МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«РОССИЙСКИЙ УНИВЕРСИТЕТ ДРУЖБЫ НАРОДОВ»

Факультет физико-математических и естественных наук

Кафедра информационных технологий

«Допустить к защите»
Заведующий кафедрой
информационных технологий
д.ф.-м.н.

_____ Ю.Н. Орлов

«____»_____ 2021Г.

Выпускная квалификационная работа бакалавра

Направление 02.03.02 «Фундаментальная информатика и информационные технологии»

ТЕМА «Разработка мобильного приложения под iOS с применением технологий машинного обучения»

Выполнил студент **Леонова Алина Дмитриевна**

(Фамилия, имя, отчество)

Группа НФИбд-02-17

Студ. билет № 1032172731

Руководитель выпускной
квалификационной работы

Шорохов Сергей Геннадьевич,
к.ф.-м.н, доцент кафедры ИТ
(Ф.И.О., степень, звание, должность)

_____
(Подпись)

Автор _____
(Подпись)

г. Москва

2021 г.

Федеральное государственное автономное образовательное учреждение высшего образования
«Российский университет дружбы народов»

АННОТАЦИЯ
выпускной квалификационной работы

Леонова Алина Дмитриевна

(фамилия, имя, отчество)

на тему: Разработка мобильного приложения под iOS с применением технологий машинного обучения

В выпускной квалификационной работе бакалавра были рассмотрены вопросы использования методов и средств машинного обучения при разработке мобильных приложений. Возможности различных библиотек и фреймворков, обеспечивающих реализацию технологий машинного обучения на языках Swift и Python, а также их использование при разработке приложений для iOS в среде Xcode. В рамках работы рассматривается вся технологическая цепочка от подготовки и обучения модели до её использования в мобильном приложении.

Практические результаты работы включают разработку структуры мобильного приложения для распознавания изображений продуктов питания, перенос в формат Core ML обученной модели, подготовленной с помощью приложения на языке Python, и разработку прототипа мобильного приложения для платформы iOS.

Автор ВКР _____ Леонова А.Д.

(Подпись) (ФИО)

Оглавление

Введение

В работе рассматриваются вопросы использования методов и средств машинного обучения при разработке мобильных приложений.

Накопление больших объёмов данных в информационных системах стало причиной роста интереса к их изучению в целях выявления различных закономерностей. В свою очередь, рост доступных вычислительных мощностей сделал возможным практическое повсеместное применение подходов, ранее представлявших в основном теоретический интерес. Одним из таких подходов стало машинное обучение — раздел искусственного интеллекта, целью которого является создание алгоритмов, способных к обучению на основе поступающих данных, причём чем разнообразнее эти данные, тем проще становится обнаружение закономерностей, и тем точнее полученный результат.

Эта тенденция сочетается со взрывным ростом популярности мобильных устройств — так, например, уже в 2016 году мобильный трафик превысил трафик с традиционных компьютеров. Появление в устройствах ведущих производителей аппаратных реализаций и программных интерфейсов, ускоряющих применение технологий машинного обучения, сделало их доступными для рядовых пользователей.

В рамках дипломной работы рассматривается вся технологическая цепочка от подготовки и обучения модели до её использования в мобильном приложении. В качестве примера практической реализации использовалась задача классификация изображений.

Достижение поставленной цели требует решения следующих основных задач:

- изучение методов машинного обучения и, в частности, подходов к классификации изображений;
- выбор и реализация конкретного метода классификации;
- выбор платформы и средств разработки;
- разработка мобильного приложения с использованием выбранных средств.

Структура выпускной работы

Выпускная работа состоит из трёх разделов. В первом разделе описываются основные теоретические аспекты: история машинного обучения, решаемые задачи и их классификация, а также методы их решения; кроме того, рассматриваются проблемы в разработке кроссплатформенных и нативных мобильных приложений. По итогам первого раздела формулируется постановка задачи исследования в рамках выпускной квалификационной работы бакалавра.

Второй раздел работы посвящён средствам разработки мобильных приложений и средствам, реализующим машинное обучение. Представлено сравнение популярных фреймворков для языка Python, а также описание архитектур нейронных сетей, ориентированных на работу с изображениями.

В заключительном разделе описана практическая часть работы: структура разработанного приложения, процесс создания модели машинного обучения, описание процесса реализации интерфейса приложения, а также оценка результата и перспективы развития.

Апробация работы

Предварительные результаты работы были представлены на конференции «Information and Telecommunication Technologies and Mathematical Modeling of High-Tech Systems» ITTMM 2021 (22 апреля 2021 г.) и опубликованы в сборнике материалов конференции [1].

Также подана заявка на получение свидетельства о государственной регистрации программы для ЭВМ. Леонова А.Д., Шорохов С.Г. «Программа для анализа эффективности архитектур нейронных сетей на основе построенных моделей машинного обучения на заданном наборе изображений NeuroTestAL».

1. Технологии машинного обучения в современных информационных системах

1.1. Задачи машинного обучения

1.1.1. Эволюция подходов к машинному обучению

Термин «машинное обучение» был введён в 1959 году одним из пионеров в области исследований искусственного интеллекта (ИИ) Артуром Сэмюэлем как определение процесса, в результате которого информационные системы способны проявлять поведение, не запрограммированное в них изначально. Уже в первые годы развития ИИ ряд исследователей активно занимался задачей использования данных для обучения машин. К тому же времени относится разработка Фрэнком Розенблаттом первого персептрона, заложившего основы технологии нейронных сетей, которая в настоящее время неразрывно связана с понятием ИИ.

Затем на некоторое время основной интерес исследователей ИИ переключился на логический подход, основанный на знаниях, что привело к доминированию в 1980-х годах в ИИ экспертных систем. Работы, связанные со статистическими исследованиями, распознаванием образов и нейронными сетями практически вышли за рамки ИИ.

Возобновление интереса к машинному обучению относится к концу 90-х годов, чему способствовал рост доступности оцифрованной информации и развитие телекоммуникационных технологий. Это позволило сформулировать новые цели для ИИ, ориентированные на разрешимые проблемы практического характера.

И, наконец, на всплеск практического интереса к машинному обучению в последние десятилетия повлияли растущие объёмы и разнообразие доступных данных, сочетающиеся с возросшими и подешевевшими вычислительными мощностями. Следствием этого стало активное развитие программных и аппаратных архитектур, обеспечивающих эффективную реализацию алгоритмов машинного обучения. В настоящее время машинное обучение на больших объёмах данных превратило нейронные сети, ранее имевшие довольно ограниченные возможности, в эффективный практический инструмент.

Машинное обучение активно использует методы математической статистики и теории вероятностей, методов оптимизации и других классических математических дисциплин, но также является и инженерной дисциплиной, активно использующей различные эвристики, эксперименты на модельных и реальных данных. Многие методы машинного обучения тесно связаны с интеллектуальным анализом данных.

1.1.2. Общая постановка задачи машинного обучения

Формально можно выделить два типа обучения. Первое из них, дедуктивное, подразумевает перенос в машинную форму знаний экспертов и традиционно относится к области экспертных систем. Второе, индуктивное, к которому фактически и сводится машинное обучение, занимается анализом собранных эмпирических данных (прецедентов) в целях выявления некоторых общих закономерностей [2].

Задача обучения по прецедентам формулируется следующим образом. Имеется конечное множество объектов или ситуаций, для каждого из которых известны некоторые данные, также называемые описанием прецедента. Совокупность описаний прецедентов называется обучающей выборкой. Задачей обучения является выявление (или восстановление) зависимостей или закономерностей, присущей как полученной обучающей выборке, так и другим прецедентам, в том числе ещё не наблюдаемым (так называемая способность к обобщению). Прецеденты обычно описываются признаками, т.е. совокупностями заданного количества показателей. Описания могут быть представлены как числовые векторы с размерностью, соответствующей количеству показателей, либо могут быть описаны в более сложной форме временных рядов, изображений, аудио- и видеорядов и т.п.

Решение задачи обучения по прецедентам требует формирования модели зависимости, которую требуется восстановить. Оценка соответствия модели полученным данным получается с помощью некоторого заданного функционала качества. Наконец, целью алгоритма обучения является определение таких значений параметров модели, которые дадут оптимальное значение функционала качества на имеющейся обучающей выборке.

1.1.3. Классификация задач машинного обучения

Наиболее распространённым классом задач является **обучение с учителем (контролируемое)**, которое можно считать обобщением классической задачи аппроксимации функций, и в рамках которого прецеденты образуют пары «объект/ответ», между которыми существует некоторая неизвестная зависимость. Задачей обучения в данном случае является восстановление этой зависимости путём построения алгоритма, обеспечивающего получение достаточно точного классифицирующего ответа для любого возможного входного объекта. В качестве функционала качества обычно берётся средняя ошибка полученных ответов по всем объектам выборки.

В рамках обучения с учителем выделяются следующие задачи:

- **классификация**: конечное множество допустимых ответов образует совокупность классов, каждому из которых присвоена своя метка;
- **регрессия**: допустимый ответ представляет собой действительное число либо вектор;
- **ранжирование**: ответы получаются сразу на множестве объектов и далее сортируются; обычно сводится к задачам классификации или регрессии;
- **прогнозирование**: ответы образуют временные ряды, требующие продолжения, т.е прогноза на будущее; также часто сводится к классификации или регрессии.

Второй крупный класс задач — **обучение без учителя (неконтролируемое),** при котором ответы не заданы, т.е. требуется найти зависимости между объектами. Наиболее популярные задачи этой группы:

- **кластеризация**: группировка объектов в кластеры на основе информации об их попарном сходстве; функционал качества задаётся, например, как отношение средних внутрикластерных и межкластерных расстояний;
- **уменьшение размерности:** позволяет свести большое число признаков к меньшему (как правило, 2-3), например, для удобства их последующей визуализации без потери существенной информации об объектах выборки;
- **выявление аномалий (фильтрация выбросов):** обеспечивает обнаружение небольшого числа нетипичных объектов (например, отсечение шумов или

выявление случаев мошенничества).

Частичное обучение объединяет в себе черты обучения с учителем и без учителя. Как и в обучении с учителем, каждому прецеденту соответствует пара «объект/ответ», но ответы известны не для всех прецедентов.

Трансдуктивное обучение подразумевает необходимость предсказания только для прецедентов из тестовой выборки; как правило, сводится к задаче частичного обучения.

При **обучении с подкреплением** для каждого прецедента имеется пара «ситуация/принятое решение», играющая роль объекта. Ответы определяются как значения функционала качества, который оценивает эффективность принятого решения. Таким образом, целью является поиск стратегии действий в некотором окружении для максимизации долговременного выигрыша.

1.2. Методы решения задач машинного обучения

За всю историю своего развития машинное обучение интегрировало в себя самые разнообразные подходы, как основанные на классической математической статистики, так и носящие более эмпирический характер. Рассмотрим наиболее популярные из них.

1.2.1. Статистическая классификация

Байесовские методы

Этот подход к классификации основан на фундаментальной теореме Байеса, позволяющей определить вероятность одного из статистически взаимосвязаных событий: для двух заданных событий A и B, условная вероятность A, при условии, что B верно, выражается формулой (1).

$$\mathrm{P}(A|B) = \frac{P(B|A)\,P(A)}{P(B)}\text{, где } P(B) \neq 0. \qquad (1)$$

На практике плотности распределения классов, как правило, приходится восстанавливать по обучающей выборке, поскольку априори они не известны. Поскольку такое восстановление возможно лишь с некоторой погрешностью, байесовский алгоритм перестаёт быть оптимальным. А в случае небольших объёмов выборки усиливается риск получения эффекта переобучения за счёт подгона распределения под конкретные данные.

В зависимости от подхода к оценке плотности, выделяются группы байесовских классификаторов, использующих параметрическую и непараметрическую оценку плотности, а также определение итоговой плотности как смеси параметрических плотностей.

Наконец, выделяют так называемый наивный байесовский классификатор, основанный на предположении о статистической независимости признаков. Он может быть как параметрическим, так и непараметрическим.

Байесовская сеть

Байесовская сеть доверия представляет собой ориентированный ациклический граф, каждой вершине которого соответствует случайная переменная, а дуги графа соответствуют отношениям условной независимости между этими переменными.

Выделяют следующие типы байесовских сетей:

- дискретная: переменные представляют собой дискретные случайные величины;
- динамическая: моделирует последовательности переменных;
- гибридная: включает как дискретные переменные, так и непрерывные;
- причинно-следственная (causal bayesian): её дуги кодируют не только отношения условной независимости, но и отношения причинности.

Скрытая марковская цепь

В скрытой марковской модели, которую можно интерпретировать как частный случай простой байесовской сети, рассматриваемый процесс считается марковским, т.е. вероятности переходов в другие состояния зависят только от текущего состояния. При этом текущее состояние системы неизвестно (поэтому модель и называется скрытой). Популярные применения — распознавание текста, речи, движений.

1.2.2. Метрические алгоритмы классификации

Метрические алгоритмы классификации используются в ситуациях, допускающих задание матрицы попарных расстояний между классифицируемыми объектами. Классификация объектов по сходству исходит из гипотезы

компактности, согласно которой классы образуют компактно локализованные подмножества в пространстве объектов.

К самым известным метрическим алгоритмам относятся метод ближайшего соседа (классификация объекта соответствует ближайшему объекту обучающей выборки) и метод k ближайших соседей (соответствует большей части ближайших k объектов в многомерном пространстве признаков).

1.2.3. Линейные классификаторы

Эти методы предполагают явное построение разделяющей поверхности в пространстве объектов. Методы их обучения различаются выбором функции L(M), способом регуляризации и выбором численного метода оптимизации.

Можно выделить следующие методы:

- **Линейный дискриминант Фишера:** минимизирует евклидово расстояние между векторами восстановленных и фактических значений зависимой переменной.
- **Логистическая регрессия** — соответствует логарифмической аппроксимации:

$$[M < 0] \leq log_2(1 + e^{-M}) \qquad (2)$$

- **Метод опорных векторов** (SVM — Support Vector Machine) — соответствует кусочно-линейной аппроксимации:

$$[M < 0] \leq (1 - M)_+ \qquad (3)$$

- **Экспоненциальная аппроксимация** — используется в алгоритмах бустинга, в частности, в алгоритме AdaBoost:

$$[M < 0] \leq \exp(-M) \qquad (4)$$

1.2.4. Логические и ансамблевые методы

Логические алгоритмы, такие как решающее дерево, лес, дерево регрессии и т.п., представляют собой композиции простых, легко интерпретируемых правил. Например, в случае решающего дерева вводится понятие решающего правила как функции от объекта, позволяющей определить, какой из дочерних вершин принадлежит классифицируемый объект.

Ансамблевые методы, ансамбли или алгоритмические композиции были предложены для улучшения стабильности и точности алгоритмов машинного обучения. Ансамбль методов комбинирует несколько алгоритмов в расчёте, что они исправят ошибки друг друга. Основная гипотеза заключается в том, что при удачном сочетании слабых моделей можно получить более точную и/или надёжную модель.

Можно выделить три техники построения ансамблей:

Бэггинг — использует композиции независимо обучаемых алгоритмов. Результат определяется путём усреднения результатов отдельных предсказателей — например, может формироваться как взвешенное среднее, нормальное среднее или голосованием большинства.

Бустинг — использует не независимое, а последовательное построение предсказателей. То есть сперва обучается первый алгоритм, отмечаются его ошибки. Затем обучается второй алгоритм, уже с учётом ошибок первого. И так далее по цепочке.

Стекинг — сперва обучаются несколько алгоритмов, затем, с учётом полученных результатов, обучается последний контрольный алгоритм. Кроме того, возможно расширение — многоуровневый стекинг. Например, 3-х уровневый стекинг: обучение группы алгоритмов 1-го уровня, обучение группы алгоритмов 2-го уровня с учётом ошибок первой группы и, наконец, обучение модели 3-его уровня с учётом ошибок второй группы.

Случайный лес

Случайный лес (Random Forest) представляет собой множество решающих деревьев. Это один из немногих универсальных алгоритмов машинного обучения, который широко часто используется для решения практически всех перечисленных задач. В задаче регрессии ответы решающих деревьев усредняются, в задаче классификации решение определяется большинством «голосов».

Градиентный бустинг

Градиентный бустинг (GBM) ориентирован на решение задач классификации и регрессии. Модель предсказания строится в форме ансамбля слабых предсказывающих моделей, в роли которых, как правило, выступают деревья решений.

Главную роль в популяризации градиентного бустинга сыграли соревнования по машинному обучению, а эффективно реализующая его библиотека XGBoost быстро завоевала популярность вскоре после своего появления.

1.2.5. Глубокое обучение и нейронные сети

Общие принципы. Понятие нейрона

Глубокое обучение — это метод машинного обучения с помощью искусственной нейронной сети, имитирующей человеческий мозг, состоящий из нейронов, организованных в сеть.

Нейронная сеть или просто нейросеть — это математическая модель, строящаяся на принципах функционирования биологических нейросетей нервных клеток живых организмов, и реализованная в программном и/или аппаратном виде.

Они образуются множеством соединений между простыми процессорами (нейронами сети), а обучение сводится к построению оптимальной структуры связей и настройке параметров этих связей. Некоторые нейроны сети представляют собой входы, а некоторые — выходы сети. Задавая значения на входе сети, на выходе получается отклик, таким образом входной вектор преобразуется в выходной (Рисунок 1). Управление процессом преобразования осуществляется путём задания весов связей сети, которые формируются в процессе обучения. Таким образом, обучение позволяет выявить зависимости между входными и выходными векторами, формируя способность сети к прогнозированию.

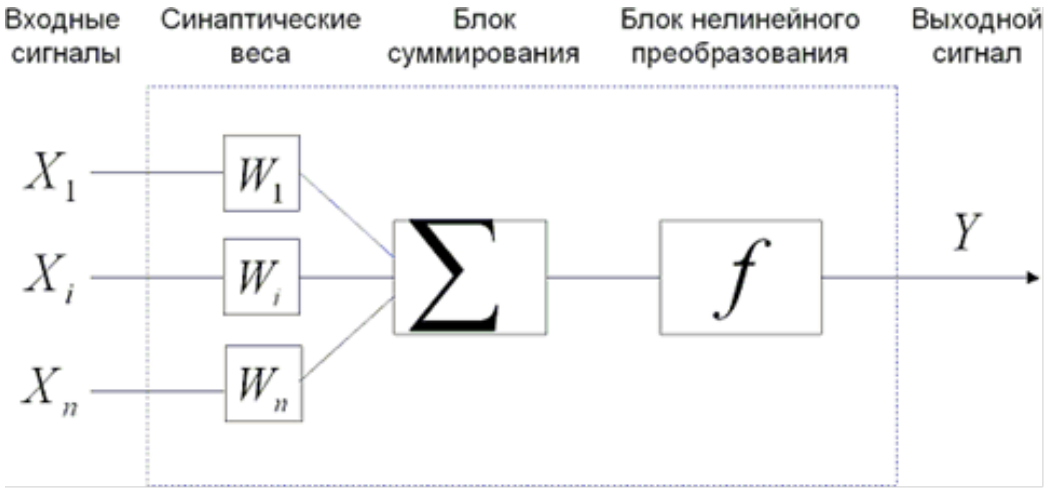

Рисунок 1 — Модель искусственного нейрона

В случае обработки изображений в качестве сигналов могут рассматриваться отдельные участки изображений, вплоть до нескольких пикселей. Каждый входной сигнал умножается на соответствующий ему вес Wi. Уровень активации нейрона определяется как сумма всех полученных произведений. Далее полученный сигнал поступает на вход функции активации и преобразуется, после чего передаётся на вход нейронов следующего слоя. Функция активации задаётся как функция, приводящая сигнал нейрона к определённому диапазону, например, сигмоида. Сигналы, поступающие на вход нейросети, должны быть нормализованы, для этого значения исходного ряда пересчитываются как отношение разности значения с минимальным к размаху, таким образом, исходный диапазон значений ряда приводится к диапазону от 0 до 1.

Виды сетей

К настоящему времени было предложено множество архитектур нейронных сетей, которые можно классифицировать по следующим критериям:

- по топологии: полносвязные, многослойные, слабосвязные;
- по способу обучения: с учителем, без учителя, с подкреплением;
- по модели: прямого распространения, рекуррентные (с обратными связями), основанные на радиально-базисных функциях, сети Кохонена, свёрточные;
- по способу настройки весовых коэффициентов: с фиксированными и динамическими коэффициентами.

Одной из первых моделей нейронных сетей, заложившей основный все этой теории, стал персептрон, созданный Фрэнком Розенблаттом с целью построения модели мозга [3]. Передающая сеть, образующая персептрон, включает генераторы сигнала трёх типов: сенсорных (S-элементы, вырабатывающие сигналы на основе внешнего воздействия), ассоциативных (A-элементы, выдающие сигнал, когда сумма входов превышает пороговое значение) и реагирующие (R-элементы, выдающие +1, если сумма входов положительная, и -1, если сумма входов отрицательна). Схема элементарного персептрона представлена на рисунке 2.

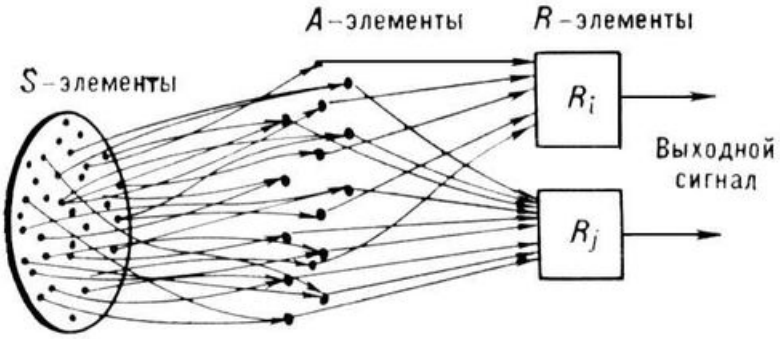

Рисунок 2 — Схема элементарного персептрона

Свёрточные сети

Остановимся подробнее на свёрточных сетях, активно используемых при распознавании и классификации изображений.

Свёрточные нейросети произвели революцию в распознавании образов и компьютерном зрении, кроме того, они успешно используются и для других сложных для компьютера человеческих задач, таких как распознавание речи и анализ текстов. Нередко свёрточные нейронные сети называют наиболее успешной моделью глубокого обучения.

Они появились в результате изучения особенностей зрительной коры головного мозга. Эксперименты Х. Дэвида и В. Торстена на кошках и обезьянах показали, что многие нейроны в зрительной коре имеют небольшое локальное рецепторное поле, а потому реагируют только на зрительные раздражители, находящиеся в ограниченной области поля зрения. Также одни нейроны реагируют исключительно на изображения вертикальных линий, а другие на линии с различными направлениями.

Таким образом, рецепторные поля разных нейронов в совокупности охватывают всё поле зрения. Кроме того, рецепторные поля нейронов могут быть разных размеров, нейроны с большим рецепторным полем реагируют на более сложные образы. Это привело к мысли, что существуют разные уровни нейронов и что нейроны более высокого уровня получают и учитывают выводы нейронов более низкого уровня.

Особенность свёрточной нейронной сети заключается в её принципиальной многослойности: нейроны в ней, как в зрительной коре, сгруппированы в слои

разных типов, при этом нейроны каждого слоя в свою очередь обладают своими особенностями и только некоторые из них связаны некоторыми другими из соседних слоёв, реагируя на различные особенности изображений:

- в **свёрточных** слоях активация нейронов следующего уровня формируется из линейной комбинации активаций нейронов предыдущего уровня за счёт умножения фрагмента на матрицу свёртки; результат каждой свёртки подаётся на вход некоторой нелинейной функции активации, фактически представляющей собой слой активации; обычно слой активации явно не выделяется, а логически объединяется со свёрточным слоем;
- в **субдискретизирующих** слоях активация нейронов следующего уровня воспроизводит активацию нейронов предыдущего уровня, но размеры изображения уменьшаются за счёт процедуры пулинга (подвыборки или субдискретизации), заключающейся в замене активации рядом расположенных нейронов на их максимум или их среднее.

Данные, выходящие из свёрточной сети, передаются полносвязному слою (или полносвязной сети, которая может в свою очередь состоять из нескольких слоёв), где осуществляется окончательная классификация самых абстрактных понятий, выявленных свёрточной сетью из исходного изображения (Рисунок 3).

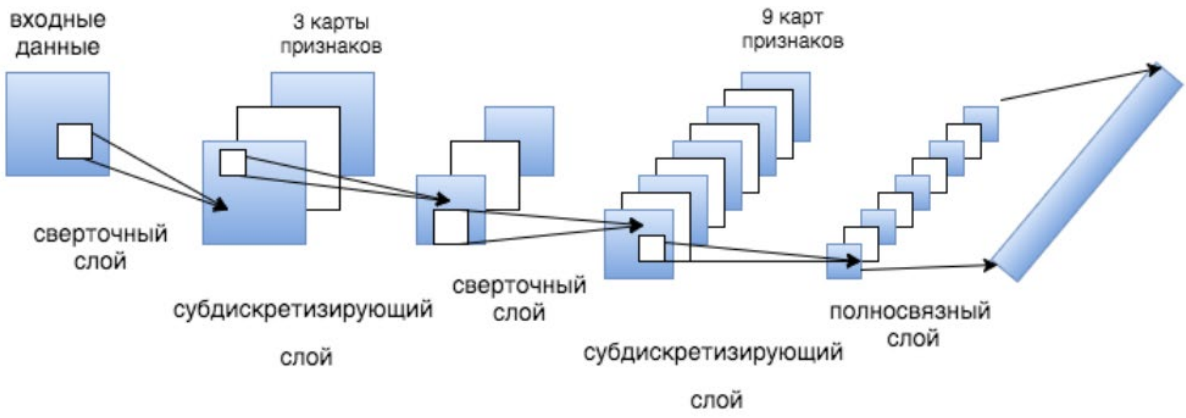

Рисунок 3 — Пример архитектуры свёрточной сети

Такая структура свёрточных сетей показала свою эффективность при работе с изображениями, поскольку позволяет распознавать визуально незначительно

отличающиеся фрагменты, расположенных в разных частях обрабатываемых изображений.

При обучении свёрточных сетей используются традиционные методы обучения нейронных сетей, как правило, метод обратного распространения ошибки. Для уменьшения эффекта переобучения сети популярно применение метода дропаута (исключения), заключающийся в случайном исключении определённой доли нейронов на разных эпохах обучения. Это приводит к приданию обученным нейронам большего веса и значительному увеличению скорости и качества обучения.

1.3. Методы разработки кроссплатформенных и мобильных приложений

1.3.1. Проблема переносимости программного обеспечения

Проблема адаптации программных систем для работы в окружении, отличающемся от первоначального целевого окружения, с той или иной степенью остроты существовала практически в течение всей истории развития вычислительной техники.

Задача кроссплатформенной разработки появилась от естественного желания разработчиков минимизировать усилия, требуемые для создания и поддержке своего программного обеспечения в сложившемся многообразии аппаратного и системного программного обеспечения. Кроссплатформенность реализуется с использованием инструментальных средств, обеспечивающих переносимость программного обеспечения на том или ином уровне.

Актуальность и способы решения этой задачи меняются с появлением новых платформ или с популяризацией и захватом доминирующих позиций отдельными платформами. Так, в течение длительного времени доминирующей платформой для потребительских систем была программная платформа Windows в сочетании с процессорами Intel (даже получившая общее название Wintel), для серверных систем предпочтение разработчиков было отдано различным вариантом Unix. На изменение ситуации в последние десятилетия существенно влияло появление мобильных систем, рост популярности Internet и «тонких клиентов», а для российских разработчиков также курс на импортозамещение и использование отечественных платформ.

1.3.2. Решение задачи переносимости

Можно выделить следующие подходы к решению задачи переносимости программного обеспечения.

Переносимость бинарного кода

В этой, наиболее комфортной для программистов ситуации, как исходная, так и целевая платформы поддерживают общий бинарный интерфейс приложений. Бинарная совместимость требует точного совпадение форматов исполняемых файлов и библиотек, а также совпадение функций программного интерфейса операционной системы. В результате исполняемый файл приложения не требует никакой дополнительной обработки или перекомпиляции.

Подобное благоприятное стечение обстоятельств происходит, как правило, лишь в случае родственных платформ и в основном в целях обеспечения обратной совместимости. Примером могут служить выполнение 32-битных Windows-приложений в 64-битных системах семейства Windows, а также выполнение мобильных iOS-приложений в операционной системе macOS на платформе M1/ARM. Также совместимость может обеспечиваться за счёт вспомогательных подсистем, компенсирующих различия в аппаратной платформе либо в API, примером могут служить подсистема Wine, позволяющая запускать Windows-приложения в Unix-системах либо встроенные в macOS динамические трансляторы Rosetta и Rosetta 2, используемые Apple при миграции своей основной аппаратной платформы с PowerPC на Intel и с Intel на M1/ARM.

Переносимость программного кода

Данный подход подразумевает перекомпиляцию исходного кода приложения без необходимости внесения в него изменений (либо с минимальными изменениями). Это требует поддержки целевыми платформами некого общего программного интерфейса, позволяющего абстрагировать особенности программно-аппаратной платформы, и реализованного в виде так называемой библиотеки среды выполнения (runtime library, RTL).

Одним из первых и наиболее распространённых стандартов такого рода стал переносимый интерфейс операционных систем POSIX (Portable Operating System

Interface), реализованный всеми современными Unix-системами, а также рядом других платформ.

Хотя POSIX успешно решает задачу базового взаимодействия с системой, отдельную проблему представляет построение пользовательских интерфейсов. Даже в случае стандартного для Unix интерфейса X Window существует ряд интерфейсных библиотек, оконных менеджеров и графических оболочек, свой набор интерфейсов имеет и основанная на Unix MacOS, и даже в Windows существует два интерфейсных API: для стандартных приложений и приложений Microsoft Store.

Возросшая популярность мобильных приложений ещё больше обострила ситуацию. Несмотря на то, что ряд мобильных операционных систем (RIM, Symbian, webOS, Windows Mobile, Windows Phone) успел потерять актуальность, по состоянию на 2021 год рынок поделён между двумя плохо совместимыми операционными системами Android и iOS (71.93% и 27.47% соответственно) [4]. Причём для каждой платформы требуется поддержка двух форм-факторов (планшетов и телефонов), обеспечение обратной совместимости с ранними версиями систем, а также (в случае Android) поддержка различных графических оболочек и аппаратных реализаций.

К наиболее популярным библиотекам, позволяющим реализовывать кроссплатформенный пользовательский интерфейс, относятся Qt, GTK+, wxWidgets, Tk. Все они в первую очередь ориентированы на десктопные приложения и имеют существенные проблемы при работе в мобильных системах. Большая часть мобильных переносимых интерфейсных решений в той или иной степени основана на виртуализации, рассмотренной в следующем разделе.

Также необходимо упомянуть некоторые специализированные библиотеки, такие CUDA, OpenCL, обеспечивающие распараллеленное решение математических задач на современных программно-аппаратных платформах; оптимизированную для распределённого окружения библиотеку поддержки нейронных сетей OpenNN; библиотеки для работы с мультимедийными средствами (OpenGL для графики, OpenAL для аудиоданных) и т.п.

Интерпретируемые языки и байт-код

В данном случае вместо преобразования исходного кода приложения в бинарный код целевой программно-аппаратной платформы задачу его последовательного выполнения берёт на себя интерпретатор. Таким образом, переносимость обеспечивается наличием на целевой платформе реализации соответствующего интерпретатора и набором транслируемых им системных API. Фактически речь идёт о создании для исполняемого приложения замкнутой среды выполнения, в предельном случае реализуемая с помощью специализированной виртуальной машины, такой как виртуальная Java-машина (JVM) либо движок V8, отвечающий за исполнение JavaScript в браузерах семейства Google Chrome.

Поскольку традиционная построчная интерпретация исходного кода приложения существенно уступает в производительности скомпилированному бинарному коду, популярным подходом стала трансляция исходного кода в промежуточный код процессора виртуальной машины (так называемый байт-код). Эта трансляция может осуществляться как на этапе сборки приложения, так и в момент его запуска (так называемая трансляция времени исполнения just in time translation, JIT).

В качестве примеров подобных средств можно назвать Pyhon, JavaScript, платформы Java и .NET. Получивший огромную популярность Python разрабатывался как универсальный язык с минималистичным синтаксисом, реализующий различные парадигмы программирования: допустимо использование объектно-ориентированного, структурного и функционального подходов. Удобные встроенные средства и множество специализированных библиотек сделали его популярным инструментом среди исследователей в области машинного обучения и искусственного интеллекта.

JavaScript, изначально созданный как инструмент обеспечения интерактивности web-страниц, далее стал популярным встроенным языком и для других приложений, обеспечивающим взаимодействие их встроенных объектов. Уже упоминавшийся движок V8 и построенная на его основе платформа Node.js позволили использовать JavaScript для разработке серверных web-приложений, а также автономных тонких клиентов.

Программная платформа Java, основанная на виртуальной Java-машине, стала одной из первых доступных массовому пользователю реализаций подхода с замкнутой средой выполнения. Несмотря на то, что производители web-браузеров отказались от первоначальной идеи включения Java на уровне web-страниц, Java остаётся популярным средством разработки кроссплатформенных приложений. Кроме того, на платформе Android используется модифицированная версия JVM как основа исполнения приложений (Dalvik Virtual Machine для ранних версий и Android Runtime начиная с Android 4.4).

Платформа .NET, созданная Microsoft в 2002 году, использует принципы, аналогичные Java, но изначально была ориентирована на одну платформу (Windows), хотя и в сочетании с несколькими языками (C++, C#, Visual Basic, F# с возможностью расширения списка). В дальнейшем были созданы сторонние кроссплатформенные реализации .NET (например, платформа Mono), чуть позже Microsoft создала свою кроссплатформенную реализацию .NET Core, в конце 2020 года окончательно реинтегрированную в основную платформу .NET 5.

Перечисленные средства были в основном ориентированы на решение проблем, связанных с разработкой кроссплатформенных серверных и десктопных приложений. Android использует вариант Java лишь для абстрагирования от конкретной аппаратной платформы, а iOS не предполагает использование средств виртуализации на уровне системы. Тем не менее, наиболее популярные средства разработки кроссплатформенных мобильных приложений относятся именно к этому классу. К ним относятся:

- Flutter: кросс-платформенная библиотека для построения интерфейсов, разработанная Google; поддерживаемые платформы — Android, iOS, Linux, macOS, Windows; для работы используется виртуальная машина Dart;
- Xamarin: кросс-платформенная библиотека, созданная разработчиками Mono и ориентированная на C#; набор поддерживаемых платформ аналогичен Fluter; для работы используется кросс-платформенная реализация .NET;
- Unity: кросс-платформенная среда разработки компьютерных игр, ориентированная на C# и поддерживающая свыше 25 платформ.

Тонкие клиенты

Тонкие клиенты получили широкое распространение с ростом популярности Internet, хотя их идея восходит ещё к первым «большим компьютерам», доступ к которым осуществляется в терминальном режиме. Под современными тонкими клиентами понимаются программные решения, визуализирующие информацию, обработка которой преимущественно происходит на удалённом сервере. В то же время, тонкие клиенты могут обладать относительной автономностью — например, современные браузерные приложения, использующие возможности JavaScript и HTML5, способны реализовывать достаточно сложную функциональность. Примером могут служить приложения семейства Google Apps, а также первые версии приложений для iOS, которые предлагалось реализовывать с помощью HTML5. Основные их недостатки с точки зрения разработки мобильных приложений — необходимость постоянного сетевого подключения и ограниченный доступ к возможностям конкретной аппаратной платформы.

1.3.3. Недостатки кроссплатформенных решений и необходимость использования нативных реализаций

Несмотря на то, что рассмотренные в предыдущих разделах кроссплатформенные решения упрощают разработку приложений, готовых к исполнению на различных платформах, они имеют и ряд недостатков, наиболее остро проявляющихся в случае мобильных приложений, ориентированных на использование наиболее актуальных возможностей платформы.

Во-первых, все они в той или иной степени используют средства виртуализации либо вспомогательные библиотеки, скрывающие API операционной системы от прикладного программного кода. Это неизбежно приводит к возрастанию нагрузки на процессор и потребления энергии, т.е. к снижению производительности приложения и автономности устройства.

Что более критично в контексте рассматриваемой задачи, кроссплатформенные средства мобильной разработки неизбежно ограничивают возможности приложений, предоставляя некий усреднённый API, ориентированный в первую очередь на построение пользовательского интерфейса.

В отличие от кроссплатформенных, нативные решения (использующие родные для данной платформы средства разработки, такие как Java для Android или Swift для iOS), обеспечивают полный доступ ко всем возможностям платформы, позволяют произвольно настраивать интерфейс и лишены любых проблем с производительностью. С другой стороны, для охвата пользователей хотя бы двух популярных платформ потребуется создание двух отдельных приложений, что потребует больше времени, усилий и ресурсов [5].

Поэтому наиболее популярным решением является комбинированный подход, при котором для реализации пользовательского интерфейса используются кроссплатформенные средства, а доступ к специфичным возможностям платформы, таким, как использование аппаратного ускорения операций, характерных для машинного обучения, обеспечивается за счёт использования отдельных компонентов, реализованных нативными средствами.

1.4. Постановка задачи

С учётом изложенного, цель работы может быть сформулирована как разработка мобильного приложения, реализующего классификацию изображений на основе рассмотренных методов и средств, реализующих технологию машинного обучения.

Для этого необходимо решить следующие задачи:

- рассмотреть инструментальные средства, реализующие методы машинного обучения;
- выбрать конкретный метод классификации и реализовать использующую его модель машинного обучения;
- обосновать выбор платформы и средств разработки мобильного приложения;
- изучить используемые на выбранной платформе API для машинного обучения;
- разработать мобильное приложение с использованием выбранных средств.

2. Методы и средства разработки мобильных приложений, использующих машинное обучение

Разработка мобильных приложений, использующих машинное обучение, разбивается на три основных этапа:

- подготовка и обучение модели;
- конвертация модели в вид, пригодный для переноса на мобильную платформу, если она создавалась универсальными средствами, не ориентированными на конкретную платформу;
- реализация итогового мобильного приложения.

Рассмотрим подробнее средства разработки, характерные для этих этапов.

2.1. Специфика подготовки данных для машинного обучения

Данные — наиболее важный элемент любого обучения. Обучение модели машинного обучения невозможно без данных, значимых для рассматриваемой задачи и представленных в правильном масштабе и формате.

2.1.1. Аугментация данных

Для достижения действительно хороших результатов глубокие сети должны обучаться на огромном объеме данных. Под аугментацией данных (data augmentation) понимается методика подготовки дополнительных обучающих данных на основе существующих данных путём внесения в них небольших изменений. Подобные методы сильно помогают при работе с задачей классификации изображений. Небольшие модификации изображения в процессе обучения не затрачивают много ресурсов, но в результате значительно экономят память в сравнении с тем, сколько бы потребовалось хранить информации в случае обучения на таком же объёме данных без использования аугментации. Кроме того, эти методы особенно полезны для задачи распознавания объектов, где перед обучением требуются проводить разметку изображений.

Методы аугментации можно поделить на следующие типы:

- операции с цветом (изменение цветовых каналов, например, увеличение контраста или яркости);

- геометрические операции (поворот изображения, отражение изображения и другие);
- операции с объектами (поворот объекта, отражение объекта и другие)

В аргументах функций для аугментации задаются коэффициенты или диапазоны, в которых будет случайно генерироваться значения коэффициентов, определяющих те или иные модификации изображении, например, для изменения цвета, поворота, зеркального отражения (по горизонтали) или случайной обрезки изображения. Таким образом, применение различных комбинаций подобных изменений даёт возможность в разы увеличить число обучающих данных. Во множестве исследований именно применение подобных методов преподносится в качестве объяснения достигнутых успехов. Например, благодаря данному подходу модель научилась идентифицировать объекты независимо от яркости и цвета освещения [6].

2.1.2. Предобработка данных

Для повышения эффективности и обеспечения корректного применения в дальнейшем анализе применяются методы предварительной обработки данных (data preprocessing).

В целом наиболее распространённые методы предобработки данных:

- **стандартизация** — преобразование значений набора данных, позволяющее получить распределение с нулевым средним и стандартным отклонением, равным 1:

$$z_i = \frac{x_i - \bar{X}}{\sigma_x}, \qquad (5)$$

где x_i — исходное значение признака, $\bar{X}$ и σ_x — соответственно, среднее значение и стандартное отклонение признака. В стандартизированных шкалах среднее значение величин $\bar{Z}=0$, стандартное отклонение $\sigma_z=1$.

- **нормализация (или масштабирование)** — изменение масштаба каждой строки данных, чтобы привести все данные к некоторой общей шкале без потери информации, чаще всего приводят к диапазонам [0,1] и [-1,1]. Формула для приведения набора значений к произвольному диапазону [a, b]:

$$X' = a + \frac{x - x_{min}}{x_{max} - x_{min}}(b - a) \quad (6)$$

Неприменимо к категориальным данным. Это полезно в наборе разреженных данных, где присутствует много нулей. Частный случай — бинаризация — преобразование данных в двоичные в зависимости от заданного порогового значения (значения ниже порогового преобразуются в 0, а выше — в 1).

В работе с изображениями целью предобработки обычно является удаление шумов и выявление наиболее важной части изображения, но в зависимости от исследуемой задачи и набора данных эти методы могут сильно отличаться. Например, при работе над задачей распознавания какого-то широкого диапазона, все обучающие данные часто можно поделить на категории классов, к которым целесообразно было бы применить какой-то общий наиболее подходящий метод предварительной обработки. Такой подход поможет, например, с распознаванием растений, в качестве категорий тут можно выделить: цветок, лист, стебель, растение целиком. Важную роль играет просмотр обучающего набора. Если замечено, что объекты того или иного класса стабильно находятся в центре кадра и всегда остаётся пустое пространство до краёв изображения, то имеет смысл обрезать все такие изображения. Если обычно интересующий объект находится на светлом фоне, имеет смысл нормализовать изображение белым цветом (покрасить в белый все пиксели, значения которых меньше порогового значения).

Зашумление изображение обычно возникает из-за: искажений от отражающих свет предметов (стекло, вода, металлы), неравномерной прозрачности воздуха, загрязнений объектива или качества используемой техники. Можно выделить следующие способы фильтрации зашумлённых изображений:

Гауссов шум (белый шум) — спектральные составляющие этого типа шума распределены равномерно по всему спектру частот (типичный пример белого шума — электромагнитные помехи или шум водопада). График плотности распределения показан на рисунке 4. Функция плотности распределения Гауссового шума случайной величины z задаётся формулой:

$$p(z) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(z-\mu)^2/2\sigma^2}, \quad (7)$$

где z — значение яркости, μ - среднее значение случайной величины z, σ – её среднеквадратичное отклонение.

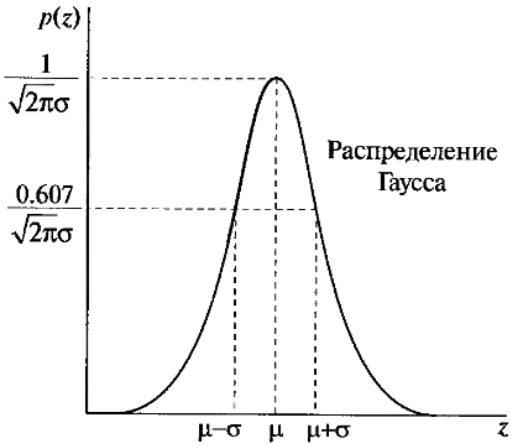

Рисунок 4 — График плотности распределения Гауссового шума

- **Шум Релея** — Впервые был введён при рассмотрении огибающей суммы гармонических колебаний различных частот (например, на фотографиях). График плотности распределения показан на рисунке 5. Функция плотности распределения задаётся выражением:

$$\begin{cases} \frac{2}{b}(z-a)e^{-(z-a)^2/b}, \text{при } z \geq a; \\ 0 \text{ при } z < a. \end{cases} \quad (8)$$

Среднее значение и дисперсия имеют вид:

$$\mu = a + \sqrt{\pi b/4},\ \sigma^2 = \frac{b(4-\pi)}{4} \quad (9)$$

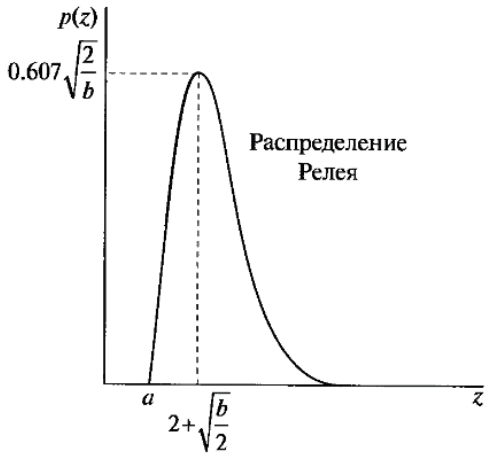

Рисунок 5 — График плотности распределения шума Релея

- **Шум Эрланга** — можно наблюдать на снимках, получаемых с помощью средств лазерной и радиолокационной разведки. График плотности распределения показан на рисунке 6. Функция плотности распределения задаётся формулой:

$$\begin{cases} \frac{a^b z^{b-1}}{(b-1)!} e^{-az} \; при \; z \geq 0; \\ 0 \; при \; z < 0. \end{cases} \qquad (10)$$

Среднее значение и дисперсия имеют вид:

$$\mu = \frac{b}{a}, \sigma^2 = \frac{b}{a^2} \qquad (11)$$

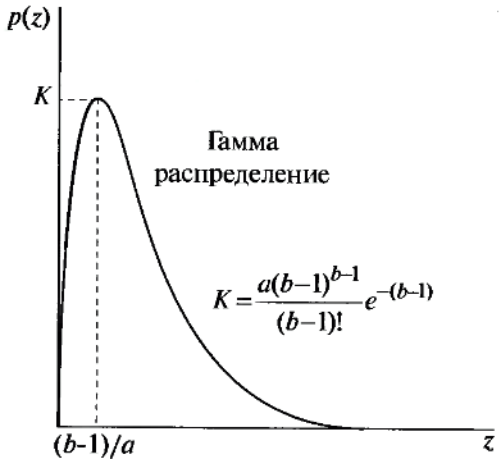

Рисунок 6 — График плотности распределения шум Эрланга

2.2. Реализация методов машинного обучения

При выборе языка программирования для реализации методов машинного обучения были рассмотрены современные тенденции и исследования, сравнивающие преимущества использовании того или иного языка для актуальных задач. В результате выяснилось, что во всех рейтингах языков программирования последних лет одну из лидирующих позиций всегда занимает Python. Например, IEEE Spectrum в своих рейтингах уже который год ставит его на первую строчку [7].

Python — это высокоуровневый, динамически типизированный и платформенно независимый язык программирования с открытым исходным кодом (open-source). Он был создан в 1991 году и широко используется для анализа данных.

Благодаря активному сообществу для Python разрабатывается множество разнообразных библиотек и инструментов, в том числе для машинного обучения. В целях ускорения работы многие из данных библиотек используют средства аппаратного ускорения, в первую очередь ориентирующиеся на графические ускорители GPU (Graphics Processing Unit) и технологию CUDA (Compute Unified Device Architecture — архитектура параллельных вычислений), используемую в GPU-ускорителях компании Nvidia.

Также Python является одним из поддерживаемых и наиболее используемых языков для импорта модели машинного обучения в Xcode в приложение для iOS. Подробнее об этом позже.

Далее рассмотрены наиболее популярные средства машинного обучения, их особенности и функционал, предположительно подходящий мне для дальнейшей работы.

2.2.1. Инструменты машинного обучения Python

Разница между API, фреймворком и библиотекой

API (application programming interface) — это интерфейс взаимодействия с программной системой извне. В API описаны методы взаимодействия с внешними программами. Использование API даёт приложению возможность обратиться к коду вне этого приложения.

Библиотека — это набор готовых функций для решения некоторых типичных задач. При подключении библиотеки она становится частью приложения.

Фреймворк — это каркас для будущего приложения. Это заготовка, включающая в себя: начальный код, структуру, библиотеки, полезные для некоторой задачи. Фреймворки используются для облегчения разработки и объединения разных компонентов большого программного проекта.

Популярные библиотеки

Shogun — открытая (open-source) библиотека машинного обучения с фокусировкой на SVM (Support Vector Machines). Написана на C++. Поддерживаемые языки: Python, Octave, R, Java/Scala, Lua, C#, Ruby, etc.

PyTorch — фреймворк для глубокого машинного обучения с открытым исходным кодом. Написана на Python, C++, CUDA. Используется для решения задач

компьютерного зрения и обработки естественного языка. Оперирует тензорами, под которыми в данном контексте понимаются массивы, которые могут обрабатываться на GPU.

Theano — библиотека с открытым исходным кодом для глубокого обучения и численного вычисления. Написана на Python и CUDA. Основные возможности: интеграция с NumPy, параллельные вычисления, прозрачное использование ресурсов CPU (Central Processing Unit — центральный процессор) и GPU [8].

TensorFlow — библиотека с открытым исходным кодом, предоставляющая возможности по построению и тренировке нейронных сетей, ориентированных на автоматическое нахождение и классификацию образов. Написана на: Python, C++, CUDA. Особенность в возможности использования аппаратного ускорителя собственной разработки — тензорного процессора (TPU) на основе специализированной интегральной схемы, что даёт на порядок лучшие показатели работы. Используется по умолчанию в основе работы рассмотренного далее Keras (заменила Theano).

Keras — это высокоуровневый API глубокого обучения, написанный на Python, работающий поверх фреймворка машинного обучения TensorFlow, Theano или другого.

Caffe (Convolution Architecture For Feature Extraction, Свёрточная архитектура для извлечения признаков) — это фреймворк для глубокого обучения, созданный в первую очередь для решения задач классификации и сегментации изображений. Написан на C++.

LIBSVM (A Library for Support Vector Machines) — библиотека машинного обучения с открытым исходным кодом, реализующая в первую очередь методы опорных векторов. Написана на Java и C++. Есть интерфейсы для Python, R, MATLAB, Perl, Ruby и других языков.

XGBoost (A Scalable Tree Boosting System, Масштабируемая система повышения качества дерева) — это оптимизированная распределенная система повышения градиента, разработанная для обеспечения повышения эффективности, гибкости и портативности. Она реализует алгоритмы машинного обучения в рамках

платформы Gradient Boosting. XGBoost обеспечивает усиление параллельного дерева (GBDT, GBM).

ONNX — это кроссплатформенный ускоритель логического вывода и обучения, совместимый с популярными фреймворками ML/DNN, включая PyTorch, TensorFlow/Keras, scikit-learn и другие. Обучение происходит на Python, но использоваться может в приложениях на C#, C++ и Java.

Scikit-Learn (sklearn) — библиотека с открытым исходным кодом, широко используемая в задачах анализа данных и машинного обучения. Она основана на NumPy и SciPy [9].

Особенности всех упомянутых ранее фреймворков были изучены и собраны в таблицу 1 для обоснования последующего выбора. Было учтено основное предназначение, с которым создавался фреймворк, языки программирования, на которых он написан и какие поддерживаются для использования. Также тут указано, существует ли для фреймворка специальный конвертер в CoreML (об этом будет подробнее в следующем разделе), возможно они работают лучше универсального для Python CoreMLTools. А также скорость работы, которая в принципе является следствием наличия возможности ускорения с помощью графических процессоров.

Изначально в рамках данной работы предполагалось использование библиотеки Scikit-Learn, но в итоге выяснилось, что для работы с изображениями полезнее иметь возможность ускорить процесс обучения с помощью GPU, в то время как Scikit-Learn, ориентированные на решение широкого круга задач, такой поддержки не имеет.

Таким образом, при рассмотрении доступных вариантов фреймворков особое внимание уделялось наличию в них реализованных нейронных сетей. В итоге было принято решение продолжить своё дальнейшее изучение этой темы с Keras и TensorFlow. Также существенную роль при выборе сыграло наличие в TensorFlow большого выбора из готовых датасетов, с которыми сразу удобно экспериментировать и работать.

Таблица 1 — Сравнение фреймворков машинного обучения

Фреймворк	Основное предназначение	Написан на языках	Поддерживаемые языки	Специальный конвертер в CoreML	Скорость работы	Возможность ускорения	Поддержка нейронных сетей
CreateML	упрощает разработку моделей	Swift	Swift	CreateML	++	GPU (BNNS, Metal CNN)	+
Turi Create	упрощает разработку моделей	Python, C++	Python, C++	Turi Create	+	GPU	-
Keras	работа поверх других фреймворков МО	Python, C++	Python, C++	Keras	+	GPU	+
TensorFlow	классификация образов	Python, C++	Python, C++	TF-coreml	+	GPU, TPU	+
ONNX	глубокое обучение и ускорение логического вывода	Python, C++	Python, C#, C++, Java	—	+	GPU	+
Scikit-learn	для взаимодействия с числовыми и научными библиотеками NumPy и SciPy	Python, Cython, C, C++	Python, C++	—	-	—	+
XGBoost	повышение качества дерева	Python, C++	Python, C++	—	+	GPU	-
PyTorch	глубокое обучение	Python, C++	Python, C++	—	++	GPU	+
LibSVM	методы опорных векторов	Java, C++	Python, R, MATLAB, Perl, Ruby и др.	—	-	—	-
Caffe	классификация и сегментация изображений	C++	Python, C++, MATLAB	Caffe	+	GPU	+
Torch	для глубинного обучения и научных расчётов	C, C++, Lua	C++, Lua	Torch2CoreML	+	GPU	+

Основные возможности Scikit-learn

Предварительная обработка данных (`sklearn.preprocessing`). Подготовка данных к алгоритмам машинного обучения: масштабирование данных (`sklearn.preprocessing.MinMaxScaler`), стандартизация (класс `StandardScaler`) или нормализация (`Normalizer`) данных, кодирование категориальных переменных (`OneHotEncoder`).

Уменьшение размерности. Выбор или извлечение наиболее важных признаков многомерного набора данных. В библиотеке реализованы: метод анализа основных компонентов PCA (определяет комбинации признаков, для которых набор данных имеет наибольшую дисперсию, методом машинного обучения без учителя) — `sklearn.decomposition.PCA`; метод линейного дискриминантного анализа (LDA) (определяет комбинации признаков, для которых классы разделяются наилучшим образом, методом машинного обучения с учителем) — `sklearn.discriminant_analysis.LinearDiscriminantAnalysis`.

Выбор модели (`sklearn.model_selection`). Позволяет случайным образом разбивать данные на подмножества для обучения и тестирования моделей — функция `train_test_split`.

Помимо разделения набора реализована задача автоматического подбора наилучших параметров путём обычного перебора (`GridSearchCV`). При этом для каждой возможной комбинации параметров создаётся модель и вычисляются величины, которые показывают производительность модели (например, коэффициент детерминации, среднеквадратичная ошибка, показатель отклонения, матрица ошибок, отчет о классификации, f-показатели и другие). Потому такой подход может быть весьма затратным по ресурсам [10].

Построение конвейера (`sklearn.pipeline`). Конвейеры позволяют объединить линейную последовательность преобразований данных, что завершается процессом моделирования, который можно оценить. Это позволяет создавать свои ансамблевые методы.

Регрессия. В Scikit-learn реализовано множество методов регрессии:

- линейная регрессия (`sklearn.linear_model.LinearRegression`);
- регрессия на базе стохастического градиентного спуска (`sklearn.linear_model.SGDRegressor`);
- полиномиальная регрессия (для нелинейных данных, в этом случае нужно преобразовать начальные данные `sklearn.preprocessing.PolynomialFeatures`);
- регрессии опорных векторов (`sklearn.svm.SVR`);
- регрессия на основе деревьев принятия решений (`sklearn.tree.DecisionTreeRegressor`);
- регрессия LASSO (Least Absolute Shrinkage and Selection Operator, с использованием регуляризации через манхэттенское расстояние — `sklearn.linear_model.Lasso`);
- остальные [11].

Для повышения качества модели можно объединить несколько методов в один ансамбль (`sklearn.ensemble`). По правилу большинства работает `VotingRegressor`, кроме него есть вариант сложнее с использованием `StackingRegressor`.

Также есть реализованные ансамблевые методы (`sklearn.ensemble`): случайный лес (бэггинг деревьев решений — `RandomForestRegressor`), сверхслучайные деревья (больше случайности в вычислении разделения в узлах — `ExtraTreesRegressor`), градиентный бустинг (GBM — `GradientBoostingRegressor`) и другие.

Классификация. В Scikit-learn реализованы следующие методы:

- логистическая регрессия (`sklearn.linear_model.LogisticRegression`);
- наивный байесовский классификатор (`sklearn.naive_bayes.GaussianNB`);
- k-ближайших соседий (`sklearn.neighbors.KNeighborsClassifier`);
- опорных векторов (Support Vector Classification, `sklearn.svm.SVC`);
- дерево принятия решений (`sklearn.tree.DecisionTreeClassifier`) и другие.

Для задачи классификации также есть возможность соединения нескольких классификаторов в один ансамбль (`sklearn.ensemble`). По правилу большинства работает `VotingClassifier`, кроме него есть вариант сложнее с использованием `StackingClassifier` [12].

Также есть реализованные ансамблевые методы (`sklearn.ensemble`):

- случайный лес (бэггинг деревьев решений —`RandomForestClassifier`);
- сверхслучайные деревья (больше случайности в вычислении разделения в узлах — `ExtraTreesClassifier`);
- Классификатор Ada бустинга (`AdaBoostClassifier`);
- градиентный бустинг (GBM — `GradientBoostingClassifier`).

Кластерный анализ (`sklearn.cluster`). Часто кластеризация применяется при анализе данных в качестве первого шага. Такое предварительное разбиение позволяет применить к различным полученным группам свои модели машинного обучения.

В библиотеке реализованы:

- алгоритм k-средних (`KMeans`);
- модифицированная версия алгоритма k-средних (`MiniBatchKMeans`);
- алгоритм сдвига среднего значения (`MeanShift`);
- алгоритм иерархической кластеризации (`AgglomerativeClustering`);
- алгоритм BIRCH (balanced iterative reducing and clustering using hierarchies, есть возможность динамической кластеризации по мере поступления новых многомерных данных — `Birch`);
- алгоритм DBSCAN (Density-based spatial clustering of applications with noise, работа с шумами, алгоритм группирует в один кластер точки в области с высокой плотностью точек — `DBSCAN`);
- алгоритм OPTICS (Ordering Points To Identify the Clustering Structure, изменённый DBSCAN — `OPTICS`);
- алгоритм распространения близости/похожести (попарное распространение сообщений о похожести объектов для выбора типичных представителей каждого кластера — `AffinityPropagation`);
- алгоритм спектральной кластеризации (создан для случая, когда мера центра и разброса кластера не является подходящим описанием всего кластера — `SpectralClustering`);
- алгоритм спектральной бикластеризации (предполагает, что данные имеют структуру шахматной доски — `SpectralBiclustering`);
- алгоритм совместной спектральной кластеризации (`SpectralCoclustering`);
- модель гауссовской смеси (Gaussian Mixture Model, использует допущение, что каждый кластер характеризуется многомерным нормальным распределением с со своим матожиданием и ковариационной матрицей, а плотность распределения набора данных представляет собой взвешенную сумму плотностей распределения для всех клстеров — `GaussianMixture`).

Наборы данных (`sklearn.datasets`). Готовые наборы, подходящие для изучения возможностей библиотеки и тестирования моделей. Включают в себя

известные наборы данных, например, Ирисы Фишера (`load_iris`) для задачи классификации.

Основные возможности Tensorflow и Keras

Keras — предназначенная для построения нейронных сетей библиотека с открытым исходным кодом, реализованная на языке Python, первая версия которой создавалась как надстройка над фреймворками Deeplearning4j, TensorFlow и Theano. Она содержит реализации таких широко применяемых компонентов нейронных сетей, как слои, целевые и передаточные функции, оптимизаторы. По сравнению с используемыми фреймворками, Keras предоставляет высокоуровневый набор абстракций, упрощающий процесс формирования нейронных сетей.

TensorFlow является системой машинного обучения, разработанной Google Brain и открытой для свободного доступа в 2015 году. TensorFlow поддерживает распараллеленную работу как на CPU, так и на GPU, используя в последнем случае архитектуру CUDA [13].

Кроме того, поддерживается специализированный тензорный процессор (TPU) — нейронный процессор со специализированной интегральной схемой, разработанной Google и обеспечивающей по сравнению с графическими процессорами более высокую производительность для больших объёмов вычислений со сниженной точностью.

С выходом TensorFlow 2.0 Google объявила о преобразовании Keras в официальный высокоуровневый API TensorFlow.

Keras предлагает ряд специализированных API [14]:

`tf.keras` предоставляет встроенный в TensorFlow высокоуровневый программный интерфейс для построения и обучения моделей.

Keras позволяет собирать слои для построения моделей. Наиболее распространены модели — стек слоев `tf.keras.Sequential`.

В `tf.keras.layers` собраны различные реализации слоев, использующие типовые аргументы конструктора:

- `activation` —функция активации для слоя.
- `kernel_initializer` и `bias_initializer` — схемы инициализации, создающие веса слоя.

- `kernel_regularizer` и `bias_regularizer` — схемы регуляризации, добавляемые к весам слоёв.

Далее для сконструированной модели методом compile задаются параметры для процесса её обучения:

- optimizer —процедура обучения. Могут быть использованы экземпляры стандартных оптимизаторов, такие как `tf.keras.optimizers.Adam` или `tf.keras.optimizers.SGD`. Также можно указать оптимизаторы ключевыми словами (например, 'adam' или 'sgd') или использовать параметры по умолчанию.
- `loss` — функция для расчёта потерь, минимизируется во время обучения. Наиболее распространено использование: среднеквадратичной ошибки (`mean_squared_error`), категориальной или бинарной кросс-энтропии (`categorical_crossentropy`, `binary_crossentropy`). Функции потерь собраны в модуле `tf.keras.losses`, их можно указывать как просто по имени, так и из модуля.
- `metrics` — метрики обучения, используемые для мониторинга.

Для обучения модели на небольших датасетах тренировочных данных используется метод `fit`, при вызове которого, как правило, задаётся число итераций обучения по всем входным данным (`epochs`) и размер каждого блока данных (`batch_size`). Модель сначала разбивает все данные на блоки, чтобы обучение на каждой итерации происходило по части данных, а не каждый раз на одних и тех же.

Если в процессе обучения и аугментации больших наборов данных весь набор обучающих данных не может уместиться в памяти компьютера, вместо метода `fit` используется `fit_generator`.

Методы для оценивания (`tf.keras.Model.evaluate`) и для предсказания (`tf.keras.Model.predict`) могут использовать данные NumPy и `tf.data.Dataset`.

Также можно расширить поведение модели во время обучения с помощью **функций обратного вызова**. Можно написать и передать модели свою функцию или использовать одну из реализаций в `tf.keras.callbacks`, позволяющих

сохранять контрольные точки модели за регулярные интервалы в отдельных файлах, динамически изменять шаг обучения, останавливать обучение и осуществлять мониторинг поведения модели. Визуализатор от Google, например, позволяет отслеживать потери и точность, визуализировать график модели и другое.

Вся модель может быть записана в файл, при этом сохраняются значения весов, а также конфигурации модели и оптимизатора. Это позволяет получить контрольную точку модели и в будущем продолжить обучение с того же момента (например, попробовать изменить параметры обучения после того, как результат со старыми параметрами перестал улучшаться). Кроме того, можно сохранять одни только значения весов или только конфигурацию модели.

Популярные архитектуры нейронных сетей, используемых при распознавании образов

Предварительно обученная модель — это обученная на некотором наборе данных модель, она уже содержит веса и смещения, которые представляют особенности того набора данных, на котором она была обучена. Использование таких моделей полезно, поскольку экономит время и вычислительные ресурсы. Изученные функции часто можно перенести на другие данные.

На сайте Keras представлена статистика по реализованным в пакете Applications (`tensorflow.keras.applications`) предобученным моделям глубокого обучения, она приведена в таблице 2 [15].

Для дальнейшей работы были выбраны три реализованные и предварительно обученные сети Keras: InceptionV3, ResNet50, InceptionResNetV2. В третьей главе описана программа для анализа эффективности их работы на основе построения моделей машинного обучения на выбранном наборе данных.

Существуют различные архитектуры сетей, ориентированных на разные задачи.

ResNet — Residual Network (остаточная сеть) — произвела революцию глубины нейросетей на соревнованиях в 2015 году. Она состояла из 152 слоёв и снизила процент ошибок до 3,57% в соревновании классификации ImageNet, а также оказалась почти в два раза эффективнее GoogleNet (Inception).

Таблица 2 — Статистика по реализованным в пакете tensorflow.keras.applications моделям глубокого обучения

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

Архитектуры нейронных сетей (InceptionV3, ResNet50, InceptionResNetV2)

Увеличение количества слоёв в первых свёрточных сетях вроде VGG либо AlexNet начиная с какого-то момента приводило к ухудшению результатов. При создании ResNet впервые было предложено пропускать некоторые слои при обучении. При этом остаточные слои больше не содержат признаков и используются для нахождения остаточной функции H(x) = F(x) + x вместо того, чтобы искать H(x) напрямую (схематично это показано на рисунке 7).

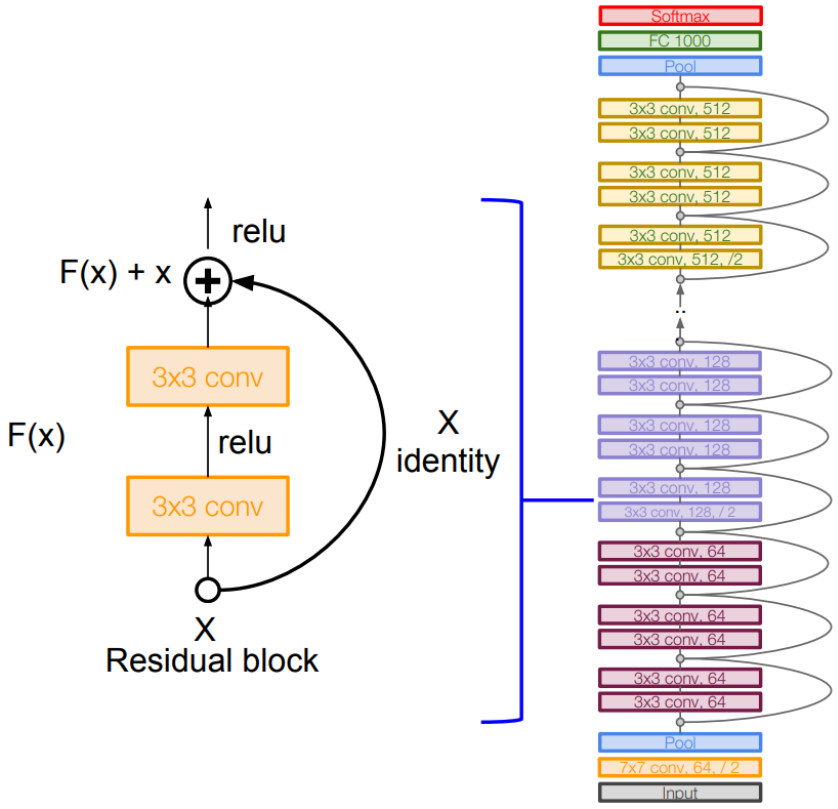

Рисунок 7 — Архитектура ResNet: остаточные блоки

В результате эксперименты с ResNet показали, что глубокие сети можно обучить без ухудшения точности. Потому после начали появляться новые версии этого семейства: сначала Wide ResNet и ResNeXT, потом ResNet50, ResNet101, ResNet152, ResNet50V2, ResNet101V2, ResNet152V2 (Рисунок 8). Разница в количестве слоёв, которое также указывается в названии и в деталях организации блоков.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Рисунок 8 — Структура сетей ResNet разной величины

InceptionV3 (GoogleNet V3) — эта архитектура нейронной сети предложена Google в 2016 году и популярна для работы с изображениями. На рисунке 9 приведена её структура. Как видно, большую часть блоков составляют свёрточные блоки.

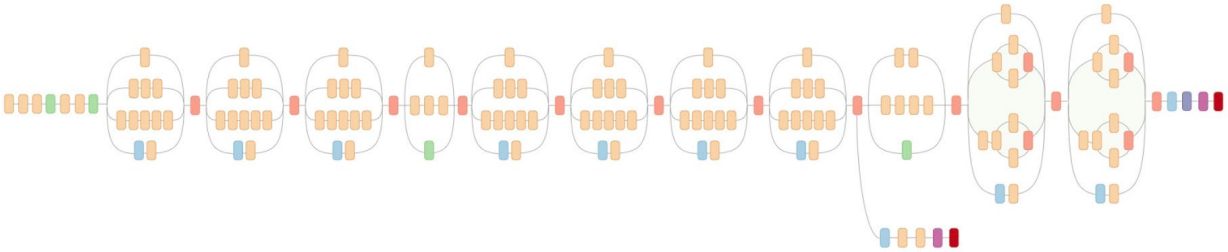

Рисунок 9 — Архитектура сетей InceptionV3

Соответствия блоков типам сетей (устойчивое название на английском языке и перевод пояснение на русском) показаны в таблице 3.

Таблица 3 — Типы блоков на схеме архитектуры сети InceptionV3

	Convolution	Свёрточные
	Average Pooling	Средний пулинг
	Max Pooling	Максимальный пулинг
	Concat	Соединение
	Dropout	Выпадение
	Fully connected	Полностью подключен
	Softmax	техника Softmax

InceptionResNetV2 — в сравнении с InceptionV3 тут добавляются остаточные (residual) блоки. Таким образом, она объединила в себе некоторые особенности семейств архитектур Inception и ResNet

На рисунке 10 сверху показана схема архитектуры сети, а снизу её сжатый упрощённый вариант.

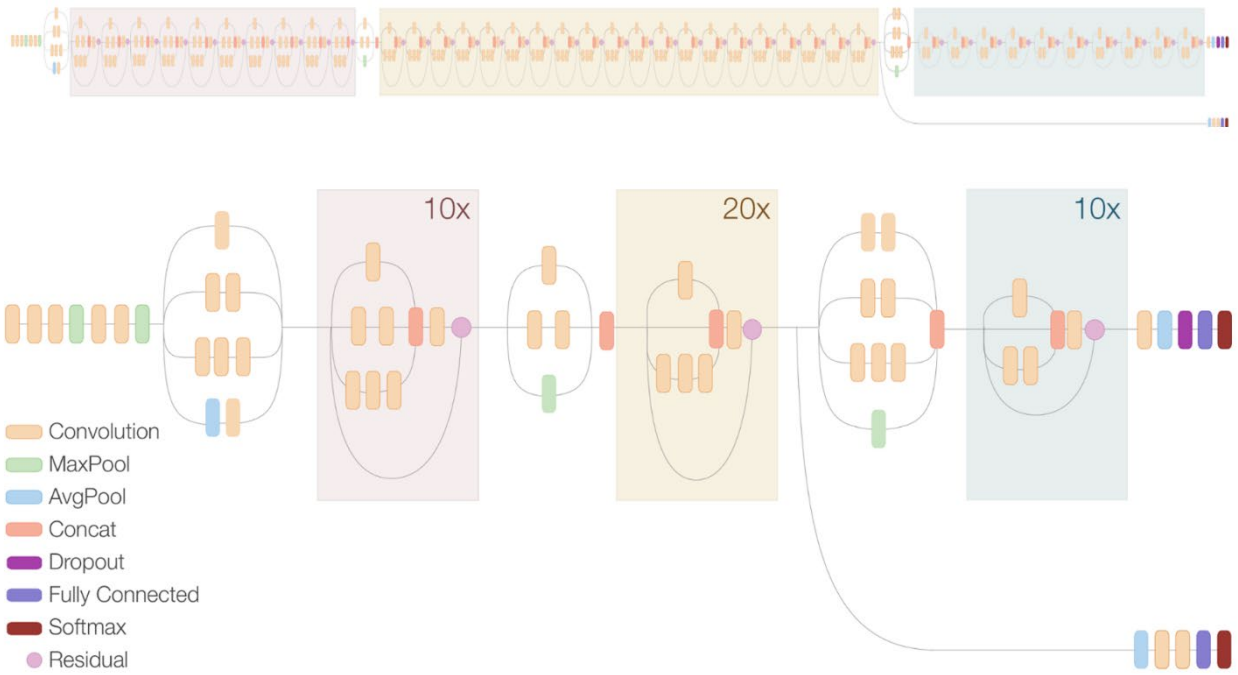

Рисунок 10 — Архитектура сетей InceptionResNetV2

2.2.2. Реализации технологии машинного обучения в современных мобильных устройствах

Компания Apple выпустила **Core ML** в версии iOS 11 в 2017 году. Core ML позволяет разработчикам интегрировать модели машинного обучения в приложения для iOS и MacOS. Это стало первой большой попыткой реализации задачи выполнения машинного обучения на мобильных устройствах. Оптимизация Core ML позволяет к минимуму объем памяти и энергопотребление модели. Работа непосредственно на устройстве также обеспечивает безопасность пользовательских данных, а приложение работает даже без сетевого подключения. Кроме того, он прост в использовании, ведь для интеграции готовой модели машинного обучения достаточно нескольких строк кода. Единственный недостаток в том, что это позволяет устройствам лишь делать прогнозы импортированными предварительно обученными моделями, а не обучать свои новые модели.

В 2018 компания Apple анонсировала **Create ML**, расширившую возможности Core ML. Это дало возможность разработчикам создавать и обучать модели непосредственно в Xcode (основной среде разработки для Apple устройств), а не

зависеть от внешних инструментов. Apple продолжает разрабатывать новые API, эффективно использующие новые процессоры компании.

С некоторым опозданием (в 2018 году) Google анонсировала библиотеку **ML Kit for Firebase**, она позволяет использовать машинное обучение в мобильных приложениях двумя способами: вывод модели в облаке через API или работать только на устройстве, как и в Core ML. Также ML Kit поддерживает загрузку модели TensorFlow Lite. Версия ML Kit для устройств предлагала низкую точность по сравнению с облачной версией, преимущество которой также состояло в поддержке как iOS, так и Android, что позволяло использовать одни и те же API на обеих платформах.

В 2020 технология ML Kit для устройств и её кроссплатформенная поддержка были значительно улучшены, но по состоянию на начало 2021 года большая часть её API находится в состоянии бета-версии.

Помимо Core ML и ML Kit существует ряд библиотек машинного обучения с открытым исходным кодом для мобильных платформ, таких как Shark, AIToolbox, BirdBrain и других. Однако эти библиотеки не получили широкого распространения, поскольку не поддерживаются какой-либо крупной организацией [16].

2.2.3. Машинное обучение для Apple-устройств

Поскольку, как было показано в предыдущем разделе, Apple предоставляет в целом более зрелые решения по применению технологий машинного обучения, было принято решение сконцентрироваться в данной работе на разработке мобильного приложения для iOS, демонстрирующего работу модели машинного обучения. Для этого требуется изучить существующие возможности, которые предоставляет компания для импортирования и создания моделей машинного обучения.

В настоящее время Apple предлагает два модуля для реализации машинного обучения на своих устройствах: Core ML и Create ML [17]. Эти модули работают с предметно-ориентированными фреймворками (для анализа изображений, обработки текста, идентификации звуков в аудио). Для оптимизирования вычисления в них используются библиотеки BNNS и Metal CNN.

Общий подход

Для реализации машинного обучения в рамках подхода Apple необходимо:

- создать модель (файл с разрешением `mlmodel`);
- добавить модель в Compile Sources, чтобы на этапе сборки приложения она каждый раз компилировалась (`mlmodelc`) и в результате давала предсказания;
- прописать использование модели в приложении.

Последние два пункта делаются непосредственно в среде Xcode, а создавать модель можно не только там, но и внешними средствами, совместимыми для импортирования.

Далее подробнее про средства машинного обучения Apple, их особенности и функционал, требующийся для дальнейшей работы.

Инструменты машинного обучения для iOS

Компания Apple предлагает два модуля для реализации машинного обучения на своих устройствах: Core ML и Create ML[17]. Эти модули работают с предметно-ориентированными фреймворками (для анализа изображений, обработки текста, идентификации звуков в аудио). Для оптимизации вычислений в них используются библиотеки BNNS и Metal CNN[18].

Существует два основных подхода к созданию моделей:

1. Импортировать предварительно созданную и обученную модель данных, используя один из совместимых фреймворков [19].

2. Создать модель и выбрать один из существующих шаблонов (реализованных предметно-ориентированных фреймворков).

Далее подробнее рассмотрим возможные способы реализации машинного обучения внутри приложения iOS, они схематично показаны ниже (Рисунок 11).

Core ML — это фреймворк, обеспечивающий использование моделей, полученных из других популярных фреймворков: TensorFlow, Keras, PyTorch, scikit-learn, Caffe, и используемый в таких собственных продуктах Apple как Siri, Camera и QuickType [20].

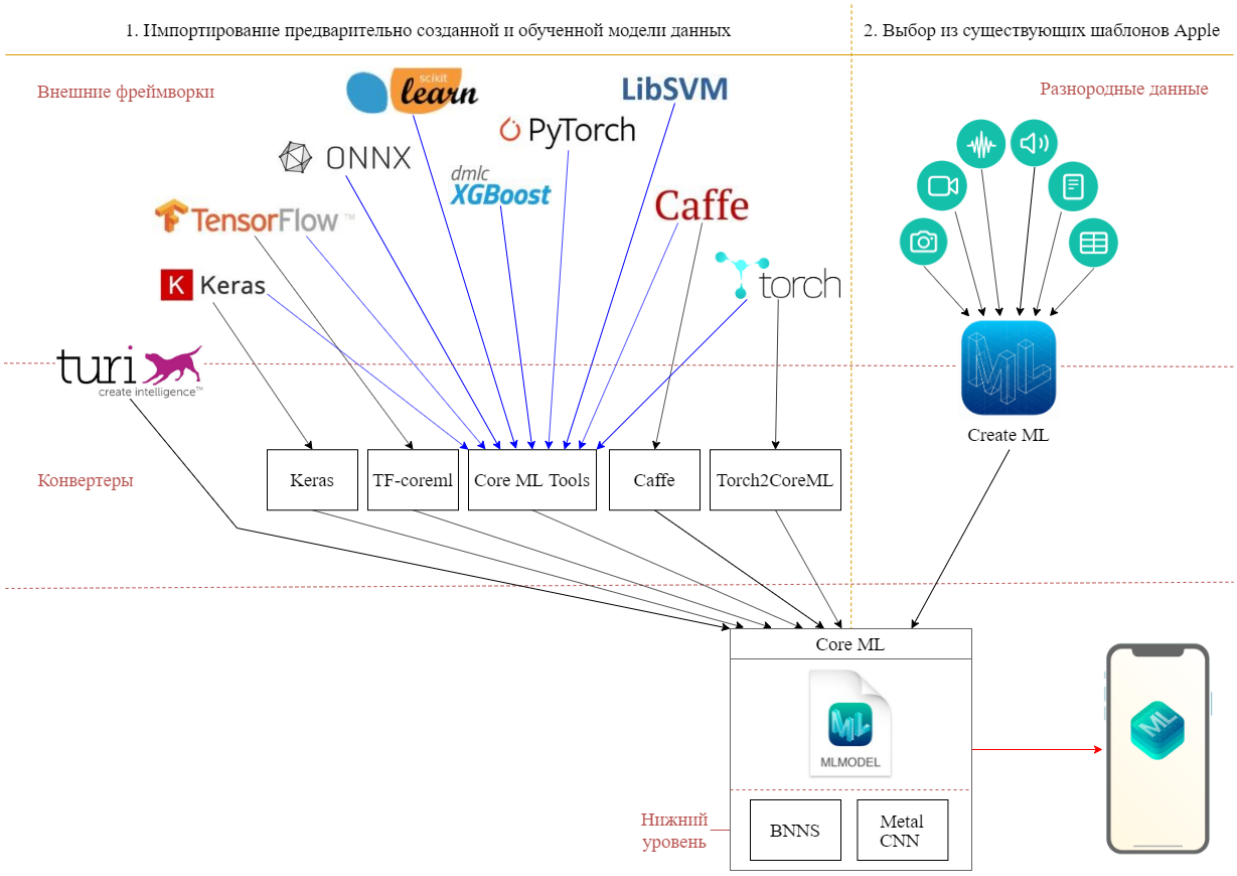

Рисунок 11 — Способы реализации машинного обучения для iOS-приложений

Core ML не может самостоятельно принимать данные и обучать модели. Он только использует некоторые типы обученных моделей, в том числе сформированных на языке Python, преобразовывать их в собственный формат и делать прогнозы.

Core ML является основой для следующих предметно-ориентированных фреймворков:

- Vision — для анализа изображений (классы, начинающиеся буквами `VN`, например, класс `VNGenerateObjectnessBasedSaliencyImageRequest` для создания тепловой карты, которая определяет части изображения, которые с наибольшей вероятностью представляют объекты);
- Natural Language — для обработки текста (`NL`, например, класс `NLTokenizer` для разделения текста на естественном языке на семантические единицы);
- Speech — для преобразования звука в текст (`SF`, например, класс `SFSpeechURLRecognitionRequest` для запросов на распознание речь в записанном аудиофайле);

- Sound Analysis — для определения звуков в аудио (`SN`, например, класс `SNClassification` для классифицирования анализируемого).

Для популярных фреймворков существуют конвертеры их моделей в Core ML:

— **Coremltools** — универсальный конвертер модели на языке Python в формат Core ML.

— **Keras** — конвертер модели из Keras в Core ML. Keras — это высокоуровневый API глубокого обучения, написанный на Python, работающий поверх фреймворка машинного обучения TensorFlow, Theano или др.

— **TF-coreml** — конвертер модели из TensorFlow в Core ML.

— **Torch2CoreML** — конвертер модели из Torch в Core ML.

— **Caffe** — конвертер модели из Caffe в Core ML. Caffe (Convolution Architecture For Feature Extraction, Свёрточная архитектура для извлечения признаков) — это фреймворк для глубокого обучения, созданный в первую очередь для решения задач классификации и сегментации изображений. Написан на C++.

Turi Create — простой фреймворк для обучения моделей с поддержкой большого числе сценариев, также обеспечивающий их конвертацию в Core ML [21]. Как и многие другие фреймворки, Turi Create ориентирован на язык Python [19], ряд моделей может использовать GPU.

Помимо использования моделей, конвертированных из популярных кроссплатформенных фреймворков, доступен ряд средств, ориентированных непосредственно на платформу Apple [22].

Create ML — собственный инструментарий Apple, позволяющий обучать модели Core ML без написания программного кода, поскольку начиная с iOS версии 12 используются модели, встроенные в систему. Для этого в Xcode есть интерфейс для выбора одного из 6 типов данных, с которым будет работать модель: изображения, видео, активность (данные движения от акселерометра и гироскопа), звук, текст и таблицы. Для каждого типа данных предусмотрены свои шаблоны, всего их сейчас 11 (Рисунок 12 и Рисунок 13).

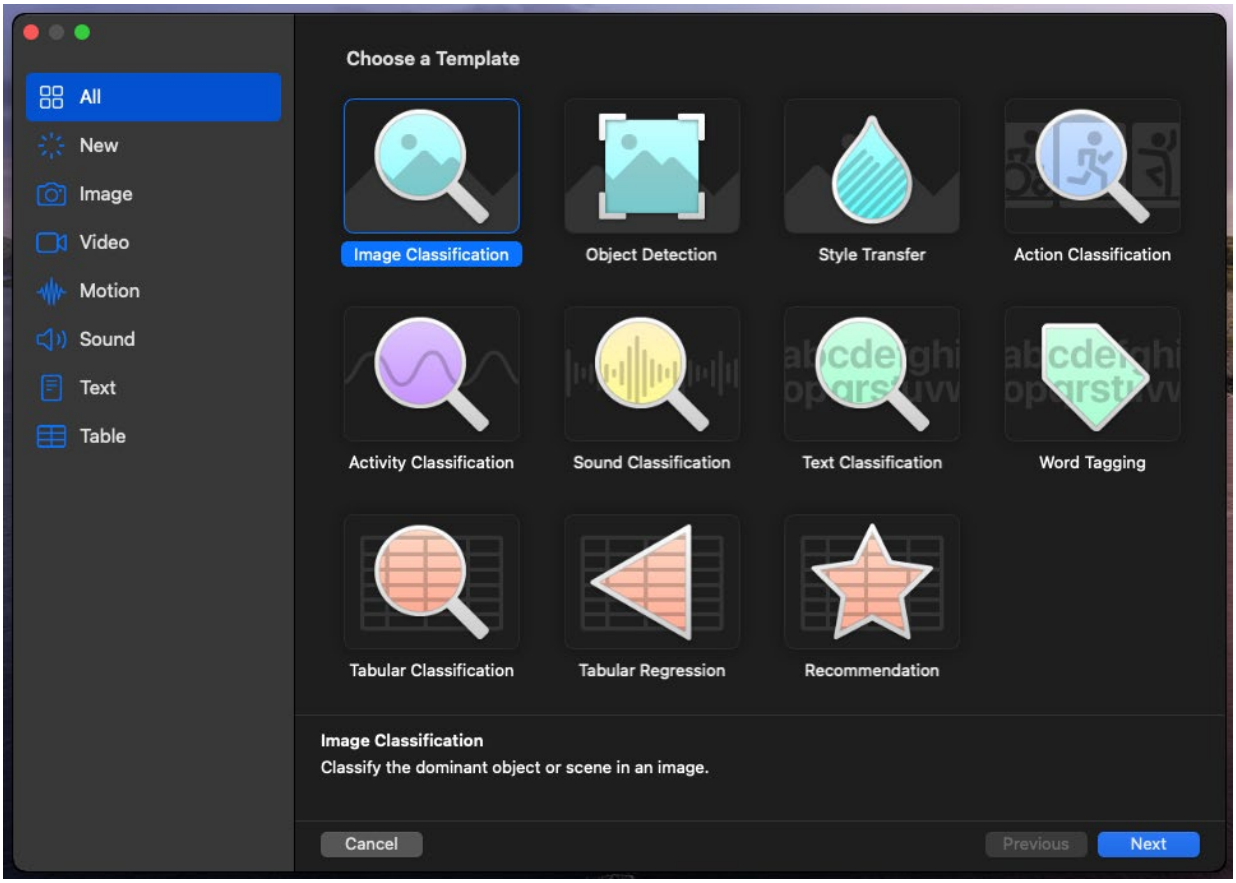

Рисунок 12 — Выбор шаблона Create ML

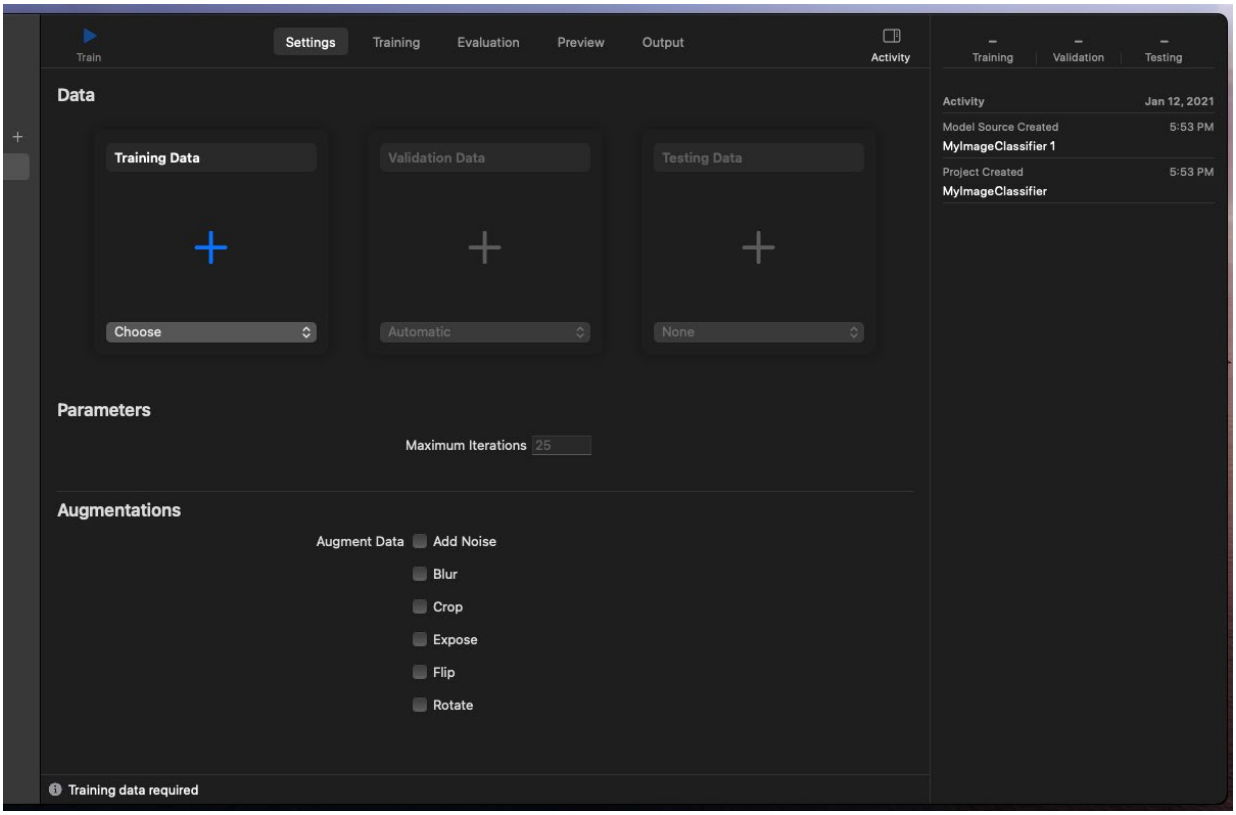

Рисунок 13 — Среда для работы с моделью Create ML

Также Create ML поддерживает мультимодельное обучение (обучение нескольких моделей на разных наборах данных в одном проекте).

Помимо этого, можно контролировать процесс обучения: приостановить, сохранить, возобновить и продлить обучение. Также можно ускорить процесс обучения модели, используя GPU [23].

Подробнее про поддерживаемые сейчас типы и шаблоны модели:

- Тип Image (модели, работающие с изображениями), шаблоны:
  - Image Classifier (классификатор изображений)
  - Object Detector (детектор объектов)
  - Style transfer (перенос стилей)
- Video (видео):
  - Action classification (классификатор действий)
  - Style transfer (перенос стилей)
- Motion (активность, данные движения от акселерометра и гироскопа):
  - Activity classification (классификатор активности)
- Sound (звуки):
  - Sound classification (классификатор звука)
- Text (текст):
  - Text classification (классификатор текста)
  - Word tagging (маркировщик слов)
- Tabular (Таблицы):
  - Tabular classification (табличный регрессор)
  - Tabular regression (табличный классификатор)
  - Recommendation (рекомендатор)

Кроме того, компания Apple предлагает несколько уровней поддержки машинного обучения, помимо использования моделей существуют низкоуровневые API, работающие напрямую с CPU и GPU:

BNNS (Basic Neural Network Subroutines) — часть системы Accelerate, которая является основой для выполнения оптимизированных вычислений на CPU [24].

Metal CNN (Convolutional Neural Networks) — часть библиотеки Metal Performance Shaders, оптимизирующей работу ядер и использующей GPU вместо CPU [25].

2.3. Средства разработки мобильных приложений

Первоначально при выборе средств разработки предполагалось использовать кроссплатформенные средства, ориентированные на универсальный подход, что позволило бы использовать один код на устройствах с разными операционными системами и аппаратными платформами. Однако рассмотрение существующих кроссплатформенных средств разработки (раздел 1.3) показало, что все они ориентированы в лучшем случае на разработку пользовательского интерфейса. При этом использование специфических возможностей конкретной платформы, таких как реализация алгоритмов машинного обучения, в любом случае требует работы с нативным кодом. Поскольку приложение, иллюстрирующее возможности использования технологий машинного обучения в мобильных приложениях, не предполагает построение сложного графического интерфейса, было решено остановиться на работе с нативным кодом. И так как в разделе 2.1.1 было выбрано решение ориентироваться на разработку мобильного приложения для iOS, далее необходимо рассмотреть средства разработки для этой платформы.

2.3.1. Среды разработки

Самыми популярными средами разработки приложений на iOS являются Xcode от Apple и AppCode от JetBrains.

Xcode — Быстрый редактор, укомплектованный полным набором инструментов для разработки под iOS, macOS и др. Можно быть получен бесплатно из App Store.

AppCode — Как и Xcode, содержит все необходимые средства для разработки приложений на языках Objective-C, Swift, C++ и на 100% совместима с Xcode. Стабильнее и быстрее Xcode, содержит более подробное описание ошибок и предупреждений, но в нём нет визуального отладчика и для сборки и запуска приложения всё равно нужен Xcode.

Таким образом, в рамках практической работы для создания небольшого приложения на Apple-устройстве для демонстрации возможностей машинного обучения целесообразнее использовать Xcode, ведь использование AppCod в данной ситуации излишне.

2.3.2. Языки программирования

Исторически первым языком, используемым при разработке приложений для платформ iOS и macOS, был **Objective-C** — компилируемый объектно-ориентированный язык программирования, являющийся надмножеством языка C. Язык создавался с целью решить проблему повторного использования кода.

Для снижения планки вхождения в процесс разработки приложений для iOS и macOS в 2014 году компания Apple разработала компилируемый язык общего назначения **Swift**, который проектировался как более лёгкий для чтения и устойчивый к ошибкам программиста язык, нежели предшествовавший ему Objective-C [26]. Swift может использовать стандартную библиотеку Objective-C, что делает возможным использование обоих языков в рамках одной программы.

Objective-C всё ещё сохраняет актуальностью, многие проекты по-прежнему используют на Objective-C. И некоторые классы Swift содержат в себе код на Objective-C. Но всё больше разработчиков предпочитают Swift, и как следствие он быстрее развивается и более востребован.

В рамках практической работы для создания небольшого приложения на Apple-устройстве актуальнее использовать более современный Swift.

Средства разработки пользовательских интерфейсов

В процессе развития компании улучшались и средства создания пользовательских интерфейсов. Классический способ создания пользовательских интерфейсов для платформы iOS — использовать **UIKit**. Этот императивный управляемый событиями фреймворк появился в 2012 году. В 2019 году Apple представила декларативный фреймворк **SwiftUI**, облегчающий создание интерфейсов приложений для платформ компании. У него есть свой интерактивный редактор интерфейса — Canvas. При написании кода, его визуальная часть автоматически генерируется в редакторе и наоборот. Чёткая структурированность сокращает количество кода и облегчает понимание кода и поиск ошибок.

3. Разработка мобильного приложения для классификации изображений

Этапы разработки приложения представлены на рисунке 14.

Первая часть работы состоит в подготовке данных, создании и обучении модели машинного обучения с использованием tensorflow.keras на языке Python. Также процесс обучения был ускорен за счёт работы GPU.

Вторая часть работы заключается в импортировании и использовании обученной ранее модели и в создании мобильного iOS приложения на языке Swift в среде Xcode в операционной системе macOS с помощью виртуальной машины VMware.

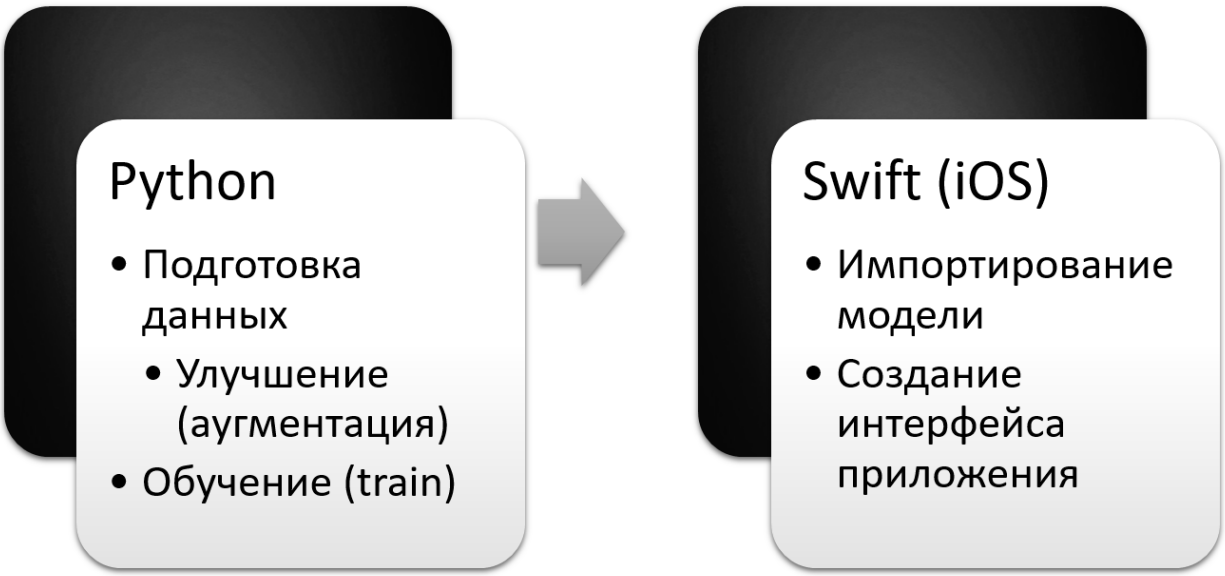

Рисунок 14 — Этапы разработки приложения

3.1. Подготовка и конвертация модели

Для реализации модели было проведено сравнение работы нейронных сетей ResNet50, InceptionV3, InceptionResNetV2, реализованных в пакете TensorFlow Keras Applications.

3.1.1. Ускорение вычислений графическим процессором

Для ускорения обучения использовался графический процессор NVIDIA, что потребовало установки драйвера CUDA. На рисунке 15 приведено окно выбора подключения компонент драйвера.

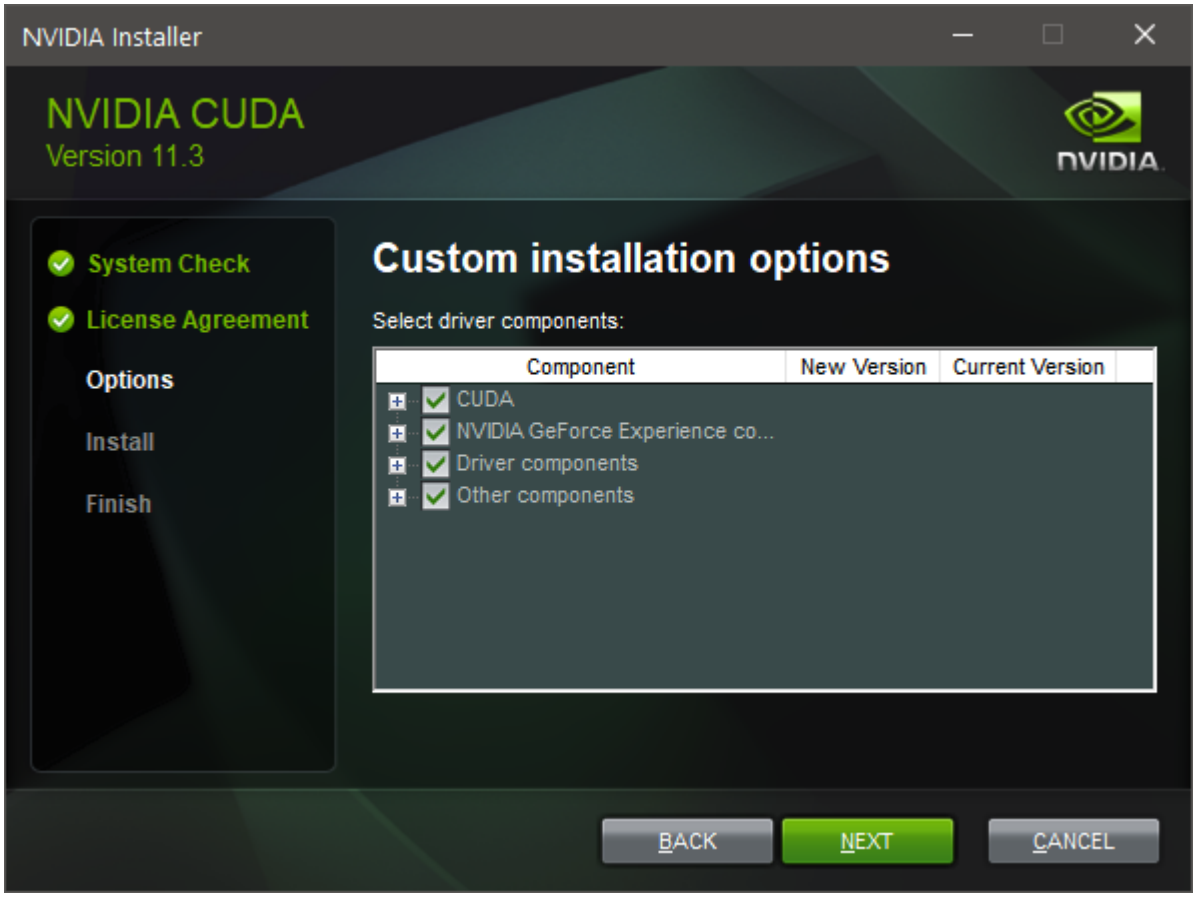

Рисунок 15 — Окно выбора подключения компонент драйвера

3.1.2. Анализ эффективности архитектур нейронных сетей

Для создания модели, которая после была импортирована в мобильное приложение, первым делом требовалось выбрать какую предобученную модель использовать. В связи с этим был проведён анализ эффективности архитектур нейронных сетей, реализована программа NeuroTestAL (полный листинг в приложении А). Анализ проводится на основе построенных моделей машинного обучения на заданном наборе изображений, в качестве базовой структуры использовался набор данных `tfds.image_classification.Food101`.

В программе рассматриваются три архитектуры нейронных сетей (ResNet50, InceptionV3, InceptionResNetV2), добавление прочих сетей в программу для их анализа можно легко осуществить, если импортировать нужный модуль и вызвать программу с соответствующим параметром.

Программа содержит несколько режимов работы, регулируемых параметрами запуска:

- подготовки данных (`prepare`);
- обучения модели с выбранной архитектурой нейронной сети (`train`);
- тестирования работы модели на одном конкретном изображении (`testimage`);
- тестирования работы модели на наборе изображений (`fulltest`);
- построения графиков зависимостей точности (accuracy) и потерь (loss) от эпохи (`plot` для конкретной архитектуры и `plots` для их сравнения);
- вызова инструкции с описанием всех предусмотренных параметров.

Инструкция вызывается параметром `-help` или `-h`, результат её вызова представлен на рисунке 16.

```
Neural Network Analyser (NeuroTestAL)

optional arguments:
  -h, --help     show this help message and exit
  -m MODE        режим работы: prepare/train/testimage/fulltest/plot/plots
  -b BATCH_SIZE  размер пакета
  -p MODEL_PATH  имя сохраненной модели
  -i IMAGE_PATH  путь до тестового изображения
  -s DATASET     каталог набора данных
  -e EPOCHS      количество эпох
  -d DROPOUT     величина dropout
  -n NETWORK     выбор сети (ResNet50, InceptionV3, InceptionResNetV2)
  -x X           ширина целевого изображения
  -y Y           высота целевого изображения
```

Рисунок 16 — Инструкция с описанием всех предусмотренных параметров

Далее показано, какие дополнительные параметры могут использоваться различными режимами (Рисунок 17). Все режимы и параметры запуска обрабатываются в функции `main` (листинг всей программы приведён в приложении А).

Задание режима работы программы необходимо, каждый режим вызывает выполнение соответствующей ему функции. Таким образом, весь код разделён на логические куски, которые удобно как использовать, так и воспринимать по отдельности. Теперь подробнее про ключевые режимы.

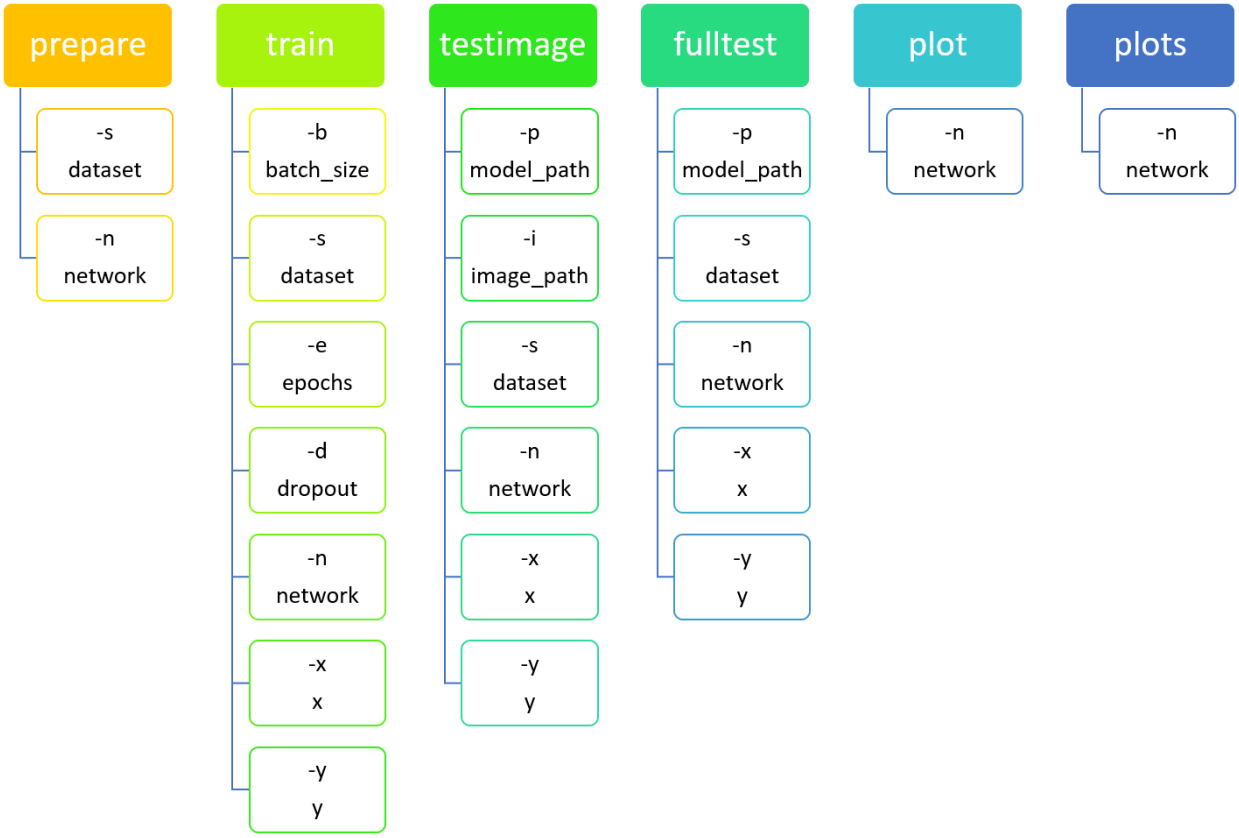

Рисунок 17 — Диаграмма, показывающая параметры, используемые в режимах

Поскольку удобнее использовать готовую функцию `train_datagen.flow_from_directory` для загрузки файлов из заданной директории с автоматической разбивкой по классам, была проведена предварительная подготовка, заключающаяся в создании отдельных каталогов для обучения (train) и тестирования (test) и копировании в них данных. Описанное ранее происходит в функции `prepare`.

Режим обучения модели задаётся в функции `train` (Листинг 1). Первым делом проводится аугментация обучающих данных, функция `setup_generator`. После задаётся логика создания названия модели, а также указывается csv файл (history_log.csv) для сохранения истории изменения ключевых характеристик (accuracy, val_accuracy, loss, val_loss). В дальнейшем это даст возможность изучить полную картину развития модели на каждой эпохе и выбрать наилучшую модель из полученных. Это особенно полезно в случае переобучения, когда показатели модели начинают ухудшаться. И наконец вызываются функции для создания

(create_model) и обучения (train_model) модели, принимающие в качестве параметров результаты ранее оговоренного.

```
def train(args):
    X_train, X_test = setup_generator(make_dataset_path(args,
                                                        'train'),
                                      make_dataset_path(args, 'test'),
                                            args.batch_size, shape[:2])
    print(X_train)
    callbacks = []
    callbacks.append(ModelCheckpoint(filepath=make_model_path(args,
                                args.dataset + '-weights.epoch-{epoch:02d}
                                                -val_loss-{val_loss:.4f}
                                 -val_accuracy-{val_accuracy:.4f}.hdf5'),
                                           verbose=1, save_best_only=True))
    callbacks.append(CSVLogger(make_model_path(args,
                                                        'history_log.csv')))

    model_final = create_model(args.network, X_train.num_classes,
                                                       args.dropout, shape)
    train_model(model_final, X_train, X_test, callbacks, args)
```

Листинг 1 — Функция train

Теперь подробнее про аугментацию (Листинг 2). Класс ImageDataGenerator создан для генерации пакетов данных тензорного изображения [27]. В нём предусмотрено множество аргументов для регулирования изменения изображений и тем самым, создания дополнительных обучающих данных. В программе задавались следующие основные аргументы:

- rotation_range: Int. Диапазон градусов для случайных поворотов.
- width_shift_range: Float. Диапазон сдвига по ширине.
- height_shift_range: Float. Диапазон сдвига по высоте.
- brightness_range: Tuple или list of two floats. Диапазон для выбора значения сдвига яркости.
- shear_range: Float. Интенсивность сдвига (угол поворота против часовой стрелки в градусах).
- zoom_range: Float или [lower, upper]. Диапазон для случайного увеличения. Если Float, [lower, upper] = [1-zoom_range, 1+zoom_range]
- channel_shift_range: Float. Диапазон для случайных смещения каналов.

- `fill_mode`: {«constant», «nearest», «reflect», «wrap»}. По умолчанию ‘nearest’.
- `horizontal_flip`: Boolean. Случайно поворачивает входные данные по горизонтали.
- `vertical_flip`: Boolean. Случайно поворачивает входные данные по вертикали.
- `rescale`: Float. Коэффициент масштабирования. По умолчанию None, и в этом случае масштабирование не применяется, иначе данные умножаются на заданное значение (после применения всех остальных преобразований).

```
def setup_generator(train_path, test_path, batch_size, dims):
    train_datagen = ImageDataGenerator(
        rotation_range=40,       width_shift_range=0.2,
        height_shift_range=0.2,  rescale=1. / 255,
        shear_range=0.2,         zoom_range=0.2,
        horizontal_flip=True,    fill_mode='nearest')

    test_datagen = ImageDataGenerator(rescale=1. / 255)

    train_generator = train_datagen.flow_from_directory(
        train_path, target_size=dims, batch_size=batch_size)

    validation_generator = test_datagen.flow_from_directory(
        test_path, target_size=dims, batch_size=batch_size)

    return train_generator, validation_generator
```

Листинг 2 — Функция setup_generator

Функция для создания модели использует переданное при запуске имя нейронной сети. Возвращает созданную модель (Листинг 3).

Выпадение (dropout) — это метод регуляризации отсева, при котором случайно выбранные нейроны сети игнорируются во время обучения сети. Параметр dropout задаёт вероятность, с которой нейроны будут исключаться в реализованной в Keras функции Dropout. Когда одни нейроны случайно выпадают из сети во время обучения, другие нейроны должны вмешаться и обработать представление, необходимое для прогнозирования отсутствующих нейронов. Считается, что это приводит к тому, что сеть изучает несколько независимых внутренних представлений. В результате сеть становится менее чувствительной к удельному

весу нейронов. Это, в свою очередь, приводит к созданию сети, способной к лучшему обобщению и с меньшей вероятностью превзойти обучающие данные.

```
def create_model(m_type, num_classes, dropout, shape):
    modelclass = globals()[m_type]
    base_model = modelclass(weights='imagenet',  include_top=False,
            input_tensor=Input(shape=shape))

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(dropout)(x)
    predictions = Dense(num_classes, activation='softmax')(x)

    model_final = Model(inputs=base_model.input, outputs=predictions)

    return model_final
```

Листинг 3 — Функция create_model

И наконец само обучение происходит функцией `train_model` (Листинг 4). Сначала данные оптимизируются и настраиваются методом `compile`, в нём задаётся функция, вычисляющая потери (loss) — категориальная кроссэнтропия — и метрика для оценки модели — accuracy. Далее метод `fit_generator` обучает модель заданное количество раз (эпох).

```
def train_model(model_final, train_generator, validation_generator,
                                                    callbacks, args):
    model_final.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

    model_final.fit_generator(train_generator,
        validation_data=validation_generator,
        epochs=args.epochs, callbacks=callbacks,
        steps_per_epoch=train_generator.samples//args.batch_size,
        validation_steps=validation_generator.samples//args.batch_size)
```

Листинг 4 — Функция train_model

При запуске этого режима начинается обучение модели. Каждая эпоха занимает 20-25 минут и ещё минут 5 требуется для сохранения. Также было замечено, что в районе 20-ых эпох показания моделей улучшались всё реже, так что чисто эпох по умолчанию было уменьшено до 25.

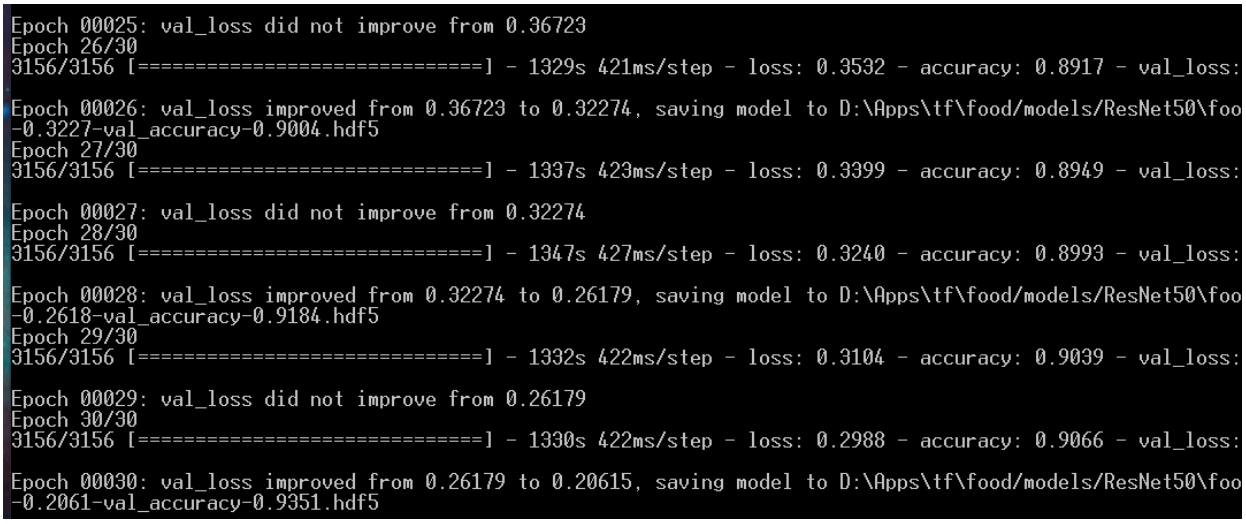

Рисунок 18 — Вывод консоли в процессе обучения (конец)

В процессе обучения используемые данные разбиваются на два внутренних подмножества: данные одного из них используются для фактического обучения (train), а данные другого для проверки состояния обучения (validation — val, val_accuracy, val_loss) на каждой эпохе.

Режимы `plot` и `plots` открывают файлы `histopy_log.csv` и рисуют графики зависимости точности (accuracy) и потерь (loss) от эпохи. Режим `plot` делает это для конкретной выбранной модели, выводит для сравнения результаты на обучающем наборе (train) и на тестовом (val) (Рисунок 19), а режим `plots` строит графики для всех рассматриваемых в приложении моделей, выводя результаты всех моделей для каждой из характеристик, высчитанных при обучении (Рисунок 20).

В результате InceptionResNetV2 дала наиболее хорошие результаты (наилучшую точность и минимальные потери на протяжении всего обучения) и была выбрана для импортирования в приложение.

Для тестирования было реализовано два режима. Тест на одном изображении производится функцией `testimage` (Листинг 5), нужно передать путь к проверяемому изображению и указать модель.

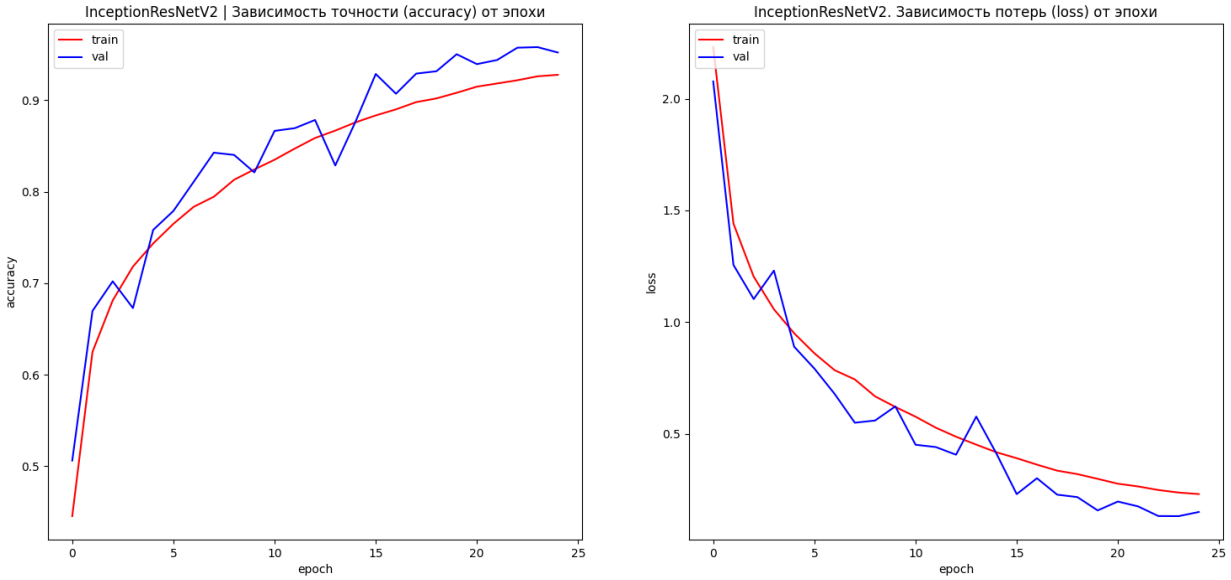

Рисунок 19 — Графики зависимости точности (accuracy) и потерь (loss) от эпохи для сети InceptionResNetV2

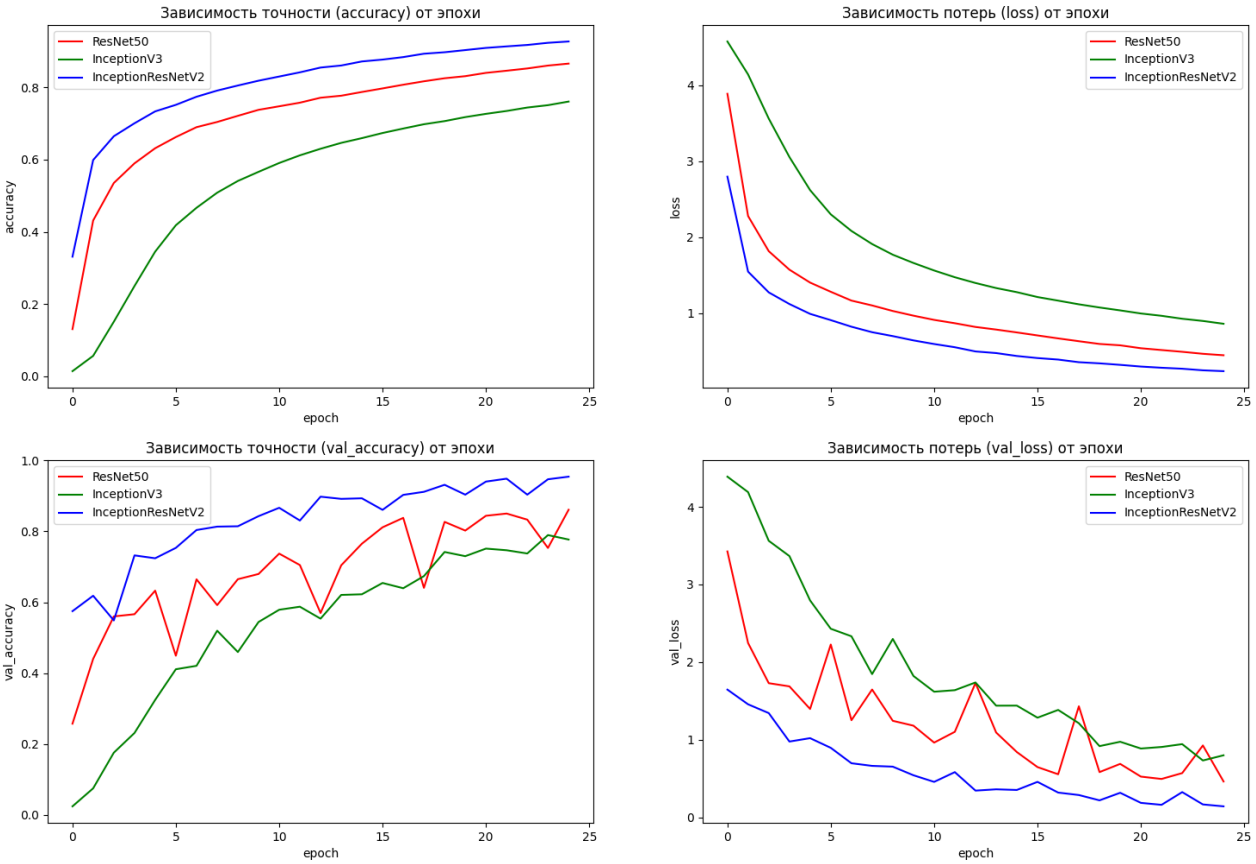

Рисунок 20 — Графики зависимости точности (accuracy) и потерь (loss) от эпохи для рассматриваемых сетей (InceptionV3, ResNet50, InceptionResNetV2)

```
def testimage(args):
    classes = load_strings(make_dataset_path(args, 'meta/classes.txt'))
    trained_model = load_model(args, len(classes), shape)
    image = load_image(args.image_path, shape[:2])
    preds = trained_model.predict(image)
    print("класс изображения: ", classes[np.argmax(preds)])
    print("с вероятностью: ", np.max(preds)*100, "%")
```

Листинг 5 — Функция testimage

Функция `fulltest` (Листинг 6) проходит по всему файлу `test.txt`, в котором собраны пути (имя_класса/имя_изображения), и для каждого из них проводится предсказание выбранной моделью. Рассчитывает число и процент правильно и ошибочно распознанных изображений и рисует гистограмму точности распознавания классов.

```
def fulltest(args):
    classes = load_strings(make_dataset_path(args, 'meta/classes.txt'))
    trained_model = load_model(args, len(classes), shape)
    test_images = load_strings(make_dataset_path(args, 'meta/test.txt'))
    n_true, n_false = 0, 0
    corrects = collections.defaultdict(int)
    incorrects = collections.defaultdict(int)
    for i in test_images:
        (iclass, iname) = i.split('/')
        image_path = make_dataset_path(args, 'test/'+i+'.jpg')
        image = load_image(image_path, shape[:2])
        preds = trained_model.predict(image)
        if classes[np.argmax(preds)] == iclass:
            n_true += 1
            corrects[iclass] += 1
        else:
            n_false += 1
            incorrects[iclass] += 1
    print('\nПравильно распознаны: ', n_true)
    print(n_true/(n_true+n_false)*100,' %')
    print('\nОшибочно распознаны: ', n_false)
    print(n_false/(n_true+n_false)*100,' %')
    class_accuracies = {}
    for ix in range(101):
        class_accuracies[ix] = corrects[classes[ix]]/250
        print(classes[ix], class_accuracies[ix])
    fig = plt.hist(list(class_accuracies.values()), bins=20, color='r')
    plt.title('Гистограмма точности распознавания классов')
    plt.savefig('histogram.png')
    plt.show()
```

Листинг 6 — Функция fulltest

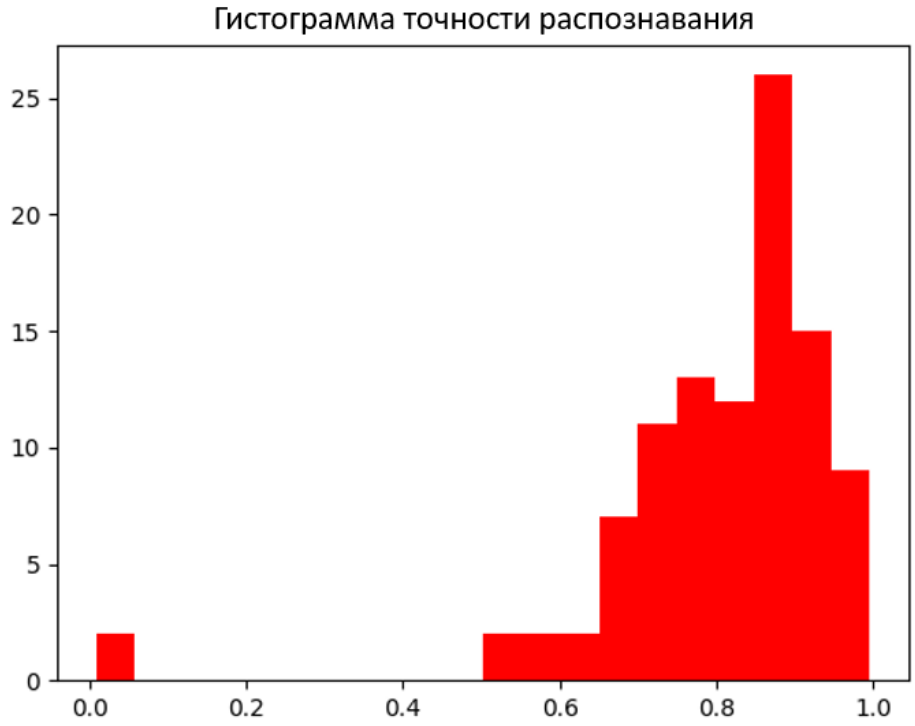

Рисунок 21 — Гистограмма точности распознавания классов сетью InceptionResNetV2

3.2. Реализация мобильного приложения

3.2.1. Конвертирование модели

Для использования в iOS приложении созданной вне Xcode модели машинного обучения требуется преобразовать её в фундаментальный для Apple девайсов формат CoreML модели.

Поскольку работа в MacOS происходила в виртуальной машине, а описанные в первом разделе действия на основной машине, первым делом потребовалось перенести созданный на предыдущем этапе работы файл модели в MacOS.

Для конвертирования модели использовался фреймворк CoreMLTools. Код для преобразования модели в CoreM (Листинг 7). Здесь указаны все параметры будущей модели CoreML, а именно: путь к файлу; исходный фреймворк; тип входных данных; название будущей модели, а также второстепенные параметры: автор; краткое описание модели, входных и результирующих данных. После запуска этого файла в активном каталоге появится модель `Food101.mlmodel`. Для использования модели в приложении она была перемещена в папку приложения.

Теперь, чтобы использовать модель CoreML для прогнозов в коде приложения, необходимо импортировать модуль CoreML и инициализировать ранее

импортированную модель. Также для отображения результатов в приложении зачастую необходимо как-то обрабатывать полученный результат.

```
import coremltools.converters as ct

mlmodel = ct.convert(model='food.h5', source="tensorflow", inputs=[ct.ImageType()])

mlmodel.author = 'Alina Leonova'
mlmodel.short_description = 'Picture prediction model'
mlmodel.input_description['input_1'] = 'Image'
mlmodel.output_description['Identity'] = 'Label'

mlmodel.save('Food101.mlmodel')
```

Листинг 7 — Файл convert.py

3.2.2. Реализация iOS приложения

Выбор изображения осуществляется с помощью функция `imagePickerController` (Листинг 8), вызов которой приводит к появлению соответствующего запроса (Рисунок 22).

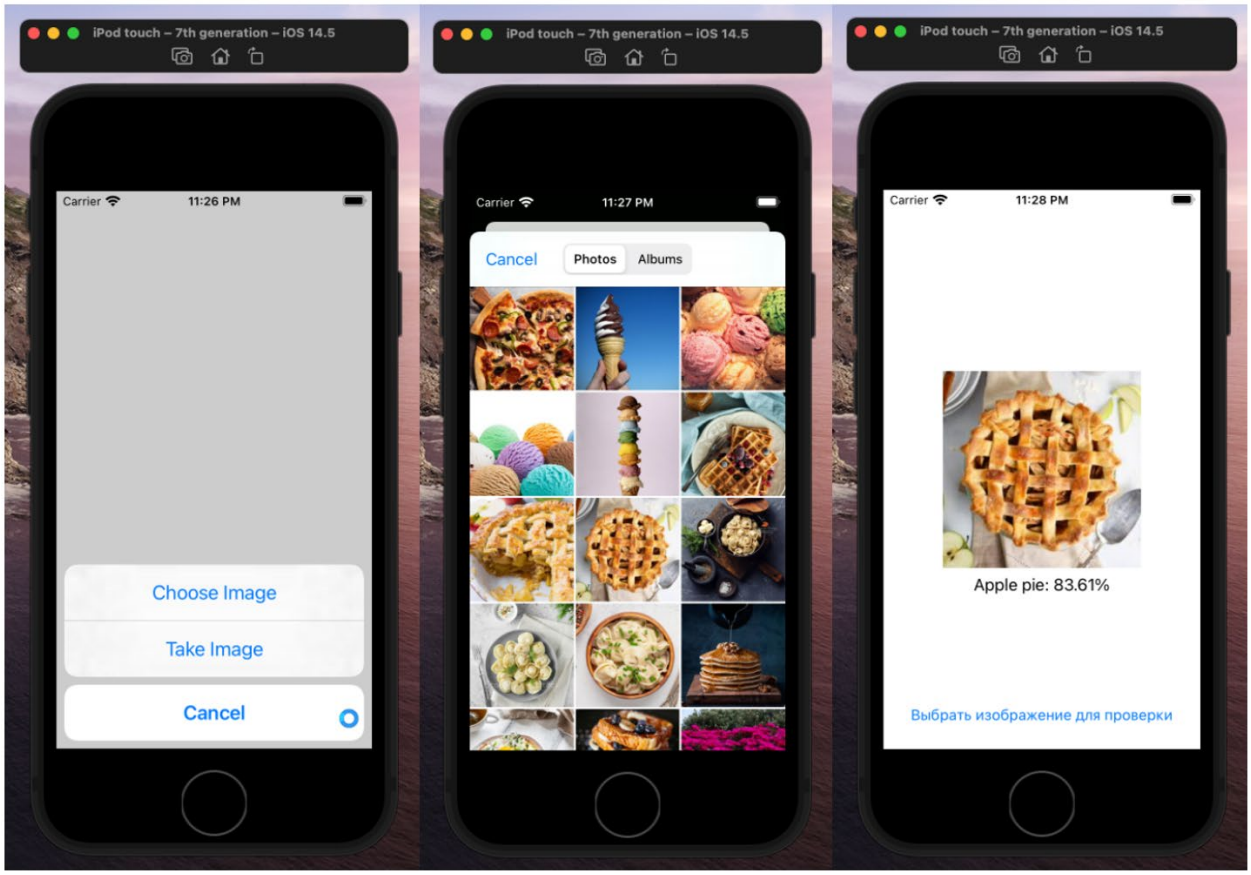

Рисунок 22 — главное окно и запрос на выбор изображения

```
func imagePickerController(_ picker: UIImagePickerController,
                                    didFinishPickingMediaWithInfo info: [String: Any])
    {
        dismiss(animated: true, completion: nil)

        guard let image = info["UIImagePickerControllerOriginalImage"] as? UIImage
        else
        {
            return
        }

        processImage(image)
    }
```

Листинг 8 — функция imagePickerController

Листинг всего файла `ViewController.swift` приведён в приложении Б.

В реализованном приложении за вызов и обработку результатов работы модели машинного обучения отвечает функция `processImage`(Листинг 9).

```
func processImage(_ image: UIImage)
    {
        let model = Food101()
        let size = CGSize(width: 256, height: 256)

        guard let buffer = image.resize(to: size)?.pixelBuffer()
        else
        {
            fatalError("Ошибка конвертации изображения!")
        }

        guard let result = try? model.prediction(input_1: buffer)
        else
        {
            fatalError("Ошибка классификации!")
        }

        var max:Float = -100
        var maxi:Int = 0
        for i in 0..<Int(result.Identity.strides[0])
        {
            if result.Identity[i].floatValue > max
            {
                max = result.Identity[i].floatValue
                maxi = i
            }
        }
        let maxs = String(format: "%.2f", max * 100.0)
        percentage.text = "\(getLabel(maxi)): \(maxs)%"
        imageView.image = image
    }
```

Листинг 9 — функция processImage

3.2.3. Результаты

На снимке экрана на рисунке 23 показаны: VMware macOS с открытым Xcode, открытый проект приложения, описание импортированной модели Food101 и встроенный в Xcode симулятор iOS, демонстрирующий работу приложение (с выбранным изображением и с прогнозом, что с вероятностью 79,26% на нем присутствует мороженое).

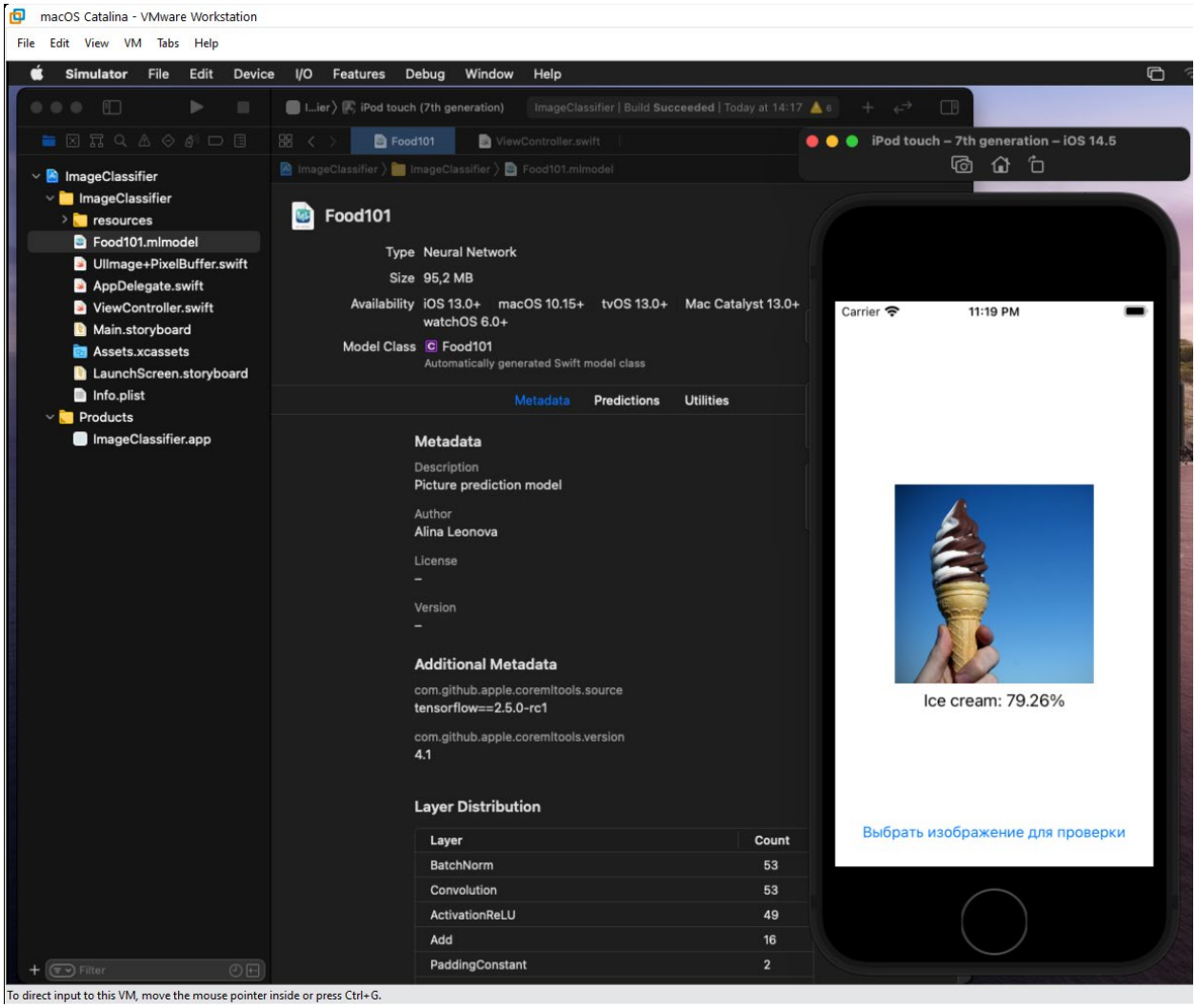

Рисунок 23 — Проект в Xcode, описание Food101.mlmodel и iOS симулятор с приложением для классификации еды по изображению

3.3. Оценка результатов и перспективы развития

В результате работы были выполнены все шаги по реализации мобильного приложения, использующего технологию машинного обучения: выбор, обучение и конвертация модели с помощью Python-приложений, а также создание в среде

XCode прототипа приложения, использующего эту модель. Можно предложить три основные направления возможного развития приложения:

- интеграция с базой данных, содержащий информацию по калорийности продуктов для ведения дневника питания;
- перенос на платформу Android на основе кроссплатформенной реализации интерфейса;
- реализация распределённого приложения, обеспечивающего доступ к дневнику питания с различных устройств, в том числе из web-браузера.

Заключение

В работе были изучены возможности использования технологии машинного обучения при разработке мобильных приложений.

Подготовлен обзор методов машинного обучения, а также их реализаций различными фреймворками языка Python. Проведено сравнение различных архитектур нейронных сетей, ориентированных на работу с классификацией изображений, а также обучение и выбор моделей для распознавания заданных классов изображений с помощью приложения, написанного на языке Python с помощью библиотеки Keros фреймворка Tensorflow.

Был обоснован выбор нативных средств разработки для платформы iOS и изучены доступные мобильные API для машинного обучения. В качестве основного средства разработки был выбран язык Swift в среде Xcode.

Практические результаты работы включают разработку структуры мобильного приложения для распознавания изображений продуктов питания, перенос в формат Core ML обученной модели, подготовленной с помощью приложения на языке Python, и разработку прототипа мобильного приложения для платформы iOS.

В качестве возможных перспектив развития предложена интеграция приложения с информацией по калорийности еды, а также возможный переход к распределённой и кроссплатформенной системе.

Литература

1. Леонова А. Д., Шорохов С. Г. Применение машинного обучения в iOS // Информационно-телекоммуникационные технологии и математическое моделирование высокотехнологичных систем: материалы Всероссийской конференции с международным участием. Москва, РУДН, 19–23 апреля 2021 г. – Москва : РУДН, 2021. – С. 190-195.
2. Машинное обучение [Электронный ресурс]. – URL: http://www.machinelearning.ru/wiki/index.php?title=Машинное_обучение.
3. Розенблатт Ф. Принципы нейродинамики: перцептроны и теория механизмов мозга. – 1965.
4. Mobile operating systems' market share worldwide from January 2012 to January 2021 [Электронный ресурс]. – URL: https://www.statista.com/statistics/272698/global-market-share-held-by-mobile-operating-systems-since-2009/.
5. Yakimov S. Кроссплатформенная разработка мобильных приложений в 2020 году [Электронный ресурс]. – URL: https://habr.com/ru/post/491926/.
6. Krizhevsky A. ImageNet Classification with Deep Convolutional Neural Networks.
7. Top Programming Languages 2020 [Электронный ресурс]. – URL: https://spectrum.ieee.org/at-work/tech-careers/top-programming-language-2020.
8. Топ-10 инструментов Python для машинного обучения и data-science [Электронный ресурс]. – URL: https://habr.com/ru/company/skillbox/blog/420819/.
9. Что такое Scikit Learn - гайд по популярной библиотеке Python для начинающих [Электронный ресурс]. – URL: https://datastart.ru/blog/read/chto-takoe-scikit-learn-gayd-po-populyarnoy-biblioteke-python-dlya-nachinayuschih.
10. Элбон К. Машинное обучение с использованием Python. Сборник рецептов. – 2019.
11. Scikit-learn API Reference.
12. Scikit-learn User Guide.
13. Жерон О. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow. – 2018.
14. Keras API reference [Электронный ресурс]. – URL: https://keras.io/api/.

15. Keras Applications [Электронный ресурс]. – URL: https://keras.io/api/applications/.
16. Core ML vs ML Kit: Which Mobile Machine Learning Framework Is Right for You? [Электронный ресурс]. – URL: https://heartbeat.fritz.ai/core-ml-vs-ml-kit-which-mobile-machine-learning-framework-is-right-for-you-e25c5d34c765.
17. Apple Inc. Machine Learning [Электронный ресурс]. – URL: https://developer.apple.com/machine-learning/.
18. Hollemans M. Machine Learning by Tutorials - Beginning Machine Learning for Apple and iOS. – 2018.
19. Рулин Р. Инструменты Apple для машинного обучения [Электронный ресурс]. – URL: https://habr.com/ru/company/redmadrobot/blog/418307/.
20. Mishra A. Machine Learning for iOS Developers. – 2020.
21. Thakkar M. Beginning Machine Learning in iOS CoreML Framework. – 2019.
22. How to create CoreML models [Электронный ресурс]. – URL: https://medium.com/@asteroidzone1/how-to-create-coreml-models-366d43f3e438.
23. Apple Inc. Create ML [Электронный ресурс]. – URL: https://developer.apple.com/machine-learning/create-ml/.
24. Розов Д. Тестирование и обзор Core ML [Электронный ресурс]. – URL: https://habr.com/ru/company/mobileup/blog/332500/.
25. Apple’s deep learning frameworks: BNNS vs. Metal CNN [Электронный ресурс]. – URL: http://machinethink.net/blog/apple-deep-learning-bnns-versus-metal-cnn/.
26. Усов В. Swift. Основы разработки приложений под iOS, iPadOS и macOS. 5-е изд., дополненное и переработанное. – СПб: Питер, 2020.
27. Image data preprocessing [Электронный ресурс]. – URL: https://keras.io/api/preprocessing/image/.

Приложение А — neurotest.py

```
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import InceptionResNetV2
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.applications.inception_v3 import InceptionV3

from tensorflow.keras.layers import GlobalAveragePooling2D, Dropout
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger
from tensorflow.keras.models import Model, load_model
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Input

import shutil
import os, sys
from os import listdir
from os.path import isfile, join
import argparse

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import collections
from collections import defaultdict

shape = ()

def make_path(p):
    return sys.path[0] + '/' + p

def make_model_path(args, p):
    return make_path('models/' + args.network + '/' + p)

def make_dataset_path(args, p):
    return make_path(args.dataset + '/' + p)

# случайная модификация изображений
# (поворот, сдвиг, растяжение, зеркалироваание,..)
def setup_generator(train_path, test_path, batch_size, dims):
    train_datagen = ImageDataGenerator(
        rotation_range=40,
        width_shift_range=0.2,
        height_shift_range=0.2,
        rescale=1. / 255,
        shear_range=0.2,
```

```
        zoom_range=0.2,
        horizontal_flip=True,
        fill_mode='nearest')

    test_datagen = ImageDataGenerator(rescale=1. / 255)

    train_generator = train_datagen.flow_from_directory(
        train_path,
        target_size=dims,
        batch_size=batch_size)

    validation_generator = test_datagen.flow_from_directory(
        test_path,
        target_size=dims,
        batch_size=batch_size)

    return train_generator, validation_generator

def load_image(img_path, dims, rescale=1. / 255):
    img = load_img(img_path, target_size=dims)
    x = img_to_array(img)
    x = np.expand_dims(x, axis=0)
    x *= rescale
    return x

def load_strings(file_path):
    with open(file_path) as f:
        strings = f.read().splitlines()
    return strings

def create_model(m_type, num_classes, dropout, shape):
    modelclass = globals()[m_type]
    base_model = modelclass(
            weights='imagenet',
            include_top=False,
            input_tensor=Input(
            shape=shape))

    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dropout(dropout)(x)
    predictions = Dense(num_classes, activation='softmax')(x)

    model_final = Model(inputs=base_model.input, outputs=predictions)

    return model_final

def train_model(model_final, train_generator, validation_generator,
                                                    callbacks, args):
```

```
    model_final.compile(
        loss='categorical_crossentropy',
        optimizer='adam',
        metrics=['accuracy'])

    model_final.fit_generator(train_generator,
        validation_data=validation_generator,
        epochs=args.epochs, callbacks=callbacks,
        steps_per_epoch=train_generator.samples//args.batch_size,
        validation_steps=validation_generator.samples//args.batch_size)

def load_model_by_name(args, network, num_classes, shape):
    model_final = create_model(network, num_classes, 0, shape)
    model_final.load_weights(make_path('models/' + network + '/' +
                                                    args.model_path))
    return model_final

def load_model(args, num_classes, shape):
    model_final = create_model(args.network, num_classes, 0, shape)
    model_final.load_weights(make_model_path(args, args.model_path))
    return model_final

def plot_accuracy_loss(args):
    log = pd.read_csv(make_path('models/'+args.network+
                                                '/history_log.csv'))

    fig, ax = plt.subplots(1, 2, figsize=(18, 8))

    # accuracy
    ax[0].plot(log['accuracy'], 'r')
    ax[0].plot(log['val_accuracy'], 'b')
    ax[0].set_title(args.network+' | Зависимость точности (accuracy)
                                                        от эпохи')
    ax[0].set(xlabel='epoch', ylabel='accuracy')
    ax[0].legend(['train', 'val'], loc='upper left')

    # loss
    ax[1].plot(log['loss'], 'r')
    ax[1].plot(log['val_loss'], 'b')
    ax[1].set_title(args.network+'. Зависимость потерь (loss) от эпохи')
    ax[1].set(xlabel='epoch', ylabel='loss')
    ax[1].legend(['train', 'val'], loc='upper left')

    # сохранение графика в файл
    plt.savefig(make_path('models/'+args.network+'/'+args.network+
                                            '_accuracy_loss.png'))

    # демонстрация графика
    plt.show()
```

```
def plots(args):
    networks = ['ResNet50', 'InceptionV3', 'InceptionResNetV2']
    colors = ['r', 'g', 'b']
    c = 0

    fig, ax = plt.subplots(2, 2, figsize=(18, 12))

    for n in networks:
        log = pd.read_csv(make_path('models/'+ n +'/history_log.csv'))

        ax[0,0].plot(log['accuracy'], colors[c])
        ax[1,0].plot(log['val_accuracy'], colors[c])
        ax[0,1].plot(log['loss'], colors[c])
        ax[1,1].plot(log['val_loss'], colors[c])

        c += 1

    ax[0,0].legend(networks, loc='upper left')
    ax[1,0].legend(networks, loc='upper left')
    ax[0,1].legend(networks, loc='upper right')
    ax[1,1].legend(networks, loc='upper right')

    fig.legend(networks, loc='upper left')

    # accuracy
    ax[0,0].set_title('Зависимость точности (accuracy) от эпохи')
    ax[0,0].set(xlabel='epoch', ylabel='accuracy')

    # val_accuracy
    ax[1,0].set_title('Зависимость точности (val_accuracy) от эпохи')
    ax[1,0].set(xlabel='epoch', ylabel='val_accuracy')

    # loss
    ax[0,1].set_title('Зависимость потерь (loss) от эпохи')
    ax[0,1].set(xlabel='epoch', ylabel='loss')

    # val_loss
    ax[1,1].set_title('Зависимость потерь (val_loss) от эпохи')
    ax[1,1].set(xlabel='epoch', ylabel='val_loss')

    # сохранение графика в файл
    plt.savefig(make_path('models/models_accuracy_loss.png'))

    # демонстрация графика
    plt.show()

def train(args):
    X_train, X_test = setup_generator(make_dataset_path(args, 'train')
```

```
            , make_dataset_path(args, 'test'), args.batch_size, shape[:2])
    print(X_train)
    callbacks = []
    callbacks.append(ModelCheckpoint(filepath=make_model_path(args,
                                args.dataset + '-weights.epoch-{epoch:02d}-
                                      val_loss-{val_loss:.4f}-val_accuracy-
              {val_accuracy:.4f}.hdf5'), verbose=1, save_best_only=True))
    callbacks.append(CSVLogger(make_model_path(args, 'history_log.csv')))

    model_final = create_model(args.network, X_train.num_classes,
                                                          args.dropout, shape)
    train_model(model_final, X_train, X_test, callbacks, args)

def testimage(args):
    classes = load_strings(make_dataset_path(args, 'meta/classes.txt'))
    trained_model = load_model(args, len(classes), shape)
    image = load_image(args.image_path, shape[:2])
    preds = trained_model.predict(image)
    print("класс изображения: ", classes[np.argmax(preds)])
    print("с вероятностью: ", np.max(preds)*100, "%")


def fulltest(args):
    classes = load_strings(make_dataset_path(args, 'meta/classes.txt'))
    trained_model = load_model(args, len(classes), shape)

    test_images = load_strings(make_dataset_path(args, 'meta/test.txt'))

    n_true, n_false = 0, 0
    corrects = collections.defaultdict(int)
    incorrects = collections.defaultdict(int)
    for i in test_images:
        (iclass, iname) = i.split('/')
        image_path = make_dataset_path(args, 'test/'+i+'.jpg')
        image = load_image(image_path, shape[:2])
        preds = trained_model.predict(image)

        if classes[np.argmax(preds)] == iclass:
            n_true += 1
            corrects[iclass] += 1
        else:
            n_false += 1
            incorrects[iclass] += 1

    print('\nПравильно распознаны: ', n_true)
    print(n_true/(n_true+n_false)*100,' %')
    print('\nОшибочно распознаны: ', n_false)
    print(n_false/(n_true+n_false)*100,' %')
```

```
    print('Точность распознавания каждого класса:')
    class_accuracies = {}
    for ix in range(101):
        class_accuracies[ix] = corrects[classes[ix]]/250
        print(classes[ix], class_accuracies[ix])

    fig = plt.hist(list(class_accuracies.values()), bins=20, color='r')
    plt.title('Гистограмма точности распознавания классов')

    plt.savefig('histogram.png')
    plt.show()


def prepare(args):
    if os.path.isdir(make_dataset_path(args, 'test')) or
                        os.path.isdir(make_dataset_path(args, 'train')):
        return

    def generate_dir_file_map(path):
        dir_files = defaultdict(list)
        with open(path, 'r') as txt:
            files = [l.strip() for l in txt.readlines()]
            for f in files:
                dir_name, id = f.split('/')
                dir_files[dir_name].append(id + '.jpg')
        return dir_files

    train_dir_files = generate_dir_file_map(make_dataset_path(args,
                                                    'meta/train.txt'))
    test_dir_files = generate_dir_file_map(make_dataset_path(args,
                                                     'meta/test.txt'))

    def ignore_train(d, filenames):
        print(d)
        subdir = d.split('/')[-1]
        to_ignore = train_dir_files[subdir]
        return to_ignore

    def ignore_test(d, filenames):
        print(d)
        subdir = d.split('/')[-1]
        to_ignore = test_dir_files[subdir]
        return to_ignore

    shutil.copytree(make_dataset_path(args, 'images'),
           make_dataset_path(args, 'test'), ignore=ignore_train)
    shutil.copytree(make_dataset_path(args, 'images'),
           make_dataset_path(args, 'train'), ignore=ignore_test)
```

```
def main():
    modes = {
        'prepare':      prepare,
        'train':        train,
        'testimage':    testimage,
        'fulltest':     fulltest,
        'plot':         plot_accuracy_loss,
        'plots':        plots
    }

    parser = argparse.ArgumentParser(description='Neural Network Analyser
                                                    (NeuroTestAL)')
    parser.add_argument('-m',   help='режим работы:
                            prepare/train/testimage/fulltest/plot/plots',
                                dest='mode', type=str, default='train')
    parser.add_argument('-b',   help='размер пакета данных',
                                dest='batch_size', type=int, default=32)
    parser.add_argument('-p',   help='имя сохраненной модели',
                                dest='model_path', type=str,
                                                    default='food.hdf5')
    parser.add_argument('-i',   help='путь до тестового изображения',
                                dest='image_path',type=str, default='')
    parser.add_argument('-s',   help='каталог набора данных',
                                dest='dataset', type=str,
                                                     default='food-101')
    parser.add_argument('-e',   help='количество эпох',
                                dest='epochs', type=int, default=25)
    parser.add_argument('-d',   help='величина dropout',
                                dest='dropout', type=float, default=0.2)
    parser.add_argument('-n',   help='выбор сети (ResNet50,
                                        InceptionV3, InceptionResNetV2)',
                                dest='network', type=str,
                                                default='InceptionResNetV2')
    parser.add_argument('-x',   help='ширина целевого изображения',
                                dest='x', type=int, default=256)
    parser.add_argument('-y',   help='высота целевого изображения',
                                dest='y', type=int, default=256)

    args = parser.parse_args()
    global shape
    shape = (args.x, args.y, 3)

    #проверка и создание при необходимости каталога для моделей
    if not os.path.exists(make_path('models/' + args.network)):
        os.makedirs(make_path('models/' + args.network))
    modes[args.mode](args)

if __name__ == '__main__':
    main()
```

Приложение Б — ViewController.swift

```
import UIKit
import CoreML

class ViewController: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate
{

    @IBOutlet weak var imageView: UIImageView!
    @IBOutlet weak var percentage: UILabel!

    override func viewDidLoad()
    {
        super.viewDidLoad()
    }

    @IBAction func buttonPressed(_ sender: Any)
    {
        let alert = UIAlertController(title: nil, message: nil,
preferredStyle: .actionSheet)
        let imagePickerView = UIImagePickerController()
        imagePickerView.delegate = self

        alert.addAction(UIAlertAction(title: "Choose Image", style:
.default)
        { _ in
            imagePickerView.sourceType = .photoLibrary
            self.present(imagePickerView, animated: true, completion: nil)
        })

        alert.addAction(UIAlertAction(title: "Take Image", style:
.default)
        { _ in
            imagePickerView.sourceType = .camera
            self.present(imagePickerView, animated: true, completion: nil)
        })

        alert.addAction(UIAlertAction(title: "Cancel", style: .cancel,
handler: nil))
        self.present(alert, animated: true, completion: nil)
    }

    func imagePickerControllerDidCancel(_ picker:
UIImagePickerController)
    {
        dismiss(animated: true, completion: nil)
    }
```

```
    func imagePickerController(_ picker: UIImagePickerController,
didFinishPickingMediaWithInfo info: [String: Any])
    {
        dismiss(animated: true, completion: nil)

        guard let image = info["UIImagePickerControllerOriginalImage"] as?
UIImage
        else
        {
            return
        }

        processImage(image)
    }

    func getLabel(_ n: Int) -> String
    {
        if let filepath = Bundle.main.path(forResource: "labels", ofType:
"txt")
        {
            do
            {
                let contents = try String(contentsOfFile: filepath)
                return contents.components(separatedBy: "\n")[n]
            }
            catch {}
        }
        return ""
    }

    func processImage(_ image: UIImage)
    {
        let model = Food101()
        let size = CGSize(width: 256, height: 256)

        guard let buffer = image.resize(to: size)?.pixelBuffer()
        else
        {
            fatalError("Ошибка конвертации изображения!")
        }

        guard let result = try? model.prediction(input_1: buffer)
        else
        {
            fatalError("Ошибка классификации!")
        }

        var max:Float = -100
        var maxi:Int = 0
        for i in 0..<Int(result.Identity.strides[0])
```

```
            {
                if result.Identity[i].floatValue > max
                {
                    max = result.Identity[i].floatValue
                    maxi = i
                }
            }
            let maxs = String(format: "%.2f", max * 100.0)
            percentage.text = "\(getLabel(maxi)): \(maxs)%"
            imageView.image = image
        }

        override func didReceiveMemoryWarning()
        {
            super.didReceiveMemoryWarning()
        }

    }
```