МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»
Институт математики, физики и информационных технологий
(наименование института полностью)

Кафедра «Прикладная математика и информатика»
(наименование)

01.04.02 Прикладная математика и информатика
(код и наименование направления подготовки)

Математическое моделирование
(направленность (профиль))

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)

на тему «Исследование сверточных нейронных сетей для создания приложений дополненной реальности»

Студент Д.В. Овчинников
(И.О. Фамилия) (личная подпись)

Научный руководитель к.т.н., доцент, Э.В. Егорова
(ученая степень, звание, И.О. Фамилия)

Тольятти 2020

Оглавление

Введение

Чтобы понять, для чего проект был реализован, важно узнать не только историю изученных областей, но и перспективы использования технологий. В разделе «Перспективы» рассматривается текущее и потенциальное использование этих технологий, а также какие существуют перспективы у этой отрасли. В разделе «Предыдущие исследования» рассматриваются некоторые из предыдущих исследований, которые были проведены в областях, изучающихся в данной работе. В разделе «Цель работы» цели проекта. Раздел «Макет запланированного приложения» иллюстрирует макет того, как должен выглядеть конечный продукт.

Идея дополненной реальности, которую часто называют просто AR, состоит в том, чтобы визуализировать виртуальные объекты в реальном мире. Это обычно требует аппаратного обеспечения в виде камеры и дисплея, процессора и программного обеспечения. Обычными устройствами, способными к AR, являются HoloLens [1], Google Glass [2] и огромное количество мобильных устройств, таких как Apple iPhone X [3]. Общий интерес к AR за последние годы, несомненно, вырос, благодаря мобильным играм и приложениям, таким как Niantic's Pokémon Go [4] и IKEA Place [5], индустрия AR достигла общего интереса, и источники предполагают, что это может быть индустрия с бюджетом в 90 миллиардов долларов к 2022 году [6]. Но дополненная реальность достигла не только простых пользователей, технология также вызвала интерес в нескольких отраслях. Одним из таких проектов является Fieldbit Hero [7], платформа, позволяющая техническим специалистам мгновенно получать аннотации AR на свои AR-устройства от инженера, позволяя техническому персоналу получать визуальные инструкции и одновременно работать в режиме громкой связи. Однако на данный момент большая часть локализации и идентификации объектов в AR осуществляется с помощью обнаружения маркеров [8]. Альтернативой этому было бы использование сверточных нейронных сетей, поскольку было

доказано, что они очень полезны для обнаружения объекта на изображениях. Шумиха вокруг нейронных сетей была особенно сильной с тех пор, как AlexNet набрал высокие баллы в конкурсе ImageNet LSVRC-2010 и конкурсе ImageNet ILSVRC-2012 [9], которые являются соревнованиями по классификации изображений. Они совершили эти подвиги, используя новые методы, например, отсев. С тех пор было создано несколько различных типов сетевых моделей. Теперь есть сетевые модели, которые могут определять, как местоположение объекта на изображении, так и объект, за один раз. Одной из таких сетей является YOLO [10]. Применив такую сеть, необходимость в маркировке объектов значительно упала, и это могло бы открыть для многих других возможных вариантов использования AR.

Цель работы - объединить дополненную реальность с обнаружением объекта и его распознаванием, чтобы выяснить, возможно ли это и есть ли фактический вариант использования для этого метода.

Объект исследования – разработка приложения, совмещающего нейронные сети и дополненную реальность.

Предмет исследования – приложение для сборки мебели с элементами дополненной реальности.

Гипотеза исследования состоит в том, что можно объединить нейронные сети с дополненной реальностью в одном приложении если:

- подобран метод обнаружения и локализации объектов
- нейронная сеть правильно обучена
- создано приложение, сочетающее сверточную нейронную сеть и дополненную реальность

Исходя из цели исследования и для проверки выдвинутой гипотезы, необходимо решить следующие **задачи:**

- подобрать метод обнаружения и локализации объектов
- создать и обучить сверточную нейронную сеть
- написать приложение, использующее сверточные нейронные сети и дополненную реальность

- создать графический интерфейс
- провести тестирование полученного приложения.

Научная новизна исследования состоит в совмещении нейронных сетей и дополненной реальности в мобильном приложении.

Положения, выносимые на защиту:

- Приложение для сборки мебели с элементами дополненной реальности.
- Компьютерная и математическая модели приложения с использованием нейронных сетей и дополненной реальности.

Структура магистерской диссертации. Работа состоит из введения, 4 глав, заключения.

Во введении обосновываются актуальность темы исследования, определяются объект, предмет, цель, ведущая идея, выдвигается гипотеза и формулируются задачи работы, характеризуются научная новизна, теоретическая и практическая значимость результатов исследования.

В первой главе «Теоретические основы дополненной реальности и нейронных сетей» описана историческая справка, а также то, как работает дополненная реальность. Объясняется несколько различных типов AR. Затем краткая история AR представлена в разделе. После этого описан метод сопоставления среды, называемый S.L.A.M. Описывается инструментарий ARKit. Далее внимание уделено нейронным сетям. Во-первых, некоторые теоретические основы, окружающие нейронные сети в целом, а затем, в частности.

Во второй главе «Анализ существующих методов и подходов для создания приложения дополненной реальности» раскрывается о предыдущих исследованиях в области mobile augmented reality. Так же рассказывается о способе обнаружения объектов с традиционным машинным обучением, описан метод обнаружения лиц Виолы-Джонса. Далее рассматривается прототип matlab для обнаружения объектов, и описаны результаты его работы.

В третьей главе «Объединение выбранных методов в приложении дополненной реальности» создается макет запланированного приложения. Решается вопрос сбора и дополнения данных для обучения нейронной сети. Описываются методы для обнаружения объекта.

В четвертой главе «Готовое приложение» описывается само приложение, показаны диаграмма классов, показывается готовый интерфейс, проводится его тестирование, а также представлены результаты опроса пользователей, опробовавших приложение.

В заключении представлены основные результаты поставленных задач исследования и сделаны следующие выводы:

Возможность объединения дополненной реальности и нейронных сетей существует.

Данное приложение гораздо удобнее бумажного руководства по сборке мебели.

Содержит 38 рисунков, 3 таблицы, 47 источников, 12 приложений. Основной текст работы изложен на 77 страницах.

Глава 1 Теоретические основы дополненной реальности и нейронных сетей

1.1 Дополненная реальность

Существуют разные типы дополненной реальности. Некоторые из них основаны на маркерах, на основе местоположения, наложения и проекции. Описание каждого из них:

– AR на основе маркеров - это когда маркеры в виде изображений должны быть размещены в реальном мире и обнаружены приложением. Виртуальный объект отображается поверх этих маркеров. Примером может служить изображение в журнале, которое, когда на него указывает камера, приложение отображает трехмерный объект сверху. Пример показан на рисунке 1.1.

– AR на основе местоположения - это когда содержимое на экране пользователя отличается в зависимости от местоположения пользователя. Этот тип сильно зависит от сигнала GPS. Примером того, где это может быть полезно, является музей, в котором пользователю может быть предоставлена различная информация в зависимости от того, в какой комнате он находится.

– AR, основанный на наложении, использует распознавание объектов, чтобы улучшить этот объект некоторой визуальной информацией. Он заменяет реальный объект на расширенный виртуальный. Он может быть использован в розничной продаже для отображения различных узоров на предмете одежды.

– AR на основе проекции - это когда виртуальный объект можно поместить в комнату, чтобы он выглядел так, как если бы он был там. Популярным примером этого будет IKEA Place.

Рисунок 1.1 - Виртуальный объект в виде автомобиля, отображаемый поверх маркера в журнале

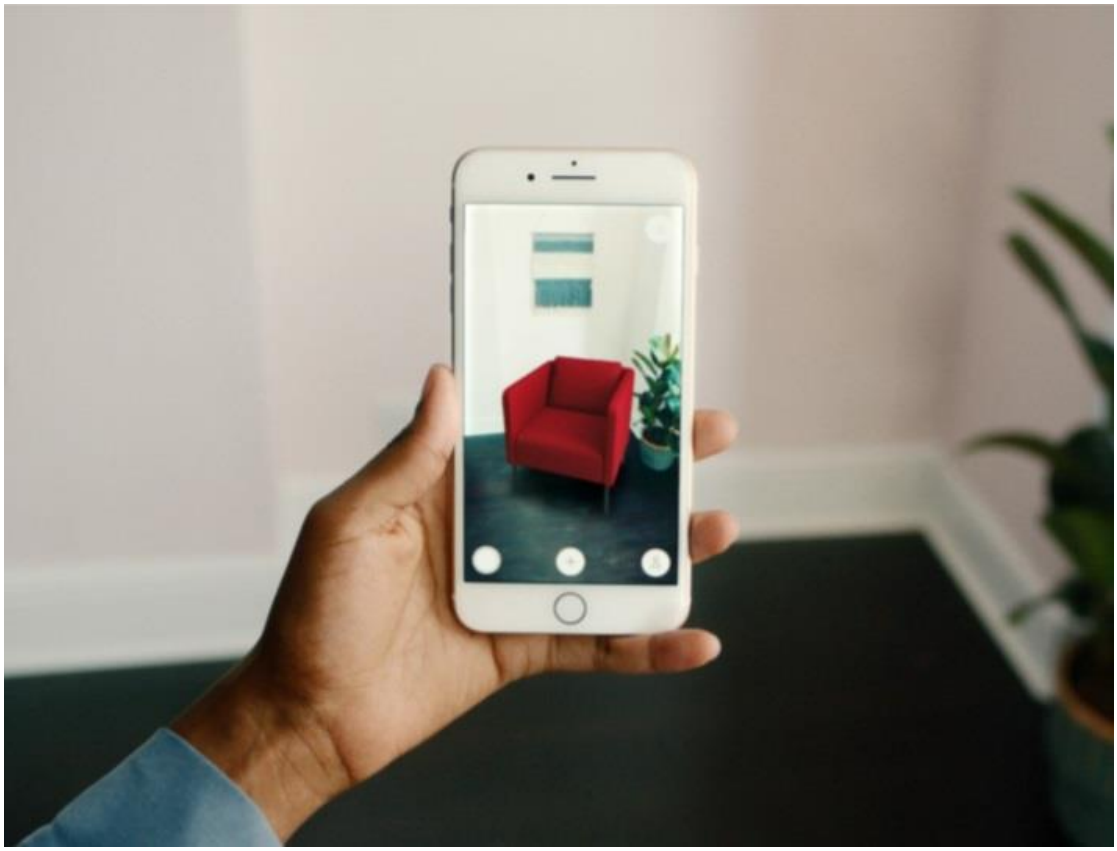

Рисунок 1.2 - Виртуальное кресло, добавляемое в комнату в приложении IKEA Place

В этой работе будет использован AR на основе проекций с формой распознавания. Прогнозы будут представлены в виде указателей, используемых в качестве инструкций для пользователя, чтобы выполнить следующий шаг при сборке мебели. Эти инструкции могут визуализировать, как части будут выглядеть после завершения шага, и показывать стрелки-указатели между частями, которые должны быть соединены.

1.2 Использование дополненной реальности

Идея дополненной реальности существует уже давно, эта фраза используется около 30 лет, но лишь недавно технология стала реальной. Это происходит главным образом из-за того, что она становится достаточно доступной, для использования обычным человеком. Сегодня каждый может просто загрузить приложение на смартфон, чтобы насладиться технологией.

Краткая история развития AR:

1968 г., The Sword of Damocles - первая установленная гарнитура. Это устройство было установлено на голове и могло отображать кубический каркас, плавающий в воздухе. Его изобрел Ivan Sutherland.

1975 Myron Krueger - Videoplace. Использование камер для взаимодействия с цифровым миром и тенями. Это приложение использовалось для рисования или для игры в простые видеоигры с тенью вашей руки [16].

1990 Впервые термин «Дополненная реальность» использовался исследователем компании Boeing, Tom Caudell.

2009 AR выходит в Интернет в форме набора инструментов с открытым исходным кодом под названием ARToolKit.

2017 Apple запускает ARKit, a Google запускает AR Core.

Один из видов использования дополненной реальности это S.L.A.M (simultaneous localization and mapping) - способ для машины узнать окружение, в котором она находится. Система используется в автономных

роботах, но также полезна в дополненной реальности [17]. На рисунке 1.3 робот перемещается по комнате, собирая данные. Камеру робота можно увидеть в левом нижнем углу. Справа - трехмерная модель с распознанными характерными точками.

iPhone X так же может отслеживать несколько опорных точек в пространстве и создавать из них трехмерную модель окружения, используя форму техники S.L.A.M. Это достигается путем сохранения карты функций при отслеживании пути, по которому идет наблюдатель. Это возможно благодаря ряду аппаратных компонентов, включая гироскоп, акселерометр и компас. [46]

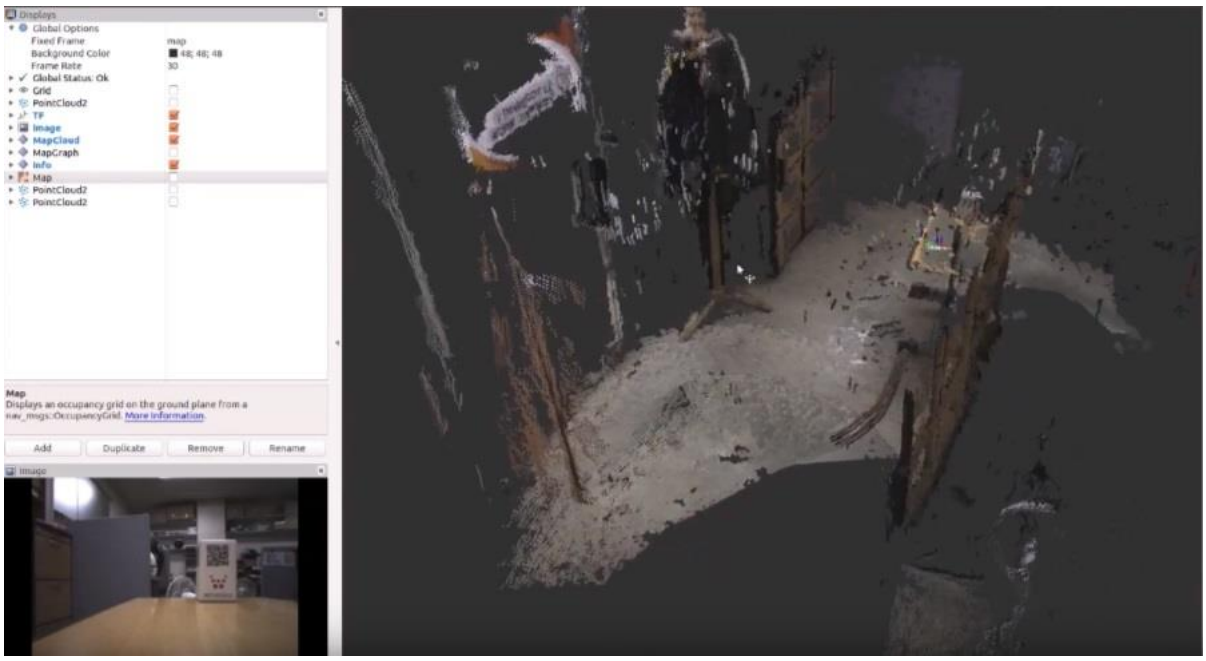

Рисунок 1.3 - Робот, выполняющий S.L.A.M в среде, и созданная 3D-модель

По мимо S.L.A.M. дополненная реальность используется в ARKit. Самый простой способ использования ARKit посредством Xcode. Языком кодирования может быть Objective-C или Swift. ARKit работает аналогично SceneKit, где сцена, которая в основном является трехмерной средой, содержащая трехмерные модели, загружается при запуске и взаимодействует с ними. ARScene содержится внутри ARSCNView (AR Scene View), который

также имеет ARSession, который управляет отслеживанием движения и обработкой изображения с камеры. Чтобы ARScene работал, он должен иметь запущенную ARSession. Сеанс начинается с конфигурации ARCon. Эта конфигурация может быть разных видов, наиболее распространенными из которых являются конфигурации ARWorldTrackingCon и ARFaceTrackingCon. Для этого проекта будет использован ARWorldTrackingCon, так как для отслеживания лица используется фронтальная камера. Пример настройки конфигурации и запуска сеанса с использованием Swift приведен в приложении А. Но прежде чем сеанс может быть запущен, ARScene должен существовать и быть загруженным. ARScene - это обычный .scn-файл, имеющий начальные узлы, которые вместе образуют начальную среду, с которой пользователь будет взаимодействовать. Пример того, как выглядит эта сцена, показан на рисунке 1.4. Чтобы загрузить сцену, необходимо создать новый файл .scn в папке art.scnassets и извлечь его, как видно из фрагмента кода в приложении Б.

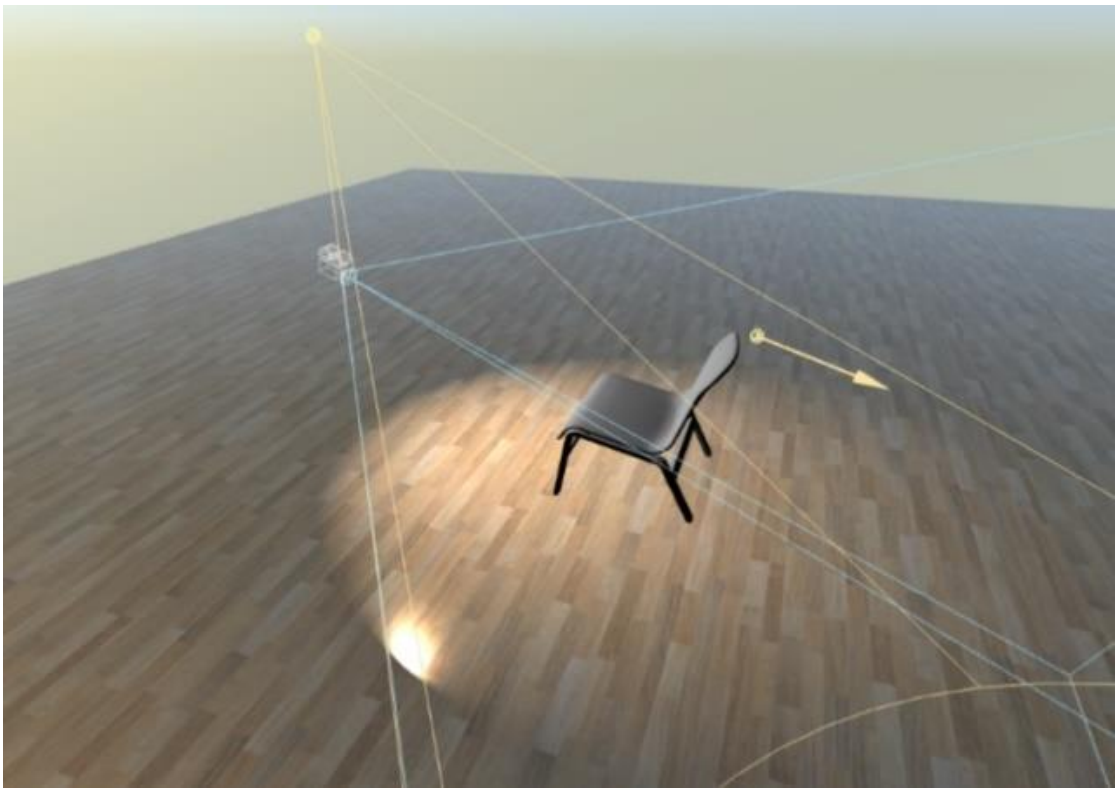

Рисунок 1.4 - Пример трехмерной сцены, созданной в XCode с камерой, плоскостью, направленным светом и тремя геометрическими узлами

Когда ARScene обнаруживает объекты, изображения, плоскости и т. д., он вызывает функцию рендеринга. Эту функцию можно реализовать, настроив ViewController на соответствие ARSCNViewDelegate. ARScene добавляет привязку (ARAnchor) и узел (SCNNode) для места, где он его обнаружил, и это можно использовать внутри функции для рендеринга новых узлов или другой логики. Код, который использован для этого, представлен в приложении В.

Таким образом, был проанализирован механизм создания приложений с использованием технологий AR, и для реализации дополненной реальности в запланированном приложении можно использовать ARKit.

1.3 Описание работы и способы использования нейронных сетей

Сегодня существует множество различных методов машинного обучения. В связи с его высокой эффективностью и актуальностью, эта работа сосредоточена на высоко популярных методофизических нейронных сетях. Вариант, сверточно-нейронные сети, является проверенным методом для работы с изображениями и поэтому очень актуален для этого проекта.

Искусственный нейрон является простейшей формой нейронной сети. Имеет набор входов и выходов. Искусственный нейрон сначала суммирует все входные значения, x, умноженные на значение веса, w. После этого он передает эту сумму через функцию активации. Функция активации может быть любой: от простой $f(x) = x$ до более сложной сигмоидальной функции, в зависимости от необходимости. Смещение также существует в каждом узле, который не основан на каком-либо входе. Функция смещения в объективном нейроне не похожа на то, что делает m в $y = kx + m$. Это дает функции возможность перемещаться вверх и вниз по графику для большей возможности разделения набора данных. Смещение обычно игнорируется при иллюстрации искусственного нейрона.

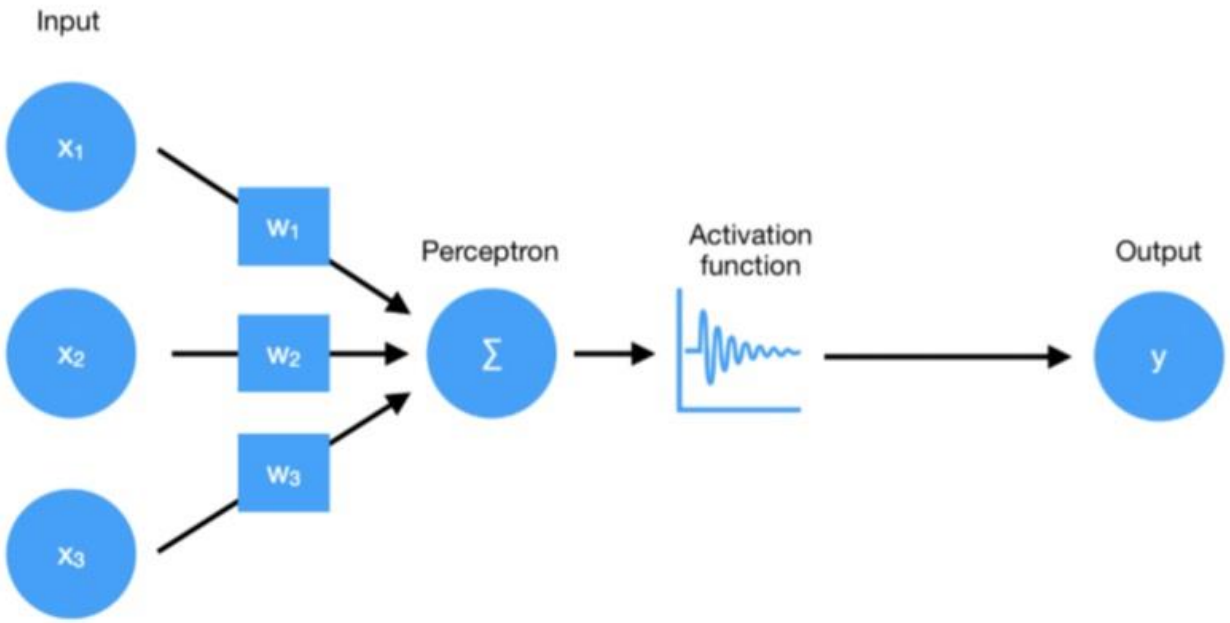

Рисунок 1.5 - Иллюстрация искусственного нейрона, простейшей модели нейронной сети

Искусственный нейрон обладает способностью рисовать только одну линию и, следовательно, способен разбивать только простые наборы данных. Выход искусственного нейрона описывается следующей формулой:

$$y = b + \sum_{i=1}^{n} x_i * w_i, \quad (1)$$

$$y = f(b + \sum_{i=1}^{n} x_i * w_i). \quad (2)$$

где b - значение смещения, x - вход, w - вес для этого входа, n - количество входов, а $f\ (*)$ - функция активации.

Функция активации $\varphi(v_i)$ получает сумму от входных данных, входных сигналов и передает их через функцию перед выдачей выходных данных. Это полезно, когда, например, выходные данные должны находиться в интервале между 0 и 1, или, возможно, когда отрицательные значения не имеют смысла. Обычно функции активации относятся к слоям, а не к отдельным узлам.

Часто используемые функции - это ReLu, Tanh, Sigmoid и Softmax. ReLu описывается как:

$$f\ (x)\ =\ max\ (0, x)\ (3)$$

и является хорошим вариантом, когда отрицательные значения следует игнорировать. Это также хороший выбор, чтобы сеть не становилась тяжелой в вычислительном отношении, поскольку она выводит только одно и то же входное значение, но устанавливает все отрицательные значения в ноль. Обычно это хороший выбор в больших сетях с большим количеством нейронов, которые могут стать медленными.

Параметр Tahn имеет большую гиперболическую функцию и используется для вывода -1 или 1 как бинарный оператор. В отличие от бинарного оператора, производная Tanh всегда определяется, что делает возможным обратное распространение.

Sigmoid описывается как:

$$\frac{e^x}{1+e^x}\ (4)$$

и работает как функция Tanh, но сохраняет значения от 0 до 1 и вместо этого устанавливает $y\ (0)\ =\ 0{,}5$.

Softmax немного сложнее. Он берет вектор v размерности n и превращает его в вектор $\sigma(v)$ того же измерения, где $\sum_{i=1}^{n} \sigma_1(v) = 1$ и каждое значение элемента находится между нулем и единицей.

Каждый элемент в массиве:

$$\sigma_i(v) = \frac{e^{v_i}}{\sum_{j=1}^{n} e^{v_k}}\ (5)$$

Этот вывод хорош для описания вероятности правильности каждого элемента и поэтому обычно используется в выходном слое.

При обучении модели, она сначала делает предположение, какой правильный выбор основан на случайных начальных значениях. В начале модель обычно ошибочна, и тогда важно знать, насколько она ошибочна и в каком направлении она должна идти. Для этого модель использует функцию потерь для расчета этой ошибки. Для различных приложений функция потерь

может быть различной. Один из способов вычислить ошибку E - просто взять предсказанное значение y_p и вычесть его с правильным значением y_c.

$$E = y_p - y_c \text{ (6)}$$

Иногда признак ошибки не важен. Тогда вместо этого можно рассчитать абсолютное значение. $E = |y_p - y_c|$.

Другой подход - это среднеквадратическая ошибка, которая возводит в квадрат разность и принимает среднее значение для нескольких прогнозов. n - количество предсказаний.

$$E(n) = \frac{1}{n} * \sum^{n}(y_p - y_c)^2 \text{ (7)}$$

Как только ошибка будет рассчитана, сеть внесет изменения в себя, чтобы улучшить производительность за счет уменьшения значения ошибки. Это называется обратным распространением. Принцип, лежащий в основе обратного распространения, заключается в том, чтобы вернуться назад от выходного узла и изменить значения веса, ω. Оптимизаторы решают, как должны изменяться веса.

Популярный метод минимизации ошибки - это использовать Gradient Decent, который шаг за шагом работает в обратном направлении, минимизируя ошибку. Направление определяется путем взятия производной функции ошибки. Новое значение веса рассчитывается соответственно.η - это выбранный параметр, называемый скоростью обучения.

$$\Delta\omega_{ik} = -\eta \frac{\delta E}{\delta \omega_{ik}} \text{ (8)}$$

где функция ошибки

$$E = \frac{1}{N}\sum_{i=1}^{N} E(n) \text{ (9)}$$

где N - общее количество нейронов. Это ведет к

$$\Delta\widehat{\omega_{ik}} = \frac{1}{N}\sum_{i=1}^{N} \Delta\omega_{ik0}(n) \text{ (10)}$$

$E\,(n)$ может быть, например, среднеквадратичной ошибкой, упомянутой выше. Наиболее распространенная форма градиентного спуска

называется Stochastic gradient decent, SGD. Различие состоит в том, что SGD оценивает только на небольшом числе узлов P и обновляет все веса из наблюдения.

$$\Delta\widehat{\omega_{ik}} = \frac{1}{P}\sum_{i=1}^{P}\Delta\omega_{ik0}(p) \quad (11)$$

Adam [18], сокращение от Adaptive moment estimation, является еще одним популярным оптимизатором. Причина в том, что во многих различных случаях использования он оказался очень эффективным алгоритмом. Adam является своего рода комбинацией использования RMSPROP [19] и SGD с импульсом. Не вдаваясь в подробности, SGD с импульсом следит за тем, в каком направлении идет улучшение, и поддерживает его. Оптимизация идет быстрее, когда направления совпадают, и медленнее, когда они различны. RMSPROP делает нечто подобное и использует скользящее среднее каждого значения веса, и важность предыдущих значений можно контролировать параметром. Он также использует только знак направления, а не его значение. Также RMSPROP имеет индивидуальные веса. Adam сохраняет среднее значение как прошлых градиентов, так и квадратов градиентов прошлого. Полное математическое описание Adam описывается формулой ниже, где t - индекс итерации времени.

$$\omega_i(t+1) = \omega_i(t) - \eta\frac{m_i}{\sqrt{v_i}+e} \quad (12)$$

где

$$m_i(t+1) = \beta_1 m_1(t) + (1-\beta_1)\frac{\delta E(t)}{\delta\omega_i}, \quad (13)$$

$$v_i(t+1) = \beta_2 v_i(t) + (1-\beta_2)(\frac{\delta E(t)}{\delta\omega_i})^2. \quad (14)$$

и η, β_1 и β_2 - настраиваемые параметры и e небольшое значение для предотвращения деления уравнения на ноль.

Есть много других оптимизаторов, таких как Adagrad, Adadelta, Nadam [20]. Все они имеют свои преимущества и варианты использования.

С увеличением количества слоев и количества нейронов параметры и сложность сетей начинают расти. Так же, как и время обучения, размер модели и ресурсозатратность выполнения прогнозов. Таким образом, эти сети способны описывать гораздо более сложные наборы данных. Но в результате этого риск перегрузки становится слишком велик, а вследствии увеличивается и вероятность того, что новые наборы данных не будут распознаны. Вот почему сложная сеть не всегда нужна. На рисунке 1.6 приведен пример того, как может выглядеть сеть с несколькими уровнями.

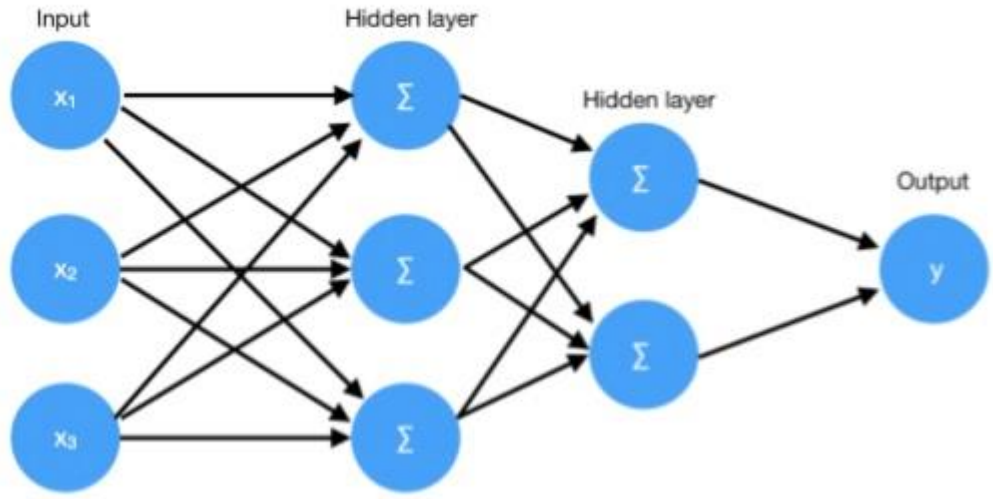

Рисунок 1.6 - Два полностью связанных слоя. Один с 3 нейронами и один с 2 нейронами

При обучении нейронной сети нужно быть осторожным, чтобы не перетренировать модель, иначе может произойти перегрузка. Перегрузка - это когда модель действительно хорошо предсказывает данные, на которых она обучается, но не может точно предсказать новые данные. Это потому, что она начинает уделять слишком много внимания деталям, а в некоторых случаях даже шуму. По этой причине она не в состоянии уловить общие тенденции, то более ценно. Иллюстрация приведена на рисунке 1.7. Слева - обобщенная обученная модель, а справа - модель с перетренированностью.

Есть несколько методов, которые могут помочь решить проблему перегрузки. Как правило, они включают в себя рост размеров небольшого веса. Гипотеза заключается в том, что, если вес намного больше, чем у остальных, он также имеет большее влияние на окончательный прогноз. Таким образом, небольшая деталь в данных может иметь гораздо большее влияние, чем общая тенденция.

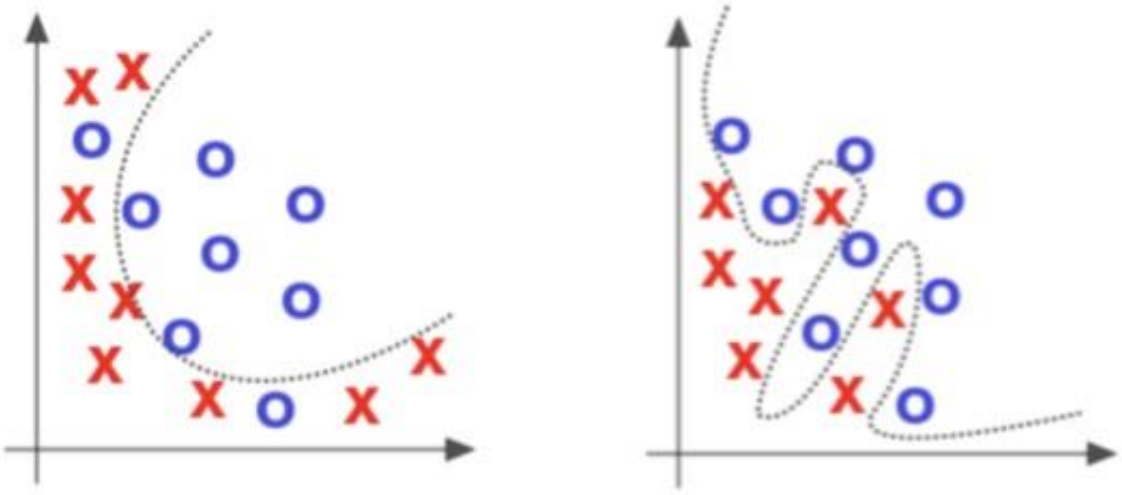

Рисунок 1.7 - Пере обстановка набора данных. Изображение взято с OReilly.com [21]

Один из способов сделать это - обрезать случайные связи между слоями в эпохе, указав определенное количество. Таким образом, модель не полагается на небольшое количество узлов, чтобы делать правильные прогнозы. Этот метод называется dropout [22].

Другой способ - ввести случайный шум на слое во время тренировки. Это работает, потому что, если узел с большим весом получает шум, он будет усилен и, вероятно, даст неверный прогноз. При этом обычно реализуется гауссов шум, который в основном представляет собой случайный шум с гауссовым распределением. Чтобы заставить модель сохранять малые веса, одним из способов является добавление штрафа к потере для каждого веса в зависимости от его размера. Это называется регуляризация веса и широко используется и существует в двух формах, L1 и L2. L1 просто добавляет

размер весов, умноженный на член L1, выбранный разработчиком. Другой, L2, должен сделать то же самое, но в этом случае возвести в квадрат это значение. Это приводит к тому, что значения больше 1 становятся еще больше. Таким образом, он не влияет на потери так же, как L1, когда не перетренирован. L2 также называется снижением веса и является более распространенным методом из двух.

Другой способ сохранить значения небольшими - нормализовать входные данные для каждого слоя и установить среднее значение в 0, а дисперсию - в 1. В режиме Tensor это можно сделать с помощью слоя BatchNormalization [23].

Важно отметить, что все эти методы, упомянутые до сих пор, активны только во время тренировок и ничего не делают при реальных предсказаниях.

Если существует много обучающих данных, метод ансамбля может быть хорошим способом. Этот метод делит обучающие данные на меньшие наборы и обучает модель для каждого набора данных.

Окончательный прогноз является усредненным прогнозом всех моделей. Этот метод работает, потому что, если модель сильно перетренирована в определенном направлении, есть вероятность, что другие сети заглушат этот результат многими другими моделями.

Окончательная техническая проверка приводит к преждевременной остановке и основанию для проверки модели после каждой эпохи и прекращения обучения.

График, показывающий, когда нужно сделать остановку, приведен на рисунке 1.8.

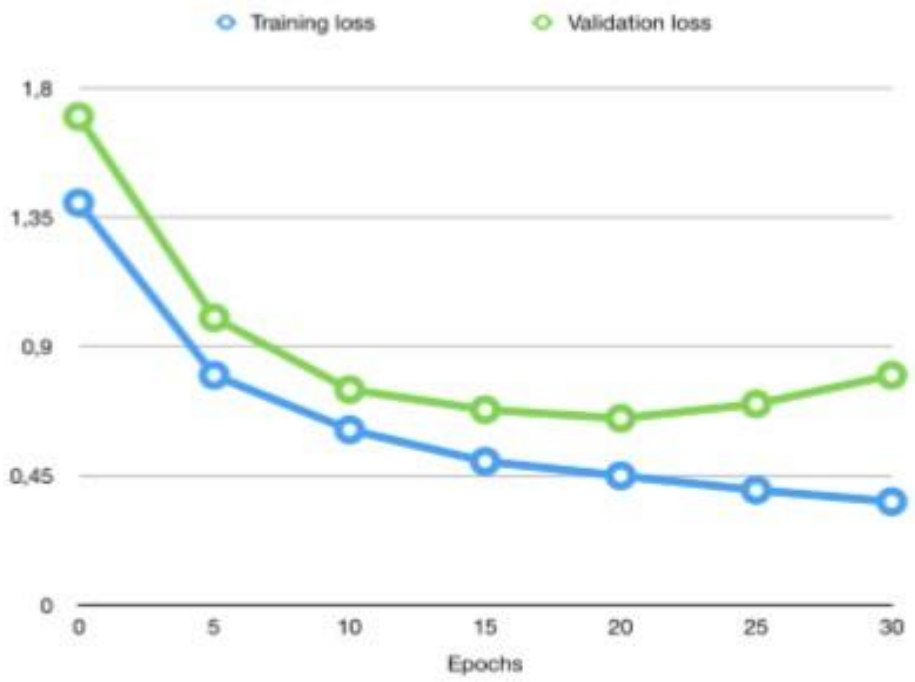

Рисунок 1.8 - График, показывающий, когда делать остановку. График показывает общую потерю за прошедшие периоды

Исходя из написанного, было решено использовать оптимизатор Adam, а также стало понятно, как правильно обучить модель и при этом не перетренировать её.

1.4 Описание работы сверточных нейронных сетей

Сверточные нейронные сети (CNN) использовались в течение достаточно долгого времени, когда речь идет о глубоком обучении, их возможности довольно поразительны, как доказано Крижевским и соавторами [9]. Они используют слои, которые выполняют свертки, отсюда и название. Входными данными в сверточную сеть являются двухмерные или трехмерные тензоры, где трехмерная альтернатива обычно имеет цветовые каналы вдоль третьего измерения. Использование сверточных нейронных сетей в распознавании изображений может быть практичным для нескольких причин, например, она может сказать о пространственных отношениях в изображении [24].

Рисунок 1.9 показывает пример того, как может выглядеть CNN. Изображение показывает поток сверточной нейронной сети слева направо. Несколько сверток, а также функции объединения выполняются по порядку на входном изображении. Последняя карта объектов затем добавляется в скрытые слои, чтобы классифицировать ее с помощью функции softmax.

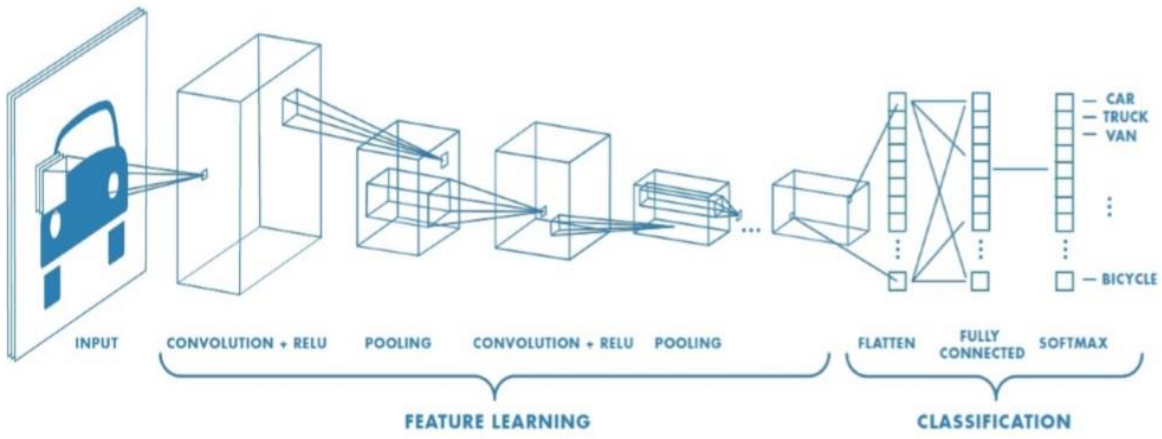

Рисунок 1.9 - Пример сверточной нейронной сети Изображение взято из [25]

Сверточный слой выполняет математическую операцию свертки на входном тензоре, который может быть представлен в виде изображения. Сверточный фильтр или ядро размером $k \cdot l$ скользит по входу. Ядро состоит из весов в каждом элементе. Эти веса находятся в процессе обучения сети. Поскольку ядро скользит по изображению, в качестве выходных данных вычисляется точечное произведение весов ядра и пикселей над ядром. После этого формируется новое изображение, содержащее все точечные произведения.

Заполнение ввода может использоваться для учета значений, когда ядро находится за пределами ввода. Использование отступов влияет на размер вывода после сверточного слоя. Поскольку изменения параметров в одной оси не приводят к появлению других вариантов, объясняющих, почему значения параметров одинаковы по обеим осям. Чем больше сверточных

слоев используется в сети, тем более сложные формы обнаруживаются. Первые слои могут обнаруживать края и углы, тогда как более поздние слои могут находить элементы, представляющие, например, автомобиль или собаку. Общим дополнением к уровням является функция активации, обычно ReLU. Это дает более быстрое обучение, позволяя только положительным весам проходить слой.

Пример оценки паддингом на рисунке 1.10. Зеленая сетка - это выход, синяя сетка - это вход, а теневая область - это текущая область, в которой находится ядро. Крайнее левое изображение показывает использование нулевого заполнения. Здесь размер ядра $k = 3$, входной размер $i = 4$, а выходной размер $o = 2$, т.е. меньше входного. В среднем изображении используется тот же отступ; размер вывода такой же, как размер ввода $o = i = 5$. В крайнем правом изображении используется полный отступ. Здесь получается больший выходной размер, чем входной размер, $o = 7$, $i = 5$. Отсутствие заполнения нулями подразумевает отсутствие заполнения вне ввода. Это означает, что ядро никогда не выходит за пределы фактического образа, когда сторона ядра попадает на сторону ввода, она переходит на следующую строку (если шаг равен 1). Такое же заполнение используется, когда требуется, чтобы выходной размер слоя совпадал с входным размером. Это достигается наличием отступа $p = [\frac{k}{2}]$ для любого размера ядра k.

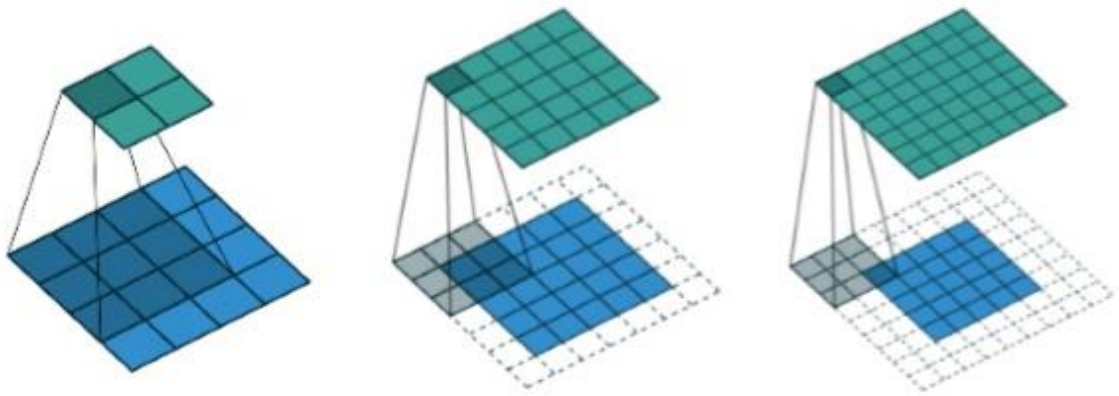

Рисунок 1.10 - Различные примеры заполнения. Изображения взяты из [24]

При полном заполнении увеличивается выходной размер по сравнению с входным, полностью используя каждую возможную комбинацию ядра и входного изображения. Это достигается за счет того, что отступ $p = k - 1$ для любого размера ядра k.

Еще одна полезная функция, используемая во многих сверточных нейронных сетях, - объединение в пулы. Слои пула несколько похожи на сверточные слои тем, что они используют скользящее окно, которое выполняет операцию над содержимым окна и выводит новое значение. Тем не менее, разница в том, что они используют другие функции вместо линейного сложения. Двумя общими функциями пула являются максимальный пул, где выходные данные являются наибольшим значением в окне, и средний пул, где выходные данные - это среднее значение компонентов в окне.

На рисунке 1.11 показан пример использования пула. Синяя сетка представляет входные данные, зеленая сетка представляет выходные данные, а заштрихованная область - то, где находится скользящее окно. Левое изображение показывает использование среднего пула, в то время как правое показывает использование максимального пула.

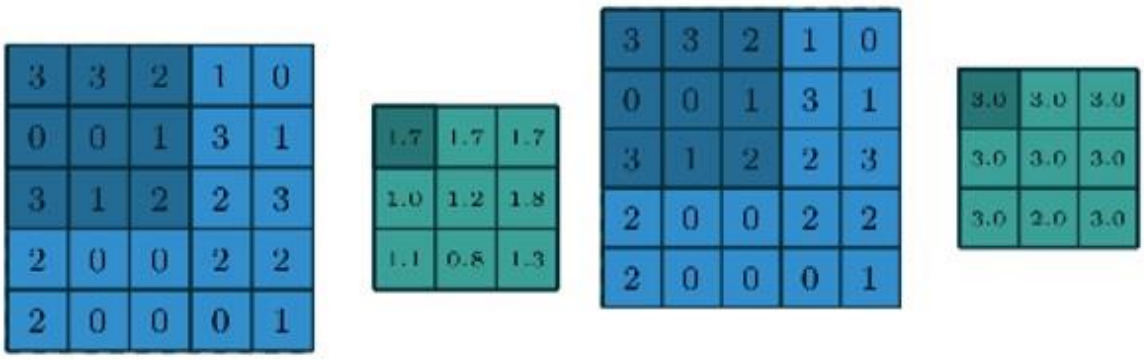

Рисунок 1.11 - Примеры двух типов пула. Изображения взяты из [24]

Когда в CNN должна быть проведена классификация, форму необходимо сместить в массив. Это делается после последнего пула или

сверточного слоя. Этот массив затем может быть передан в скрытый слой для классификации.

Исходя из изложенного выше, становится понятно, как работают по отдельности дополненная реальность и нейронные сети. В дальнейшем необходимо рассмотреть аппаратные возможности устройства, на котором предполагается использование приложения. А также создать компьютерную модель для локализации объектов на изображении.

Таким образом, были описаны основные понятия дополненной реальности, определены технологии, которые нужно использовать для решения поставленной задачи; показаны преимущества от применения технологий дополненной реальности для решения разного класса задач. Также описаны основные методы, средства и инструментарии, которые применяются при работе с дополненной реальностью. Проведенный анализ позволил сделать выбор на тех технологиях, которые максимально смогут быть применены для решения заявленной в исследовании задачи.

Глава 2 Анализ существующих методов и подходов для создания приложения дополненной реальности

D. Chatzopoulos и др. (2017) описывают основы MAR, или Mobile Augmented Reality, ее достижения и преимущества. По их оценкам, MAR является наиболее многообещающей областью мобильных приложений, и это окажет сильное влияние на то, как люди взаимодействуют с реальным миром. Тем не менее, они также проходят через некоторые из нынешних проблем с технологией, таких как ограничение пропускной способности и требуемая вычислительная мощность [11]. S.Gould и др. (2009) пишет об иерархической модели для совместного обнаружения объектов и сегментации изображения, где они в основном пытались сегментировать все объекты в изображении и классифицируют каждый пиксель [12]. Y LeCun и M.A. Ranzato подробно рассказывают об истории и прогрессе технологии, начиная с первой модели машинного обучения, Perceptron, и заканчивая задачами глубокого обучения. Они пишут о случаях использования для машинного обучения и о том, как в целом работают модели. Они также классифицируют различные виды моделей на отдельные категории. Также написано о том, как построены сверточные нейронные сети [13].

2.1 Обнаружение объектов с традиционным машинным обучением

Один из типичных способов обнаружить объекты в неподвижном состоянии - попытаться найти шаблоны или элементы изображения. Либо конкретное изображение может быть сопоставлено с большим изображением, либо может быть найден ряд признаков. Примером последнего являются особенности Хаара, которые используются в Альте-Джонсе для обнаружения лица, рисунок 2.1. Пиксели в белых областях суммируются и вычитаются из пикселей в черной области. Алгоритм позже

решит, была ли конкретная особенность найдена или нет, в зависимости от полученного значения [28].

Используя интеграл изображения (который является суммой значений пикселей в определенной области), можно получить различные особенности.

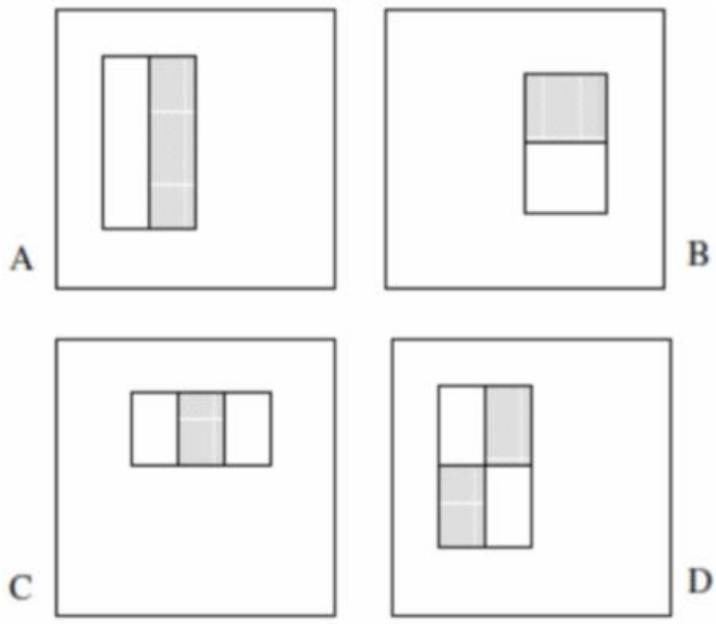

Рисунок 2.1 - Набор функций Хаара, используемых для обнаружения лиц в методе Виолы-Джонса

Этот метод основан на том факте, что каждое лицо имеет некоторые общие сходства. Даже объекты одного типа могут иметь некоторые общие черты.

Еще один способ найти объекты, представляющие непростую картину, - классифицировать каждый пиксель либо на заднем, либо на переднем плане, обычно называемый сегментация переднего плана / фона. Это обычно требует некоторых знаний о том, как обычно выглядит фон, например, это может быть - трава или бетонный пол.

Одной из основных функций для разделения фона и переднего плана является метод заливки. Любой, кто использовал Paint в Windows, точно знает, что это такое. Он заполняет сегмент похожих пикселей одинаковым цветом. Этот метод лучше всего работает на одном фоне только с одним

цветом. В сети есть много разных вариантов, но все они достигают одной и той же цели. [29]

Одним из возможных способов обнаружения объектов на изображении является использование данных глубины или RGB-D, которые представляют собой данные глубины, внедренные в фотографию RGB. Популярным устройством, которое использует данные о глубине для обнаружения объектов, является камера Kinect для Xbox.

На iPhone X, датчиком глубины может быть либо передняя камера, либо две камеры на задней панели. Камера True Depth работает благодаря тому, что точечный проектор излучает светлые точки (в основном на лицо, именно поэтому функция FaceID находится на передней камере) и снимает их с помощью инфракрасной камеры.

Двойная камера на задней панели работает, делая две фотографии и сравнивая их, чтобы найти сдвиги пикселей на тех же объектах. Расстояние рассчитывается по

$$\frac{\text{сдвиг пикселя}}{\text{фокусное расстояние} \cdot \text{базовая линия в метрах}} \quad (15)$$

и дает единицу в 1 / метр. Это то же самое, что человек видит расстояние, когда два глаза смотрят в одном направлении. [30]

Однако получение данных о глубине от двух камер в режиме реального времени невозможно, так как это требует слишком большой вычислительной мощности. К сожалению, это делает невозможным использование данной технологии в ARScene и, следовательно, невозможно в проекте.

Несмотря на все доступные методы, описанные выше, обнаружение объектов без машинного обучения все еще очень сложно. Эти методы работают лучше всего, когда изображения находятся в контролируемой среде, как правило, в промышленности, например, при поиске винтов на белом фоне. [31] Когда обстановка становится более непринужденной, например, при распознавании мебели в помещении, задача становится более сложной. По этой причине обнаружение объектов с использованием чистых

алгоритмов не очень распространено в бытовых приложениях. Вместо этого обнаружение объектов с помощью методов машинного обучения, таких как R-CNN (регионально-сверточная нейронная сеть), в настоящее время встречается гораздо чаще.

Следует что для использования дополненной реальности в разрабатываемом приложении можно использовать сдвоенную основную камеру, установленную на iphone X.

2.2 Прототип MATLAB для обнаружения объектов

Методы сегментации объектов, основанные на правильной характеристике, были реализованы в виде прототипа в MATLAB. Это было сделано для того, чтобы увидеть, как будет обучаться модель для обнаружения объектов. Идея заключалась в том, что если кто-то отправит изображение в систему, то в результате будет получен ограничивающий прямоугольник для каждого соответствующего объекта. Эти меньшие изображения позже будут отдельно классифицироваться с использованием глубоких нейронных сетей. Ограничительные рамки также будут полезны для пользовательского интерфейса в приложении.

Во-первых, метод обнаружения края был запущен на всем изображении, чтобы найти края, которые послужили бы хорошей переменной для сегментации. Затем пороговый алгоритм Брэдли [32] будет преобразовывать изображение в двоичную форму. Метод Брэдли является локально адаптивным и вычисляет пороговое значение для каждого пикселя, просматривая среднюю интенсивность его окрестности. Из этого можно найти вложенные большие двоичные объекты, и, найдя определенное количество пикселей устранить шумовые ошибки, и найти объекты. Затем ограничивающие рамки определяются путем выбора минимальных и максимальных значений высоты и веса больших двоичных объектов.

Сценарий MATLAB был опробован, чтобы найти возможные способы сегментирования областей, содержащих объекты на изображении. Затем выходные сегменты будут классифицироваться по отдельности. Таким образом, будет известно, как местоположение, так и классификация каждого объекта. На рисунке 2.2 показан пример того, как этот метод работает. Верхнее левое изображение показывает исходное изображение. На верхнем правом изображении показан результат обнаружения бегущей кромки исходного изображения. Нижнее левое изображение показывает результат после бинаризации верхнего правого изображения. Нижнее правое изображение показывает сгенерированную ограничивающую рамку, наложенную на исходное изображение. Однако результаты были непредсказуемыми, а параметры были слишком зависимы от освещения, теней, фона и других особенностей. На рисунке 2.3 показан пример применения того же метода к другому входному изображению. Из-за этих ненадежных результатов этот метод был отменен.

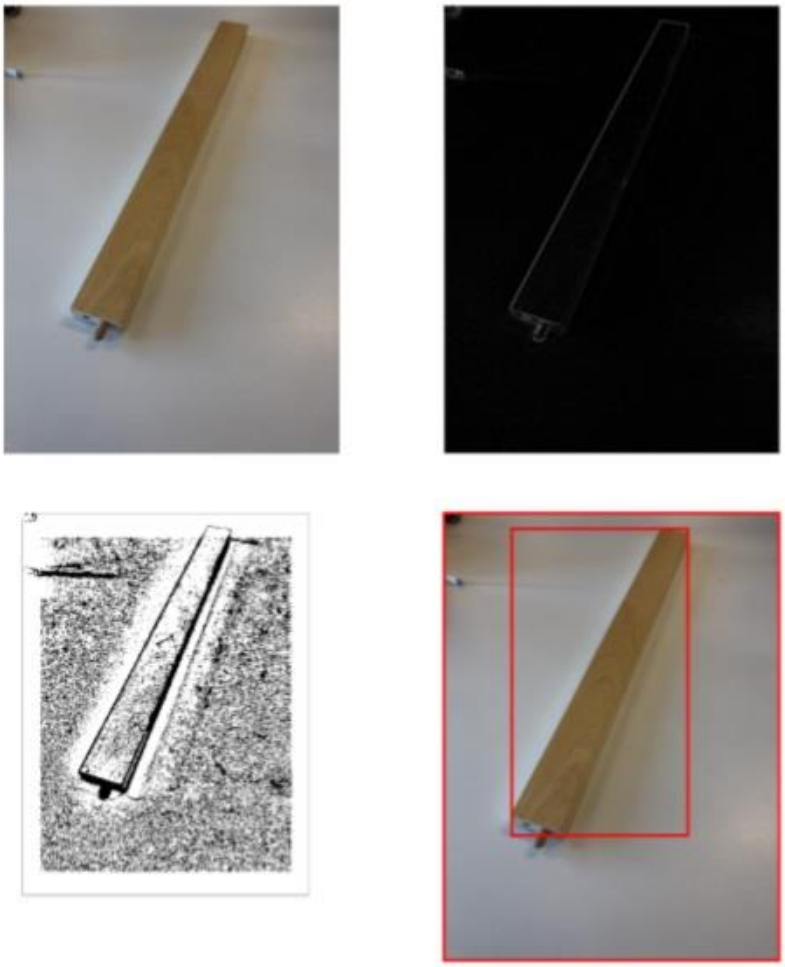

Рисунок 2.2 - Изображения, показывающие результаты обнаружения объектов в MATLAB

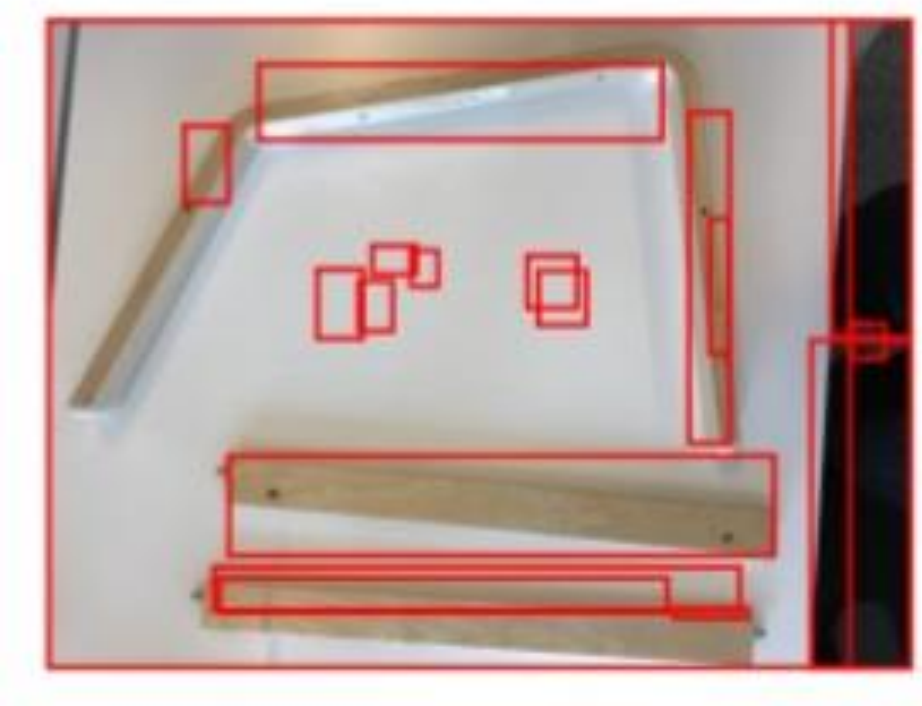

Рисунок 2.3 - Пример изображения, когда этот метод не работал

К сожалению, создание компьютерной модели с использованием MATLAB не увенчался успехом поэтому было решено создавать приложение, адаптировать и проводить испытания сразу на устройство, которое предполагалось использовать.

Из главы можно сделать выводы то, что приложение лучше разрабатывать сразу адаптированное под итоговое устройство с использованием уже существующих методов, разработанных для этого устройства.

Глава 3 Объединение методов в приложении дополненной реальности

3.1 Макет запланированного приложения

В начале был создан макет приложения, иллюстрирующий информацию и порядок использования. Макет был создан с использованием бесплатного онлайн-инструмента под названием MockFlow [14]. Когда приложение открывается, предоставляется возможность выбора мебели, см. Рисунок 3.1, на котором пользователь решает, какую мебель он хочет собрать. Пользователь может выбрать мебель, прокручивая список, используя панель поиска или используя встроенный сканер штрих-кода. Для первого использование важно понять назначение приложения, когда пользователь впервые его открывает. Для этого, при первом использовании, всплывает уведомление, объясняющее цель приложения. После выбора мебели пользователь попадает в подробный вид, рисунок 3.2. Это предназначено для того, чтобы пользователь мог видеть увеличенные изображения мебели и ее уникальный артикульный номер. Это полезно, поскольку некоторые модели мебели выглядят очень похоже. Здесь пользователь также может прочитать другую конкретную информацию о предмете мебели, а также просмотреть ее 3D-модель, которая показывает, как она будет выглядеть после сборки. Когда пользователь готов, он должен нажать кнопку «Собрать», чтобы продолжить. В представлении «Сборка» пользователю предоставляется вид с камеры прямой трансляции, а также поле с информацией, содержащее инструкции, рисунок. 3.3. Первым шагом для пользователя является сканирование фрагментов на полу, что позволяет модели машинного обучения находить элементы, которые предполагается собрать. Как только первые части автоматически найдены, они выделяются с помощью ограничивающих рамок, чтобы предупредить пользователя об их текущей важности. Затем пользователь может нажать кнопку «Далее», чтобы продолжить. После этого показывается анимация с изображением

трехмерных деталей, с правильной сборкой. Когда пользователь собрал части, он нажимает «Далее». Затем этот процесс будет сопровождаться следующим набором инструкций и будет продолжаться до тех пор, пока предмет мебели не будет полностью собран.

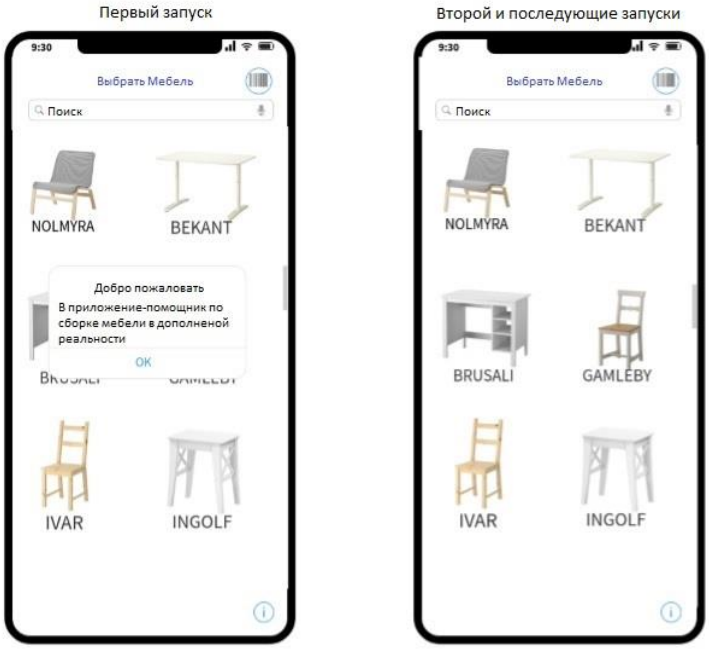

Рисунок 3.1 - Экран выбора мебели при запуске приложения. На левом изображении показано, как оно выглядит для первого запуска, а справа - как оно выглядит для последующих запусков

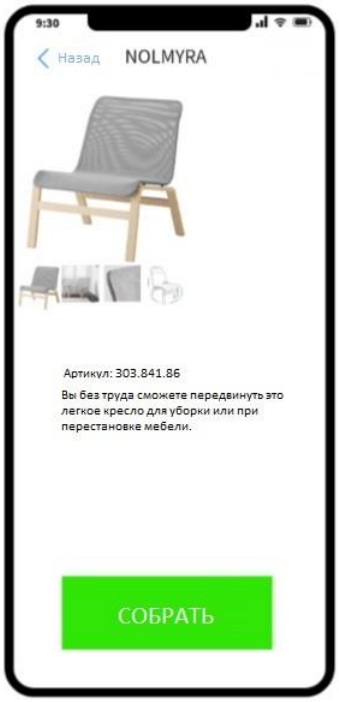

Рисунок 3.2 - Подробный вид

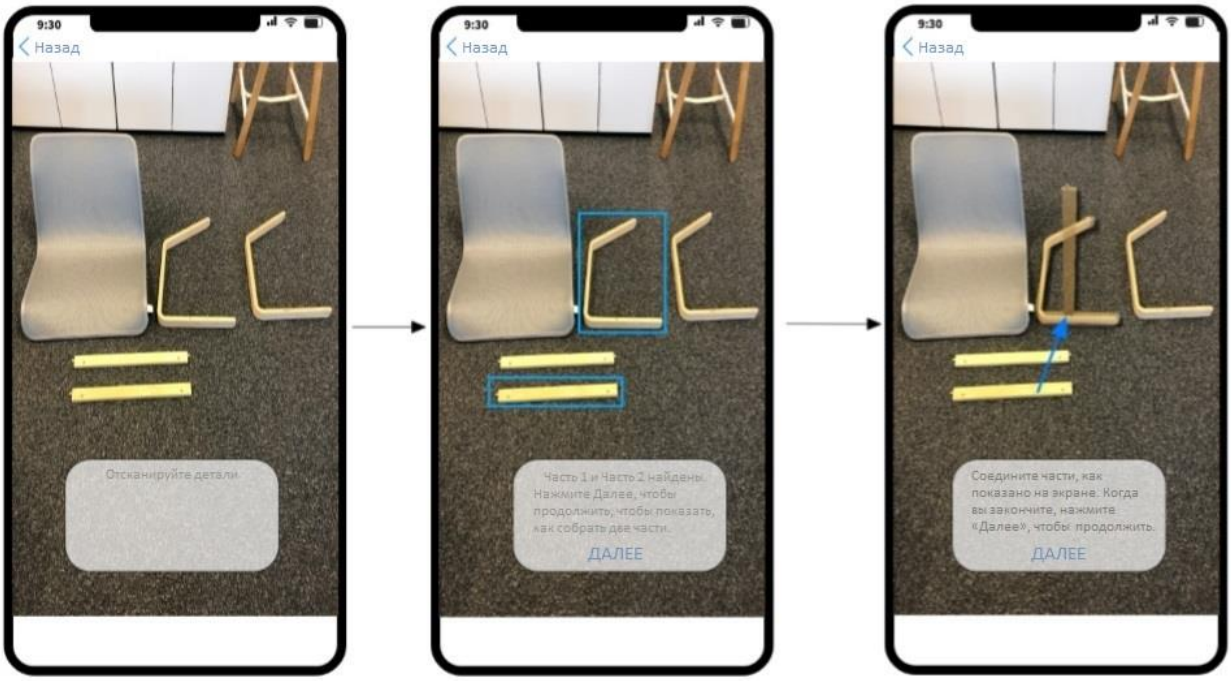

Рисунок 3.3 - Слева направо показано, как пользователь соединяет две части

Далее, на основе макета, созданного в этом пункте, будет создан графический интерфейс приложения.

3.2 Работа с данными для сверточной нейронной сети

При работе с машинным обучением для получения хорошей модели часто требуются большие наборы данных. Проблема с этим заключается в том, что может быть сложно получить такой большой набор данных, поскольку обычно требуется также, чтобы метки с этими данными вводились вручную. Эти метки будут использоваться нейронной сетью, чтобы во время обучения проверить правильность предсказанных значений.

В этой работе, данные представляют собой наборы изображений различных частей мебели. Этих изображений не было нигде в Интернете, поэтому их нужно было собирать, делая множество фотографий. Фотографии были сделаны на разных фонах и с разных углов. Таблица 3.1 показывает,

сколько было изображений каждого класса. Фоны были в основном типичными для окружающей среды. Все фотографии были затем изменены до 256x256 пикселей. Причина выбора размера квадрата заключается в том, что при их повороте на изображениях не будет ни черных полос по бокам, ни растяжения.

Таблица 3.1 - Таблица, показывающая, сколько было изображений каждого класса

Предмет	Количество изображений
Ножка	203
Часть каркаса	210
Сиденье	216

Для первой мебели Nolmyra было только 3 уникальных детали (исключая винты и аналогичные детали). Поэтому модель была обучена распознавать 4 вещи; разные части и неизвестный объект. Неизвестная метка была вызвана тем, что алгоритм обнаружения объектов мог обнаруживать другие объекты, и тогда нежелательно, чтобы эти объекты принимались за детали мебели. Конечно, это также можно сделать, установив порог для значения доверия, то есть, если модель дает значение доверия ниже 0,8, она отбрасывает его как неизвестный объект. Проблема с этим типом состоит в том, что намного труднее отличить части от неизвестных объектов. Это связано с тем, что лучшая модель - это та, которая может дать почти 100% уверенность при каждом прогнозе.

По этой причине фотографии случайных объектов также были добавлены в набор данных. Большинство из этих фотографий были получены из Интернета. Изображения были помещены в папки с определенным элементом, что означало, что их маркировка стала намного проще.

Обучение модели обычно требует сотен тысяч или даже миллионов фотографий. Трудно получить такое количество данных, и на его получение уйдет много времени. Пример сделанных фотографий показаны на рисунке

3.4, изображения, показывающие различные образцы обучающих данных, использованных для обучения модели. Слева направо: сиденье, часть каркаса, ножка.

Рисунок 3.4 - Изображения, показывающие различные образцы обучающих данных

Вместо обучения только исходными изображениями, некоторые изображения могут быть созданы из предыдущих, например, путем поворота, смещения, изменения яркости, насыщенности, контрастности и т. д. Это будет по существу изображение одного и того же объекта, но данные будут выглядеть по-разному, что дает модели более релевантные данные, рисунок 3.5.

Рисунок 3.5 - Изображения, показывающие некоторые результирующие изображения из дополнения

Верхнее левое изображение показывает исходное изображение. Верхнее среднее изображение показывает результат после поворота на 180 градусов. На верхнем правом изображении показан результат после поворота на 90 градусов. Нижнее левое изображение показывает результат после уменьшения контрастности на 30%. Нижнее среднее изображение показывает результат после уменьшения цветового баланса на 70%. На правом нижнем изображении показан результат после увеличения цветового баланса на 60%. При этом важно помнить, что дополненные данные должны соответствовать реальным ситуациям.

Кроме того, изображения объектов, представляющих интерес, могут быть вырезаны и вставлены в случайные среды для создания еще большего количества данных, Рисунок. 3.6. Крайнее левое изображение показывает вырез одной из частей мебели. Два других изображения показывают результат после того, как они были вставлены в различные среды для искусственного увеличения тренировочных данных.

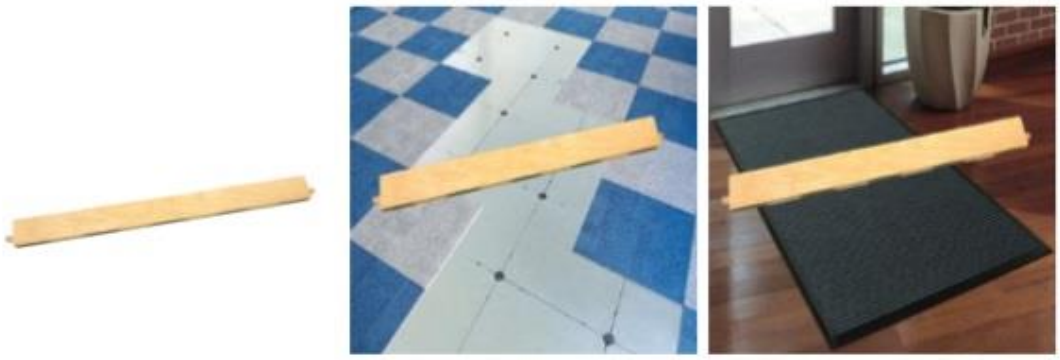

Рисунок 3.6 - Изображения, показывающие, как использовались вырезы изображений

Идея состоит в том, чтобы попытаться заставить модель понять, что акцент должен быть сделан на мебельной части, а фоновая среда не имеет значения. При этом несмотря на то, что детали можно было поместить на совершенно случайные среды, для лучшего обучения они размещались на

вставках в соответствующие пространства, такие как полы или ковры. В данной работе это дало хороший результат, так как повысило точность теста на 8%.

Из данного пункта следует что для правильного обучения нейронной сети необходимо большое разнообразие данных. Что и было проделано.

3.3 Рассмотрение вопроса классификации изображений

Проектирование нейронной сети для классификации изображений - не самая простая задача, и большая часть работы состоит в том, чтобы пробовать что-то и делать догадки. Как указано в разделе о сборе данных, было 4 разных класса для классификации, что означает, что точность теста более 25% была бы значительной. Цель - добиться точности около 90% с относительно небольшим количеством данных.

В начале было много сверточных слоев, наслоенных объединяющими слоями и несколькими плотными слоями в конце. В конечном слое было четыре узла и функции активации Softmax, чтобы обеспечить вероятность классификации между четырьмя классами. Функции активации других слоев были установлены на ReLu, так как они самые быстрые и предпочтительнее для больших сетей.

SGD с импульсом в качестве оптимизатора был протестирован, но был заменен на Adam, так как он имел лучшую производительность. Были рассмотрены только линейные оптимизаторы, поскольку нелинейные занимают много вычислительной мощности и обычно рекомендуются только для небольших сетей.

Были опробованы разные размеры ядра, количества сверточных слоев, порядок объединяющих слоев в разных местах, размеры шагов, количество плотных слоев и их узлов.

Обучение моделей с самого начала было серьезным недостатком. Проверенные методы борьбы с этим состояли из отсева с различными

значениями, регуляризации веса L2, нормализации партии и уменьшения количества весов.

В зависимости от весовых коэффициентов, оценка обычно сводилась к 25% при каждой попытке. При использовании метода отсева значение 50% при минимуме 64 узлов в слое уменьшало перенапряжение наиболее эффективно. Это в сочетании с регуляризацией веса и нормализацией партии дало хорошие результаты. На рисунке 3.7 показано, как избежать перегрузки. Графики показывают потерю тренировки (зеленый) и потерю оценки (красный). Верхнее левое изображение показывает тренировку без какого-либо отсева или регулирования веса. На верхнем правом изображении показана тренировка с выполнением отсева. На нижнем изображении отображается тренировка с выполнением отсева и регулирования веса.

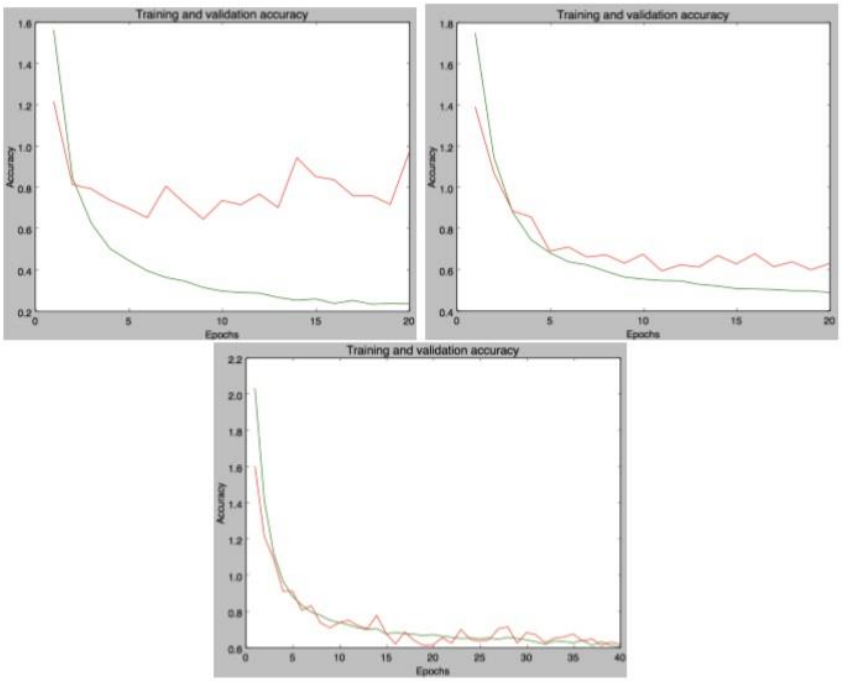

Рисунок 3.7 - Изображения показывают результат с учетом регуляризации веса и отсева моделей

Gaussian Noise дал очень хорошие результаты, но, к сожалению, его было невозможно преобразовать из формата модели, сгенерированного в keras, в формат mlmodel, необходимый для Xcode.

Самая важная вещь, которую стоит упомянуть, - это различие в количестве фотографий, которые были сделаны. Добавление небольшого количества фотографий внесло огромный вклад в оценку результатов теста. После добавления еще 50 изображений, оценка теста увеличилась с 80,55% до 83,45% погрешности.

При окончательном тестировании модели была попытка разделить данные обучения и данные испытаний. Правда некоторые дополненные данные попали в тестовый набор. Модель уже тренировалась на исходном изображении, и она была достаточно похожа, чтобы дать отличные результаты в 99,52%. Это было окончательно исправлено путем разделения теста и передачи данных в отдельные папки. На рисунке 3.8 показан график одного из этих учебных занятий.

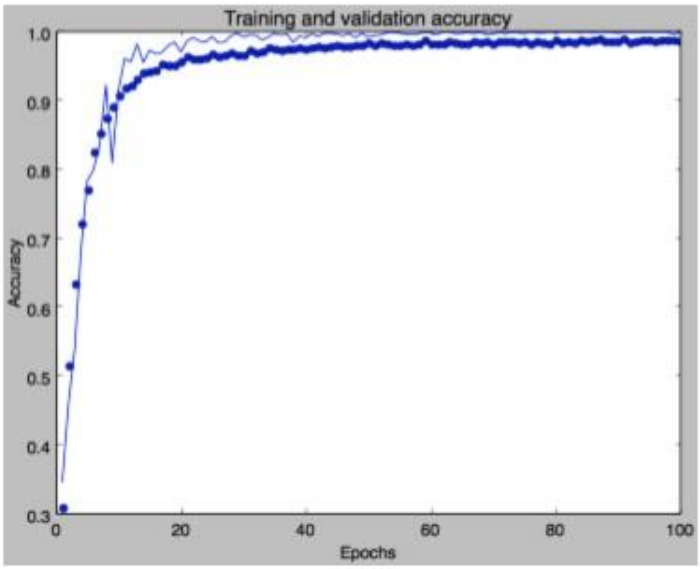

Рисунок 3.8 - Проверка и обучение точности в течение 100 эпох. Производительность кажется намного лучше, чем есть на самом деле

Как показано в коде в приложении Г, ранняя остановка также была реализована с обратным вызовом для сохранения наиболее эффективной модели в конце процесса.

Сеть, которая зарекомендовала себя как самая эффективная (точность 92,25%), имела конфигурацию, показанную в приложении Д.

Модель со многими сверточными слоями и несколькими плотными слоями в конце, дала наилучшие результаты. Можно было бы внедрить и более крупную сеть, а также протестировать ее, но это бы привело к увеличению размера модели и времени обучения.

После пробного проектирования сети с нуля было решено попробовать использовать предварительно обученные сети, а затем переподготовить их для другой цели. Были выбраны модели, обученные на imageNet [26]. Модели с соответствующими весами были загружены без полностью связанного слоя. Веса верхнего слоя затем замораживались на разных этапах, а сверху добавлялись новые полностью связанные слои, и обучались новые классификаторы. Было опробовано несколько различных моделей, в том числе ResNet50, InceptionV3 и VGG16, которые изначально были интегрированы в Keras.

Код, указанный в приложении Е. показывает загрузку сети VGG16 с весами, предварительно настроенными на imageNet. Затем полностью связанный слой удаляется и вместо него добавляется пользовательский верхний слой для обучения.

Наилучший результат, который был достигнут для каждой из моделей, использующих этот метод, приведен в таблице 3.2.

Таблица 3.2 - Результаты выполнения трансферного обучения

Используется предварительно обученная модель	Лучшая достигнутая точность
VGG16	70.8%.
ResNet	25.0%
InceptionV3	72.9%

Это показывает, что использование модели InceptionV3 с предварительно подготовленными весами дало лучший результат. Однако несколько факторов, таких как, где модели были заморожены, как добавление и удаление слоев, скорее всего, повлияли на результаты. Как

видно из использования ResNet, лучшая модель достигла 25% точности, что равносильно предположению, поскольку существует только четыре возможных класса. Эти модели могли бы быть улучшены, если бы не было решено использовать новый подход к проблеме.

3.4 Одновременное обнаружение нескольких объектов на изображении

На этом этапе работы существовала модель машинного обучения, способная неплохо классифицировать, какие части мебели были на изображении. Однако при использовании приложения есть вероятность, что в одном кадре присутствуют несколько частей. Следовательно, метод локализации нескольких объектов очень востребован. Идея заключалась в том, чтобы найти такой метод, сегментирующий области, содержащие объект, а затем использовать эти области в качестве входных изображений для модели машинного обучения.

Использование ARKit. В ARKit от Apple есть функция, позволяющая обнаруживать отсканированные 3D-объекты. Для сканирования они разработали приложение, которое можно загрузить с их сайта. [27] Рисунок 3.9 Слева ограничивающий прямоугольник определен так, что в него не включаются опорные точки из других объектов. Посередине объект сканируется, направляя камеру вокруг объекта под всеми углами. Справа созданная модель тестируется. В этом случае объект обнаруживается через 0,4 секунды. После завершения сканирования приложение позволяет экспортировать модель, чтобы затем включить ее в проект ARKit. Попав в проект, она просто импортируется в конфигурацию ARWorldTrackingCon, как показано в приложении Ж.

После импорта проект удалось проверить на эффективность распознавания и отслеживания трех предметов мебели одновременно. К сожалению, этот метод оказался неподходящим для поставленной цели. При

тестировании в реальном времени в приложении время обнаружения было более 1 секунды, что делало отслеживание сложным. Отслеживание было не плавным, а скорее прерывистым. Много раз, объект не был обнаружен вообще. Это происходило главным образом из-за того, что объекты были большими, чтобы уместить в экран, находясь близко к камере, а также из-за того, что в пределах ограничивающего прямоугольника у объекта было много пустого пространства.

Обнаружение работало лучше всего, находясь в той же среде, статично при одинаковых условиях освещения, но не срабатывало, когда объект двигался. Поэтому, поскольку пользователь собирается держать части вручную и перемещать их, этот метод не может быть использован.

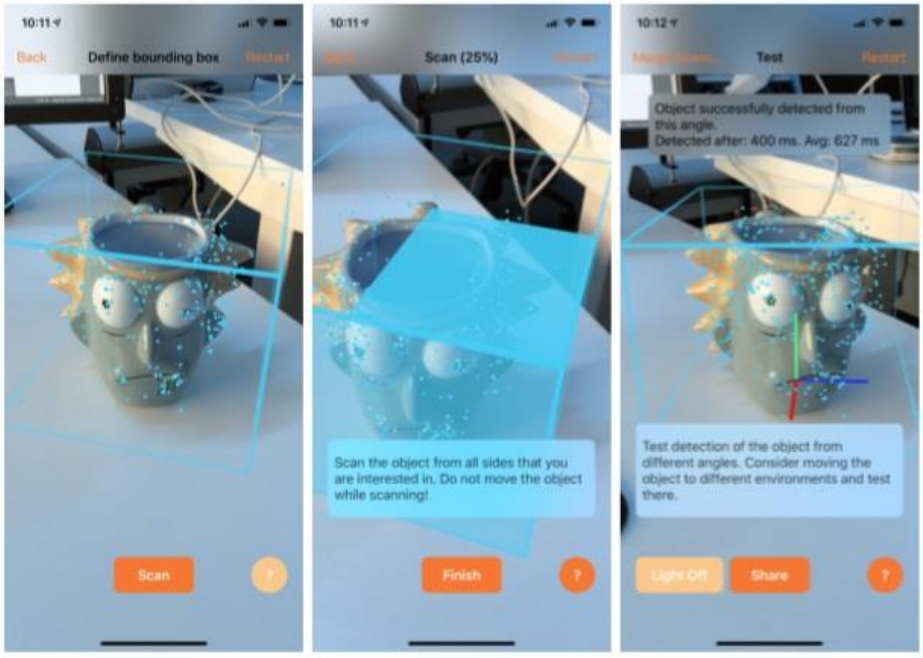

Рисунок 3.9 - 3D-сканирование с использованием приложения Apple

YOLO, или You Only Look Once, - это одностадийный детекторный подход к обнаружению и распознаванию объектов [10]. Он берет изображение и предсказывает как ограничивающие прямоугольники, так и вероятности того, что классы находятся в этих прямоугольниках за один проход, отсюда и берется его название. Он был разработан, чтобы быть

быстрым и удобным в сценариях реального времени. Поскольку YOLO видит все изображение во время обучения и тестирования, он получает контекстную информацию о классах и уменьшает количество ошибок при сопоставлении фоновых патчей для объектов.

Архитектура для YOLO состоит в основном в виде сверточной нейронной сети с 24 сверточными слоями и двумя полностью связанными слоями. Была также обученная нейросеть меньшего размера под названием Fast YOLO training, состоящая всего из 9 слоев, которая была разработана для создания еще более быстрой системы обнаружения объектов.

Система сначала делит входное изображение на сетку $S \times S$, где каждая ячейка предсказывает количество B ограничивающих прямоугольников и, соответственно, оценки достоверности для этих блоков. Каждое предсказание ограничительной рамки состоит из 5 отдельных предсказаний: координаты x, y, представленные в виде их положения относительно ячейки сетки, ширины и высоты относительно всего изображения и значения достоверности. Каждая ячейка сетки также предсказывает вероятности C для каждого класса, при условии, что внутри блоков находится объект.

Пересечение по объединению, также известное как индекс Жакара, является способом измерения сходства между множествами. Это можно записать как

$$IOU(A,B) = \frac{|A \cap B|}{|A \cup B|} - \frac{|A \cap B|}{|A|+|B|-|A \cap B|}, 0 \leq IOU(A,B) \leq 1 \quad (16)$$

если A, B два конечных множества. Если два набора равны, то

$$IOU(A,A) = \frac{|A \cap A|}{|A \cup A|} - \frac{|A \cap A|}{|A|+|A|-|A \cap A|} = \frac{|A|}{|A|} = 1 \quad (17)$$

При тестировании эти оценки затем объединяются, чтобы придать классу определенные значения доверия для каждого из блоков, таким образом, он получает как уверенность от класса, находящегося в блоке, так и от того, насколько хорошо блок соответствует объекту. Рисунок 3.10 суммирует поток в YOLO. Сначала погружается в сетку $S \times S$. Отдельно предсказывает ограничивающие рамки с соответствующей уверенностью, а

также вероятностью класса. Затем объединяет, чтобы сформировать окончательные прогнозы.

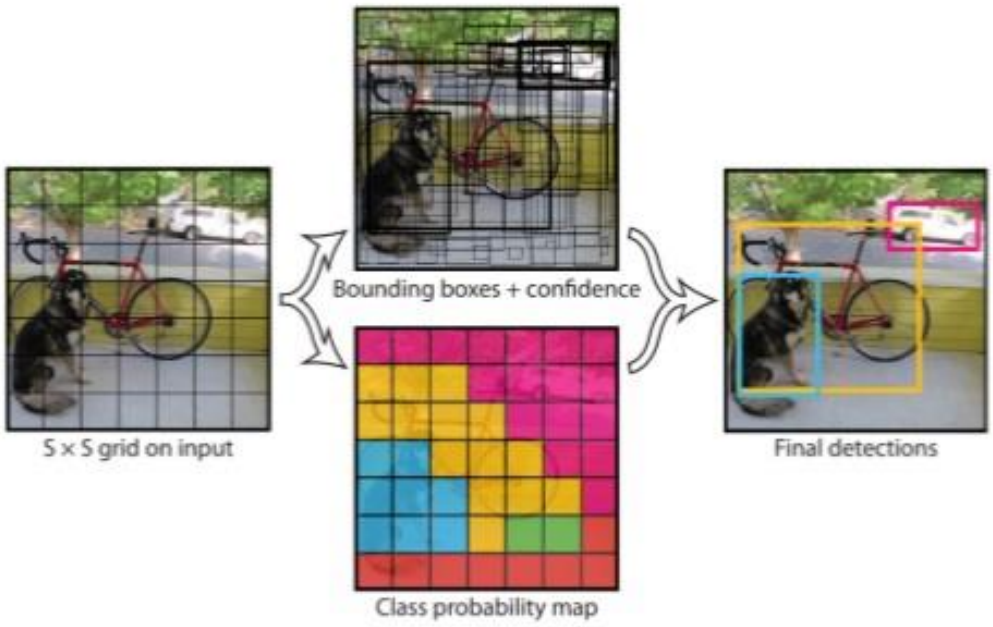

Рисунок 3.10 - Этапы YOLO. Изображение взято из [10]

Сравнения показали, что по сравнению с другими системами реального времени Fast YOLO и YOLO превзошли их всех, причем Fast YOLO - самый быстрый, но YOLO - более точный по наборам данных PASCAL VOC [35].

На протяжении многих лет Redmon et al. работает над обновлением шаблона проектирования сети YOLO. Сначала в 2016 году, когда они представили YOLOv2, а затем в апреле 2018 года с YOLOv3 [33] [34]. YOLOv2 добавил в систему несколько концепций, чтобы сделать ее еще более быстрой и точной, чем предыдущая. Он удалил полностью связанные слои, и теперь стало возможным тренироваться на нескольких различных разрешениях входного изображения, а также предсказывать гораздо больше ограничивающих рамок, чем его предшественник. В YOLOv3 добавлены некоторые изменения, которые улучшили его способность обнаруживать небольшие объекты, но при поиске объектов более крупного размера производительность была хуже.

Когда дело доходит до оценки модели, которая обнаруживает объект, существует большая разница, по сравнению с классической классификацией

изображений. Когда речь идет о последнем, точность показывает, как часто модель делает правильный прогноз. Однако при обнаружении объекта это делается по-другому. Общим показателем является средняя точность (mAP). Эта метрика использует IoU (Intersection over Union). IOU и классификация определяют, считается ли прогноз правильным. Классификация, очевидно, должна быть такой же, как основополагающая истина, и IoU должно превышать установленный порог, это известно, как истинные положительные результаты. Предсказанное поле, которое не проходит порог, называется ложным срабатыванием. Если существует более одного прогноза, который превышает пороговое значение, только один будет воспринят как истинно положительный, а другой - как ложно положительный. Когда предсказание не было сделано для основания истинности, оно называется ложным отрицанием [38].

Из ложных положительных, истинных положительных и отрицательных, можно рассчитать один отзыв, а также точные оценки [36]. Точность заключается в том, насколько хороша модель для идентификации только соответствующих данных, и может быть записана как

$$\text{Точность} = \frac{\text{Истинные позитивы}}{\text{Истинные позитивы} + \text{Ложные позитивы}} \quad (18)$$

Насколько хороша модель, чтобы найти все соответствующие данные, и она записывается как

$$\text{Напоминание} = \frac{\text{Истинные позитивы}}{\text{Истинные позитивы} + \text{Ложные негативы}} (19)$$

Когда классификация сделана, вы также получаете значение достоверности. Это называют предсказанием, если достоверность предсказаний проходит установленный порог. Однако, если рассчитать возврат и точность для всех возможных порогов, можно получить кривую Recall x Precision. Из этой кривой можно получить mAP, взяв область под кривой.

Существуют различные стандарты при расчете mAP. В PASCAL VOC Challenge используется средняя точность со статическим порогом 50% [35].

Это обычно называется mean_average_precision_50. Другим распространенным стандартом является стандарт, используемый COCO [37]. Этот метод вычисляет mAP при пороге IoU от 50% до 95%, в пределах 5% каждой итерации, и в среднем превышает среднее значение. Это известно, как mean_average_precision. В этой работе используется последний названный метод.

3.5 Использование машинного обучения Turi Create

У Apple есть лицензионный инструментарий для разработки пользовательских моделей машинного обучения под названием Turi Create [39]. Он был создан, чтобы помочь разработчикам легко реализовать свои собственные идеи в приложении. Включая методы обнаружения объектов.

В этом проекте протестирован метод обнаружения объектов, включенный в Turi Create. Для начала, необходимо создать правдивые данные для каждого изображения, которое было обучено и проверено. Это было сделано с использованием Simple Image Annotator [40], где нужно нарисовать ограничивающие рамки для объектов на изображениях и пометить их. Данные выводятся в формате .csv. Turi Create использует другой формат для аннотаций, называемых. sframe, однако они предоставляют простой скрипт Python, который автоматически выполняет преобразование.

Затем Turi обучает модель, используя повторную реализацию сети TinyYOLO. Он также использует обучение передачи, начиная с классификатора изображений, который был обучен в наборе данных imageNet, а затем выполняет сквозную настройку, чтобы изменить сеть на одноступенчатый детектор.

Несколько моделей были обучены с различным количеством обучающих данных, чтобы оценить, сколько их необходимо, чтобы получить достойный результат. На веб-сайте Turi утверждается, что для создания

достаточно хорошей модели требуется не менее 30 образцов каждого класса. Наименьшее количество обучающих изображений в проекте было около 50, а наивысшее было около 1000. Средняя точность была измерена для каждой модели с использованием 200 тестовых изображений, а затем нанесена на график, который показывает, как он изменился, в зависимости от количества использованных обучающих данных.

Код в приложении И показывает, как модель была создана с использованием Turi в python. Результаты также могут быть дополнительно проверены путем рисования новых предсказанных ограничивающих рамок поверх исходного изображения, из которых можно визуализировать, как на самом деле выполнялась модель. Это было сделано с использованием кода в приложении К. Из-за огромного количества возможных вариантов использования было решено уменьшить цель. Модель обучалась только на нескольких фонах.

Средняя точность от различных моделей, которые были обучены, наблюдалась и находится в таблице 3.3. Эти значения были нанесены на график, который показан на рисунке 3.11.

Таблица 3.3 - Средняя точность в зависимости от количества используемых данных

Количество тренировочных образов	**Процент от общей суммы**	**Средняя точность**
≈50	5%	0.17064
≈100	10%	0.30188
≈140	15%	0.30342
≈185	20%	0.39269
≈280	30%	0.41588
≈370	40%	0.40595
≈415	45%	0.49917
≈450	50%	0.47397
≈570	60%	0.50454
≈700	75%	0.54433

Количество тренировочных образов	Процент от общей суммы	Средняя точность
926	100%	0.57618

При обучении разным объемам данных может показаться, что больше данных иногда дает худший результат. Это можно объяснить двумя способами. Первое объяснение состоит в том, что при обучении моделей данные случайным образом отделялись от полного набора данных обучения. Это означает, что время от времени могут быть «хорошие» данные. В некоторых изображениях больше объектов, а в других меньше. Таким образом, если есть набор данных с изображениями с большим количеством объектов, это, вероятно, даст лучшие результаты. Однако важно отметить, что для хорошей работы модели также необходимы области изображения без объекта.

Второе объяснение состоит в том, что при обучении не всегда возвращается самая оптимальная модель для этого набора данных. Нужно принять во внимание раннюю остановку. Иногда ранняя остановка может произойти в хорошем месте, а иногда продолжение тренировки могло бы дать лучший результат.

На графике испытаний, рисунок 3.11, не видно признаков насыщения, поэтому большее количество данных улучшит производительность. Причиной отказа от тестирования с большим количеством данных является сложность сбора большего количества данных и время, необходимое для обучения. На данный момент время обучения с 926 изображениями на ноутбуке Lenovo b590 с процессором Intel Core i3 с частотой 2,4 ГГц составляет более 24 часов.

На рисунке 3.11 видно резкое возрастание до тех пор, пока около 20% тренировочного набора или 185 изображений не использованы для обучения модели. Это соответствует примерно 30 изображениям в классе. После этого график начинает выравниваться. Следовательно, утверждение о том, что для обучения достаточно приличной модели требуется не менее 30 изображений,

является правильным. Однако, как видно, большее количество данных обычно дает еще лучший результат.

На рисунке 3.12 показаны три классификации изображений. Верхний ряд классифицируется с использованием модели, которая была обучена на всем тренировочном наборе. В нижнем ряду показаны изображения, которые были обучены только на 15% тренировочного набора.

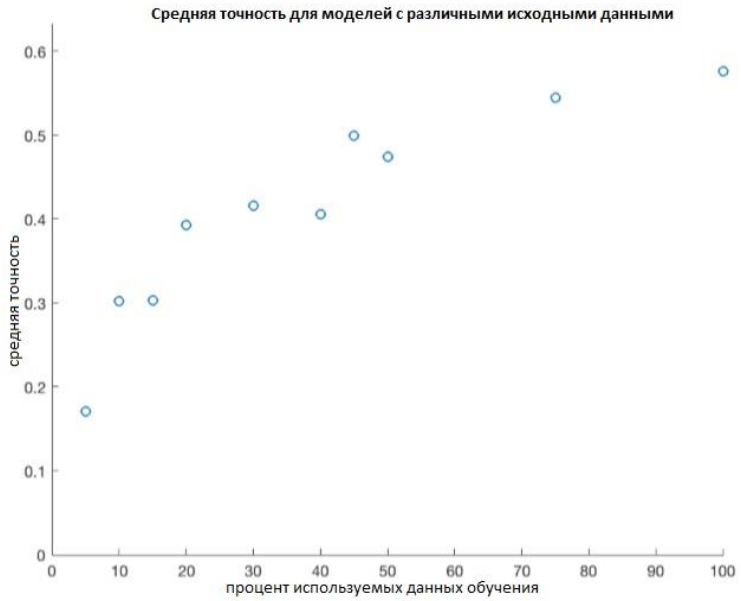

Рисунок 3.11 - Средняя точность из таблицы 3.1, построенная в зависимости от количества используемых данных обучения. 100% соответствует 926 изображениям

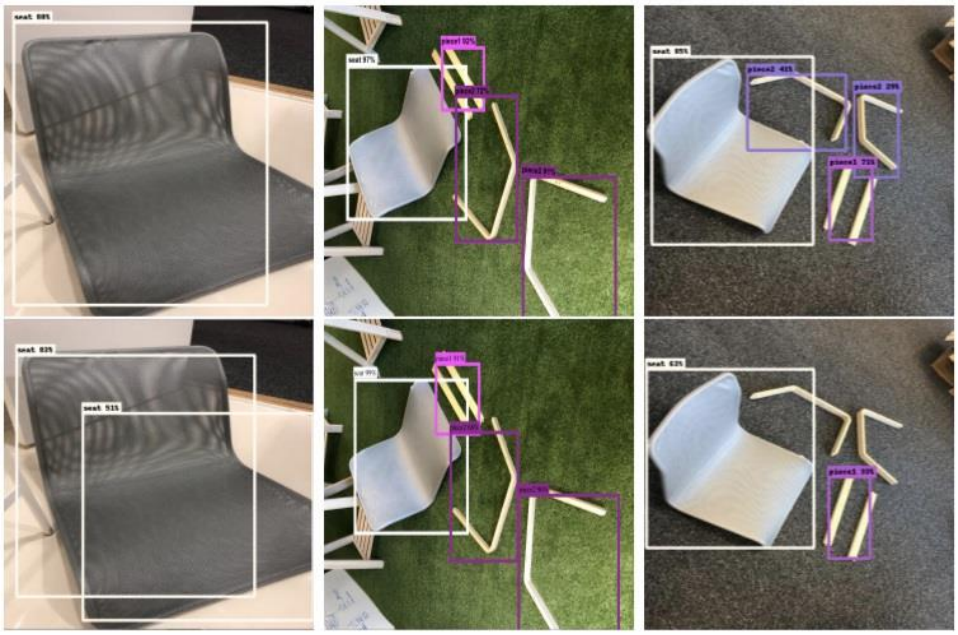

Рисунок 3.12 - Изображения, показывающие, как эти три изображения были классифицированы по-разному с использованием различных моделей

После обучения и оценки соответствия модели стандартному уровню, включенному в этот проект, она была преобразована в формат .mlmodel, используемый в Swift. При вводе кадра из приложения в модель, модель выводит мультимассив, содержащий массив для каждого найденного объекта. Эти предсказания были максимально подавлены, так как эта функциональность теряется при преобразовании из модели, используемой в python, в формат .mlmodel. Таким образом, формат должен быть переопределен в самом приложении. Был использован не-максимальный порог подавления 0,5, так как он считается традиционным стандартом [41]. Код в приложении Л показывает, как было реализовано не-максимальное подавление.

3.6 Отслеживание найденных объектов при их перемещении на изображении

Отслеживание объектов, подразумевает, что приложение будет отслеживать, где определенные объекты находятся на экране. Идея

заключалась в том, что приложению не нужно будет постоянно пропускать текущий кадр через модель YOLO, чтобы отслеживать, где находятся объекты, что может привести к прерывистому взаимодействию с пользователем.

Отслеживание объектов с помощью пакета Vision. Vision - это пакет от Apple, который содержит множество различных методов для изображений и видео. Он включает в себя анализ изображения, анализ последовательности изображений, отслеживание объектов, обнаружение лиц и т. д.

На веб-сайте Apple есть проект, который позволяет любому пользователю попробовать отслеживать объект на видео [42]. Опробовав это на мебели, результат был очень многообещающим. Пока детали лежали на полу с одновременным перемещением камеры, объекты были отслежены. Только когда камера двигалась таким образом, что заставляло части вращаться на изображении, это вызывало трудности в отслеживании.

Для решения этой проблемы можно выполнять обнаружение объекта за разумный промежуток времени и отслеживать его до тех пор, пока обнаружение не потребуется выполнить еще раз и т. д.

Разница между этим проектом и тестовым проектом Apple заключается в том, что отслеживание объектов будет выполняться в режиме реального времени. Это накладывает ограничение на то, с каким количеством кадров в секунду может происходить отслеживание объектов, поскольку при воспроизведении видео вы можете просто выбрать, насколько быстро вы хотите передавать новые изображения, ведь в реальном времени мир не перестает двигаться.

После внедрения системы в приложение были протестированы различные частоты кадров. Оптимальное значение было где-то между 10-30 кадрами в секунду. Если поднять выше, приложение станет не стабильным и в конечном итоге закроется.

При падении ниже 10 кадров в секунду пользователь начнет ощущать, что объекты трудно отслеживать быстрыми движениями.

Однако для этого приложения пользователь не столкнется с какими-либо сценариями, когда объекты быстро летают вокруг. Поэтому было решено оставить 20 кадров в секунду как оптимальное значение для производительности.

Код представлен в приложении М, где показано, как было установлено отслеживание объекта. При настройке этого параметра важно понимать, что тяжелые вычисления выполняются в другом потоке, а не в основном.

Сочетание обнаружения объекта с его отслеживанием. Цель трекера объекта состоит в том, чтобы избежать выполнения обнаружения объекта и его распознавания каждый кадр. Кроме того, выполнение этого с сетями YOLO показало прерывистые результаты, когда объект может быть идентифицирован в одном кадре, не идентифицирован в следующем, а затем снова идентифицирован.

Один из способов решить эту проблему - выполнить обнаружение и распознавание один раз, а затем продолжить отслеживать положение этих предметов с помощью более простого алгоритма. Обнаружение и распознавание объектов будут повторяться, но обновлять состояние прямоугольников можно только в том случае, если идентифицированы правильные объекты.

В этом проекте был использован ARKit от Apple. Как указывалось, ранее при работе с ARKit кадр из камеры выбирается либо путем вызова функции snapshot () объекта ARSCNView, либо путем получения атрибута currentFrame из ARSession. Однако оба эти изображения содержат как изображение с камеры, так и все виртуальные элементы, представленные в ARScene.

Весь этот сценарий создал своего рода петлю положительной обратной связи, потому что виртуальная стрелка, которая отображалась между объектами, обычно находилась внутри одного из прямоугольников отслеживания. На рисунке 3.13 показана ранняя версия приложения, когда сеть распознала две части, и вокруг них был нарисован контур. Между ними

изображена стрелка, показывающая, как соединить две части. Объекты постоянно отслеживаются в кадре, и стрелка также обновляется. Таким образом, положение стрелки определялось путем отслеживания правильного прямоугольника, отклоняющего положение стрелки. Даже малейшее изменение на изображении заставляло стрелку подпрыгивать вверх и вниз, пока она не выходила из рамки.

Вывод - пропустить отслеживание объектов и решить проблему другим способом.

Было сделано предположение, что, как уже говорилось ранее, детали мебели лежат на земле, пока пользователь использует приложение. Это означает, что можно положиться на ARKit, чтобы отслеживать, где находится объект.

Применение этого решения сделало все приложение намного лучше и проще для кодирования. Вывод из этого заключается в том, что добавление новых технологий в проект не всегда приводит к получению лучшего результата. Иногда лучше использовать компоненты так, как они предназначены для использования, и не усложнять их.

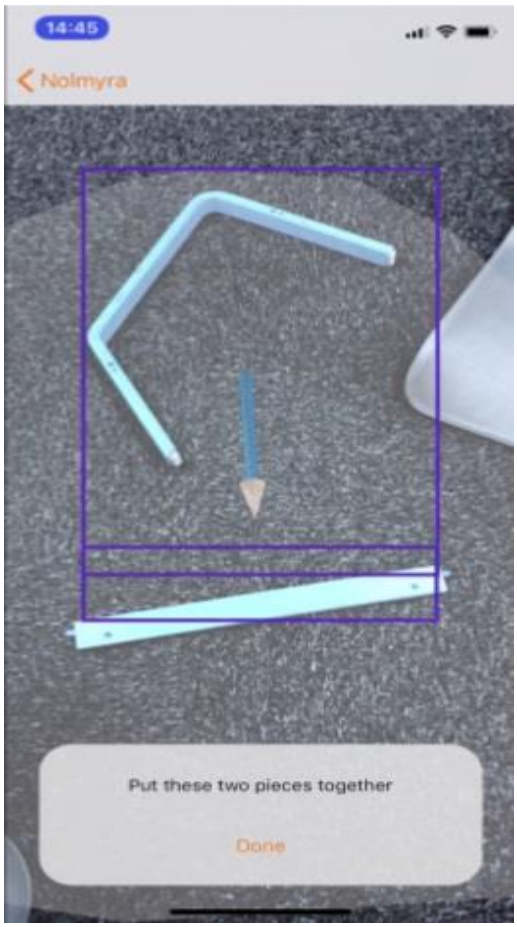

Рисунок 3.13 - Ранняя версия приложения

Таким образом в данной главе были выбраны методы для реализации мобильного приложения объединяющего сверточные нейронные сети и дополненную реальность.

Глава 4 Финальная версия приложения по сборке мебели с элементами дополненной реальности

4.1 Создание приложения, совмещающего нейронные сети и дополненную реальность для iOS

Приложение построено для iOS с использованием Xcode и Swift 4.0. Приложение было разработано специально для iPhone X. Оно также было протестировано только на iPhone X. Однако приложение должно работать на других моделях iPhone, которые также поддерживают ARKit 2.0.

4.1.1 Диаграммы классов приложения для iOS

Ниже приведены некоторые диаграммы классов. Диаграммы разделены на три части, чтобы их можно было прочитать.

Приложение следует шаблону проектирования Model View Controller, а также шаблону проектирования Delegate. Это решение было принято, поскольку многие инструменты и функции в iOS и Swift реализованы таким образом.

AssemblerViewController является контроллером для представления, которое содержит ARSCNView. Он отвечает за то, что происходит, когда пользователь нажимает на кнопку в приложении и отображает правильную информацию в нужное время. Он также содержит Instruction Executioner, который содержит набор инструкций, которые он получает при инициализации. Исполнитель инструкции выполняет текущую инструкцию и поэтому решает, что произойдет, когда она будет выполнена. Каждая инструкция выполняется в рабочем потоке очереди, чтобы избежать зависания видео обратной связи во время выполнения. На рисунках 4.1 – 4.3 изображены диаграммы классов приложения.

Рисунок 4.1 диаграмма классов ассемблерного представления части приложения. Класс Мебель в правом нижнем углу является неполным и показан на рисунке 4.2.

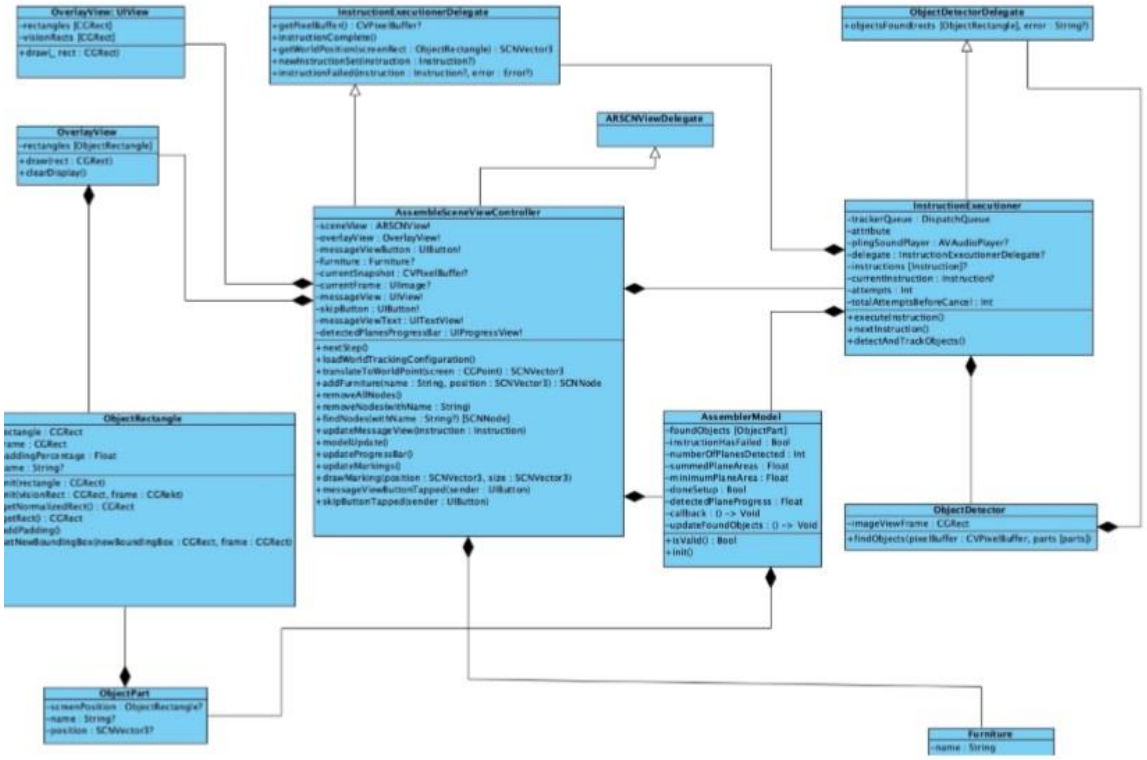

Рисунок 4.1 - Диаграмма классов приложения

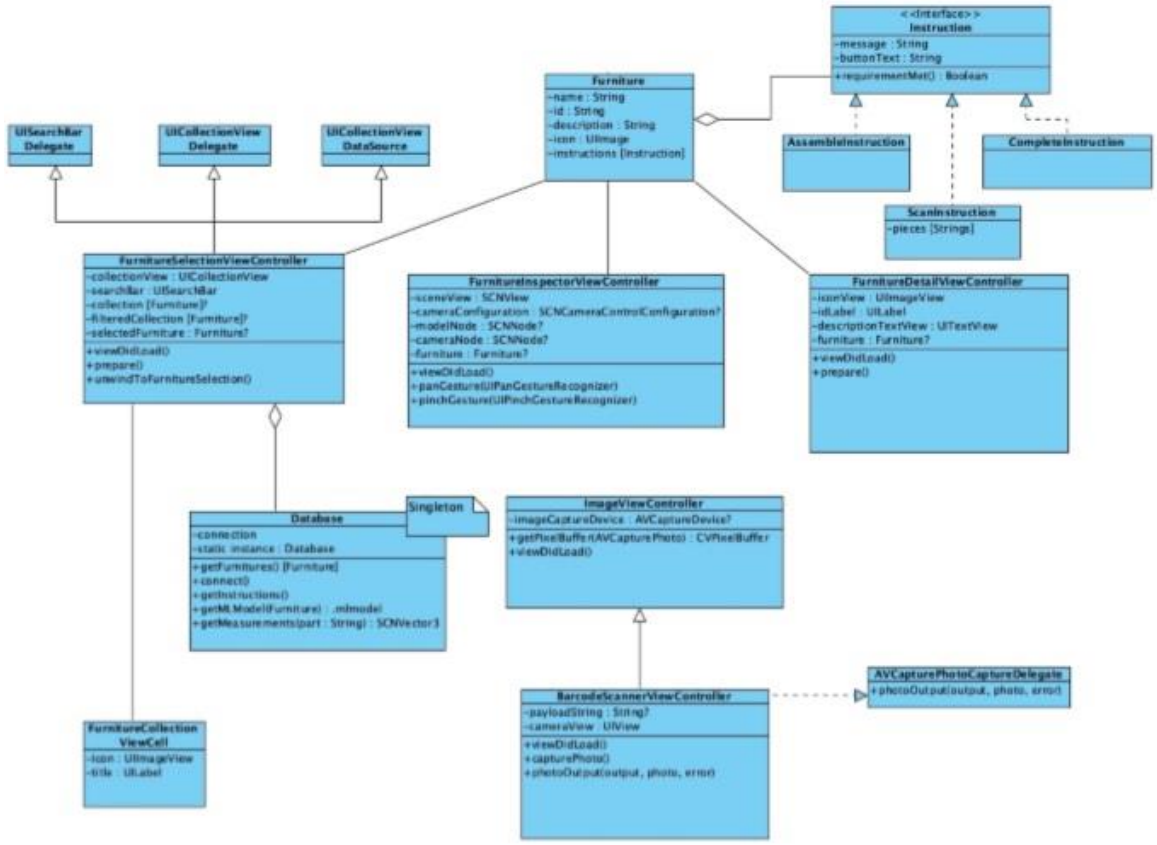

Рисунок 4.2 – Продолжение диаграмма классов приложения

На рисунке 4.3 изображена диаграмма классов служебной части приложения. Это некоторые классы, которые упрощают понимание и использование остальной части кода в приложении.

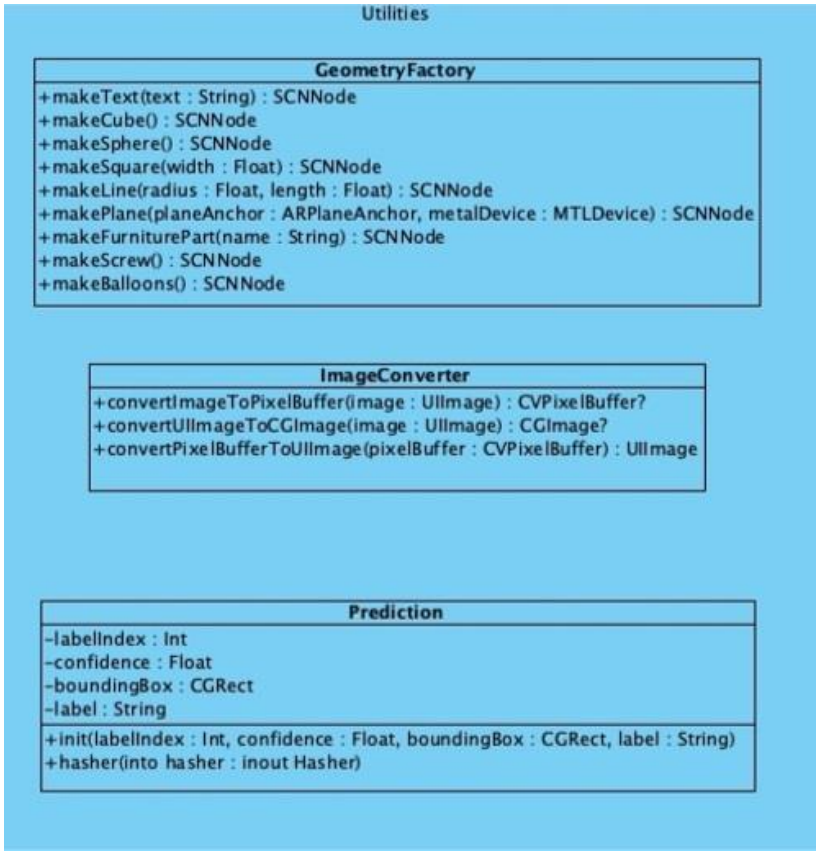

Рисунок 4.3 - Диаграмма классов служебной части приложения

InstructionExecutioner содержит ObjectDetector, который находит конкретные объекты в обученной модели из пиксельного буфера. Найденные объекты передаются обратно как ObjectRectangles. Прямоугольники объекта представляют собой ограничивающие рамки на экране объекта. Они представлены в виде CGRECT и могут быть сохранены и извлечены как обычные или нормализованные прямоугольники. Нормализованные варианты необходимы во всех видах целей машинного обучения, а обычные формы используются во всех целях графического интерфейса, например, в OverlayView. (Нормализованные прямоугольники имеют значения 0-1 для ширины, высоты, x- и y-позиции. Начало координат в левом нижнем углу.)

AssemblerModel является моделью в AssemblerViewController, но также используется в InstructionExecutioner, поскольку они сильно зависят друг от друга. Наиболее важные атрибуты, которые он содержит, - это ObjectParts, которые были обнаружены приложением во время выполнения текущей инструкции. Таким образом, все части, необходимые для текущей инструкции, не обязательно должны быть в одном изображении вместе, а вместо этого могут быть выделены отдельно.

Мебель, которую пользователь хочет собрать, выбирается в представлении коллекции FurnitureSelectionViewController, которое соответствует протоколу UICollectionViewDelegate и UICollectionViewDataSource для функциональности представления коллекции.

Данные для просмотра в контроллере выбираются из класса Database. Возвращаемые данные из класса закодированы с самого начала, но их можно изменить на получение данных с сервера.

Класс мебель содержит всю информацию о конкретной мебели, а также набор инструкций по ее сборке. Набор инструкций состоит из инструкций, которые могут иметь вид ScanInstruction, AssembleInstruction или CompleteInstruction. Инструкции обычно представляют собой просто текстовые инструкции для пользователя, в то время как инструкции сканирования говорят исполнителю инструкций, что нужно искать определенные части на этом этапе. Инструкция по сборке запускается, когда пользователь находится в процессе сборки двух частей. Наконец дается полная инструкция, которая говорит пользователю, что мебель полностью собрана.

Для сканирования штрих-кодов используется BarcodeScannerViewController, который наследует все функции захвата изображений из ImageViewController.

GeometryFactory создает всю трехмерную геометрию для ARScene и возвращает узлы, содержащие эту геометрию. Примером является то, что он

может создавать все детали мебели как узлы виртуальных объектов для размещения на сцене.

Произведено много преобразований между пиксельными буферами и UIImage, и ImageConverter облегчает необходимость делать это во многих местах.

Наконец, класс Prediction помогает детектору объектов при получении результатов из VNCoreMLRequest.

4.1.2 Соединение двух деталей мебели в дополненной реальности

После нахождения определенных частей, которые должны быть соединены, они создаются как виртуальные объекты в сцене в соответствующих местах. Местоположение определяется с помощью тестов попадания позиций экрана в обнаруженную плоскость из ARScene.

Виртуальные объекты визуализируются в сцене с использованием SCNNode, и положение узла является источником трехмерной модели. Источник выбирается в месте, которое касается пола. Когда обе части помещены в сцену, они должны быть перемещены с анимацией соединения.

Каждый узел виртуального объекта может встраивать другой узел, называемый «точкой привязки». Положение этого узла в родительском узле - то, где другой объект должен быть соединен. Для того, чтобы соединить их, либо объект без узловой точки перемещается в положение привязанной точки, или два объекта двигаются так, что каждая из их опорных точек находится в одинаковом положении. Это достигается с помощью алгоритма, описанного в коде в приложении Н.

4.1.3 Готовый графический интерфейс

Готовый графический интерфейс похож на прототип, но некоторые детали отличаются. Полученные снимки экрана можно увидеть на рисунках

4.4 изображения при запуске приложения и 4.5 изображения из приложения при запуске ARScene, поиск объектов и анимация при их объединении. Зеленый прямоугольник на полу указывает на то, что деталь была распознана приложением.

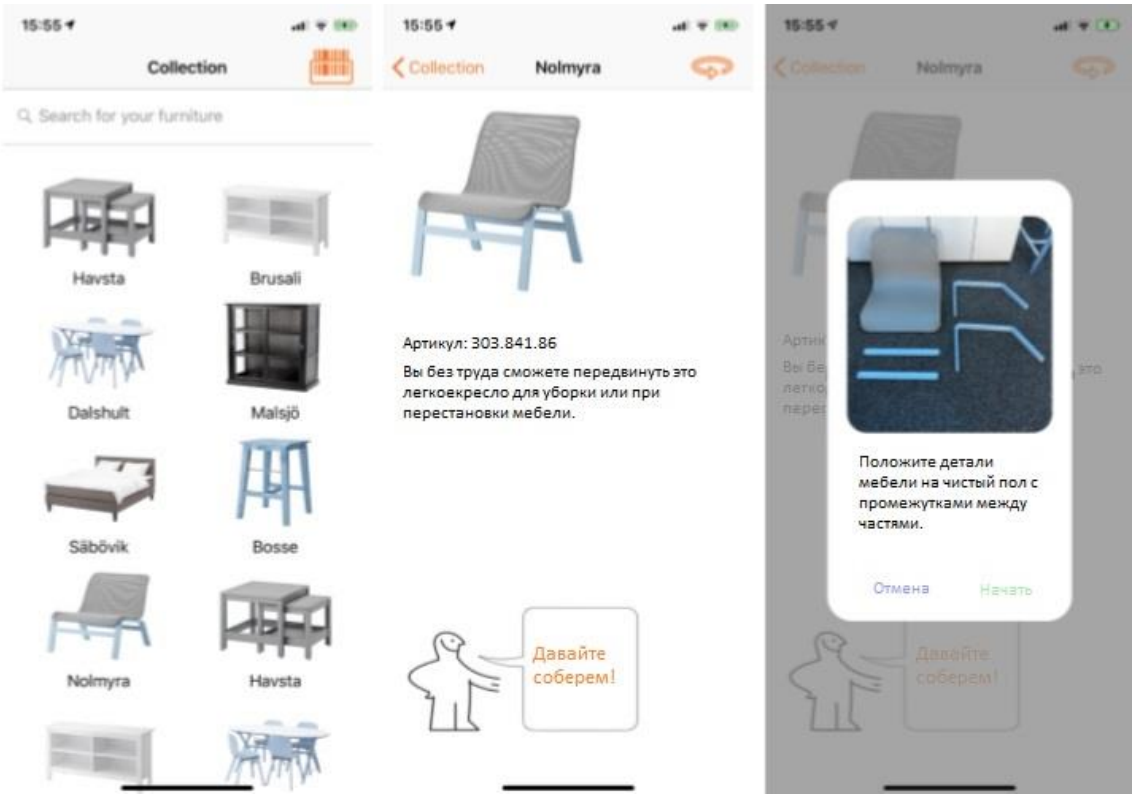

Рисунок 4.4 - Изображения при запуске приложения

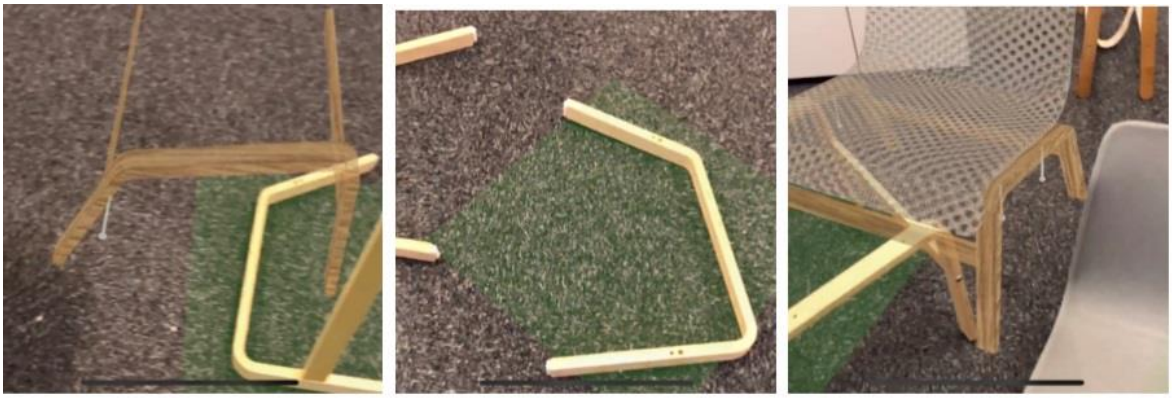

Рисунок 4.5 - Изображения из приложения при запуске ARScene.

Таким образом полностью описано созданное приложение, объединяющее сверточные нейронные сети и дополненную реальность.

4.2 Пользовательское тестирование

Оценка работы возможна на техническом уровне, чтобы понять, возможна ли комбинация AR с распознаванием объекта. Это можно сделать, например, убедившись, что приложение работает с приемлемой частотой кадров. Однако, так как в этой работе требуется исследование комбинации технологий, такой подход не является правильным. Поэтому, необходимо протестировать приложение, а также получить оценки с помощью пользовательских тестов.

При настройке пользовательского тестирования было решено использовать в основном методы наблюдения и демонстрации задач, а также вопросник. Эти методы рекомендованы в книге Soren Lauesen's «Требования к программному обеспечению, стили и методы» [45].

Пользователям было предложено воспользоваться приложением, а затем ответить на несколько вопросов в форме анкеты. Так же во время тестирования использовалась запись экрана на телефоне, чтобы потом можно было повторить сценарий.

Анкета задала участникам следующие вопросы:

– Знаете ли вы, что можете пропустить инструкции?

– Насколько легко было понять, как перейти к следующему шагу?

– Насколько легко было понять, как части соединяются?

– Насколько легко было использовать приложение?

– По сравнению с использованием бумажных инструкций, приложение помогло понять, как собрать мебель? Если нет, то почему?

– Предложения по улучшению

На рисунках с 4.6 по 4.10 показаны графики и круговые диаграммы, на которых представлены ответы участников на некоторые из заданных им вопросов. На рисунке 4.6 изображены результаты, когда пользователей спрашивали: «Насколько легко было понять, как перейти к следующему шагу?». Ось X - это их оценка от 1 до 5, а ось Y - это количество

отвечающих. На рисунке 4.7 показаны результаты, когда пользователей спрашивали: «Насколько легко было понять, как части соединяются?». Ось X - их оценка от 1 до 5, а ось Y - это количество отвечающих. На рисунке 4.8 находятся результаты, когда пользователям был задан вопрос: «Знаете ли вы, что можете пропустить инструкции?». На рисунке 4.9 изображены результаты, когда пользователей спрашивали: «Насколько легко было понять, как добраться до следующего шага?» Ось X - это их оценка от 1 до 5, а ось y - это количество отвечающих. На рисунке 4.10 показаны результаты, когда пользователей спрашивали: «По сравнению с использованием бумажных инструкций, приложение облегчило понимание того, как собрать мебель?»

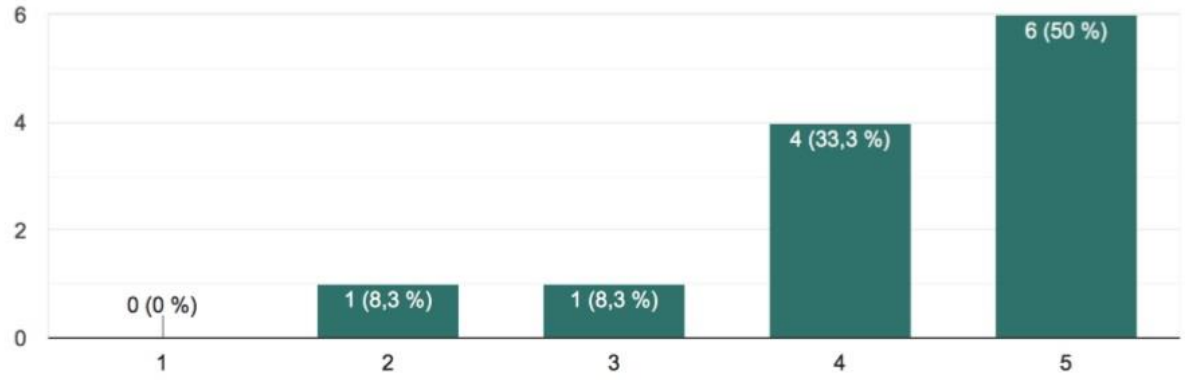

Рисунок 4.6 - Результаты пользовательского тестирования

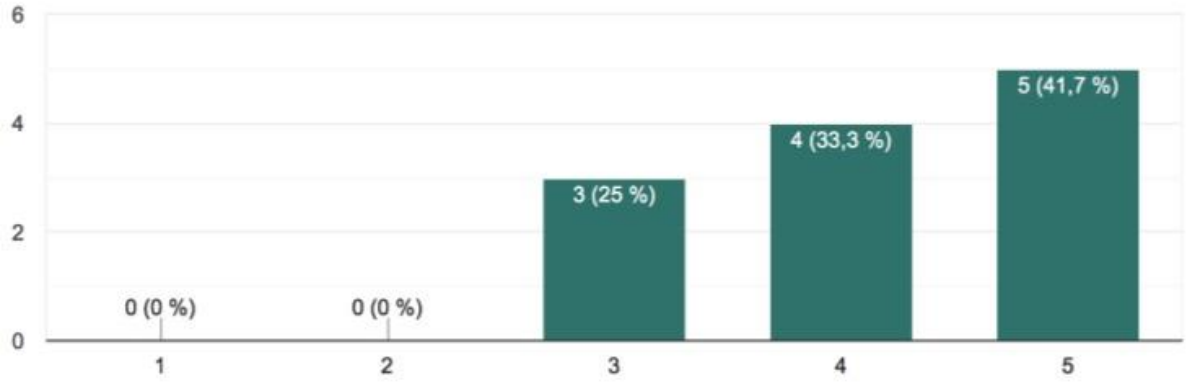

Рисунок 4.7 - Результаты пользовательского тестирования

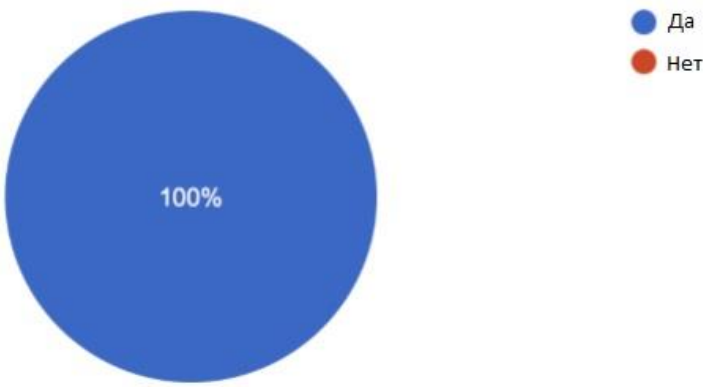

Рисунок 4.8 - Результаты пользовательского тестирования

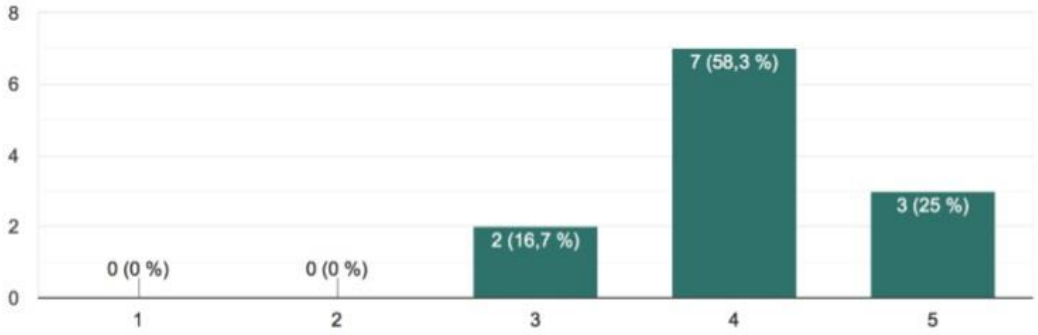

Рисунок 4.9 - Результаты пользовательского тестирования

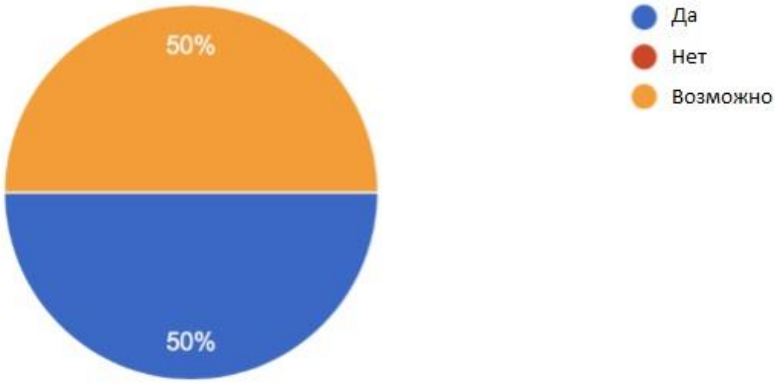

Рисунок 4.10 - Результаты пользовательского тестирования

Нужно отметить, что половина участников думает, что было проще использовать данное приложение для сборки мебели, чем использование обычной бумажной инструкции, и никто не думал, что оно более сложно, как показано на рисунке 4.10.

Ответы показывают, что большинство людей могут понять, как использовать приложение, и могут интуитивно запустить приложение и запустить процесс сборки без посторонней помощи, как показано на рисунках 4.6, 4.7 и 4.9. В некоторых случаях люди были немного смущены интерфейсом дополненной реальности. Несколько участников с трудом понимали назначение зеленых прямоугольников, которые отображались вокруг объекта. Главным образом из-за того, что эти треугольники были значительно больше самого объекта, следовательно, они часто перекрывают другие объекты.

Возникла необходимость откладывать телефон между просмотром анимации и фактической сборкой деталей мебели. Некоторым участникам приходилось поднимать телефон несколько раз на определенных этапах, чтобы убедиться, что они правильно выполнили задание.

Иногда участники путались, потому что приложение не могло найти правильные части. Это произошло по нескольким причинам. Во-первых, они случайно пропустили шаг в инструкции, поэтому приложение искало модели, которые еще не были собраны. Другая причина в том, что модель машинного обучения иногда не могла определить определенную ориентацию, в которой модель находилась в настоящее время. Это также происходило, когда пользователи не выполняли первую задачу, заданную приложением, а именно то, что они должны были убедиться, что части лежали на полу и не перекрывали друг друга. Решением этих проблем может быть предоставление пользователям более понятного и более интуитивного набора инструкций, чтобы пользователь случайно не пропустил важную этап. Другое решение состоит в том, чтобы переобучить модель машинного обучения, чтобы иметь возможность идентифицировать детали в большей ориентации.

Данная глава полностью описывает созданное приложение показаны диаграммы классов, описан функционал и разобран графический интерфейс. А также проведено и описано независимое тестирование пользователями.

Заключение

Основная цель этого проекта состояла в том, чтобы выяснить, возможно ли использовать обнаружение объектов и распознавание объектов вместе с дополненной реальностью, как в отношении того, насколько это технически возможно, так и создания ценности для пользователя.

С технической точки зрения это безусловно возможно. В работе показано, что нейронная сеть, такая как YOLO, может использоваться для обнаружения объектов, в то же время имея активную сессию AR для информирования об окружающем пространстве и о том, где находится устройство. Основная задача заключается в сборе большого количества соответствующих данных и обучении нейронной сети, чтобы дать хорошее обобщение этих данных. Это может быть частично облегчено с помощью трансферного обучения. Кроме того, существует множество платформ, которые необходимо использовать, чтобы избежать разработки с нуля.

С точки зрения потребительской ценности также существует потенциальный, но не такой большой интерес к текущему оборудованию. Из пользовательских тестов получены несколько интересных результатов. Хотя пользователям понравилось решение по безбумажному руководству по сборке в AR, они не стремились держать телефон во время просмотра инструкций, чтобы потом положить его, когда собирали предмет мебели. Кроме того, все они видели большой потенциал в использовании этого типа технологий, но возможность использования очков дополненной реальности была бы действительно великолепна. Тем не менее, использование такого оборудования не представлялось возможным.

Полностью рабочий прототип для сочетания обнаружения объектов с дополненной реальностью был разработан для iPhone X.

В связи с быстрым развитием областей, обсуждаемых в этой работе, многое может быть улучшено в будущем. С появлением новых аппаратных средств дополненной реальности и смешанной реальности, разработанных крупными франшизами, такими как Facebook [43], производительность будет

расти очень быстро. Ниже описаны несколько способов улучшения не только этого приложения, но и комбинации используемых технологий в целом.

Одним из улучшений этого приложения было бы перенести его не только на портативные устройства, но и на головной убор, что позволяет пользователю строить обеими руками и не останавливаться во время процесса сборки.

Еще одна проблема, связанная с моделью, заключается в том, что программа не может обнаружить винты и болты из-за их сравнительно небольшого размера, вместо этого она просто информирует пользователя, что винты следует использовать во время определенного этапа с использованием анимации. Вероятно, эту проблему можно решить, адаптировав модель к меньшим объектам. Будущая работа может состоять в том, чтобы выяснить, возможно ли это.

Поскольку в ходе пользовательских тестов возникла путаница в отношении зеленой маркировки, было бы лучше отметить только точное положение детали, а не окружающее пространство. Для этого приложения известен только ограничивающий прямоугольник 2D-изображения, а не плоскость, где лежит предмет.

Возможным решением этой проблемы в будущем может стать внедрение сети Mask R-CNN. Однако проблема с этим типом сейчас заключается в том, что он не работает в приложениях реального времени (определенно не на мобильных устройствах). Если бы это работало, то могло бы быть реализовано в приложении для более приятного использования.

С тех пор как ARKit 2.0 был выпущен, можно поделиться опытом AR с другим пользователем. Поскольку приложение уже использует ARKit 2.0, оно будет относительно простым для реализации общего опыта. Общий опыт может помочь нескольким людям работать вместе, чтобы собрать мебель. Большая мебель редко собирается одним человеком.

Чтение инструкций на экране при просмотре видео не является наиболее понятным в среде AR. Глаза в основном сосредоточены на том, что

происходит в окружающей среде, и текстовые инструкции, данные на оверлее, можно легко пропустить. Кроме того, людям с плохим зрением могут быть трудны как чтение инструкций, так и просмотр 3D-моделей. Эту проблему можно решить, введя озвученные инструкции в приложении.

Поскольку винты были слишком малы для обнаружения, то же самое можно сказать и о точках крепления на деталях мебели, которые соединяют друг друга. Улучшение в будущем может заключаться в том, чтобы попытаться определить точные опорные точки на реальных деталях мебели и визуализировать виртуальный объект прямо в этом месте. Это потребует дополнительной работы по обучению нейронной сети для обнаружения этих мелких частей и, возможно, более высокого разрешения на изображениях.

Одной из текущих проблем с дополненной реальностью является то, что модели отображаются поверх реального мира. Когда в сцене нет другого объекта и есть простая плоскость для его визуализации, результат может выглядеть довольно реалистичным. Однако, когда другие объекты находятся на сцене, иллюзия реализма легко теряется. Пример этого можно увидеть на рисунке 5.1. Слева нет предметов спереди, поэтому выглядит реалистично. Справа на книжной полке есть лампа.

Одним из способов решения этой проблемы было бы создание трехмерной модели реального мира, позволяющей находить объекты переднего плана и, таким образом, добавлять маску прозрачности к объекту, который нужно визуализировать в реальном мире, чтобы усилить эффект иллюзии.

На данный момент ARKit не может сделать это в режиме реального времени. Тем не менее, есть ученые, работающие над решением проблемы окклюзии в дополненной реальности, такие как Шах, Нияти.

С более подробными выводами можно ознакомиться в приложении О.

Список используемых источников

1. Microsoft HoloLens, Software Asset Management – Microsoft SAM, Microsoft https://www.microsoft.com/en-us/hololens
2. Glass Explorer Edition, Google Developers, Google https://developers. google.com/glass/
3. Augmented Reality, Apple https://www.apple.com/lae/ios/ augmented-reality/
4. Pokémon GO, NIANTIC https://www.pokemongo.com/en-us/
5. IKEA Place, App Store, IKEA Systems B.V, (2017) https://itunes.apple. com/us/app/ikea-place/id1279244498?mt=8
6. Digi-Capital. Ubiquitous $90 billion AR to dominate focused $15 billion VR by2022 (January26,2018)https://www.digi-capital.com/news/2018/01/ ubiquitous-90-billion-ar-to-dominate-focused-15-billion-vr-by-2022/
7. Fieldbit Hero, FieldBit https://www.fieldbit.net/products/ fieldbit-hero/
8. Sanni Siltanen, Theory and applications of marker-based augmented reality, Copyright c •VTT2012(ISBN:978-951-38-7449-0)https://www.vtt.fi/inf/ pdf/science/2012/S3.pdf
9. Alex Krizhevsky and Sutskever, Ilya and Hinton, Georey E, ImageNet Classification with Deep Convolutional Neural Networks in Advances in Neural Information Processing Systems 25 ed.F.PereiraandC.J.C.BurgesandL.Bottou and K. Q. Weinberger, pp 1097-1105 (2012) http://papers.nips.cc/paper/ 4824-imagenet-classification-with-deep-convolutional-neural-networks. pdf
10. Joseph Redmo, Santosh Divvala, Ross Girshick, Ali Farhadi (2016), University of Washington , Allen Institute for AI , Facebook AI Research You Only Look Once: Unified, Real-Time Object Detection https://arxiv.org/pdf/1506.02640.pdf
11. D. Chatzopoulos, C. Bermejo, Z. Huang and P. Hui, "Mobile Augmented RealitySurvey: FromWhereWeAretoWhereWeGo,"inIEEE Access,

vol.5, pp.6917-6950,2017.doi: 10.1109/ACCESS.2017.2698164http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7912316&isnumber=7859429

12. Region-based Segmentation and Object Detection S. Gould, T. Gao and D. Koller, in Advances in Neural Information Processing Systems 22, ed. Y. Bengio and D. Schuurmans and J. D. Laerty and C. K. I. WilliamsandA.Culotta, pp655–663(2009) http://papers.nips.cc/paper/ 3766-region-based-segmentation-and-object-detection.pdf

13. YannLeCunn, Marc'AurelioRanzato, DeepLearningTutorial, ICML,Atlanta, (2013), https://cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf

14. MockFlow, A Produle Systems Pvt Ltd https://mockflow.com/

15. Extreme Programming O'Reilly Media ISBN-13: 978-0596004859

16. MyronKrueger - Videoplace, Responsive Environmen https://www.youtube. com/watch?v=dmmxVA5xhuo

17. SLAM: Simultaneous Localization and Mapping - Wolfram Burgard, Cyrill Stachniss, Kai Arras, Maren Bennewitz http://ais.informatik. uni-freiburg.de/teaching/ss12/robotics/slides/12-slam.pdf

18. Adam: A Method for Stochastic Optimization, 22 Dec 2014 Diederik P. Kingma, Jimmy Ba https://arxiv.org/abs/1412.6980

19. Neural Networks for Machine Learning, Lecture 6a Georey Hinton http:// www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

20. An overview of gradient descent optimization algorithms Sebastian Ruder http://ruder.io/optimizing-gradient-descent/index.html#fn16

21. Beware of overfitting and underfitting, O'Reilly https://www.oreilly.com/library/view/scala-and-spark/9781785280849/ 3c1c7845-811d-47b9-a54f-c2584fe930b3.xhtml

22. Dropout: A Simple Way to Prevent Neural Networks from Overfitting Nitish Srivastava, Georey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov http://jmlr.org/papers/v15/srivastava14a.html

23. Batch Normalization layer, Tensorflow https://www.tensorflow.org/api_ docs/python/tf/nn/batch_normalization

24. Vincent Dumoulin, Francesco Visin, A guide to convolution arithmetic for deep learning MILA, Université de Montréal, AIRLab, Politecnico di Milano (2018) (arXiv:1603.07285v2) https://arxiv.org/pdf/1603.07285.pdf

25. Convolutional Neural Network 3 things you need to know, MathWorks https://www.mathworks.com/solutions/deep-learning/ convolutional-neural-network.html

26. ImageNet2016StanfordVisionLab, StanfordUniversity, PrincetonUniversity http://www.image-net.org

27. Scanning and Detecting 3D Objects, Apple https://developer.apple.com/ documentation/arkit/scanning_and_detecting_3d_objects

28. Face Detection System Based on Viola - Jones Algorithm, Mehul K Dabhi1, Bhavna K Pancholi2 https://pdfs.semanticscholar.org/f63c/ fcdd63bd34bcd6ec028169e6fe144e9cc83c.pdf

29. Lode's Computer Graphics Tutorial - Flood Fill, Lode Vandevenne https: //lodev.org/cgtutor/floodfill.html

30. AVDepthData, Apple Documentation https://developer.apple.com/ documentation/avfoundation/avdepthdata

31. Object recognition without deep-learning, Combine http://combine.se/ object-recognition-without-deep-learning/

32. Derek Bradley, Gerhard Roth, Adaptive Thresholding Using the Integral Image, Carleton University, Canada & National Research Council of Canada (2005), http://people.scs.carleton.ca/~roth/iit-publications-iti/ docs/gerh-50002.pdf

33. Joseph Redmon, Ali Farhadi, YOLO9000: Better, Faster, Stronger, University of Washington & Allen Institute for AI (2016) https://arxiv.org/pdf/1612. 08242.pdf

34. JosephRedmon,AliFarhadYOLOv3: AnIncrementalImprovement University of Washington (2018) https://arxiv.org/pdf/1804.02767.pdf

35. The PASCAL Visual Object Classes http://host.robots.ox.ac.uk/pascal/VOC/

36. Rafael Padilla, Metrics for object detection https://github.com/rafaelpadilla/Object-Detection-Metrics

37. Detection Evaluation, Common Objects in Context (COCO) http://cocodataset.org/#detection-eval

38. Advanced Usage, Turi Create, Apple https://apple.github.io/turicreate/docs/userguide/object_detection/advanced-usage.html

39. Turi Create, Apple https://github.com/apple/turicreate

40. Simple Image Annotator, Sebastian G. Perez (@sgp715) https://github. com/sgp715/simple_image_annotator

41. Jan Hosang, Rodrigo Benenson, Bernt Schiele Learning non-maximum suppression, Max Planck Institut für Informatik, Saarbrücken, Germany (9 May 2017) https://arxiv.org/pdf/1705.02950.pdf

42. Tracking Multiple Objects or Rectangles in Video, Apple https://developer.apple.com/documentation/vision/tracking_multiple_objects_or_rectangles_in_video

43. TechCrunch,Facebook's Head Of Augmented reality on its plans for AR glasses Published on Oct 24, 2018 https://youtu.be/JEGqc9wzC0o?t=1041

44. Shah, Niyati. Realtime Object Occlusion In Augmented Reality Environments. (2018) , B. Thomas Golisano College of Computing and Information Sciences Rochester Institute of Technology, https://pdfs.semanticscholar.org/0329/d772efe01b5d818dfc22e4a357d03324a01e.pdf

45. Software requirements, Styles and techniques Soren Lauesen ISBN: 0-20174570-4

46. Seene SLAM Technology - culturengine https://www.youtube.com/watch? v=434SsV9nGHc

47. Keras: The Python Deep Learning library, Keras https://keras.io

Приложение А
Код программы

```
1 func loadWorldTrackingConfiguration()
2 {
3 let configuration = ARWorldTrackingConfiguration()
4 configuration.planeDetection = [.horizontal]
5
6 // Все отслеживаемые объекты содержатся в папке «Objects».
7 guard let detectingObjects = ARReferenceObject.referenceObjects(inGroupNamed: "Objects", bundle: nil) else { return }
8 configuration.detectionObjects = detectingObjects
9
10 // Настройка отслеживания изображений
11 for imageURL in trackingImageURLs
12 {
13 guard let image: CGImage = UIImage(named: imageURL)?.cgImage else { return }
14 let referenceImage = ARReferenceImage(image, orientation: CGImagePropertyOrientation.up, physicalWidth: 0.3)
15 configuration.detectionImages.insert(referenceImage)
16 }
17
18 configuration.maximumNumberOfTrackedImages = trackingImageURLs.count 19
20 // Запуск сеанса с настройкой
21 sceneView.session.run(configuration)  }
```

Загрузка сцены

```
1 // Загрузка сцены
2 let scene = SCNScene(named: "art.scnassets/world.scn")!
sceneView.scene = scene


1 func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor)
2 {
3 // Если обнаружен объект
4 if let objectAnchor = anchor as? ARObjectAnchor
5 {
6 // Добавить 3D текст в сцене
7 let objectName = objectAnchor.referenceObject.name!
8 let textNode = GeometryFactory.makeText(text: objectName)
9 node.addChildNode(textNode)
10 }
11 // Если обнаружена плоскость
12 else if let planeAnchor = anchor as? ARPlaneAnchor
13 {
14 // Добавить геометрию плоскости обнаруженного пола к сцене
15 node.addChildNode(GeometryFactory.createPlane(planeAnchor: planeAnchor, metalDevice: metalDevice!))
16 model.numberOfPlanesDetected += 1
17 }
18 }
1 #Скрипт Python с тренеровки
2 #Project path:
```

```
3 master-thesis/Training/trainer.py
4 import tensorflow as tf
5 from tensorflow import keras
6 import numpy as np
7 import matplotlib.pyplot as plt
8 from PIL import Image
9 from sklearn.utils import shuffle
10
11 train_images = []
12 train_labels = []
13 loadImages(train_images, train_labels, "Train", 200)
14 train_images = reshapeArray(train_images)
15
16 test_images = []
17 test_labels = []
18 loadImages(test_images, test_labels, "Test", 39)
19 test_images = reshapeArray(test_images)
20
21 # Create the neural network
22 model = keras.Sequential([
23 keras.layers.Conv2D(4, kernel_size=(5, 5), strides=(2, 2), input_shape=(image_height, image_width, number_of_color_channels)),
24
25 ["The code for the hidden layers"]
26
27 keras.layers.Dense(4, activation=tf.nn.softmax)
28 ])
29
30 model.compile(optimizer=keras.optimizers.Adam(),
31 loss='sparse_categorical_crossentropy',
32 metrics=['accuracy'])
33
34 train_data, train_labels = shuffle(train_data, train_labels)
35
36 early_stopping = keras.callbacks.EarlyStopping(monitor='val_acc', patience=5, verbose=1)
37 checkpoint = keras.callbacks.ModelCheckpoint("./Models/Nolmyra.h5", monitor='val_acc', 38 verbose=1, save_best_only=True, save_weights_only=False, mode='auto', period=1)
39
40 history = model.fit(train_data, train_labels, epochs=40, batch_size=10, validation_data=(test_images, test_labels), callbacks=[early_stopping, checkpoint] , verbose=1)
41
42 model.save("./Models/recognizer.h5")
```

```
1 Conv2D(4, kernel_size=(5, 5), strides=(2, 2), input_shape=(image_height, image_width, number_of_color_channels)),
2 Conv2D(4, kernel_size=(3, 3), strides=(1, 1), input_shape=(image_height, image_width, number_of_color_channels)),
3 MaxPool2D(pool_size=(2, 2), padding="valid"),
4 BatchNormalization(),
```

```
5 LeakyReLU(),
6 Conv2D(8, kernel_size=(3, 3), strides=(1, 1)),
7 Conv2D(8, kernel_size=(3, 3), strides=(1, 1)),
8 Conv2D(8, kernel_size=(3, 3), strides=(1, 1)),
9 MaxPool2D(pool_size=(2, 2), padding="valid"),
10 BatchNormalization(),
11 LeakyReLU(),
12 Conv2D(16, kernel_size=(3, 3), strides=(1, 1)),
13 Conv2D(16, kernel_size=(3, 3), strides=(1, 1)),
14 Conv2D(16, kernel_size=(3, 3), strides=(1, 1)),
15 MaxPool2D(pool_size=(2, 2), padding="valid"),
16 BatchNormalization(),
17 LeakyReLU(),
18 Flatten(),
19 Dropout(0.5),
20 Dense(64, kernel_regularizer=keras.regularizers.l2(0.003), activation=tf.nn.relu),
21 GaussianNoise(0.2),
22 Dense(64, kernel_regularizer=keras.regularizers.l2(0.003), activation=tf.nn.relu),
23 Dropout(0.25),
24 Dense(4, activation=tf.nn.softmax)
```

```
1 #Project path: master-thesis/FeatureTraining/transferLearning.py
2 from keras import applications
3 from keras.preprocessing.image import ImageDataGenerator
4 from keras import optimizers
5 from keras.models import Sequential, Model
6 from keras.layers import Dropout, Flatten, Dense, GlobalAveragePooling2D, Input, Conv2D, MaxPool2D
7 from keras import backend as k
8 from keras.callbacks import ModelCheckpoint, LearningRateScheduler, TensorBoard, EarlyStopping
9
10 #Load data and pretrained model
11 img_width, img_height = 256, 256
12 train_data_dir = "data/train"
13 validation_data_dir = "data/val"
14 nb_train_samples = 129
15 nb_validation_samples = 21
16 batch_size = 16
17 epochs = 50
18 input_layer = Input(shape=(256,256,3))
19 model = applications.VGG16(include_top=False, weights='imagenet', input_tensor=input_layer, pooling=None)
20
21 #Cut network and add own layers
22 x = model.get_layer( 'block5_pool').output
23 x = Flatten()(x)
24 x = Dense(512, activation= "relu")(x)
25 predictions = Dense(4, activation="softmax")(x)
26
27 # creating the composed model
28 model_final = Model(inputs = model.input, outputs = predictions)
```

```
29 for layer in model_final.layers[:-2]:
30 layer.trainable = False
31
32 # compile the model
33 model_final.compile(loss = "categorical_crossentropy", optimizer = optimizers.SGD(lr = 0.0001, momentum = 0.9), metrics=["accuracy"])
34
35 #Hidden lines of code
36 .....
37
38 #Train the model
39 hist = model_final.fit_generator(
40 train_generator,
41 epochs = epochs,
42 validation_data = validation_generator,
43 callbacks = [checkpoint, early])
```

```
1 let configuration = ARWorldTrackingConfiguration()
2 guard let detectingObjects = ARReferenceObject.referenceObjects(inGroupNamed: "Objects", bundle: nil) else { return }
configuration.detectionObjects = detectingObjects
```

```
1 import turicreate as tc
2 tc.config.set_num_gpus(0)
3
4 # Load the data
5 train_data = tc.SFrame("Train_Data.sframe")
6
7 #Random split train data to get specific training size
8 train_data, unused_data = train_data.random_split(0.3)
9
10 test_data = tc.SFrame("Test_Data.sframe")
11
12 # Create a model
13 model = tc.object_detector.create(train_data)
14
15 # Evaluate the model and save the results into a dictionary
16 metrics = model.evaluate(test_data,metric='mean_average_precision')
17 print(metrics)
18
19 # Save the model for later use in Turi Create
20 model.save('Nolmyra030.model')
21
22 # Export for use in Core ML
23 model.export_coreml('Nolmyra030.mlmodel', include_non_maximum_suppression=False)
```

```
1 #Load test data
2 test_data = tc.SFrame("Test_Data.sframe")
3
4 #Load created model
```

```
5 model = tc.load_model('Nolmyra.model')
6
7 # Save predictions to an SArray and draw predicted bounding boxes
8 predictions = model.predict(test_data)
9 predictions_stacked = tc.object_detector.util.stack_annotations(predictions)
10 image_prediction = tc.object_detector.util.draw_bounding_boxes(test_data['image'], predictions)
11
12 #Look through the predicted bounding boxes
13 image_prediction.explore()
```

```
1 private func filterOverlappingPredictions(unorderedPredictions: [Prediction], nmsThreshold: Float) -> [Prediction]
2 {
3 var predictions = [Prediction]()
4 let orderedPredictions = unorderedPredictions.sorted { $0.confidence > $1.confidence }
5 var keep = [Bool](repeating: true, count: orderedPredictions.count)
6 for i in 0..<orderedPredictions.count {
7 if keep[i] {
8 predictions.append(orderedPredictions[i])
9 let bbox1 = orderedPredictions[i].boundingBox
10 for j in (i+1)..<orderedPredictions.count {
11 if keep[j] {
12 let bbox2 = orderedPredictions[j].boundingBox
13 if bbox1.IoU(other: bbox2) > nmsThreshold {
14 keep[j] = false
15 }
16 }
17 }
18 }
19 }
20 return predictions
21 }
23 extension CGRect
24 {
25 func IoU(other: CGRect) -> Float
26 {
27 let intersection = self.intersection(other)
28 let union = self.union(other)
29 return Float((intersection.width * intersection.height) / (union.width * union.height))
30
```

```
1 // Project path:
2 // master-thesis/Application/ObjectDetectionInAR/Assembler/ObjectTracker.swift
3
4 func track()
5 {
6 // Инициировать все переменные
7 cancelTracking = false
8 var trackingObservations = [UUID: VNDetectedObjectObservation]()
9 var trackedObjects = [UUID: ObjectRectangle]()
10 let requestHandler = VNSequenceRequestHandler()
```

```
11 let boundingFrame = delegate?.getBoundingFrame()
12
13 // Добавьте объекты для отслеживания в созданные списки выше
14 for object in objectsToTrack
15 {
16 let observation = VNDetectedObjectObservation(boundingBox: object.getNormalizedRect(frame: viewFrame))
17 trackingObservations[observation.uuid] = observation
18 trackedObjects[observation.uuid] = object
19 }
20
21 // Цикл, пока не будет сделан запрос на отмену отслеживания
22 while true
23 {
24 if cancelTracking { break }
25
26 var rects = [ObjectRectangle]()
27 var trackingRequests = [VNRequest]()
28
29 guard let frame = delegate?.getFrame() else {
30 usleep(useconds_t(millisecondsPerFrame * 1000))
31 continue
32 }
33
34 for trackingObservation in trackingObservations
35 {
36 // Создание запроса
37 let request = VNTrackObjectRequest(detectedObjectObservation: trackingObservation.value)
38 request.trackingLevel = .fast
39 trackingRequests.append(request)
40 }
41
42 // Выполнение запроса
43 try? requestHandler.perform(trackingRequests, on: frame, orientation: CGImagePropertyOrientation.up)
44
45 for processedRequest in trackingRequests
46 {
47 // Обработка результатов от запросов
48 guard let observation = processedRequest.results?.first as? VNDetectedObjectObservation else { continue }
49
50 if observation.confidence > confidenceThreshold
51 {
52 guard let object = trackedObjects[observation.uuid] else { continue }
53 // Установить новую ограничивающую рамку
54 object.setNewBoundingBox(newBoundingBox: observation.boundingBox, frame: boundingFrame)
55 trackedObjects[observation.uuid] = object
56 trackingObservations[observation.uuid] = observation
57
```

```
58 rects.append(object)
59 }
60 }
61
62 DispatchQueue.main.async {
63 rects = rects.sorted { $0.name! < $1.name! }
64 self.delegate?.trackedRects(rects: rects)
65 }
66
67 // Отслеживание остановится, если ни одно наблюдение не имеет высокого значения достоверности
68 if rects.isEmpty
69 {
70 DispatchQueue.main.async {
71 self.requestCancelTracking()
72 self.delegate?.trackingLost()
73 }
74 }
75
76 usleep(useconds_t(millisecondsPerFrame * 1000))
77 }
78
79 DispatchQueue.main.async {
80 self.delegate?.trackingDidStop() 81 } 82 }
```

```
1 var furniturePartNodes = [SCNNode]()
2
3 for object in model.foundObjects
4 {
5 guard object.name != nil else { return }
6 guard object.position != nil else { return }
7
8 let furnitureNode = addFurniture(part: object.name!, position: object.position!)
9 furniturePartNodes.append(furnitureNode)
10 }
11
12 var previousNode: SCNNode? = nil
13 var previousAnchorPoint: SCNNode? = nil
14
15 var nodeActions = [(SCNNode, [SCNAction])]() // A list for storing animations to run on a node later
16
17 for node in furniturePartNodes
18 {
19 var actions: [SCNAction] = []
20 actions.append(SCNAction.rotate(by: -CGFloat.pi / 2, around: SCNVector3(0, 0, 1), duration: 1))
21
22 let anchorPoint = node.childNode(withName: ANCHOR_POINT, recursively: true)
23
24 if anchorPoint == nil
25 {
```

```
26 if previousAnchorPoint != nil
27 {
28  actions.append(SCNAction.move(to:  previousAnchorPoint!.worldPosition,  duration:
2))
29 }
30
31 previousNode = node
32 }
33 else
34 {
35 previousNode?.runAction(SCNAction.move(to: anchorPoint!.worldPosition, duration:
2))
36 if previousAnchorPoint != nil
37 {
38  actions.append(SCNAction.move(to:  previousAnchorPoint!.worldPosition,  duration:
2))
39          actions.append(SCNAction.move(by:          node.worldPosition.substract(other:
anchorPoint!.worldPosition), duration: 2))
40 }
41
42 previousAnchorPoint = anchorPoint
43 }
44
45 // добавляет дополнительное действие без содержимого в конце, чтобы
обработчик завершения ждал, пока не будет выполнено последнее действие
46 actions.append(SCNAction.move(by: SCNVector3Zero, duration: 1))
47
48 nodeActions.append((node, actions)) 49 } 50 }
```