

Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Алтайский государственный технический университет
им. И.И. Ползунова»

Факультет информационных технологий
Кафедра Прикладной математики

Отчет защищен с оценкой

Руководитель работы

Е.Н.Крючкова
(подпись, должность, и.о. фамилия)
“ ____ ” _____ 2016 г.

КУРСОВОЙ ПРОЕКТ

Параллельные алгоритмы в задачах распознавания дорожных знаков в
видеопотоке

Пояснительная записка
по дисциплине «Проектирование сетевых и многопоточных приложений»
КП 09.04.04.06.000 ПЗ

Работу выполнил
студент группы 8ПИ-61 _____ А.О.Корней
(подпись) (и.о., фамилия)

Руководитель
проекта ____ профессор, к.ф.-м.н. _____ Е.Н.Крючкова
(должность, ученое звание) (подпись) (и.о., фамилия)

БАРНАУЛ 2016

Задание

Учебная дисциплина: Проектирование сетевых и многопоточных приложений

Выполнил: студент группы 8ПИ-61 Корней Алена Олеговна

Тема курсового проекта: Параллельные алгоритмы в задачах распознавания дорожных знаков в идеопотоке

Разделы работы и сроки выполнения:

- 1) Постановка задачи: октябрь 2016
- 2) Изучение литературы и проектирование: октябрь-ноябрь 2016
- 4) Реализация и тестирование: ноябрь 2016
- 5) Написание отчёта: декабрь 2016

Руководитель: _____ профессор, к.ф.-м.н., Крючкова Е.Н.

(подпись)

					КП 09.04.04.06.000 ПЗ			
		№ докум.	Подпись		ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ В ЗАДАЧАХ РАСПОЗНАВАНИЯ ДОРОЖНЫХ ЗНАКОВ В ВИДЕОПОТОКЕ			
Разработал		КОРНЕЙ А.О.						
Пров.		КРЮЧКОВА Е.Н.						
Н.контролер		КРЮЧКОВА Е.Н.						
УТВ .		КАНТОР С.А.			АлтГТУ, ФИТ гр 8ПИ-61			

Аннотация

В данной работе исследована возможность применения параллельных алгоритмов в задачах распознавания дорожных знаков в видеопотоке. Разработана собственная тактика обнаружения знаков в кадре, построен алгоритм, предоставляющий возможности для распараллеливания вычислений.

Содержание

Введение.....	5
1 Общие проблемы распознавания дорожных знаков	6
1.1 Обнаружение дорожного знака в кадре.....	7
1.2 Распознавание дорожных знаков	10
2 Параллельный алгоритм распознавания дорожных знаков	13
2.1 Этап обнаружения знака в кадре	13
2.2 Этап распознавания	16
2.3 Схема алгоритма и ожидаемые результаты его применения ..	17
3 Реализация.....	19
3.1 Структура ПО	19
3.2 Результат работы.....	20
Список источников	22

Введение

В современном мире личный автомобиль перестал быть атрибутом роскоши, теперь это – общедоступное средство передвижения. В связи с этим последние годы значительно возросло число автомобилей на дорогах. Обеспечивать безопасность движения в таких условиях становится все труднее и труднее.

Одним из средств обеспечения безопасности являются дорожные знаки. С их помощью устанавливаются оптимальные скоростные ограничения, регулируются приоритеты, обозначаются опасные участки. Однако, находясь в транспортном потоке, водитель прежде всего старается следить за обстановкой и может случайно проигнорировать тот или иной знак, а это – потенциально опасная ситуация.

Для того, чтобы помочь водителям, создаются специализированные системы для распознавания дорожных знаков. Использование подобных систем приводит к снижению рисков и уменьшению числа аварийных ситуаций, т.к. водитель обладает более полной информацией.

В данной работе рассматриваются возможности применения параллельных алгоритмов в задачах распознавания дорожных знаков в видеопотоке.

1 Общие проблемы распознавания дорожных знаков

Распознавание дорожных знаков является одной из наиболее активно изучаемых задач в области компьютерного зрения. На сегодняшний день производители автомобилей оснащают некоторые модели продвинутыми системами помощи водителю, в состав которых входят и модули распознавания дорожных знаков.

Задача распознавания знаков в видеопотоке делится на два последовательных этапа:

- Обнаружение дорожного знака в кадре
- Распознавание обнаруженного знака.

Каждый из этапов сопряжен с рядом сложностей.

Во-первых, алгоритмы обнаружения и распознавания должны быть достаточно быстрыми, чтобы обеспечить работу с видеопотоком в реальном времени.

Во-вторых, алгоритмы обнаружения должны работать достаточно точно – без лишних ложных срабатываний, без пропусков.

В-третьих, алгоритмы распознавания должны быть достаточно точными, чтобы справиться со сложными классами знаков. К примеру, большую сложность представляют знаки ограничения скорости. Отдельные знаки очень слабо отличаются друг от друга, что серьезно затрудняет процесс распознавания.

В-четвертых, алгоритмы обнаружения и распознавания должны быть адаптированы для различных погодных условий и разного освещения.

Очевидно, что требования, предъявляемые к системам распознавания дорожных знаков, достаточно противоречивы и построение любой такой системы является компромиссом.

Рассмотрим, какие подходы применяются на каждом из этапов, и проанализируем их достоинства и недостатки.

1.1 Обнаружение дорожного знака в кадре

Дорожные знаки имеют ряд особенностей, которые позволяют водителям легко видеть и узнавать их даже в динамичном дорожном потоке. В 1968 году была принята Венская конвенция о дорожных знаках и сигналах, согласно которой дорожные знаки во многих странах были приведены к общему стандарту и разделены на 8 классов [1]:

Таблица 1 – категории дорожных знаков и примеры оформления

Класс	Изображения
Предупреждающие знаки	
Знаки преимущественного права проезда.	
Запрещающие и ограничивающие знаки.	
Предписывающие знаки.	
Знаки особых предписаний.	

Информационные знаки, знаки, обозначающие объекты и знаки сервиса.	
Указатели направлений и информационно-указательные знаки.	
Дополнительные таблички.	

Очевидно, что при наличии международных стандартов задача локализации дорожного знака в кадре несколько упрощается. Однако, существование особых знаков, а так же большие внутриклассовые различия, характерные для некоторых категорий, представляют собой серьезные трудности.

В работе [2] рассмотрены основные подходы к решению задачи поиска дорожных знаков в кадре. Все методы можно разделить на несколько групп:

1. Основанные на цветовых пространствах. Используется цветовая сегментация для поиска областей интереса. Как правило, к сегментированным изображениям применяются пороговые преобразования, либо иные алгоритмы, позволяющие отбросить максимальное количество лишней информации.
2. Основанные на поиске геометрических фигур. Наиболее часто используют преобразования Хафа и их модификации.
3. Методы, основанные на машинном обучении. Чаще всего – метод Виолы-Джонса, представляющий собой комбинацию признаков, похожих на признаки Хаара и алгоритма обучения – вариации AdaBoost.

Остановимся подробнее на методах цветовой сегментации изображения.

В работе [3] предлагается алгоритм разделения RGB-изображения на цветовые каналы и дальнейшей обработки полученных слоев.

В работе [4] в качестве основы для сегментации предложено изображение в цветовом пространстве HSV. Такое представление позволяет легко отделить красные и синие пиксели, а красный и синий цвета наиболее часто используются в оформлении дорожных знаков. Пространство HSV, в отличие от RGB, позволяет определить цвет пикселя по одному компоненту (H-hue), не производя лишних вычислений.

В работах [5] и [6] используется цветовое пространство CIELab, работа с которым позволяет очень быстро отделить области интересующих цветов при помощи разделения изображения на отдельные каналы и дальнейшей бинаризации. Пространство Lab имеет ряд преимуществ:

- 1) В Lab значение светлоты отделено от значения хроматической составляющей цвета. Светлота хранится в канале L^* . По нему легко определяется в какое время суток получен кадр и в зависимости от этого применять разные методы классификации.
- 2) Хроматические каналы a и b обозначают положение цвета в разных диапазонах. Так канал a обозначает положение цвета в диапазоне от зеленого до красного, а b — от синего до желтого.

В работе [7] авторы предлагают преобразование изображения в оттенки серого с использованием формулы (1):

$$\Omega_{RB} = \max\left(\frac{R}{R+G+B}, \frac{B}{R+G+B}\right). \quad (1)$$

Такое преобразование позволяет получить изображение, на котором наиболее яркими будут области интереса. К полученному изображению затем применяется адаптированный алгоритм поиска максимальной стабильной области (MSER)[7][8], в котором снижено количество применяемых пороговых преобразований для повышения быстродействия.

Зачастую после выделения областей интереса необходима дополнительная верификация, для того чтобы отбросить ошибочно выделенные области. Верификация особенно нужна при перемещении по городу, поскольку рекламные щиты, вывески зачастую могут содержать фрагменты, которые система может ошибочно принять за дорожный знак.

Как видно из всего вышеперечисленного, для обнаружения знаков в видеопотоке используются подходы, которые сильно отличаются друг от друга. Тем не менее, все они позволяют локализовать дорожный знак в кадре, а значит, возможным становится распознавание.

1.2 Распознавание дорожных знаков

После обнаружения дорожного знака основной трудностью становится распознавание. Корректность распознавания является критически важной, поскольку предоставление ложной информации может привести к серьезным последствиям.

Однако, в условиях работы с видеопотоком существует противоречащее требование – классификатор должен работать максимально быстро, чтобы опеспечить распознавание в реальном времени.

Т.к. работа ведется с фрагментами изображений, то объем информации для распознавания получается достаточно большим. Без предварительной обработки обнаруженный знак размером 30×30 представляет собой $30 \times 30 \times 3$ байт информации. Такой объем непригоден для работы в реальном времени, и поэтому одной из главных задач является понижение размерности входных данных классификатора.

В работах [7-10] используются гистограммы ориентированных градиентов, которые позволяют значительно понизить размерность задачи. К примеру, для изображения 24×24 px строится вектор дескрипторов длиной 144 элемента, что значительно меньше полноразмерного изображения.

Авторы работы [11] используют горизонтальные, вертикальные гистограммы и средние значения цветовых компонентов, получая для изображения 30×30 вектор размерностью 63 элемента.

Для классификации, как правило, используется метод опорных векторов. Однако возможно и применение нейросетей.

Для эффективной работы классификация знаков происходит поэтапно. Первичные данные о классе знака можно получить уже при цветовой сегментации. Следующим этапом становится распознавание формы знака и уже после этого распознается содержимое.

При таком подходе к решению задачи возможно построение дерева классификаторов, ветви которого представляют собой независимые вычисления, пригодные для распараллеливания.

На рисунке 1 приведен пример построения дерева классификаторов, предложенный авторами работы [7]. При цветовой сегментации авторы отдельно рассматривают знаки с белым фоном и цветным контуром, и отдельно – цветные знаки. При таком разделении на категории получается достаточно оптимальная структура классификаторов.

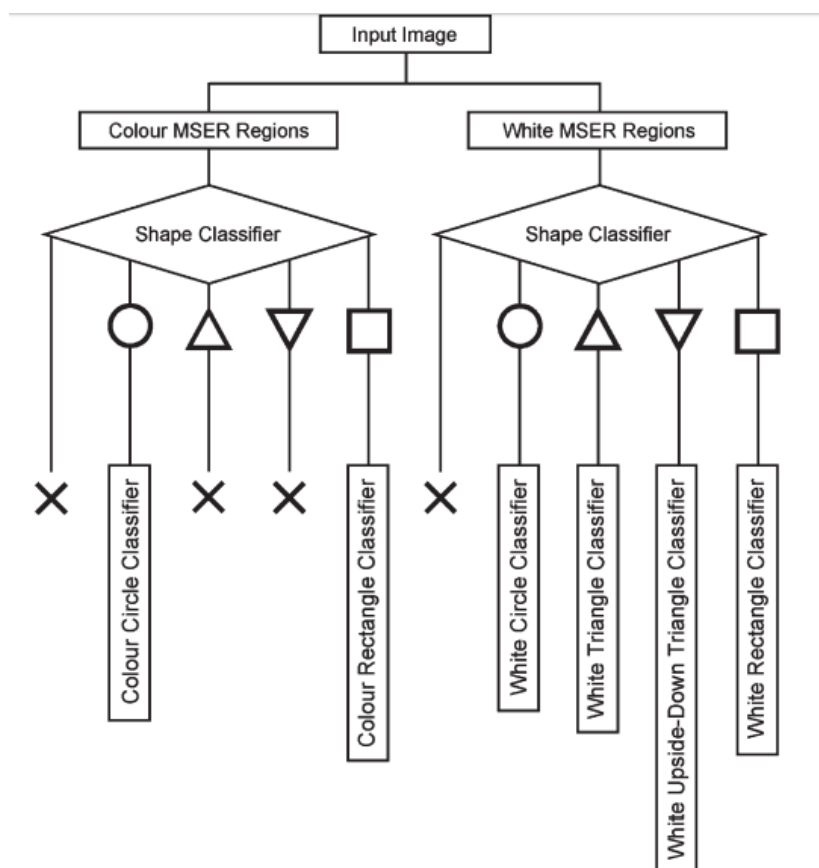


Рисунок 1 – Пример дерева классификаторов

При анализе существующих подходов становится очевидно, что задачу распознавания дорожных знаков можно решать при помощи параллельных алгоритмов. Главной проблемой становится построение классификатора таким образом, чтобы распараллеливание его работы положительно сказывалось на быстродействии системы.

2 Параллельный алгоритм распознавания дорожных знаков

В данной работе рассматривается параллельный алгоритм распознавания дорожных знаков в видеопотоке. Для исследования выбраны две категории знаков – запрещающие и предписывающие. Из каждой категории выбрано несколько знаков, на основе которых обучены классификаторы.

Данные категории выбраны для исследования, поскольку имеют разное цветовое оформление (рис. 2)



Рисунок 2 – примеры запрещающих и предписывающих знаков.

Алгоритм состоит из двух этапов – обнаружение знака в кадре и распознавание обнаруженного знака. Рассмотрим подробнее каждый этап.

2.1 Этап обнаружения знака в кадре

В качестве основы для алгоритма обнаружения выбран подход, описанный в работе [7], с некоторыми модификациями.

1. На первом этапе входное изображение преобразуется в оттенки серого. Отдельно выделяются красные области, отдельно – синие. Для вычисления яркости применяются формулы (2) и (3):

$$\Omega_R = \frac{R}{R+G+B} \quad (2),$$

$$\Omega_B = \frac{B}{R+G+B} \quad (3).$$

2. Полученные изображения затем обрабатываются с целью поиска стабильных областей. Для этого к изображениям применяется бинаризация по нескольким порогам, а полученные бинаризованные изображения затем суммируются. Таким образом области, остающиеся белыми при применении большего числа порогов, оказываются ярче в изображении-аккумуляторе. К полученному

аккумулятору затем применяется алгоритм шумоподавления, а затем бинаризация методом Оцу.

3. Последним шагом в поиске областей интереса является применение детектора границ Кенни. Результат работы детектора затем исследуется – выбираются только внешние контуры, размеры которых позволяют предположить наличие дорожного знака в данной области.

Ход работы детектора изображен на рисунках 3-5.



Рисунок 3 – тестовое изображение



Рисунок 4 – результат цветовой сегментации



Рисунок 5 – результат шага 2 (аккумулятор + шумоподавление + бинаризация Оцу)

На рисунке 6 выделена область интереса, полученная после шага 3 (работа детектора границ + фильтрация результатов)



Рисунок 6 – Обнаруженный в кадре дорожный знак

Предложенный алгоритм обнаружения показывает приемлемые результаты как для запрещающих знаков, так и для предписывающих. Различие детекторов заключается в наборе порогов, применяемых для бинаризации на шаге 2.

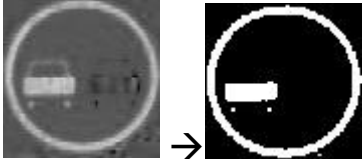
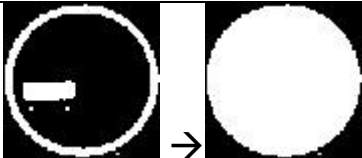


2.2 Этап распознавания

В качестве классификатора в данной работе выбрана нейронная сеть, на вход которой подается полутоновое изображение выделенного знака, предварительно обработанное для исключения лишних областей.

Выделенные на исходном изображении дорожные знаки предварительно обрабатываются. Целью обработки является избавление от лишних элементов – фона, контура (в случае с запрещающими знаками контур не является значимым для распознавания).

Для предварительной обработки так же используются изображения, фрагментированные по цвету. На их основании строится фильтр, который затем применяется к исходному изображению, переведенному в оттенки серого обычным способом. В таблице 2 приведен пример обработки запрещающего знака.

Таблица 2 – этапы предварительной обработки обнаруженных знаков

Этап	Изображение
Бинаризация сегментированного изображения	
Заливка замкнутой области (при заливке определяются границы заливаемой области)	
Применение операции ХоР к бинаризованному и изображению и результату заливки	
Наложение маски на полутоновое исходное изображение + обрезка краев, не относящихся к области заливки	

В результате получается изображение знака, с которого удалена лишняя информация – фон, красный контур.

Для обучения нейронной сети была сформирована небольшая выборка, основанная на открытой базе российских знаков [12]. Элементы выборки подвергались аналогичной обработке.

2.3 Схема алгоритма и ожидаемые результаты его применения

Обобщенная схема предложенного алгоритма приведена на рисунке 7.

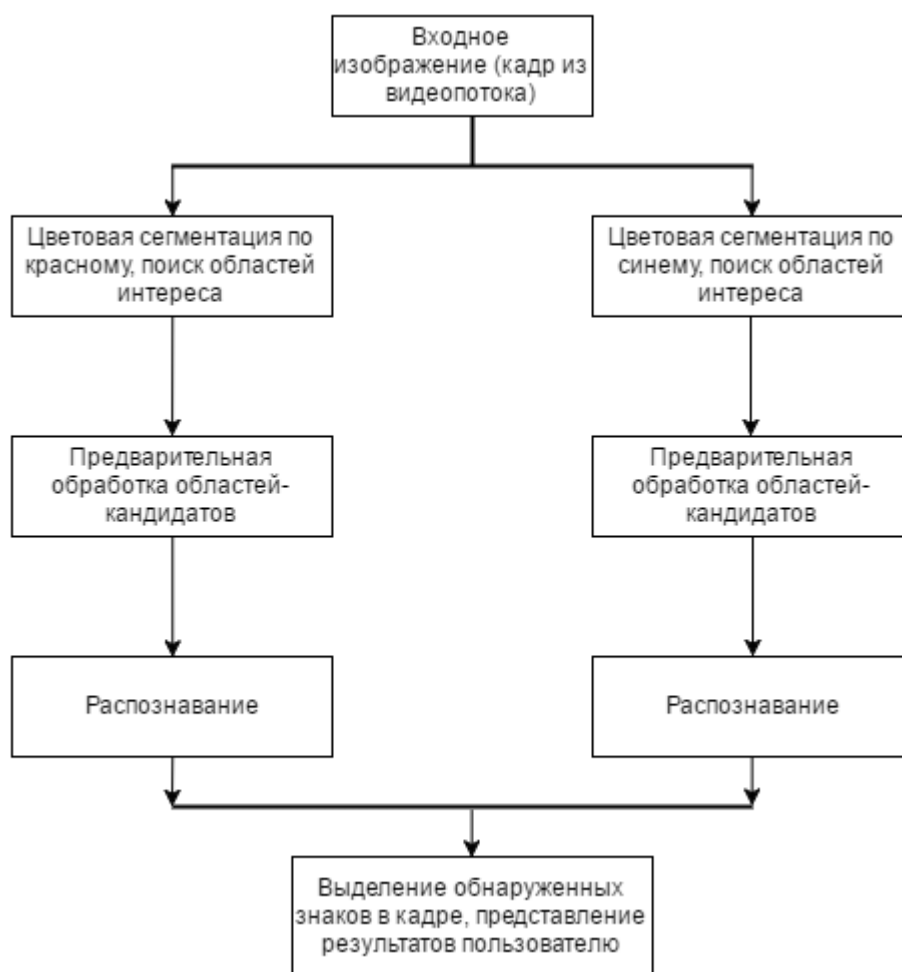


Рисунок 7 – обобщенная схема алгоритма поиска и распознавания.

Очевидно, что при таком построении процесс обработки можно выполнять параллельно. В данной работе поиск каждого типа знаков и дальнейшее распознавание выполняется в отдельном потоке. Распределение нагрузки в данном случае статическое. Детекторы для красных и синих знаков реализованы таким образом, что при параллельном исполнении

отсутствует гонка данных. Общими данными является исходное изображение, доступ к которому осуществляется только на чтение.

Однако, при работе с полным перечнем дорожных знаков процесс обработки кадра более сложен и включает в себя этап определения формы знака (как, например, в работе [7] – структура процесса обработки кадра приведена на рисунке 1). Для решения такой задачи непригодно статическое распределение задач между вычислительными узлами. Необходимо распределять нагрузку динамически.

Теоретическая оценка ускорения при описанном подходе к распределению задач сводится к тому, что общее время работы классификаторов будет примерно совпадать со временем работы более трудоемкого из них. Время работы каждого классификатора зависит от ряда факторов. Среди них и размер изображения, и его содержимое. Так, при преобладании красных областей в кадре выделение областей интереса, отбор пригодных к работе контуров будет дольше производиться для красного сегмента изображения, чем для синего.

В лучшем случае, если в кадре нет явного преобладания синего или красного цветов, и количество красных и синих знаков одинаковое, ожидается ускорение приблизительно вдвое.

3 Реализация

Для разработки системы был выбран язык C#. Параллельные вычисления реализованы при помощи библиотеки TPL(Task Parallel Library), входящей в состав .NET Framework. Работа с изображениями ведется при помощи библиотеки EmguCV – кроссплатформенной обертки над библиотекой OpenCV.

3.1 Структура ПО

Приложение представляет собой несколько модулей.

1. Модуль обнаружения дорожных знаков TrafficSign.Detection состоит из следующих классов:
 - DetectorBase – базовый класс, абстрактный. Содержит в себе базовые операции над изображением – цветовую сегментацию по выбранному каналу, определение границ. Кроме того, в данном классе реализован общий алгоритм поиска областей интереса
 - RedCircleDetector – наследник DetectorBase, содержит специфические операции по работе с красными областями
 - BlueCircleDetector – наследник DetectorBase, содержит специфические операции для работы с синими областями
2. Модуль распознавания TrafficSign.Recognition содержит в себе классы для работы с нейронной сетью. Нейросеть реализована набором классов NeuralLayer, NeuralNetwork, TrainingSet. Нейронная сеть после обучения сохраняется в формат XML, а затем используется в конечном приложении (считывается при инициализации)
3. Непосредственно приложение для распознавания. Состоит из демонстрационной формы.

3.2 Результат работы

Результаты работы программы представлены на рисунках 8-10

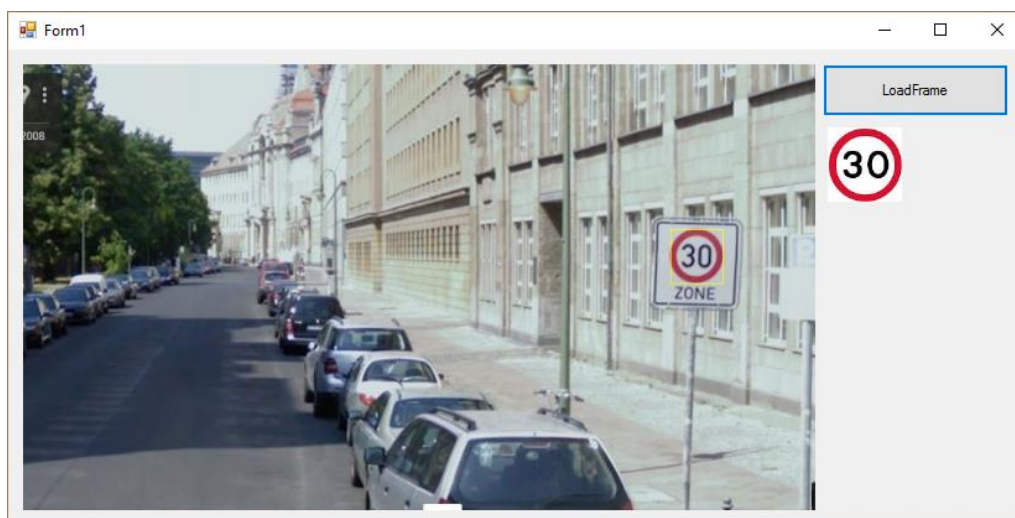


Рисунок 8- пример распознанного знака ограничения скорости



Рисунок 9 – распознанный знак «обгон запрещен»



Рисунок 10 – распознанный знак «объезд справа»

Фактический прирост производительности колеблется в пределах 15-50%. Однако, данные, полученные в ходе экспериментов, нельзя трактовать однозначно. В реальных задачах количество знаков, которые обнаруживаются и распознаются, намного больше. Гораздо сложнее иерархия классификаторов. Кроме того, в условиях максимальной оптимизации процессов локализации и распознавания накладные расходы, связанные с созданием потоков, могут оказаться достаточно велики и последовательный вариант выполнения может быть более быстроедейственным.

СПИСОК ИСТОЧНИКОВ

1. Vienna Convention on Road Signs and Signals [Электронный ресурс]//Wikipedia – Режим доступа: https://en.wikipedia.org/wiki/Vienna_Convention_on_Road_Signs_and_Signals
2. Brkic, K., An overview of traffic sign detection methods / Karla Brkic// Режим доступа: https://www.fer.unizg.hr/_download/repository/BrkicQualifyingExam.pdf
3. Deshmukh, Vishal R., Real-Time Traffic Sign Recognition System based on Color Image Segmentation / Vishal R. Deshmukh, G. K. Patnaik, M. E. Patil, International Journal of Computer Applications (0975 – 8887) Volume 83 – No3, December 2013 – Режим доступа: <http://research.ijcaonline.org/volume83/number3/pxc3892575.pdf>
4. Якимов, П.Ю. Предварительная обработка цифровых изображений в системах локализации и распознавания дорожных знаков./ Якимов, П.Ю. // Компьютерная оптика. –2013. – Том 37. - № 3. – С. 401-405. – Режим доступа: <http://www.computeroptics.smr.ru/KO/PDF/KO37-3/370316.pdf>
5. Lopez, L. Color-based road sign detection and tracking. Image Analysis and Recognition / L. Lopez, O. Fuentes // Lecture Notes in Computer Science. – Springer. – 2007. – P. 1138-1147. – Режим доступа: http://www.cs.utep.edu/ofuentes/lopez_ICIAR07.pdf
6. Уваров Д. А. Система автоматического распознавания дорожных знаков // Тезисы докладов XVI Всероссийской конференции молодых ученых по математическому моделированию и информационным технологиям. — Красноярск: Институт вычислительных технологий СО РАН, 2015. — С. 9 —95. — Режим доступа: <http://conf.nsc.ru/files/conferences/ym2015/298631/YM2015.pdf>.
7. Greedhalgh, J., Real-Time Detection and Recognition of Road Traffic Signs /Jack Greenhalgh and Majid Mirmehdi// IEEE TRANSACTIONS ON

- INTELLIGENT TRANSPORTATION SYSTEMS, VOL. 13, NO. 4,
DECEMBER 2012 – P. 1498-1506. – Режим доступа:
https://www.cs.bris.ac.uk/home/greenhal/jg_trans_its_2012.pdf
8. Greedhalgh, J., TRAFFIC SIGN RECOGNITION USING MSER AND
RANDOM FORESTS /Jack Greenhalgh and Majid Mirmehdi// EURASIP,
2012 – Режим доступа:
<http://bridgingthegaps.bristol.ac.uk/Publications/Papers/2001570.pdf>
9. Karthik, B., Identification Classification and Monitoring of Traffic Sign Using
HOG and Neural Networks / Karthik B , Hari Krishna Murthy , Mukul
Manohar// International Journal of Science and Research (IJSR), Volume 4
Issue 8, August 2015, P. 782-786 – Режим
доступа:<http://www.ijsr.net/archive/v4i8/SUB157353.pdf>
10. Zaklouta, F., Warning Traffic Sign Recognition using a HOG-based K-d Tree
/Fatin Zaklouta and Bogdan Stanciulescu// 2011 IEEE Intelligent Vehicles
Symposium(IV) P.109-1024 – Режим доступа:
http://www.lara.prd.fr/_media/users/iv2011.pdf
11. Lorsaku, A. Traffic Sign Recognition Using Neural Network on OpenCV:
Toward Intelligent Vehicle/Driver Assistance System /Auranuch Lorsaku and
Jackrit Suthakorn// Режим доступа -
http://bartlab.org/Dr.%20Jackrit's%20Papers/ney/1.TRAFFIC_SIGN_LorsakuISR.pdf
12. Traffic sign recognition [Электронный ресурс]//Graphics and Media Lab
Режим доступа:
<http://graphics.cs.msu.ru/en/research/projects/imagerecognition/trafficsign>

Приложение А

Код программы

```
using Emgu.CV;
using Emgu.CV.CvEnum;
using Emgu.CV.Structure;
using Emgu.CV.Util;
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TrafficSign.Detection.Common
{
    public abstract class DetectorBase
    {
        public abstract Image<Gray, Byte> FilterColorChannel(Image<Rgb, Byte> source);
        protected Image<Gray, Byte> FilterColorChannel(Image<Rgb, Byte> source, int
channelIndex)
        {
            Image<Gray, Byte>[] channels = source.Split();
            Image<Gray, Byte> accumulator = new Image<Gray, byte>(source.Width,
source.Height, new Gray(0));
            Image<Gray, Byte> result = new Image<Gray, byte>(source.Size);
            CvInvoke.AddWeighted(channels[0], 0.333, accumulator, 1, 1, accumulator);
            CvInvoke.AddWeighted(channels[1], 0.333, accumulator, 1, 1, accumulator);
            CvInvoke.AddWeighted(channels[2], 0.333, accumulator, 1, 1, accumulator);

            CvInvoke.Divide(channels[channelIndex], accumulator, result, 255 / 3.0);
            return result;
        }

        public void Highlight(List<Rectangle> ROIs, Image<Rgb, byte> target)
        {
            foreach (var roi in ROIs)
            {
                CvInvoke.Rectangle(target, roi, new MCvScalar(255, 255, 0));
            }
        }

        public abstract void ThresholdDetection(Image<Gray, byte> img, Image<Gray, byte>
accumulator);

        public List<Rectangle> FilterContours(double cannyThreshold, double
cannyThresholdLinking, Image<Gray, byte> accumulator)
        {
            List<Rectangle> ROIs = new List<Rectangle>();
            UMat cannyEdges = new UMat();
            CvInvoke.Canny(accumulator, cannyEdges, cannyThreshold,
cannyThresholdLinking);
            using (VectorOfVectorOfPoint contours = new VectorOfVectorOfPoint())
            {
                Mat hierarchy = new Mat();
```



```

        CvInvoke.FindContours(cannyEdges, contours, hierarchy, RetrType.External,
ChainApproxMethod.ChainApproxSimple);
        for (int i = 0; i < contours.Size; i++)
        {
            var rectangle = CvInvoke.BoundingRectangle(contours[i]);
            double ratio = (1.0*rectangle.Size.Width) / rectangle.Size.Height;
            int min = accumulator.Rows / 10, max = accumulator.Rows / 3;
            if (rectangle.Size.Width > min && rectangle.Size.Height > min
                && rectangle.Size.Height < max && rectangle.Size.Width < max
                && ratio > 0.75 && ratio < 1.25)
                ROIs.Add(rectangle);
        }
    }
    return ROIs;
}

```

```

    public List<Rectangle> FindROIs(Image<Gray, Byte> img, Image<Gray, byte>
accumulator)
    {
        double cannyThreshold = 180;
        double cannyThresholdLinking = 120;

        UMat uimage = img.ToUMat();
        Image<Gray, Byte> filterBuffer = new Image<Gray, byte>(img.Size);

        ThresholdDetection(img, accumulator);

        List<Rectangle> rois = FilterContours(cannyThreshold, cannyThresholdLinking,
accumulator);
        return rois;
    }

```

```

    public abstract Image<Gray, Byte> FindObjectOfInterest(Image<Gray, byte> img,
Image<Gray, byte> accumulator, Size desiredSize);
}

```

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Emgu.CV;
using Emgu.CV.Structure;
using System.Drawing;
using Emgu.CV.CvEnum;

```

```

namespace TrafficSign.Detection.Red
{

```

```

    public class RedCirclesDetector : Common.DetectorBase
    {
        public override Image<Gray, byte> FilterColorChannel(Image<Rgb, byte> source)
        {
            return base.FilterColorChannel(source, 0);
        }

        public override Image<Gray, byte> FindObjectOfInterest(Image<Gray, byte> img,
Image<Gray, byte> accumulator, Size desiredSize)
        {
            var binarized = new Image<Gray, byte>(img.Size);

```

```

        CvInvoke.Threshold(accumulator, binarized, 0, 255, ThresholdType.Binary |
ThresholdType.Otsu);
        Image<Gray, Byte> filled = new Image<Gray, byte>(binarized.Size);
        var cpmsk = new Image<Gray, Byte>(binarized.Cols, binarized.Rows, new
Gray(255));
        CvInvoke.cvCopy(binarized, filled, cpmsk);
        Image<Gray, byte> mask = new Image<Gray, byte>(filled.Width + 2,
filled.Height + 2);
        MCvConnectedComp comp = new MCvConnectedComp();
        Point point1 = new Point(filled.Cols / 2, filled.Rows / 2);
        Rectangle rect;
        CvInvoke.FloodFill(filled, mask, point1, new MCvScalar(255),
            out rect,
            new MCvScalar(0),
            new MCvScalar(0));
        var masked = filled.Xor(binarized);
        Image<Gray, Byte> gsImg = new Image<Gray, byte>(img.Size);
        var resultRaw = img.And(masked);
        resultRaw.ROI = rect;
        var result = new Image<Gray, Byte>(resultRaw.Size);
        CvInvoke.cvCopy(resultRaw, result, IntPtr.Zero);
        result = result.Resize(desiredSize.Width, desiredSize.Height, Inter.Linear);
        return result;
    }

    public override void ThresholdDetection(Image<Gray, byte> img, Image<Gray, byte>
accumulator)
    {
        UMat uimage = img.ToUMat();
        Image<Gray, Byte> filterBuffer = new Image<Gray, byte>(img.Size);

        int threshmin = 80, threshmax = 180, step = 20;
        double alpha = (double)step / (threshmax - threshmin);
        for (int t = threshmin; t <= threshmax; t += step)
        {
            CvInvoke.Threshold(uimage, filterBuffer, t, 255,
Emgu.CV.CvEnum.ThresholdType.Binary);
            CvInvoke.AddWeighted(filterBuffer, alpha, accumulator, 1, 0,
accumulator);
        }
        CvInvoke.PyrDown(accumulator, filterBuffer);
        CvInvoke.PyrUp(filterBuffer, accumulator);
        CvInvoke.Threshold(accumulator, accumulator, 100, 255, ThresholdType.ToZero);
    }
}

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Emgu.CV;
using Emgu.CV.Structure;
using Emgu.CV.CvEnum;

namespace TrafficSign.Detection.Blue
{
    public class BlueCirclesDetection : Common.DetectorBase
    {
        public override Image<Gray, byte> FilterColorChannel(Image<Rgb, byte> source)

```

```

        {
            return base.FilterColorChannel(source, 2);
        }
        public override void ThresholdDetection(Image<Gray, byte> img, Image<Gray, byte>
accumulator)
        {
            UMat uimage = img.ToUMat();
            Image<Gray, Byte> filterBuffer = new Image<Gray, byte>(img.Size);

            int threshmin = 60, threshmax = 180, step = 10;
            double alpha = (double)step / (threshmax - threshmin);
            for (int t = threshmin; t <= threshmax; t += step)
            {
                CvInvoke.Threshold(uimage, filterBuffer, t, 255,
Emgu.CV.CvEnum.ThresholdType.Binary);
                CvInvoke.AddWeighted(filterBuffer, alpha, accumulator, 1, 0,
accumulator);
            }
            CvInvoke.PyrDown(accumulator, filterBuffer);
            CvInvoke.PyrUp(filterBuffer, accumulator);
            CvInvoke.Threshold(accumulator, accumulator, 100, 255, ThresholdType.ToZero);
        }
        public override Image<Gray, byte> FindObjectOfInterest(Image<Gray, byte> img,
Image<Gray, byte> accumulator, Size desiredSize)
        {
            var binarized = new Image<Gray, byte>(img.Size);
            CvInvoke.Threshold(accumulator, binarized, 0, 255, ThresholdType.BinaryInv |
ThresholdType.Otsu);
            Image<Gray, Byte> filled = new Image<Gray, byte>(binarized.Size);
            var cpmsk = new Image<Gray, Byte>(binarized.Cols, binarized.Rows, new
Gray(255));
            CvInvoke.cvCopy(binarized, filled, cpmsk);
            Image<Gray, byte> mask = new Image<Gray, byte>(filled.Width + 2,
filled.Height + 2);

            Point point1 = new Point(filled.Cols / 2, filled.Rows / 2);
            while(binarized.Data[point1.Y, point1.X, 0] != 0 && point1.Y > 2 && point1.X
< filled.Width-2)
            {
                point1.X += 2;
                point1.Y -= 2;
            }
            if(binarized.Data[point1.Y, point1.X, 0] !=0)
            {
                return img;
            }
            Rectangle rect;
            CvInvoke.FloodFill(filled, mask, point1, new MCvScalar(255),
out rect,
new MCvScalar(0),
new MCvScalar(0));

            img.ROI = rect;
            var result = new Image<Gray, Byte>(img.Size);
            CvInvoke.cvCopy(img, result, IntPtr.Zero);
            result = result.Resize(desiredSize.Width, desiredSize.Height, Inter.Linear);
            return result;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;

```

```

using System.Text;

namespace nn.classes
{
    /// <summary>
    /// Класс, описывающий слой полносвязной нейросети.
    /// Слой характеризуется числом входов,
    /// нейронов и матрицей весов.
    /// Параметры задаются.
    /// </summary>
    class NeuralLayer
    {
        #region fields
        /// <summary>
        /// генератор синапсов для начальной инициализации сети
        /// создаю статическим, чтоб для всех слоев работал один генератор
        /// </summary>
        private static Random randomGenerator = new Random();
        /// <summary>
        /// число входов и выходов слоя нейросети
        /// </summary>
        private int outputsCount, inputsCount;
        /// <summary>
        /// матрица весов, [число входов * число нейронов]
        /// </summary>
        private double[,] weights;
        /// <summary>
        /// состояние выходов на текущий момент
        /// </summary>
        private double[] lastOut;
        /// <summary>
        /// Накопленное на входах в данный момент
        /// </summary>
        private double[] lastIn;
        /// <summary>
        /// Храним производные по выходам, входам и весам
        /// </summary>
        private double[] dEdY;
        private double[] dEdX;
        private double[,] dEdW;
        private const int Bias =1;
        #endregion
        #region methods
        /// <summary>
        /// Инициализирует новый слой нейросети (веса не генерятся)
        /// </summary>
        /// <param name="inputsCount">число входов</param>
        /// <param name="outputsCount">число выходов</param>
        public NeuralLayer( int inputsCount,int outputsCount)
        {
            this.outputsCount = outputsCount;
            this.inputsCount = inputsCount;
            dEdY = new double[outputsCount];
            dEdX = new double[outputsCount];
            dEdW = new double[inputsCount + Bias, outputsCount];
        }
        /// <summary>
        /// Генерирует случайные входные веса
        /// </summary>
        public void GenerateWeights()
        {
            //выделим память под веса
            weights = new double[inputsCount+Bias,outputsCount];
            //всю матрицу

```

```

        for (int i = 0; i < inputsCount+Bias; i++)
        {
            for (int j = 0; j < outputsCount; j++)
                //заполним случайными числами
                weights[i,j] = randomGenerator.NextDouble()-0.5;
        }
    }
    /// <summary>
    /// Для последнего слоя
    /// </summary>
    /// <param name="idealOut">Идеальный выходной вектор</param>
    public void CountOutDetivativesAsLast(double[] idealOut)
    {
        for (int i = 0; i < outputsCount; i++)
        {
            //простейшая формула производной от ошибки
            dEdY[i] = lastOut[i] - idealOut[i]; //dE/dYk3
        }
    }
    /// <summary>
    /// Считает производные по выходам промежуточного слоя
    /// </summary>
    /// <param name="dEdXPrev">Производные по входам слоя, следующего за данным
    /// (того, в который входят связи, исходящие из данного слоя)</param>
    /// <param name="wPrev">Веса связей, исходящих из данного слоя
    /// (берем набор связей, входящих в следующий слой)</param>
    public void CountOutDerivatives(double[] dEdXPrev, double[,] wPrev)
    {
        //по каждому выходу
        for (int n = 0; n < outputsCount; n++)
        {
            //производная есть сумма, т.к.  $E = E(x_1(y_1, y_2, \dots, y_n), \dots, x_k(y_1, y_2, \dots, y_n))$ 
            //k - число входов след. слоя
            //n - число выходов текущего слоя
            //т.о. на каждой веточке вносится вклад в ошибку ~ весу
            double dEdYn = 0.0;
            for (int i = 0; i < dEdXPrev.Length; i++)
            {
                dEdYn += dEdXPrev[i] * wPrev[n, i];
            }
            dEdY[n] = dEdYn;
        }
    }
    /// <summary>
    /// Считает производные по входам слоя
    /// </summary>
    public void CountInDerivatives()
    {
        //проходя через слой, получаем  $X \rightarrow Y$ ,  $Y = Sg(X)$ 
        //производная сигмоида выражается через его значение
        for (int i = 0; i < outputsCount; i++)
        {
            double dYdX = Sigmoid.betta * lastOut[i] * (1 - lastOut[i]);
            dEdX[i] = dEdY[i] * dYdX;
        }
    }
    /// <summary>
    /// Считает производные по весам
    /// </summary>
    /// <param name="yprev">Выход предыдущего слоя</param>
    public void CountWeightDerivatives(double[] yprev)
    {
        //yprev.Lenght = inputsCount
        //dEdX.Lenght = outputsCount
    }

```

```

//Вход для каждого нейрона - линейная комбинация
//выходов предыдущего (к-т равен весу). Т.о. производная по ij весу
//(для i входа j нейрона) равна i компоненте входного в-ра данного слоя
//т.е. выходного в-ра предыдущего
for (int i = 0; i < inputsCount+Bias; i++)
{
    for (int j = 0; j < outputsCount; j++)
    {
        if (i == inputsCount)
        {
            dEdW[i, j] = dEdX[j] * 1;
        }
        else
        {
            dEdW[i, j] = dEdX[j] * yprev[i];
        }
    }
}
}
/// <summary>
/// Обновляет веса на основании рассчитанных производных
/// </summary>
/// <param name="delta"></param>
public void Update(double delta)
{
    for(int i=0;i<inputsCount + Bias;i++)
        for (int j = 0; j < outputsCount; j++)
        {
            weights[i, j] -= delta * dEdW[i, j];
        }
}
private double mc = 0.5;

private double[,] _dxPrev;
public void UpdateWithMomentum(double delta)
{
    //dX = lr * dperf / dX
    //dX = delta * dEdW

    //normal

    //dX = mc * dXprev + delta * (1 - mc) * dperf / dX
    //dX = mc * delta * _savedDeDw + delta * (1-mc)* DEDW
    //momentum
    //
    if (_dxPrev == null)
    {
        _dxPrev = new double[dEdW.GetLength(0), dEdW.GetLength(1)];

        for (int i = 0; i < inputsCount + Bias; i++)
        {
            for (int j = 0; j < outputsCount; j++)
            {
                _dxPrev[i, j] = delta * dEdW[i, j];
                weights[i, j] -= _dxPrev[i, j];
            }
        }
    }
    else
    {
        for (int i = 0; i < inputsCount + Bias; i++)
        {
            for (int j = 0; j < outputsCount; j++)
            {

```

```

        _dxPrev[i, j] = mc * _dxPrev[i, j] + delta * (1 - mc) * dEdW[i,
j];
        weights[i, j] -= _dxPrev[i, j];
    }
}

}
}
#endregion
#region properties
/// <summary>
/// Возвращает или устанавливает число входов
/// </summary>
public int InputsCount { get { return inputsCount; } set { inputsCount = value; }
}

/// <summary>
/// Возвращает или устанавливает число выходов
/// </summary>
public int OutputsCount { get { return outputsCount; } set { outputsCount =
value; } }

/// <summary>
/// Возвращает или устанавливает вес
/// </summary>
/// <param name="inp">Номер нейрона в слое</param>
/// <param name="outp">Номер входа</param>
/// <returns></returns>
public double this[int inp, int outp]
{
    get { return weights[inp, outp]; }
    set { weights[inp, outp] = value; }
}

/// <summary>
/// состояние всех нейронов слоя
/// </summary>
public double[] LastOut { get { return lastOut; } set { lastOut = value; } }
/// <summary>
/// входные сигналы
/// </summary>
public double[] LastIn { get { return lastIn; } set { lastIn = value; } }
public double[] DEDX { get { return dEdX; } }
public double[,] W { get { return weights; } }

private double[] _savedLastOut;
private double[] _savedLastIn;
private double[] _savedDEX;
private double[] _savedDEY;
private double[,] _savedW;
private double[,] _savedDEW;

public void RestoreState()
{
    for (int i = 0; i < _savedLastOut.Length; i++)
    {
        lastOut[i] = _savedLastOut[i];
    }

    for (int i = 0; i < _savedLastIn.Length; i++)
    {
        lastIn[i] = _savedLastIn[i];
    }

    for (int i = 0; i < _savedDEX.Length; i++)

```

```

    {
        dEdX[i] = _savedDEDX[i];
    }

    for (int i = 0; i < _savedDEDY.Length; i++)
    {
        dEdY[i] = _savedDEDY[i];
    }

    for (int i = 0; i < _savedW.GetLength(0); i++)
    {
        for (int j = 0; j < _savedW.GetLength(1); j++)
        {
            weights[i, j] = _savedW[i, j];
        }
    }

    for (int i = 0; i < _savedDEDW.GetLength(0); i++)
    {
        for (int j = 0; j < _savedDEDW.GetLength(1); j++)
        {
            dEdW[i, j] = _savedDEDW[i, j];
        }
    }
}

public void SaveState()
{
    if (_savedLastOut == null)
    {
        _savedLastOut = new double[lastOut.Length];
    }
    for (int i = 0; i < _savedLastOut.Length; i++)
    {
        _savedLastOut[i] = lastOut[i];
    }

    if (_savedLastIn == null)
    {
        _savedLastIn = new double[lastIn.Length];
    }
    for (int i = 0; i < _savedLastIn.Length; i++)
    {
        _savedLastIn[i] = lastIn[i];
    }

    if (_savedDEDX == null)
    {
        _savedDEDX = new double[dEdX.Length];
    }
    for (int i = 0; i < _savedDEDX.Length; i++)
    {
        _savedDEDX[i] = dEdX[i];
    }

    if (_savedDEDY == null)
    {
        _savedDEDY = new double[dEdY.Length];
    }
    for (int i = 0; i < _savedDEDY.Length; i++)
    {
        _savedDEDY[i] = dEdY[i];
    }
}

```



```

        if (_savedW == null)
        {
            _savedW = new double[weights.GetLength(0), weights.GetLength(1)];
        }
        for (int i = 0; i < _savedW.GetLength(0); i++)
        {
            for (int j = 0; j < _savedW.GetLength(1); j++)
            {
                _savedW[i, j] = weights[i, j];
            }
        }

        if (_savedDEDW == null)
        {
            _savedDEDW = new double[dEdW.GetLength(0), dEdW.GetLength(1)];
        }
        for (int i = 0; i < _savedDEDW.GetLength(0); i++)
        {
            for (int j = 0; j < _savedDEDW.GetLength(1); j++)
            {
                _savedDEDW[i, j] = dEdW[i, j];
            }
        }
    }
}
#endregion

public StringBuilder GetWeightsStringRepr()
{
    StringBuilder sb = new StringBuilder();
    for (int i = 0; i < inputsCount; i++)
    {
        for (int j = 0; j < outputsCount; j++)
        {
            sb.AppendFormat("{0} ", W[i, j]);
        }
        sb.Append("\n");
    }
    return sb;
}

public void Parse(string s)
{
    var ws = s.Split(new char[] { '\n', ' ' },
StringSplitOptions.RemoveEmptyEntries);
    int idx = 0;
    weights = new double[inputsCount + Bias, outputsCount];
    for (int i = 0; i < inputsCount; i++)
    {
        for (int j = 0; j < outputsCount; j++)
        {
            weights[i, j] = Double.Parse(ws[idx++]);
        }
    }
}
}

}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Xml;

namespace nn.classes

```

```

{
class Sigmoid
{
    #region static members
    public static readonly double betta = 1.0;
    /// <summary>
    /// сигмоидальная функция нейрона
    /// </summary>
    /// <param name="input">Вход</param>
    /// <returns>Значение Sg(x)</returns>
    public static double Sg(double input)
    {
        return (1 / (1 + Math.Exp(-betta * input)));
    }
    #endregion
}
/// <summary>
/// Класс, описывающий нейронную сеть,
/// которая может состоять из множества слоев.
/// Число входов, выходов и размеры слоев задаются.
/// </summary>
public class NeuralNetwork
{
    #region fields
    /// <summary>
    /// массив слоев сети
    /// </summary>
    private NeuralLayer[] layers;
    /// <summary>
    /// число входов, выходов, слоев
    /// </summary>
    private int IN=0, OUT=0, layersCount=0;
    private double delta;
    private Method _method;

    #endregion

    #region methods
    /// <summary>
    /// Генерирует полносвязную многослойную
    /// нейросеть
    /// </summary>
    /// <param name="inputs">Число входных сигналов</param>
    /// <param name="layersSizes">Набор параметров переменной длины,
    /// содержит размеры слоев сети</param>
    public NeuralNetwork(int inputs, int[] layersSizes)
    {
        //число слоев определяется длиной массива с их размерами
        layersCount = layersSizes.Length;
        //выделим память под слои
        layers = new NeuralLayer[layersCount];
        //текущее число входных сигналов.
        //сначала равно числу входов всей сети
        int currentInputs = inputs;
        //создадим слои сети
        for (int i = 0; i < layersCount; i++)
        {
            //создадим слой с числом входов currentInputs
            //и числом нейронов (и выходов) равным его размеру
            layers[i] = new NeuralLayer(currentInputs, layersSizes[i]);
            //сгенерируем синапсы (весовые коэффициенты)
            //случайным образом

```

```

        layers[i].GenerateWeights();
        //для следующего слоя размер текущего
        //станет числом входов
        currentInputs = layersSizes[i];
    }
    //число входов известно
    IN = inputs;
    //число выходов = размер последнего слоя
    OUT = currentInputs;
}
/// <summary>
/// Рассчитывает выходной вектор для нейросети
/// </summary>
/// <param name="inX">Вектор входных сигналов</param>
/// <param name="outY">Вектор выхода</param>
public void NetworkOut(double[] inX, out double[] outY)
{
    double[] input = inX;
    //по всем слоям сети
    for (int i = 0; i < layersCount; i++)
    {
        double[] output;
        double[] currentInput;
        output = new double[layers[i].OutputsCount];
        currentInput = new double[layers[i].OutputsCount];
        //по всем нейронам в слое
        for (int j = 0; j < layers[i].OutputsCount; j++)
        {
            //расчитаем входное значение
            double arg = 0.0;
            //по всем входам нейрона
            for (int k = 0; k < layers[i].InputsCount+1; k++)
            {
                //на каждом шаге надо добавить очередной "вклад"
                //для j нейрона в i слое по k входу:
                if (k == layers[i].InputsCount)
                {
                    //bias
                    arg += 1 * layers[i][k, j];
                }
                else
                {
                    arg += input[k] * layers[i][k, j];
                }
            }
            currentInput[j] = arg;
            //выход нейрона = сигмоидальная функция от входного значения
            output[j] = Sigmoid.Sg(currentInput[j]);
        }
        //сохраним состояние слоя
        layers[i].LastOut = output;
        layers[i].LastIn = currentInput;
        //выход текущего слоя становится входом для следующего
        input = output;
    }
    //когда прошли через выходной слой, в input попал как раз
    //его выходной вектор возбуждения
    outY = input;
}
/// <summary>
/// Расчет ошибки сети
/// </summary>
/// <param name="realVector">Вектор возбуждения выходных нейронов</param>
/// <param name="ideal">"Идеальный" вектор - желаемый ответ сети</param>
/// <returns></returns>

```

```

private double E(double[] realVector, double[] ideal)
{
    double Err = 0.0;
    for (int i = 0; i < realVector.Length; i++)
        Err += (realVector[i] - ideal[i]) * (realVector[i] - ideal[i]);
    return Err / 2;
}
/// <summary>
/// Настройка весов под вектора входа и желаемого выхода
/// </summary>
/// <param name="ideal"></param>
/// <param name="inX"></param>
private void CorrectError(double[] ideal, double[] inX, Method method)
{
    //надо пройти по всем слоям и посчитать производные
    for (int i = layersCount - 1; i >= 0; i--)
    {
        //Если слой выходной, считаем ошибку по выходу
        //из желаемого вектора
        if (i == layersCount - 1)
            layers[i].CountOutDerivativesAsLast(ideal);
        //для промежуточных слоев - используем исходящие веса и
        //производные по входам для исходящих связей
        else
            layers[i].CountOutDerivatives(layers[i + 1].DEDX, layers[i + 1].W);
        //перепрыгиваем с выхода на вход
        layers[i].CountInDerivatives();
        //посчитаем производные по слою
        //если это входной слой, то для него входным вектором служит inX
        if (i == 0)
            layers[i].CountWeightDerivatives(inX);
        //Если слой промежуточный, то на его входе "висит"
        //выход предыдущего
        else
            layers[i].CountWeightDerivatives(layers[i - 1].LastOut);
    }
    //Все dE/dW посчитаны, обновим веса
    for (int i = layersCount - 1; i >= 0; i--)
    {
        if (_method == Method.Momentum)
        {
            layers[i].UpdateWithMomentum(delta);
        }
        else
        {
            layers[i].Update(delta);
        }
    }
}
/// <summary>
/// Функция обучения нейросети
/// </summary>
/// <param name="sample">Обучающая выборка</param>
/// <param name="ideals">Идеальные вектора</param>
/// <param name="Epochs">Число итераций</param>
/// <param name="e0">Порог допустимой ошибки</param>
/// <param name="sampleSize">Размер выборки</param>
/// <returns>"Successfull" - если за Epochs итераций ошибка стала меньше
порогового значения
/// "Unsuccessfull" - если по прошествии Epochs итераций ошибка все еще больше
пороговой</returns>
public String Train(double[][] sample, double[][] ideals, int Epochs, double e0,
    int sampleSize, int displayStep)
{

```

```

//накопленная ошибка итерации
double EpochError=0;
double oldError;
//номер текущей итерации
int epoch;
//цикл по всем итерациям
for (epoch = 0; epoch < Epochs; epoch++)
{
    oldError = EpochError;
    //обнулить накопленную ошибку
    EpochError = .0;
    if (Method == Method.Adaptive && epoch > 0)
    {
        foreach (var layer in layers)
        {
            layer.SaveState();
        }
    }

    //по всем векторам выборки
    for (int i = 0; i < sampleSize; i++)
    {
        //возьмем текущий вектор выборки
        //и идеальный для него выходной вектор
        double[] Sample = sample[i];
        double[] Ideal = ideals[i];
        double[] nwOut;
        //пропустим вектор из выборки через сеть
        this.NetworkOut(Sample, out nwOut);
        //расчитаем ошибку сети для данного элемента выборки
        EpochError += E(nwOut, Ideal);
        //скорректируем веса методом backpropagation
        CorrectError(Ideal,Sample,_method);
    }

    if (Method == Method.Adaptive && epoch>0)
    {
        if (EpochError < oldError)
        {
            delta = delta * 1.02;
        }
        if (EpochError > oldError * 1.04)
        {
            foreach (var layer in layers)
            {
                layer.RestoreState();
            }
            delta = delta * 0.7;
        }
    }

    if (EpochError < e0) break;
}
//проверим, по какому условию завершился цикл обучения
if (epoch < Epochs) return "Successfull";
if (EpochError > e0) return "Unsuccessfull";
return "Finished";
}
#endregion
public double Delta { get { return delta; } set { delta = value; } }

public Method Method

```

```

    {
        get
        {
            return _method;
        }

        set
        {
            _method = value;
        }
    }

    public void Save(string fileName)
    {
        using (XmlWriter writer = XmlWriter.Create(fileName))
        {
            writer.WriteStartElement("NeuralNetwork");
            writer.WriteElementString("LayerCount", layersCount.ToString());
            writer.WriteElementString("ClassesCount", layers[layersCount -
1].OutputsCount.ToString());
            writer.WriteStartElement("Layers");
            for (int i = 0; i < layersCount; i++)
            {
                writer.WriteElementString("LayerInputs",
layers[i].InputsCount.ToString());

                writer.WriteEndElement();
                for (int i = 0; i < layersCount; i++)
                {
                    writer.WriteElementString("Layer",
layers[i].GetWeightsStringRepr().ToString());
                }
                writer.WriteEndElement();
            }
        }
    }

    public NeuralNetwork(string fileName)
    {
        using (XmlReader reader = XmlReader.Create(fileName))
        {
            List<int> szList = new List<int>();
            reader.Read();
            reader.Read();
            if (reader.Name != "NeuralNetwork") return;
            reader.Read();
            this.layersCount = reader.ReadElementContentAsInt();

            this.OUT = reader.ReadElementContentAsInt();
            this.layers = new NeuralLayer[layersCount + 1];
            reader.Read();
            if (reader.Name != "LayerInputs") return;
            for (int i = 0; i < layersCount; i++)
            {
                szList.Add(reader.ReadElementContentAsInt());
            }
            szList.Add(OUT);
            reader.Read();
            for (int i = 0; i < layersCount; i++)
            {
                layers[i] = new NeuralLayer(szList[i], szList[i + 1]);
            }

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

```

```

        {
            layers[i].Parse(reader.ReadElementContentAsString());
        }
    }
}

public enum Method
{
    Normal,
    Adaptive,
    Momentum,
}
}
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;

namespace nn.classes
{
    public class TrainingSet
    {
        private static Char [] separators = new Char[] { ' ' };

        private int _fullSamplesCount;
        private int _step;
        public int SampleWidth { get; private set; }
        public int SampleHeight { get; private set; }

        public int SampleSize
        {
            get; private set;
        }

        public int SamplesCount { get; private set; }

        public bool IsBinary { get; private set; }

        public double[][] Samples { get; private set; }

        public double[][] Answers { get; private set; }

        public int ClassesCount { get; private set; }

        public TrainingSet(string filePath, int step)
        {
            _step = step;
            using (FileStream fileStream = new FileStream(filePath, FileMode.Open))
            {
                using (StreamReader reader = new StreamReader(fileStream))
                {

                    string trainingInfoString = reader.ReadLine();
                    ParseInfo(trainingInfoString);
                    Prepare();

                    for (int sampleIndex = 0; sampleIndex < _fullSamplesCount;
sampleIndex++)
                    {
                        //parse one sample

```

```

        StringBuilder sampleStringBuilder = new StringBuilder();
        for (int i = 0; i < SampleHeight; i++)
        {
            sampleStringBuilder.Append(" ");
            sampleStringBuilder.Append(reader.ReadLine());
        }
        if (sampleIndex % step == 0)
        {
            string sampleContent = sampleStringBuilder.ToString().Trim();
            string[] sampleValuesBuffer = sampleContent.Split(separators,
StringSplitOptions.RemoveEmptyEntries);

            for (int i = 0; i < SampleSize; i++)
            {
                double currentValue;
                double.TryParse(sampleValuesBuffer[i], out currentValue);
                Samples[sampleIndex / _step][i] = currentValue;
            }
        }
        int expectedClass;
        int.TryParse(reader.ReadLine(), out expectedClass);
        if (sampleIndex % step == 0)
        {
            Answers[sampleIndex / _step][expectedClass - 1] = 1.0;
        }
    }
}

private void Prepare()
{
    Samples = new double[SamplesCount][];
    Answers = new double[SamplesCount][];
    for (int i = 0; i < SamplesCount; i++)
    {
        Samples[i] = new double[SampleSize];
        Answers[i] = new double[ClassesCount];
    }
}

private void ParseInfo(string trainingInfoString)
{
    int binaryFlag, width, height, classesCount, samplesCount;

    string[] infoBuffer = trainingInfoString.Split(separators,
StringSplitOptions.RemoveEmptyEntries);

    int.TryParse(infoBuffer[0], out binaryFlag);
    IsBinary = binaryFlag == 0;

    int.TryParse(infoBuffer[1], out width);
    int.TryParse(infoBuffer[2], out height);
    int.TryParse(infoBuffer[3], out classesCount);
    int.TryParse(infoBuffer[4], out samplesCount);

    SampleWidth = width;
    SampleHeight = height;
    ClassesCount = classesCount;
    _fullSamplesCount = samplesCount;
    SamplesCount = samplesCount / _step;
    if (_step > 1 && samplesCount % _step > 0)
    {
        SamplesCount += 1;
    }
}

```



```

        }
        SampleSize = SampleWidth * SampleHeight;
    }
}

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace TrafficSignRecognizer.Recognition.Neural
{
    public class PredictionResult
    {
        public int ClassId { get; private set; }
        public Rectangle ROI { get; private set; }
        public PredictionResult(double[] answer, Rectangle roi)
        {
            ROI = roi;
            double max = double.MinValue;
            int maxIdx = -1;
            for (int i = 0; i < answer.Length; i++)
            {
                if(answer[i]>max)
                {
                    max = answer[i];
                    maxIdx = i;
                }
            }
            ClassId = maxIdx + 1;
        }
    }
}

using Emgu.CV;
using Emgu.CV.Structure;
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using TrafficSign.Detection.Common;

namespace TrafficSign.Recognition.TrainingSetExtraction
{
    public class TextExtractor
    {
        DetectorBase _detector;
        public TextExtractor(DetectorBase detector)
        {
            _detector = detector;
        }
        public void PrepareData(string rootPath, int classesCount, Size sampleSize)
        {
            Mat img = new Mat();
            Image<Gray, float> inputMat = new Image<Gray, float>(sampleSize.Width,
sampleSize.Height);

            int fullSampleSize = sampleSize.Width * sampleSize.Height;

```

```

        //prepare full samples list
        int samplesCount = 0;
        string[][] fileNames = new string[classesCount][];
        for (int i = 0; i < classesCount; i++)
        {
            string[] files = Directory.GetFiles(string.Format("{0}/{1}", rootPath,
i));
            fileNames[i] = files;
            samplesCount += files.Length;
        }

        /**
        using (StreamWriter writer = new StreamWriter(string.Format("tstask{0}.txt",
_detector.GetType().Name)))
        {
            writer.WriteLine(string.Format("{0} {1} {2} {3} {4}", 1,
sampleSize.Height, sampleSize.Width, classesCount, samplesCount));
            for (int i = 0; i < classesCount; i++)
            {
                for (int j = 0; j < fileNames[i].Length; j++)
                {
                    Image<Rgb, Byte> source = new Image<Rgb, byte>(fileNames[i][j]);
                    Write(writer, GetImgOfInterest(source, sampleSize), i + 1);
                }
            }
        }

private Image<Gray, Byte> GetImgOfInterest(Image<Rgb, Byte> img, Size size)
{
    Image<Gray, Byte> filtered = _detector.FilterColorChannel(img);
    filtered.Bitmap.Save("fltr.jpg");

    Image<Gray, Byte> accumulator = new Image<Gray, byte>(img.Width, img.Height,
new Gray(0));
    _detector.ThresholdDetection(filtered, accumulator);
    return _detector.FindObjectOfInterest(img.Convert<Gray, Byte>(), accumulator,
size);
}

private void Write(StreamWriter writer, Image<Gray, Byte> img, int v)
{
    for (int i = 0; i < img.Height; i++)
    {
        for (int j = 0; j < img.Width; j++)
        {
            writer.Write(string.Format("{0} ", img.Data[i, j, 0]));
        }
        writer.Write(writer.NewLine);
    }
    writer.WriteLine(string.Format("{0}", v));
}

public void ExtractFromFile(string destPath, string fileName, Size size)
{
    Mat inputMat = CvInvoke.Imread(fileName,
Emgu.CV.CvEnum.LoadImageType.Grayscale);
    Image<Rgb, Byte> source = new Image<Rgb, byte>(fileName);
    using (StreamWriter writer = new StreamWriter(destPath, true))
    {
        Write(writer, GetImgOfInterest(source, size), -1);
    }
}

```

```

    }
}

}
using Emgu.CV;
using Emgu.CV.Structure;
using nn.classes;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Diagnostics;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using TrafficSign.Detection.Blue;
using TrafficSign.Detection.Red;
using TrafficSign.Recognition.Conversion;
using TrafficSign.Recognition.TrainingSetExtraction;
using TrafficSignRecognizer.Recognition.Neural;

namespace TrafficSignRecognizer
{
    public partial class MainForm : Form
    {
        RedCirclesDetector _reds = new RedCirclesDetector();
        BlueCirclesDetection _blues = new BlueCirclesDetection();
        NeuralNetwork _redNet = new NeuralNetwork("ad.xml");
        NeuralNetwork _blueNet = new NeuralNetwork("normal.xml");
        Dictionary<int, Bitmap> _redClasses = new Dictionary<int, Bitmap>();
        Dictionary<int, Bitmap> _blueClasses = new Dictionary<int, Bitmap>();
        public MainForm()
        {
            InitializeComponent();
            _redClasses.Add(1, new
Bitmap(@"D:\stud_repo\astu\TrafficSignRecognition\TrafficSignRecognizer\res\r0.jpg"));
            _redClasses.Add(2, new
Bitmap(@"D:\stud_repo\astu\TrafficSignRecognition\TrafficSignRecognizer\res\r1.jpg"));
            _redClasses.Add(3, new
Bitmap(@"D:\stud_repo\astu\TrafficSignRecognition\TrafficSignRecognizer\res\r2.png"));

            _blueClasses.Add(1, new
Bitmap(@"D:\stud_repo\astu\TrafficSignRecognition\TrafficSignRecognizer\res\b0.jpg"));
            _blueClasses.Add(2, new
Bitmap(@"D:\stud_repo\astu\TrafficSignRecognition\TrafficSignRecognizer\res\b1.jpg"));
            _blueClasses.Add(3, new
Bitmap(@"D:\stud_repo\astu\TrafficSignRecognition\TrafficSignRecognizer\res\b2.jpg"));

            //TextExtractor ext = new TextExtractor(_blues);
            //ext.PrepareData(@"D:\stud_repo\astu\train\blue", 3, new Size(40, 40));
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Task[] classifiers = new Task[2];
            flowLayoutPanel1.Controls.Clear();
            using (OpenFileDialog ofd = new OpenFileDialog()
                { Filter = "Image files (*.jpg, *.jpeg, *.jpe, *.jfif, *.png) | *.jpg;
*.jpeg; *.jpe; *.jfif; *.png"

```

```

        })
    {
        if(ofd.ShowDialog() == DialogResult.OK)
        {
            Emgu.CV.Image<Rgb, byte> source = new Emgu.CV.Image<Rgb,
byte>(ofd.FileName);
            List<PredictionResult> resultRed = new List<PredictionResult>()
            , resultBlue = new List<PredictionResult>();
            Stopwatch watch = Stopwatch.StartNew();
            classifiers[0] = Task.Factory.StartNew(() =>
            {
                resultBlue = RunBlue(source);

            });
            classifiers[1] = Task.Factory.StartNew(() =>
            {
                resultRed = RunRed(source);

            });
            Task.WaitAll(classifiers);
            watch.Stop();
            System.Diagnostics.Debug.WriteLine("parallel: " +
watch.ElapsedMilliseconds);
            var redRois = from pr in resultRed select pr.ROI;
            var bluRois = from pr in resultBlue select pr.ROI;
            var rois = redRois.Concat(bluRois);
            foreach (var roi in rois)
            {
                CvInvoke.Rectangle(source, roi, new MCvScalar(255, 255, 0));
            }
            pictureBox1.Image = source.Bitmap;

            foreach (var res in resultBlue)
            {
                PictureBox pb = new PictureBox() { Size = new Size(60, 60),
SizeMode = PictureBoxSizeMode.Zoom };
                pb.Image = _blueClasses[res.ClassId];
                flowLayoutPanel1.Controls.Add(pb);
            }

            foreach (var res in resultRed)
            {
                PictureBox pb = new PictureBox() { Size = new Size(60, 60),
SizeMode = PictureBoxSizeMode.Zoom };
                pb.Image = _redClasses[res.ClassId];
                flowLayoutPanel1.Controls.Add(pb);
            }
            //List<PredictionResult> resultRed = RunBlue(ofd.FileName);
        }
    }
}

private List<PredictionResult> RunBlue(Emgu.CV.Image<Rgb, byte> source)
{
    List<PredictionResult> results = new List<PredictionResult>();
    // = new Emgu.CV.Image<Rgb, byte>(path);
    Stopwatch watch = Stopwatch.StartNew();
    var gray = source.Convert<Gray, byte>();
    var blueTarget = _blues.FilterColorChannel(source);
    //blueTarget.Save("colored.jpg");
    Emgu.CV.Image<Gray, Byte> accumulator = new Emgu.CV.Image<Gray,
byte>(source.Width, source.Height, new Gray(0));
    var ROIs = _blues.FindROIs(blueTarget, accumulator);
    //_blues.Highlight(ROIs, source);
}

```

```

        //pictureBox1.Image = source.Bitmap;
        foreach (var roi in ROIs)
        {
            gray.ROI = roi;
            accumulator.ROI = roi;
            var img = _blues.FindObjectOfInterest(gray, blueTarget, new Size(40,
40));

            double[] answer;
            _blueNet.NetworkOut(ImageToSample.ConvertImageToSample(img), out answer);
            results.Add(new PredictionResult(answer, roi));
            //MessageBox.Show(string.Format("ans: {0}\n{1}\n{2}", answer[0],
answer[1], answer[2]));
        }
        watch.Stop();
        System.Diagnostics.Debug.WriteLine("blue " + watch.ElapsedMilliseconds);
        return results;
    }
    private List<PredictionResult> RunRed(Emgu.CV.Image<Rgb, byte> source)
    {
        List<PredictionResult> results = new List<PredictionResult>();
        //Emgu.CV.Image<Rgb, byte> source = new Emgu.CV.Image<Rgb, byte>(path);
        Stopwatch watch = Stopwatch.StartNew();
        var gray = source.Convert<Gray, byte>();
        var redTarget = _reds.FilterColorChannel(source);
        Emgu.CV.Image<Gray, Byte> accumulator = new Emgu.CV.Image<Gray,
byte>(source.Width, source.Height, new Gray(0));
        var ROIs = _reds.FindROIs(redTarget, accumulator);
        //_reds.Highlight(ROIs, source);
        //pictureBox1.Image = source.Bitmap;
        foreach (var roi in ROIs)
        {
            gray.ROI = roi;
            redTarget.ROI = roi;
            var img = _reds.FindObjectOfInterest(gray, redTarget, new Size(40, 40));
            double[] answer;

            _redNet.NetworkOut(ImageToSample.ConvertImageToSample(img), out answer);

            results.Add(new PredictionResult(answer, roi));
            //MessageBox.Show(string.Format("ans: {0}\n{1}\n{2}", answer[0],
answer[1], answer[2]));
        }
        watch.Stop();
        System.Diagnostics.Debug.WriteLine("red " + watch.ElapsedMilliseconds);
        return results;
    }
}
}
}

```