Санкт-Петербургский государственный университет

***Уткин Илья Николаевич***

Выпускная квалификационная работа

# Деревья решений для настройки параметров алгоритмов ADAS

Уровень образования: бакалавриат

Направление *02.03.03 «Математическое обеспечение и администрирование информационных систем»*

Основная образовательная программа *СВ.5006.2019 «Математическое обеспечение и администрирование информационных систем»*

Научный руководитель:
доцент кафедры СП, к.т.н. Ю. В. Литвинов

Консультант:
инженер-программист АО “Кама” М. С. Осечкина

Рецензент:
технический руководитель проектов К. В. Свитков

Санкт-Петербург
2023

Saint Petersburg State University

***Ilia Utkin***

Bachelor's Thesis

# ADAS algorithms hyperparameters tuning with decision trees

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Programme *CB.5006.2019 "Software and Administration of Information Systems"*

Scientific supervisor:
C.Sc., docent Y.V. Litvinov

Consultant:
Software engineer at JSC "KAMA" M.S. Osechkina

Reviewer:
Technical project leader K.V. Svitkov

Saint Petersburg
2023

# Оглавление

**Введение** **4**

**1. Постановка задачи** **6**

**2. Обзор** **7**

2.1. Методы для подбора гиперпараметров . . . . . 7

2.2. Дерево решений . . . . . 9

2.3. Задача обнаружения контуров на дороге . . . . . 11

2.4. Задача обнаружения дорожной полосы . . . . . 12

**3. Обучение дерева решений** **14**

3.1. Генерация признаков . . . . . 14

3.2. Обучение на основе классических методов . . . . . 15

3.3. Дистилляция нейросетевых методов . . . . . 16

3.4. Реализация . . . . . 20

**4. Эксперимент** **25**

4.1. Постановка эксперимента . . . . . 25

4.2. Качество . . . . . 28

**Заключение** **31**

**Список литературы** **32**

# Введение

Безопасность на дороге является одним из главных приоритетов для участников дорожного движения. Несмотря на понятность этой проблемы и различные меры по регуляции движения, в России за 2021 год произошло 133 тысячи автомобильных аварий с пострадавшими [24]. Чаще всего такие происшествия случаются не по вине производителей транспортных средств, а из-за человеческих ошибок во время вождения. Для борьбы с этой проблемой изучается возможность как можно больше автоматизировать процесс вождения, чтобы минимизировать человеческий фактор.

Усовершенствованная система помощи водителю (Advanced Driver-Assistance System, ADAS) — класс систем дополненной реальности для помощи водителю на дороге. ADAS использует различные сенсоры для моделирования окружающего мира, в частности, камеры и лидары, чтобы избежать столкновений, предупреждать водителя об опасностях, помогать оставаться на дороге или даже взять контроль над транспортным средством при необходимости.

Для моделирования окружающей среды ADAS использует различные алгоритмы компьютерного зрения, такие как детектирование линий или сегментирование точек лидаров на объекты. Часто такие алгоритмы имеют глубокое пространство для настройки. Так, например, алгоритм Canny [3], позволяющий находить контуры объектов на изображении, имеет следующие параметры: два пороговых значения для удаления шумных линий, и степень сглаживания. Такие параметры называются гиперпараметрами алгоритма, так как задаются пользователем, а не рассчитываются самим алгоритмом.

В зависимости от значений этих гиперпараметров поведение алгоритма может сильно меняться. По этой причине в АО «Кама» есть запрос на подход, с помощью которого можно будет автоматически по характеристикам изображения (например по степени освещенности или гистограмме серого) определять оптимальные гиперпараметры алгоритмов.

Стоит отметить, что системы для подбора гиперпараметров уже давно существуют. К ним относятся как простейший перебор пространства гиперпараметров, так и более сложные алгоритмы, такие как Байесовская оптимизация [7], оперирующая вероятностными функциями, Multi-fidelity Optimization [5], нацеленная на эффективность подбора, и, наконец, сложные нейронные сети, основанные на мета-обучении [6] и обучении с подкреплением [22], анализирующие обратную связь от метрик и нормы градиентов. Однако подобные системы позволяют подобрать лишь один набор гиперпараметров под большинство ситуаций. Часто этого недостаточно и один и тот же набор гиперпараметров может выигрывать в одних ситуациях, но проигрывать в других. В таком случае можно рассмотреть экспериментальные методы подбора [12], принимающие во внимание контекст задачи, однако они довольно сложны, нестабильны и имеют плохую производительность.

В связи с этим в качестве основы для разрабатываемого подхода заказчиком было выдвинуто требование исследовать дерево решений [16]. Данная модель может быть полностью проанализирована разработчиком (что считается ценным качеством в области технологий ADAS), а также имеет высокую скорость работы и хорошее качество предсказаний, несмотря на свою простоту.

Таким образом, в рамках данной работы предполагается исследовать возможность применения модели дерева решений к задаче подбора гиперпараметров алгоритмов ADAS. Предлагается разработать некоторый метод обучения дерева решений на задаче подбора гиперпараметров, реализовать разработанный метод и сравнить с существующими аналогами.

# 1. Постановка задачи

Целью данной работы является исследование применимости деревьев решений к задаче подбора гиперпараметров к алгоритмам ADAS.

В связи с этим поставлены следующие задачи.

1. Провести обзор существующих подходов к подбору гиперпараметров, выполнить обзор датасетов и метрик для задач ADAS.

2. Разработать и реализовать метод для обучения дерева решений на задаче подбора гиперпараметров.

3. Сравнить созданный метод с существующими подходами к подбору гиперпараметров на задачах ADAS.

# 2. Обзор

## 2.1. Методы для подбора гиперпараметров

Как уже было сказано ранее, проблема нахождения оптимальных гиперпараметров далеко не нова. Ниже будет приведен краткий обзор существующих решений, будут описаны их преимущества и недостатки в контексте данной работы.

### 2.1.1. Перебор

Одними из самых примитивных методов подбора гиперпараметров являются сеточный метод и метод случайного поиска, по сути являющиеся перебором подмножества точек из пространства гиперпараметров.

В случае **сеточного метода** каждому гиперпараметру сопоставляется некоторый фиксированный набор значений. Затем составляются все возможные комбинации элементов из этих наборов (сетка), модель обучается на каждой комбинации (или происходит запуск некоторого алгоритма), собираются значения метрик и выбирается наилучшая комбинация. **Случайный метод** использует похожий подход, однако не требует от пользователя наборы значений, а выбирает их сам из пространства гиперпараметров случайным образом.

Несмотря на свою простоту, эти методы до сих пор используются, если пространство гиперпараметров не слишком велико. Однако, если комбинаций много (например в случае непрерывных гиперпараметров) и проверка каждой комбинации требует большого количества времени, то избыточность вычислений заставляет обратиться к более продвинутым методам.

### 2.1.2. Перебор с учетом предыдущих попыток

При оптимизации некоторых гиперпараметров можно сделать предположение, что оценка модели будет зависеть от них непрерывно. То

есть, если качество модели при каких-то гиперпараметрах будет низким, то и в некоторой окрестности этих гиперпараметров качество также будет низким. Учитывая это предположение, можно итеративно подбирать точки в пространстве гиперпараметров, оценивать их, и использовать те точки, которые теоретически имеют наибольшую оценку.

Так, например, устроен алгоритм **Байесовской оптимизации** [7]. С помощью специальной функции приобретения, учитывающей все предыдущие измерения, находится точка, которая в теории даст наилучший результат. Данный процесс повторяется фиксированное количество раз, и затем из всей истории выбирается самая оптимальная точка.

В **эволюционной оптимизации** [18] итерации происходят по популяциям. Индивиды текущей популяции смешиваются и мутируют, а затем случайно отбираются в следующую популяцию, где с большей вероятностью пройдут гиперпараметры с наивысшей оценкой. Таким образом средняя оценка популяции со временем улучшается. Данный способ подходит для случаев, когда подсчет метрик занимает сравнительно небольшое количество времени, но пространство гиперпараметров очень велико.

Также стоит упомянуть про метод Multi-fidelity optimization (MFO) — подход к оптимизации, который использует несколько уровней разрешения модели для улучшения процесса оптимизации. Он позволяет быстрее находить оптимальные параметры, используя более грубые модели на начальных этапах оптимизации, а затем переходить к более точным моделям на более поздних этапах. Примером Mfo алгоритма являются Successive Halving и его модификация HyperBand [9].

### 2.1.3. Подбор гиперпараметров с помощью машинного обучения с подкреплением

Чаще всего задача подбора гиперпараметров поставлена как оптимизация некоторой «черной коробки». Однако в последнее время стало популярным использование машинного обучение с подкреплением, благодаря которому можно ускорить процесс оптимизации и обобщить его

на несколько задач.

Машинное обучение с подкреплением — это один из способов машинного обучения, в ходе которого испытуемая система (агент) обучается, взаимодействуя с некоторой средой. Агент совершает некоторое действие, получает награду и обновляет свое поведение. В рамках данной работы фокус будет на алгоритме REINFORCE [21] и его модификации Actor-Critic [19]. Основным отличием от классического машинного обучения с учителем в функции потерь, которую модель пытается минимизировать:

$$L = -log(p_i) \cdot R$$

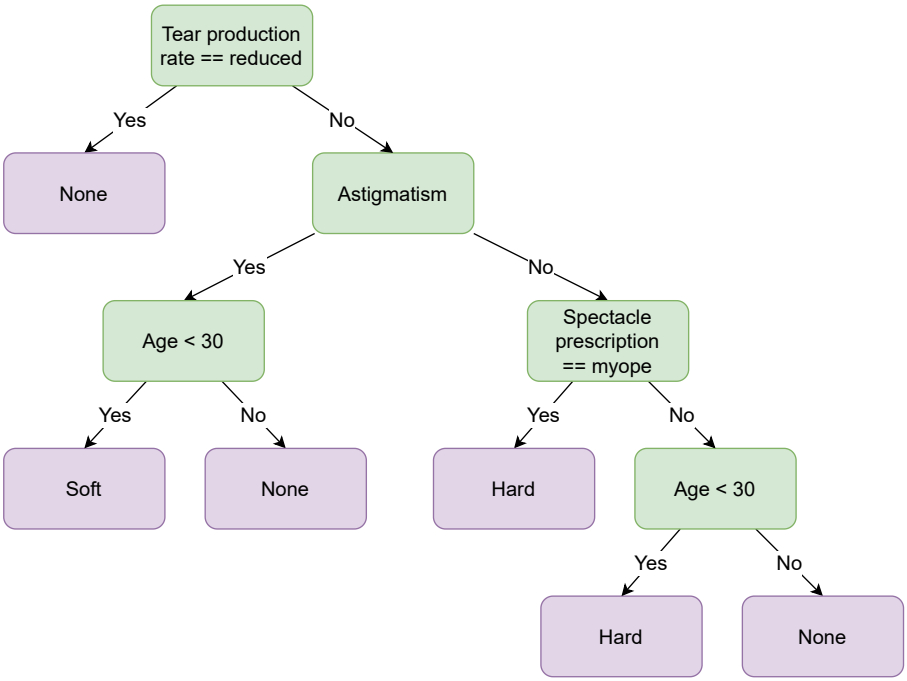
где $p_i$ — вероятность выполненного агентом действия (выход модели), а $R$ — награда. Таким образом модель наказывают за все действия, но чем больше вероятность P при больших R, тем меньше ошибка.

## 2.2. Дерево решений

Дерево решений — средство поддержки принятия решений, использующееся в машинном обучении, анализе данных и статистике, чаще всего для задачи классификации (рис. 1). Оно состоит из ребер и двух типов вершин: терминальных и нетерминальных. Нетерминальные вершины представляют собой некоторый предикат, использующий один из атрибутов входного вектора, ребра представляют значение предиката (Истина/Ложь), а терминальные вершины представляют метку некоторого класса. Когда дереву на вход подается некоторый вектор, происходит рекурсивный запуск от корня дерева. Вычисляется предикат, и рекурсивный алгоритм запускается от левого или правого поддерева в зависимости от значения предиката. Если алгоритм запускается от терминальной вершины, то возвращается метка класса, хранящаяся в этой вершине.

Обучение дерева происходит рекурсивно по методу минимизации энтропии в каждом узле или минимизации коэффициента Джини [20]. Аналогично тестированию во время обучения запускается рекурсивный

Рис. 1: Пример дерева решений для определения типа линз, которые носит пациент: мягкие, жесткие, или же пациент не носит линзы совсем.




алгоритм. На каждом шаге выборка разбивается по значению некоторого предиката так, чтобы минимизировать энтропию в каждом из разбиений. После этого алгоритм рекурсивно запускается для вычисления левого и правого поддерева. Если размер выборки достиг некоторого порогового значения или невозможно выбрать хорошее разбиение, то вершина объявляется терминальной. Меткой вершины в таком случае будет самый часто встречающийся класс.

Также дерево решений можно применять для задачи регрессии. Для этого достаточно во время разбиений уменьшать стандартное отклонение, а терминальной вершине присваивать среднее значение попавшей в неё выборки.

Для регуляризации дерева решений применяют следующие параметры: максимальная глубина дерева, минимальное количество элементов в терминальных вершинах и минимальное уменьшение значения энтропии при разбиении. Данные параметры применяются для борьбы с переобучением, к которому дерево решений очень склонно. В рамках данной работы параметры для дерева решений будут находиться с помощью метода сетки, который был описан выше в секции 2.1.1. Про-

странство гиперпараметров дерева решений невелико (в рамках данной работы всего 20 конфигураций), поэтому выбор такого простого метода оправдан.

Основной проблемой для данной работы являются требования на данные в случае классического алгоритма дерева решений: на начало обучения должны быть доступны все наборы признаков и правильные метки классов (или значения предсказываемой функции). Другими словами: в классической постановке алгоритма обучение возможно только по «offline»-методу. Решение данной проблемы составляет основную сложность данной работы.

## 2.3. Задача обнаружения контуров на дороге

Рассмотрим некоторые задачи ADAS, на которых будет проводиться апробация описанного метода.

Задача выделения контуров объектов на дороге является одной из ключевых в компьютерном зрении и автоматическом управлении транспортом. Она заключается в определении границ объектов на дороге, таких как автомобили, пешеходы, знаки и другие препятствия.

В рамках данной работы эта задача будет решаться с помощью алгоритма Canny [3].

### 2.3.1. Алгоритм Canny

Алгоритм Canny [3] — популярный метод обнаружения контуров на изображении, основанный на градиентах изображения. Под градиентом подразумевается перепад в яркости изображения.

Алгоритм Canny имеет три гиперпараметра: степень размытия изображения перед применением алгоритма и два пороговых значения, определяющих чувствительность алгоритма к градиенту.

## 2.4. Задача обнаружения дорожной полосы

Частной задачей выделения контуров на дороге является задача обнаружения границ дорожной полосы. В рамках данной работы эта задача будет решаться с помощью алгоритма кластеризации углов градиента и с помощью преобразования Хафа[10]

### 2.4.1. Алгоритм кластеризации углов градиентов

Одной из основных проблем алгоритма Canny является его чувствительность к неоднородным поверхностям, которые распознаются как множество хаотичных контуров. Чтобы избавиться от линий, которые имеют большое количество углов и поворотов, можно воспользоваться следующим алгоритмом кластеризации углов градиентов:

1. Удалить шум и найти вертикальный и горизонтальный градиент.
2. Подсчитать угол направления градиента для каждого пикселя.
3. Найти пиксели, значения норм градиентов которых меньше некоторого порогового значения.
4. Для каждого пикселя с достаточной нормой градиента по углу градиента определить его кластер. Для этого промежуток от -$\pi$ до $\pi$ делится на несколько подпромежутков и алгоритм проверяет в каком подпромежутке лежит угол градиента.
5. Для каждого кластера найти все его границы (кластер может состоять из нескольких непересекающихся областей, так как кластеризация идет по углу градиента, а не по позиции на изображении). Отфильтровать короткие и длинные границы.

Алгоритм кластеризации углов имеет больше степеней настройки: выбор пороговых значений для каждого кластера, выбор максимальных и минимальных длин выделенных линий и выбор порогового значения для фильтрации по абсолютному значению градиента.

### 2.4.2. Преобразование Хафа

Преобразование Хафа [10] — алгоритм для параметрической идентификации геометрических элементов на изображении. Вообще преобразование Хафа может идентифицировать произвольные фигуры, однако в рамках данной работы будут идентифицироваться только прямые.

Алгоритм Хафа имеет следующие гиперпараметры: гиперпараметры алгоритма Canny (Canny используется для предварительной обработки), минимальная длина линий, максимальный размер между сегментами линий, разрешение сетки и пороговое значение для голосования (что можно интерпретировать как чувствительность алгоритма к результату алгоритма Canny).

# 3. Обучение дерева решений

В данной секции будет описан теоретический подход к обучению дерева решений задаче подбора гиперпараметров, а также будет описана его реализация на языке Python.

## 3.1. Генерация признаков

Так как дерево решений — достаточно простая модель, то необходимо перевести исходное изображение в вектор признаков, который будет меньше по размеру и более понятен для модели. В данной секции будет приведен обзор используемых техник для генерации признаков.

### 3.1.1. Степень сжатия

В данном методе используется степень сжатия нескольких областей изображения в JPEG.

Картинка разбивается на NxN секторов. Сжатие проводится по всем секторам и по изображению в целом, а затем коэффициенты сжатия записываются в результирующий вектор.

### 3.1.2. Гистограмма серого

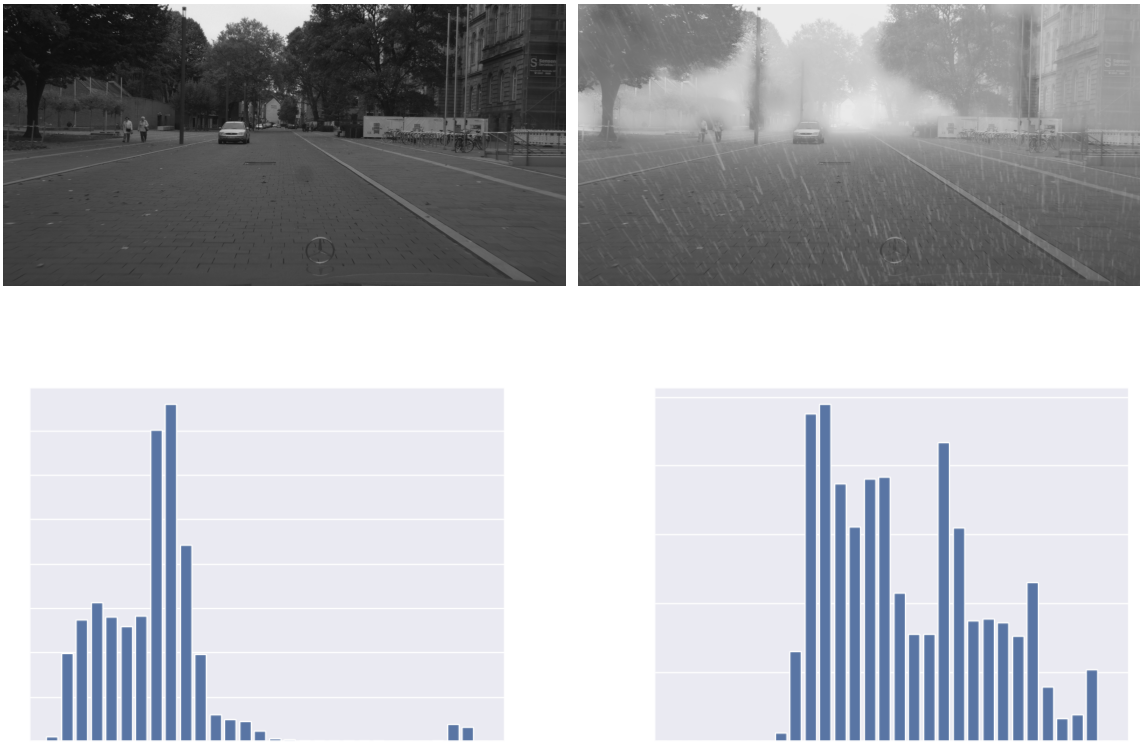
Было замечено, что в сценах с туманом цветовой охват изображения сильно меняется в сравнении с изображениями без тумана [23]. Аналогичные изменения можно также заметить в заснеженных сценах и сценах с дождем (рисунок 2). Таким образом, используя гистограмму серого, модель может получить информацию о погодных условиях.

Для получения этого признака картинка переводится в серый, собирается гистограмма для 32 диапазонов и нормализуется относительно разрешения картинки. Загрубление шкалы происходит для минимизации вероятности переобучения, что критично для дерева решений.

Рис. 2: Примеры гистограмы серого

(a) Оригинальное изображение

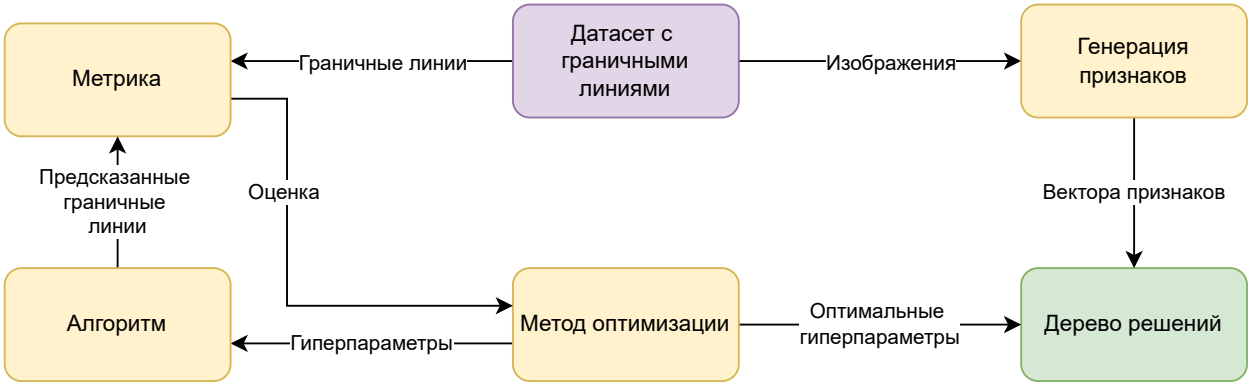
(b) Аугментированное туманом избражение



## 3.2. Обучение на основе классических методов

Как уже было сказано ранее, дерево решений предполагает, что на начало обучения есть корпус данных из векторов признаков в целевые классы. В рассматриваемой задаче (подбор оптимальных гиперпараметров в зависимости от характеристик изображений) найти такой корпус данных практически невозможно. Однако можно поставить следующую задачу оптимизации: для каждого изображения надо найти такой набор гиперпараметров, с которым алгоритм выдаст наиболее правдоподобный по некоторой метрике ответ. В такой постановке задачи (Рис. 3) больше не нужен набор оптимальных гиперпараметров, а только набор признаков изображения, правильно подобранные результаты алгоритма, метрика для оценки и метод для оптимизации.

Рис. 3: Пример обучения дерева на задаче поиска граничных линий. Метод оптимизации пытается подобрать гиперпараметры, максимизирующие некоторую оценку от метрики, которая оценивает результат работы алгоритма. Затем дерево решений обучается на оптимальных гиперпараметрах.




## 3.3. Дистилляция нейросетевых методов

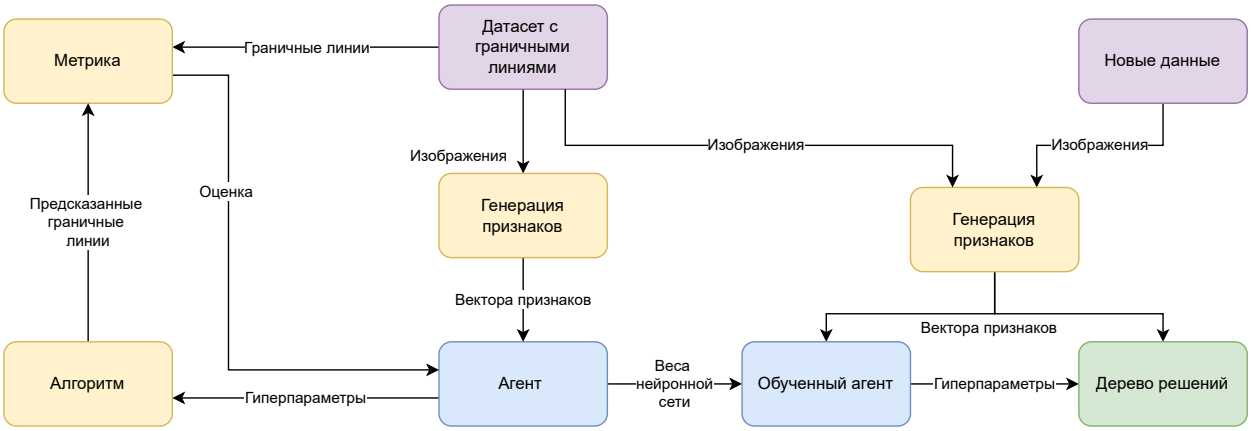
Как можно заметить из предыдущей секции, классические методы не учитывают контекст изображения. Это приводит к ситуациям, когда в обучающей выборке для дерева решений две похожие по признакам сцены имеют оптимальные, но совершенно разные гиперпараметры (в случае если метрика имеет несколько локальных максимумов). В таком случае данные имеют очень прерывистую зависимость от признаков, что может помешать обучению.

В дистилляции нейросетевых подходов предполагается создание некоторого агента (например, нейронной сети) с использованием обучения с подкреплением, а затем последующей имитации этого агента с помощью дерева решений (рис. 4). Имитация будет заключаться в разметке нового датасета и последующем обучении дерева.

Основными преимуществами данного метода являются: более непрерывная зависимость гиперпаметров от признаков в данных для обучения (более качественные данные), а также возможность разметки большого корпуса новых произвольных данных.

Основным недостатком является нестабильность методов машинного обучения с подкреплением по сравнению с классическими методами оптимизации.

Рис. 4: Описание разметки данных с обучением. Слева агент обучается на основе вектора признаков подобрать такие гиперпараметры, чтобы максимизировать некоторую метрику. Агент знает о контексте задачи, поэтому может разметить данные более корректно. Затем дерево решений учится имитировать агента. Во время имитирования можно использовать новые данные.



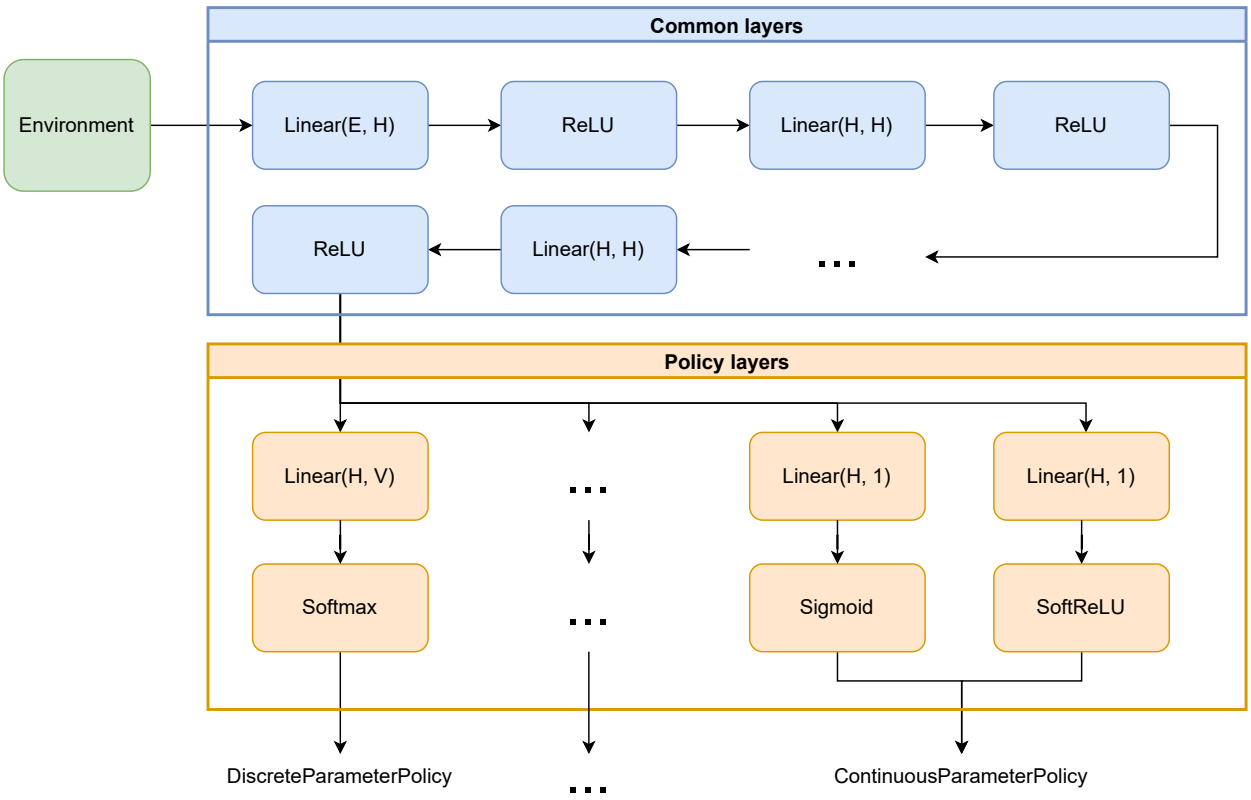

### 3.3.1. Архитектура модели

Из-за непрерывности входов и выходов алгоритма (вектор признаков и гиперпараметры) в рамках данной работы будут использоваться алгоритм REINFORCE[21] и его модификация Actor-Critic method[11].

Чтобы архитектура агента могла обобщаться под любое количество и разные типы гиперпараметров (дискретные и непрерывные), было принято решение сделать несколько общих линейных слоев и по одному или два выходных слоя на каждый гиперпараметр в зависимости от его типа (рис. 5).

Для дискретных гиперпараметров используется один выходной линейный слой и функция активации softmax. Для непрерывных гиперпараметров используется два слоя: для математического ожидания гиперпараметра и его стандартного отклонения. В качестве функций активации для математического ожидания используется сигмоида, а для стандартного отклонения softplus.

Рис. 5: Архитектура агента. Вектор среды проходит через общие слои, а затем преобразуется в политику для каждого гиперпараметра

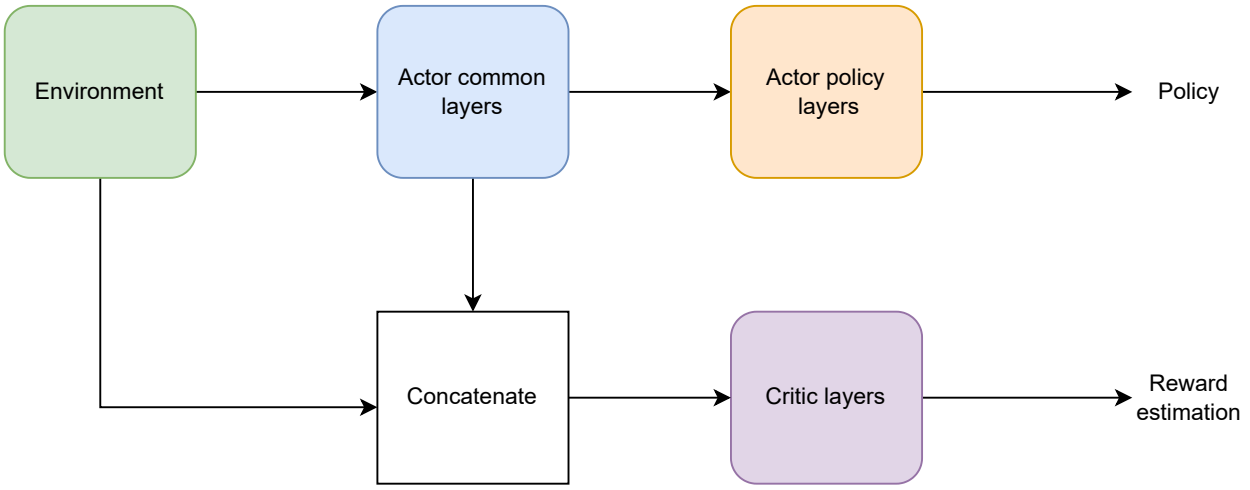



### 3.3.2. Модель-критик

Часто для стабилизации обучения используют дополнительную сущность — критика. Критик пытается предсказать значение награды на основе состояния или состояния и решения агента (рис. 6). Затем из реального значения награды вычитается приближенное значение критика и получается значения «преимущества», которым оценивается агент. Данная методика позволяет нормализовать значение награды, а также наказать модель за стагнацию и сильнее поощрять в случае нахождения нового качественного решения.

Архитектура критика тривиальна: несколько линейных слоев. В рамках данной работы критик принимает вектор признаков и вектор из последнего общего слоя агента, а выходом критика является единственное число — предсказанное значение метрики.

Рис. 6: Прямой проход во время обучения с критиком.




### 3.3.3. Entropy bonus

Чтобы понять одну из важных проблем в машинном обучении с подкреплением, вспомним формулу для функции потерь, приведенную ранее:

$$L = -log(p_i) \cdot R$$

где $p_i$ — вероятность совершенного агентом действия, а $R$ — награда. Можно заметить, что если агент полностью уверен в своем действии (вероятность $p_i$ равна единице или очень близка к ней), то значение функции потерь обращается в ноль, а значит и производная, необходимая для обучения, тоже обращается в ноль. При этом не важно в каком конкретно действии уверена модель. Другими словами, модель перестает обучаться, так как сильно эксплуатирует текущее решение, и не пытается подобрать новое.

Для борьбы с этим в рамках данной работы используется «entropy bonus», призванный наказывать модель за излишнюю уверенность:

$$L = -log(p_i) \cdot R - \alpha \cdot Entropy(P)$$

, где $\alpha$ — некоторый коэффициент меньше 1. Таким образом модель поощряется за равномерное распределение вероятностей, и наказывается в противном случае.

## 3.4. Реализация

Как уже было сказано ранее, алгоритм реализован на языке Python. Помимо этого были использованы библиотеки для машинного обучения: Pytorch [15] и Scikit-Learn [17].

Реализацию подхода можно разделить на три части:

- Модуль для подбора гиперпараметров, абстрактный от предметной области.
- Модуль для задач ADAS, включающий обертки над популярными датасетами и метриками.
- Модуль для возможности подбора гиперпараметров в алгоритмах на C++.

Далее будет кратко описана архитектура и функциональность каждого из модулей.

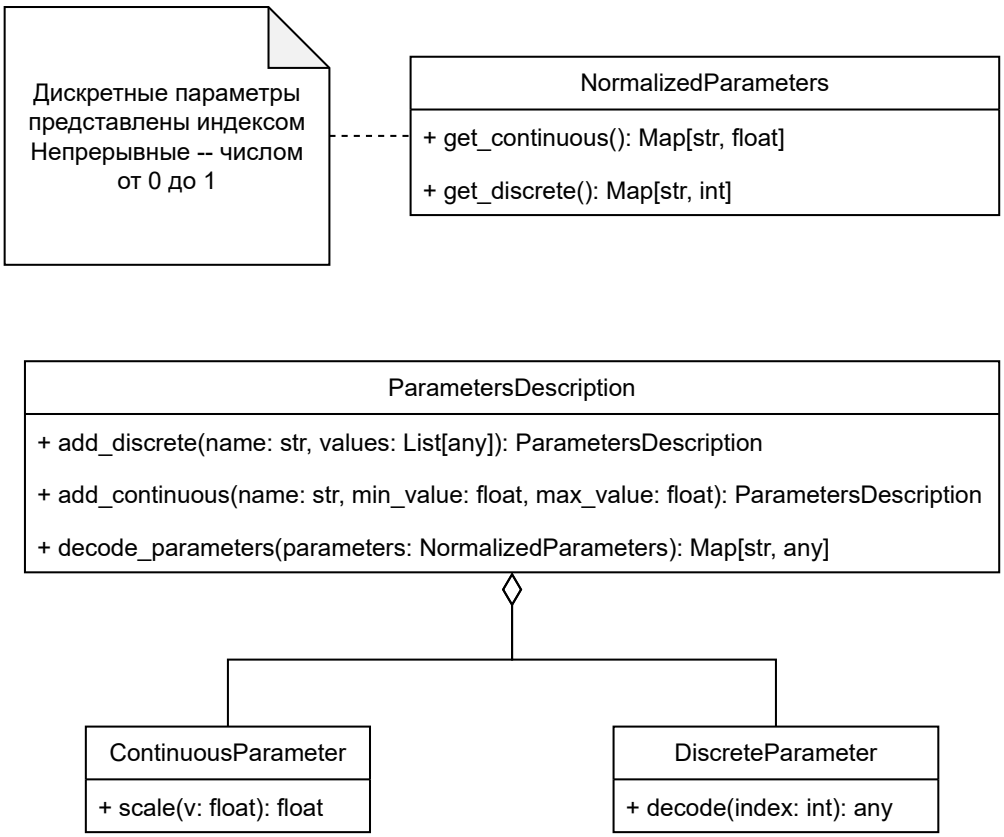
### 3.4.1. Модуль для подбора гиперпараметров

Данный модуль содержит реализацию вышеописанного метода.

Чтобы подобрать гиперпараметры для произвольного алгоритма с помощью этого модуля, для начала пользователь должен декларативно описать, какие гиперпараметры он хочет подобрать: их имена, тип, граничные значения. Как уже было сказано ранее, система поддерживает два типа гиперпараметров: дискретные (с конечным количеством значений) и непрерывные. Классы для описания гиперпараметров можно увидеть на рис. 7.

Стоит отметить, что несмотря на то, что пользователь может ввести для дискретных гиперпараметров любые возможные значения (числа, строки, произвольные объекты), а для непрерывных гиперпараметров значения из любого диапазона, все методы оптимизации видят только нормализованные гиперпараметры, где дискретные гиперпараметры приводятся к некоторому индексу, а непрерывные нормализуются к значению между 0 и 1.

Рис. 7

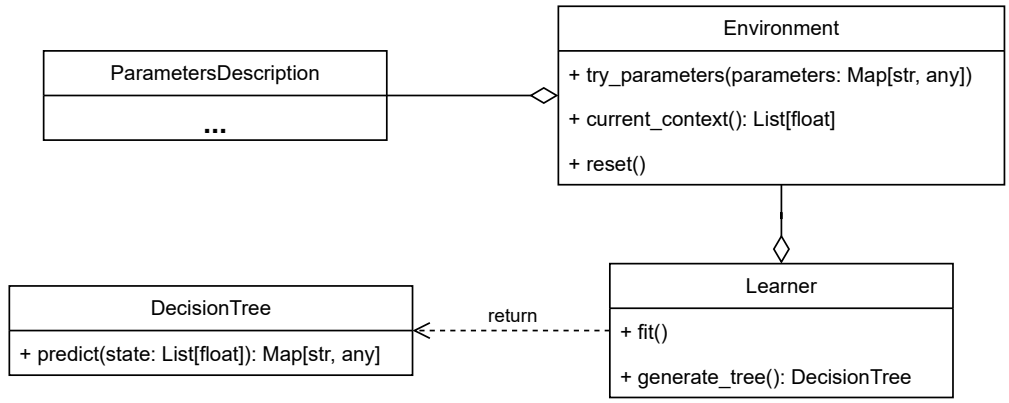



После декларативного описания гиперпараметров алгоритма, пользователь императивно описывает в классе `Environment`, как оценивать гиперпараметры и как считать вектор контекста, на основании которого дерево будет принимать решение (рис. 8).

На данный момент существуют следующие реализации `Learner`:

- `HyperOptLearner` — оболочка над библиотекой HyperOpt. Позволяет обучить дерево, используя все алгоритмы, реализованные в библиотеке HyperOpt, такие как: случайный поиск, Tree of Parzen Estimators [1], Adaptive TPE [14].
- `GeneticLearner` — обучение дерева на основе генетической оптимизации.
- `HyperBandLearner` — обучения дерева на основе multi-fidelity оптимизации, а именно с помощью алгоритма HyperBand [9]
- `ReinforceLearner` — обучение путем дистилляции нейросети. До-

Рис. 8: Основные классы с которыми взаимодействует пользователь. Параметры и процесс оценивания описывается в ParametersDescription и Environment. Затем Environment передается в Learner и создается экземпляр DecisionTree.




ступен как классический алгоритм REINFORCE [21], так и его модификация Actor-Critic method [11].

Несмотря на то, что инструмент написан в первую очередь для подбора гиперпараметров с помощью деревьев решений, пользователь может напрямую использовать внутренние оптимизации. Это позволяет при необходимости полностью отойти от деревьев решений и использовать популярные методы оптимизации с теми же интерфейсами `Environment` и `ParametersDescription`.

### 3.4.2. Утилиты для ADAS

Так как функциональность для задач ADAS оказалась достаточно отчуждаемой, было принято решение выделить ее в обособленный модуль.

В данном модуле реализованы метрики для рассматриваемых в данной работе задач ADAS, унифицированные оболочки над датасетами, алгоритмы, а также генерация векторов признаков, описанная раннее.

Помимо этого также были добавлены реализации интерфейса `Environment` для каждого из рассмотренных алгоритмов.

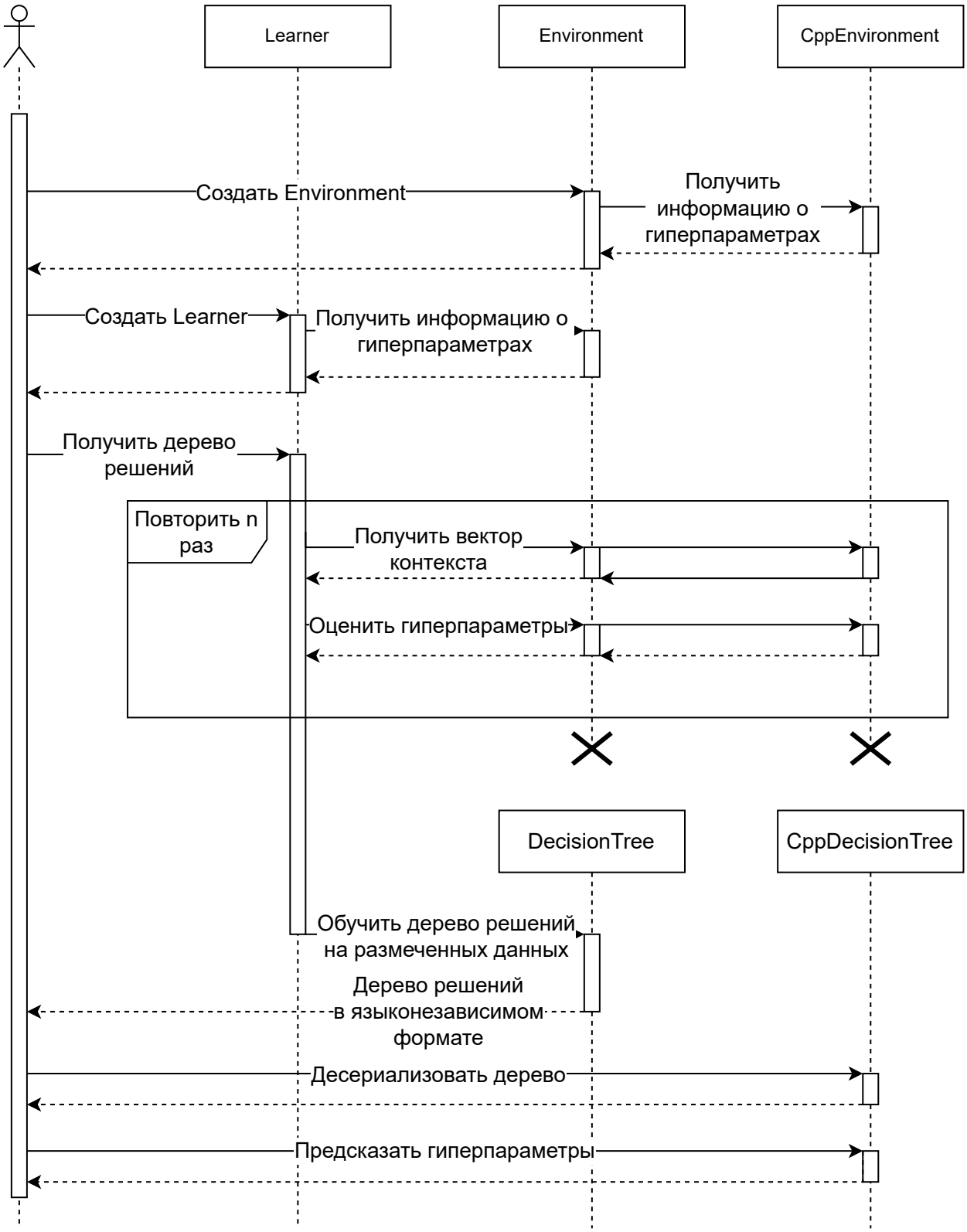
### 3.4.3. Утилиты для работы с C++

Алгоритмы компьютерного зрения не новы. Они пишутся на разных языках под разные платформы. То же самое можно сказать и про реализации метрик, датасетов и прочего инструментария для компьютерного зрения.

В частности компания Кама пишет алгоритмы на языке C++, поэтому для совместимости вышеописанной реализации была добавлена возможность сериализовать дерево в языконезависимый формат, а также была добавлена оболочка для класса Environment, которая общается с соответствующим классом в C++ с помощью потоков ввода-вывода.

Работу системы вместе с оболочкой можно увидеть на рисунке 9.

Рис. 9: Диаграмма последовательности работы алгоритма. Learner взаимодействует с Environment, которая переадресует все вызовы CppEnvironment. Затем обучается DecisionTree, которое можно сериализовать и десериализовать в CppDecisionTree.



# 4. Эксперимент

## 4.1. Постановка эксперимента

Как уже было сказано ранее, в данной работе будут рассматриваться задачи детектирования контуров на дороге и детектирования дорожной полосы. Далее будут описаны метрики и датасеты, выбранные для оценивания качества подбора гиперпараметров в ранее описанных алгоритмах.

### 4.1.1. Метрика

Выбор метода оценивания детектирования линий — нетривиальная задача. Так, например, если взять популярный в компьютерном зрении метод оценивания Intersection Over Union, то окажется что даже небольшой сдвиг предсказанных линий сильно наказывается. То же самое можно сказать и об оценке детекции линий как о задаче класификации (метод пытается для каждого пикселя предсказать граничный он или нет). Необходимо, чтобы метрика поощряла даже приблизительно верные границы, пусть и с меньшим весом.

Для борьбы с этим, в данной работе была выбрана метрика **Normalized Figure of Merit** [13], созданная специально для оценки задачи детектирования линий.

$$NFOM(GT, PRED) =$$
$$\frac{1}{FP + FN}\left[\frac{FP}{|PRED|}\cdot\sum_{p\in PRED}\frac{1}{1+k_{FP}\cdot d^2_{GT}(p)} + \frac{FN}{|GT|}\cdot\sum_{p\in GT}\frac{1}{1+k_{FN}\cdot d^2_{PRED}(p)}\right]$$

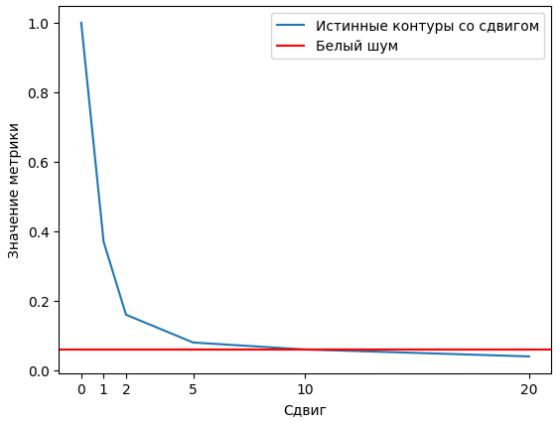
где $PRED$ — границы, предсказанные алгоритмом, $GT$ (Ground truth) — истинные граничные пиксели, $FP$ (False Positives) и $FN$ (False Negatives) — количество пикселей, который алгоритм ошибочно пометил как граничные и ошибочно пометил как неграничные соответственно. Функция $d_A(p)$ — евклидово расстояние от граничного пикселя $p$ до ближайшего граничного пикселя в множестве $A$. Наконец $k_{FP}$ и $k_{FN}$ — гиперпараметры, которые позволяют настроить в метрике, какой пока-

затель штрафовать больше — ложное срабатывание или недостаточную агрессивность. Если $k_{FP} > k_{FN}$, то $N$ больше штрафует алгоритм за False Positives, а при $k_{FP} < k_{FN}$ за False Negatives.

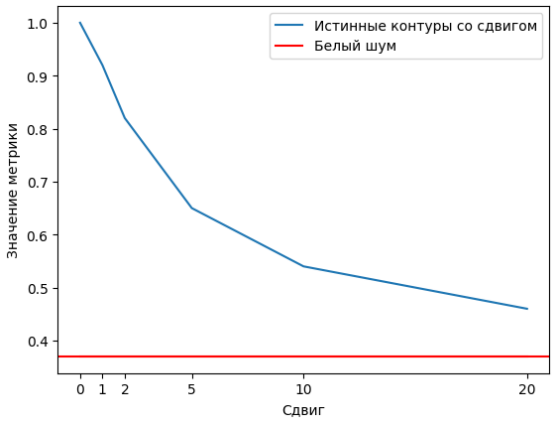
Данная метрика позволяет учитывать небольшой сдвиг предсказания (рис. 10), при этом давая возможность штрафовать алгоритм за ложные срабатывания, или наоборот, за недостаточную агрессивность.

Рис. 10: Сравнение метрик Intersection over Union и Normalized figure of Merit. Значения метрик считались на истинных контурах и их немного сдвинутых версиях. Также приводится значения метрик на белом шуме.

(a) Intersection over Union




(b) Normalized Figure of Merit



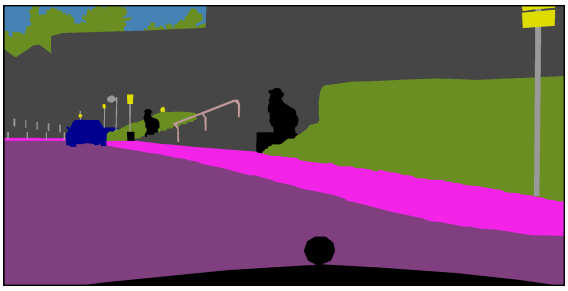

### 4.1.2. Датасет CityScapes

CityScapes [4] — датасет для семантической сегментации городских изображений, специализирующийся на задачах, связанных с пониманием сцен на дороге. Преимущество данного датасета в большом количестве очень качественно сегментированных изображений с большим количеством классов, хорошей видимостью и фиксированной камерой (рис. 11a). Несмотря на то, что данный датасет предоставляет только сегментированное изображение (рис. 11b), не составляет труда конвертировать его в граничные линии (рис. 11c).

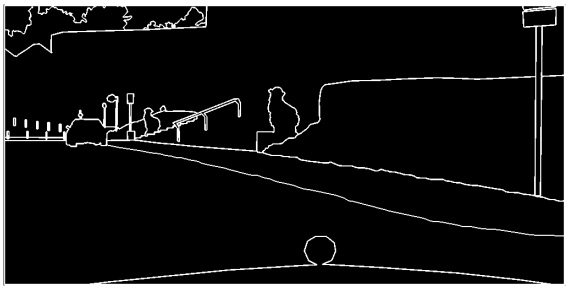
Рис. 11: Примеры из датасета CityScapes

(a) Оригинальное изображение

(b) Сегментирование по классам



(c) Границы классов сегментации



### 4.1.3. Датасет BDD100k

BDD100k [2] — датасет, призванный разнообразить существующие данные для задачи сегментации. Особенность этого датасета в большом разнообразии сцен — как с разными погодными условиями (рис. 12a), так и в разное время суток (рис. 12b), и в большом количестве изображений (подходящих для данной работы изображений почти в три раза больше, чем в CityScapes). Однако, несмотря на хорошую сегментацию, изображения в данном датасете хуже нормализованы (разное положение камеры, не все сцены подходят для выбранной задачи).

### 4.1.4. Датасет Kitti

KITTI Vision Benchmark Suite [8] — датасет для большого количество задач ADAS: от трехмерного отслеживания пешеходов, до сегментации облаков лидара. В рамках данной работы рассматривается часть датасета, предназначенная для оценивания детекции дорожных линий.

Рис. 12: Изображения в BDD100k

(a) Пример снежной сцены (b) Пример ночной сцены

Рис. 13: Пример дорожной сцены из Kitti

(a) RGB изображение (b) Линии дороги

## 4.2. Качество

После реализации вышеописанного алгоритма было проведено сравнение ранее описанных методов. Результаты сравнения можно увидеть на таблице 1. Сравнением велось как по среднему значению метрики, так и по 5-му и 95-му перцентилю из выборки значений на тестовом датасете, чтобы сравнить разброс значений. Зеленым отмечен лучший результат среди всех методов на датасете и алоритме. Так, например, лучше всего гиперпараметры для алгоритма GradientAngles на датасете Kitti подобрала генетическая оптимизация. В двух из четырех задач (по среднему) выигрывает метод Actor-Critic. После идет HyperBand и генетическая оптимизация. По стабильности метода на сложнейших дорожных сценах (пятому перцентилю) нет явных победителей. Интересен тот факт, что Reinforce показал достаточно слабый результат, не опередив ни один метод ни в одной задаче. Это показывает, что обучение с критиком важно для стабильности данного метода.

Таким образом, гиперпараметры к алгоритмам ADAS лучше всего

подбираются на основании контекста, либо с помощью формулирования задачи в виде multi-fidelity оптимизации.

Таблица 1: Результаты различных методов оптимизации на задачах ADAS.

| | 5% | Mean | 95% |
|---|---|---|---|
| | Genetic | | |
| CityScapes + Canny | 45.73 | 58.87 | 67.88 |
| Bdd100k + Canny | 37.07 | 50.03 | 62.37 |
| Kitti + Hough | 27.77 | 34.13 | 41.82 |
| Kitti + GradientAngles | 21.87 | 31.28 | 39.17 |
| | Bayesian | | |
| CityScapes + Canny | 45.00 | 58.61 | 67.81 |
| BDD100k + Canny | 35.88 | 49.88 | 62.69 |
| Kitti + hough | 26.92 | 33.39 | 42.16 |
| Kitti + GradientAngles | 22.88 | 30.70 | 37.07 |
| | HyperBand | | |
| CityScapes + Canny | 50.01 | 60.01 | 68.47 |
| BDD100k + Canny | 35.29 | 50.74 | 63.30 |
| Kitti + Hough | 29.18 | 34.84 | 40.82 |
| Kitti + GradientAngles | 22.02 | 30.96 | 37.67 |
| | REINFORCE | | |
| CityScapes + Canny | 48.42 | 61.49 | 69.13 |
| BDD100k + Canny | 36.46 | 49.03 | 61.51 |
| Kitti + Hough | 28.78 | 34.71 | 43.66 |
| Kitti + GradientAngles | 22.24 | 30.59 | 37.29 |
| | Actor Critic | | |
| CityScapes + Canny | 49.94 | 62.10 | 69.50 |
| BDD100k + Canny | 36.61 | 49.88 | 62.27 |
| Kitti + Hough | 30.42 | 35.48 | 40.86 |
| Kitti + GradientAngles | 21.62 | 30.29 | 38.60 |

Теперь рассмотрим как хорошо дерево решений может обобщить выше описанные методы (табл. 2).

Как можно заметить, результат на классических методах оптимизации без обучения сильно хуже, чем подбор одного набора гиперпараметров для всего датасета. Как уже было сказано ранее, это связано с плохим качеством размеченных данных, а именно в отсутствии непрерывности. С другой стороны дерево решений смогло полностью

Таблица 2: Результаты обучения дерева на размеченных данных. Зеленым отмечен лучший результат среди всех методов, включая вышеупомянутые.

| | 5% | Mean | 95% |
|---|---|---|---|
| | Genetic + Tree | | |
| CityScapes + Canny | 31.52 | 47.49 | 60.42 |
| Bdd100k + Canny | 34.37 | 49.22 | 61.66 |
| Kitti + Hough | 8.99 | 17.97 | 27.01 |
| Kitti + GradientAngles | 0.0 | 20.98 | 36.80 |
| | Bayesian + Tree | | |
| CityScapes + Canny | 29.85 | 48.40 | 62.83 |
| BDD100k + Canny | 33.30 | 48.74 | 61.79 |
| Kitti + hough | 8.86 | 18.17 | 28.29 |
| Kitti + GradientAngles | 15.00 | 25.40 | 34.24 |
| | REINFORCE + Tree | | |
| CityScapes + Canny | 48.42 | 61.49 | 69.13 |
| BDD100k + Canny | 36.46 | 49.03 | 61.51 |
| Kitti + Hough | 28.78 | 34.71 | 43.66 |
| Kitti + GradientAngles | 22.24 | 30.59 | 37.29 |
| | Actor-Critic + Tree | | |
| CityScapes + Canny | 49.94 | 62.10 | 69.50 |
| BDD100k + Canny | 36.61 | 49.88 | 62.27 |
| Kitti + Hough | 30.42 | 35.48 | 40.86 |
| Kitti + GradientAngles | 21.84 | 30.41 | 37.66 |

повторить поведение нейронной сети.

Стоит также отметить, что несмотря на то, что дерево решений имеет лучшие метрики на некоторых задачах, прирост не сильно большой (около 2-5%) по сравнению с существующими решениями для подбора гиперпараметров.

# Заключение

Таким образом были достигнутые следующие результаты:

1. Проведен обзор популярных методов для оптимизации гиперпараметров. В качестве классических методов были рассмотрены: поиск по сетке, случайный поиск, байесовская оптимизация, генетическая оптимизация. В качестве современного метода MFO был рассмотрен метод HyperBand. Конкретно в данной работе были рассмотрены: алгоритм REINFORCE и алгоритм Actor-Critic.

2. Предложен метод для обучения дерева-оптимизатора на основе существующих методов оптимизаций, рассмотренных в обзоре. Метод был реализован на языке Python, с использованием библиотек: Pytorch, Scikit-learn и Hyperopt. Помимо дерева решений были добавлены другие популярные методы оптимизации. Реализация выложена в открытый доступ. [1]

3. Проведено сравнение предложенного метода как с классическими методами оптимизаций (поиск по сетке, случайный поиск, байесовская оптимизация, генетическая оптимизация), так и с нейросетевыми методами (REINFORCE, Actor-Critic). Было выявлено, что предложенный метод может улучшить результат на некоторых задачах ADAS, а в остальных не сильно проигрывает современным методам оптимизации.

4. Результаты представлены на научной конференции “Современные технологии в теории и практике программирования”

[1] https://github.com/osechkina-masha/adas_spbu/tree/tree_hyperopt/DecisionTrees

# Список литературы


[1] Algorithms for hyper-parameter optimization / James Bergstra, Rémi Bardenet, Yoshua Bengio, Balázs Kégl // Advances in neural information processing systems. — 2011. — Vol. 24.

[2] Yu Fisher, Chen Haofeng, Wang Xin et al. BDD100K: A Diverse Driving Dataset for Heterogeneous Multitask Learning. — 2018. — URL: https://arxiv.org/abs/1805.04687 (дата обращения: 2022-12-15).

[3] Canny John. A computational approach to edge detection // IEEE Transactions on pattern analysis and machine intelligence. — 1986. — no. 6. — P. 679–698.

[4] Cordts Marius, Omran Mohamed, Ramos Sebastian et al. The Cityscapes Dataset for Semantic Urban Scene Understanding. — 2016. — URL: https://arxiv.org/abs/1604.01685 (дата обращения: 2022-12-15).

[5] Fernández-Godino M. Giselle. Review of multi-fidelity models. — 2023. — 1609.07196.

[6] Finn Chelsea, Abbeel Pieter, Levine Sergey. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. — 2017. — 1703.03400.

[7] Frazier Peter I. A Tutorial on Bayesian Optimization. — 2018. — URL: https://arxiv.org/abs/1807.02811 (дата обращения: 2022-12-15).

[8] Fritsch Jannik, Kuehnl Tobias, Geiger Andreas. A New Performance Measure and Evaluation Benchmark for Road Detection Algorithms // International Conference on Intelligent Transportation Systems (ITSC). — 2013.

[9] Li Lisha, Jamieson Kevin, DeSalvo Giulia et al. Hyperband: A Novel

Bandit-Based Approach to Hyperparameter Optimization. — 2018. — 1603.06560.

[10] Illingworth John, Kittler Josef. A survey of the Hough transform // Computer vision, graphics, and image processing. — 1988. — Vol. 44, no. 1. — P. 87–116.

[11] Konda Vijay, Tsitsiklis John. Actor-critic algorithms // Advances in neural information processing systems. — 1999. — Vol. 12.

[12] Liu Xiyuan, Wu Jia, Chen Senpeng. A context-based meta-reinforcement learning approach to efficient hyperparameter optimization // Neurocomputing. — 2022. — Vol. 478. — P. 89–103.

[13] Magnier Baptiste. Edge Detection Evaluation: A New Normalized Figure of Merit // ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). — 2019. — P. 2407–2411.

[14] Multiobjective tree-structured parzen estimator for computationally expensive optimization problems / Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, Masaki Onishi // Proceedings of the 2020 genetic and evolutionary computation conference. — 2020. — P. 533–541.

[15] PyTorch: An Imperative Style, High-Performance Deep Learning Library / Adam Paszke, Sam Gross, Francisco Massa et al. // Advances in Neural Information Processing Systems 32. — Curran Associates, Inc., 2019. — P. 8024–8035. — URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-pdf.

[16] Sammut Claude, Webb Geoffrey I. Encyclopedia of machine learning. — Springer Science & Business Media, 2011.

[17] Scikit-learn: Machine Learning in Python / F. Pedregosa, G. Varoquaux, A. Gramfort et al. // Journal of Machine Learning Research. — 2011. — Vol. 12. — P. 2825–2830.

[18] Simon Dan. Evolutionary optimization algorithms. — John Wiley & Sons, 2013.

[19] A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients / Ivo Grondman, Lucian Busoniu, Gabriel A. D. Lopes, Robert Babuska // IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews). — 2012. — Vol. 42, no. 6. — P. 1291–1307.

[20] Tangirala Suryakanthi. Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm // International Journal of Advanced Computer Science and Applications. — 2020. — Vol. 11, no. 2. — P. 612–619.

[21] Williams Ronald J. Simple statistical gradient-following algorithms for connectionist reinforcement learning // Reinforcement learning. — 1992. — P. 5–32.

[22] Wu Jia, Chen SenPeng, Liu XiYuan. Efficient hyperparameter optimization through model-based reinforcement learning // Neurocomputing. — 2020. — Vol. 409. — P. 381–393.

[23] П. А. Курников Д. Л. Шоломов А. В. Панченко. Система определения туманных дорожных сцен, основанная на ансамбле классификаторов // ИТиВС. — 2018. — P. 70–77.

[24] Показатели состояния безопасности дорожного движения. — URL: http://stat.gibdd.ru/ (дата обращения: 2022-12-15).