САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Меньшиков Александр Сергеевич

Выпускная квалификационная работа
Распознавание пешеходов для беспилотных автомобилей

Уровень образования: бакалавриат

Направление 01.03.02 «Фундаментальная информатика и информационные технологии»

ООП «Программирование и информационные технологии»

Научный руководитель:

доктор физ.-мат. наук,

профессор кафедры КТиС

Сотникова М.В.

Санкт-Петербург

2019

Содержание

Введение

Дорожно-транспортные происшествия являются повсеместным явлением, основной из причин которых является человеческий фактор, так как по различным причинам человек не способен всегда своевременно реагировать на изменяющуюся обстановку на дороге. А из-за постоянного роста населения в мире число участников дорожного движения только увеличивается, что приводит к ещё большему числу аварий, часть которых с летальным исходом. Поэтому одним из способов предупреждения ДТП в настоящее время используются системы обнаружения пешеходов. Такая система используется для отслеживания положения и перемещения пешеходов, чтобы предотвратить столкновение с их участием, либо уменьшить сопутствующий ущерб в критических ситуациях, когда избежать аварии невозможно.

В связи с возрастающей популярностью беспилотных транспортных средств актуальность создания таких систем только увеличивается.

Изначально в беспилотных транспортных средствах для обнаружения пешеходов использовалось данные не с камеры, а с радара, лидара и других датчиков. Но подобная система из сенсоров имеет высокую цену и требует дорогого обслуживания в случае выхода из строя, что делает приобретение такого автомобиля экономически затруднительным для покупателя. Поэтому разработка методов для обычной камеры приобрела такую популярность.

Обнаружение пешеходов до сих пор считается не до конца решённой задачей компьютерного зрения. Виной этому является неоднородность внешнего вида пешеходов, которая возникает по причине разных поз, одежды, также влияет фон, на котором запечатлён пешеход, и освещённость, а также скорость обработки поступающих изображений, которая очень критически важна для беспилотных автомобилей.

Существует множество алгоритмов, часть которых предлагает высокую точность, но низкую скорость работы, а другие наоборот, высокую скорость

распознавания, но не с такой большой точностью детектирования. И в такой сфере, как дорожное движение, беспилотный автомобиль должен ориентироваться в пространстве как минимум с реакцией обычного водителя, так как является средством повышенной опасности, в связи с чем любая задержка или неточность распознавания сможет привести к человеческим жертвам.

Постановка задачи

Целью данной работы является разработка системы детектирования пешеходов на изображениях, полученных с обычной камеры.

Пусть имеется входящий поток изображений с камеры, установленной в транспортном средстве. Изображения пешеходов могут быть разного размера, принимать различные позы, носить разную одежду, задний фон не привязан к определённой среде обитания. Необходимо наиболее точно и быстро находить положение пешехода, его размер в каждом изображении.

Данную цель можно разбить на следующие задачи:

1) Изучение предметной области.
2) Анализ существующих методов, алгоритмов и выбор одного из них для разработки системы.
3) Разработка программной части системы детектирования пешеходов.
4) Анализ результатов работы системы.

Обзор литературы

В этой главе будут рассмотрены различные алгоритмы распознавания пешеходов на изображениях, полученных с камеры.

Детектирование пешеходов можно разбить на 2 этапа ([]):

1) Выбор частей изображения, которые потенциально могут быть пешеходами – генерация кандидатов.
2) Определение того, является ли выбранный участок пешеходом - классификация.

Улучшение, модификация этих двух этапов и приводит к появлению новых методов детектирования пешеходов.

Саму же структуру работы системы детектирования пешеходов можно описать в виде следующих модулей:

1) Преобразование входных данных для дальнейшей работы с ними.
2) Отделение участков изображения, в которых потенциально могут быть пешеходы – генерация кандидатов.
3) Определение наличия или отсутствия пешехода на выбранных участках изображения – классификация.
4) Проверка на ложноположительные срабатывания и уточнение положения пешехода на изображении.

В работе [2] описывается каскадный алгоритм Виолы - Джонса, в основе которого лежат признаки Хаара. Поиск пешеходов производится путём сканирования изображения скользящим окном, плотно заполненном признаками Хаара. Для этого применяются признаки различных типов, масштабов и положений. В работе [4] объясняется выбор данного подхода, и описывается представление самих признаков. Преимущество его в том, что он работает намного быстрее систем, которые построены на попиксельном анализе изображения. Внутри скользящего окна таких признаков генерируется приблизительно 200 000, за счет изменения масштаба признаков и их положения в окне. Сами признаки Хаара представляют из

себя разность сумм пикселей двух смежных областей внутри прямоугольника. Каждый признак может показать наличие или отсутствие какой-либо конкретной характеристики изображения (границы или изменение текстур). Само сканирование производится последовательно для различных масштабов. Для ускорения работы алгоритма на маломощных системах подбирается подмножество признаков, на котором производится обучение. Все признаки поступают на вход классификатора, который даёт результат «верно» либо «ложь».

Авторы работы [1] предлагают использовать HOG (Histograms of Oriented Gradients) дескрипторы. Реализация его заключается в том, что производится разбиение всего изображения на маленькие регионы, каждому из них соответствует одномерная гистограмма направлений градиентов (ориентации рёбер). Определённый диапазон направлений градиентов соответствует каждому из бинов гистограммы, которые так же содержат суммарную магнитуду градиентов соответствующего направления. Благодаря такому подходу, рёбра на изображении представляются в виде совокупности всех таких гистограмм. Для решения проблемы с перепадами освещения делается нормировка значений в полученных гистограммах относительно области, которая содержит данную ячейку. Для этого нужно объединить соседние ячейки в группы, а затем провести нормирование значений в гистограммах всех ячеек внутри блока на суммарное значение магнитуды градиентов по тому блоку, который рассматривается в данный момент. В качестве метода машинного обучения применяется линейная машина опорных векторов, описанная в работе [5].

Сравнение приведённых систем распознавания пешеходов описано в работе [2]. Для обучения использовалась база данных Daimler Pedestrian Detection Benchmark, которая содержит как изображения с пешеходами, так и без них. К системам предъявляют такие требования, как обнаружение как можно большей доли объектов и минимизирование количества ложных срабатываний. По итогам сравнения описанный HOG-алгоритм имел самое

высокое качество обнаружения, опережая каскадный алгоритм, основанный на признаках Хаара (точность в среднем около 85% и около 73% у HOG-алгоритма и каскадного алгоритма соответственно), но при этом было отставание по скорости процесса обнаружения (430 миллисекунд и 20 миллисекунд у HOG-алгоритма и каскадного алгоритма соответственно).

В работе [3] было решено совместить описанные выше методы, чтобы получить скорость работы каскадного алгоритма и качество обнаружения HOG-алгоритма. В этом алгоритме признаки представляют из себя блоки, состоящие из 4 ячеек, заполненных HOG признаками, дискретизированными по 9 бинам. Таким образом, каждый признак (блок) представляется 36-ти размерным вектором HOG признаков внутри блока. Подобный подход существенно ускорил процесс распознавания пешеходов (около 26 миллисекунд), но при этом сохранил высокое качество обнаружения, которое сравнимо с качеством обнаружения HOG-алгоритма (около 83%).

У описанных раннее методов каждый из модулей реализовывается в виде отдельных решений. В работе [7] анализируется работа свёрточной нейронной сети, которая представляют из себя особую архитектуру нейронной сети, предназначенную для задач по обработке изображений. Их особенность заключается в том, что модули со второго по четвёртый прорабатываются в виде одной нейронной сети, либо их комплекса. Такая модель с различными модификациями демонстрировала качество обнаружения до 88%. Идея работы свёрточной нейронной сети описана в статье [8], которая заключается в том, что важные для распознавания признаки формируются в местах взаимодействия близких друг к другу пикселей, а не расположенных в противоположных углах. Так же не будет иметь значение участок изображения, на котором была обнаружена важная черта.

По проанализированной литературе было решено разрабатывать систему детектирования пешеходов на основе свёрточной нейронной сети, так как для её обучения не нужно самостоятельно формировать признаки, что

довольно трудоёмкая и долгая задача. Но нужно сформировать такую структуру и задать параметры, чтобы получить удовлетворяющее качество работы системы.

Глава 1. Выбор архитектуры свёрточной нейронной сети

1.1. Описание работы свёрточной нейронной сети.

Свёрточная нейронная сеть – особая архитектура искусственных нейронных сетей, предназначенная для эффективного распознавания образов на изображениях.

В её основе лежит оператор свёртки ([8]) (ядро свёртки), который графически кодирует тот или иной признак. Производится наложение ядра, представленного в виде матрицы **K**, на двумерное изображение **I** различными какими возможно способами. После чего сумма произведений элементов ядра и начального изображения записывается:

$$(I * K)_{xy} = \sum_{i=1}^{h} \sum_{j=1}^{w} K_{ij} \times I_{x+i-1, y+j-1},$$

где h – высота матрицы ядра, w – ширина матрицы ядра.

Принцип наложения ядра на изображение представлен на рисунке 1.

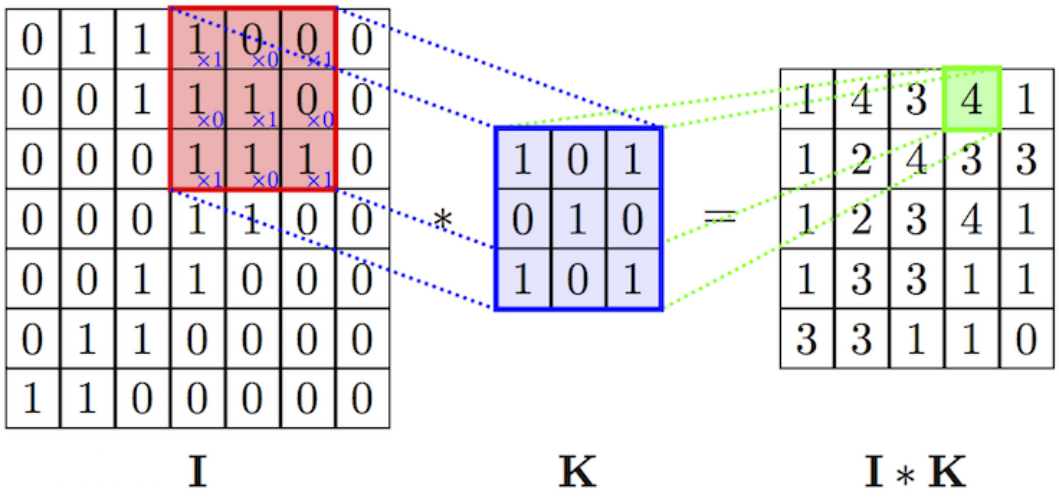

Рис. 1: Операция свёртки

Основой свёрточного слоя в свёрточной нейронной сети является сам оператор свёртки. Этот слой включается некоторое количество ядер K (фильтры), с помощью которых вычисляется свёртка изображения,

полученного из предыдущего слоя сети, при этом каждый раз нужно прибавлять составляющую смещения для каждого ядра. Затем, чтобы выявить какой-либо признак на изображении, на определённых слоях сети к преобразованному изображению нужно применить функцию активации. В результате этой операции получается карта признаков.

За счёт применения операции свёртки уменьшается количество параметров, если сравнивать с полносвязным слоем, но при этом используется больше гиперпараметров, которые выбираются перед обучением.

Гиперпараметры, которые есть у свёрточных слоёв:

- *Глубина* задаёт количество коэффициентов смещения и ядер, которые задействуются в слое;
- *Ширина*, *высота* каждого из ядер;
- *Шаг* определяет смещение ядра для каждого шага при вычислении элементов полученного изображения. С увеличением этого параметра уменьшается размер выходного изображения;
- *Отступ* необходим в тех случаях, когда нужно сохранить изначальных размер изображения после применения операции свёртки. Для этого к рисунку по краям добавляются нули, количество которых определяется этим параметром.

Существует возможность построить сеть только с помощью свёрточных слоёв, но их основное применение – это выявление наиболее важных признаков перед субдискретизацией.

В качестве способа субдискретизации изображения чаще всего используется слой подвыборки (слой субдискретизации), который при получении небольших частей входящего изображения производит объединение каждой из частей в одно значение. Чаще всего способом агрегации является выбор максимального элемента из имеющихся значений

пикселей (рис. 2), либо вычисление среднего значения.

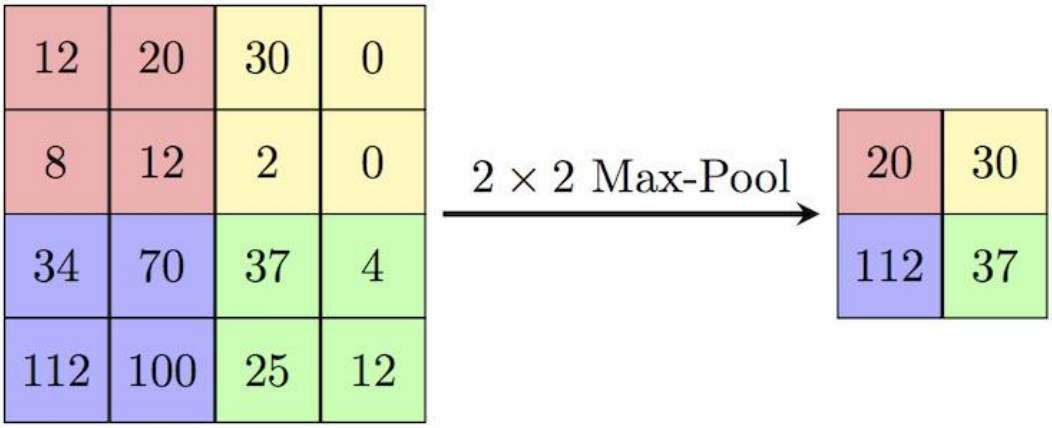

Рис. 2: Субдискрецизация изображения

Если рассматривать архитектуру свёрточной нейронной сети целиком, то она из себя представляет чередование слоёв свёртки и дискретизации, при этом слои свёртки иногда могут идти подряд. Проход по этим слоям производится с целью того, чтобы сократить длину и ширину конкретного канала, но за счёт этого увеличивается его значение. И в конце добавляется несколько слоёв, принимающих пиксели в виде независимых друг от друга значений, которые называются полносвязными слоями. А заключительным является слой вычисления функции цели.

1.2. Архитектура сети для детектора

Архитектура сети выбиралась на основе предыдущих работ ([12]) в сфере детектирования пешеходов, а также с учётом конфигурации вычислительной системы, на которой будет проводится обучение и тестирование модели.

На рисунке 3 описана архитектура детектора, который представляет из себя свёрточную нейронную сеть с чередующимися слоями свёртки и субдискретизации. В качестве функции активации для свёрточных слоёв будет использоваться **ReLU**, так как её вычисление не требует много ресурсов, благодаря чему обучение будет проходить быстрее, и нет риска

разрастания/затухания градиента.

Так же для того, чтобы решить проблему переобучения сети, когда часть нейронов может стать лишней, так как не влияет на результат работы сети, либо небольшое количество нейронов влияет на результат так, что остальным нейронам приходится исправлять получившиеся ошибки, применяется метод **dropout**. Он заключается в том, что с заданной вероятностью при прохождении по каждому нейрону конкретного слоя за одну итерацию обучения, этот нейрон полностью исключается на время итерации. Благодаря этому сеть будет учитывать все нейроны внутри слоя, а не полагаться на один из них или группу.

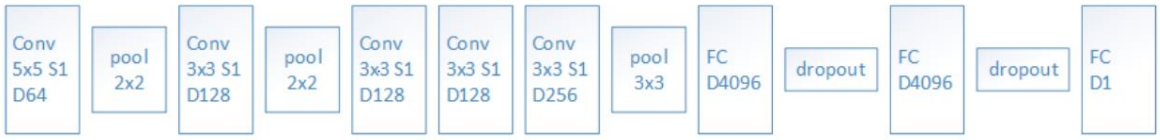

Рис. 3: Архитектура детектора ([12]).

Conv обозначает свёрточный слой. После указан размер ядра свёртки, параметр **S** задаёт шаг смещения ядра при его прохождении по изображению.

Pool – это слой субдискрецизации, при размере 2x2 уменьшающий входящее изображение в 2 раза, а при размере 3x3 – в 3 раза.

FC – полносвязный слой, принимающий каждый элемент входящего изображения.

Параметр **D** определяет количество ядер свёртки или количество блоков в слое.

1.3. Обучение свёрточной нейронной сети

В процессе обучения нейронная сеть получает на вход образ из тренировочной выборки. После чего полученный результат сравнивается с тем, который ожидался от сети. В итоге получаем результат функции ошибки – дельта ошибки, который представляет из себя разность между желаемым и

полученным ответом. После чего полученная разница распространяется по всей сети, на все нейроны, которые связаны друг с другом ([9]).

Величина ошибки определяется следующим образом:

$$E_p = \frac{1}{2}\sum_j (t_{pj} - y_{pj})^2 ,$$

где E – величина функции ошибки для образа p;

t – желаемый выход нейрона j для образа p;

y – активированный выход нейрона j для образа p.

Неактивированное состояние каждого нейрона j для образа p представляется в виде взвешенной суммы:

$$s_{pj} = \sum_i w_{ij} y_{pi} ,$$

где s – взвешенная сумма выходов связанных нейронов предыдущего слоя на все связи;

w – вес связи между i и j нейронами;

y – активированное состояние нейрона j предыдущего слоя для образа p.

Выход всех нейронов j является значение функции активации f. Активированное состояние нейрона вычисляется по формуле:

$$y_{pj} = f_j(s_{pj}) ,$$

где y – активированное состояние нейрона j для образа p;

f – функция активации;

s – неактивированное состояние нейрона j для образа p.

Метод градиентного спуска используется для уменьшения ошибки, который является движением вдоль вектора градиента.

Градиент функции потерь представляет из себя вектор частных производных, вычисляемых по формуле:

$$\nabla E(W) = [\frac{dE}{dw_1}, ..., \frac{dE}{dw_n}] \ ,$$

где $\nabla E(W)$ - градиент функции потери от матрицы весов;

$\frac{dE}{dw}$ - частная производна функции ошибки по весу нейрона;

n – общее количество весов сети.

Производную функции ошибки по конкретному образу можно записать по правилу цепочки ([10]):

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} * \frac{\partial y_j}{\partial s_j} * \frac{\partial s_j}{\partial w_{ij}} \ ,$$

где $\frac{\partial E}{\partial w_{ij}}$ - значение производной функции ошибки по весу w, между i и j нейронами;

$\frac{\partial E}{\partial y_j}$ - ошибка нейрона j;

$\frac{\partial y_j}{\partial s_j}$ - значение производной функции активации по её аргументу для нейрона j, эту часть достаточно просто вычислить;

$\frac{\partial s_j}{\partial w_{ij}}$ - выход i нейрона предыдущего слоя (по отношению к нейрону j).

Примем ошибку нейрона за δ. Ошибка нейрона для скрытого слоя вычисляется следующим образом:

$$\delta_i = \frac{\partial y_i}{\partial s_i} * \sum_j \delta_j * w_{ij} ,$$

где $\frac{\partial y_i}{\partial s_i}$ - значение производной функции активации по её аргументу для нейрона i;

δ_i - ошибка нейрона i скрытого слоя;

δ_j - ошибка нейрона j следующего слоя;

w_{ij} - вес связи между нейроном i текущего (скрытого) слоя и нейроном j выходного или тоже скрытого слоя.

Для слоя субдискретизации, который располагается перед свёрточным слоем, дельта ошибки с помощью обратной свёртки [11]. Для этого надо ядро карты свёртки повернуть на 180 градусов и выполнить стандартную операцию свёртки по вычисленным ранее дельтам карты свёртки, но при этом сканирующее окно должно выходить за пределы карты.

Так как субдискретизация производится путём выбора из групп нейронов с максимальным значением, то при обратном распространении результат функции ошибки из слоя субдискретизации копируется ранее выбранному максимальному нейрону, а отсеянным нейронам передаётся нулевая дельта ошибки.

Глава 2. Реализация системы

2.1. Подготовка среды для разработки

Обучение проводится на компьютере с характеристиками: процессор – Intel Core i5-3470 3.2GHz, объём оперативной памяти – 8 ГБ, данные хранятся на жёстком диске типа – HDD. Графический процессор в системе отсутствует, что значительно увеличивает время обучения и дальнейшей работы детектора, так как графический процессор лучше справляется с различными математическими вычислениями, в отличие от центрального, в связи с чем лучше подходит для работы со свёрточными нейронными сетями. Язык программирования – Python, так как он хорошо подходит для сложных вычислений и поддерживает большое количество полезных библиотек.

В качестве фреймворка для глубокого обучения был выбран **Caffe**, так как он обладает всем необходимым набором функций для моделирования сети, поддерживает язык программирования Python и поддерживает адаптацию для центрального процессора. Изначально есть возможность задания свёрточных, субдискретизирующих, полносвязных слоёв. Модель сети описывается с помощью текстового протокола, что значительно упрощает настройку параметров слоёв.

База пешеходов для обучения и тестирования – Caltech. Она содержит 10 часов видео в разрешении 640x480, снятого с камеры на транспортном средстве. Включает около 250000 фреймов и 2300 уникальных пешеходов. Подобных данных более чем достаточно для обучения и тестирования сети.

2.2. Подготовка базы пешеходов

Скачанная база представляет из себя набор seq-файлов. Преобразовываем их в изображения в формате png, названия которых содержат номер сета и seq-файла, из которого было взято изображения, а

также его порядковый номер в полученных тренировочных и тестовых выборках.

Ещё нужно подготовить файл с данными, содержащими координаты пешеходов. На основе этих данных после передачи их сети будет производится обучение. Данные берутся из аннотаций к seq-файлам и содержат:

- Номер фрейма;
- Высота изображения;
- Ширина изображения;
- Сколько пешеходов находится в кадре;
- Координаты участка с пешеходом;

2.3. Создание модели

Основная сеть представляет из себя чередование свёрточных слоёв, применение к ним функции активации ReLU, субдискритизирующих слоёв, а в конце применяется операция dropout к полносвязным слоям для уменьшения влияния отдельных нейронов.

Для свёрточных слоёв надо задать:

- количество фильтров;
- их размер;
- количество нулей, добавляемых со всех сторон изображения.

У слоя субдискретизации задаётся:

- размер окна;
- способ уменьшения размера изображения (в данном случае это будет выбор максимального элемента).

У метода dropout надо задать вероятность исключение нейрона из слоя на одну итерацию.

Для обучения указывается:

- написанная модель сети, на которой будет происходить

обучение;

- коэффициент влияния обратного градиента на изменение параметров сети;
- максимальное количество итераций;
- Использование CPU или GPU.

Для обучения детектора обычно выставляется минимум 10 тысяч итераций. Но так как на имеющейся аппаратуре нет подходящего для работы детектора графического процессора, процесс обучения займёт слишком много времени. Поэтому обучение проводилось на 1000 итерациях, которые потребовали 51 час.

Глава 3. Исследование детектора

3.1. Критерии тестирования работы детектора

В качестве критериев тестирования будут выступать:

- выступать среднее время работы детектора не тестовой выборке, которое вычисляется отношением общего времени работы детектора к количеству изображений в тестовой выборке;
- полнота, равная отношению релевантных обнаружений (удовлетворяющих условию) к общему количеству пешеходов.

3.2. Оценка критериев

Рис. 4: Пример работы детектора

Полнота: параметр полноты получился равен **0.513**. Это означает, что примерно в половине случаев детектор был способен обнаружить пешехода. Но стоит учитывать количество итераций при обучении, с увеличением которых этот критерий мог увеличиться.

Среднее время работы: при непрерывной работе детектора среднее его время анализа изображения составил **10.3** секунды. Но эти исследования проводились с использованием ресурсов центрального процессора, а не графического, подключение которого должно было ускорить работу детектора.

3.3. Выводы

Детектор показал качество работы ниже, чем у методов, описанных в научных работах. И скорость анализа изображений не позволяет использовать данный детектор в системах реального времени.

В дальнейшем планируется приобрести подходящий графический процессор, для более точной оценки работы сети. А также изменение параметров сети, добавление новых слоёв способно увеличить качество и скорость работы детектора.

Заключение

В результате работы были получены следующие результаты:

- Проведён анализ существующих методов и алгоритмов распознавания пешеходов и был сделан выбор одного из них для дальнейшего исследования.
- Проанализирована архитектура выбранного метода: свёрточные нейронные сети.
- Подобрана структура свёрточной нейронной сети для детектора.
- Реализован прототип системы детектирования пешеходов.
- Выполнен анализ работы полученного прототипа.

Список литературы

1. N. Dalal and B. Triggs. Histograms of oriented gradients for human detection // Conference on Computer Vision and Pattern Recognition (CVPR), 2005.
2. M. Enzweiler and Dariu M. Gavrila. Monocular Pedestrian Detection: Survey and Experiments // Pattern Analysis and Machine Intelligence, 2009.
3. Qiang Zhu, Sai Avidan, Mei-Chen Yeh, and Kwang-Ting Cheng. Fast Human Detection Using a Cascade of Histograms of Oriented Gradients // Computer Vision and Pattern Recognition, 2006.
4. P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features // Accepted Conference on Computer Vision and Pattern Recognition, 2001
5. V.N. Vapnik. The Nature of Statistical Learning Theory // Springer-Verlag, 1995.
6. Zhaowei Cai and Mohammed Saberian. Learning Complexity-Aware Cascades for Deep Pedestrian Detection // International Conference on Computer Vision, Dec. 15, 2015.
7. Yichong Xu, Tianjun Xiao, Jiaxing Zhang, Kuiyuan Yang, Zheng Zhang. Scale-Invariant Convolutional Neural Networks. https://arxiv.org/pdf/1411.6369
8. Глубокое обучение для новичков: распознаем изображения с помощью сверточных сетей. https://habr.com/ru/company/wunderfund/blog/314872/
9. A Step by Step Backpropagation Example. https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/
10. Backpropagation in Convolutional Neural Networks. https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/
11. Convolutional Neural Networks backpropagation: from intuition to derivation. https://grzegorzgwardys.wordpress.com/2016/04/22/8/

12. Anelia Angelova, Alex Krizhevsky, Vincent Vanhoucke, Abhijit Ogale, Dave Ferguson. Real-Time Pedestrian Detection With Deep Network Cascades. https://static.googleusercontent.com/media/research.google.com/ru//pubs/archive/43850.pdf

Приложение

Модель свёрточной нейронной сети для обучения:

```
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  transform_param {
    scale: 0.003921568859368563
  }
  data_param {
    source: "data/annotations_train.txt"
    batch_size: 64
  }
}
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  convolution_param {
    num_output: 64
    kernel_size: 5
    stride: 1
    pad: 1
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "pool1"
  type: "Pooling"
  bottom: "conv1"
  top: "pool1"
  pooling_param {
    pool: MAX
    kernel_size: 2
  }
}
layer {
  name: "conv2"
  type: "Convolution"
  bottom: "pool1"
  top: "conv2"
  convolution_param {
    num_output: 128
    kernel_size: 3
    stride: 1
    pad: 1
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "relu1"
  type: "ReLU"
  bottom: "conv2"
  top: "conv2"
}
layer {
```

```
  name: "pool2"
  type: "Pooling"
  bottom: "conv2"
  top: "pool2"
  pooling_param {
    pool: MAX
    kernel_size: 2
  }
}
layer {
  name: "conv3"
  type: "Convolution"
  bottom: "pool2"
  top: "conv3"
  convolution_param {
    num_output: 128
    kernel_size: 3
    stride: 1
    pad: 1
    weight_filler {
      type: "xavier"
    }
  }
}
layer {
  name: "conv4"
  type: "Convolution"
  bottom: "conv3"
  top: "conv4"
  convolution_param {
    num_output: 128
    kernel_size: 3
    stride: 1
    pad: 1
    weight_filler {
      type: "xavier"
    }
  }
}
      layer {
        name: "conv5"
        type: "Convolution"
        bottom: "conv4"
        top: "conv5"
        convolution_param {
          num_output: 256
          kernel_size: 3
          stride: 1
          pad: 1
          weight_filler {
            type: "xavier"
          }
        }
      }
      layer {
        name: "relu2"
        type: "ReLU"
        bottom: "conv5"
        top: "conv5"
      }
      layer {
        name: "pool3"
        type: "Pooling"
        bottom: "conv5"
        top: "pool3"
        pooling_param {
          pool: MAX
          kernel_size: 3
```

```
    }
  }
  layer {
    name: "fc1"
    type: "InnerProduct"
    bottom: "pool3"
    top: "drop1"
    top: "fc2"
    inner_product_param {
      num_output: 4096
      weight_filler {
        type: "xavier"
      }
    }
  }
  layer {
    name: "drop1"
    type: "Dropout"
    bottom: "fc1"
    top: "fc1"
    dropout_param {
      dropout_ratio: 0.5
    }
  }
  layer {
    name: "relu3"
    type: "ReLU"
    bottom: "fc1"
    top: "fc1"
  }
  layer {
    name: "fc2"
    type: "InnerProduct"
    bottom: "fc1"
    top: "drop2"
    top: "fc3"
    inner_product_param {
      num_output: 4096
      weight_filler {
        type: "xavier"
      }
    }
  }
  layer {
    name: "drop2"
    type: "Dropout"
    bottom: "fc2"
    top: "fc2"
    dropout_param {
      dropout_ratio: 0.5
    }
  }
  layer {
    name: "relu4"
    type: "ReLU"
    bottom: "fc2"
    top: "fc2"
  }
  layer {
    name: "fc3"
    type: "InnerProduct"
    bottom: "fc2"
    top: "score"
    inner_product_param {
      num_output: 4096
      weight_filler {
        type: "xavier"
      }
```

```
197.         }
198.       }
199.       layer {
200.         name: "score"
201.         type: "InnerProduct"
202.         bottom: "fc3"
203.         top: "score"
204.         inner_product_param {
205.           num_output: 10
206.           weight_filler {
207.             type: "xavier"
208.           }
209.         }
210.       }
211.       layer {
212.         name: "loss"
213.         type: "SoftmaxWithLoss"
214.         bottom: "score"
215.         bottom: "label"
216.         top: "loss"
217.       }
```