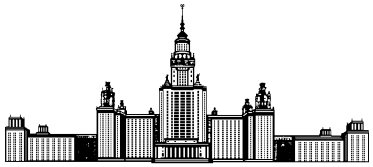
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Автоматизации Систем Вычислительных Комплексов

Лаборатория Компьютерной Графики и Мультимедиа

Курс "МАШИННАЯ ГРАФИКА"

Задание 2: Классификация автодорожных знаков.

Авторы:

Моисеев Борис

Конушин Антон

Москва, 2013

Содержание

Привет!

Как вам известно из курса машинной графики, компьютерное зрение является активно развивающейся областью, имеющей применение в самых разных сферах жизни общества. Компьютерное зрение повсюду: в вашем смартфоне (распознавание QR-кода), фотоаппарате (распознавание лиц), в сервисах, которыми вы регулярно пользуетесь (images.google.com и images.yandex.ru). Здесь не бывает шаблонных решений: каждая задача требует индивидуального подхода, и каждая идея может о казаться ключевой. Здесь используются новейшие технологии, мощнейшие компьютеры, качественные 3D - сканеры, используются громадные массивы данных. В этом задании у вас будет возможность прикоснуться к многообразному и все ещё малоизученному миру компьютерного зрения. Дерзайте!

1 Введение

В этом задании вам предстоит заняться задачей классификации автодорожных знаков. Классификаторы знаков находят свое применение при создании систем помощи водителю на дороге, а также при создании автомобилей, способных обходиться без водителя.

Вот примерная схема работы классификатора:

- Загрузка размеченных изображений.
- Извлечение признаков. Каждое изображение описывается некоторым множеством признаков, называемым дескриптором. Мы будем использовать дескриптор на основе гистограмм ориентированных градиентов, или HOG (не пугайтесь, алгоритм подробно описан ниже, его вам и предстоит реализовать).
- Обучение классификатора. Классификатор - некая подпрограмма, которая умеет, обучаясь на переданных нами данных, отличать изображения разных классов друг от друга. При этом на вход классификатору поступают именно дескрипторы картинок.
- Тестирование классификатора на контрольной выборке.

2 Загрузка изображений

Загрузка изображений уже реализована в шаблоне, этот пункт можно пропустить.

В задании используются только автодорожные знаки ограничения скорости, всего 7 классов. Их стоит скачать и распаковать в корневую директорию этого проекта. Функции загрузки данных уже реализованы, поэтому вы можете пропустить этот раздел.

Данные разбиты на обучающую и тестовую выборки, каждая из которых в свою очередь состоит из двух частей: собственно фотографий знаков (поддиректория data/train и data/test соответственно) и списка файлов и соответствующих меток

классов (data/train_labels.txt и data/test_labels.txt). Все изображения имеют размер 32x32 пикселя и находятся в RGB формате.

3 Гистограммы ориентированных градиентов

Подробно об этом дескрипторе можно прочитать на википедии , а также в работе Навнита Далала и Билла Триггса. Здесь же мы кратко опишем суть реализуемого алгоритма.

3.1 Преобразование изображения из цветного в оттенки серого (grayscale).

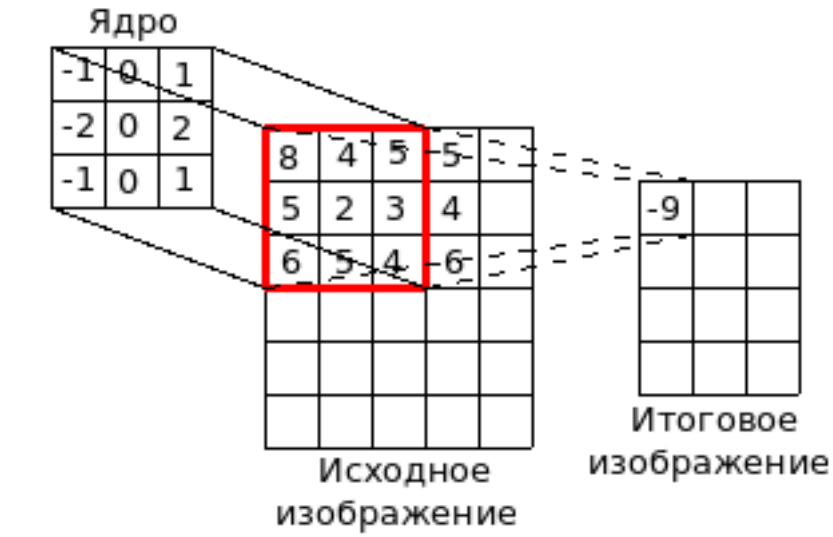
Яркость пикселя grayscale-изображения рассчитывается по следующей формуле:

$$Y = 0.299R + 0.587G + 0.114B$$

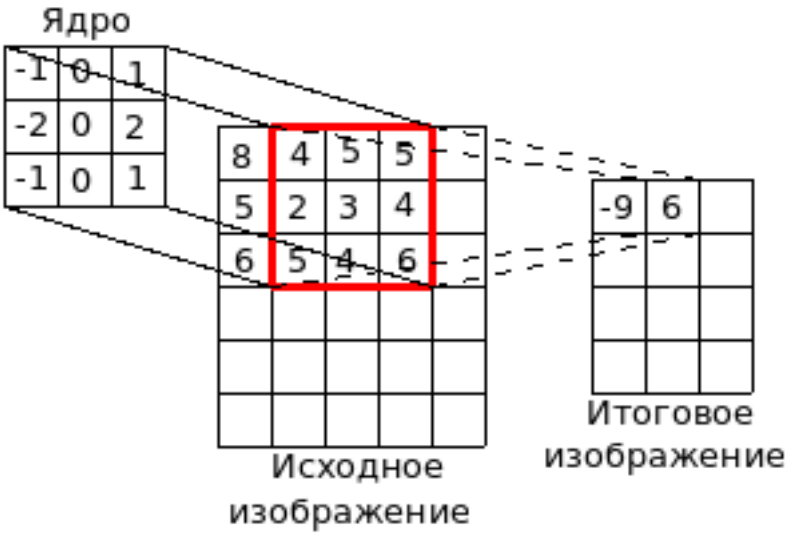
3.2 Свертка с фильтрами Собеля

Сначала опишем основные понятия.

Понятие **свертки** изображений проиллюстрированно на приведенных рисунках, подробнее можно прочитать на википедии. Там же можно прочитать и о различных способах обработки границ изображения чтобы выбрать оптимальный.



$$-1 \times 8 + -2 \times 5 + -1 \times 6 + 0 \times 4 + 0 \times 2 + \\ + 0 \times 5 + 1 \times 5 + 2 \times 3 + 1 \times 4 = -9$$



$$-1 \times 4 + -2 \times 2 + -1 \times 5 + 0 \times 5 + 0 \times 3 + \\ + 0 \times 4 + 1 \times 5 + 2 \times 4 + 1 \times 6 = 6$$

Фильтром Собеля называется свертка изображения с ядром $(-1; 0; 1)$ (горизонтальный фильтр Собеля) и $\begin{pmatrix} 1 \\ 0 \\ -1 \end{pmatrix}$ (вертикальный фильтр Собеля).

Градиент яркости пикселей — вектор роста яркости писелей. Направление вектора показывает, в каком направлении от данного пикселя изображение становится светлее, а модуль вектора — на сколько светлее.

Итак, второй шаг алгоритма заключается в свертке полученного после прошлого шага grayscale-изображения с горизонтальным и вертикальным фильтром Собеля, получая два изображения-матрицы. Каждый элемент первой матрицы равен горизонтальной составляющей градиента яркости пикселей, а второй — вертикальной.

3.3 Вычисление градиента в каждом пикселе

В результате предыдущего шага мы получили две матрицы, одна из которых содержит горизонтальную составляющую градиента для каждого пикселя, другая — вертикальную. Посчитать по этой информации модуль и направление градиента яркости в каждой точке не должно составить для вас труда.

3.4 Вычисление гистограмм градиентов

В результате предыдущего шага мы получили матрицу направлений и матрицу значений градиентов яркости пикселей. На этом шаге нужно разбить изображение на прямоугольники (клетки) и вычислить гистограмму градиентов в каждой клетке.

Для начала разобьем всю область изменения направления градиента (например, $[-\pi, \pi]$) на некоторое (обычно не очень большое, порядка 8-32) число сегментов. Выберем некоторую клетку и создадим для неё массив размера, равного количеству сегментов. Для каждого пикселя клетки посчитаем, в какой сегмент попадает направление градиента в этом пикселе и прибавим значение модуля градиента в соответствующую ячейку массива. В результате мы получили гистограмму ориентированных градиентов пикселей выбранной клетки. Повторим процедуру получения гистограммы для всех клеток.

3.5 Нормализация гистограмм

В зависимости от контрастности изображения значения градиентов могут существенно различаться. Поэтому важно нормализовать гистограммы, чтобы свойства освещения не влияли на результаты.

Каждая гистограмма по сути является вектором — упорядоченным набором из нескольких чисел. Нормализация гистограммы — это нормирование вектора, т.е. деление всех его значений на некоторую норму этого вектора. Нормы могут быть разные, но проще всего воспользоваться евклидовой нормой (корень из суммы квадратов элементов). Если же вы хотите добиться лучших результатов, то можете попробовать выбрать другую норму. Также может привести к успеху нормализация не одной клетки, а нескольких вместе (как длинного вектора, полученного конкатенацией гистограмм каждой клетки)

3.6 Объединение гистограмм в дескриптор

Все, что осталось сделать — это соединить полученные нормализованные гистограммы в один массив, получив тем самым дескриптор — вектор признаков, который описывает некоторые характерные признаки изображения. Именно с такими признаками, вместо самих изображений, будет работать классификатор.

4 Требования к заданию

- Задание должно быть полностью выполнено только студентом, который указан в readme.txt как автор работы. Запрещается публиковать, распространять текст выполненного задания до окончания срока приема работ (жесткого дедлайна).
- Обязательной частью задания является реализация дескриптора HOG согласно алгоритму, описанному в соответствующем разделе. Допускается изменение отдельных деталей, например, применение нормализации по блокам из нескольких клеток или и использование другой версии фильтра Собеля, однако основные шаги алгоритма (приведение к grayscale-формату, свертка с фильтрами Собеля, рассчет направления и значения градиента, рассчет и нормализация гистограмм) должны быть выполнены.
- Точность классификации на приведенной тестовой выборке должна быть не ниже 50%.

- Работа выполняется на языке C++.
- Работа должна успешно компилироваться и собираться с помощью компилятора gcc версии 4.8 вне зависимости от платформы. Работать можно на любой платформе, но важно не использовать платформозависимых библиотек, например windows.h, unistd.h и прочих.
- Запрещается использовать сторонние библиотеки помимо тех, что даны в шаблоне задания.
- Для обучения модели должна использоваться только приведенная обучающая выборка без каких-либо модификаций.
- Время обучения модели не должно превышать 30 минут на ноутбуке Dell Latitude E4310 (его характеристики можно найти в интернете). Время работы классификатора на тестовой выборке не должно привышать 10 секунд.

5 Формат выполненной работы и критерии оценки.

В качестве результата своей работы вы должны предоставить архив в формате tar (можно со сжатием, например tar.gz), внутри которого сразу располагается корневая директория проекта (аналогично архиву с шаблоном), в которой располагается (помимо файлов с исходным кодом) файл с обученной моделью и заполненный readme.txt.

Все добавленные вами файлы исходного кода должны быть добавлены в CMakeLists.txt для корректной сборки с помощью cmake (см. ниже).

Файл отчета readme.txt должен содержать ФИО автора, номер группы, описание выполненных дополнительных частей работы. В случае реализации методов, отличных от описанных в этом руководстве, нужно кратко объяснить принципы их работы.

Выполненная базовая часть работы оценивается в 15 баллов. При невыполнении одного или нескольких пунктов требований работа будет оценена в 0 баллов. При нарушении первого требования к заданию работа будет оценена в -5 баллов. Баллы за дополнительные части задания начисляются только при условии успешного

выполнения базовой части. Колличество начисленных за задание баллов не может превышать 20. При отправке работы после срока с каждым днем опоздания будет начисляться штраф в 1 балл, кроме первых двух дней, когда штраф составит 0.5 балла в день. Для решения спорных вопросов и ситуаций предусмотрена апелляция, время и место проведения которой определяется проверяющими для каждой группы отдельно. Также на апелляцию могут быть приглашены студенты, авторство работ которых вызывает сомнения.

6 Сборка работы

Для того, чтобы собрать шаблонный классификатор, нужно выполнить следующие шаги:

- Установить программу *cmake* (с официального сайта или из репозиториев).
- Установить компилятор *g++* (*mingw* для пользователей Windows)
- Распаковать архив с шаблоном задания
- Распаковать в корень полученной директории архив с обучающими и тестовыми данными
- Создать в корне проекта директорию *build*
- Открыть консоль и перейти в директорию *build*
- Выполнить

  ```
  cmake -G <тип проекта> ../
  ```

  Допустимые типы проектов:

  Для *nix: *"Unix Makefiles"*.

  Для Windows *"MinGW Makefiles"*.

  Можете использовать и другие типы проектов на свой страх и риск (например, *"Visual Studio 10"*, полный список можно получить с помощью *cmake –help*

- использовать полученный проект. Для сборки в Linux использовать команду make в директории build, в Windows — mingw-make. В случае генерации проекта Visual Studio его можно использовать непосредственно с помощью упомянутой IDE.

Если вы добавляете новые файлы в проект, их следует добавить в список файлов для сборки главного проекта *cmake/main.cmake*

Опции запуска собранной программы можно посмотреть, запустив её с опцией *--help*. Примеры:

Обучить модель:

```
task2.exe -d ../data/train_labels.txt -m model.txt --train
```

Классифицировать изображения из тестовой выборки с помощью обученного классификатора:

```
task2.exe -d ../data/test_labels.txt -m model.txt -l predictions.txt --predict
```

Проверка точности классификации:

```
task2_test.exe -g ../data/test_labels.txt -p predictions.txt
```

7 Детали реализации

В данном разделе содержится информация о некоторых деталях реализации шаблона.

Загруженные изображения хранятся в виде объектов класса BMP. Обращение к ним осуществляется следующим образом:

```
RGBApixel pixel = image.GetPixel(0,0);
int s = pixel.Red + pixel.Blue + pixel.Green;
```

Основные используемые структуры данных описаны в начале файла main.cpp:

```
typedef vector<pair<BMP*, int> > TDataSet;
typedef vector<pair<string, int> > TFileList;
typedef vector<pair<vector<float>, int> > TFeatures;
```

TDataSet — вектор, состоящий из пар "Указатель на изображение, Метка класса".

TFileList — вектор, состоящий из пар "Имя файла с картинкой, Метка класса".

TFeatures — вектор, состоящий из пар "Вектор признаков, Метка класса".

8 Дополнительные задания

8.1 Повышенное качество классификации (до +5 баллов)

Все полученные решения подвергаются тестированию на скрытой тестовой выборке, и в результате составляется рейтинг классификаторов, которые удалось получить.

- Лучшая работа получит +5 баллов.
- Работы, занявшие 2-е и 3-е места получат +4 балла.
- Работы, попавшие в число 10% лучших (среди работ, получивших положительную оценку), получат +3 балла.
- Работы, попавшие в число 25% лучших (среди работ, получивших положительную оценку), получат +2 балла.
- Работы, попавшие в число 50% лучших (среди работ, получивших положительную оценку), получат +1 балл.

Существует много способов повышения качества классификации, например подбор оптимальных параметров разбиения изображения на клетки при подсчёте HOG изображения, использование других, дополнительных признаков. Проявите фантазию!

В качестве ориентира стоит отметить, что один из авторов задания добился точности в 93,7% всего лишь примерно за полчаса работы.

8.2 Нелинейные ядра SVM (+2 балла)

В этом задании необходимо реализовать нелинейное ядро для классификатора SVM. Подробнее об этом задании можно прочитать в прошлогоднем задании по машинной графике здесь.

8.3 Детекция автодорожных знаков (до +5 баллов)

В этом задании необходимо реализовать детекцию (т.е. нахождение) автодорожных знаков на фотографии. При этом классифицировать найденные знаки не нужно. Данные для обучения классификатора можно найти здесь

В readme нужно описать вид команды для обучения детектора, а также приложить обученную модель. Детекция знаков должны осуществляться командой:

```
task2.exe -d file_list.txt -m model.txt --detect
```

Здесь *file_ list.txt* — файл со списком изображений, на которых нужно произвести детекцию знаков, *model.txt* — файл с обученной моделью. В результате на файлах для детекции знаки должны быть обведены в рамку. Также стоит обратить внимание на подавление немаксимумов — детектированный знак должен быть окружен только одной рамкой, наложение её с соседними не должно превосходить половины площади. При отсутствии подавления немаксимумов за задание можно получить не более +3 баллов. Это задание — творческое, поэтому конкретные способы реализации вам предлагается найти или придумать самостоятельно.

Время работы алгоритма на одном изображении должно оставаться в пределах 10 секунд, точность работы определяется проверяющим визуально (но она должна быть определенно выше случайной).

9 Часто задаваемые вопросы

- **Можно ли использовать класс для работы с матрицами из первого задания?**

  Да, можно. Для этой библиотеки было решено сделать исключение.

- **Нужно ли реализовывать классификатор, или достаточно реализовать дескриптор HOG?**

  Результатом выполнения задания должен быть классификатор — программа, способная классифицировать изображения автодорожных знаков будучи запущенной с помощью команд, указанных выше. Однако в шаблоне задания реализованы все части классификатора, кроме дескриптора HOG, поэтому обе задачи фактически эквивалентны.