Содержание

[Введение 4](#_Toc419103053)

[1. Постановка задачи 5](#_Toc419103054)

[1.1. Формат входных и выходных данных 5](#_Toc419103055)

[1.2. Методы оценки качества детекции 6](#_Toc419103056)

[2. Общий подход к решению задачи 7](#_Toc419103057)

[2.1. Гистограмма ориентированных градиентов (HOG) 7](#_Toc419103058)

[3. Используемые инструменты 8](#_Toc419103059)

[4. Общий подход к реализации 8](#_Toc419103060)

[5. Результаты тестирования 9](#_Toc419103061)

[Заключение 10](#_Toc419103062)

[Список используемых источников 11](#_Toc419103063)

[Приложение А. Результаты работы детектора. 12](#_Toc419103064)

[Приложение Б. Исходный код 13](#_Toc419103065)

# Введение

Задача распознавания образов, в том или ином виде, встречается в различных областях человеческой деятельности – от военного дела и систем безопасности до оцифровки аналоговых сигналов. С развитием вычислительных мощностей появляются все новые и новые задачи, в качестве подзадач включающие в себя детекцию объектов. Широкое применение такие алгоритмы получили в области дорожного движения. В частности, системы распознавания пешеходов могут применятся для анализа загруженности городских улиц, контроля за соблюдением правил дорожного движения и повышения безопасности на дорогах. Особенное значение такого рода системы имеют при разработке беспилотных транспортных средств. Основной их целью является предотвращение столкновения автомобиля с человеком.

В области распознавания пешеходов проводятся многочисленные исследования. Основными проблемами в ходе решения задачи – выбор классификаторов и способа формирования векторов признаков. На данный момент существует целое множество решений этой задачи [1][2].

Однако, задача распознавания пешеходов – не единственная задача распознавания, с которой сталкиваются разработчики беспилотных автомобилей. В рамках масштабного проекта по созданию к 2020 году беспилотного автомобиля компанией Cognitive Technologies был объявлен конкурс на разработку программного обеспечения детекции транспортных средств движущихся по дорогам, на изображениях, полученных с движущейся видео-камеры (видео регистраторa). Целью данной работы является разработка алгоритма, решающего данную задачу, основанного на уже известных подходах к детекции пешеходов.

# Постановка задачи

В рамках решения задачи предполагалась разработка алгоритма, позволяющего определить факт наличия и расположение автомобиля на фото с видео регистратора. Для заданного изображения необходимо получить список четырехугольников, описывающих изображенные на нем транспортные средства и степень уверенности в полученных координатах.

## Формат входных и выходных данных

Для решения задачи были предоставлены два вида файлов: данные для обучения детектора и тестовая выборка.

Данные для обучения детектора были представлены в виде единого архива, внутри которого были расположены следующие директории:

* image - цветные изображения размера 480x270;
* label - разметка изображений на предмет наличия.

Файлы разметки именовались по правилу <image-name>.txt и содержали следующую информацию:

* Флаг о полностью размеченном кадре (true или false)
* Список ограничивающих квадратов автомобилей в формате x y width height (четыре числа, разделенные пробелом, задающие левый верхний угол и размеры квадрата; начало отсчета – левый верхний угол)

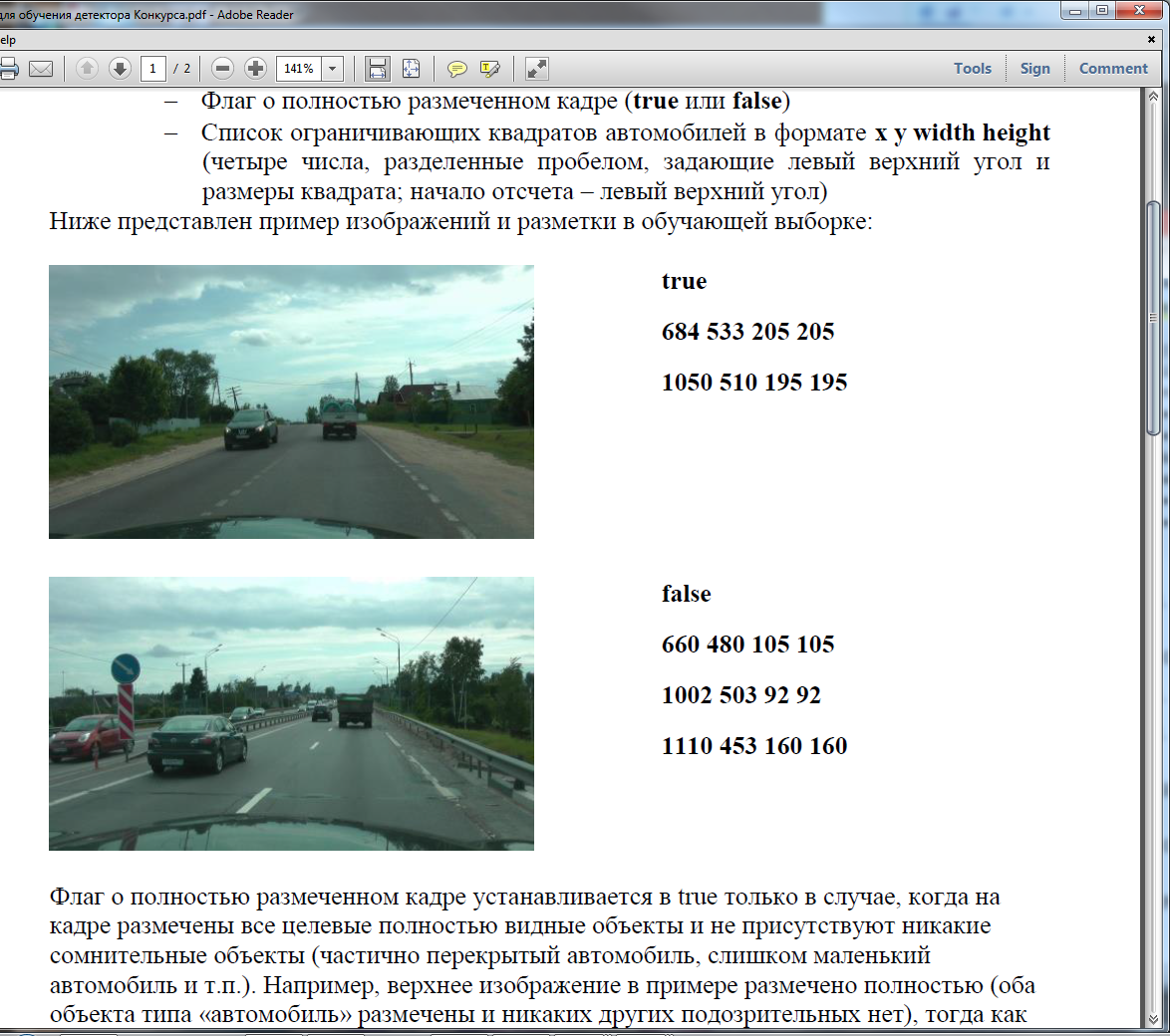


Рисунок 1.1 – Пример изображений и разметки в обучающей выборке

Флаг о полностью размеченном кадре устанавливался в true только в случае, когда на кадре были размечены все целевые полностью видные объекты и не присутствовали никакие сомнительные объекты (частично перекрытый автомобиль, слишком маленький автомобиль и т.п.). Например, верхнее изображение в примере размечено полностью (оба объекта типа «автомобиль» размечены и никаких других подозрительных нет), тогда как нижнее изображение не полностью размечено (присутствую несколько частично перекрытых автомобилей на встречной полосе).

В качестве результата работы программа должна была для каждого изображения создавать текстовый файл с результатами детектирования. Файл результатов должен был именоваться по правилам <image-name>.txt и содержать список ограничивающих квадратов найденных на изображении автомобилей в следующем формате:

x1 y1 width1 height1 score1

x2 y2 width2 height2 score2

…

xn yn widthn heightn scoren

Здесь x y width height – четыре числа, разделенные пробелом, задающие левый верхний угол и размеры квадрата, а score – число от 0 до 255, указывающее степень уверенности в найденном объекте (чем больше – тем более надежным считается найденный объект).

## Методы оценки качества детекции

Для оценки качества работы детектора была использована кривая точность-полнота (precision-recall curve), построенная на основе степени уверенности.

Для оценки того, что объект найден правильно, использовалась мера Жаккара с коэффициентом 0.5. Иными словами, пусть у нас есть идеальное расположение автомобиля на карте *Ideal*, а также результат детектирования *Real.* Тогда считаем, что результат детектирования верный, если:

Далее введем несколько понятий. Общее количество объектов в разметке будем называть «количество положительных объектов» и обозначать P (Positive). Правильно детектированные объекты будем называть «точное срабатывание» и обозначать TP (True Positive). Неправильно детектированные объекты будем называть «ложными срабатываниями» и обозначать FP (False Positive).

Таким образом, полнота (True Positive Rate, TPR) и точность (Positive Predictive Value, PPV) выражается по следующим формулам:

Для общего анализа работы детектора вычислялась F- метрика, которая представляет собой среднее гармоническое величин TPR и PPV:

# Общий подход к решению задачи

Для решения поставленной задачи было решено использовать HOG-дескрипторы. Для классификации был использован метод опорных векторов.

## Гистограмма ориентированных градиентов (HOG)

Гистограммы ориентированных градиентов - один из способа построения дескрипторов особых точек. Данный подход основан на подсчете количества направлений градиента в локальных областях изображения и отличается тем, что вычисляется на плотной сетке равномерно распределенных ячеек и использует нормализацию перекрывающегося локального контраста для увеличения точности.

Основной идеей алгоритма является допущение, что внешний вид и форма объекта на участке изображения могут быть описаны распределением градиентов интенсивности или направлением краев.

При описании фрагмента изображения оно разбивается на несколько небольших участков, называемых ячейками. В ячейках вычисляются гистограммы направленных градиентов внутренних точек. Обычно они объединяются в одну гистограмму ℎ = 𝑓(ℎ1, ..., ℎ𝑘), после чего она нормализуется по яркости. Таким образом достигается инвариантность к изменениям освещения. Для аппроксимации градиента изображения могут быть использованы различные дискретные дифференциальные операторы, такие как оператор Собеля, оператор Превитта или оператор Щарра.

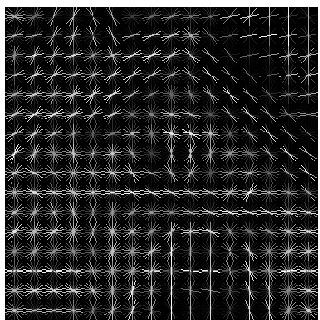


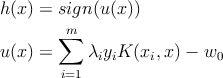
Рисунок 2.1 – Гистограмма ориентированных градиентов

* 1. Метод опорных векторов (SVM)

Данный метод является одним из решений задачи классификации, которая состоит в определении к какому классу из, как минимум, двух изначально известных относится данный объект. При таком подходе обычно предполагается, что выборка линейно разделима (задача бинарной классификации). Метод основан на построении гиперплоскости, разделяющей исходную выборку на два класса. Так как существует бесконечно много плоскостей, вводится требования, чтобы она располагалась максимально далеко от ближайших к ней точек обоих классов.

Перед началом работы, классификатор необходимо обучить на выборке, разделенной на «положительные» и «отрицательные» примеры.

Классификатор описывается следующими соотношениями



где – порог (свободный член); – коэффициент; – пара из учебного набора, (векторы , для которых > 0 называются опорными); K ( , x ) – функция ядра

Машина опорных векторов отображает входной вектор в пространство более высокой размерности и находит разделяющую гиперплоскость с максимальным зазором между точками разных классов. Отображение это осуществляется с помощью функции ядра.

Изначально SVM это линейный классификатор, т.е. он может решать только линейно разделимые задачи. Однако, применяя нелинейное ядро, можно отобразить исходные данные в пространство большей размерности, где уже может существовать оптимальная разделяющая гиперплоскость.

Обучение SVM сводится к нахождению коэффициентов λ (множителей Лангранжа) и .

Для классификации объекта достаточно определить его положение относительно гиперплоскости.

# Используемые инструменты

Существует большое количество библиотек, реализующих основные алгоритмы компьютерного зрения. Для решения данной задачи была выбрана открытая библиотека VLFeat, качественно реализующая весь необходимый инструментарий, а также предоставляющая возможность многопоточной обработки данных с использованием OpenMP. На официальном сайте [3] представлена подробная документация.

# Общий подход к реализации

Для решения задачи была принята следующая схема:

1. Получение обучающей выборки.

Положительная выборка изначально была получена из размеченных файлов по указанным координатам. Отрицательная – случайным образом из пустых полностью размеченных файлов.

1. Обучение модели.

Рассматривалось два класса: автомобиль и не автомобиль

1. Обработка тестовой выборки:
   1. Сканирующим окном выделить очередную область изображения.
   2. Найти HOG – дескриптор для данной области
   3. Используя модель, определить, к какому классу принадлежит изображение.

В ходе выполнения задачи были разработаны следующие классы:

1. SVM – класс для обучения модели и вычисления HOG – дескриптора для изображения.

*static VlSvm \*getModel(vector<QImage> positive, vector<QImage> negative)* – обучение модели

*static float\* getHog(VlHog \*hog, float \*image, int size, int \*hogSize, vl\_size cellSize)* - вычисление HOG-дескриптора

1. Image – класс изображения

*shared\_ptr<float> getPixels() –* получение массива пикселей изображения

*shared\_ptr<float> getWindow(int x, int y, int w, int h) –* получение окна в заданных координатах

1. FileWorker – вспомогательный класс для чтения входных данных.

*static void getLearningExamples(const QString& textFolder, const QString& imageFolder)* – получение обучающей выборки

*static void getCars(const QString& textFolder, const QString& imageFolder) –* получение положительных примеров

*static void resize(const QString& folder, int height, int width)* – изменение размера изображений в директории

*static void getEmpty(const QString& textFolder)* – получение отрицательных примеров из пустых размеченных изображений

1. Detector – класс для поиска автомобилей на изображении

*static Answer detectImage(QImage qImage, const QString& img, const QString& folder, const QString& text, const double \*model, int imgSize); -* получение ответа для изображения

*static double mult(const double\* model, float\* hog, int size) –* скалярноеумножение векторов

1. Answer - результат обработки одного файла

*void setFileName(QString name)* – запомнить название файла, для которого получен результат

*void addRect(QRect rect)* – добавить описывающий автомобиль прямоугольник в результаты

*void save()* – сохранить результат

# Результаты тестирования

Полученное решение было протестировано на выборке размером 10 000 изображений. Обученная на случайным образом полученной выборке модель позволила достичь F-метрики в 0.49. Введение дополнительных условий на положение возможного автомобиля и подбор параметров HOG-дескриптора и SVM улучшили результат до 0.63. Обучение модели на выборке, составленной вручную позволило достичь итогового результата в 0.74.

Примеры изображений, размеченных системой, представлены в приложении А.

# Заключение

В ходе данной работы была реализована система детекции транспортных средств движущихся по дорогам, на изображениях, полученных с движущейся видео-камеры (видео регистраторa). Система успешно решает поставленную задачу, показывая довольно хорошие результаты на тестовой выборке.

При выполнении была изучена литература по данному вопросу, рассмотрены различные подходы к решению задач распознавания и классификации. Также был получен опыт работы с библиотеками VLFeat и OpenMP.

# Список используемых источников

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. – C. 1030-1037
3. VLFeat.org. [Электронный ресурс]: - Режим доступа: http://www.vlfeat.org/
4. Navneet Dalal, Bill Triggs, Cordelia Schmid: Human Detection Using Oriented Histograms of Flow and Appearance. ECCV (2), 2006. – C. 428-441

# Приложение А. Результаты работы детектора.



Рисунок А.1 – Пример правильной детекции

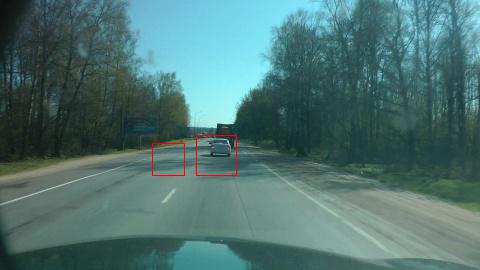
****

Рисунок А.2 – Пример избыточной детекции



Рисунок А.3 – Пример неполной детекции

# Приложение Б. Исходный код

answer.h

#ifndef ANSWER\_H

#define ANSWER\_H

#include <vector>

#include <QString>

#include <QRect>

#include <QFile>

#include <QTextStream>

using namespace std;

class Answer

{

vector<QRect> results;

QString fileName; //полный путь до текстового файла

public:

Answer();

void setFileName(QString name);

void addRect(QRect rect);

void save();

~Answer();

};

#endif

answer.cpp

#include "answer.h"

Answer::Answer()

{

}

void Answer::setFileName(QString name)

{

fileName = name;

}

void Answer::addRect(QRect rect)

{

results.push\_back(rect);

}

void Answer::save()

{

QFile file( fileName );

if ( file.*open*(QIODevice::ReadWrite) )

{

QTextStream stream( &file );

for(int i=0; i<results.size(); i++)

{

stream << results[i].x() << " " << results[i].y() << " " << results[i].width()<<" " << results[i].height()<<" "<< 255 <<endl;

}

file.*close*();

}

}

Answer::~Answer()

{

}

detector.h

#ifndef DETECTOR\_H

#define DETECTOR\_H

#include "image.h"

#include "svm.h"

#include "qpainter.h"

#include "answer.h"

#include <QFile>

#include <QTextStream>

class Detector

{

public:

Detector();

static Answer detectImage(QImage qImage, const QString& img, const QString& folder, const QString& text, const double \*model, int imgSize);

static double mult(const double\* model, float\* hog, int size);

~Detector();

};

#endif

detector.cpp

#include "detector.h"

Detector::Detector()

{

}

double Detector::mult(const double\* model, float\* hog, int size)

{

double mult = 0;

for(int i=0; i<size; i++)

{

mult = mult + model[i]\*hog[i];

}

return mult;

}

Answer Detector::detectImage(QImage qImage, const QString& img, const QString& folder, const QString& text, const double \*model, int imgSize)

{

QRect rect;

unique\_ptr<Image> image = make\_unique<Image>(qImage);

shared\_ptr<float> window;

int hogSize;

VlHog \*hog;

float \*hogArray;

double prediction;

vl\_size cellSize(5);

int maxWindow = qImage.height()/3 - (qImage.height()/3)%8;

Answer answer;

vector<QRect> rectangles;

vector<double> predictions;

bool intersec;

int intersectNum;

QRect intersection;

for(int windowSize = maxWindow; windowSize >= imgSize; windowSize -= 8)

{

cellSize = windowSize/8;

for(int y = qImage.height()/4; y < qImage.height()\*3/4 - windowSize; y += 4)

{

for(int x = 0; x < qImage.width() - windowSize; x+=4)

{

rect = QRect(x,y,windowSize, windowSize);

intersec = false;

intersectNum = -1;

for(int i=0; i<rectangles.size();i++)

{

if(rect.intersects(rectangles[i]))

{

if(rect.width() < rectangles[i].width())

{

intersec = true;

break;

}

intersection = rect.intersected(rectangles[i]);

if(intersection.width()\*intersection.height() > rect.width()\*rect.height()/2)

{

intersec = true;

break;

}

else

{

intersectNum = i;

break;

}

}

}

if(!intersec)

{

window = image->getWindow(x, y, windowSize, windowSize);

hogArray = SVM::getHog(hog, window.get(), windowSize, &hogSize, cellSize);

prediction = mult(model, hogArray, hogSize);

vl\_free(hogArray);

if(intersectNum > -1 && prediction > predictions[intersectNum])

{

predictions[intersectNum] = prediction;

rectangles[intersectNum] = rect;

}

else if(intersectNum == -1 && prediction > 0.2)//0.48

{

// qDebug() << prediction;

rectangles.push\_back(rect);

predictions.push\_back(prediction);

}

}

}

}

}

answer.setFileName(text + img + ".txt");

for(int i=0; i<rectangles.size(); i++)

{

answer.addRect(rectangles[i]);

}

return answer;

}

Detector::~Detector()

{

}

fileworker.h

#ifndef FILEWORKER\_H

#define FILEWORKER\_H

#include <QString>

#include <QDir>

#include <QDebug>

#include <QImage>

class FileWorker

{

static int minWidth;

static int minHeight;

public:

FileWorker();

static void getLearningExamples(const QString& textFolder, const QString& imageFolder);

static void getCars(const QString& textFolder, const QString& imageFolder);

static void resize(const QString& folder, int height, int width);

static void getEmpty(const QString& textFolder);

~FileWorker();

};

#endif

fileworker.cpp

#include "fileworker.h"

FileWorker::FileWorker()

{

}

int FileWorker::minHeight = 10000000;

int FileWorker::minWidth = 10000000;

void FileWorker::getLearningExamples(const QString& textFolder, const QString& imageFolder)

{

QDir textDir(textFolder);

QStringList textFiles = textDir.entryList(QStringList("\*.txt"));

QStringList::iterator textFile = textFiles.begin();

while(textFile != textFiles.end())

{

QFile file(textFolder + "\\" + \*textFile);

file.*open*(QIODevice::ReadOnly);

while(!file.*atEnd*())

{

// qDebug()<< \*textFile;

//читаем строку

QString str = file.readLine();

if(QString::compare(str, "False\r\n") == 0) break;

if(QString::compare(str, "True\r\n") == 0) continue;

else {

// qDebug()<<str;

QFile::copy(textFolder + "\\" + \*textFile, "E:\\learnP\\text\\" + \*textFile);

QFile::copy(imageFolder + "\\" + (\*textFile).remove(".txt"), "E:\\learnP\\image\\" + (\*textFile).remove(".txt"));

break;

}

}

file.*close*();

textFile++;

}

qDebug()<< "Finished";

}

void FileWorker::getCars(const QString& textFolder, const QString& imageFolder)

{

QDir textDir(textFolder);

QStringList textFiles = textDir.entryList(QStringList("\*.txt"));

QStringList::iterator textFile = textFiles.begin();

int num = 0;

minWidth = minHeight = 100000;

QImage image;

while(textFile != textFiles.end())

{

QFile file(textFolder + "\\" + \*textFile);

file.*open*(QIODevice::ReadOnly);

while(!file.*atEnd*())

{

QString str = file.readLine();

if(QString::compare(str, "True\r\n") == 0 || QString::compare(str, "False\r\n") == 0)

{

image.load(imageFolder + "\\" + (\*textFile).remove(".txt"));

}

else {

QStringList split = str.split(" ");

QRect rect(split[0].toInt(), split[1].toInt(),split[2].toInt(),split[3].toInt());

QImage car = image.copy(rect);

car.save("E:\\learn\\cars\\" + QString::number(num) + ".jpg");

num++;

if(rect.width() < minWidth) minWidth = rect.width();

if(rect.height() < minHeight) minHeight = rect.height();

}

}

file.*close*();

textFile++;

}

resize("E:\\learn\\cars\\", minHeight, minWidth);

qDebug()<< "Finished";

}

void FileWorker::resize(const QString &folder, int height, int width)

{

QDir imageDir(folder);

QStringList files = imageDir.entryList(QStringList("\*.png"));

QStringList::iterator imageFile = files.begin();

QImage image;

while(imageFile != files.end())

{

QImage image(folder + "\\" +\*imageFile);

image = image.scaled(width, height);

image.save(folder + "\\" +\*imageFile);

imageFile++;

}

qDebug()<< "Finished";

}

void FileWorker::getEmpty(const QString& textFolder)

{

QDir textDir(textFolder);

QStringList textFiles = textDir.entryList(QStringList("\*.txt"));

QStringList::iterator textFile = textFiles.begin();

QDir emptyDir("E:\\empty\\image");

QStringList emptyFiles = emptyDir.entryList(QStringList("\*.jpg"));

QStringList::iterator emptyFile = emptyFiles.begin();

int num = 0;

minWidth = minHeight = 100000;

QImage image;

while(textFile != textFiles.end())

{

QFile file(textFolder + "\\" + \*textFile);

file.*open*(QIODevice::ReadOnly);

while(!file.*atEnd*())

{

QString str = file.readLine();

if(QString::compare(str, "True\r\n") == 0 || QString::compare(str, "False\r\n") == 0)

{

}

else {

image.load("E:\\empty\\image\\" + \*emptyFile);

QStringList split = str.split(" ");

QRect rect(rand() % (image.width() - split[2].toInt()), rand()%(image.height()/2 - split[3].toInt())+ image.height()/4,split[2].toInt(),split[3].toInt());

QImage empty = image.copy(rect);

empty.save("E:\\learn\\notcars\\" + QString::number(num) + ".jpg");

num++;

emptyFile++;

if(emptyFile == emptyFiles.end()) emptyFile = emptyFiles.begin()+1;

rect = QRect(rand() % (image.width() - split[2].toInt()), rand()%(image.height()/2 - split[3].toInt())+ image.height()/4,split[2].toInt(),split[3].toInt());

empty = image.copy(rect);

empty.save("E:\\learn\\notcars\\" + QString::number(num) + ".jpg");

num++;

if(emptyFile == emptyFiles.end()) emptyFile = emptyFiles.begin()+1;

}

}

file.*close*();

textFile++;

}

qDebug()<< "Finished";

resize("E:\\learn\\notcars\\", 25,25);

qDebug()<< "Finished";

}

FileWorker::~FileWorker()

{

}

image.h

#ifndef IMAGE\_H

#define IMAGE\_H

#include <memory>

#include <QImage>

using namespace std;

template< typename T >

struct array\_deleter

{

void operator ()( T const \* p)

{

delete[] p;

}

};

class Image

{

shared\_ptr<float> pixels;

int width, height;

public:

Image(const QImage& image);

shared\_ptr<float> getPixels();

shared\_ptr<float> getWindow(int x, int y, int w, int h);

~Image();

};

#endif

image.cpp

#include "image.h"

Image::Image(const QImage& image)

{

pixels = shared\_ptr<float>(new float[image.width()\*image.height()], array\_deleter<float>());

QRgb original;

float color;

for(int i=0; i<image.height(); i++)

{

for(int j=0; j<image.width(); j++)

{

original = image.pixel(j,i);

color = 0.299 \* qRed(original) +

0.587 \* qGreen(original) +

0.114 \* qBlue(original);

\*(pixels.get() + i\*image.width() + j) = color;

}

}

width = image.width();

height = image.height();

}

shared\_ptr<float> Image::getPixels()

{

return pixels;

}

shared\_ptr<float> Image::getWindow(int x, int y, int w, int h)

{

shared\_ptr<float> window(new float[w\*h], array\_deleter<float>());

for(int i = 0; i<h; i++)

{

for(int j = 0; j < w; j++)

{

\*(window.get() + i\*w + j) = \*(pixels.get() + (i+y)\*width + j+x);

}

}

return window;

}

Image::~Image()

{

}

svm.h

#ifndef SVM\_H

#define SVM\_H

#include <QDir>

#include <QImage>

#include <QDebug>

#include <QString>

#include <memory>

#include <image.h>

#include <vl/hog.h>

#include <vl/svm.h>

using namespace std;

class SVM

{

public:

SVM();

static VlSvm \*getModel(vector<QImage> positive, vector<QImage> negative);

static float\* getHog(VlHog \*hog, float \*image, int size, int \*hogSize, vl\_size cellSize);

~SVM();

};

#endif

svm.cpp

#include "svm.h"

SVM::SVM()

{

}

float\* SVM::getHog(VlHog \*hog, float \*image, int size, int \*hogSize, vl\_size cellSize)

{

vl\_size numOrientations(8);

hog = vl\_hog\_new(VlHogVariantDalalTriggs, numOrientations, VL\_FALSE);

vl\_hog\_put\_image(hog, image, size, size, 1, cellSize) ;

vl\_size hogWidth = vl\_hog\_get\_width(hog);

vl\_size hogHeight = vl\_hog\_get\_height(hog);

vl\_size hogDimension = vl\_hog\_get\_dimension(hog);

\*hogSize = hogWidth\*hogHeight\*hogDimension;

float \*hogArray = (float\*)vl\_malloc(\*hogSize\*sizeof(float)) ;

vl\_hog\_extract(hog, hogArray);

vl\_hog\_delete(hog);

return hogArray;

}

VlSvm\* SVM::getModel(vector<QImage> positive, vector<QImage> negative)

{

uint posNum = positive.size();

uint negNum = negative.size();

int hogSize;

VlHog \*hog;

shared\_ptr<float> imagePixels;

int i=0, j=0;

float \*hogArray;

unique\_ptr<Image> floatImage;

unique\_ptr<double[]> x;

floatImage = make\_unique<Image>(positive[0]);

imagePixels = floatImage->getPixels();

hogArray = getHog(hog, imagePixels.get(), positive[0].width(), &hogSize, 5);

x = make\_unique<double[]>(hogSize \* (posNum + negNum));

unique\_ptr<double[]> y = make\_unique<double[]>(posNum + negNum);

for(int j=0; j<hogSize; j++)

{

x[i++] = hogArray[j];

}

y[j++] = 1;

vl\_free(hogArray);

for(uint pos = 1; pos < positive.size(); pos++)

{

floatImage = make\_unique<Image>(positive[pos]);

imagePixels = floatImage->getPixels();

hogArray = getHog(hog, imagePixels.get(), positive[pos].width(), &hogSize, 5);

for(int j=0; j<hogSize; j++)

{

x[i++] = hogArray[j];

}

y[j++] = 1;

vl\_free(hogArray);

}

for(uint neg = 0; neg < negative.size(); neg++)

{

floatImage = make\_unique<Image>(negative[neg]);

imagePixels = floatImage->getPixels();

hogArray = getHog(hog, imagePixels.get(), negative[neg].width(), &hogSize, 5);

for(int j=0; j<hogSize; j++)

{

x[i++] = hogArray[j];

}

y[j++] = -1;

vl\_free(hogArray);

}

//train

double lambda = 0.1;

VlSvm \* svm = vl\_svm\_new(VlSvmSolverSdca,

x.get(), hogSize, posNum + negNum,

y.get(),

lambda);

vl\_svm\_train(svm);

return svm;

}

SVM::~SVM()

{

}

main.cpp

#include <QCoreApplication>

#include "svm.h"

#include "fileworker.h"

#include "detector.h"

#include "answer.h"

#include <vl/generic.h>

#include <QDir>

#include <omp.h>

int main()

{

vector<QImage> cars;

vector<QImage> notcars;

vector<QImage> images;

vector<Answer> answers;

FileWorker::resize(QDir::currentPath() + "/info/learning/notcars", 40,40);

FileWorker::resize(QDir::currentPath() + "/info/learning/cars", 40,40);

QDir carsDir(QDir::currentPath() + "/info/learning/cars/");

QDir notcarsDir(QDir::currentPath() + "/info/learning/notcars/");

QDir testDir(QDir::currentPath() + "/info/test/images/");

if(!QDir(QDir::currentPath() + "/info").exists())

{

qFatal() << "Не создана система каталогов. Подробности - в инструкции по установке";

}

qDebug() << "loading data";

QStringList posImages = carsDir.entryList(QStringList("\*.png"));

for(uint i = 0; i<posImages.size(); i++)

{

cars.push\_back(QImage(carsDir.absolutePath() + "\\" + posImages[i]));

}

QStringList negImages = notcarsDir.entryList(QStringList("\*.png"));

for(uint i = 0; i<negImages.size(); i++)

{

notcars.push\_back(QImage(notcarsDir.absolutePath() + "\\" + negImages[i]));

}

qDebug() << "learning";

SVM svmFactory;

vl\_set\_num\_threads(vl\_get\_thread\_limit());

omp\_set\_num\_threads(omp\_get\_max\_threads());

VlSvm \* svm = svmFactory.getModel(cars, notcars);

cars.clear();

notcars.clear();

qDebug() << "detecting";

QStringList testFiles = testDir.entryList(QStringList("\*.jpg"));

int s = 10;

int stepNum = testFiles.size()%s == 0 ? testFiles.size()/s : 1 + testFiles.size()/s;

for(uint step = 0; step < stepNum; step++)

{

qDebug() << "detecting";

for(uint i = step\*s; i<step\*s + s && i<testFiles.size(); i++)

{

images.push\_back(QImage(testDir.absolutePath() + "\\" + testFiles[i]));

}

#pragma omp parallel for

for(int i=0; i<images.size(); i++)

{

answers.push\_back(Detector::detectImage(images[i], testFiles[i + step\*s], QDir::currentPath() + "\\info\\test\\images\\",

QDir::currentPath() + "\\info\\result\\text\\", vl\_svm\_get\_model(svm), 25));

}

qDebug() << "saving";

for(uint i=0; i<answers.size(); i++)

{

answers[i].save();

}

qDebug()<<"Обработано " << step\*s + images.size()<< " файлов";

answers.clear();

images.clear();

}

return 0;

}