



Нижегородский государственный университет им. Н.И. Лобачевского

***Разработка мультимедийных приложений
с использованием библиотек OpenCV и IPP***

Лабораторная работа
Детектирование пешеходов

Дружков П.Н., кафедра математической
логики и высшей алгебры, факультет ВМК

Содержание

- ❑ Цели и задачи работы.
- ❑ Описание некоторых подходов к детектированию пешеходов на изображениях:
 - HOG-признаки.
 - Метод бегущего окна и HOG-детектор.
 - LatentSVM.
- ❑ Разработка приложения для решения задачи детектирования пешеходов.

Цели работы

- ❑ Познакомиться с некоторыми подходами к детектированию пешеходов на изображениях.
- ❑ Научиться использовать программные реализации компонентов детекторов и готовые детекторы из библиотеки OpenCV.

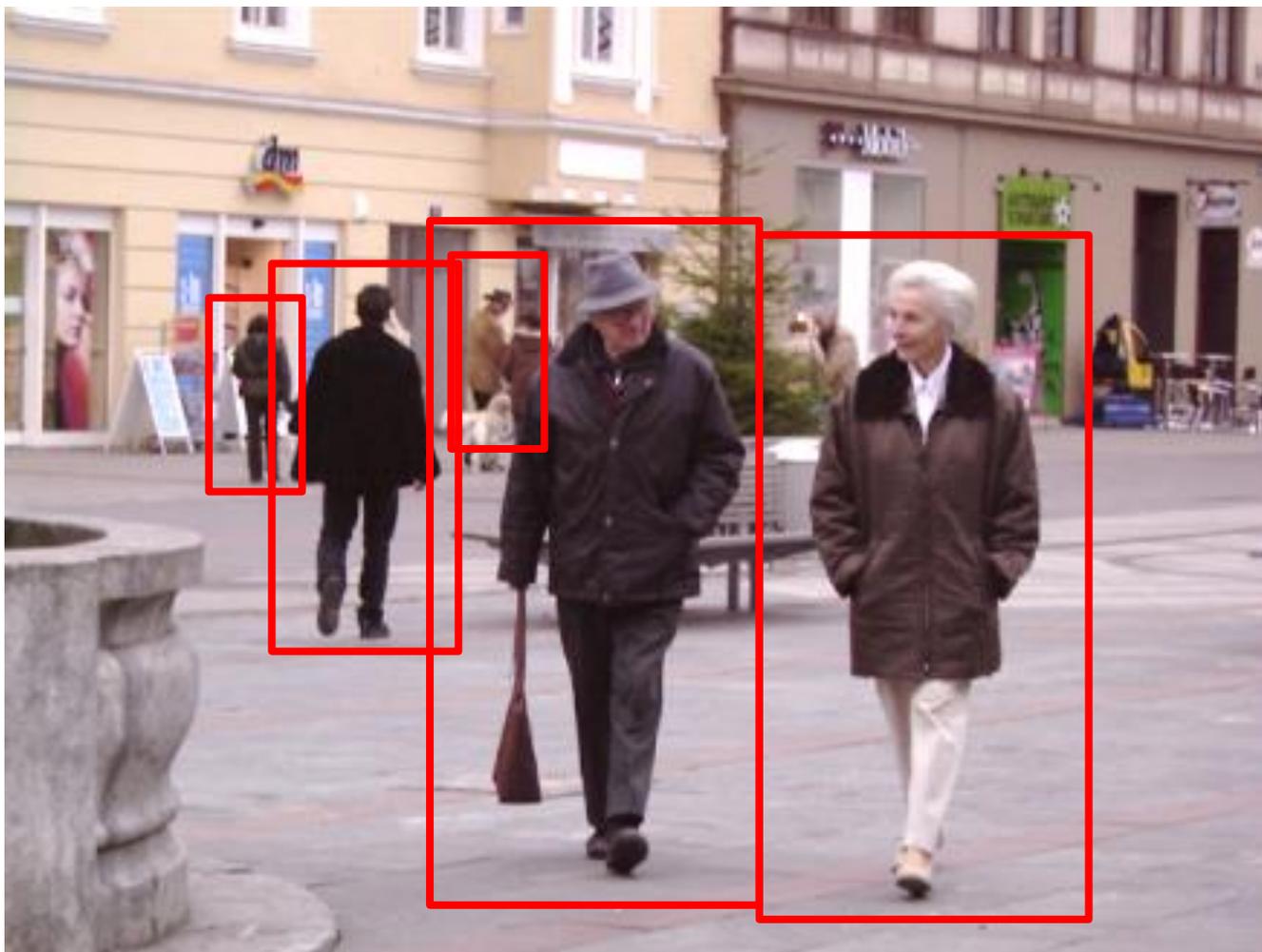
Задачи работы

- ❑ Изучить алгоритм вычисления HOG-признаков.
- ❑ Изучить принципы работы детектора пешеходов, основанного на использовании HOG-признаков.
- ❑ Рассмотреть интерфейс класса, реализующего HOG-дескриптор и HOG-детектор, в библиотеке OpenCV.
- ❑ Рассмотреть интерфейс класса, реализующего детектор LatentSVM.

ОПИСАНИЕ ВОЗМОЖНОСТЕЙ БИБЛИОТЕКИ ОРЕНСВ ПО ДЕТЕКТИРОВАНИЮ ПЕШЕХОДОВ



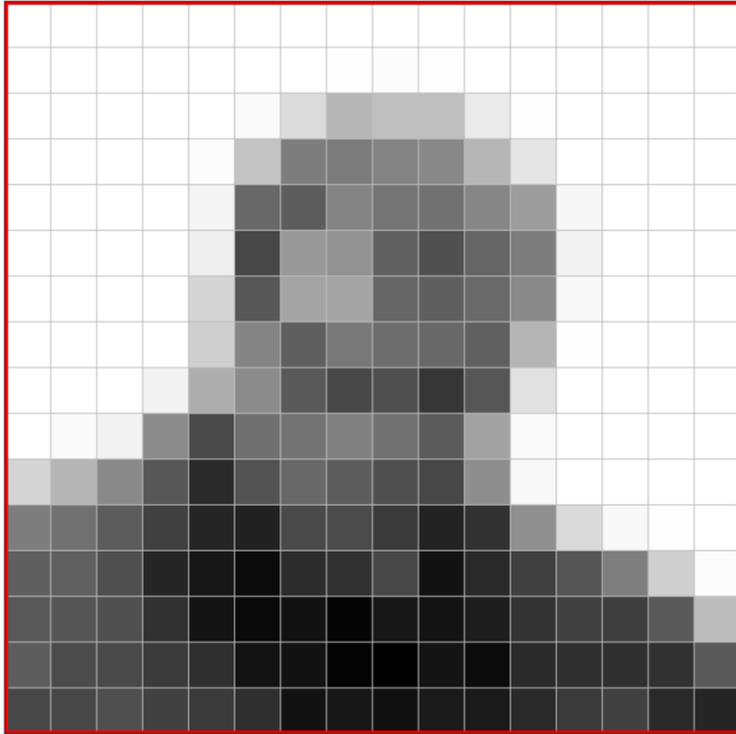
Задача детектирования пешеходов на изображении



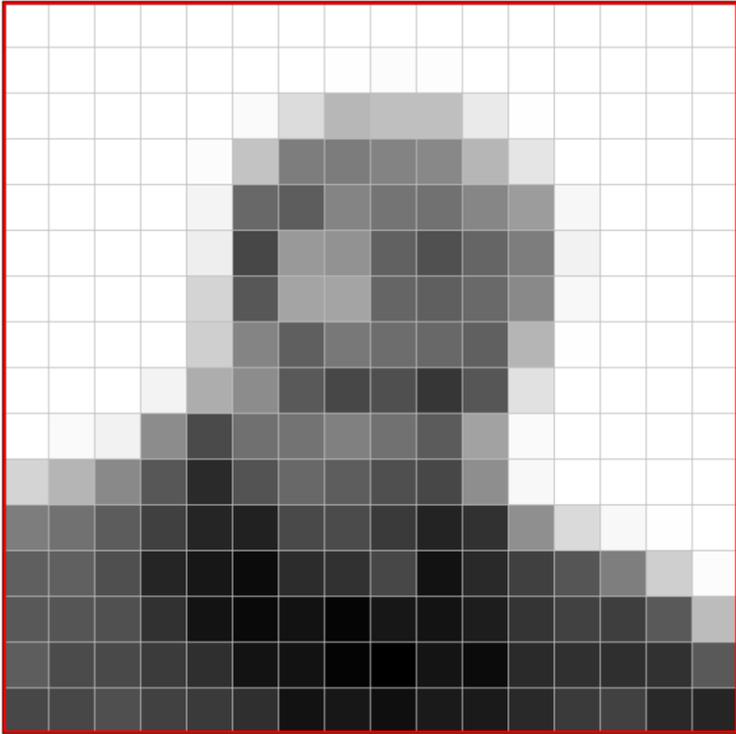
НОG-дескриптор. Алгоритм



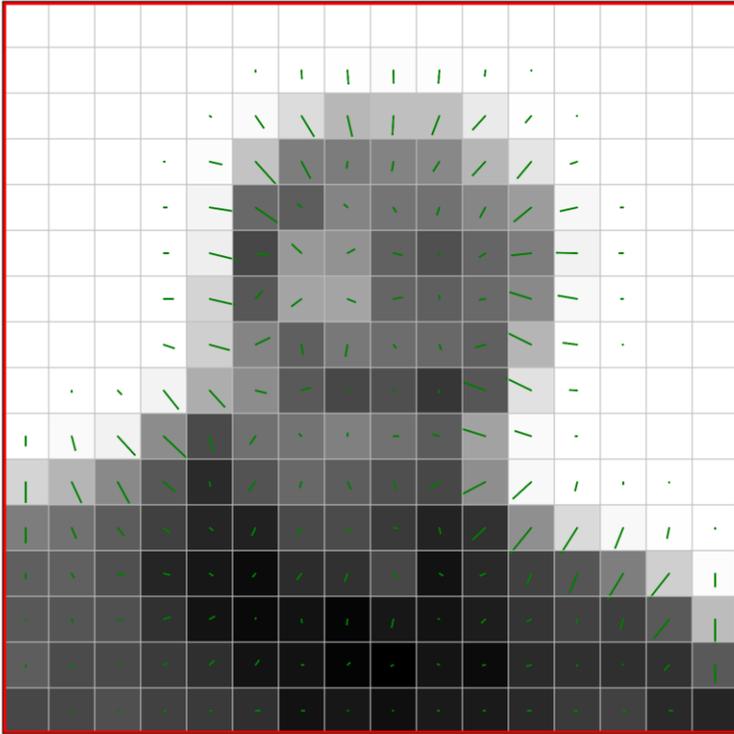
НОG-дескриптор. Алгоритм



НОG-дескриптор. Алгоритм

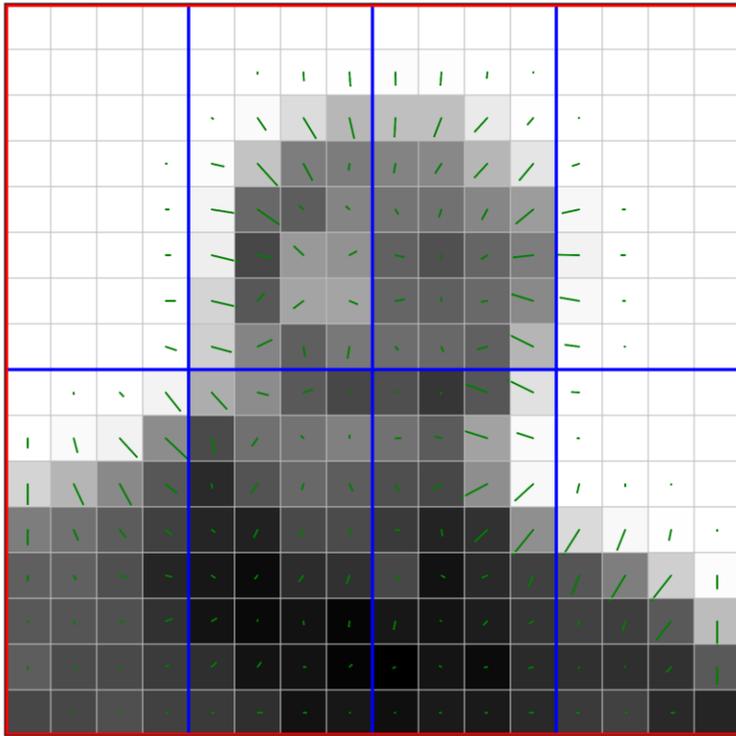


HOG-дескриптор. Алгоритм

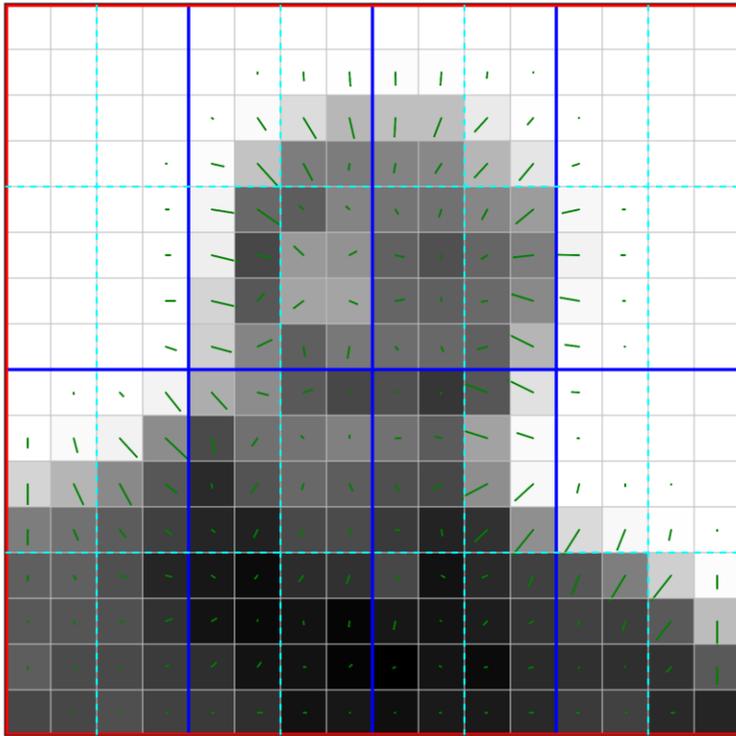


$$\begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$

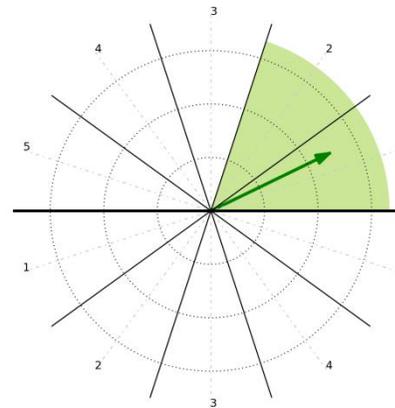
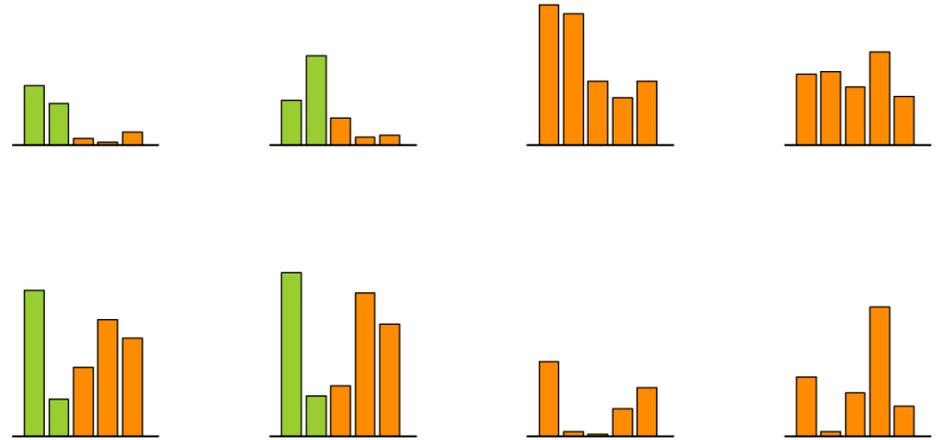
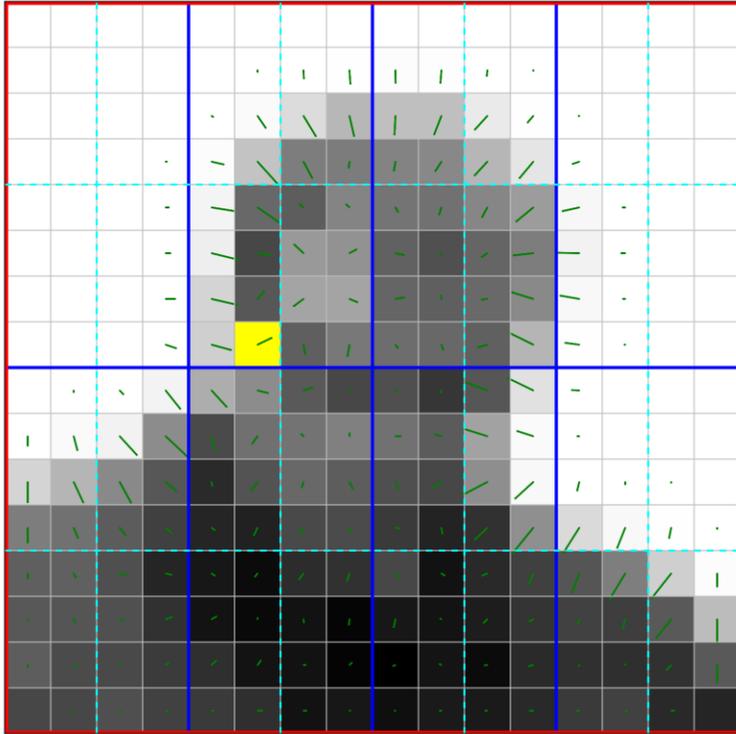
НОG-дескриптор. Алгоритм



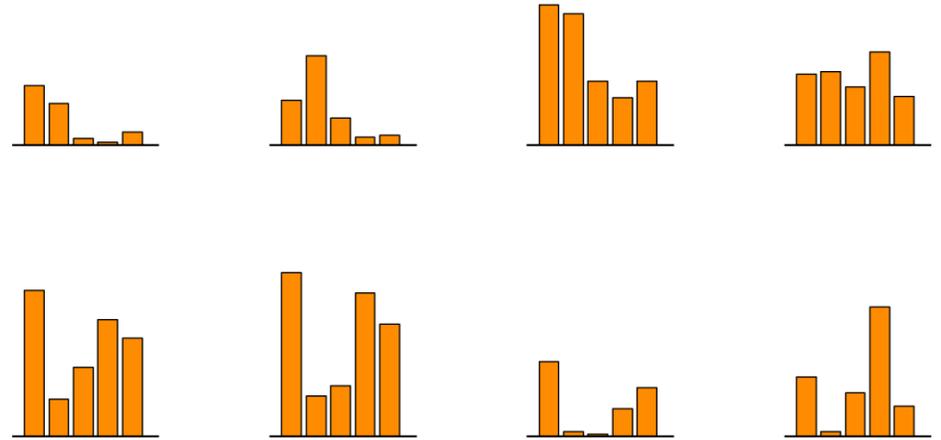
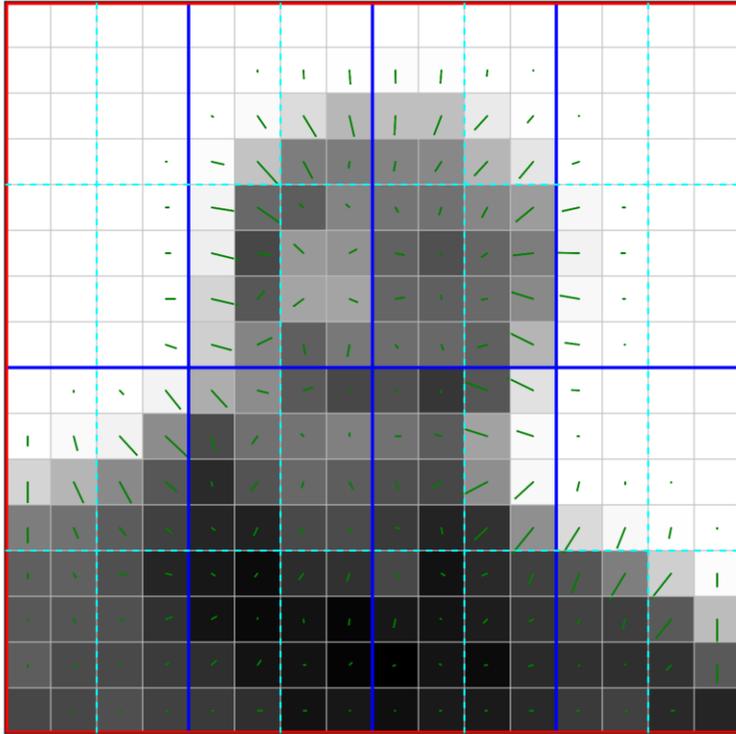
НОG-дескриптор. Алгоритм



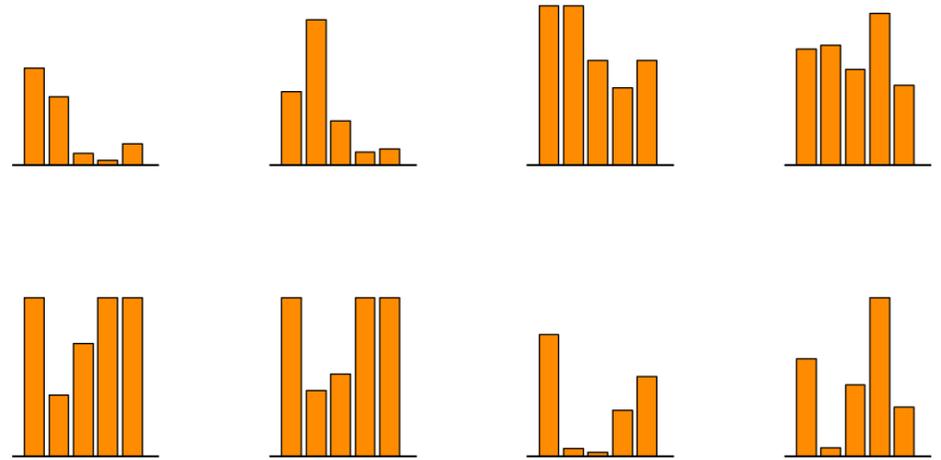
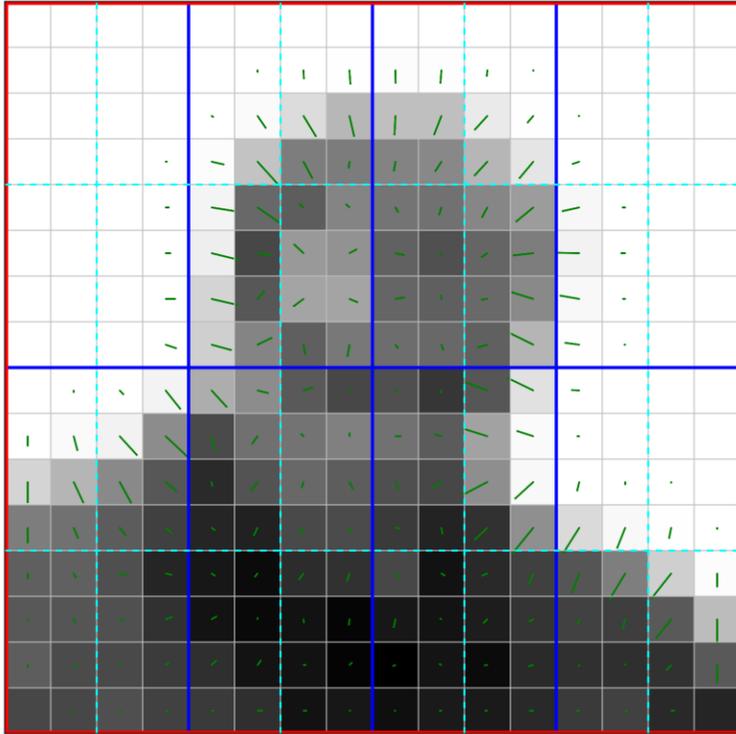
HOG-дескриптор. Алгоритм



НОG-дескриптор. Алгоритм



HOG-дескриптор. Алгоритм



[0.1474, 0.1039, 0.0189, ...
..., 0.2408, 0.0758]

НОG-дескриптор. Алгоритм



[0.177, 0.034, . . . , 0.130,
]

НОG-дескриптор. Алгоритм



```
[0.177, 0.034, ..., 0.130,  
0.015, 0.072, ..., 0.049,  
]
```

HOG-дескриптор. Алгоритм



```
[0.177, 0.034, ..., 0.130,  
0.015, 0.072, ..., 0.049,  
    ...,  
0.222, 0.118, ..., 0.185]
```

HOG-дескриптор. Реализация

- ❑ Алгоритмы вычисления HOG-признаков и их использования для детектирования объектов в OpenCV реализованы в классе `HOGDescriptor` (модуль `objdetect`)

```
HOGDescriptor(Size winSize,  
              Size blockSize,  
              Size blockStride,  
              Size cellSize,  
              int nbins,  
              int derivAperture=1,  
              double winSigma=-1,  
              int histogramNormType=HOGDescriptor::L2Hys,  
              double L2HysThreshold=0.2,  
              bool gammaCorrection=false,  
              int nlevels=HOGDescriptor::DEFAULT_NLEVELS)
```



НОГ-дескриптор. Реализация

- ❑ `winSize` – размер изображения (окна детектирования), для которого будут вычисляться НОГ-признаки.
- ❑ `blockSize` – размер блока в пикселях.
- ❑ `blockStride` – шаг блока по горизонтали и вертикали.
- ❑ `cellSize` – размер ячейки в пикселях.
- ❑ `nBins` – количество полос в гистограмме ориентации градиентов одной ячейки.
- ❑ `derivAperture` – не используется.
- ❑ `winSigma` – величина стандартного отклонения гауссиана, взвешивающего вкладов магнитуд градиентов в гистограммы. По умолчанию используется величина $\text{winSigma} = (\text{blockSize.width} + \text{blockSize.height}) / 8$.

НОG-дескриптор. Реализация

- ❑ `histogramNormType` – алгоритм нормировки вектора признаков блока.
- ❑ `L2HysThreshold` – пороговое значение компонент вектора признаков (параметр алгоритма нормировки).
- ❑ `gammaCorrection` – выполнять ли гамма-коррекцию изображения.
- ❑ `nlevels` – максимальное количество изменений масштаба изображения (параметр детектора).

НОG-детектор. Классификация

Если

- все изображения одинакового и фиксированного размера
- объекты занимают (приблизительно) одинаковую область изображения

То

- вектора признаков данных изображений имеют одинаковую длину
- с помощью алгоритмов обучения с учителем можно достаточно точно классифицировать изображения с объектом и без него



НОG-детектор. Бегущее окно

- ❑ Обучаем классификатор на изображениях размера $w \times h$
- ❑ Рассматриваем области изображения размера $w \times h$ с помощью бегущего окна
- ❑ Выполняем классификацию для каждого окна



НОG-детектор. Бегущее окно

- ❑ Обучаем классификатор на изображениях размера $w \times h$
- ❑ Рассматриваем области изображения размера $w \times h$ с помощью бегущего окна
- ❑ Выполняем классификацию для каждого окна



НОG-детектор. Бегущее окно

- ❑ Обучаем классификатор на изображениях размера $w \times h$
- ❑ Рассматриваем области изображения размера $w \times h$ с помощью бегущего окна
- ❑ Выполняем классификацию для каждого окна



НОG-детектор. Бегущее окно

- ❑ Обучаем классификатор на изображениях размера $w \times h$
- ❑ Рассматриваем области изображения размера $w \times h$ с помощью бегущего окна
- ❑ Выполняем классификацию для каждого окна



НОG-детектор. Масштабирование

- ❑ Выполняем масштабирование изображения для поиска объектов разного размера



НОG-детектор. Реализация

- Для вычисления НОG-признаков для всех или выбранных окон детектирования в классе `HOGDescriptor` реализован метод `compute`

```
void compute(const Mat& img,  
            vector<float>& descriptors,  
            Size winStride=Size(),  
            Size padding=Size(),  
            const vector<Point>& locations=vector<Point>()) const;
```



НОG-детектор. Реализация

- ❑ `img` – изображение, матрица типа `CV_8UC1` или `CV_8UC3`.
- ❑ `descriptors` – вектор, в который последовательно будут записаны признаковые описания заданных окон детектирования.
- ❑ `winStride` – шаг окна детектирования по горизонтали и вертикали. По умолчанию используется `winStride`, равный размеру ячейки НОG.
- ❑ `padding` – размер рамки, добавляемой к изображению.
- ❑ `locations` – положения окон детектирования, для которых требуется вычислить векторы НОG-признаков.

НОГ-детектор. Реализация

- ❑ В классе `HOGDescriptor` также реализован НОГ-детектор, использующий в качестве классификатора машину опорных векторов с линейным ядром.
- ❑ С помощью метода `void setSVMdetector (InputArray svmdetector);` можно загрузить обученную модель в виде коэффициентов разделяющей гиперплоскости.
- ❑ В OpenCV имеются две готовых модели для детектирования пешеходов, получить которые можно с помощью функций:
`vector<float> HOGDescriptor::getDefaultPeopleDetector ();`
`vector<float> HOGDescriptor::getDaimlerPeopleDetector ();`

НОG-детектор. Реализация

- ❑ Осуществить детектирование на изображении без масштабирования
можно с помощью метода `detect`

```
void detect(const Mat& img,  
           vector<Point>& foundLocations,  
           vector<double>& weights,  
           double hitThreshold=0,  
           Size winStride=Size(),  
           Size padding=Size(),  
           const vector<Point>& searchLocations=vector<Point>()  
) const;
```

НОG-детектор. Реализация

- ❑ `img` – изображение, матрица типа `CV_8UC1` или `CV_8UC3`.
- ❑ `foundLocations` – координаты верхних левых углов окон детектирования, классифицированных как содержащие объект (срабатываний детектора).
- ❑ `weights` – веса, присвоенные классификатором срабатываниям детектора.
- ❑ `hitThreshold` – минимальное значение веса, при котором происходит срабатывание детектора.
- ❑ `winStride` – шаг окна детектирования по горизонтали и вертикали.
- ❑ `padding` – размер рамки, добавляемой к изображению.
- ❑ `searchLocations` – положения окон детектирования, по умолчанию используются все.

HOG-детектор. Реализация

- Осуществить детектирование на изображении с использованием масштабирования можно с помощью метода `detectMultiScale`

```
void detectMultiScale(const Mat& img,  
                    vector<Rect>& foundLocations,  
                    vector<double>& foundWeights,  
                    double hitThreshold=0,  
                    Size winStride=Size(),  
                    Size padding=Size(),  
                    double scale=1.05,  
                    double finalThreshold=2.0,  
                    bool useMeanshiftGrouping=false) const;
```

НОG-детектор. Реализация

- ❑ Параметры `img`, `foundLocations`, `foundWeights`, `hitThreshold`, `winStride`, `padding` по смыслу совпадают с аналогичными параметрами метода `detect`.
- ❑ `scale` – мультипликативный шаг изменения масштаба. Исходное изображение последовательно уменьшается в 1 , $scale$, $scale^2$, ..., $scale^t$ раз.
- ❑ `finalThreshold` – параметр группировки (non-maximum suppression) срабатываний детектора на всех рассматриваемых масштабах. Значение данного параметра зависит от типа используемой группировки.
- ❑ `useMeanshiftGrouping` – параметр, определяющий использовать ли группировку методом сдвига среднего (Mean Shift), или группировку на основе разбиения на классы эквивалентности.

Детектор LatentSVM. Алгоритм

- ❑ Алгоритм для детектирования на изображении объектов различных классов, в том числе пешеходов.
- ❑ Основан на идеи независимого моделирования частей объектов с последующим объединением в единое целое с помощью описания их возможного взаимного расположения.
- ❑ Для описания частей объектов используются HOG-признаки.

Детектор LatentSVM. Реализация

- ❑ Функционал по использованию LatentSVM-детектора в библиотеке OpenCV сосредоточен в классе `LatentSvmDetector`.
- ❑ Перед выполнением детектирования необходимо загрузить модели искомых объектов. Это может быть сделано путем использования конструктора

```
LatentSvmDetector(const vector<string>& filenames,  
                 const vector<string>& classNames=vector<string>());
```

или конструктора по умолчанию с последующим вызовом функции

```
bool load(const vector<string>& filenames,  
          const vector<string>& classNames=vector<string>());
```

- ❑ Для детектирования людей необходимо загрузить модель `person.xml`.



Детектор LatentSVM. Реализация

- ❑ Детектирование с помощью загруженных моделей осуществляется с использованием метода `detect`:

```
void detect(const Mat& image,
            vector<ObjectDetection>& objectDetections,
            float overlapThreshold=0.5f,
            int numThreads=-1);
```

- ❑ `image` – изображение.
- ❑ `objectDetections` – срабатывания детектора, представляемые структурой `ObjectDetection`. Данная структура содержит следующие поля: `rect` – окаймляющий прямоугольник, `score` – вес срабатывания, `classID` – номер класса объекта.
- ❑ `overlapThreshold` – параметр группировки срабатываний.
- ❑ `numThreads` – количество используемых вычислительных потоков.



РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ ДЕТЕКТИРОВАНИЯ ПЕШЕХОДОВ



Требования к приложению

- Предлагается разработать приложение для детектирования пешеходов на изображениях с использованием рассмотренных алгоритмов.
- К приложению предъявляются следующие требования:
 - Выбор алгоритма детектирования.
 - Детектирование на множестве изображений.
 - Визуализация результата работы детектора для оценки качества детектирования на наборе изображений.

Структура приложения

- Приложение должно:
 - Осуществлять последовательную загрузку указанных (например, всех в заданной директории) изображений.
 - Осуществлять на них детектирование выбранным алгоритмом с заданными параметрами.
 - Выводить результат в виде отрисованных на исходном изображении срабатываний детектора.

Задания

- ❑ Реализуйте приложение, удовлетворяющее предъявляемым требованиям.
- ❑ Изучите влияние параметров алгоритмов детектирования (особенно параметров группировки срабатываний и порога веса срабатываний) на конечный результат.

Авторский коллектив

- Дружков Павел Николаевич,
аспирант кафедры математической логики и высшей
алгебры факультета ВМК ННГУ
druzhkov.paul@gmail.com