МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

СИСТЕМА РАСПОЗНАВАНИЯ ДОРОЖНЫХ ЗНАКОВ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Алексеевой Марии Олеговны

Научный руководитель
зав. каф. техн. пр., к.ф.-м.н.,
доцент _____ И. А. Батраева

Заведующий кафедрой
к.ф.-м.н., доцент _____ С. В. Миронов

Саратов 2023

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Водитель транспортного средства во время движения может пропустить дорожные знаки, например, из-за человеческого фактора, плохих погодных условий или перекрытия обзора. Вследствие этого повышается процент дорожно-транспортных происшествий. Помимо этого количество зарегистрированных транспортных средств с каждым годом возрастает, что также влечет увеличение аварийных ситуаций.

Для снижения риска возникновения ДТП в автомобили внедряются продвинутые системы помощи водителю «Advanced Driving Assistance Systems» (ADAS), включающие систему распознавания дорожных знаков.

Ведущие автопроизводители внедряют подобные системы в свои продукты. Однако далеко не все люди могут позволить автомобили, оснащенные подобной технологией, ввиду высокой стоимости. Впрочем на настоящий момент практически не осталось людей, которые бы не имели смартфона с встроенной камерой. Следовательно, можно решить возникшую проблему, разработав мобильное приложение, способное обнаруживать и классифицировать дорожные знаки в реальном времени.

Соответственно, целью данной работы является разработка системы распознавания российских дорожных знаков в режиме реального времени, работающей на базе операционной системы Android.

Для достижения поставленной цели необходимо выполнить ряд вспомогательных задач:

- выбрать датасет дорожных знаков и выполнить его предобработку;
- выбрать подходящую модель и настроить ее конфигурацию под свои цели;
- обучить и преобразовать выбранную модель;
- разработать Android-приложение на языке программирования Java;
- встроить полученную модель в приложение;
- протестировать разработанную систему на дороге.

1 Теоретическая часть

1.1 Подходы к решению задачи обнаружения объектов

Обнаружение объектов — это задача, для решения которой необходимо обнаружить объект (определить его местоположение) и оценить его принадлежность каждому из классов.

Раннее для обнаружения объектов использовались методы, основанные на традиционных методах обнаружения. Однако в 2010 году производительность таких детекторов достигла плато. В 2012 году произошло возрождение сверточных нейронных сетей (CNN), способных извлекать высокоуровневые признаки изображения. С тех пор обнаружение объектов стало развиваться с беспрецедентной скоростью.

На данный момент выделяется два основных параллельно развивающихся подхода:

1. Двухэтапные методы (two-stage methods) или же «методы, основанные на регионах». Из названия очевидно, что подход разделяется на два этапа. Первый этап — выделение регионов интереса (RoI), областей содержащий с высокой вероятностью объекты, при помощи селективного поиска или специального слоя нейронной сети. Второй этап — определение принадлежности выбранных регионов к классам с помощью классификатора и уточнение местоположения ограничивающих рамок при помощи регрессора.
2. Одноэтапные методы (one-stage methods). Данный подход не использует специальный алгоритм для определения регионов. Вместо этого предсказываются координаты определенного количества ограничивающих рамок с разными характеристиками (метки классов, степень уверенности) и в дальнейшем корректируется местоположение рамок.

Вторую группу детекторов отличает высокая скорость работы, однако одноэтапные детекторы в основном уступают по качеству обнаружения детекторам, работающим в два этапа. Так как в системе распознавания дорожных знаков в режиме реального времени приоритетной задачей является скорость работы, то для разрабатываемого приложения было решено использовать одноэтапный детектор SSD с использованием в качестве базовой сети MobileNetV2 FPNLite из зоопарка моделей Tensorflow. Данная модель имеет лучшее соотношение между скоростью работы и точностью.

1.2 Модель SSD MobileNetV2 FPNLite

Архитектура SSD MobileNet V2 FPNLite (см. рис. 1) представляет собой комбинацию двух архитектур:

— MobileNetV2 — используется для извлечения признаков;

— Single Shot Detector (SSD) — применяется для обнаружения объектов с добавлением модуля Feature Pyramid Network Lite (FPNLite).

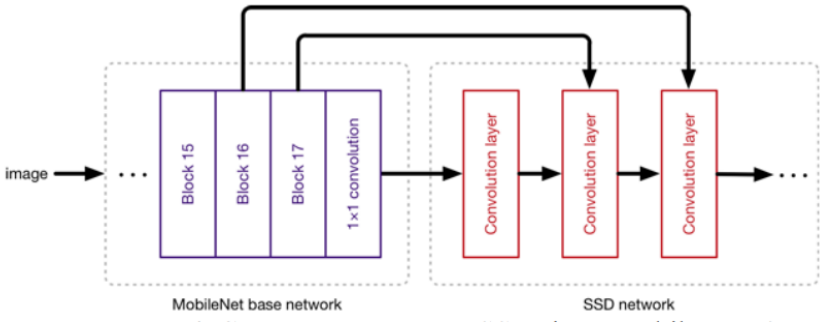

Рисунок 1 – Схема архитектуры модели SSD MobileNetV2 FPNLite

Ключевыми особенностями такой архитектуры является использование инвертированных остаточных блоков (inverted residuals) и линейных узких мест (linear bottlenecks). Помимо этого стандартная свертка заменяется факторизованной версией, уменьшающей вычислительную сложность и размер модели. Она реализуется в два этапа: сначала выполняется глубинная свертка, выполняющая легкую фильтрацию, применяя единственный сверточный фильтр для каждого входного канала; затем применяется точечная свертка размерностью 1×1, отвечающая за создание новых функций путем вычисления линейных комбинаций между входными каналами.

Архитектура модели MobileNetV2 содержит начальный полностью сверточный слой, за которым следует 17 основных блоков (bottleneck residual block). Для нелинейности используется функция активации ReLU6 из-за ее устойчивости при использовании с вычислениями с низкой точностью.

2 Практическая часть

2.1 Используемые технологии

В ходе разработки приложения было использовано 2 языка программирования: Python 3.10 для работы с данными и Java 8 для разработки Android-приложения. Для работы с нейронной сетью была использована библиотека TensorFlow, для отслеживания результатов обучения была задействована система TensorBoard, также для использования GPU во время обучения была использована Cuda 11.2.

2.2 Формирование обучающей базы

Одним из важнейших шагов на пути обучения какой-либо модели является выбор исходных данных (датасета), которые наилучшим образом опишут исследуемую область. Поэтому первоначальной задачей является выбор датасета и его предобработка.

2.2.1 Обзор датасета

В качестве исходных данных был выбран датасет российских дорожных знаков Russian traffic sign images dataset (RTSD). В этом наборе данных имеются изображения двух разрешений: 1280×720, 1920×1080. Кадры сняты в разные времена года (весна, осень, зима), разное время суток (утро, день вечер) и при разных погодных условиях (дождь, снег, яркое солнце). Данные являются размеченными, у каждого из объектов выборки указаны координаты левого верхнего угла ограничивающей рамки, а также ее высота и ширина. Всего в выборке присутствует 198 классов дорожных знаков. Общее количество размеченных объектов составляет 104358 штук.

2.2.2 Преобразование датасета

База дорожных знаков RTSD была отредактирована, для дальнейшего корректного использования при обучении нейронной сети:

1. Были удалены классы, объем которых составлял менее 100 объектов. После преобразования осталось 88 классов.
2. Так как выбранная модель нейронной сети принимает на вход изображения размером 640×640, то для того, чтобы не терять качество исходных изображений, они были преобразованы к требуемому размеру. Данное преобразование учитывало размер исходного изображения: если он был

равен 1280×720, то на выходе получалось 2 изображения (см. рис. 2) размером 640×640; если исходный размер равнялся 1920×1080, то на выходе получалось 3 изображения (см. рис. 3) размером 640×640. На новых изображениях координаты ограничивающих рамок приняли новые значения.

Рисунок 2 – Преобразование изображения, имеющего размер 1280×720

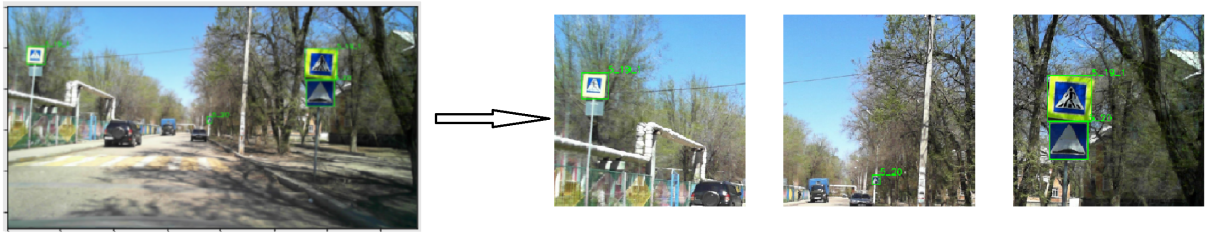

Рисунок 3 – Преобразование изображения, имеющего размер 1920×1080

3. Данные для обучения были разделены на обучающую и проверочную выборки в процентом соотношении 80% и 20% от исходных данных, соответственно.
4. Была выполнена аугментация, так как выборка является несбалансированной. Для хорошего качества работы обученной нейронной сети необходимо минимум 1000 объектов в каждом из классов. Поэтому при помощи аугментации тренировочный набор был расширен до необходимого минимального количества объектов в каждом из классов. Для проведения аугментации использовалась библиотека albumentations, из которой использовались следующие преобразования:
    a. **Поворот**. Изображение поворачивается на случайный угол, не превышающий $20°$, выбранный из равномерного распределения. Данное преобразование применяется во всех случаях.

b. **Понижение качества и добавление гауссова шума**. Качество изображения понижается, путем уменьшения и обратного увеличения масштаба. Данные два преобразования объединены в один блок, который применяется к изображению с вероятностью 90%. Если данный блок используется, то выбирается одно их двух преобразований: первое применяется с вероятностью 50%, второе — с вероятностью 30%.

c. **Изменения контраста**. Данное преобразование применяется в 70% случаев;

d. **Добавление тумана, дождя и размытия**. Размытие реализуется при помощи применения ядра случайного размера, ограниченного сверху значением 2. Перечисленные 3 преобразования объединены в блок, который применяется с вероятностью 60%. При использовании данного блока первое преобразование применяется с вероятностью 50%, второе — с вероятностью 50%, третье — с вероятностью 30%.

e. **Изменение резкости**. Резкость изображения повышается путем изменения резкости исходного изображения и наложения преобразованного изображения на исходное. Данное преобразование применяется в 80% случаев.

Если в ходе аугментации отношение площади ограничивающей рамки после преобразования к площади рамки до преобразования станет меньше 0.5, то данная рамка отбрасывается.

5. Была изменена исходная разметка изображений. Изначально изображения были аннотированы в формате $\mathrm{x\_from}$, $\mathrm{y\_from}$, width, height (координаты верхнего левого угла рамки, ее ширина и высота), однако выбранная модель SSD MobileNetV2 FPNLite требует разметку другого формата: $\mathrm{x\_min}$, $\mathrm{y\_min}$, $\mathrm{x\_max}$, $\mathrm{y\_max}$ (координаты верхнего левого и нижнего правого углоа ограничивающей рамки). Также для дальнейшей работы в датасет были добавлены размеры изображений (высота и ширина).
6. Весь набор данных (csv-файл и изображения) был переведен в формат TFRecord, собственный двоичный формат хранения Tensorflow, увеличивающий скорость обучения модели.

2.3 Обучение нейронной сети и преобразование модели

Для приложения была выбрана нейронная сеть SSD MobileNetV2 FPNLite из зоопарка моделей Tensorflow. Данная модель была предварительно обучена на наборе данных COCO 2017. SSD MobileNetV2 FPNLite имеет лучшее соотношение между скоростью работы и точностью из всех доступных в репозитории моделей.

Модель обучалась методом моментов на протяжении 200 000 шагов с размером батча 8. В качестве оптимизатора использовался градиентный спуск с моментов, приведенный в уравнении. Такой подход приводит к ускорению сходимости и сглаживанию колебаний.

В качестве функции потерь при классификации использовалась функция Focal Loss, представленная в уравнении. Она является расширением Cross-Entropy Loss, предназначенным для борьбы с проблемой дисбаланса классов в однопроходных моделях определения объектов.

Для вычисления потерь при локализации использовалась $SmoothL_1loss$, при помощи которой измерялось, насколько далеко находятся ограничительные рамки, предсказанные нейронной сетью, от истинных ограничивающих прямоугольников из обучающего набора.

Для отслеживания точности модели примерно каждые 50 000 шагов вычислялась метрика mAP@50. Значение mAP в процессе обучения росло и на 200 000 шаге достигло максимальной отметки 0,38. Данное значение превышает точность, приведенную в официальной документации, составляющую 28,2. Следовательно, для дальнейшей работы была выбрана модель, полученная на 200 000 шаге.

Далее полученная модель была преобразована формат TensorFlow Lite Model File, пригодный для интеграции в Android-приложение. Для большей оптимизации к модели была применена операция целочисленного квантования после обучения.

2.4 Реализация обнаружения дорожных знаков

Для использования обученной модели внутри приложения было использовано API интерпретатора TensorflowLite (InterpreterApi), работающее с моделями формата .tflite.

Однако на вход модели подавался не целый кадр, полученный с камеры,

а только его часть. Так как дорожные знаки стоят на правой стороне дороги и расположены довольно высоко, то было решено передавать на вход нейронной сети часть кадра: квадрат со стороной, равной 75% от высоты изображения, находящийся в верхнем правом углу кадра. Данное изменение улучшает скорость распознавания, поскольку модель обрабатывает не все изображение, а только его часть, что снижает количество обрабатываемых признаков.

Процесс обнаружения дорожных знаков может быть описан следующей последовательностью шагов:

1. Получить изображение с камеры телефона.
2. Обрезать область изображения.
3. Подать преобразованное изображение на вход нейронной сети.
4. Получить выходные значения нейронной сети: оценки принадлежности к классу, координаты ограничивающих рамок, количество обнаруженных рамок и метки классов.
5. Оставить те ограничивающие рамки, у которых уверенность в принадлежности к классу превышает значение 0,25 (значение подобрано экспериментально).
6. Отрисовать оставшиеся ограничивающие прямоугольники, указав сверху и снизу класс и оценку принадлежности к классу, соответственно.
7. Вывести распознанные знаки в соответствующую область на экране смартфона.

   Примеры обнаружения дорожных знаков представлены на рисунке 4.

Рисунок 4 – Пример работы приложения

2.5 Тестирование

Для того, чтобы оценить результат работы разработанного приложения обнаружения дорожных знаков на платформе Android, было проведено тестирование системы на дорогах города Саратова. Приложение тестировалось в светлое и сумеречное время суток на телефоне Samsung Galaxy A50 (SM-A505FZKUSER) 2019 года. Были подсчитаны следующие данные:

— количество обнаруженных знаков;
— число ненайденных дорожных знаков;
— количество ложных обнаружений;
— количество неверно классифицированных объектов;
— количество верно классифицированных знаков.

Всего было протестировано 205 дорожных знаков. Выбирались только те знаки, экземпляры которых присутствовали в обучающей выборке. Подсчет метрик осуществлялся по формулам 1 и 2.

$$\text{точность локализации} = \frac{\text{кол-во локализованных}}{\text{общее число знаков}} \tag{1}$$

$$\text{точность классификации} = \frac{\text{кол-во верно классифицированных}}{\text{кол-во локализованных}} \tag{2}$$

В итоге, точность локализации составила 91,2%, точность классификации 95,7%; не было обнаружено 18 объектов; ложное обнаружение произошло 8 раз; неверно был определен класс 8-ми объектов.

Среднее время, затрачиваемое на обработку одного кадра на смартфоне Samsung Galaxy A50 (SM-A505FZKUSER), составило 500мс, что примерно соответствует частоте обновления кадров 2 FPS.

При этом нагрузка на телефон субъективно ощущалась средней. Приложение загружало GPU смартфона в среднем на 25%, CPU — на 35% при условии, что все фоновые приложения были закрыты. Данные показатели сравнивались со значениями, полученными во время работы игры «Subway Surfers» со средней графикой. Игра загружала GPU смартфона в среднем на 60%, CPU — на 45%. Опираясь на полученные значения, можно сделать вывод о том, что разработанное приложение загружает GPU в среднем на 35% меньше, CPU — на 10% по сравнению со средней мобильной игрой.

Также был исследован расход заряда батареи. Замер осуществлялся 3 раза по 25 минут: приложение, обнаруживающее дорожные знаки, работало на протяжении 25-ти минут при условии, что все фоновые приложения были отключены. Аналогичный замер расхода батареи был проведен при работе мобильной игры «Subway Surfers», а также только для включенного экрана. Процент расхода заряда батареи при работе приложения и при работе мобильной игры практически совпадает: для первого случая значение расхода батареи составило 10%, для второго — 7%. Если сравнивать полученный показатель, равный 10%, со значением расхода батареи телефона только при включенном экране, то мобильное приложение потребляет заряд в 2,5 раза больше.

ЗАКЛЮЧЕНИЕ

В ходе исследования была достигнута поставленная цель: разработана система автоматического распознавания российских дорожных знаков, работающая на базе операционной системы Android.

В данной работе были достигнуты следующие результаты:

— Для обнаружения дорожных знаков была обучена нейронная сеть SSD MobileNetV2 FPNLite. Средняя время, затрачиваемое на обработку одного кадра, с соотношением сторон 1 к 1 и размером стороны равной 75% от высоты изображения, на смартфоне Samsung Galaxy A50 (SM-A505FZKUSER) составляет 500 мс (2 FPS). Беря во внимание тот факт, что вычисления происходили на мобильной платформе, указанное время является удовлетворительным.

— При тестировании приложения на автомобильных дорогах города Саратова такое время распознавания не доставило дискомфорта: объекты достаточно хорошо локализовались и классифицировались; кадры, получаемые с камеры смартфона, были без искажений. В результате тестирования были получены следующие результаты: точность локализации составила 91,2%, точность классификации — 95,7%. Так как разработанная система работает в режиме реального времени, то если объект не будет обнаружен на текущем кадре, то он может быть найден на следующем, поэтому полученная точность удовлетворяет потребностям приложения. Это объясняется следующими причинами:

  a. Камера. У смартфонов есть недостатки при съемке в движении: пропадает автофокус, вследствие чего изображение может размываться.
  b. Окружающая обстановка. Некоторые окружающие объекты могут иметь визуальную схожесть с дорожными знаками, но ими не являться. Возможным решением данной проблемы может быть смена размеров окна поиска или изменение порогового значения, отвечающего за вывод распознанного объекта.

  Возможными способами увеличения пропускной способности сети является, например: обучение модели, с учетом квантования; изменение входного размера модели на меньший, например, на 320×320. Перечисленные способы могут существенно уменьшить скорость обработки кадра и ускорить процесс обнаружения.