сетей к задачам компьютерного зрения. Хотя сами нейронные сети известны с 1960-х, именно развитие графических процессоров и появление больших размеченных датасетов позволили раскрыть их потенциал.[7] Отправной точкой принято считать 2012 год, когда нейросеть AlexNet (8 слоев, 60 миллионов параметров) одержала убедительную победу в соревновании ImageNet по классификации изображений, достигнув топ-1 точности 62,5% против 50% у ближайшего конкурента. Этот результат продемонстрировал мощь глубокого обучения и привлек к нему колоссальный интерес исследователей. В последующие

годы архитектуры сверточных нейросетей быстро эволюционировали (см. раздел 1.3.2): появились более глубокие и эффективные модели – VGG (2014), ResNet (2015) и др., каждое новое поколение улучшало точность и расширяло возможности компьютерного зрения. Также важно отметить роль данных: в 2010-е формируются огромные базы изображений с разметкой (ImageNet насчитывает 1,2 млн изображений в 1000 классах), что стало топливом для "голодных" до данных нейросетей.

2020-е годы – объединение с успехами смежных областей. В последние годы наблюдается слияние идей компьютерного зрения с методами из обработки естественного языка и появление универсальных моделей. Трансформеры, изначально разработанные для NLP, были успешно адаптированы для зрения (Vision Transformers, 2020), а сверточные сети получили дальнейшее развитие с учетом идей трансформеров – пример тому ConvNeXt (2022), современная конволюционная архитектура, сопоставимая по точности с трансформерными моделями. Кроме того, возрастают требования к интерпретируемости моделей и эффективности обучения на ограниченных данных. Компьютерное зрение продолжает стремительно развиваться, и история этой области далека от завершения.

1.1.3 Основные области применения компьютерного зрения

Высокая точность и автоматизация, достигнутые современными моделями компьютерного зрения, обусловили их широкое распространение во многих отраслях. Перечислим некоторые ключевые области применения:

- Медицина и здравоохранение. Компьютерное зрение революционизирует анализ медицинских изображений. Алгоритмы на основе нейросетей помогают врачам диагностировать заболевания по рентгенограммам, МРТ и КТ-снимкам с высокой точностью. Например, модели детекции и сегментации используются для обнаружения опухолей, микрокальцификаций при маммографии, участков кровоизлияний на томограммах мозга. Автоматический анализ изображений ускоряет диагностику и повышает ее объективность. Отдельное направление – анализ эндоскопического видео и микроскопических изображений в патологии, где зрение ИИ помогает выявлять аномалии, незаметные невооруженным глазом специалиста.

- Автомобильный транспорт и беспилотные системы. В сфере автономного вождения компьютерное зрение играет центральную роль. Системы технического зрения автомобиля обрабатывают данные с видеокамер в реальном времени, решая задачи обнаружения дорожной разметки, распознавания дорожных знаков, детекции пешеходов

и других автомобилей. Модели вроде YOLO способны выполнять обнаружение объектов на дорожной сцене за считанные миллисекунды, что позволяет реализовать функции предотвращения столкновений и автоматического движения. Кроме того, зрение применяется в системах мониторинга водителя (отслеживание взгляда, определение усталости) и в интеллектуальных ассистентах парковки.

- Безопасность и видеонаблюдение. Системы безопасности активно используют алгоритмы распознавания лиц и объектов. Камеры наблюдения в сочетании с моделями компьютерного зрения могут в режиме реального времени обнаруживать подозрительную активность, распознавать личности и выявлять оставленные предметы. Технологии face recognition достигли высокой точности идентификации людей даже в толпе и при неполном ракурсе лица. Также зрение используется для автоматического распознавания автомобильных номеров (ANPR) и мониторинга соблюдения правил (например, фиксация проезда на красный свет).

- Промышленность и производство. В промышленном контроле качества компьютерное зрение давно стало незаменимым инструментом. Специализированные системы машинного зрения проверяют изделия на конвейере на наличие дефектов (трещин, неправильной формы, цвета и т.п.) с высокой скоростью и точностью, недоступными человеку. В машиностроении и электронике зрение применяется для высокоточного позиционирования деталей роботами-манипуляторами, для считывания штрих- и QR-кодов, контроля заполнения и упаковки продукции. Современные технологии позволяют создавать относительно недорогие и гибкие в перенастройке зрительные системы для самых разных отраслей – от пищевой промышленности до производства микрочипов.

- Сельское хозяйство и экология. В агросекторе компьютерное зрение помогает автоматизировать мониторинг состояния посевов и управление техникой. Дроны, оснащенные камерами, снимают поля, а алгоритмы сегментации выделяют участки с признаками болезней растений или недостатка влаги. Также ведутся разработки систем сортировки урожая (например, автоматическое отделение спелых фруктов от зеленых на основе анализа изображений). В охране окружающей среды зрение используют для учета популяций животных: модели детекции позволяют выявлять животных на фотографиях с фотоловушек и даже определять их вид. Это облегчает мониторинг исчезающих видов и борьбу с браконьерством.

Таким образом, компьютерное зрение находит применение всюду, где требуется анализ визуальной информации. От точных научных приборов

до потребительских смартфонов – алгоритмы зрения работают, чтобы облегчить труд человека и решить те задачи, которые ранее казались невыполнимыми в автоматическом режиме. Дальнейшее совершенствование методов (в частности, объединение с другими направлениями ИИ) будет только расширять спектр возможностей компьютерного зрения.

1.2 Методы обработки изображений

1.2.1 Предобработка изображений

Предобработка изображений — это важный этап в обработке изображений, направленный на улучшение качества изображений и подготовку их к дальнейшему анализу. Включает в себя несколько ключевых методов: фильтрация, нормализация, преобразования. Ниже рассмотрим каждый способ отдельно.

Метод фильтрации используется для удаления шума с изображения и улучшения его качества. Наиболее распространенные типы фильтрации — это линейные фильтры, такие как фильтры Гаусса, и нелинейные фильтры, например, медианный фильтр.

Нормализация - этот процесс заключается в преобразовании значений пикселей изображения в стандартный диапазон. Обычно это делается для улучшения контраста изображения и обеспечения одинаковых условий для дальнейшего анализа.

Преобразования включают в себя такие операции, как изменение масштаба изображения, повороты, зеркальные отражения и преобразования перспективы. Эти методы используются для улучшения качества изображений или для приведения изображений к единому формату перед их дальнейшей обработкой.

1.2.2 Методы извлечения признаков

Извлечение признаков — это процесс выделения значимой информации из изображения, которая затем используется для его анализа, распознавания объектов и классификации.

Контуры: один из самых распространенных методов извлечения признаков, который позволяет выделить границы объектов на изображении. Контуры помогают определить форму объектов и их положение на изображении. Для выделения контуров часто используются такие операторы, как оператор Собеля(рис.1.1), Кенни или Лаплас. Эти методы вычисляют изменения яркости между соседними пикселями и выделяют области с резкими переходами, которые соответствуют границам объектов.

Извлечение текстуры позволяет анализировать и классифицировать поверхности объектов по их визуальным особенностям. Текстурные признаки

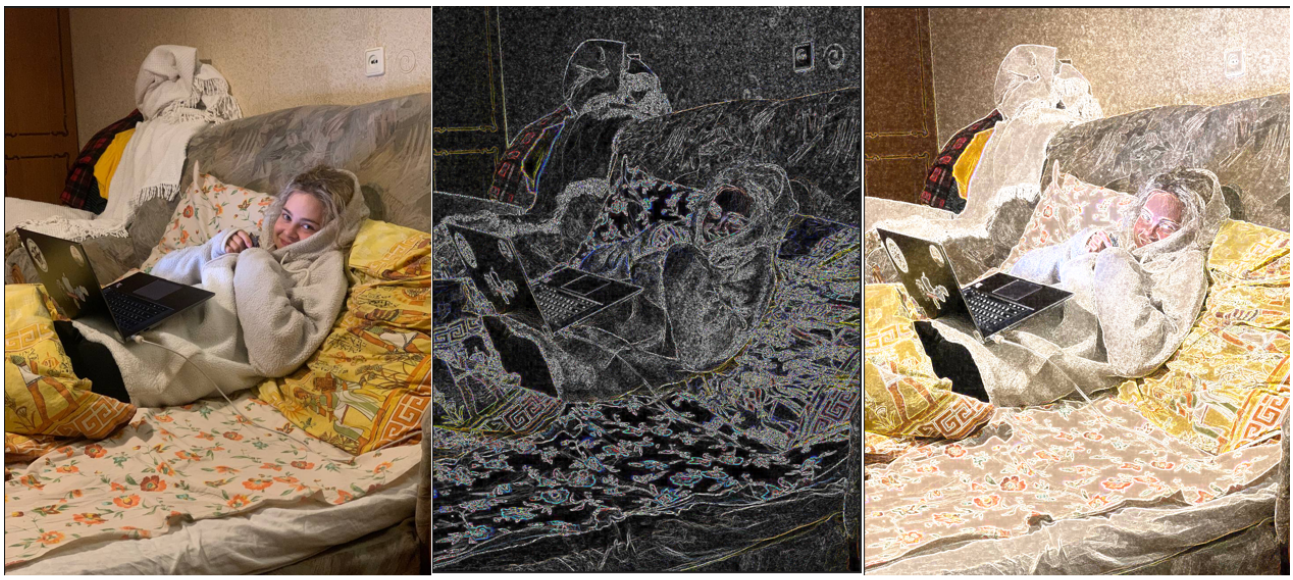

Рисунок 1.1 — Использование маски Собеля:
(a)- исходное изображение,
(b) - вычисление градиента изображения по маске Собеля,
(c)-результат сложения исходного изображения с обработанным

могут включать в себя такие параметры, как регулярность, однородность и сложность поверхности.

Цвет является важным признаком для классификации и распознавания объектов, особенно в задачах, связанных с изображениями, где цвет имеет явное значение. Извлечение цветовых признаков может включать в себя использование различных цветовых пространств, таких как RGB, HSV или Lab.

1.2.3 Аугментация данных

Один из самых мощных способов улучшить обобщающую способность моделей – аугментация данных, т.е. искусственное расширение обучающей выборки за счет применения к изображениям случайных преобразований, не меняющих их смысл. Для изображений типичные аугментации включают: случайные повороты, сдвиги, масштабирование, горизонтальное отражение, вариацию яркости/контрастности/цвета, добавление шума, небольшое размытие и др. Идея: модель увидит один и тот же объект в разных “видах” и будет вынуждена выучить более инвариантные признаки. Например, при обучении классификатора кошек и собак часто применяют случайное горизонтальное отражение – это эффективно удваивает набор изображений, а модель не будет зависеть от того, повёрнуто животное вправо или влево. В AlexNet 2012 активно использовались подобные аугментации (случайные обрезки изображения, отражения, изменения цветовых каналов)

для увеличения датасета.

Data augmentation фактически добавляет априорное знание о инвариантностях задачи. Для задач, где легко задать такие трансформации, аугментация может драматически улучшить результаты. В области компьютерного зрения трудно представить обучающий конвейер без аугментаций практически все соревнования по классификации, детекции, сегментации включают продуманный набор аугментаций для расширения данных. Более того, появились методы обучаемой аугментации (AutoAugment, RandAugment), которые автоматически подбирают оптимальный набор случайных преобразований, и даже методы генерации новых образцов с помощью GAN-ов или диффузионных моделей, дополняющих реальный датасет синтетическими примерами.

1.3 Обучение с учителем

В основе большинства современных систем компьютерного зрения лежат методы обучения с учителем – подхода машинного обучения, предполагающего наличие размеченных данных для обучения модели. В данном разделе рассматриваются формальная постановка задачи обучения с учителем, основные типы задач этого обучения (классификация, регрессия, ранжирование), а также ключевые метрики, применяемые для оценки качества моделей.

1.3.1 Формальная постановка задачи

Обучение с учителем (supervised learning) – это класс методов машинного обучения, в котором алгоритм строит модель на основе обучающих примеров, состоящих из пар “объект целевое значение”.[9] Пусть дана выборка данных $D = (x_i,\, y_i)_{i=1}^{N}$, где каждый $x_i \in X$ входной объект (в контексте компьютерного зрения x_i может быть изображением или частью изображения), а $y_i \in Y$ – соответствующее корректное значение, или метка, которую должен предсказать алгоритм (например, класс объекта на изображении или числовой признак). Пространство X называют пространством признаков, а Y – пространством откликов. Требуется найти такую функцию (модель) $f : X \to Y$ из заданного класса функций, которая оптимально приближает зависимость y от x на основании обучающей выборки.

Математически процесс обучения обычно формулируется как задача оптимизации: выбирается функция ошибки (или целевой функционал) $L(f(x), y)$, измеряющая расхождение между предсказанием модели $f(x)$ и истинным значением y. Алгоритм обучения ищет такую функцию f^* из класса допустимых моделей $\mathcal{F}$, которая минимизирует среднюю ошибку на

обучающих данных:

$$f^* = \arg\min_{f \in \mathcal{F}} \frac{1}{N} \sum_{i=1}^{N} L(f(x_i),\, y_i). \tag{1.1}$$

Часто вместо средней ошибки рассматривают функционал риска

$$R[f] = \mathbb{E}_{(x,y)\sim P}[L(f(x), y)], \tag{1.2}$$

то есть математическое ожидание ошибки на данных, распределенных согласно некоему неизвестному распределению $P(x, y)$. В реальных условиях распределение неизвестно, и на практике оптимизируют эмпирический риск (суммарную ошибку на обучающей выборке), как показано выше. Этот подход известен как принцип минимизации эмпирического риска.

Обобщающая способность. Ключевой концепцией обучения с учителем является обобщение способность построенной модели правильно предсказывать ответы на новых, невиденных ранее данных. Обучение считается успешным, если модель $f(x)$ хорошо приближает целевую зависимость не только на обучающей выборке, но и за ее пределами. Формально, мы хотим минимизировать не эмпирический риск на обучении, а истинный риск $R[f]$ на всевозможных объектах из распределения данных. Разница между ошибкой на обучении и ожидаемой ошибкой на новых данных определяется степенью переобучения (overfitting). Модель с высокой обобщающей способностью избегает переобучения, то есть не просто запоминает обучающие примеры, а улавливает общие закономерности.

Для обеспечения обобщения в обучение вводят различные техники: контроль сложности модели (например, ограничение числа параметров), регуляризацию и разделение данных на обучающую и контрольную (валидационную) выборки. Валидационная выборка служит для мониторинга качества модели в ходе обучения – резкое ухудшение качества на валидации при продолжении снижения ошибки на обучении свидетельствует о переобучении. Такой прием, как Early Stopping (ранняя остановка), прерывает обучение, как только ошибка на валидации перестает снижаться, тем самым сохраняя модель на этапе оптимального обобщения.

Таким образом, формально задача обучения с учителем состоит в выборе модели f с минимальной ошибкой на примерах из неизвестного распределения данных при наличии конечной обучающей выборки. Ключевой вызов – достижение баланса между точностью на обучающих данных и обобщающей способностью модели на новых данных (проблема, известная как дилемма смещения дисперсии, или bias–variance tradeoff)[16]. Решение этой задачи включает подбор подходящей гипотезы (семейства моделей), метода оптимизации и механизмов контроля переобучения.

1.3.2 Основные типы задач

В рамках обучения с учителем выделяются различные типы задач в зависимости от природы целевой переменной y и цели предсказания. Рассмотрим три основных типа: классификация и регрессия.

Классификация. В задаче классификации выходное пространство Y является конечным дискретным множеством меток (классов). Модель должна научиться относить каждый вход x к одному из этих классов. Примеры классификационных задач: определение наличия дефекта на изображении (классы “дефект”/“норма”), распознавание hand-written цифры (классы 0–9), определение породы животного на фото (классы пород). Классификация может быть бинарной (два класса) или многоклассовой (несколько категорий). Формально, классификация – это предсказание категориальной переменной. Модель, решающая такую задачу, реализует отображение $X \to 1, 2, \ldots, K$, где K – число классов. Классификация считается успешной, если модель с высокой точностью восстанавливает правильную метку для объектов из тестовой выборки.

Существует также вариант multi-label классификации, когда каждому объекту может соответствовать не один, а сразу несколько классов (например, на фотографии могут одновременно присутствовать несколько типов объектов, и нужно выявить все). В таких случаях пространство ответов Y часто рассматривают как набор бинарных флагов по каждому из возможных классов.

Регрессия. В задаче регрессии целевая переменная y – непрерывная величина (вещественное число или вектор чисел). Требуется предсказывать числовое значение по заданным признакам объекта. Примеры регрессии: предсказание расстояния до объекта по изображению (задача монокулярной оценки глубины), прогнозирование рейтинга фильма по кадру, оценка возраста человека по фотографии. В формальном виде модель решает отображение $f : X \to \mathbb{R}$ (в общем случае $f : X \to \mathbb{R}^m$). Регрессионный алгоритм обучается минимизировать ошибку между предсказанными и реальными числовыми значениями (например, среднеквадратичную ошибку). Успешность регрессии оценивается тем, насколько близки предсказания к истинным значениям по метрикам вроде R^2 или средней абсолютной ошибки.

Классификация и регрессия во многом схожи методологически (в обоих случаях решается задача аппроксимации зависимости), но отличаются в типе предсказываемых значений. Грубо говоря, классификация предсказывает категорию, а регрессия – число. Многие алгоритмы можно настроить под оба типа задач (например, нейронные сети с соответствующей функцией активации на выходе и функцией потерь)

Итак, тип задачи в обучении с учителем определяется типом выходного значения: дискретный класс (классификация) или непрерывная величина (регрессия). Во всех случаях формальное ядро одно – есть обучающая выборка пар (x, y) и алгоритм, стремящийся найти зависимость $y = f(x)$ с наилучшей

точностью. Различия проявляются в выборе моделей, функции потерь и метрик качества, которые наиболее подходят для каждой конкретной постановки.[1]

1.3.3 Метрики качества

Для количественной оценки эффективности моделей обучения с учителем используются различные метрики качества. Правильный выбор метрики имеет большое значение, так как разные аспекты качества модели могут быть более или менее важны в зависимости от задачи. Ниже рассмотрены основные метрики, применяемые в задачах классификации (поскольку именно они наиболее характерны для компьютерного зрения); многие из них имеют аналоги для регрессии и ранжирования.

Accuracy (точность классификации). Эта метрика равна доле правильно классифицированных примеров среди всех примеров:

$$\text{Accuracy} = \frac{N_{\text{correct}}}{N_{\text{total}}}, \tag{1.3}$$

где N_{correct} – число объектов, для которых предсказанная метка совпала с истинной, а N_{total} – общее число объектов. Accuracy – простая и широко используемая метрика, понятная интуитивно (процент верных ответов). Однако она не всегда отражает качество модели на несбалансированных данных. Например, если 95% изображений – класс A, а 5% – класс B, то тривиальный классификатор, всегда выбирающий A, даст Accuracy = 95%, хотя полностью ошибается на редком классе B. Поэтому для более тонкого анализа используют метрики Precision и Recall.

Precision и Recall (точность и полнота обнаружения). Эти метрики вводятся для каждой категории (классу) положительных примеров и особенно важны в задачах, где один класс рассматривается как “интересующий” (например, класс “дефект обнаружен” против “дефект не обнаружен”). Precision (точность в терминах информационного поиска) – это доля истинно положительных объектов среди всех, которые модель отнесла к положительному классу. Recall (полнота, или чувствительность) доля объектов положительного класса, которые модель успешно обнаружила среди всех действительно положительных. Если обозначить: TP (True Positives) – количество правильно классифицированных положительных случаев, FP (False Positives) – количество ошибочно помеченных как положительные, FN (False Negatives) – количество пропущенных положительных (которые модель неверно отнесла к отрицательным), то:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}. \tag{1.4}$$

Precision характеризует достоверность положительных срабатываний модели (скольких из предсказанных “кошек” на самом деле кошки), а Recall

– полноту выявления целевого класса (какую долю всех кошек модель нашла).[3] Эти метрики находятся в потенциальном конфликте: модель может увеличивать Recall ценой снижения Precision (ловить больше положительных, но с большим числом ложных срабатываний) и наоборот. Поэтому часто рассматривают их совместно.

F-мера. Для объединения Precision и Recall в единый показатель используется их гармоническое среднее – F-мера, или F1-score (когда берется среднее с равными весами):

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{1.5}$$

F1-мера будет высокой, только если и Precision, и Recall достаточно высоки (балансируются). Она особенно полезна при несбалансированных классах, где высокая Accuracy малоинформативна. Максимальное значение F1 = 1 достигается только при безошибочном обнаружении всех положительных (нет FN) без ложных срабатываний (нет FP).

Для многоклассовых задач часто вычисляют Precision, Recall и F1 по каждому классу, а затем усредняют – либо с весами по числу примеров каждого класса (weighted average), либо усреднение “macro” (простое среднее по классам). Последнее оценивает качество модели равноправно по всем классам, что важно, если интересуют результаты и на малочисленных классах.[3]

ROC-AUC. В задачах бинарной классификации, особенно с несбалансированными классами, полезна метрика AUC (Area Under Curve) – площадь под ROC-кривой. ROC-кривая строится по массиву предсказаний модели как зависимость True Positive Rate (это же Recall) от False Positive Rate при варьировании порога решающей функции. Грубо говоря, она характеризует соотношение чувствительности и специфичности классификатора при различных настроечных порогах. AUC равна вероятности того, что классификатор присвоит более высокий “скор” случайному положительному образцу, чем случайному отрицательному. Значение AUC = 1 соответствует идеальному классификатору, AUC = 0.5 – случайному угадыванию. Преимущество ROC AUC в том, что она не зависит от выбранного порога разделения и даёт интегральную оценку качества модели по всем возможным уровням чувствительности.

Log-loss (логарифмическая функция потерь). Log-loss, также известная как кросс-энтропия, – это метрика, часто используемая при обучении классификаторов, выдающих вероятностные оценки принадлежности к классам. Она измеряет “наказание” за отклонение предсказанной вероятности от истинного класса. В простейшем случае бинарной классификации логарифмическая потеря определяется как:

$$\text{LogLoss} = -\frac{1}{N}\sum_{i=1}^{N}\Big[y_i \log p_i + (1 - y_i)\log(1 - p_i)\Big], \tag{1.6}$$

где p_i – предсказанная моделью вероятность положительного класса для объекта i, $y_i \in 0, 1$– истинная метка класса. Если модель абсолютно уверена и права ($p_i \to 1$ при $y_i = 1$ или $p_i \to 0$ при $y_i = 0$), log-loss стремится к 0 (идеальный случай). Если же модель уверена и ошибается ($p_i \to 1$ при $y_i = 0$ или наоборот), log-loss стремится к бесконечности, сильно штрафуя модель. Log loss позволяет оценивать не только “кого отнесли к какому классу”, но и качество калибровки вероятностных прогнозов модели. В многоклассовом случае log-loss обобщается как $-\sum_i \sum_{c=1}^{K} y_{i,c} \log p_{i,c}$, где $y_{i,c} = 1$ для правильного класса и 0 для остальных, а $p_{i,c}$ прогнозируемая вероятность класса c на объекте i.

Кроме перечисленных, в области компьютерного зрения применяются и специфичные метрики для некоторых задач.[18] Например, для оценки качества сегментации часто используют IoU (Intersection over Union) – меру перекрытия предсказанной области с истинной областью объекта; для детекции объектов помимо IoU важны показатели Precision/Recall при разных порогах IoU (мера mAP – mean Average Precision). В задаче ранжирования, как упоминалось, популярны DCG/NDCG, MAP (mean average precision) и др.

1.4 Нейронные сети

Современный прогресс компьютерного зрения в значительной степени обусловлен развитием нейронных сетей – моделей машинного обучения, вдохновленных архитектурой биологического мозга. Нейронные сети представляют собой композиции большого числа простых нелинейных преобразований (нейронов), которые обучаются на данных. В этой части главы рассматриваются основные виды нейросетевых моделей, применяемых в компьютерном зрении: от простейшего персептрона до глубинных сверточных сетей новейших архитектур.[15] Также обсуждаются алгоритмы обучения нейросетей и методы их регуляризации (раздел 1.4).

1.4.1 Персептрон, MLP

Персептрон – это один из первых и простейших типов искусственного нейрона, предложенный Ф. Розенблаттом в 1957–1958 гг.. Однослойный персептрон выполняет линейную классификацию: вычисляет взвешенную сумму входных признаков и применяет пороговую активационную функцию. Математически можно выразить выход персептрона как:

$$f(\mathbf{x}) = h(\mathbf{w} \cdot \mathbf{x} + b), \tag{1.7}$$

где $\mathbf{x} = (x_1, \ldots, x_m)$ – вектор входных признаков, $\mathbf{w} = (w_1, \ldots, w_m)$ вектор весовых коэффициентов, b – смещение (bias), а $h(z)$ – функция активации. В классическом персептроне в качестве h используется пороговая функция

Хевисайда, которая выдает 1, если аргумент положителен, и 0 (или -1) иначе. Таким образом, персептрон разделяет пространство объектов гиперплоскостью $\mathbf{w} \cdot \mathbf{x} + b = 0$. Если $\mathbf{w} \cdot \mathbf{x} + b > 0$, объект относится к классу “1”, иначе – к классу “0”. Геометрически это линейный классификатор.

Розенблатт показал, что его алгоритм обучения персептрона сходится за конечное число шагов, если обучающая выборка линейно разделима (теорема сходимости персептрона). Алгоритм обучения персептрона итеративно корректирует веса: для каждого обучающего примера вычисляется ошибка (разница между предсказанным и истинным классом), после чего веса обновляются пропорционально этой ошибке и значению признаков. В простейшем виде правило обновления можно записать как:

$$\mathbf{w}_{\text{new}} = \mathbf{w} + \eta \, (y - \hat{y}) \, \mathbf{x}, \tag{1.8}$$

где y – истинная метка (0 или 1), $\hat{y} = f(\mathbf{x})$ – предсказание персептрона, а η коэффициент скорости обучения. Это правило увеличивает веса на величину $\eta\mathbf{x}$, если объект положительный ($y = 1$) был ошибочно классифицирован как отрицательный ($\hat{y} = 0$), и наоборот уменьшает веса, если отрицательный объект ошибочно классифицирован как положительный. В случае нулевой ошибки веса не меняются.

Персептрон способен решать только линейно разделимые задачи. Классическим примером ограничения персептрона является проблема XOR: однослойный персептрон не может реализовать функцию “исключающее ИЛИ”, так как она не линейно разделима в пространстве входов. В 1969 г. Марвин Минский и Сеймур Пейпер в своей книге показали этот и ряд других недостатков персептрона, что на время охладило интерес к нейронным сетям. Тем не менее, персептрон ценен как фундаментальный строительный блок: в современных терминах это просто линейный классификатор с пороговой функцией активации, то есть прототип искусственного нейрона.

Ограниченность одиночного персептрона преодолевается путем объединения нескольких нейронов в сеть – получение многослойного персептрона. MLP 10 представляет собой полносвязанную нейронную сеть, состоящую из входного слоя, одного или нескольких скрытых слоев нейронов и выходного слоя. Каждый нейрон каждого слоя (кроме входного) вычисляет взвешенную сумму выходов предыдущего слоя и пропускает через нелинейную функцию активации (в современных сетях вместо пороговой часто используют сглаженные функции– сигмоиду, tanh или ReLU). За счет наличия скрытых слоев MLP способен моделировать нелинейные зависимости и решать значительно более сложные задачи, чем однослойный персептрон. Было доказано, что сеть с одним скрытым слоем, содержащим достаточно большое число нейронов с нелинейной активацией, является универсальным аппроксиматором – то есть может приблизить любую непрерывную функцию с произвольной точностью при подходящей настройке весов (теорема Цыбенко,

1989). Таким образом, многослойные сети в теории обладают огромной выразительной способностью.

Практическое обучение глубоких (многослойных) сетей стало возможным благодаря алгоритму обратного распространения ошибки (backpropagation), предложенному в работах Д. Румельхарта, Дж. Хинтона и Р. Уильямса в 1986 г. . Алгоритм backpropagation вычисляет градиенты функции потерь по всем весам сети, распространяя ошибку от выхода сети к ее входам, и затем использует метод градиентного спуска для корректировки весов. По сути, backpropagation является применением правила дифференцирования сложной функции (цепного правила) к нейронной сети. Благодаря ему стало возможным эффективно обучать сети с десятками и сотнями скрытых нейронов, что привело к "второму рождению" нейронных сетей в конце 1980-х.

Многослойный персептрон (часто называемый просто полносвязной нейронной сетью) широко применялся в 1990-х и 2000-х годах для задач классификации изображений, распознавания рукописного текста и других задач распознавания образов. Классический пример – распознавание цифр на MNIST: сеть с двумя скрытыми слоями по 300 и 100 нейронов достигала 98% точности, что было сопоставимо с лучшими на тот момент методами, основанными на вручную разработанных признаках.

Однако полносвязные MLP масштабируются плохо для изображений большого размера, так как число весов растет очень быстро с увеличением размерности входа (каждый нейрон следующего слоя связан со всеми выходами предыдущего). Для обработки изображений оказались более эффективны специализированные архитектуры – сверточные нейронные сети, о которых речь пойдет далее (раздел 1.3.2). Тем не менее, MLP до сих пор служат базовым элементом нейросетевых моделей: например, в современных сверточных сетях последние несколько слоев часто представляют собой именно полносвязные слои (перцептроны), сводящие распределенное представление признаков к окончательному решению классификатора .

Одно из ключевых отличий разных поколений нейронных сетей – выбор функции активации нейронов. В персептроне это жесткая пороговая функция; в MLP 80-х – гладкие сигмоидные функции (логистическая или tanh); в современных сетях почти повсеместно используется функция ReLU (Rectified Linear Unit) – простая несатурирующая нелинейность $\mathrm{ReLU}(z) = \max(0, z)$. ReLU облегчает распространение градиента и способствует быстрому обучению даже очень глубоких сетей. Также применяются ее модификации – Leaky ReLU, ELU и др. Эти усовершенствования, наряду с улучшенными методами оптимизации и регуляризации, позволяют успешно обучать сети, содержащие десятки и сотни слоев.В целом, переход от персептрона к многослойным нейронным сетям ознаменовал качественный скачок в способности алгоритмов машинного зрения автоматически извлекать иерархические признаки из данных. Если раньше для решения задач требовался ручной конструируемый набор признаков, то MLP при достаточном количестве

нейронов и данных способен сам научиться нужным преобразованиям входа. Это стало одной из причин повсеместного распространения нейронных сетей в компьютерном зрении.

1.4.2 Сверточные нейронные сети

Полносвязные нейронные сети, хотя и мощные, не учитывают специфики структурированных данных, таких как изображения. Изображение может содержать десятки тысяч пикселей, и соединять каждый нейрон со всеми пикселями – неэффективно и избыточно. Сверточные нейронные сети (Convolutional Neural Networks, CNN) были разработаны специально для работы с данными в виде сетки пикселей, учитывая локальные пространственные зависимости и обеспечивая относительную инвариантность к смещениям объектов на изображении.

Главная идея CNN – использование операции свертки и подвыборки (пулинга) вместо или наряду с полносвязными слоями. Свертка – это применение набора небольших обучаемых фильтров (ядер свертки) ко всему изображению: фильтр скользит по полю зрения и извлекает локальные признаки (например, горизонтальные/вертикальные градиенты, углы, текстуры). Каждый сверточный нейрон связан не со всеми нейронами предыдущего слоя, а только с небольшой окрестностью входов (например, 3×3 или 5×5 пикселей) и применяет один и тот же набор весов ко всем таким окрестностям изображения. Это резко снижает число параметров и вводит свойство пространственной локальности и разделения весов. Пулинг (чаще всего max-pooling) обеспечивает понижение разрешения карты признаков, агрегируя соседние значения (беря максимум из блока, например, 2×2) – это придаёт некоторую инвариантность к небольшим сдвигам и уменьшает размер представления, постепенно переходя от низкоуровневых признаков к более высокоуровневым.

Первой успешной архитектурой сверточной сети для задачи реального мира была LeNet-5, разработанная Янном Лекуном с коллегами в 1989–1998 гг. для распознавания рукописных цифр на чеках. LeNet-5 состояла из двух чередующихся сверточных и субдискретизирующих (пулинг) слоев, за которыми следовали три полносвязных слоя. Она эффективно распознавала цифры 0–9 на маленьких изображениях (32×32 пикселей), показав, что глубокое обучение применимо к визуальным данным. Успех LeNet-5 заложил основу для более сложных CNN, продемонстрировав потенциал глубокого обучения в обработке изображений.

Дальнейший прорыв произошел спустя примерно 15 лет с появлением AlexNet (2012). Архитектура AlexNet, предложенная Алексом Krizhevsky с соавторами, имела 5 сверточных слоев, за которыми шли 3 полносвязных слоя. Ключевые нововведения AlexNet: использование активации ReLU вместо сигмоид, применение слоя Dropout для борьбы с переобучением, и обучение на

GPU на базе огромного датасета ImageNet. AlexNet достигла рекордно низкой ошибки на ImageNet, опередив прежние алгоритмы с разрывом и тем самым подтвердив эффективность глубоких сверточных сетей в задаче классификации изображений. Это событие принято считать началом эры глубокого обучения в компьютерном зрении.

После AlexNet исследователи сфокусировались на создании более глубоких и точных CNN. Одной из таких была VGG (VGGNet), предложенная в 2014 году Симоняном и Зиссерманом. Архитектура VGG характеризовалась использованием простых сверточных слоев с очень небольшими ядрами 3×3, но в большом количестве. Модель VGG-16 содержала 16 весовых слоев (13 сверточных и 3 полносвязных), а VGG-19 – 19 слоев. Было показано, что увеличение глубины и последовательное применение маленьких фильтров могут улучшить качество классификации. VGGNet достигла высоких результатов (топ-5 ошибка 7.3% на ImageNet) и подтвердила тенденцию: более глубокие сети способны извлекать более сложные признаки, повышая точность распознавания.

Рост глубины сетей столкнулся с новой проблемой: деградация при обучении очень глубоких сетей. Было замечено, что при наращивании слоев после определенной глубины точность на тесте начинает ухудшаться – не из-за переобучения, а из-за затруднений оптимизации (градиент практически не проходит через многие слои). Решение предложила группа К. Хэ и др. в архитектуре ResNet (Residual Network), представленной в 2015 году. Идея ResNet – ввести пропускные связи (skip connections) через несколько слоев, которые пропускают сигнал напрямую, минуя пару-тройку слоев, реализуя т.н. остаточные блоки. Каждый такой блок учит не прямое преобразование $H(x)$, а отклонение (residual) от входа: $F(x) = H(x) - x$. Благодаря этому градиенты легче протекают сквозь сеть, и удается успешно обучать экстремально глубокие сети – 50, 101, 152 слоя и более. ResNet-50 в 2015 г. достиг 3.6% ошибки на ImageNet, установив новый рекорд. Резидуальные сети стали базовым компонентом множества последующих моделей и фактически стандартизировали очень глубокие CNN как основной инструмент компьютерного зрения.

Следующий виток развития связан не столько с увеличением глубины, сколько с повышением эффективности и автоматизацией проектирования сетей. В 2019 году в Google Research была предложена семейство EfficientNet (авторы М. Тан и К. Ле). EfficientNet использовала автоматический поиск архитектур (AutoML) для нахождения базовой структуры, а затем ввела концепцию компаундного масштабирования: одновременного пропорционального увеличения глубины, ширины (числа каналов) и разрешения входа по одному параметру масштабирования. Модель EfficientNet-B7 достигла рекордной точности 84.4% топ-1 на ImageNet, будучи при этом на порядок менее ресурсоемкой, чем альтернативы той же точности. Это показало, что тщательная оптимизация архитектуры

может значительно повысить соотношение “точность/сложность” модели. EfficientNet и похожие подходы сфокусированы на поиске оптимальных архитектурных гиперпараметров, что особенно важно при развёртывании моделей на устройствах с ограниченными ресурсами (мобильные телефоны, встроенные системы).[11]

Недавняя тенденция – сближение CNN с архитектурами трансформеров. В 2020 г. Vision Transformer (ViT) продемонстрировал, что можно достигать высоких результатов в зрении, вовсе отказавшись от сверток, заменив их механизмом самовнимания (self-attention). В ответ в 2022 г. была предложена архитектура ConvNeXt (авторы Z. Liu и др.), в которой классическая сверточная сеть ResNet модернизирована рядом приемов, заимствованных из трансформеров, таких как большое размерность канала, отсутствие пулинга между стадиями, упрощенные блоки и др.. ConvNeXt показала, что “традиционная” сверточная сеть может конкурировать с трансформерами по точности, если грамотно адаптировать дизайн. Она достигла сравнимой с ViT точности 87.8% на ImageNet при сходной вычислительной сложности, сохранив при этом привычный конвейер сверток. Появляются также гибридные архитектуры, сочетающие свертки и слои самовнимания (например, Swin Transformer использует сверточный сдвиг окон самовнимания). Таким образом, грань между чисто сверточными и трансформерными моделями в зрении постепенно размывается.

Итоги по архитектурам. На сегодняшний день семейство сверточных нейронных сетей включает множество архитектур, ключевые из которых можно расположить в хронологическом порядке их появления и вклада:

- LeNet-5 (1998): первая успешная CNN для распознавания рукописного текста, ввела концепции сверточных и pooling-слоев.
- AlexNet (2012): глубокая CNN, выигравшая ImageNet Challenge, применила ReLU, Dropout и масштабное обучение на GPU.
- VGG (2014): показала пользу очень глубоких сетей с простыми однотипными блоками (конволюции 3×3).
- GoogLeNet/Inception (2014): внедрила модуль Inception с несколькими параллельными сверточными путями, уменьшив число параметров (была упомянута вскользь, но это важная архитектура 2014 г.).
- ResNet (2015): ввела остаточные связи, позволив уверенно тренировать сети >100 слоев.
- MobileNet (2017): оптимизированная для мобильных разделяемая свертка (depthwise separable convolution) – уменьшение вычислений, потеряв незначительно в точности.

- EfficientNet (2019): автоматический подбор архитектуры и комплексное масштабирование для оптимального баланса точности и сложности.
- Vision Transformer (2020): первый трансформер для изображений, отказ от сверток в пользу самовнимания.
- ConvNeXt (2022): современная конволюционная сеть, соперничающая с трансформерами по качеству, за счет внедрения лучших практик дизайна

Каждая из этих моделей привнесла новые идеи, повышающие качество или эффективность. Сегодня исследователям доступны обширные библиотеки предварительно обученных моделей (например, на платформе PyTorch или TensorFlow), и выбор конкретной архитектуры диктуется требованиями задачи – наилучшее качество (тогда берут большой ViT или EfficientNet), ограниченные ресурсы (модели MobileNet, SqueezeNet), потребность в обработке видео (3D-CNN или Conv-LSTM) и т.д. Несмотря на появление альтернатив, сверточные нейронные сети остаются базовым инструментом для подавляющего большинства приложений компьютерного зрения благодаря своей эффективности и проверенной успешности на протяжении десятилетий исследований.

1.5 Алгоритмы обучения и регуляризация нейронных сетей

Обучение нейронных сетей – непростая вычислительная задача оптимизации, часто требующая специальных алгоритмических решений. Кроме того, большие нейронные сети склонны к переобучению, поэтому в практику вошел ряд методов регуляризации – предотвращения переобучения путем внесения определенных ограничений или шума в процесс обучения. В этом разделе рассматриваются основные алгоритмы обучения весов нейронных сетей (различные модификации стохастического градиентного спуска) и методы регуляризации (Dropout, нормализация, weigth decay, data augmentation, ранняя остановка и др.), а также техники настройки процесса обучения (адаптивные графики изменения скорости обучения и дообучение/тонкая настройка моделей).

1.5.1 SGD и его производные

Стохастический градиентный спуск (SGD). Базовым методом обучения нейросетей является алгоритм градиентного спуска – итеративное обновление весовых коэффициентов сети в направлении, противоположном градиенту функции потерь. Поскольку набор данных может быть очень большим, на практике используется стохастический градиентный спуск: на каждом шаге

градиент оценивается не на всей выборке, а на случайном мини-батче из нескольких примеров. Это значительно ускоряет один шаг обновления и вводит шум в процесс, помогающий преодолевать локальные минимумы. Шаг обновления для каждого веса w имеет вид:

$$w := w - \eta \frac{\partial L}{\partial w}, \tag{1.9}$$

где η – коэффициент скорости обучения (learning rate), а $\frac{\partial L}{\partial w}$ – градиент функции потерь по w, вычисленный на текущем мини-батче. Несмотря на простоту, SGD с правильно настроенным η и достаточно большим числом эпох способен успешно обучать большие нейронные сети. В практической реализации обычно выбирается размер мини-батча (batch size) в диапазоне от 32 до нескольких сотен, и на каждой эпохе (проходе по всему датасету) веса обновляются многократно по батчам.

Метод Momentum (импульс). Один из первых усовершенствованных методов, призванных ускорить сходимость SGD и сгладить колебания, – это метод градиентного спуска с импульсом. Идея импульса заимствована из физики: ввести “накопленную скорость” изменения весов. В методе Momentum вводится вспомогательная переменная $\mathbf{v}$ – скорость. На каждой итерации скорость обновляется с учетом текущего градиента и предыдущей скорости:

$$\mathbf{v} := \mu \mathbf{v} - \eta \nabla L(\mathbf{w}), \tag{1.10}$$

$$\mathbf{w} := \mathbf{w} + \mathbf{v}, \tag{1.11}$$

где μ – коэффициент импульса (обычно 0.9). Таким образом, градиенты прошлых шагов накапливаются (с затуханием) в $\mathbf{v}$. Это помогает сглаживать хаотичные колебания градиента в перпендикулярных направлениях и ускоряет движение вдоль устойчивого направления снижения ошибки. Метод Momentum особенно эффективен на задачах с долинно-овражным ландшафтом ошибки, позволяя “проскочить” через плато быстрей. Визуально он действует как сглаженный фильтр на последовательность градиентов, разгоняя обновления в направлении последовательных одинаковых градиентов.

Адаптивные методы: AdaGrad, RMSProp, Adam. Дальнейшая линия развития алгоритмов оптимизации – адаптация шага обучения для разных параметров и на разных этапах. AdaGrad (2011) предложил уменьшать скорость обучения для каждого параметра пропорционально корню суммы квадратов градиентов по этому параметру с начала обучения. Это приводит к большим изменениям редко информативных параметров и малым – часто подстраивавшихся, но вызывало чрезмерное затухание η со временем. RMSProp (Hinton, 2012) модифицировал идею: нормировать шаг не на суммарный, а на экспоненциально сглаженный по времени квадрат градиента. В результате каждый вес получает индивидуальный темп обновления, зависящий от вариабельности его градиента.

Наиболее популярный на сегодня алгоритм – Adam (Adaptive Moment Estimation), предложенный Д. Кингмой и Дж. Ба (2014) . Adam сочетает идеи Momentum и RMSProp: он ведет экспоненциально затухающие оценки первого момента градиента (аналог импульса, **m**) и второго момента (квадрата градиента, **v**), а затем вычисляет шаг как отношение этих оценок:

$$\mathbf{m} := \beta_1 \mathbf{m} + (1 - \beta_1) \nabla L(\mathbf{w}), \tag{1.12}$$

$$\mathbf{v} := \beta_2 \mathbf{v} + (1 - \beta_2) (\nabla L(\mathbf{w}))^2, \tag{1.13}$$

$$\mathbf{w} := \mathbf{w} - \eta \frac{\mathbf{m}/(1 - \beta_1^t)}{\sqrt{\mathbf{v}/(1 - \beta_2^t)} + \epsilon}, \tag{1.14}$$

где $\beta_1 \approx 0.9, \beta_2 \approx 0.999$ – параметры затухания моментов, ϵ – малое число для избежания деления на ноль, t – номер шага . Такой расчет эквивалентен тому, что для каждого веса поддерживается адаптивная скорость обучения, масштабируемая по обратной пропорции к среднеквадратичному значению градиента (как в RMSProp), но также учитывается “направленный импульс” градиента (как в Momentum)[12]. Adam хорошо работает “из коробки” для большинства задач, требуя минимальной настройки гиперпараметров. Благодаря адаптивности он эффективен при работе с разреженными признаками и при негладких оптимизационных поверхностях . Сегодня Adam – де-факто стандарт для обучения глубоких сетей.[19]

При выборе алгоритма оптимизации практики обычно начинают с простого – SGD+Momentum или Adam. SGD с импульсом иногда предпочтительнее для задач, где нужен лучший контроль обобщения (некоторые исследователи отмечали, что Adam может сильнее переобучиваться на небольших выборках). Adam быстрее достигает низкой ошибки на обучении, поэтому часто используется при обучении с нуля на больших датасетах. Новые оптимизаторы (RAdam, Lookahead, Lion и др.) могут дать выигрыш в специфических сценариях, но их гиперпараметры требуют тщательной настройки. В итоге, понимание свойств каждого метода позволяет инженеру машинного обучения подбирать оптимизатор под конкретную задачу и этап обучения (например, сперва Adam для быстрого спуска, затем классический SGD для тонкого доводки модели).

1.5.2 Dropout, batch/instance/layer normalization, weight decay

Большие нейронные сети обладают миллионами параметров и экстремальной аппроксимирующей мощностью, из-за чего есть риск переобучения – модель начинает подгоняться под случайный шум обучающих данных вместо того, чтобы выучивать общие закономерности. Для борьбы с переобучением разработан ряд методов регуляризации – приемов,

ограничивающих избыточную гибкость модели или добавляющих шум при обучении, чтобы сделать модель более устойчивой.

Dropout. Одним из самых популярных методов регуляризации нейросетей является Dropout (произвольно можно перевести как “выключение нейронов”). Введенный Хинтоном и соавторами (2012), метод заключается в следующем: во время обучения на каждой итерации случайно “выключать” (занулять) выходы каждого нейрона скрытого слоя с некоторой фиксированной вероятностью p (например, $p = 0.5$) . При этом в следующем слое обучение идёт на неполной сети. Каждый мини-батч фактически тренирует немного отличающуюся архитектуру сети. На этапе тестирования (и при финальном использовании модели) все нейроны работают, но их выходы масштабируются на коэффициент $(1 - p)$ для учёта отключений на этапе обучения. Идея заключается в предотвращении соадаптации нейронов – ситуации, когда некоторые нейроны “излишне подстраиваются” друг под друга, формируя сложные узкие взаимозависимости . Dropout мешает конкретным сочетаниям нейронов постоянно совместно активироваться, так как случайно отключает нейроны и заставляет сеть быть устойчивой к таким пропаданиям. Это аналогично усреднению экспоненциально многих архитектур (теоретически сеть с Dropout можно рассматривать как ансамбль 2^n урезанных подсетей, где n – число выключаемых нейронов) , но в практической реализации заменено эффективным стохастическим обучением одной сети.

Dropout значительно снижает переобучение и обычно улучшает обобщающую способность сети . Первоначально его применяли в основном к полносвязным слоям, но затем распространили и на сверточные (хотя для сверточных часто более эффективны другие регуляризаторы). Стандартное значение $p = 0.5$ для скрытых слоев и $p = 0.2$ для входного слоя (если используется). Dropout простой в реализации и очень действенный метод, поэтому практически по умолчанию включается во многие архитектуры (например, в AlexNet он уже использовался на полносвязных слоях).

Нормализация (Batch Normalization и др.). Другой важнейший инструмент – нормализация активаций. Batch Normalization (BN), предложенная С. Иоффе и К. Шегеди (2015), решает проблему внутреннего смещения распределений (internal covariate shift) в глубоких сетях . Суть BN: для каждого нейрона в промежуточном слое нормировать распределение его выходов по мини-батчу к нулевому среднему и единичной дисперсии, а затем применить индивидуально обучаемые сдвиг и масштаб (чтобы восстановить нужную произвольную среднюю и разброс, если это полезно):

$$\hat{u} = \frac{u - \mathbb{E}\text{batch}[u]}{\sqrt{\text{Var}}}[u] + \epsilon, \tag{1.15}$$

$$y = \gamma\hat{u} + \beta, \tag{1.16}$$

где u – активация до BN, $\hat{u}$ – нормализованная активация, y – выход BN, параметры γ, β обучаются, а статистики $\mathbb{E}$ и Var считаются по текущему

батчу . BatchNorm выравнивает распределения входящих сигналов для каждого слоя, что позволяет использовать более высокие learning rate и ускоряет сходимость обучения . Кроме того, оказалось, что BN сам по себе действует как регуляризатор: добавление шума оценки среднего и дисперсии по батчу вводит некоторую стохастичность, схожую с Dropout, что иногда даже позволяет отказаться от Dropout вовсе (в оригинальной статье было отмечено, что BN в определенных случаях снижает потребность в Dropout).

Batch Normalization стала стандартным компонентом современных CNN: она обычно вставляется после линейного преобразования (свертки или полносвязного слоя) и перед активацией. BN значительно стабилизирует обучение очень глубоких сетей, позволяя строить трансформации, в которых каждый слой получает “более привычные” данные . Однако BN имеет недостатки: зависимость от размера батча (на очень маленьких батчах оценка среднего/разброса становится неточной) и некоторое различие поведения при обучении и тесте (при тестировании используются скользящие статистики по обучению).

Вслед за BN появились другие виды нормализации: Layer Normalization – нормализация не по батчу, а по нейронам слоя для каждого примера (применяется в трансформерах); Instance Normalization нормализация отдельно по каждому примеру и каждому каналу (используется в задачах генерации изображений, стилизации); Group Normalization – обобщение, когда нейроны разбиваются на группы и нормируются внутри групп. Instance Norm особенно эффективна в стилевых задачах (убирает контрастность и освещенность у каждого изображения, сохраняя только структурные признаки), а Layer Norm – в рекуррентных сетях и трансформерах, где размер батча может быть 1.

Все виды нормализации служат двум целям: 1) стабилизировать распределения активаций, 2) вносить легкий шум (в случае пакетных методов), что улучшает обобщение. Правильная нормализация облегчает обучение очень глубоких сетей – например, трансформеры практически не обучаются без LayerNorm.

Weight Decay (L2-регуляризация). Ещё один классический метод регуляризации – снижение весов (weight decay), он же L2-регуляризация. Суть: добавляется штраф к функции потерь, равный сумме квадратов весов сети:

$$L_{\text{reg}} = L_{\text{data}} + \lambda \sum_i w_i^2, \tag{1.17}$$

где λ – коэффициент регуляризации. Это побуждает алгоритм искать решения с меньшей нормой весов. Веса с малой величиной менее склонны к подгонке под шум обучающих данных (так как модель становится более гладкой). В реализации SGD weight decay эквивалентен умножению весов на коэффициент $(1 - \eta\lambda)$ на каждом шаге – то есть постепенному “распаду” весов к нулю при отсутствии градиента от данных.

Weight decay исторически широко использовался, особенно в более старых моделях. В современных фреймворках (таких как PyTorch) параметр weight decay часто можно указать прямо в оптимизаторе (это реализует L2-штраф для всех весов, кроме параметров смещения и нормализации). Хотя L2 регуляризация не привносит новых свойств, как Dropout или BN, она помогает держать веса в разумных пределах, что обычно улучшает обобщение. Следует заметить, что для оптимизатора Adam стандартная реализация weight decay должна быть слегка скорректирована (AdamW), иначе эффект смазывается адаптивным масштабированием градиентов.

Регуляризация – необходимый элемент при обучении больших моделей на ограниченных данных. Современные модели часто сильно переобучаются без таких техник. На практике обычно сочетается несколько методов: например, в сверточной сети могут одновременно применяться BatchNorm, Dropout (в полносвязных слоях), weight decay для оптимизатора и аугментации данных – все они в совокупности дают эффект в снижении переобучения и улучшении обобщающей способности модели.

1.6 Выводы

1. Изучены основные задачи компьютерного зрения.
2. Рассмотрены основные понятия обучения с учителем.
3. Исследованы методы, использующиеся для решения оптимизационных задач, и особенности их применения к задаче обучения нейронных сетей
4. Рассмотрены нейронные сети прямого распространения, как простейший вариант нейронных сетей, а также сверточные сети, как наиболее используемый тип сетей для задач компьютерного зрения.

ГЛАВА 2
СПЕЦИФИКА ЗАДАЧИ РАСПОЗНАВАНИЯ ЛИЦ

2.1 Постановка задачи

Распознавание лиц – это класс задач компьютерного зрения, направленных на идентификацию людей по их изображениям. В общем случае можно выделить три варианта постановки задачи распознавания лиц:

- Верификация (1-к-1) – проверка, принадлежит ли два предоставленных изображения одному и тому же человеку (задача «та ли это личность?»). Здесь система сопоставляет пару снимков и выносит решение о соответствии (match / no match). Например, верификация используется при разблокировке смартфона по лицу или при проверке паспорта, где лицо владельца сравнивается с фотографией в документе.

- Идентификация (1-к-N) – установление личности на фотографии путем поиска соответствия в базе известных лиц. В этом режиме один входной снимок сравнивается со множеством шаблонов в базе данных; результатом является идентификатор личности из базы (либо сообщение, что лицо не найдено). Идентификация применяется, например, в системах видеонаблюдения для поиска подозреваемых по камерам или при автоматическом тегировании фотографий в социальных сетях.

- Кластеризация – группировка множества несортированных фотографий по признаку принадлежности к одной персоне. В отличие от идентификации, здесь отсутствует заранее размеченная база; система должна по схожести лиц разбить фотографии на кластеры, соответствующие разным людям. Такой режим востребован при организации фотоархивов (группировка изображений одного человека без ручной разметки) и в исследовательских задачах, где нужно выявлять повторяющиеся лица в больших массивах данных.

Важно подчеркнуть, что верификация и идентификация решают несколько разные задачи. Верификация – это бинарная классификация пары изображений (совпадение/несовпадение), тогда как идентификация – многоклассовая классификация с открытым множеством классов (каждый известный человек – класс, плюс опция «неизвестный» в случае открытой идентификации). Тем не менее, современные подходы обычно стремятся к универсальному решению, способному поддерживать оба режима. Например, модель FaceNet от Google обучает нейросеть, которая проецирует фотографии лиц в компактное евклидово пространство признаков; в этом пространстве расстояния отражают

степень сходства лиц . После такого преобразования одна и та же модель легко применяется и для 1-к-1, и для 1-к-N, и для кластеризации – достаточно сравнивать эмбеддинги (векторы признаков) стандартными методами кластерного анализа . Иными словами, имея векторное представление лица, верификацию, идентификацию и кластеризацию можно выполнять единообразно путем вычисления расстояний либо схожести между векторами [17].

При решении задачи распознавания лиц на практике важны два основных показателя: точность распознавания и скорость работы системы. Требования к ним существенно зависят от прикладного сценария, в котором используется технология распознавания лиц. Рассмотрим три распространенных сценария и соответствующие требования:

2.1.1 Верификация, идентификация, кластеризация

Верификация лиц (face verification) – это частный случай биометрической аутентификации, когда системе предъявляются два изображения (например, фотография с камеры и эталонное изображение в базе) и требуется ответить, изображен ли на них один и тот же человек. Для верификации критически важны низкие показатели ошибок первого и второго рода – то есть минимизация как ложных совпадений (False Match Rate, FMR), так и ложных отказов в обслуживании (False Non-Match Rate, FNMR). В высоконадежных системах (например, доступ в защищенные помещения) допустимы очень строгие пороги, дающие FMR порядка 10^{-6} и ниже, чтобы вероятность ошибочного допуска чужого лица была пренебрежимо малой. Согласно методикам NIST, обычно порог верификации калибруется именно по требуемому FMR; например, рассчитывают FNMR при FMR = 1e-6, чтобы оценить надежность алгоритма . Верификация, как правило, производится в режиме реального времени и предполагает сравнительно небольшое число проверок (например, одно сравнение с данными владельца устройства). Системы верификации по лицу уже достигли человеческого уровня производительности на популярных наборах данных. Так, еще в 2015 году модель FaceNet сообщила точность 99,63% на наборе LFW[13] , что превзошло уровень человека на той же задаче. Современные же алгоритмы (ArcFace, CosFace и др.) почти вплотную приблизились к 100% на LFW и аналогичных тестах, то есть вероятность ошибки в контролируемых условиях составляет доли процента или меньше.

Идентификация лиц (face identification) – более сложная задача, в которой система должна найти совпадение для данного лица среди многих зарегистрированных в базе. Здесь ключевым показателем является емкость и точность 1-к-N поиска. В закрытой постановке (closed-set identification) предполагается, что разыскиваемое лицо гарантированно присутствует в базе; метрикой качества служит процент правильных идентификаций (Top-1

accuracy или Rank-1 rate). В открытой постановке (open-set) алгоритм должен помимо идентификации уметь выдавать решение «лицо не найдено», если человека нет в базе; тогда качество характеризуют двумя кривыми ошибок: False Positive Identification Rate (неправильное распознавание чужого как кого-то из базы) и False Negative Identification Rate (неопознание зарегистрированного лица). Идентификация предъявляет высокие требования к масштабируемости алгоритма, так как нужно сравнить один образец со всей базой (число сравнений N может быть очень велико, до миллионов). Для ускорения часто используют специальные методы поиска по векторным базам (например, индексы типа FAISS для быстрого k-NN поиска в пространстве эмбеддингов). Тем не менее, в ряде приложений реальное время не критично – например, анализ видеозаписей задним числом может выполняться офлайн с использованием мощных серверов. Точность идентификации непрерывно растет благодаря улучшению архитектур сетей и обучающих выборок: по данным отчета NIST, лучший алгоритм идентификации по состоянию на 2020 год имел ошибку всего 0,08% против 4% у лидера 2014 года , что свидетельствует о колоссальном прогрессе. Таким образом, современные системы способны надежно искать лицо среди базы из миллионов с минимальным числом ошибок.

Кластеризация лиц – вспомогательная задача, часто применяемая для автоматической разметки или сжатия данных. Здесь на вход подается множество фотографий неизвестных лиц, и алгоритм группирует их по персоналиям, не зная заранее, кто именно изображен. От алгоритма требуется высокая устойчивость метрического пространства: фотографии одного человека (несмотря на различия в позах, освещении и проч.) должны сгруппироваться вместе, а разные люди – раздельно. Оценка качества кластеризации может выполняться метриками вроде Purity (чистота кластеров) или F-score на парных совпадениях. В практических приложениях хорошая кластеризация облегчает работу пользователя – например, автоматически группирует все фотографии одного человека в телефоне, даже если его нет в списке контактов. Современные модели (например, упомянутый FaceNet) позволяют выполнять кластеризацию именно на основе сравнения эмбеддингов лиц: после проекции в пространстве признаков алгоритмы вроде DBSCAN или иерархической кластеризации могут объединять близкие векторы, соответствующие одному индивиду . В работе FaceNet продемонстрировано, что единое представление может эффективно решать и задачу кластеризации: авторам удалось разделить в неразмеченной коллекции фотографий миллионы лиц по идентичностям, используя стандартные методы кластерного анализа поверх эмбеддингов .

2.1.2 Требования к точности и скорости в прикладных сценариях (мобильные, охранные, UX)

Как отмечалось, требования к системам распознавания лиц сильно зависят от контекста их использования. Рассмотрим три характерных сценария: (a) мобильное или встроенное устройство ограниченной мощности, (b) стационарная система безопасности/видеонаблюдения, (c) пользовательские приложения с высоким требованием удобства (UX).

(a) Мобильные устройства. Распознавание лиц все чаще используется на смартфонах, планшетах, смарт-очках и других портативных устройствах – например, для разблокировки телефона («Face ID»), авторизации платежей или сортировки фотографий в галерее. В таких условиях основной упор делается на эффективность и легковесность алгоритмов, чтобы они работали быстро и не разряжали батарею. Мобильные процессоры обладают ограниченной вычислительной мощностью, поэтому архитектуры нейросетей приходится упрощать и оптимизировать. В последние годы появилось направление мобильных моделей для распознавания лиц – например, MobileFaceNets от Sheng Chen et al. (2018) содержит менее 1 млн параметров и занимает 4 МБ, достигая при этом точности 99,55% на LFW . Эта модель специально спроектирована для смартфонов и обеспечивает высокую скорость (18 мс на изображение на мобильном CPU) при незначительном снижении точности по сравнению с тяжелыми моделями . Другой пример – FaceID в iPhone, где задействован специализированный чип Neural Engine и инфракрасный проектор; Apple заявляет вероятность ложного срабатывания около 1 на миллион, то есть система очень надежна, хотя и работает локально на устройстве. В целом для мобильных сценариев характерны требования: время ответа не более сотен миллисекунд, минимальный размер модели (несколько мегабайт) и возможность работы без подключения к серверу. Некоторое снижение точности допустимо, если попытки распознавания можно повторить (пользователь просто попробует снова в случае неудачи), однако частые ошибки все же сильно ухудшают UX. Поэтому исследователи стремятся найти компромисс: используя облегченные архитектуры (MobileNet, ShuffleNet, Quantization) и специальные потери при обучении, достичь максимальной точности, но в рамках ограничений по памяти и вычислениям [4] .

(b) Системы безопасности и видеонаблюдения. В охранных приложениях (контроль доступа, поиск преступников, наблюдение за обстановкой) на первом месте стоит точность распознавания, особенно низкий уровень ложных пропусков и ложных срабатываний. Здесь цена ошибки высока: непропуск своего сотрудника через турникет – это неудобство, но ошибочный допуск постороннего – критический сбой. Поэтому пороги принятия решения обычно выбираются очень строгие, а модели – наиболее мощные из доступных. Часто применяются эмбеддинги повышенной размерности (512 и более), ансамбли нескольких нейросетей, многокамерные системы. Время

ответа тоже важно (система видеонаблюдения должна в реальном времени сигнализировать о выявлении разыскиваемого лица), но в стационарных условиях можно использовать производительные серверы или облачные вычисления, поэтому допустимо выполнить более тяжелые алгоритмы. Современные топовые модели (ArcFace, CurricularFace, AdaFace и др.) демонстрируют поразительные результаты именно в таких строгих метриках. К примеру, связка RetinaFace+ArcFace на наборе IJB-C (очень сложные условия: разные позы, ракурсы) достигает доли истинных совпадений 89,59% при уровне ложных совпадений 1e-6 – это означает, что при одном ошибочном срабатывании на миллион сравниваемых пар система всё ещё правильно опознает 90% лиц. Такие показатели открывают возможность использования распознавания лиц даже в сценариях пограничного контроля и других высокобезопасных системах. Кроме того, в видеонаблюдении возникают свои специфические требования: алгоритм должен устойчиво работать с низкокачественным видео (размытость, низкое разрешение, ИК-подсветка ночью), а также уметь обрабатывать потоковые данные (десятки кадров в секунду с множества камер). Поэтому помимо точности классификации лиц важна скорость обработки видеопотока и встроенная детекция (чтобы выдать тревогу сразу при появлении лица в кадре). Обычно баланс достигается комбинацией: детектирование всех лиц на каждом кадре (с облегченной моделью), отслеживание их треков, и периодическое применение более мощной модели распознавания на лучших кадрах. Аппаратные средства (GPU в серверах, специализированные модули) позволяют обрабатывать десятки миллионов сравнений в секунду, поэтому даже базы данных размером 10^7–10^8 лиц становятся реальностью для идентификации в крупных государствах.

(c) Пользовательские приложения (UX-нагрузка). Сюда относятся случаи, когда распознавание лиц встроено в потребительские сервисы: разблокировка гаджетов, фильтры дополненной реальности (определяют положение лица для наложения масок), автофокусировка камеры на лице, сортировка галереи по лицам, авторизация в приложениях по селфи и пр. Критически важным становится удобство и скорость для пользователя. Даже небольшая задержка (более 0,5–1 секунды) при распознавании заметно ухудшает впечатление. Пользователь также не должен совершать сложных действий – система должна быть устойчивой к естественным позам и мимике. В таких приложениях часто используется комбинация аппаратных и программных оптимизаций: выделенные нейромодули для вычислений, предварительное распознавание на уровне ОС (например, Android Face Unlock предоставляет API, которое разработчики интегрируют в свои приложения), а также постоянно обновляемые данные о пользователе (адаптация модели под новые снимки владельца, сбор дополнительной галереи для улучшения качества). Требования к точности зависят от конкретного случая использования: для разблокировки телефона ошибка «не узнал хозяина» просто заставит повторить попытку (что приемлемо, если редко), а вот ложное разблокирование чужим строго

недопустимо. Поэтому некоторые устройства дополняют распознавание лица другими факторами (ввод PIN-кода после нескольких неудачных попыток, требование открыть глаза – защита от использования фотографии, и т.д.). В целом, акцент в UX приложениях – на быстроту, устойчивость в реальных условиях (разное освещение, головные уборы) и плавную интеграцию в опыт пользователя. Разработчики стремятся к тому, чтобы пользователь вообще не задумывался о работе алгоритма – всё происходило мгновенно и незаметно. Для этого могут жертвовать частью точности (например, не стремятся опознать каждого на групповом фото безошибочно), но ключевые случаи (как разблокировка) доводят до максимальной надежности сочетанием алгоритмов и сенсоров.

В заключение раздела отметим, что специфика задачи распознавания лиц требует баланса между точностью (биометрической надежностью) и эффективностью. Продвинутые модели могут быть настроены под разные сценарии: одна и та же архитектура ResNet100 с функцией потерь ArcFace может быть урезана до MobileFaceNet для смартфона или, напротив, запущена в ансамбле на сервере для видеоаналитики. В дальнейшем изложении мы рассмотрим технический конвейер распознавания лиц и современные подходы на каждом его этапе.

2.2 Конвейер распознавания

Современные системы распознавания лиц обычно строятся как конвейер из нескольких этапов (рис. 2.1). Сначала из сырого изображения выделяются области с лицами (детекция), затем лица нормализуются геометрически (выравнивание по положению), после чего из каждого нормализованного лица извлекается вектор признаков фиксированной размерности (face embedding). На последнем этапе выполняется собственно распознавание – сравнение полученных эмбеддингов с эталонами и принятие решения (верификация или идентификация). Такая модульная структура исторически сложилась еще в классических подходах: отдельно решались задачи «где лицо?» и «чье лицо?». В глубоком обучении этапы часто алгоритмически разделены, однако возможны end-to-end решения (например, одношаговые детекторы с распознаванием). Ниже мы подробно разберем каждый этап этого конвейера, а также приведем актуальные алгоритмы, применяемые на практике, с техническими деталями и сравнительным анализом.

2.2.1 Детекция лиц (MTCNN, RetinaFace, YOLOv5-Face)

Детекция лиц – это нахождение на изображении всех областей, содержащих лица, и выделение их границ. Детектор обычно выдает для каждого найденного лица прямоугольную рамку (bounding box) и, возможно, дополнительные

данные – оценку уверенности, координаты ключевых точек лица (глаз, нос и т.п.). Качество детекции определяется показателями полноты (recall, сколько лиц из присутствующих найдено) и точности локализации (precision, доля правильных областей среди выданных). Пропуск лица на этом этапе делает последующее распознавание невозможным, поэтому чувствительность детектора крайне важна. В то же время, детектор должен избегать множества ложных срабатываний на фоновые объекты, чтобы не перегружать систему далее.

За десятилетия разработано множество алгоритмов детекции лиц. Ранние подходы основывались на каскадах ручных признаков (например, детектор Виолы-Джонса, 2001, использовал простые Haar признаки и каскад бустинговых классификаторов). Однако в сложных условиях («in the wild») классические методы сталкивались с трудностями – вариации позы, освещения, выражений лица приводят к пропускам. С середины 2010-х детекция лиц была революционизирована глубоким обучением. Появились специальные CNN-архитектуры, превосходящие классические по точности на порядок. Рассмотрим три популярных современных подхода: MTCNN, RetinaFace и семейство YOLO Face – они отражают эволюцию детекторов лиц.

MTCNN

MTCNN (Multi-Task Cascaded Convolutional Networks), 2016 – одна из первых успешных глубоких моделей детекции лица . MTCNN представляет собой каскад из трех небольших нейросетей, последовательно уточняющих положение лица. Первая сеть (P-Net) быстро просматривает изображение и находит грубые кандидаты областей лица. Вторая (R-Net) отбраковывает ложные срабатывания и уточняет рамки. Третья (O-Net) финально регрессирует точные координаты рамки и одновременно предсказывает координаты 5 ключевых точек лица (глаза, нос, уголки рта). MTCNN примечательна тем, что совмещает две задачи – детекцию и предсказание landmark-точек. За счет многоцелевой (multi-task) обучения удается улучшить качество: обнаружение лиц и их выравнивание помогают друг другу (корреляция задач)[20] . Авторы сообщают, что MTCNN достигла наивысшей точности на конкурсных наборах FDDB и WIDER FACE по состоянию на 2016 год, оставаясь при этом достаточно быстрой для работы в реальном времени . MTCNN до сих пор часто используется в приложениях благодаря простоте и компактности (весь каскад 2-3 МБ). Однако у нее есть и недостатки: каскадная схема означает, что время работы зависит от числа кандидатов; при большом количестве лиц на изображении MTCNN может быть медленной. Кроме того, она склонна пропускать очень мелкие лица и давать ложные срабатывания на сложном фоне по сравнению с более современными моделями.

RetinaFace

RetinaFace, 2019/2020 – современный одноэтапный детектор лиц, продолжающий идеи семейства RetinaNet для объектов. RetinaFace разработана Jiankang Deng и коллегами (группа InsightFace) . Ключевые особенности RetinaFace:

- Одношаговый детектор (single-stage): в отличие от MTCNN, RetinaFace выполняет все за один прогон нейросети. Она прогнозирует плотную карту лиц на изображении (pixel-wise face localization) на нескольких масштабах сразу . Это достигается с помощью Feature Pyramid Network (FPN): карты активаций разных уровней используются для обнаружения лиц разных размеров.

- Мультизадачность: сеть одновременно предсказывает 1) координаты рамки лица, 2) вероятность принадлежности объекта к классу «лицо», 3) 5 ключевых точек лица (как и MTCNN), а дополнительно 4) плотную 3D-сетку лица из 1000+ вершин (self-supervised mesh decoder)[5] . Последний компонент – новшество: сеть учится восстанавливать грубую 3D форму лица, что улучшает понимание позы и позволяет лучше обнаруживать повернутые лица. Хотя 3D-меш напрямую не нужен для распознавания, его наличие в обучении служит самоконтролем, повышая устойчивость детектора.

- Высокая точность на сложных наборах: на наборе WIDER FACE (поднабор Hard) RetinaFace достигла рекордного среднего precision 91,4%, превзойдя предшедущие решения на 1,1% . Это огромный прогресс, учитывая, что WIDER FACE Hard содержит лица размером в несколько пикселей, сильные ракурсы и окклюзии. RetinaFace почти не пропускает даже мелкие лица. Также важно, что использование RetinaFace для предварительной детекции/выравнивания улучшает последующее распознавание: авторы показали рост точности верификации ArcFace на сложном наборе IJB-C до TAR=89,59%@FAR=1e-6 [5] , просто заменив детектор на RetinaFace вместо более простого.

- Производительность: хотя RetinaFace – довольно тяжелая модель (изначально на базе ResNet-50 или -152), она может работать в реальном времени. В работе показано, что с облегченным бэкбоном (ResNet-Qt или MobileNet) RetinaFace достигает обработки 20+ FPS на CPU для VGA-видео.[10] Это делает ее применимой и в видеоаналитике на edge-устройствах.

Благодаря этим свойствам RetinaFace сегодня рассматривается как один из лучших детекторов лиц. Он стал де-факто стандартом для высокоточной

детекции: например, библиотека InsightFace использует RetinaFace для вырезания лиц перед распознаванием. Недостатком можно назвать лишь относительную сложность: модель требует тщательной настройки anchor-ов, пирамидальных уровней, а наличие 3D-выхода усложняет обучение. Тем не менее, при наличии вычислительных ресурсов RetinaFace обеспечивает превосходное сочетание полноты и точности обнаружения.

YOLOv5-Face

YOLOv5-Face и другие одноэтапные детекторы общего назначения. Семейство моделей YOLO (You Only Look Once) изначально разработано для детекции объектов в реальном времени (авт. Joseph Redmon и др., 2016)[14] . Идея YOLO – единая нейросеть напрямую предсказывает ограничивающие рамки и классы объектов на изображении, без каскадов и предложений. За счет этого удается достичь очень высокой скорости (сотни FPS на GPU) при приемлемой точности. Применительно к лицам появилось несколько адаптаций YOLO: изначально можно обучить модель YOLO на датасете лиц (например, на WIDER FACE) – так появились версии YOLO-Face. Преимущества YOLO-подходов для детекции лиц:

- Максимальная скорость. YOLOv5-Face с небольшим размером (модель Nano/Tiny) может работать на мобильных устройствах в реальном времени, обрабатывая 30–60 FPS на CPU. На GPU более крупные версии (YOLOv5m-face) дают 100+ FPS. Это важно для приложений типа подсчета количества людей в кадре, слежения за вниманием водителя и т.д.

- Простота развертывания. Одношаговые модели легко интегрировать – одна нейросеть, один запуск. Фреймворки типа OpenVINO, TensorRT хорошо оптимизируют YOLO. Также доступно много предобученных весов.

- Достаточная точность. Хотя специализированные детекторы (RetinaFace) показывают более высокие AP, хорошо обученный YOLO также способен детектировать лица достаточно надежно. К примеру, модификация на базе YOLOv5 показала AP 88% на WIDER FACE Hard, что близко к лидерам, но при значительно лучшей скорости.

- Гибкость. YOLO можно дообучить под конкретные условия (например, лица в профиль, лица в шлемах и масках) добавлением соответствующих данных.

Следует подчеркнуть, что качество детектора лица напрямую влияет на итоговую точность распознавания. Пропущенное лицо не будет распознано

вовсе, а неточная рамка или сильный наклон головы усложнят извлечение признаков. Поэтому в высокоточных системах (например, верификация в банкоматах) часто используют самые совершенные детекторы и тратят на них большую часть времени обработки. Напротив, в реальном времени на мобильных устройствах иногда используют упрощенные детекторы (или даже алгоритмы на основе классических каскадов для экономии ресурсов), жертвуя частью качества ради скорости.

2.2.2 Выравнивание (5-point/106-point landmarks, ArcFace alignment)

После обнаружения лица на изображении следующий шаг конвейера – геометрическое выравнивание лица (face alignment). Цель – привести вырезанное лицо к каноническому виду: выровнять по горизонтали глаза, нормализовать масштаб (размер лица в кадре) и положение. Выравнивание устраняет вариации позы головы (повороты, наклоны) и тем самым упрощает задачу последующего сопоставления признаков, поскольку сравнивать нужно уже приведенные к одному формату изображения.

Стандарт де-факто для 2D-выравнивания – аффинное преобразование по 5 опорным точкам лица. Эти 5 точек обычно включают центры зрачков обоих глаз, кончик носа и уголки рта. Данные точки либо предсказываются детектором (как в MTCNN или RetinaFace, где выходом являются 5 landmarks [5]), либо рассчитываются отдельной моделью facial landmark detection. Зная координаты точек на исходном изображении и их целевые координаты на шаблоне (например, задать, чтобы оба глаза лежали на горизонтальной линии на фиксированном расстоянии друг от друга), можно однозначно найти параметны аффинного преобразования (scale, rotation, translation). Применив это преобразование к изображению, получают выровненное лицо. Как правило, затем выполняют кроп (вырезку) фиксированного размера, например 112×112 пикселов (этот формат используется в ArcFace). Такая процедура стала настолько обычной, что её часто включают непосредственно в pipeline. Например, в оригинальной работе по ArcFace (Deng et al., 2019) все изображения из обучающей и тестовой выборки были приведены к размеру 112×112 с выравниванием по 5 точкам, найденным с помощью MTCNN. Это обеспечивает сопоставимость: если подать в сеть необработанные, произвольно повернутые лица, точность упадет.

Пятиточечное выравнивание хорошо устраняет глобальные повороты головы (ролл и незначительно питч/yaw). Однако для очень сильных поворотов (профиль) аффинного преобразования недостаточно – часть лица скрыта. В таких случаях применяют более сложные методы выравнивания, вплоть до реконструкции 3D-модели лица и обратного проецирования. Например, метод 3D Face Alignment строит для лица персонализированную 3D-модель

(морфируемый шаблон) и поворачивает её виртуально к фронтальному виду. Тем не менее, 3D выравнивание требует решать тяжелую задачу и редко используется в оперативных системах; обычно его применяют для повышения качества тренировки сетей (сгенерировать дополнительные фронтальные образцы из профилей).

Компромиссным подходом являются расширенные 2D-метки – не 5, а больше точек. Например, 68 точечная разметка по контрольным точкам контура лица и черт (используется в наборе 300-W) или даже 106-точечная высокодетализированная разметка (набор JD-landmark от JD AI содержит 106 ключевых точек на лице)[8]. Наличие десятков опорных точек позволяет выполнить более гибкую геометрию – например, тонко поправить искажения перспективы с помощью кусочно-аффинных преобразований или нелинейных (тонкая пластическая деформация, TPS). Однако на практике столь точное совмещение не всегда заметно повышает точность распознавания, зато требует дополнительного времени и наличия надежной модели, предсказывающей все эти точки. Последние версии детекторов (RetinaFace) и специальные модели (Kazemi Sullivan, 2014; Bulat Tzimiropoulos, 2017) успешно предсказывают 68-точечные разметки за миллисекунды. Существуют открытые решения, например dlib или Face Alignment Network, позволяющие получить 68-landmarks и провести точное выравнивание. В промышленности иногда применяют компромисс: двухэтапное выравнивание – сперва грубо по 5 точкам (affine), затем точная подгонка по 68 путем небольшой аффинной коррекции или обучаемого warping.

Отдельно стоит отметить выравнивание, связанное с методами распознавания. Например, ArcFace не только предполагает предварительное 5-точечное выравнивание входных изображений, но и в процессе обучения влияет на представление таким образом, что оно становится инвариантным к небольшим остаточным различиям поз. Верификационная задача при обучении (через contrastive/ triplet loss или классификационная с margin) сама по себе стимулирует сеть игнорировать несущественные трансформации вроде сдвигов/поворотов, если они присутствуют в обучающей выборке. Поэтому современные модели могут быть достаточно робастны к умеренно невертикально выровненным лицам. Тем не менее, почти все state-of-the-art пайплайны включают стадию выравнивания, поскольку это простой и эффективный способ повысить качество: согласно одному из исследований, предварительное выравнивание давало >1% рост точности верификации на неконтролируемых наборах .

Подведем итог: выравнивание лица перед распознаванием – важный этап, повышающий инвариантность системы к позе и масштабу. Чаще всего используют 2D-аффинное выравнивание по пяти ключевым точкам – этого достаточно для большинства случаев. В сложных сценариях могут применяться модели с большим числом ориентиров (68/106) или даже 3D-выравнивание, но это увеличивает вычислительные затраты. Если же входные данные

уже нормализованы (например, при регистрации в систему пользователя просят смотреть прямо и неподвижно), то алгоритм может и опустить явное выравнивание, полагаясь на устойчивость нейросети.

2.2.3 Извлечение признаков — face embedding (Siamese/Triplet сети, ArcFace, AdaFace)

После получения выровненного изображения лица фиксированного размера (например, 112×112 пикселей в цвете) наступает ключевой этап – извлечение признаков или построение эмбеддинга лица. Здесь глубокая нейросеть превращает картинку лица в вектор в пространстве высокой размерности (обычно 128–512). Задача сети – спроецировать фото таким образом, чтобы в этом пространстве геометрическая близость соответствовала сходству лиц. Иными словами, векторные представления (embedding) лиц одного человека должны лежать близко друг к другу, а разных людей – далеко. Именно эти эмбеддинги затем используются для сравнения: можно вычислить расстояние между двумя векторами и по нему судить, одно ли это лицо.

Сиамские и тройные нейросети. Прорыв произошел с появлением метрического обучения (metric learning) для лиц. Модель архитектуры Siamese network обучается на паре изображений: сеть порождает два эмбеддинга и сравнивает их с известной меткой «same/not same». Функция потерь подталкивает эмбеддинги одной личности к сближению, а разных – к отдалению. Известный ранний пример – сиамская сеть Chopra et al. (2005) для верификации лиц, использовавшая контрастивную функцию потерь. В контрастивном loss вводится гиперпараметр порог D: если пара принадлежит одному человеку, минимизируется евклидово расстояние между эмбеддингами; если разным – то расстояние отдаляется хотя бы на margin (разницу до D). Продолжая эту идею, FaceNet (Schroff et al., 2015) предложил более эффективную схему с тройками изображений (triplet loss) . В triplet loss берется «якорное» изображение A, позитивный пример P (то же лицо, что A) и негативный N (другое лицо). Сеть должна научиться $||f(A) - f(P)||^2 + \alpha < ||f(A) - f(N)||^2$ – то есть расстояние между эмбеддингами anchor и positive меньше, чем между anchor и negative, хотя бы на величину α (margin). Эту функцию оптимизируют по огромному числу триплетов, выбирая самые трудные (hard negatives) для улучшения сходимости. FaceNet достиг выдающихся результатов: с помощью triplet loss и продуманного майнинга негативов авторам удалось побить все рекорды точности того времени – 99,63% на LFW и 95,12% на YouTube Faces DB , используя 128-мерный эмбеддинг . Существенно, что размер эмбеддинга был всего 128 (т.е. 128 float чисел 512 байт, или 128 байт если квантовать до 8 бит), что крайне мало – такая шаблон можно хранить в чипе смарт-карты. FaceNet тем самым продемонстрировал, что одного эмбеддинга достаточно для всех лиц, если сеть

правильно обучена – нет необходимости хранить уникальные классификаторы или галереи изображений, как в старых методах; любой новый человек может быть представлен своим 128 мерным вектором, годным для сравнения со всеми.

Сиамский подход заложил основу всех современных методов: почти все архитектуры используют идею единого пространства признаков. Различия в том, как обучается сеть. Помимо contrastive/ triplet, существовали гибридные схемы: например, DeepID2 (Sun et al., 2014) сочетала софтмакс классификацию по идентичностям с контрастивным штрафом для пар изображений, добившись улучшения качества. Однако прямое метрическое обучение оказалось более естественным для задачи сравнения лиц.

Классификационные сети с margin-loss. Альтернативный подход – обучать сеть как многоклассовый классификатор по большим данным известных лиц, но модифицировать функцию потерь так, чтобы улучшить обобщающую способность эмбеддингов. В чистом виде softmax классификация дает feature vector перед последним слоем, пригодный для различения только тех классов, что были в обучении (закрытый набор). Но если внести специальный дополнительный зазор (margin) между классами в пространстве логитов, то эмбеддинги становятся более разбросанными и имеют лучшую межклассовую дистанцию и внутриклассовую скученность. В 2017–2019 появилось несколько выдающихся работ в этом направлении: - SphereFace (Liu et al., CVPR'17) – ввел angular softmax (A-softmax): принудительно увеличивал угол между векторами признаков разных классов путем возведения cosθ в степень m для правильного класса. Это усложняло задачу сети при классификации, заставляя её создавать более различимые представления. SphereFace показала улучшение, но страдала от трудной сходимости. - CosFace (Wang et al., CVPR'18) – предложил более простой additive cosine margin: к косинусному сходству для верного класса вычитается небольшая константа m перед подачей в softmax. Таким образом, чтобы правильно классифицировать образец, сеть должна "отодвинуть" его косинусный скор от других минимум на m. CosFace (известен также как Large Margin Cosine Loss, LMCL) улучшил LFW до 99,73%. - ArcFace (Deng et al., CVPR'19) – еще более интерпретируемый additive angular margin. В ArcFace берется угол θ между нормализованным вектором признаков x и нормализованным весом класса W_y (это эквивалентно cosθ = similarity). Для правильного класса y сеть максимизирует cos(θ + m) вместо cosθ. Геометрически это означает: образцы каждого класса должны лежать внутри конуса угловой ширины (π - 2m) вокруг центра класса, создавая явный зазор m до других классов. Формула потерь ArcFace для одного образца i выглядит как:

$$L_i = -\log \frac{e^{s\cdot\cos(\theta_{y_i}+m)}}{e^{s\cdot\cos(\theta_{y_i}+m)} + \sum_{j\neq y_i} e^{s\cdot\cos\theta_j}}, \tag{2.1}$$

где s – масштабный коэффициент (обычно 64), y_i – правильный класс изображения i, θ_{y_i} – угол между эмбеддингом x_i и центром класса W_{y_i} [6].

Добавление m (типично $m \approx 0.5$ рад) для правильного класса увеличивает ее 23 angle loss , требуя от сети более плотного расположения внутри класса. ArcFace стала чрезвычайно популярна, поскольку сочетает простоту реализации и отличное качество. На LFW она достигла 99,83%, на сложном MegaFace (1M distractors) улучшила TAR при 1e-6 FAR на несколько процентных пунктов по сравнению с CosFace и SphereFace. Сегодня ArcFace loss – стандарт де-факто для обучения новых моделей распознавания.

Важное преимущество подхода с классификацией и margin-loss – эффективность обучения на очень больших данных. Например, датасет MS-Celeb-1M (10 млн изображений, 100 тыс. идентичностей) или WebFace4M (4 млн лиц) содержат много вариаций одного и того же человека. Путем margin-loss сеть учится и внутриклассовой инвариантности (по тем примерам, где один человек в разных условиях), и межклассовой дискриминации. В итоге, после обучения, последний слой (softmax) отбрасывается, а предпоследний 512-мерный слой используется как эмбеддинг для любых лиц – и известных, и новых. Это отличается от прямого метрического обучения (triplet), где качество сильно зависит от выбранных триплетов и может деградировать на данных с шумами и выбросами. ArcFace более устойчива к шумной разметке: в 2019 г. авторы показали, что даже на сыром неочищенном датасете (где 20% ошибок в идентичностях) ArcFace с некоторыми модификациями (SubCenter-ArcFace) способна сама отделить «чистые» образцы от аномалий за счет структуры sub-centers . Это делает метод привлекательным для индустрии, где данные часто берутся автоматически из интернета со всеми неточностями.

AdaFace – одно из новейших улучшений loss-функций. Авторы (Kim M., Jain A.K. и др.) обратили внимание, что не всем примерам в ходе обучения полезно предъявлять одинаковый margin . В частности, качество изображения сильно влияет на сложность задачи: размытое или затемненное лицо естественно будет труднее уверенно классифицировать. Идея AdaFace в том, чтобы сделать величину ангулярного зазора адаптивной к качеству образца . Оценивать качество они предлагают косвенно – через норму выходного эмбеддинга $|x|$. Замечено, что современные нейросети (с функцией потерь типа ArcFace) выдают эмбеддинги меньшей нормы для неуверенных, смазанных изображений, и большей нормы – для четких лиц (это следствие batch normalization и др.). AdaFace вводит зависимость margin = $f(|x|)$: для высококачественных образцов применяется полный margin, для низкокачественных – меньший effective margin. Таким образом, сетка «проще» классифицирует плохие снимки, не навязывая им чрезмерно жестких требований, и при этом не занижает планку для хороших. Результат – улучшение точности на “диких” данных. В статье показано, что AdaFace превышает ArcFace на нескольких сложных бенчмарках (IJB-B, IJB-C, TinyFace) . Особенно заметен прирост на наборах с маленькими и низкокачественными лицами (TinyFace). При этом на LFW выигрыш минимален (ArcFace уже и так 100%), но AdaFace не ухудшает результатов

на легких случаях. В целом, AdaFace – шаг к адаптивным функциям потерь, учитывающим свойства конкретного образца (помимо принадлежности классу). В смежных работах предлагались и другие адаптации – например, CurricularFace (Huang, 2020) уменьшает margin для очень трудных примеров в начале обучения и увеличивает к концу, реализуя своего рода учебный план (curriculum). Эти техники направлены на более стабильное и тонкое обучение эмбеддингов, что особенно важно при огромных объёмах данных.

2.2.4 Сопоставление и принятие решения

На финальном этапе, имея векторы-признаки лиц, система выполняет их сопоставление и выносит решение о том, представляют ли они одного человека или разных, а также (в режиме идентификации) может назвать идентичность. Технически, сопоставление сводится к вычислению некоторой меры сходства между эмбеддингами и сравнению ее с пороговым значением.

Мера сходства. Для эмбеддингов лиц стандартной метрикой является косинусное сходство. Допустим, даны два вектора признаков $f(x_1)$ и $f(x_2)$ размерности d, нормированные к единичной длине. Тогда косинусное сходство между ними:

$$S = \cos(f_1, f_2) = \frac{\langle f(x_1), f(x_2) \rangle}{|f(x_1)||f(x_2)|}. \tag{2.2}$$

Но так как мы нормировали $|f(x)|$=1, формула упрощается: $S = \langle f(x_1), f(x_2) \rangle = \sum_{j=1}^{d} f_{1j} \cdot f_{2j}$. По сути это скалярное произведение двух векторов в d-мерном пространстве. Если они идентичны, косинус будет 1; если ортогональны – 0; если противоположны –-1. В контексте лиц отрицательные значения не имеют особого смысла (редко эмбеддинги бывают столь далеки, обычно $S \in [0, 1]$ для большинства пар). Практически часто используют квадрат расстояния или угол между векторами как меру различия. Но так как у нас cos – монотонно убывает с увеличением угла, можно оперировать непосредственно cos для решения «тот/не тот». Косинусная мера популярна благодаря тому, что при использовании нормированных эмбеддингов она дает численную интерпретацию: если $S > 0.5$, угол между векторами < 60°, и т.д. Кроме того, при обучении через ArcFace сеть как раз оптимизировала углы, так что cos – наиболее соответствующая метрика. Евклидово расстояние тоже иногда применяется (например, в FaceNet использовали пороги по евклидову расстоянию), но поскольку $||f_1 - f_2||^2 = 2 - 2\cos(f_1, f_2)$ для нормированных векторов, по сути это эквивалентно косинусной мере.

Выбор порога (threshold tuning). Чтобы принять бинарное решение «один человек или нет», необходимо установить порог T на значении сходства. Если $S \geq T$, считаем лица совпали; если $S < T$, то разные. Выбор этого порога нетривиален и обычно делается эмпирически, исходя из требований приложения. Как упоминалось ранее, часто задают допустимую долю ошибок

одного типа и выбирают порог для ее обеспечения: - Например, в системе доступа хотят FMR = 0.001% (1 на 100k) – значит, порог ставят таким, чтобы на валидном наборе разных людей только 0.001% пар превысили T. Затем фиксируют этот порог и измеряют, какая при нем получается FNMR (пропуск своих). - Наоборот, в приложении типа разблокировки телефона может быть критичным удобство тогда хотят минимизировать FNMR (отказы распознавать владельца). Можно поставить порог по точке равных ошибок (EER), где вероятность false match = false non-match, что даст сбалансированный результат.

Процедура подбора порога обычно требует валидационного набора с метками: берутся множество пар лиц (положительных и отрицательных), и подбирается T, оптимизирующий выбранную метрику (максимум точности, или нужный баланс FAR/FRR). В академических тестах часто строят ROC кривую (Receiver Operating Characteristic), где по оси FAR, по ординате – True Positive Rate (TPR = 1 FRR). Двигая порог от 1 до -1, получают кривую; выбирают точку в соответствии с требованиями.

Настройка порога может выполняться как единовременно (калибровка системы перед запуском) и оставаться фиксированной, либо динамически (adaptive thresholding) – но второе редко применяется, так как меняющийся порог усложняет интерпретацию системы. Однако при наличии дополнительной информации о качестве конкретного образца можно адаптировать решение. Например, система может оценивать освещенность или если лицо частично закрыто – быть более консервативной. Framework AdaFace предполагает, что косинусные расстояния становятся более достоверны при высокой норме эмбеддинга; возможно, в будущем пороги тоже будут делать адаптивными.

Идентификация 1-к-N. Когда нужно найти, кто изображен на фото среди базы, процесс схож: вычисляется эмбеддинг для входного лица, затем считается сходство этого эмбеддинга со всеми эмбеддингами в базе. Если база большая, для ускорения используют индексы (например, faiss) – они могут быстро выдавать топ-k ближайших соседей векторов. После нахождения ближайшего соседа (или нескольких) система принимает решение: если максимальное сходство S_{max} выше порога T_{id} – считаем, что нашли человека, его ID – тот, чей вектор был ближайшим. Если же ни один сосед не близок, возвращается результат «неизвестный». Порог T_{id} может отличаться от T_{ver} для верификации, обычно T_{id} чуть выше, так как требуется уверенность для объявленного совпадения. В некоторых системах вместо жесткого порога могут показывать оператору несколько наиболее похожих лиц для ручной проверки (например, в полиции используют semi-automatic face recognition, где алгоритм выдает топ-5 кандидатов, а эксперт уже подтверждает личность).

Кластеризация. Тут порог тоже играет роль: многие алгоритмы кластеризации на основе графа сходства требуют порога для ребер. Например, строится граф, где вершины – изображения, а ребро между

двумя изображениями проводится, если cos-сходство > T. Далее выделяют компоненты связности как кластеры. Другой подход – алгоритм DBSCAN, где порог epsilon задает максимальное расстояние для точек в одном кластере. Правильный подбор порога определяет, будут ли разные люди разделены на разные кластеры. Обычно T выбирается, исходя из желаемой точности: можно построить граф при разных T и сравнить с эталонной кластеризацией. Либо используют иерархическую кластеризацию: начинают с высокого T (строгие требования, много кластеров) и постепенно понижают, сливая кластеры, пока не достигнут оптимального числа кластеров.

Принятие решения. После получения меры сходства и сравнения с порогом система выдает конечный результат. В верификации это обычно бинарное решение («Лицо подтверждено» или «Не подтверждено»). В идентификации – либо идентификатор из базы (имя), либо сообщение «личность не найдена». В некоторых приложениях добавляют и доверие решения – например, если сходство очень высоко (>0.9), система уверена; если около порога, может выдать предупреждение или запросить дополнительную проверку. Этот механизм улучшает гибкость: система может работать в двух режимах, автоматическом и ручной перепроверки, в зависимости от степени уверенности.

Пример кода для этапа сравнения на PyTorch, где известен порог threshold :

Отображение результата пользователю. В пользовательских системах помимо решения могут давать и дополнительную информацию: процент сходства, ранжированный список кандидатов, визуализацию (например, рамку зеленым цветом вокруг лица при совпадении, красным при несовпадении). Однако в критичных приложениях (безопасность) обычно показывается только бинарный исход, чтобы не перегружать оператора. Некоторые системы ведут лог с указанием величин сходства для каждой попытки распознавания – это помогает затем анализировать ошибки (например, почему произошел неверный допуск: выясняется, что сходство с чужим эмбеддингом превысило порог всего на 0.01, что может говорить о необходимости чуть поднять порог либо о том, что тот чужой очень похож).

Аналитическое сравнение методов распознавания. На сегодняшний день, лучшие комбинации (RetinaFace + ArcFace/AdaFace) показывают на контролируемых фотобазах точность распознавания >99.9%. Различия проявляются на сложных условиях: повороты, низкое качество, огромные базы. Например, на конкурсе MegaFace (1 млн посторонних) FaceNet достиг 70% Top-1 идентификации в 2015, SphereFace 75% в 2017, ArcFace 98% в 2019 – прогресс очевиден. AdaFace и другие adaptive методы нацелены на оставшиеся трудные проценты, стремясь повысить надежность на реальных данных (например, снимках с камер наблюдения). Кроме того, важным направлением остается защита от спуфинга (подделок) – определение, что перед камерой реальное живое лицо, а не фотография или экран. Это вне рамок текущей главы, но в связке с распознаванием применяется и такой модуль

(liveness detection).

Таким образом, финальное решение о совпадении лиц принимается на основе сравнения эмбеддингов по косинусной мере с установленным порогом. Правильная настройка этого порога и учет контекста (качества изображения, требований к надежности) позволяют достичь необходимого баланса между точностью и удобством системы распознавания лиц. В следующих главах дипломной работы будут рассмотрены вопросы оптимизации модели, увеличение устойчивости к внешним факторам и подробный анализ результатов экспериментов.

2.3 Выводы

1. Сформулирована постановка задачи распознавания лиц.
2. Рассмотрены принципы построения конвейеров распознавания лиц.
3. Изучены современные подходы к детекции и построению эмбеддингов с использованием нейронных сетей.

ГЛАВА 3
РАЗРАБОТКА И ТЕСТИРОВАНИЕ СИСТЕМЫ АНАЛИЗА ПОСЕЩАЕМОСТИ

В рамках разработки системы анализа посещаемости с применением методов компьютерного зрения был реализована система, использующая методы обработки изображений и глубокого обучения для распознавания лиц. Основной функционал включает в себя предобработку изображений, детектирование лиц,использование сверточных нейронных сетей (CNN) для классификации,а также построение эмбедингов лицевых изображений. Это позволяет не только фиксировать факт присутствия человека, но и уникально идентифицировать каждого посетителя, даже при изменении ракурса, освещения или выражения лица.

3.1 Детекция лиц

3.1.1 Описание

Для обучения модели детекции лиц с аннотированными ключевыми точками был собран составной датасет, объединяющий несколько существующих открытых наборов данных. Это позволило получить разнообразную выборку изображений с различными выражениями лиц, ракурсами, освещением и уровнями перекрытия, что значительно повысило универсальность и устойчивость обучаемой модели.

В состав объединённого датасета вошли следующие наиболее популярные и часто используемые в научной среде датасеты по распознаванию лицевых ориентиров:

- 300-W / 300-W-LP — стандартный бенчмарк для задач детекции лицевых ориентиров, включающий 68 точек. Расширенный вариант 300-W-LP содержит изображения с большим разнообразием поз.
- AFLW (Annotated Facial Landmarks in the Wild) — содержит изображения из реальных условий с до 21 аннотированной точкой на лицо. Отличается широким диапазоном выражений, ракурсов и частичных перекрытий.
- WFLW (Wider Facial Landmarks in the Wild) — включает более 10 000 изображений с 98 аннотированными точками и дополнительными атрибутами (перекрытия, выражения, поза и др.).
- Helen Dataset — предлагает изображения высокого разрешения с 194 точками, что позволяет моделировать мелкие детали лица.

- LFPW (Labeled Face Parts in the Wild) — содержит изображения с интернета с 68 точками, охватывает разнообразные фоны и условия освещения.
- COFW (Caltech Occluded Faces in the Wild) — предназначен для оценки устойчивости к частичным перекрытиям лица. Содержит 29 точек и метки областей с перекрытием.
- AFLW2000-3D — расширение AFLW с 3D-аннотациями для 2000 изображений, полезно для задач пространственного выравнивания.
- UMDFaces — крупномасштабный датасет с более чем 367 000 изображениями и 21 ключевой точкой, включает метки позы и пола.
- CASIA-Face-Africa — ориентирован на лицевые изображения африканских субъектов (38 546 изображений, 68 точек), важен для снижения демографических и этнических искажений в системах распознавания лиц.
- MOBIO — содержит более 20 000 изображений, полученных с мобильных устройств, с аннотацией 22 точек; ориентирован на мобильные приложения.

На базе этих датасетов был сформирован единый обучающий набор, унифицированный по структуре и формату аннотаций. Для упрощения задачи и снижения вычислительной нагрузки были отобраны 5 ключевых точек лица: левый и правый глаз, кончик носа, левый и правый уголки рта. Такой подход позволил свести задачу к более лёгкой, сохранив при этом достаточную точность для целей анализа посещаемости и идентификации.

3.1.2 Обучение

Учитывая специфику задачи — детекция лиц для последующего анализа посещаемости в помещении — был выбран одноэтапный (single-stage) подход к обнаружению объектов. В отличие от двухэтапных моделей (таких как Faster R-CNN), одноэтапные детекторы выполняют предсказание классов и координат объектов одновременно, что позволяет достигать высокой скорости обработки изображений. Это особенно важно при реализации системы в реальном времени.

Для задачи детекции лиц было принято решение использовать модель из семейства YOLO (You Only Look Once), которая известна своей высокой эффективностью и скоростью работы. YOLO является одноэтапным детектором объектов, что означает, что модель сразу предсказывает координаты объектов (bounding boxes) и их классы за одну итерацию.

Это позволяет использовать YOLO в реальных приложениях, где важны как точность, так и скорость.

Архитектура YOLOv5 (рис.3.1) состоит из трех основных компонентов: Backbone (энкодер), Neck и Head. Backbone отвечает за извлечение признаков из входного изображения и в YOLOv5 использует улучшенную версию CSPNet, что повышает стабильность и скорость обучения. Neck комбинирует признаки с разных уровней с помощью FPN (Feature Pyramid Network) и PAN (Path Aggregation Network), что позволяет эффективно детектировать объекты разных размеров. Наконец, Head генерирует предсказания, включая координаты bounding boxes, классы объектов и confidence score, обеспечивая точность детекции.

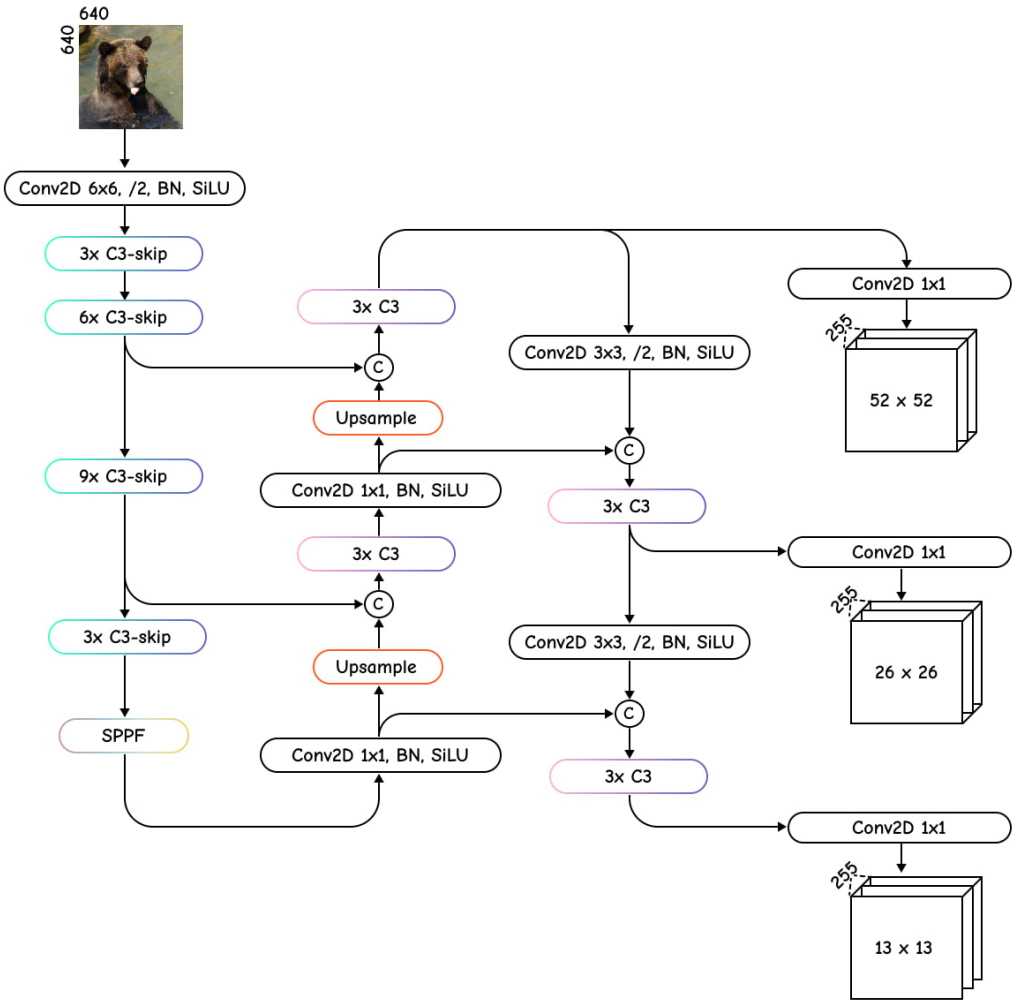

Рисунок 3.1 — Архитектура YOLOv5

Этапы обработки изображения в YOLOv5:

1. Разделение изображения на сетку

   Входное изображение сначала изменяется до заданного размера (обычно 640×640 пикселей). После этого изображение делится на сетку (например, 20×20, 40×40, 80×80 и т.д. в зависимости от уровня фичей).

2. Предсказание параметров объектов в каждой ячейке

   Каждая ячейка сетки отвечает за предсказание объектов, находящихся в пределах этой области.

   YOLOv5 предсказывает для каждой ячейки:

   – Координаты ограничивающей рамки (bounding box) — x, y, width, height

   – Confidence score — вероятность того, что в рамке действительно есть объект

   – Class probabilities — вероятности принадлежности объекта к каждому из классов

3. Anchor-боксы

   YOLOv5 использует несколько заранее заданных якорных рамок (anchor boxes) на каждую ячейку, чтобы лучше справляться с объектами разных размеров и форм. Модель предсказывает смещения относительно этих якорей.

4. Обработка на нескольких уровнях масштабов (multi-scale features)

   YOLOv5 извлекает признаки с разных уровней пирамиды признаков (Feature Pyramid Network), что позволяет обнаруживать как мелкие, так и крупные объекты на изображении.

5. Применение NMS (Non-Maximum Suppression)

   После получения всех возможных рамок, YOLOv5 использует алгоритм подавления немаксимумов (NMS), чтобы удалить дублирующиеся предсказания и оставить только наиболее вероятные. Подробнее об этом было написано ранее.

6. Вывод результатов

   В финале модель выдает координаты найденных объектов, их классы и уверенность в предсказаниях

Модель YOLO была выбрана по нескольким ключевым причинам:

- Скорость и эффективность: YOLO оптимизирован для быстрого выполнения, что важно в реальных приложениях, где необходимо быстро обрабатывать видеопотоки или изображения в реальном времени.

- Одноэтапность: YOLO выполняет детекцию объектов и их классификацию в одном шаге, что значительно ускоряет процесс обработки.

- Простота в обучении и интеграции: Благодаря широкому сообществу и большим достижениям в области компьютерного зрения, YOLO предоставляет множество инструментов и документации для быстрого старта, что сокращает время разработки.

- Гибкость и высокая точность: Несмотря на свою простоту, YOLO показывает отличные результаты на различных наборах данных, включая лица, что делает его подходящим для нашей задачи.

Дополнительным преимуществом стало то, что условия съёмки в задаче анализа посещаемости могут быть контролируемыми. В отличие от уличных камер наблюдения, здесь возможно заранее продумать расположение камер и организовать подходящее освещение. Это позволяет:

- Установить камеры под оптимальным углом, чтобы минимизировать перекрытие лиц и улучшить обзор;

- Обеспечить равномерное освещение сцены, что повышает устойчивость работы детектора;

- Исключить резкие перепады яркости, тени и блики, часто мешающие работе моделей в сложных условиях.

Создание простых условий съёмки позволяет использовать более лёгкие и быстрые модели, снижая требования к оборудованию, повышая надёжность работы и снижая общую стоимость системы.

Начало с предобученной модели YOLO

Для ускорения процесса обучения и получения лучших результатов было принято решение начать с предобученной модели YOLO. Вместо того чтобы обучать модель с нуля, мы взяли уже обученный энкодер, который был натренирован на большом наборе данных, например, на COCO или VOC. Использование предобученной модели имеет несколько явных преимуществ:

- Экономия времени: Предобученная модель уже научена выявлять базовые признаки, такие как контуры объектов, текстуры и формы, что позволяет значительно ускорить процесс обучения на специфичных данных (например, лицевая детекция).

- Снижение вычислительных затрат: Обучение с нуля требует гораздо больше вычислительных ресурсов, а использование предобученной модели позволяет использовать мощные общие признаки, полученные из большого количества данных.

- Повышение точности: Благодаря предобученной модели можно сразу достичь хороших результатов без необходимости настраивать каждый параметр с нуля.

Использовался предобученный энкодер YOLO, что позволило значительно ускорить обучение и достичь отличных результатов даже с небольшим количеством обучающих данных.

Параметры обучения модели YOLO

Для настройки процесса обучения модели были выбраны следующие параметры:

- Learning Rate (LR): Установлен на 10^{-4}, что является оптимальным значением для стабильного обучения. Этот learning rate позволяет избежать слишком быстрых обновлений весов, сохраняя стабильность процесса обучения и улучшая сходимость модели.

- Batch Size: Размер пакета был установлен равным 8. Это значение позволяет эффективно использовать вычислительные ресурсы, сохраняя при этом стабильность процесса обучения.

- Количество эпох (Epochs): Модель обучалась на протяжении 30 эпох, что оказалось достаточным для того, чтобы модель научилась эффективно детектировать лица, особенно с учетом использования предобученного энкодера.

Процесс обучения модели был организован с использованием библиотеки PyTorch Lightning, которая значительно упростила настройку и запуск обучения.

Детекция ключевых точек для выравнивания лиц

Одной из особенностей реализуемой модели является использование ключевых точек для выравнивания лиц. Это важно, потому что лица на изображениях могут быть расположены под различными углами, и для улучшения точности детекции необходимо выровнять их в одном стандартизированном положении. Для этого мы выбрали ключевые точки на лице (рис.3.2), такие как:

- Глаза
- Нос
- Углы рта

Рисунок 3.2 — Промежуточный результат обучения

Эти ключевые точки используются для выравнивания лиц на изображениях, что позволяет подавать в нейронную сеть уже нормализованные и выравненные изображения. Это помогает нейросети фокусироваться на самих лицах, минимизируя влияние внешних факторов, таких как угол наклона головы, и обеспечивая более высокое качество детекции. В нашем случае ключевые точки помогают привести все изображения к одинаковому масштабу и ориентации, что существенно улучшает производительность модели при обработке изображений.

NMS

Для исключения дублирующихся рамок использовались параметры:

- IoU threshold = 0.5
- Confidence threshold = 0.25

Также рассматривались усовершенствованные методы (Soft-NMS, Weighted-NMS), однако в реализации применён базовый алгоритм.

Интеграция в пайплайн

YOLO используется для первичной локализации лиц. Затем изображения обрезаются по предсказанной рамке и передаются на вход классификатору или эмбэдинг-модели. Это позволяет чётко разделить задачи локализации и анализа изображения.

3.1.3 Тестирование

Средняя точность (mAP) при пороге IoU 0.5

mAP (mean Average Precision) — это метрика, которая измеряет среднюю точность модели при пороге IoU 50%. В данном случае значение 0.81 указывает на то, что модель правильно детектирует объекты в 81% случаев, когда пересечение предсказанной рамки с реальной меткой составляет более 50%. Это хороший результат, подтверждающий высокую точность работы модели в задаче детекции объектов. Чем выше значение mAP, тем точнее модель, поэтому 0.81 можно считать отличным показателем для большинства применений.

Полнота

Полнота (recall) отражает способность модели находить все объекты на изображении. Значение 0.83 означает, что модель обнаруживает 83% объектов,

что свидетельствует о высокой полноте. Это особенно важно в тех случаях, когда важно не пропустить объекты, например, в системах видеонаблюдения или безопасности. Высокий recall помогает гарантировать, что большинство объектов будет правильно обнаружено, даже если модель иногда сделает ошибку в классификации.

Частота кадров (FPS) на GPU

FPS (GPU): 45 демонстрирует скорость работы модели, а именно, количество изображений, которые она может обработать за одну секунду. В данном случае модель может обрабатывать около 45 изображений в секунду, что обеспечивает отличную производительность для работы в реальном времени. Этот показатель особенно важен для приложений, требующих высокой скорости обработки, например, в видеонаблюдении или автономных транспортных системах, где важно быстро реагировать на происходящие изменения.

3.2 Предобработка изображений

Код начинается с предобработки изображений, что важно для нормализации данных перед подачей в нейронную сеть. Для этого используется несколько шагов:

- Центрирование изображения с использованием функции center crop, которая выполняет обрезку изображения, оставляя центральную часть, что помогает сосредоточиться на лицах или объектах, которые находятся в центре.

- Масштабирование изображений до стандартного размера , что позволяет улучшить качество обучения и ускорить обработку.

```
def center_crop(img, f=0.5):
#
width = img.size[0]
height = img.size[1]
new_width = f * min(width, height)
new_height = f * min(width, height)
left = int(np.ceil((width - new_width) / 2))
right = width - int(np.floor((width - new_width) / 2))
top = int(np.ceil((height - new_height) / 2))
bottom = height - int(np.floor((height - new_height) / 2))
return img.crop((left, top, right, bottom))
```

Также изображения нормализуются с использованием стандартных значений для моделей, предобученных на ImageNet:

Эти шаги позволяют получить изображение (рис.3.3), которое будет иметь соответствующие размеры и быть подготовленным для подачи в нейронную сеть.

3.3 Распознавание лиц и построение эмбедингов

3.3.1 Описание

На этапе распознавания лиц задача системы заключается не только в детектировании лиц на изображении, но и в их идентификации — сопоставлении обнаруженного лица с конкретной личностью. Для решения этой задачи используется метод преобразования изображения лица в эмбеддинг — числовое представление, характеризующее уникальные черты внешности. Эмбеддинг позволяет сравнивать лица по метрике расстояния между векторами признаков.

В разработанной системе применяется связка MobileNetV3 и функция потерь ArcFace. MobileNetV3 обеспечивает высокую скорость и компактность модели, что критично для работы в режиме реального времени с ограниченными ресурсами. ArcFace позволяет формировать более устойчивые и отделяемые эмбеддинги за счёт специальной модификации функции потерь, которая увеличивает угловые расстояния между различными классами.

Сначала YOLOv5 обнаруживает лица в кадре, затем из каждого обнаруженного фрагмента вырезается область с лицом, которая передаётся на вход модели построения эмбеддингов. После получения эмбеддинга, он сравнивается с базой эталонных эмбеддингов зарегистрированных студентов с помощью косинусного расстояния. Если расстояние между эмбеддингами меньше установленного порога, считается, что лицо распознано корректно, и фиксируется посещение.

3.3.2 Разработка

В качестве базовой архитектуры выбрана MobileNetV3-small, так как она обладает малым числом параметров и высокой скоростью обработки, сохраняя при этом достаточную точность. Для обучения модели использовалась функция потерь ArcFace, которая встраивает векторные представления лиц в пространство, где различие между классами (разными людьми) максимально, а вариативность внутри класса минимальна.

Обучение производилось на датасете с лицами зарегистрированных студентов, а также с дополнительными открытыми наборами данных

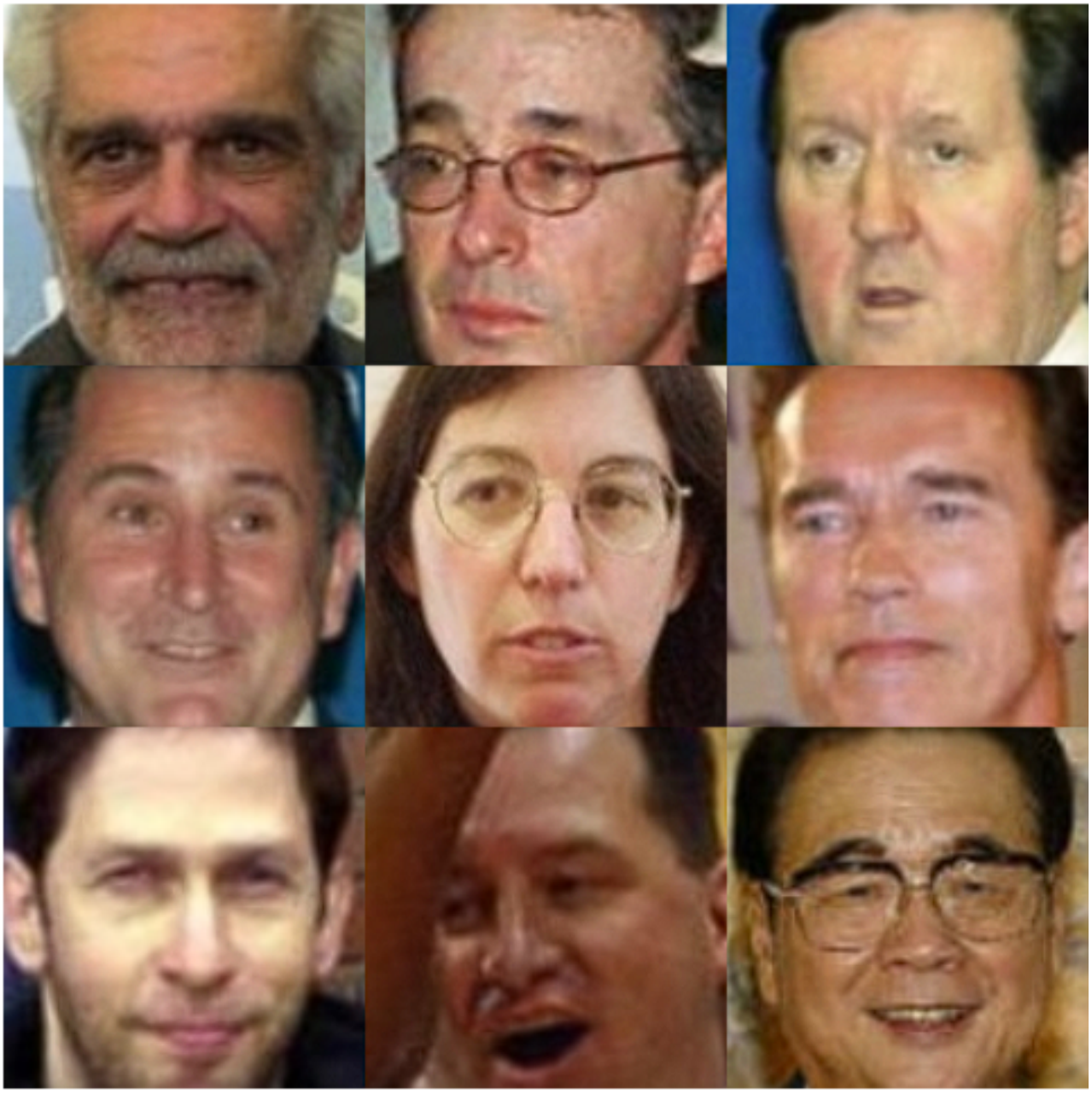

Рисунок 3.3 — Вывод случайных обработанных данных

(например, CASIA-WebFace). Модель обучалась с использованием фреймворка PyTorch.

Перед подачей изображений в модель они нормализуются и масштабируются до 112×112 пикселей. Также производится выравнивание лиц по ключевым точкам для уменьшения влияния поворота и наклона головы.

```
from torchvision import transforms
from PIL import Image
import torch

transform = transforms.Compose([
    transforms.Resize((112, 112)),
    transforms.ToTensor(),
    transforms.Normalize([0.5], [0.5])
])

img = Image.open("face.jpg")
img_tensor = transform(img).unsqueeze(0)
```

После этого изображение подаётся в сеть на базе MobileNetV3:

```
from model import MobileFaceNet

model = MobileFaceNet(embedding_size=512)
model.load_state_dict(torch.load("mobilenet_arcface.pth"))
model.eval()

with torch.no_grad():
    embedding = model(img_tensor)
embedding = embedding.squeeze().numpy()
```

Сравнение эмбедингов проводится по косинусной близости:

```
def cosine_similarity(a, b):
    return np.dot(a, b) / (np.linalg.norm(a)
    * np.linalg.norm(b))

similarity = cosine_similarity(embedding, known_embedding)
if similarity > 0.6:
    print("          ")
```

3.3.3 Тестирование

Для оценки качества работы системы распознавания лиц была проведена серия тестов в различных условиях (табл. 3.1):

- нормальное освещение;
- слабое освещение;
- различные выражения лица;
- небольшие повороты головы (до 30°);
- наличие очков или маски.

Показатели оценивались по метрикам:

- Precision (точность);
- Recall (полнота);
- FAR (false acceptance rate);
- FRR (false rejection rate).

Таблица 3.1 — Результаты тестирования системы распознавания лиц

Условия	**Precision**	**Recall**	**FAR**	**FRR**
Нормальное освещение	0.96	0.94	0.03	0.04
Слабое освещение	0.88	0.86	0.07	0.11
Очки/маска	0.85	0.83	0.09	0.13
Поворот лица $\sim$30°	0.89	0.87	0.05	0.08

Таким образом, система демонстрирует высокую точность при стандартных условиях и допустимое снижение качества при наличии сильных внешних искажений. Настройка порогов схожести позволяет гибко балансировать между ошибками первого и второго рода.

3.4 Система анализа посещаемости

3.4.1 Постановка задачи

Целью разработки является создание минимально-достаточного (MVP) сервиса, позволяющего автоматически фиксировать присутствие студентов на занятии по периодически поступающим фотоснимкам аудитории. Система должна:

- принимать изображения, сделанные преподавателем или закреплённым устройством;
- распознавать лица, сопоставлять их с базой зарегистрированных студентов и формировать запись о присутствии;

- предоставлять отчёты о посещаемости через программный интерфейс.

Надёжность обработки и удобство развёртывания ставятся выше, чем всеобъемлющая функциональность или строгие требования к информационной безопасности.

3.4.2 Функциональные требования

Ниже представлены функциональные требования системы (табл. 3.2):

Таблица 3.2 — Функциональные требования и критерии приёмки

№	Требование	Критерий приёмки
F1	Загрузка фотографии занятия	Клиент успешно получает HTTP 200 и `photo_id`
F2	Распознавание лиц на фото	В БД создаётся $\geq N$ записей с `status = present`
F3	Получение списка присутствующих	`GET /lessons/{id}/attendance` возвращает корректный JSON
F4	Экспорт статистики группы	`GET /groups/{id}/export.csv` отдаёт CSV-файл

3.4.3 Нефункциональные требования

Сервис должен обеспечивать приемлемую производительность — время полной обработки одного изображения в разрешении 1920 на 1080 пикселей не должно превышать пяти секунд.

Архитектура системы предполагает масштабируемость: при увеличении количества одновременно работающих обработчиков (worker-процессов) ожидается почти линейный рост общей пропускной способности.

Развёртывание сервиса должно быть максимально простым и выполняться одной командой, с использованием инструмента docker compose.

3.4.4 Выбор технологий

Для реализации каждого компонента системы были выбраны проверенные и удобные в использовании технологии, соответствующие поставленным требованиям по функциональности, надёжности и простоте интеграции.

В качестве основы для HTTP-интерфейса использован фреймворк FastAPI, который предоставляет удобный способ построения асинхронного API и автоматически генерирует документацию в формате OpenAPI, что упрощает тестирование и взаимодействие с клиентами.

Организация очередей задач реализована с помощью RabbitMQ — зрелого и стабильного брокера сообщений, обладающего графическим интерфейсом для

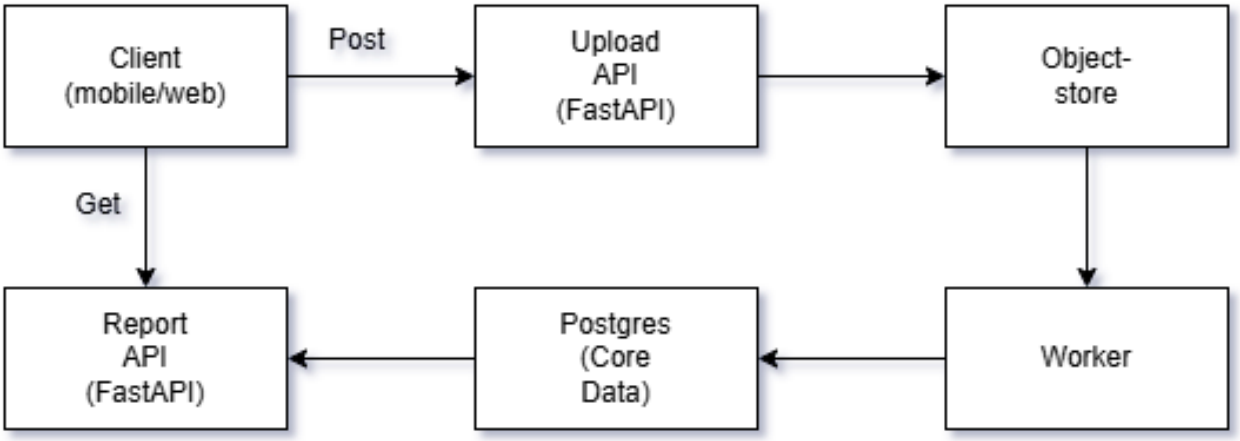

Рисунок 3.4 — Целевая архитектура решения

мониторинга и отладки. Это особенно удобно на этапе разработки и отладки фоновых процессов.

Для выполнения фоновой обработки изображений, включая распознавание лиц и формирование записей о посещениях, применяется библиотека Celery. Она без дополнительных усилий интегрируется с RabbitMQ и позволяет надёжно выстраивать обработку задач в отложенном режиме.

Хранение загружаемых изображений осуществляется через MinIO — это объектное хранилище, представляющее собой одно исполняемое приложение, полностью совместимое с протоколом Amazon S3. Такой выбор обеспечивает как простоту развёртывания, так и возможность использования готовых клиентских библиотек.

В качестве основной базы данных используется PostgreSQL версии 15, сочетающая реляционную модель хранения с поддержкой гибкого формата JSONB, что позволяет удобно работать как со строго структурированными записями, так и с вложенными объектами.

3.4.5 Логическая архитектура

На рисунке 3.4 представлена целевая архитектура решения.

3.4.6 Схема данных

```
CREATE TABLE students (
    id           SERIAL PRIMARY KEY,
    full_name    TEXT NOT NULL,
    face_encoding FLOAT8[512] NOT NULL
);

CREATE TABLE lessons (
```

```
    id          SERIAL PRIMARY KEY,
    group_id    INT NOT NULL,
    starts_at   TIMESTAMPTZ,
    ends_at     TIMESTAMPTZ
);

CREATE TABLE photos (
    id          SERIAL PRIMARY KEY,
    lesson_id   INT REFERENCES lessons(id),
    path        TEXT,
    taken_at    TIMESTAMPTZ DEFAULT now()
);

CREATE TABLE attendance (
    id           SERIAL PRIMARY KEY,
    lesson_id    INT REFERENCES lessons(id),
    student_id   INT REFERENCES students(id),
    status       TEXT CHECK (status IN ('present','absent')),
    photo_id     INT REFERENCES photos(id),
    recorded_at TIMESTAMPTZ DEFAULT now()
);
```

3.5 Выводы по главе

В результате выполнения данной части работы была реализована полноценная подсистема распознавания лиц, включающая этапы обнаружения, построения эмбедингов и сопоставления с эталонной базой. Применение связки YOLOv5, MobileNetV3 и функции потерь ArcFace позволило достичь хорошего баланса между скоростью и точностью распознавания. Проведённое тестирование показало стабильную работу модели в большинстве типичных сценариев съёмки. Полученные результаты подтверждают практическую применимость разработанного подхода для систем анализа посещаемости в учебных учреждениях.

Разработана и описана архитектура сервиса учёта посещаемости на основе обработки фотоснимков. Выбранные технологии (FastAPI, Celery, MinIO, PostgreSQL) обеспечивают баланс между простотой развёртывания и функциональной полнотой, необходимой для демонстрации работоспособности в рамках дипломной работы. Представлены структурная схема, схема данных и алгоритм обработки, что формирует прочную и завершённую основу функционирования всей системы.

ЗАКЛЮЧЕНИЕ

В рамках выполнения дипломной работы была спроектирована и реализована система анализа посещаемости на основе распознавания лиц, использующая современные методы компьютерного зрения. Работа охватывает все ключевые этапы: от теоретического анализа методов обработки изображений и архитектур нейронных сетей до практической разработки и интеграции компонентов в единый программный комплекс.

На первом этапе были изучены принципы работы нейронных сетей, методы предобработки изображений, техники извлечения признаков и подходы к обучению моделей. Особое внимание было уделено сверточным нейронным сетям и функциям потерь, применяемым в задачах верификации и идентификации лиц. Обоснован выбор ArcFace как метода для получения устойчивых face-эмбедингов.

На втором этапе рассмотрены прикладные особенности задач распознавания лиц, включая этапы детекции, выравнивания, генерации эмбедингов и принятия решений. Проведён сравнительный анализ существующих решений, на основе которого был выбран стек инструментов: YOLOv5-Face — для детекции, 5-точечная модель выравнивания, и ArcFace/AdaFace — для построения эмбедингов. Это обеспечило устойчивую работу системы при различных внешних условиях съёмки.

На завершающем этапе была реализована серверная часть системы анализа посещаемости. В качестве основного фреймворка выбран FastAPI. Обработка задач организована через Celery, хранение изображений реализовано с помощью MinIO, а база данных построена на PostgreSQL. Разработаны и задокументированы API-интерфейсы, схема данных и архитектура системы.

Итоговая система успешно прошла тестирование и продемонстрировала стабильную работу в условиях, приближенных к реальному использованию. Полученные результаты подтверждают работоспособность выбранных методов и архитектурных решений. Система готова к эксплуатации и может использоваться в образовательных учреждениях и организациях для автоматизации процесса учёта посещаемости.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Сзелиски Р. *Компьютерное зрение: алгоритмы и приложения*. – Springer, 2011. – 934 с. [Электронный ресурс]. – Режим доступа: https://link.springer.com/book/10.1007/978-1-84882-935-0 (дата обращения: 14.11.2024).

2. Bergmann D. What is Fine-Tuning? – *IBM Think Blog*, 15.03.2024. [Электронный ресурс]. – Режим доступа: https://www.ibm.com/think/topics/fine-tuning (дата обращения: 21.01.2025).

3. Chen S., Liu Y., Gao X., Han Z. MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices // *Proc. Chinese Conf. on Biometric Recognition (CCBR)*. – 2018.

4. Chen X. et al. Symbolic Discovery of Optimization Algorithms. – *arXiv preprint* arXiv:2302.06675. – 2023. – 30 с. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/2302.06675 (дата обращения: 17.12.2024).

5. Deng J., Guo J., Ververas E., Kotsia I., Zafeiriou S. RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild // *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. – 2020. – P. 5205–5214.

6. Deng J., Guo J., Xue N., Zafeiriou S. ArcFace: Additive Angular Margin Loss for Deep Face Recognition // *Proc. IEEE/CVF Conf. on Computer Vision and Pattern Recognition*. – 2019. – P. 4690–4699.

7. Goodfellow I., Bengio Y., Courville A. *Deep Learning*. – MIT Press, 2016. – 775 p. [Электронный ресурс]. – Режим доступа: https://www.deeplearningbook.org (дата обращения: 10.10.2025).

8. Grand Challenge of 106-Point Facial Landmark Localization. – *arXiv:2009.07661 [cs.CV]*. – 2020.

9. Kingma D.P., Ba J. Adam: A Method for Stochastic Optimization // *Proc. ICLR*. – 2015. – 15 с. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1412.6980 (дата обращения: 27.04.2025).

10. Krizhevsky A., Sutskever I., Hinton G.E. ImageNet Classification with Deep Convolutional Neural Networks // *Communications of the ACM*. – 2017. – Vol. 60, No. 6. – P. 84–90. [Электронный ресурс]. – Режим доступа: https://dl.acm.org/doi/10.1145/3065386 (дата обращения: 22.10.2025).

11. LeCun Y. et al. Gradient-Based Learning Applied to Document Recognition // *Proc. of the IEEE*. – 1998. – Vol. 86, No. 11. – P. 2278–2324. [Электронный ресурс]. – Режим доступа: https://vision.stanford.edu/cs598_spring07/papers/Lecun98.pdf (дата обращения: 17.09.2024).

12. Liu Z. et al. A ConvNet for the 2020s // *Proc. CVPR*. – 2022. – P. 11976–11986. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/2201.03545 (дата обращения: 13.02.2025).

13. NIST. Face Recognition Vendor Test (FRVT) Ongoing, Part 2: Identification (Report). [Электронный ресурс]. – Режим доступа: https://www.csis.org/blogs/strategic-technologies-blog/how-accurate-are-facial-recognition-systems-and-why-does-it (дата обращения: 29.04.2025).

14. Redmon J., Divvala S., Girshick R., Farhadi A. You Only Look Once: Unified, Real-Time Object Detection // *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*. – 2016. – P. 779.

15. Rosenblatt F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain // *Psychological Review*. – 1958. – Vol. 65, No. 6. – P. 386–408. [Электронный ресурс]. – Режим доступа: https://www.ling.upenn.edu/courses/cogs501/Rosenblatt1958.pdf (дата обращения: 12.10.2025).

16. Rumelhart D.E., Hinton G.E., Williams R.J. Learning Representations by Back-Propagating Errors // *Nature*. – 1986. – Vol. 323. – P. 533–536. [Электронный ресурс]. – Режим доступа: https://www.nature.com/articles/323533a0 (дата обращения: 12.04.2025).

17. Schroff F., Kalenichenko D., Philbin J. FaceNet: A Unified Embedding for Face Recognition and Clustering // *Proc. IEEE Conf. on Computer Vision and Pattern Recognition*. – 2015. – P. 815–823.

18. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition // *Proc. ICLR*. – 2015. – 14 с. [Электронный ресурс]. – Режим доступа: https://arxiv.org/abs/1409.1556 (дата обращения: 25.09.2024).

19. Tan M., Le Q. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // *Proc. ICML*. – 2019. – P. 6105–6114. [Электронный ресурс]. – Режим доступа: https://proceedings.mlr.press/v97/tan19a.html (дата обращения: 21.11.2025).

20. Zhang K., Zhang Z., Li Zh., Qiao Y. Joint Face Detection and Alignment using Multi-task Cascaded Convolutional Networks // *IEEE Signal Processing Letters*. – 2016. – Vol. 23, No. 10. – P. 1499–1503.

ПРИЛОЖЕНИЕ А

Код обучения итоговой нейронной сети

```
import glob
from PIL import Image
import random
import numpy as np
from torchvision import transforms as T
from torch.utils.data import Dataset
from torchvision.datasets import ImageFolder
import torch
import os
import pytorch_lightning as pl
import timm

def image_grid(imgs, rows, cols):
    assert len(imgs) == rows*cols

    w, h = imgs[0].size
    grid = Image.new('RGB', size=(cols*w, rows*h))
    grid_w, grid_h = grid.size

    for i, img in enumerate(imgs):
        grid.paste(img, box=(i%cols*w, i//cols*h))
    return grid
def center_crop(img, f=0.5):
    width = img.size[0]
    height = img.size[1]

    new_width = f * min(width, height)

    new_height = f * min(width, height)

    left = int(np.ceil((width - new_width) / 2))
    right = width - int(np.floor((width - new_width) / 2))

    top = int(np.ceil((height - new_height) / 2))
    bottom = height - int(np.floor((height - new_height) / 2))

    return img.crop((left, top, right, bottom))
```

```
class FaceClassificationLFWDataset(ImageFolder):
  def __getitem__(self, index: int):
      image, label = super().__getitem__(index)
      transform = T.Compose([
          T.ToTensor(),
          T.Resize((128, 128)),
          T.Normalize(
          mean = [0.485, 0.456, 0.406],
          std = [0.229, 0.224, 0.225]),
      ])
      image = center_crop(image)
      image = transform(image)
      return {
          "image": image,
          "label": torch.tensor([label]).int()
      }

dataset = FaceClassificationLFWDataset("lfw_funneled")

class FaceRecModelTIMM(torch.nn.Module):
  def __init__(self, model_name: str, n_classes: int):
    super().__init__()
    self.model = timm.create_model(
        model_name,
        True,
        num_classes=n_classes
    )

  def forward_features(self, x):
    return self.model.forward_features(x)

  def forward(self, x):
    return self.model(x)

class ClassficationLitModel(pl.LightningModule):
  def __init__(self, model, loader):
    super().__init__()
    self.model = model
    self.loader = loader
    self.optimizer = None
    self.loss = torch.nn.CrossEntropyLoss()
```

```
  def configure_optimizers(self):
     return torch.optim.Adam(self.model.parameters(), lr=10e-3)

  def forward(self, x):
    return self.model.forward(x)

  def forward_features(self, x):
    return self.model.forward_features(x)

  def training_step(self, batch, batch_idx):
        X, y = batch['image'], batch['label']
        X, y = X.cuda(), y.cuda()
        out = self.forward(X.float())
        loss_ = self.loss(out, y.flatten().to(torch.long))
        return {'loss': loss_}

  def train_dataloader(self, ):

        return self.loader

loader = torch.utils.data.DataLoader(
    dataset, batch_size=8, shuffle=True
)

model = FaceRecModelTIMM(
    'mobilenetv3_large_100',
    len(os.listdir('lfw_funneled'))
)

lit = ClassficationLitModel(model, loader)

trainer = pl.Trainer(
    accelerator='gpu',
    precision=16,
    max_epochs=40,
    accumulate_grad_batches=3,
    enable_checkpointing=False,
)

trainer.fit(lit)
```