Санкт-Петербургский государственный университет
Математико-механический факультет

Кафедра системного программирования

HOG+HOF+CSS дескриптор в задаче распознавания пешеходов

Курсовая работа студента 344 группы
Малыгина Евгения Сергеевича

Научный руководитель:
А.Т. Вахитов
доцент кафедры системного программирования

Санкт-Петербург
2014

Оглавление

Введение

Задача распознавания пешеходов становится всё более актуальной. Это обусловлено тем, что доступность вычислительных мощностей и камер постоянно увеличивается. Системы, распознающие пешеходов, могут быть встроены в транспорт, повышая безопасность на дорогах. Такие системы являются важными для беспилотных транспортных средств. Визуальное распознавание пешеходов также может быть востребовано, когда необходимо учитывать численность проходящих мимо камеры людей, например, для сбора информации о загруженности городских улиц.

В области распознавания пешеходов ведутся активные исследования. Стандартный подход к проблеме — выбор способа формирования векторов признаков и выбор классификатора. Учёные предлагают способы получить из изображений новые признаки и варьируют классификаторы, чтобы изучить полученный метод распознавания по таким критериям, как точность (вероятность правильно определить, есть ли в данном поисковом окне пешеход или нет) и скорость работы (некоторые подходы работают в реальном времени, а некоторые могут работать минуты). Некоторые учёные подходят к проблеме более широко и изучают не только метод распознавания, но и статистическую информацию о самой задаче, например, распределение пешеходов по разным областям изображения. Статья [1] представляет собой исследование, в котором приводится и обширная статистическая информация о проблеме, и сравнение множества перспективных современных детекторов пешеходов.

Возникла идея реализовать и изучить какой-либо перспективный метод распознавания пешеходов. В качестве такого метода был выбран метод, предложенный в статье [2]. Цель моей работы — реализация HOG+HOF+CSS дескриптора. Ещё одной целью является проверка точности данного дескриптора по сравнению с общеизвестным HOG дескриптором в случае его отдельного применения для распознавания пешеходов.

Формирование вектора признаков

Вектор признаков — это вектор, каким-либо образом характеризующий входные данные. В HOG+HOF+CSS дескрипторе вектор признаков формируется из 3-х векторов, полученных в результате работы HOG, HOF и CSS дескрипторов. Каждый из этих 3-х дескрипторов получает вектор признаков из входных изображений по-своему. Из изображений таким образом можно получить разноплановую информацию, которую будет проще классифицировать.

HOG – histograms of oriented gradients

Гистограммы ориентированных градиентов — известный дескриптор, имеющий множество реализаций. Его повторная реализация не несёт интереса, поэтому была использована его готовая реализация из OpenCV.

Гистограммы ориентированных градиентов как дескриптор извлекают из одиночного изображения информацию о цветовых контурах, вычисляя локальные связи между цветами на изображении.

HOG дескриптор показывает хорошую производительность в задаче распознавания пешеходов, поэтому обычно с ним сравнивают новые детекторы, например, в статьях [1], [2].

HOF – histograms of oriented optical flow

Гистограммы ориентированного оптического потока — дескриптор, предложенный в статье [3]. Этот дескриптор извлекает из 2-х последовательных изображений информацию об относительном движении близких малых областей картинки относительно друг друга.

Сначала по двум картинкам вычисляется оптический поток по какому-то уже существующему алгоритму. В данной работе для нахождения оптического потока был использован метод Farnerback из OpenCV.

После нахождения оптического потока у каждой точки в окне поиска есть информация о смещении по горизонтали и по вертикали. Окно поиска разбивается на блоки размером 8x8. В каждом блоке считается среднее смещение точек по горизонтали и по вертикали.

Затем между каждыми 4-мя смежными блоками вычисляются компоненты вектора признаков. Для этого всевозможными способами (рис. 1) из 4-х смежных блоков выбираются 2 блока, и между ними происходит вычитание усреднённых значений смещений точек.

Из каждых 4-х смежных блоков получается 8 дробных параметров. Эти параметры нормализуются между собой и добавляются в вектор признаков HOF дескриптора.

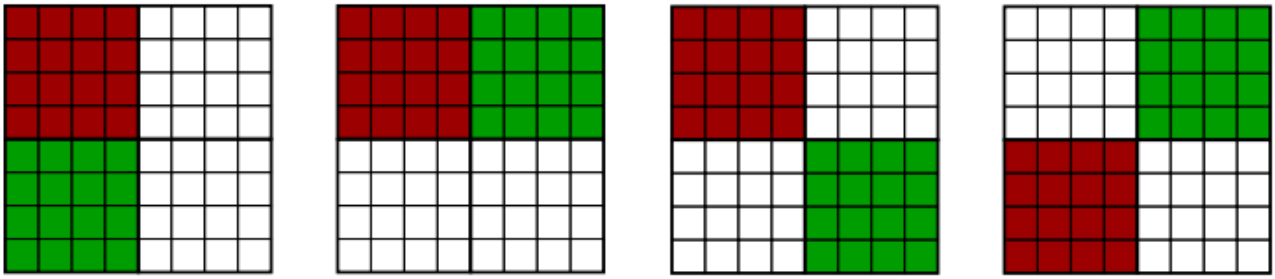

Рис. 1 (показаны блоки 4x4)

Алгоритм проходится по всем смежным блокам и заполняет указанным ранее способом вектор признаков.

Данные признаки полезны при обнаружении пешеходов, так как могут указывать на связность смежных областей картинки при движении.

CSS – color self-similarity

Дескриптор самоподобия цветов — дескриптор, предложенный авторами статьи [2]. Этот дескриптор характеризует глобальные связи между цветами на изображении.

Сначала исходное изображение в цветовом пространстве RGB разбивается на блоки размером 8x8. В каждом из этих блоков определяется средний цвет по каждой цветовой компоненте: по каждой компоненте происходит суммирование, а затем сумма делится на число пикселей в блоке. Получается изображение пониженного разрешения (рис. 2).

Затем изображение пониженного разрешения переводится в цветовое пространство HSV (это цветовое пространство показало наилучшие результаты в работе CSS дескриптора в статье [2]).

После этого начинается непосредственное вычисление вектора признаков.

Между каждыми 2-мя различными пикселями картинки в пространстве HSV вычисляется различие по формуле:

$$diffpixels(a, b) = \frac{\sqrt{(a.H - b.H)^2 + (a.S - b.S)^2 + (a.V - b.V)^2}}{MaxRoot}$$

В данной формуле a и b – пиксели, между которыми находится различие.

H, S, V – цветовые компоненты этих пикселей.

MaxRoot – констранта масштабирования, равная максимальному значению корня в числителе, необходимая для того, чтобы перенести область значений функции *diffpixels(a, b)* в отрезок [0; 1].

Значения функции *diffpixels(a, b)* заносятся в вектор признаков CSS дескриптора для каждой пары различных пикселей a и b.

Рис. 2

Рис. 3

Результаты работы CSS дескриптора легко визуализировать: можно зафиксировать точку на картинке, а потом окрашивать остальные пиксели в зависимости от их цветовой близости к зафиксированному пикселю (рис. 3, зафиксированный пиксель отмечен красным квадратом, чем темнее цвет пикселя изображения — тем ближе цвет этого пикселя к цвету зафиксированного пикселя).

Видно, что CSS дескриптор характеризует глобальные связи между цветами на картинке. Обычно у людей одежда имеет симметричную окраску, так что можно ожидать, что классификатор научится её каким-то образом учитывать при помощи признаков от CSS дескриптора.

Размерность вектора признаков

Реализованный HOG+HOF+CSS дескриптор может вычислять векторы признаков по изображениям разнообразных масштабов, получая самые разные размерности вектора признаков, однако, для стандартного окна поиска размером 128x64 размерность вектора признаков можно указать.

Вклад HOG дескриптора составляет 21060 признаков.

HOF дескриптор с блоком 8x8 выдаёт 840 признаков. Если же уменьшить размер окна, то число признаков возрастёт, однако, такой подход опасен, так как артефакты на изображении начнут оказывать больший вклад в качество распознавания. Увеличивать далее размер блока не имеет смысла.

CSS дескриптор с блоком 8x8 выдаёт 3360 признаков. Увеличивать размер блока не имеет смысла. Если же уменьшать размер блока, то число признаков будет очень быстро расти, так как число признаков квадратично зависит от числа пикселей в изображении пониженной размерности. Однако, хочется избежать непомерного роста вектора признаков, так как иначе придётся обучать классификатор на большем числе тестов.

Таким образом, HOG+HOF+CSS дескриптор формирует вектор признаков из 25260 значений для окна 128x64.

Классификатор

Для классификации был выбран метод опорных векторов (SVM). Во время работы над дескриптором были опробованы 3 реализации SVM, пока не была обнаружена оптимальная для решения поставленных задач.

SVM Light

Этот классификатор ([4]) представляет собой отдельные приложения для обучения и для классификации. Для работы с ним приходилось записывать векторы признаков в файлы в определённом формате.

С помощью этого классификатора были получены некоторые результаты, но время его работы было неадекватно большим даже с учётом того, что векторы считывались из файлов на жёстком диске. На чтение файла с векторами размером 2 GB и обучение SVM уходило примерно 4 часа, а на чтение файла с векторами размером 400 MB и классификацию уходило примерно 1.5 часа. Кроме того, во время тренировки генерировался неадекватно большой файл модели, который тоже требовалось прочитать перед классификацией.

Такая скорость работы классификатора не была удовлетворительной, поэтому SVM было решено сменить.

OpenCV SVM

В этом классификаторе возникли проблемы при масштабировании данных для обучения. Пришлось искать другой классификатор.

LIBLINEAR

В этой хорошо документированной библиотеке ([5]) содержится мощная реализация SVM, которая и была использована в итоге для получения основных результатов. Библиотека была подключена к библиотеке дескриптора, и в код дескриптора была добавлена логика работы с классификатором.

На формирование выборки, сходной по размеру выборке, используемой при тестировании SVM Light, и на обучение по ней у данной реализации SVM уходило примерно 1-3 минуты.

Реализация дескриптора

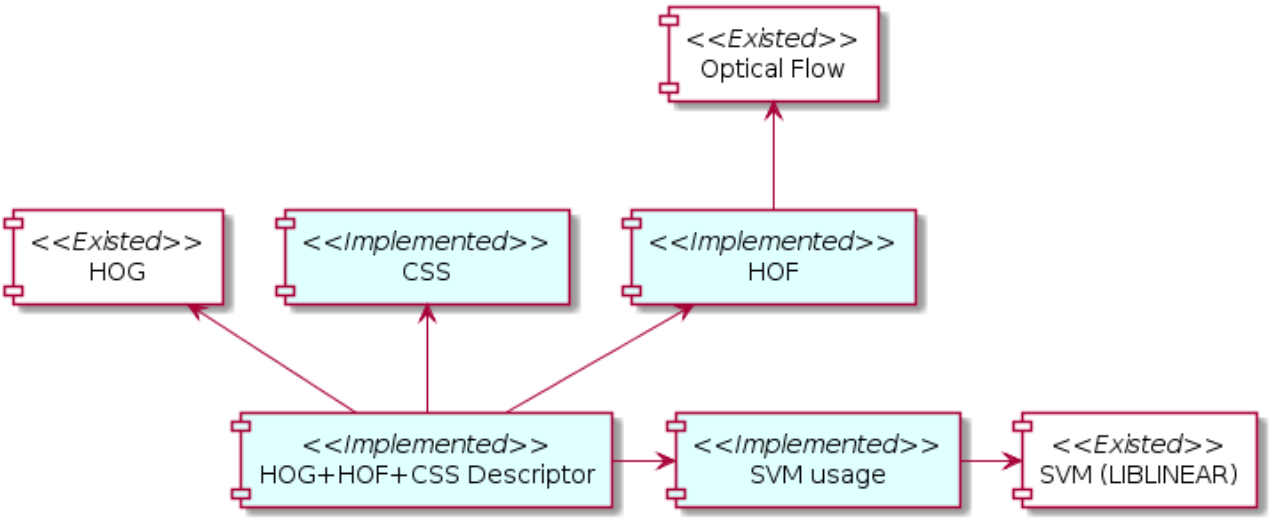

Рис. 4

В качестве языка для реализации библиотеки дескриптора был выбран C++. Разработка велась в Microsoft Visual Studio 2010. Также в процессе работы над дескриптором использовался CMake для генерации проектов OpenCV.

HOG+HOF+CSS дескриптор представляет собой класс, содержащийся в собственной библиотеке. Эта библиотека зависит от библиотек LIBLINEAR, OpenCV и некоторых стандартных библиотек. Для работы с изображениями (загрузки/записи, перевода в другие цветовые пространства и т.д.) были широко использованы средства, предоставленные OpenCV.

Зависимости разных составляющих частей класса HOG+HOF+CSS дескриптора показаны на рис. 4. Также на этом рисунке показано, какие части дескриптора были уже готовыми в сторонних библиотеках, а какие были реализованы с нуля.

Для тестирования дескриптора было создано консольное приложение. Это приложение первоначально было 32-битным, но затем было пересобрано (вместе с библиотеками OpenCV) под 64-битную архитектуру, чтобы увеличить максимальное количество доступной приложению памяти. Это было нужно в первую очередь для того, чтобы можно было разместить в памяти обучающую выборку большего размера.

Также в процессе работы над дескриптором было создано несколько вспомогательных приложений.

Было создано приложение для визуализации работы CSS дескриптора, потому что визуализация легко позволяла проверить корректность работы дескриптора.

Были созданы препроцессоры для различных наборов данных. Эти приложения переводили картинки и информацию о рамках с пешеходами в единый формат входных данных для тестового приложения.

Наборы данных

Для исследования разнообразных алгоритмов, связанных с компьютерным зрением, некоторые исследователи собрали наборы данных. Набор данных представляет собой множество картинок, часто снабжённое информацией о прямоугольных рамках, в которых находятся пешеходы. Для обучения и тестирования HOG+HOF+CSS дескриптора набор данных должен был удовлетворять нескольким требованиям:

- Необходимы цветные изображения, чтобы мог работать CSS дескриптор.
- Изображения должны идти парами, чтобы HOF дескриптор мог работать в принципе, и полученные после его работы признаки имели смысл.
- Должна присутствовать информация о рамках с пешеходами.
- Изображения должны быть представлены в удобном для работы виде.
- Число изображений и уникальных пешеходов в наборе данных должно быть достаточно велико, то есть таково, чтобы можно было максимально заполнить доступную для подгрузки обучающей выборки память.

Были выбраны 2 набора данных, которые удовлетворяли всем требованиям:

- TUD-Brussels [6]
- ETH [7]

Изображения в наборе ETH представляли собой покадровые записи с двух камер, расположенных на движущемся по тротуарам с пешеходами объекте.

Можно предположить, что этот набор данных больше подходит для исследования алгоритмов компьютерного зрения с участием двух камер, однако, все формальные требования к набору данных у набора ETH были выполнены, поэтому было решено на некоторых этапах его использовать.

Набор TUD-Brussels был наиболее важным для тестирования, так как кадры и обстановка на изображениях в наборе были весьма разнообразными по сравнению с кадрами из набора ETH, ограничивающимися какой-то одной городской улицей.

Обучение и тестирование классификатора

После того, как наборы данных были выбраны и перенесены в удобную для работы тестового приложения единую форму, нужно было определить, каким образом из изображений получать отдельные примеры-окна, которые классификатор смог бы разделить на категории «есть пешеход»/«нет пешехода».

Был реализован следующий подход.

Сначала из изображения вырезались положительные рамки, в которых точно находились пешеходы. Это можно было сделать, так как в используемых наборах данных содержалась информация о рамках с пешеходами. Но рамки в большинстве случаев были нестандартного размера, поэтому пришлось их масштабировать, чтобы они точно подходили под окно поиска 128x64.

Затем из изображения вырезались отрицательные рамки, в которых точно не может быть пешеходов. Для этого координаты рамки просто генерировались случайным образом, а затем рамка проверялась на перекрывание с положительными рамками. Если перекрывания не было — окно подходило для обучения дескриптора, иначе координаты генерировались снова. Из каждого изображения бралось несколько случайных отрицательных примеров, и это число регулировалось константой в коде программы.

По каждой из этих положительных и отрицательных рамок дескриптор считал векторы признаков, и эти векторы признаков (вместе с информацией о том, есть пешеход или нет для данного примера) добавлялись в большую структуру данных, которую затем нужно было передать SVM для обучения.

Тестовые векторы генерировались таким же способом, но для них не требовалось построить большую структуру данных, вмещающую все векторы. Поэтому было логично тестировать SVM по одному вектору.

Изображения в наборах данных разделялись на обучающую и на тестовую выборки примерно в пропорции 4:1. При этом обучение и тестирование наборов ETH и TUD-Brussels были независимыми и не пересекались.

Тестирование работы дескриптора растянулось на 3 фазы.

Фаза 1: SVM Light

```
Reading model...OK. (969 support vectors read)
Classifying test examples..100..200..300..400..done
Runtime (without IO) in cpu-seconds: 0.03
Accuracy on test set: 93.50% (374 correct, 26 incorrect, 400 total)
Precision/recall on test set: 94.39%/92.50%
```

Рис. 5 ETH

```
Reading model...OK. (1385 support vectors read)
Classifying test examples..100..200..300..400..500..600..700..done
Runtime (without IO) in cpu-seconds: 0.03
Accuracy on test set: 95.86% (671 correct, 29 incorrect, 700 total)
Precision/recall on test set: 94.76%/90.50%
```

Рис. 6 TUD-Brussels

На этом этапе тестирования в качестве классификатора была использована SVM Light. Модель SVM обучалась по набору из примерно 4000 векторов. Тестирование на наборе данных ETH происходило по 400 векторам, а на наборе TUD-Brussels по 700 векторам.

Для набора ETH была получена (рис. 5) точность классификации 93.50%.

Для набора TUD-Brussels (рис. 6) точность составила 95.86%.

Также была обучена и протестирована SVM по векторам признаков, сформированных HOG дескриптором по набору TUD-Brussels. Было получено значение точности классификации 95.50%.

По одиночным тестам нельзя судить об эффективности работы дескриптора, но многочисленные тесты произвести на SVM Light было невозможно. Поэтому на данном этапе главным обоснованным выводом стало то, что HOG+HOF+CSS дескриптор хоть как-то работает.

Фаза 2: 32-битное приложение с LIBLINEAR

```
41120 41116 0.991895 0.991799
41104 41100 0.991509 0.991413
41114 41114 0.991750 0.991750
41116 41109 0.991799 0.991630
41080 41068 0.990930 0.990641
41090 41092 0.991171 0.991220
41072 41080 0.990737 0.990930
```

Рис. 7 мутационное обучение

```
pos: 397, count: 10575, bad: 93
pos: 398, count: 10585, bad: 95
pos: 399, count: 10595, bad: 90
false pos per image: 74.879997
accuracy: 0.979099
pos: 0, count: 10605, bad: 41
pos: 1, count: 10615, bad: 89
```

Рис. 8 итеративное обучение

После перехода к другой SVM обучение и тестирование ускорились более чем в 100 раз. Также точность классификации увеличилась до 98% на наборе TUD-Brussels. При этом начали появляться мысли о том, что набор ETH не очень-то подходит для задач такого рода. Поэтому роль этого набора была понижена, а основное внимание было уделено набору TUD-Brussels.

Захотелось увеличить точность классификации. Самый простой способ это сделать — увеличить размер обучающей выборки. Однако, при 32-битной архитектуре тестового приложения нельзя было поместить в оперативную память тестов больше чем примерно на 2 GB. Поэтому захотелось опробовать какие-то другие методики обучения.

Первой опробованной методикой стало мутационное обучение. Его суть заключалась в том, что сначала генерировалась большая структура данных, принимаемая SVM для обучения, и по ней обучалась изначальная модель SVM. После этого в структуре данных производились мутации — какие-то векторы заменялись другими, сгенерированными по случайным областям случайных картинок из обучающей выборки. После этого строилась новая модель SVM, и точность обеих имеющихся моделей сравнивалась по случайным окнам из обучающей выборки (тестовую выборку во время обучения трогать нельзя). Если новая модель SVM показывала лучшую точность, то она принималась в качестве основной, и мутации сохранялись, иначе мутации в большой структуре данных для обучения SVM откатывались назад. Такой процесс продолжался указанное заранее время, и в самом конце производилось финальное тестирование полученной модели SVM на тестовой выборке.

На рис. 7 изображён фрагмент вывода тестового приложения. В каждой строчке первые 2 целых числа — количество правильно классифицированных векторов для каждой из моделей SVM, а следующие 2 дробных — точность классификации для каждой из моделей.

Была замечена интересная тенденция. После кратковременного обучения (примерно 2 часа) точность классификации в основном немного повышалась, достигая 98.3%. После длительного обучения (примерно 8 часов) точность классификации резко падала до 93.5%.

Данный способ обучения не дал прироста точности. Объяснение того, почему данный метод оказался плох, было далеко от изначальных целей данной работы, поэтому этот способ обучения был отброшен.

Вторым опробованным методом было итеративное обучение. Суть этого метода заключалась в том, что сначала генерировалась сравнительно небольшая структура данных для обучения SVM, и в неё по ходу обучения добавлялись тесты, оказавшиеся наиболее сложными для классификации. Так продолжалось, пока число векторов в структуре не достигнет определённой отметки. После этого происходило финальное обучение модели SVM, и тестовая выборка классифицировалась.

Для определения сложных для классификации векторов SVM по очереди классифицировала все окна изображения из обучающей выборки, после этого производилась проверка ответа, и, в случае его неправильности, координаты окна заносились в вектор неправильно распознанных окон. Из этого вектора случайным образом выбирались несколько окон, и соответствующие им векторы признаков заносились в структуру данных для обучения SVM.

В процессе итеративного обучения текущая модель SVM переобучалась, чтобы учесть новые векторы признаков. Переобучение после прохода по каждой картинке было слишком медленным. Поэтому было реализовано переобучение после каждого циклического прохода по всем изображениям из обучающей выборки. Также после каждого прохода производилось случайное тестирование по всей обучающей выборке.

На рис.8 показан фрагмент вывода тестового приложения. В каждой строке *pos* — это номер рассматриваемого изображения в обучающей выборке, *count* – число векторов в структуре данных для обучения SVM, *bad* – число ошибочных классификаций при полном разборе окон изображения под номером *pos*.

Также на рис. 8 показана итоговая статистика между разными проходами по обучающей выборке. Показатель *false pos per image* – это среднее число ложноположительных срабатываний («классификатор сказал, что это человек, но это не человек») классификатора по результатам данного прохода по обучающей выборке. Показатель *accuracy* – точность обученной по новым векторам признаков модели при случайном тестировании по обучающей выборке.

Данным методом также не получилось увеличить точность классификации. Была заметна тенденция к ухудшению классификации от прохода к проходу. При этом тенденцию можно было наблюдать как по показателю *accuracy*, так и по показателю *false pos per image*, которые, вообще говоря, считались разными способами.

Данный способ обучения также пришлось отбросить, но причины плохой работы подхода можно было бы изучить.

Фаза 3: 64-битное приложение с LIBLINEAR

Было решено перейти на 64-битную архитектуру, чтобы увеличить точность классификации при помощи увеличения числа векторов, используемых при обучении. Таким путём получилось добиться стабильной точности 99% на наборе TUD-Brussels.

Дальнейшего увеличения точности было бы трудно достичь, поэтому на этом этапе было решено провести уже сравнение дескрипторов. Для этого были зафиксированы координаты рамок для обучения и для тестирования в соответствующих выборках, чтобы дескрипторы оказались в равных условиях. Также было решено попробовать отключать отдельные части HOG+HOF+CSS дескриптора, чтобы увидеть примерный вклад каждой из частей.

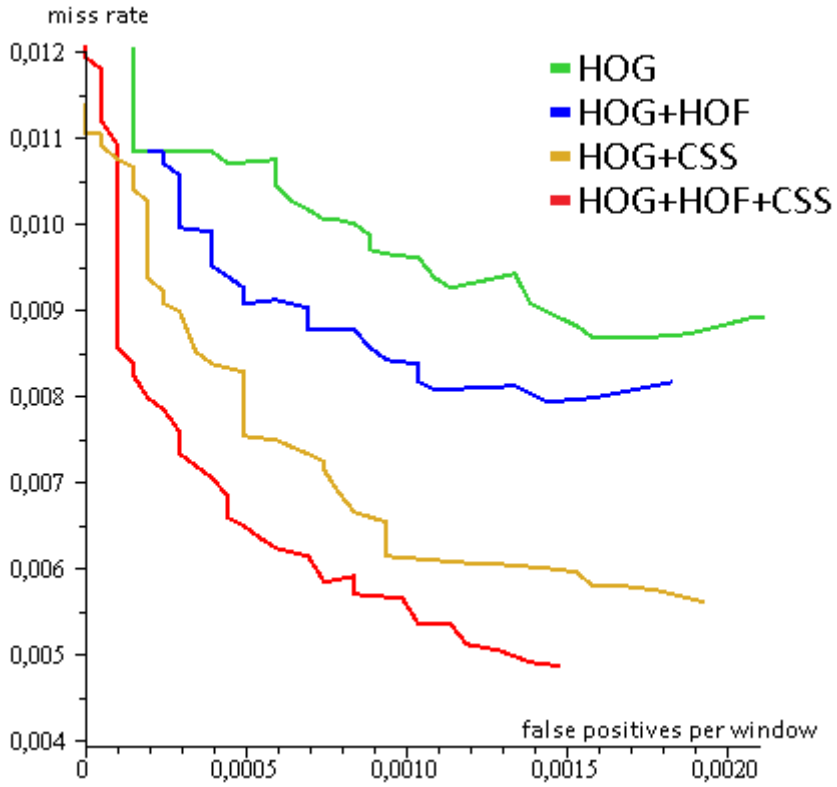

Рис. 9

Был получен график производительности дескрипторов (рис. 9). Расскажу подробнее, как он был получен и что означает.

Классификатор, обучаясь на векторах, полученных от разных дескрипторов, линейно разбивает многомерное пространство (в котором лежат векторы признаков) гиперплоскостью на 2 части. Те векторы, которые попадут в одну часть, будут классифицированы как «вектор класса 1», остальные векторы будут классифицированы как «вектор класса 2».

Во время обучения классификатора можно указать параметр *bias* – расстояние от начала координат, на котором должна находится разделяющая гиперплоскость. Вариации этого расстояния могут сместить классификацию в сторону одного из классов. При этом может пострадать точность классификации, так как алгоритмы обучения SVM проводят близкую к оптимальной гиперплоскость.

Принято смещать гиперплоскость так, чтобы число ложноположительных срабатываний было минимально. Такое смещение нужно, чтобы экран, по которому ходит окно поиска, не был хаотично заполнен рамками, на которых на самом деле нет пешеходов.

На графике отражены показатели ложноположительных срабатываний на одно окно поиска (*false positives per window*) и доля ошибок (*miss rate*). Доля ошибок получается, если из единицы вычесть точность классификации.

Для каждого из 4-х рассматриваемых дескрипторов гиперплоскость понемногу сдвигалась, и для каждого из этих сдвигов считались параметры распознавания. Так и были получены точки на графике.

Очевидно, что чем ниже и левее находится на графике ломаная линия, соответствующая дескриптору, тем данный дескриптор лучше справляется с задачей распознавания пешеходов.

На графике наглядно видно, что HOG+HOF+CSS дескриптор лучше HOG дескриптора в данной задаче, а также видно, что вклад CSS дескриптора больше, чем вклад HOF дескриптора. При этом данные результаты можно считать достоверными, так как ломаные построены в результате множественного обучения и тестирования.

Также на этом этапе был произведён замер скоростей работы отдельных частей HOG+HOF+CSS дескриптора и была построена таблица.

	1 окно, мс	окон в секунду	секунд на изображение
CSS	0.375	2667	2.8
HOG	0.730	1370	5.5
HOF	4	250	30
HOG+HOF+CSS	5.1	196	38

Таблица производительности частей дескриптора

По этой таблице видно, что вычисление признаков CSS дескриптора работает быстрее всего. HOF дескриптор показал наихудшую производительность. Для оценки времени работы дескриптора над изображением полагалось, что в изображении проверяется примерно 7500 окон (800x600 пройти окном поиска с шагом 8 по горизонтали и 8 по вертикали).

Результаты

- Изучена статья [2], и HOG+HOF+CSS дескриптор успешно реализован.
- Опробовано 3 варианта SVM и найден оптимальный.
- Рассмотрено несколько наборов данных.
- Произведено тестирование реализованного дескриптора, были выполнены замеры его характеристик и сравнение с HOG дескриптором.
- Открыты пути для дальнейшей работы по этой теме (поиск причин плохой работы алгоритмов обучения, оптимизация работы HOF дескриптора)

Список литературы

[1] Dollar P. et al. Pedestrian detection: An evaluation of the state of the art //Pattern Analysis and Machine Intelligence, IEEE Transactions on. – 2012. – Т. 34. – №. 4. – С. 743-761

[2] S. Walk, N. Majer, K. Schindler, and B. Schiele, “New features and insights for pedestrian detection,” in
IEEE Conf. Computer Vision and Pattern Recognition, 2010. 8, 11, 12

[3] Navneet Dalal, Bill Triggs, Cordelia Schmid: Human Detection Using Oriented Histograms of Flow and Appearance. ECCV (2) 2006: 428-441

[4] http://svmlight.joachims.org/

[5] http://www.csie.ntu.edu.tw/~cjlin/liblinear/

[6] http://www.d2.mpi-inf.mpg.de/tud-brussels/

[7] http://www.vision.ee.ethz.ch/~aess/dataset/