УДК 004.384 **Информационные технологии**

Засыпкин Денис Сергеевич, *студент-магистр, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*

Белов Юрий Сергеевич, *к.ф.-м.н., доцент, Калужский филиал ФГБОУ ВО «Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»*

ПОДХОДЫ К ТЕСТИРОВАНИЮ И АНАЛИЗ РАБОТЫ СИСТЕМЫ УПРАВЛЕНИЯ УМНЫМ ДОМОМ НА БАЗЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

Аннотация: Интеграция мобильных устройств в сегмент домашней автоматизации является важной задачей на данный момент. При этом пользователю необходимо предоставить возможность интуитивно взаимодействовать с системой. Ранее была создана система управления умным домом. В разработанной системе пользователь может управлять устройствами, существующими в его доме, с помощью пользовательского интерфейса камеры, имеющего визуальное представление действий и обеспечивающего интуитивное взаимодействие. В данной статье описываются данные, взятые для обучения модели, процесс обучения модели, а также точность ее обучения. Также будут проведены тесты в реальных сценариях приложений для трех объектов в системе умный дом.

Ключевые слова: умный дом, система управления умным домом, обнаружение объектов.

Abstract: Integration of mobile devices into the home automation segment is an important task at the moment. At the same time, the user must be given the opportunity to interact intuitively with the system. Previously, a smart home

management system was created. In the developed system, the user can control the devices existing in his house using the camera user interface, which has a visual representation of actions and provides intuitive interaction. This article describes the data taken to train the model, the process of training the model, as well as the accuracy of its training. Tests will also be conducted in real application scenarios for three objects in the smart home system.

Keywords: smart home, smart home management system, object detection.

Введение. Независимо от окончательного решения, процесс обучения пользовательской модели ML требует набора данных изображений в качестве входных данных. Для того чтобы модель работала в определенном сценарии, ее необходимо обучить достаточному количеству данных для изучения желаемых шаблонов и функций. Количество предоставленных изображений, количество существующих классов и разнообразие изображений внутри одного класса - все это будет влиять на точность окончательной модели. По логике вещей, большее количество изображений и различные ракурсы, яркость и масштабы одного и того же объекта дали бы лучшие результаты во время процессов обучения.

Чтобы начать первоначальный подход, рассматривающий обнаружение объектов, был собран набор данных изображений, разделенных на 5 групп. Процесс обучения можно было бы провести со многими классами и устройствами, но с целью реализации и подтверждения основной концепции был сделан выбор из 5 объектов, которые можно найти в большинстве так называемых умных домов:

- Лампочки – 608 изображений
- Кондиционеры воздуха – 508 изображений
- Оконные жалюзи – 808 изображений
- Робот-пылесосы – 765 изображений
- Умная-колонка – 526 изображений

Ручная подготовка набора данных для процесса обучения является наиболее трудоемкой частью, так как после сбора всех изображений все они

должны быть отфильтрованы, помечены и, следовательно, экспортированы в формат TFRecord для интерпретации TensorFlow.

Фильтрация состоит в проверке того, не слишком ли велики все изображения для конвейера обучения, поддержании среднего размера ниже 600x600 для предотвращения проблем, связанных с памятью, и что все они в поддерживаемом формате PNG или JPEG. Часть маркировки выполняется путем определения минимальных и максимальных координат x и y, как показано на рисунке 1, которые, следовательно, будут переданы вместе с изображением в модель.

Поскольку часть маркировки является очень медленным процессом, был использован инструмент под названием Label box [1]. Этот инструмент в основном предоставляет пользовательский интерфейс в браузере для рисования прямоугольников вдоль загруженного набора данных, а затем позволяет экспортировать уже подготовленную информацию в файлы TFRecord.

Рисунок 1. Пример маркировки изображения

Обучение реальной модели. Для локального обучения модели требовалось много вычислительной мощности для интенсивного графического процессора, чтобы получить приемлемое и реалистичное время и точность обучения. При использовании облачного решения эти аппаратные ограничения не важны, поскольку такие сервисы, как Google Cloud TPU [2], предоставляют

возможность запускать современные модели машинного обучения с производительностью, достигающей отметки 100 петафлопс.

Несмотря на то, что это шаг назад по сравнению с обнаружением объектов, подход, использующий классификацию изображений, решил обе проблемы, поскольку его можно было обучить на менее мощной машине, используя уже подготовленный набор данных и файлы, но с эквивалентным использованием API TensorFlow, возвращающего аналогичные результаты. Таким образом, результат оценки модели стал более ограниченным, учитывая тот факт, что она возвращает только метку с изображения вместо координат того же объекта, идентифицированного внутри изображения.

Конфигурация среды. Для того чтобы продвинуться в реализации, необходимо было создать условия для обучения. TensorFlow может работать как в Windows, так и в Linux, но поскольку процесс установки, управление зависимостями и выполнение в системах Linux проще и плавнее, базовой рабочей средой ОС, используемой для поддержки реализации, была Ubuntu – версия 14.0 [3].

Переподготовка сети. После настройки среды следующим шагом был процесс переподготовки. Он был сделан с использованием MobileNets [4], которые представляют собой набор моделей компьютерного зрения, оптимизированных для TensorFlow, предназначенных для получения высокой точности с использованием ограниченной вычислительной мощности и ограниченных ресурсов, создавая, таким образом, легкие сверточные нейронные сети.

Мобильная сеть настраивается с помощью 2 гиперпараметров, разрешения входного изображения и относительного размера по сравнению с крупнейшей мобильной сетью, которые масштабируют соотношение между точностью и задержкой. Логично, что выбор большего разрешения изображения приводит к более трудоемкой, но более точной модели. В соответствии с этим сценарием диссертации были сохранены параметры по умолчанию с разрешением входного изображения 224 пикселей и долей 0,5 от

модели. Эти 2 параметра были переданы внутри переменных оболочки Linux, как показано на листинге 1.

Листинг 1. Переменные оболочки Linux, представляющие гиперпараметры

```
$ IMAGE_SIZE = 224
$ ARCHITECTURE = "mobilenet_0.50_${IMAGE_SIZE}"
```

Поэтому в качестве модели использовалась MobileNet_v1_0.50_224, промежуточное решение, основанное на предварительно обученной контрольной точке классификации ImageNet и учитывающее компромисс между точностью и задержкой при 150 миллионах многократных накоплений (MAC) и 1,4 миллиона параметров [5].

Для начала обучения был использован скрипт на Python, полученный из репозитория TensorFlow. Сценарий retrain.py отвечает за загрузку предварительно обученной модели и, следовательно, добавление нового слоя для обучения в заданном наборе данных. Процесс переобучения занял много времени, но в конце, после анализа всех изображений, вычисления значений узких мест и подачи входных данных на конечный уровень классификации, вывод сценария показал окончательную точность теста 91,9%, как показано на рисунке 2.

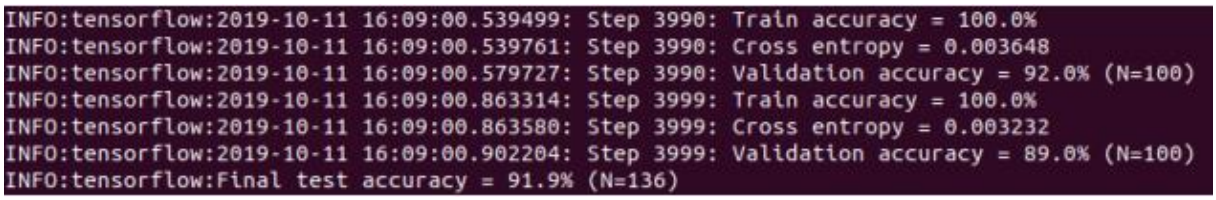

Рисунок 2. Окончательный результат работы файла retrain.py

Для каждого из этапов обучения (в данном случае 4000) случайным образом выбирается набор из 10 изображений, которые будут загружены в конечный слой для получения прогнозов, которые впоследствии сравниваются с начальными обучающими метками и, следовательно, обновляются методом

обратного распространения. Идея алгоритма обратного распространения состоит в том, чтобы, основываясь на вычислении ошибки, возникшей в выходном слое нейронной сети, пересчитать значение весов последнего слоя нейронов и, таким образом, перейти к предыдущим слоям, от начала до конца, то есть обновить все веса слоев от последнего до достижения входного слоя сети, для этого выполняя обратное распространение ошибки, полученной сетью.

Перед выполнением обучения в фоновом режиме был запущен инструмент мониторинга, включенный в TensorFlow, под названием TensorBoard. Этот процесс выполнялся параллельно с обучающим для мониторинга рядом параметров обучения.

Опираясь на TensorBoard, вовремя и после обучения можно было оценить следующие результаты:

- Точность – разделенные на точность обучения и точность проверки, эти значения представляют, соответственно, процент изображений, помеченных правильно, и точность проверки на выбранном наборе изображений. На рисунке 3 точность, представленная по оси y, является функцией прогресса обучения, представленного по оси x. Оранжевая линия представляет точность обучения модели, в то время как синяя линия показывает точность проверки. Поскольку точность проверки осталась такой же, как и точность обучения, мы можем сказать, что модель не участвовала в переобучении, что является сценарием, когда модель изучает больше правильности обучающих данных, чем сами шаблоны данных.
- Перекрестная энтропия – это положительная функция потерь, которая стремится к нулю, поскольку нейрон улучшает вычисление желаемого результата, y, для всех обучающих входов, x, как показано на рисунке 4;

После выполнения всего процесса обучения был запущен окончательный сценарий для проверки точности оценки модели, который вернул значение 91,9%, как указано выше. Это число отражает общую производительность модели в реальном сценарии классификации, и, поскольку обучение

проводилось только по 3 классам, можно было получить высокую точность.

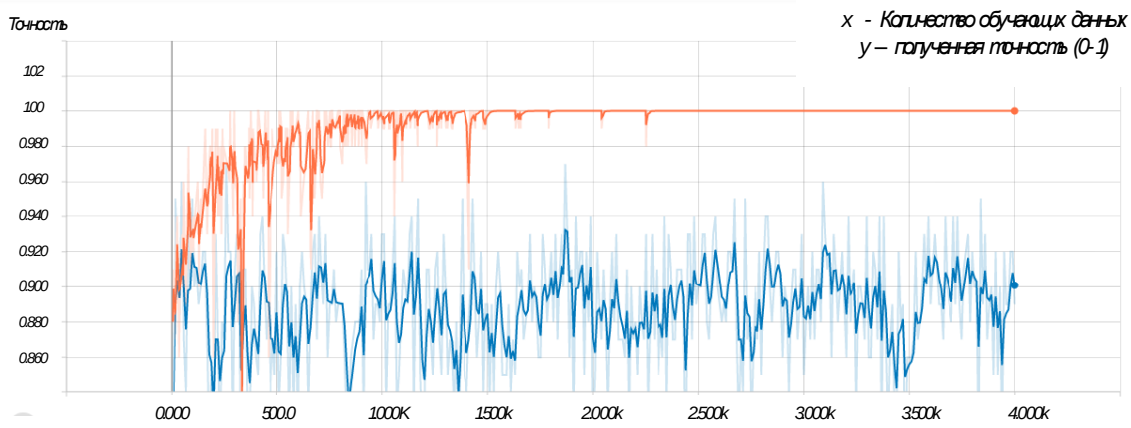

Рисунок 3. Точность обучения

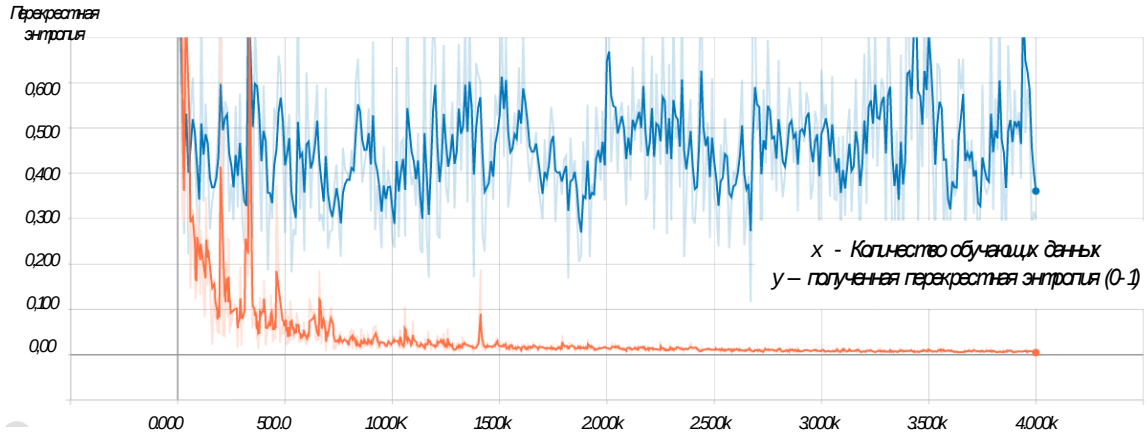

Рисунок 4. Перекрестная энтропия обучения

Описание реализованного прототипа. Разработки направлены на модельный эксперимент, в котором пользователь может управлять устройствами, существующими в его доме, с помощью пользовательского интерфейса камеры, имеющего визуальное представление действий и обеспечивающего интуитивное взаимодействие. Глядя на полученную реализацию, можно сказать, что основные визуальные результаты получены от мобильного приложения и устройства, подключенного к платформе.

Реализация привела к созданию мобильного приложения, с которым после установки на устройстве Android пользователь будет взаимодействовать. На рисунке 5 представлен главный экран приложения, содержащий

видоискатель камеры, метку, содержащую идентифицированный объект и точность оценки, а также кнопки для выполнения действий на этом устройстве при оценке с точностью выше заранее определенного порога.

На скриншоте, представленном на рисунке 6, видно, что пользователь наводил устройство на лампочку, и, поскольку результат оценки постоянно возвращает значения, близкие к 100%, представлена кнопка включения устройства, которая означает действие, которое можно предпринять в этот момент к данному объекту. При нажатии кнопки загорается лампочка. В дополнение к очевидной визуальной обратной связи с лампочкой, просматривая веб-платформу Home Assistant, можно увидеть возникновение события, как показано на рисунке 6.

Рисунок 5. Главный экран приложения

Тесты производительности разработанного приложения. Несмотря на то, что традиционно машинное обучение и нейронные сети являются концепциями, связанными с увеличением вычислительной мощности и надежностью аппаратного обеспечения, в рамках этой статьи рассматривается

повсеместность искусственного интеллекта на устройстве и его основной потенциал. Таким образом, интересным результатом для анализа является точность, полученная при использовании более легких моделей, и наличие задержек или снижение производительности при работе на менее мощных устройствах.

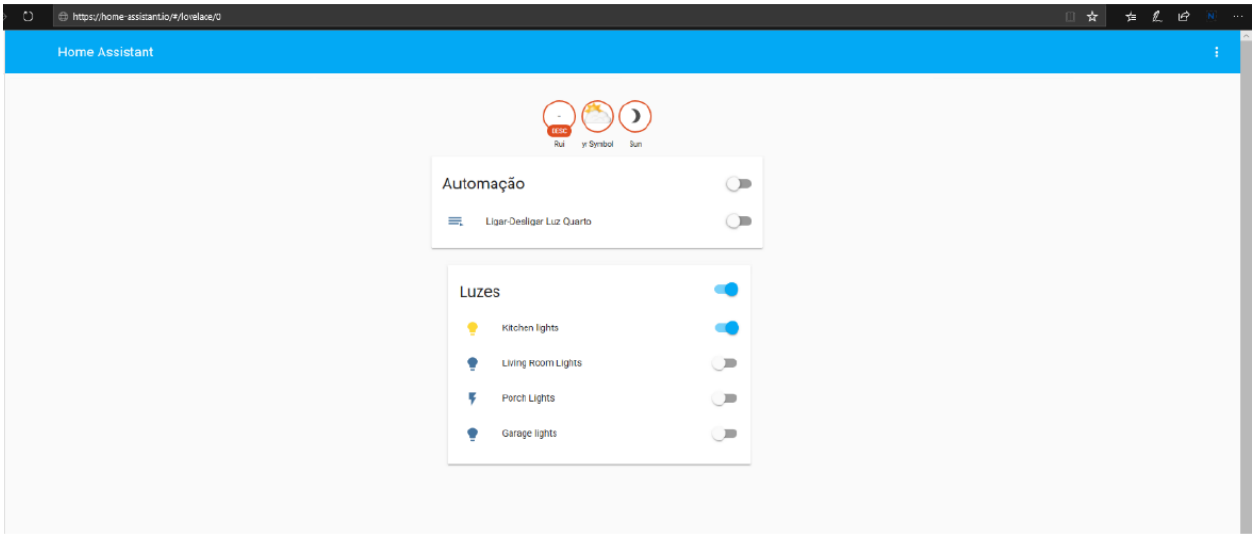

Рисунок 6. Автоматизированная платформа

Таким образом, метрикой, которая может дать показатели как преимуществ, так и недостатков этого подхода, является время, затраченное во время выполнения модели, обученной создавать выходные данные и эффективно маркировать входное изображение. Возможным способом анализа этого является запуск модели вывода на машине, где она была обучена с помощью скрипта python label_image.py показано на листинге 2, и после этого запускает ту же модель, уже интегрированную в мобильное приложение, измеряя время до и после запуска (листинг 3).

Листинг 2. Запуск скрипта label_image.py

```
ubuntu@ubuntu-virtual-machine:~/Documents/Final/tensorflow-for-poets-2$ python -m scripts.label_image \
>     --graph=tf_files/retrained_graph.pb \
>     --image-tf_files/iot_photos/bulb/bulb_123.jpg
```

Листинг 3.4. Измерение времени, затраченного на классификацию изображения

```
/** Classifies a frame from the preview stream. */
String classifyFrame (Bitmap bitmap) {
    if (tflite == null) {
        Log.e(TAG, "Image classifier has not been initialized; Skipped.");
        return "Uninitialized Classifier.";
    }
    convertBitmapToByteBuffer(bitmap);
    //Here's where the magic happens!!!
    long startTime = SystemClock.uptimeMillis();
    tflite.run(imgData, labelProbArray);
    long endTime = SystemClock.uptimeMillis();
    long timeElapsed = (endTime - startTime);
    Log.d(TAG, "Timecost to run model inference: " + Long.toString(timeElapsed)
+ " ms");

    // smooth the results
    applyFilter();

    //print the results
    String textToShow = printTopKLabels();
    return textToShow;
}
```

Данный анализ был выполнен с использованием трех разных изображений (рисунок 7) и проведен три раза на каждом из них, чтобы гарантировать минимальную погрешность результатов. В случае мобильного устройства, поскольку вход поступает в виде видеопотока, три изображения были представлены тремя разными сценариями с тремя разными лампочками. Результаты представлены в таблице 1.

Таблица 1 – Сравнение ручного вывода данных с запуском на устройстве

Время (мс)	Запуск на сервере			Запуск на устройстве		
	Запуск1	Запуск2	Запуск3	Запуск1	Запуск2	Запуск3
Изображение1	12.8	12.7	12.9	16.0	15.0	16.0
Изображение2	13.5	13.7	13.6	20.0	19.0	17.0
Изображение3	12.4	12.4	12.5	30.0	30.0	30.0

Рисунок 7 - Примеры изображений

На рисунке 7 показаны 3 примера изображений, использованных для проведения тестирования. Полученные результаты испытаний представлены в миллисекундах.

Заключение. Эта статья была посвящена демонстрации полученного окончательного прототипа, возможностей машинного обучения на устройстве и в целом результатов, полученных при эффективном тестировании надежности и согласованности разработанного решения. Анализируя полученные результаты, касающиеся сравнения, проведенного между ручным запуском модели вывода в обучающей среде и выводом, происходящим на мобильном устройстве, можно увидеть, что результаты были не такими уж отдаленными, и поэтому система может выдавать быстрые результаты в режиме реального времени без ущерба для полученной производительности.

Библиографический список:

1. Кучер М.Ю., Белов Ю.С. Подходы к распознаванию лиц и их методы // В сборнике: Технические и естественные науки. сборник избранных статей по материалам Международной научной конференции. Санкт-Петербург, 2020. С. 38-40.

2. James Le, “The 5 Computer Vision Techniques That Will Change How You See The World,” 2018, URL: https://heartbeat.fritz.ai/the-5-computervision-

techniques-that-will-change-how-you-see-the-world-1ee19334354b.

3. Tensor Flow Team, “TensorFlow Lite image classification Android example application,” 2019, URL: https://github.com/tensorflow/examples/tree/master/lite/examples/image_ classification/android.

4. Android Developer Team, “NeuralNetworks,” 2019, URL: https://developer.android.com/ndk/reference/group/neural-networks.

5. Eclipse Foundation, Inc, “Eclipse Mosquitto, An open source MQTT broker,” 2019, URL: https://mosquitto.org/.