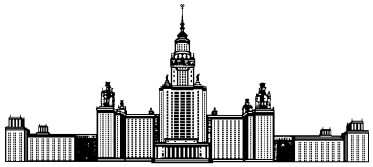
Московский государственный университет имени М. В. Ломоносова



Факультет Вычислительной Математики и Кибернетики

Кафедра Автоматизации Систем Вычислительных Комплексов

Лаборатория Компьютерной Графики и Мультимедиа

Курс "МАШИННАЯ ГРАФИКА"

Задание 2: Классификация объектов.

Авторы:

Данил Галеев

Влад Шахуро

Антон Конушин

Москва, 2014

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 бинарной классификацией. Все входные данные делятся на два класса: изображения, на которых есть самолет, и изображения, на которых его нет. Ваша задача - обучить классификатор таким образом, чтобы на новом входном изображении он смог определить наличие на нем самолета.
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.
- 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

Изображения нужно скачать и распаковать в корневую директорию этого проекта. Функции загрузки данных уже реализованы, поэтому вы можете пропустить этот раздел.

Данные для базовой части задания (бинарной классификации) расположены в подпапке data/binary.

Данные разбиты на обучающую и тестовую выборки, каждая из которых в свою очередь состоит из двух частей: собственно фотографий (поддиректория data/binary/train и data/binary/test соответственно) и списка файлов и соответствующих меток классов (data/binary/train_labels.txt и data/binary/test_labels.txt). Обратите внимание, изображения имеют различный размер.

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).

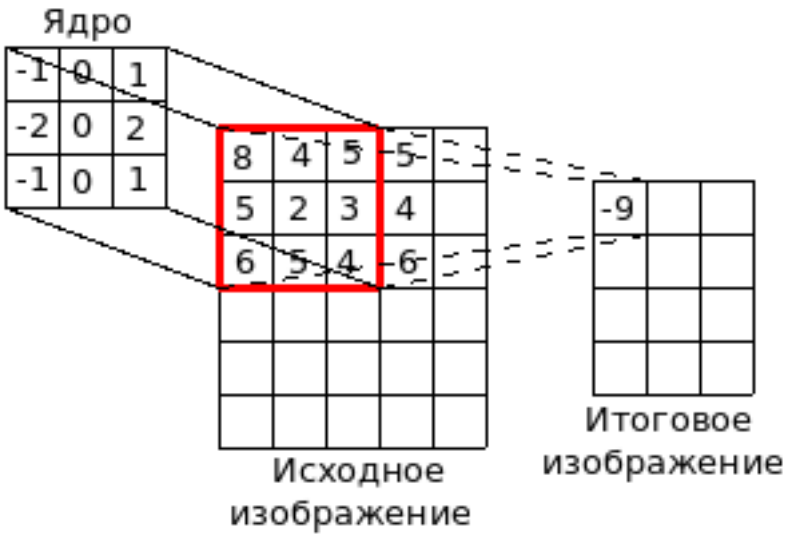
Яркость пикселя grayscale-изображения рассчитывается по следующей формуле:

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

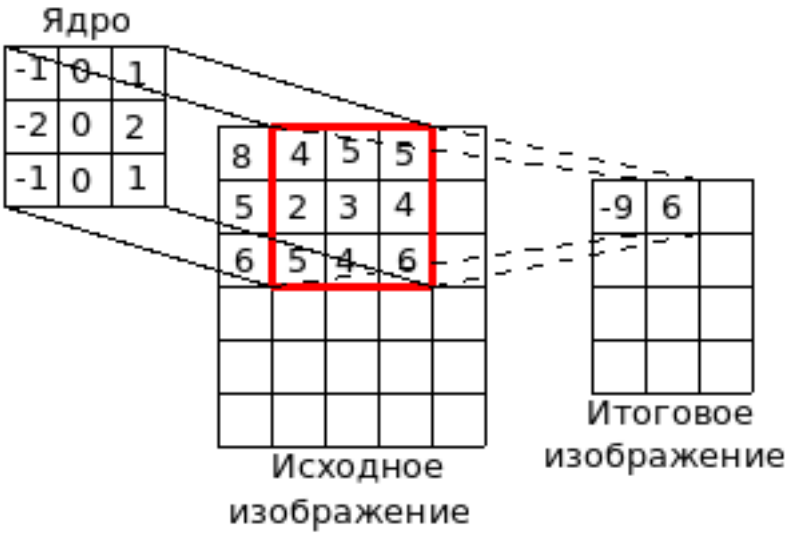
3.2 Свертка с фильтрами Собеля

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 x 8 + -2 x 5 + -1 x 6 + 0 x 4 + 0 x 2 +
+ 0 x 5 + 1 x 5 + 2 x 3 + 1 x 4 = -9



-1 x 4 + -2 x 2 + -1 x 5 + 0 x 5 + 0 x 3 +
+ 0 x 4 + 1 x 5 + 2 x 4 + 1 x 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 (конкатенировать, записать последовательно) полученные нормализованные гистограммы в один длинный массив, получив тем самым дескриптор — вектор признаков, который описывает некоторые характерные признаки изображения. Именно с такими признаками, вместо самих изображений, будет работать классификатор.

4 Среда программирования и устройство каркаса

Как и в первом задании, пользователям Windows рекомендуется использовать Cygwin. Пользователям Linux достаточно иметь gcc и GNU make.

Структура каркаса

- src — директория с исходным кодом в формате .cpp;
- include — директория с заголовочными файлами. Здесь имеются файлы `matrix.h` и `matrix.hpp` - реализация класса матрицы из первого задания;
- externals — директория с исходными кодами библиотек. Вам предоставлены три библиотеки, которые помогут вам в решении задачи: `EasyBMP`, `Liblinear` и `Argvparser`;
- bridge — директория, в которую добавляются заголовочные файлы и скомпилированные библиотеки для импорта в основной проект;

- build/bin — директория, в которой сохраняются выполняемые файлы компиляции (т.е. после того, как отработала команда make all);

- compare.py — скрипт для проверки точности работы классификатора. Запускается командой `./compare.py <test_labels.txt> <predictions.txt>`. Скрипт выводит отношение количества верно классифицированных тестовых изображений к общему числу тестовых изображений;

5 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, расчет 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 превышать 10 секунд.

6 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.

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 5 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. Количество начисленных за задание баллов не может превышать 15. Задание нужно сдать до 29 октября 23:59 (мягкий дедлайн). 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. После 5 ноября 23:59 работы приниматься не будут (жесткий дедлайн). 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.

7 Сборка работы

Разархивируйте cygwin.exe и запустите сценарий Cygwin.bat. Откроется консоль и вы окажетесь в папке /home/<username>.

Скопируйте в эту папку архив task2_project.tar.gz с каркасом задания и разархивируйте командой `tar xvf task2_project.tar.gz`. Разархивируйте в корень полученной директории архив с обучающими и тестовыми данными.

Зайдите в папку проекта, введя в консоли команду `cd task2_project`. Запустите сценарий компиляции командой `make all`. В папке build/bin появятся исполняемые файлы. Их можно запустить из консоли командой `./build/bin/<имя файла>` . Удалить все скомпилированные файлы можно командой `make clean`

Если вы добавляете новые файлы в проект, то заголовочные файлы следует поместить в папке `include/`, файлы исходного кода - в папке `src/`

Примеры:

Обучить модель:

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

Классифицировать изображения из тестовой выборки с помощью обученного классификатора:

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

Проверка точности классификации:

```
./compare.py data/binary/test_labels.txt build/bin/predictions.txt
```

8 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 task2.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".

9 Дополнительные задания

9.1 Многоклассовая классификация (+2 балла)

Бинарный классификатор из базовой части без изменения кода программы должен работать и в случае большего числа классов. Вам необходимо обучить модель на примерах из подпапки `data/multiclass`:

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

Далее, классифицировать изображения из тестовой выборки с помощью обученного классификатора:

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

И проверить точность классификатора, она должна быть не ниже 50%

```
./compare.py data/multiclass/test_labels.txt build/bin/predictions.txt
```

Файл с обученной моделью model_multiclass.txt следует поместить в корень проекта.

9.2 Нелинейные ядра SVM (+3 балла)

В этом задании необходимо реализовать нелинейное ядро для классификатора SVM.

В линейном SVM в качестве функции расстояния между векторами признаков неявно используется скалярное произведение в Евклидовом пространстве $\langle \cdot, \cdot \rangle$. Это не лучший вариант для признаков-гистограмм, поэтому совместно с HOG часто используют нелинейный SVM с χ^2 ядром: $K(\mathbf{x}, \mathbf{y}) = \sum_d k(x_d, y_d) = \sum_d \frac{x_d y_d}{x_d + y_d}$, где суммирование идёт по всем компонентам векторов. Проблема в том, что сложность обучения становится вместо линейной по числу прецедентов, кубической. К счастью, благодаря специальным свойствам ядра χ^2 (аддитивное, гомогенное), существует такое преобразование признакового пространства $\mathbf{\Psi}(\mathbf{x})$, что $K(\mathbf{x}, \mathbf{y}) = \langle \mathbf{\Psi}(\mathbf{x}), \mathbf{\Psi}(\mathbf{y}) \rangle$. Таким образом, можно сначала преобразовать признаки, а потом передавать их на вход линейному SVM.

Для каждой компоненты вектора $\mathbf{x}$ преобразование выглядит как $[\Psi(x)]_\lambda = \mathrm{e}^{-i\lambda \log x} \sqrt{x\kappa(\lambda)}$, где в случае χ^2 ядра $\kappa(\lambda) = \mathrm{sech}(\pi\lambda)$. Чтобы получить точное значение, необходимо взять конкатенацию значений этой функции по всем вещественным λ, но тогда вектор признаков будет иметь бесконечную размерность. Поэтому берут его значения в нескольких точках: $-nL, (-n+1)L, \ldots, nL$, где n - порядок аппроксимации (обычно достаточно 1 или 2), а L — шаг аппроксимации, который нужно подбирать по валидационной выборке или на скользящем контроле (начать можно со значений 0.25..0.5). Поскольку значения функции комплексные, вещественные и мнимые части рассматриваются отдельно и конкатенируются (при этом размер вектора вырастает ещё в два раза). Подробнее техника описана в статье

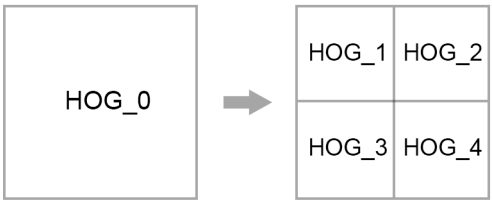
- A. Vedaldi and A. Zisserman, "Efficient Additive Kernels via Explicit Feature Maps," IEEE Conference on Computer Vision and Pattern Recognition, 2010.

9.3 Пирамида дескрипторов (+3 балла)

Пирамида дескрипторов строится следующим образом:

- Изображение делится на 4 части.

- В каждой части считается дескриптор HOG.
- Все дескрипторы конкатенируются. (1 дескриптор HOG всего изображения + 4 дескриптора HOG частей изображения)



descriptor = [HOG_0, HOG_1, HOG_2, HOG_3, HOG_4]

9.4 Цветовые признаки (+3 балла)

В этом задании вам необходимо использовать цветовые признаки, чтобы улучшить качество классификатора.

Извлекать цветовые признаки можно следующим образом:

- Изображение делится на 64 блока (8 на 8)
- В каждом блоке считается средний цвет (три значения - среднее для каждого из каналов)
- Все значения вытягиваются в один вектор и приводятся к отрезку [0, 1] (необходимо поделить на 255)

- Полученный вектор конкатенируется с вектором HOG

10 Часто задаваемые вопросы

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

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

- **Как соединить гистограммы для отдельных клеток в дескриптор изображения?**

  Нужно просто конкатенировать, т.е. записать последовательно, друг за другом, эти гистограммы, в один массив.