Министерство образования Республики Беларусь
Учреждение образования «Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Компьютерные системы и сети (КСиС)

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к курсовому проекту

на тему

Искусственная нейронная сеть

БГУИР КП 1-40 01 01 116 ПЗ

Студент: гр. 351001 Мельников А.В.

Руководитель: асс. Третьяков Ф.И.

Минск 2015

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет компьютерных систем и сетей

УТВЕРЖДАЮ
Заведующий кафедрой ПОИТ

_____
(подпись)
_____2015 г.

ЗАДАНИЕ
по курсовому проектированию

Студенту Мельникову Андрею Васильевичу

1. Тема работы Искусственная нейронная сеть
2. Срок сдачи студентом законченной работы 05.06.2015-09.06.2015
3. Исходные данные к работе Среда разработки .NET C#.
4. Содержание расчётно-пояснительной записки (перечень вопросов, которые подлежат разработке)
   Введение.
   1. Анализ литературных источников.
   2. Постановка задачи
   3.Разработка программного средства.
   4. Тестирование программного средства.
   5. Пример работы.
   Заключение. Приложения.
5. Консультант по курсовой работе
   Третьяков Ф.И.
6. Дата выдачи задания 20.02.2015
7. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и процентом от общего объёма работы):
   Введение к 28.02.2015 – 15 % готовности работы;
   разделы 2 к 15.03.2015 – 30 % готовности работы;
   раздел 3 к 15.04.2015 – 60 % готовности работы;
   раздел 4, к 10.05.2015 – 90 % готовности работы;

оформление пояснительной записки и графического материала к 01.06.2015 – 100 % готовности работы.
Защита курсового проекта с 05.06.2015 по 09.06.2015 г.

РУКОВОДИТЕЛЬ Третьяков Ф.И.
(подпись)

Задание принял к исполнению Мельников А.В. 20.02.2015г.
(дата и подпись студента)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Теория распознавания образов является одним из важнейших разделов кибернетики как в теоретическом, так и в прикладном плане. Она является полезнейшим инструментом в научных исследованиях и в ряде областей практической деятельности. Владение методами распознавания образов необходимо каждому специалисту по прикладной информатике, занимающемуся обработкой результатов экспериментов, что является востребованным в последние годы.

Тема распознавания текста попадает под раздел распознавания образов. Следует заметить, что под распознаванием текста обычно понимают три главных метода:

1) Сравнение с заранее подготовленным шаблоном;

2) Распознавание с использованием критериев, распознаваемого объекта;

3) Распознавание при помощи самообучающихся алгоритмов, в том числе при помощи нейронных сетей.

1 АНАЛИЗ ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

Задача распознавания (точнее, классификации) объекта ставится следующим образом. Имеется некоторый способ кодирования объектов принадлежащих заранее известному конечному множеству классов M=M1 ,...,Mn, и некоторое конечное множество объектов (обучающее множество), про каждый из которых известно, какому классу он принадлежит. Нужно построить алгоритм, который по любому входному объекту, не обязательно принадлежащему обучающему множеству, решает, какому классу этот объект принадлежит, и делает это достаточно хорошо. Качество распознавания оценивается как вероятность ошибки классификации на другом конечном множестве объектов с заранее известными ответами (тестовом множестве).

Типичная система распознавания состоит из трех частей: извлечение признаков, собственно распознавание и принятие решения[1].

Механизм распознавания символов представляет собой комбинацию ряда элементарных распознавателей, называемых классификаторами. Рассмотрим и проанализируем наиболее часто используемые классификаторы.

Растровый классификатор. Самое простое пространство признаков, которое можно придумать для распознавания символов – это само изображение. Все изображения одной графемы из обучающей базы приводятся к одному размеру, а затем усредняются. В итоге получается серое изображение, где цвет каждой точки отвечает вероятности того, что это точка должна быть белой или черной для данного символа. Оценкой принадлежности изображения заданному классу является обычное расстояние между черно-белым изображением и серым эталоном.

Плюсом растрового классификатора является его простота и скорость. А минусом – невысокая точность, в основном из-за того, что при приведении к одному размеру мы теряем всю информацию о геометрии символов.

Признаковый классификатор. Изображению ставится в соответствие n-мерный вектор признаков, который сравнивается с набором эталонных векторов той же размерности. Тип и количество признаков в немалой степени определяют качество распознавания. Извлечение признаков – формирование вектора (вычисление его координат в n-мерном пространстве) производится во время анализа предварительно подготовленного изображения. Эталон для каждого класса получают путём аналогичной обработки символов обучающей выборки.

Признаковый классификатор обладает хорошей обобщающей способностью и устойчивостью к изменениям формы символов. Однако на этапе извлечения признаков происходит необратимая потеря части информации о символе. Извлечение признаков ведётся независимо, поэтому информация о взаимном расположении элементов символа утрачивается.

Структурный классификатор. Данный классификатор подходит для распознавания рукописного текста и, соответственно, обладает высокой точностью распознавания.

Проблема рукописного текста в том, что его вариативность очень большая, даже если человек старается писать печатными буквами. Поэтому для рукописного текста все стандартные классификаторы не могут дать достаточного качества.

Идея данного классификатора заключается в описании структуры каждого символа через базовые элементы – линия, дуга, кольцо и т.п. Описание приходится делать вручную, это большая проблема, но пока еще никто не смог придумать автоматического выделения топологии с приемлемым качеством.

Дифференциальный классификатор. Причина появления этого классификатора в следующем – есть пары символов, которые очень похожи друг на друга, за исключением каких-то небольших очень специфических особенностей. Причем для каждой пары символов эти особенности отличаются. Если знания об этих особенностях пытаться встраивать в общий классификатор, то общий классификатор будет перегружен признаками, каждый из которых не будет приносить пользы подавляющем большинстве случаев.

Поэтому для похожих пар символов ищутся специальные признаки, по которым строится классификатор на основе этих признаков только для этих двух классов. Если в вариантах распознавания есть два похожих варианта с близкими оценками, такой классификатор может точно сказать, какой из этих двух вариантов предпочтительней.

Плюсом дифференциального классификатора Добавление каждого нового дифференциального классификатора очень заметно улучшает распознавание. Высокая трудоёмкость построения эталонов для классификатора и низкое быстродействие[2].

Каждый классификатор заслуживает отдельного внимания, однако в данной курсовой работе для распознавания и классификации образов будет использована нейронная сеть Кохонена, которая базируется на признаковом классификаторе. Преимуществом сети Кохонена является её способность к обобщению. В процессе обучения входные векторы ассоциируются с соответствующими выходными сигналам. Качественно обученная сеть гарантирует распределение входных векторов по соответствующим классам. Обобщающая способность сети позволяет получать правильный выход даже при приложении входного вектора, который является неполным или слегка неверным. Это позволяет использовать данную сеть, как для распознавания образов, так и для их восстановления[1].

2 ПОСТАНОВКА ЗАДАЧИ

Согласно техническому заданию требуется разработать программное обеспечение, способное распознавать текст при использовании изображений высокого либо среднего качества, со слабым шумом либо без него. Приложение должно распознавать буквы английского алфавита верхнего регистра. Изображение подается для распознавания непосредственно из самого приложения.Результат работы - текстовая информация содержавшаяся на входном изображении.

3 РАЗРАБОТКА ПРИЛОЖЕНИЯ

3.1 Алгоритм распознавания текста

Алгоритм распознавания текста представлен на рисунке 1.

Рис. 1: Алгоритм распознавания текста

3.2 Медианный фильтр

Множество изображений полученных при съемке подвергаются импульсным помехам. При их воздействии на изображение, наблюдаются белые или черные точки, хаотически разбросанные по кадру.

Данный фильтр применятся для минимизации шума и смазывания острых краев букв.

При медианной фильтрации используется двумерное окно обычно имеющее центральную симметрию, при этом его центр располагается в текущей точке фильтрации. На рисунке 2 показаны два примера наиболее часто применяемых вариантов окон в виде креста и в виде квадрата[3].

Значения отсчётов внутри окна фильтра сортируются в порядке возрастания (убывания); и значение, находящееся в середине упорядоченного списка, поступает на выход фильтра. Окно перемещается вдоль фильтруемого сигнала и вычисления повторяются[4].

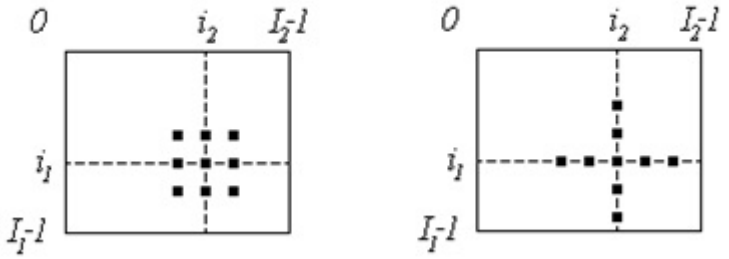

Рис. 2: Примеры окон при медианной фильтрации

3.3 Обработка исключительных ситуаций

Бинаризация изображений, т.е. перевод полноцветного или в градациях серого изображения в монохромное, где присутствуют только два типа пикселей (темные и светлые) имеет большое значение при распознавании образов. Особенно это относится к бинарным объектам, таким, как штриховые коды, текст, чертежи и т.п. Существуют различные подходы к бинаризации, которые условно можно разделить на 2 группы:пороговые и адаптивные

В рамках моей курсовой работы будет рассмотрен только пороговый метод. Пороговые методы бинаризации работают со всем изображением, находя какую-то характеристику (порог), позволяющую разделить все изображение на чёрное и белое. То есть все пиксели, значения которых меньше порога, становятся черными, а пиксели, значение которых выше порога, становятся белыми [5]. Данное действие реализуется функцией Monochrome.

Существует множество методов вычисления порога бинаризации, однако, для упрощения алгоритма, порогу было присвоено константное значение.

3.4 Сегментация

Для процесса сегментации принята следующая эвристика. Предполагается, что предложения текста расположены горизонтально и не создают пересечений друг с другом. Поэтому алгоритм нахождения поисковых узоров можно разделить на следующие этапы:

1) GetMatrix – Преобразование изображения в двумерный массив единиц (пиксели черного цвета) и нулей (пиксели белого цвета);

2) FindSymbols – Сканирование элементов матрицы, с целью поиска единичного значения;

3) AbsorbSymbol – Если единица была найдена, то, используя алгоритм поиска в глубину, вырезаем множество смежных единиц, заменяя их значения нулями, чтобы исключить возможность повторного считывания.

Во время данного процесса вычисляются и сохраняются такие значения, как верхняя, нижняя, правая, левая границы символа;

4) CutSymbol – После полного исследования матрицы, на выходе имеем массив элементов хранящих границы символов, которые были расположены на изображении. Используя полученные значения, вырезаем символы из преобразованного фильтрами изображения;

5) LeadToThePattern – Приведение размеров изображений символов к размерам обучающих шаблонов.

3.5 Нейронные сети Кохонена

3.5.1 Структура сети

Для применения нейронных сетей Кохонена в задачах классификации требуется некоторая формализация. Каждый объект, который требуется классифицировать, представляется в виде некоторого вектора, подающегося на вход нейронной сети. В программе этот вектор представлен одномерным массивов состоящим из нулей и единиц. Составленние массива происходит при построчном сканировании изображения символа, приведенного к нужному размеру. Нулевые и единичные значение присваиваются в соответствии с цветом сканируемого пикселя(черный - единица, белый - ноль) [6].

Количество нейронов во входном слое определяется количеством компонентов этого входного вектора. Количество же выходов определяется количеством классов, т.е. если необходимо определить к какой букве алфавита относится входной вектор, то количество нейронов в выходном слое будет равно размеру алфавита выбранного языка. Таким образом, каждый нейрон в выходном слое «отвечает» за свою букву. Значения, которые принимают нейроны в выходном слое, отображают насколько вектор классифицируемого символа на входе близок, по мнению нейронной сети Кохонена, к тому или иному классу. Чем больше «уверенность», что объект принадлежит к тому или иному классу, тем больше значение принимает нейрон соответствующего класса.

Каждый нейрон выходного слоя может принимать значение либо ноль, либо единица. При этом для одного входного вектора единице может быть равен один и только один нейрон выходного слоя, т.е. один объект не может относиться сразу к двум классам. Рассмотрим следующую нейронную сеть Кохонена с m входами и n выходами, т.е. нейронную сеть для классификации по n классам:

На рисунке 3 нейрон Кохонена K1 имеет веса w11, w21, ..., wm1, составляющие весовой вектор W1. Они соединяются через входной слой с входными сигналами x1, x2, ..., xm, составляющими входной вектор X. Подобно нейронам большинства сетей выход NET каждого нейрона

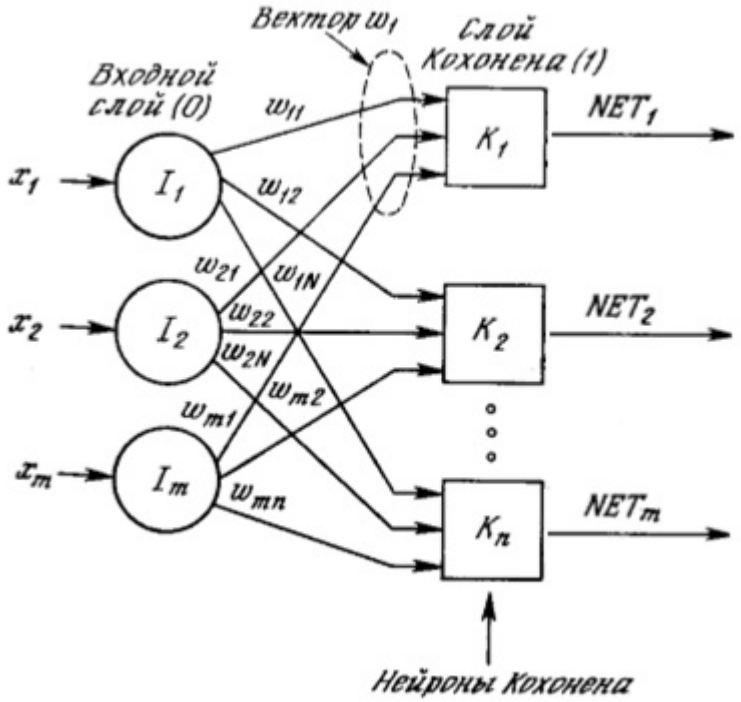

Рис. 3: Нейронная сеть Кохонена

Кохонена является просто суммой взвешенных входов.

Нейрон Кохонена с максимальным значением NET является «победителем». Его выход равен единице, у остальных он равен нулю[1].В программе выходные значения нейронов вычисляется при передаче классифицируемого вектора в функцию Parse. Нейрон с наибольшим значением NET определяется в функции FindNeuronWithMaxExcitation. Чтобы сеть можно было использовать повторно, по завершению классификации, импульс каждого нейрона обнуляется при вызове метода SetZeroExcitations.

3.5.2 Обучения слоя Кохонена

Чтобы коэффициенты входов были правильно настроены, необходимо сначала обучить сеть. Как правило, обучающее множество включает много сходных между собой входных векторов, и сеть должна быть обучена активировать один и тот же нейрон Кохонена для каждого из них. В этом случае веса этого нейрона должны получаться усреднением входных векторов, которые должны его активировать. Этим занимается отдельный модуль обучения. Данный модуль берет очередное изображение из обучающей выборки и указывает сети к какому классу оно относиться. Сеть анализирует все позиции черных пикселей и выравнивает коэффициенты нейрона отвечающего за указанный класс, минимизируя ошибку совпадения.

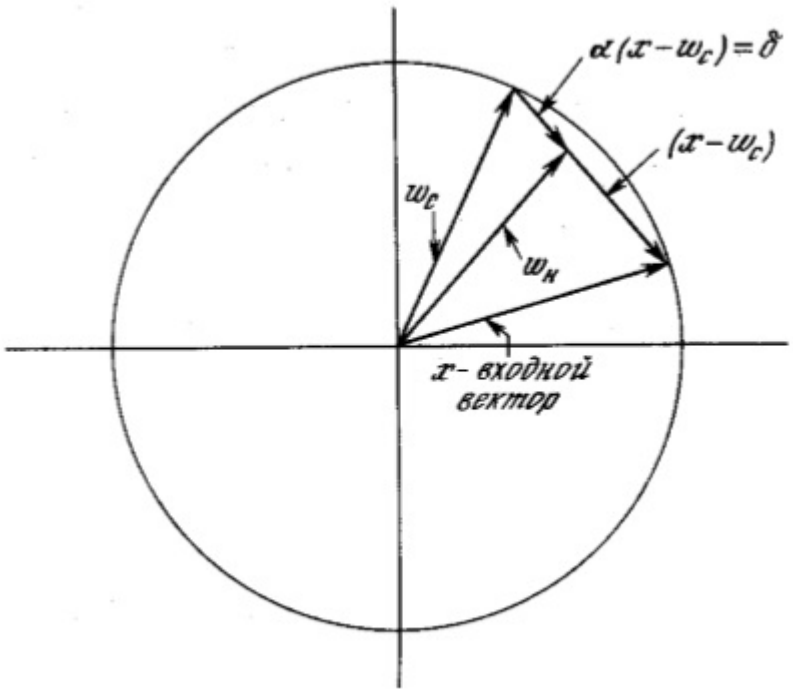

Рис. 4: Вращение весового вектора в процессе обучения (Wн – вектор новых весовых коэффициентов, Wc – вектор старых весовых коэффициентов)

Переменная a является коэффициентом скорости обучения, который вначале равен 0,7 и постепенно уменьшаться в процессе обучения, при каждом вызове метода Study. Постепенное уменьшение величины a уменьшает воздействие каждого обучающего шага, так что окончательное значение будет средней величиной от входных векторов, на которых происходит обучение.

3.5.3 Предварительная обработка входных векторов

Весьма желательно нормализовать входные векторы перед тем, как предъявлять их сети. Это выполняется с помощью деления каждой компоненты входного вектора на длину вектора. Эта длина находится извлечением квадратного корня из суммы квадратов компонент вектора.

Это превращает входной вектор в единичный вектор с тем же самым направлением, т. е. в вектор единичной длины в n-мерном пространстве. Данное действие реализуется функцией NormaizeVector.

3.5.4 Выбор начальных значений весовых векторов

Одно из решений, известное под названием метода выпуклой комбинации (convex combination method), состоит в том, что все веса приравниваются одной и той же величине.

Вначале a очень мало, вследствие чего все входные векторы имеют длину, близкую к 1/n , и почти совпадают с векторами весов. В процессе

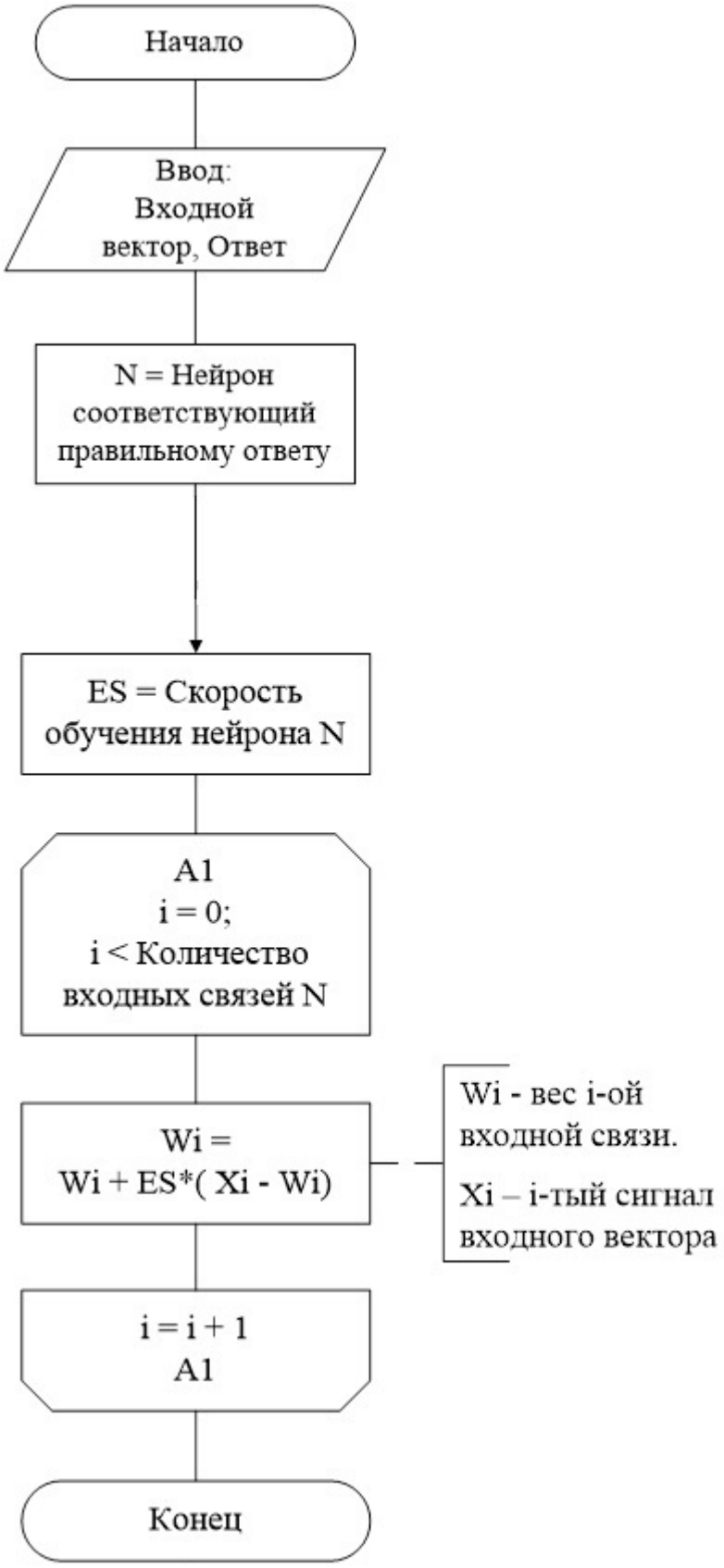

Рис. 5: Алгоритм выравнивания весовых коофициентов

обучения сети a постепенно возрастает, приближаясь к единице. Это позволяет разделять входные векторы и окончательно приписывает им их истинные значения[1].

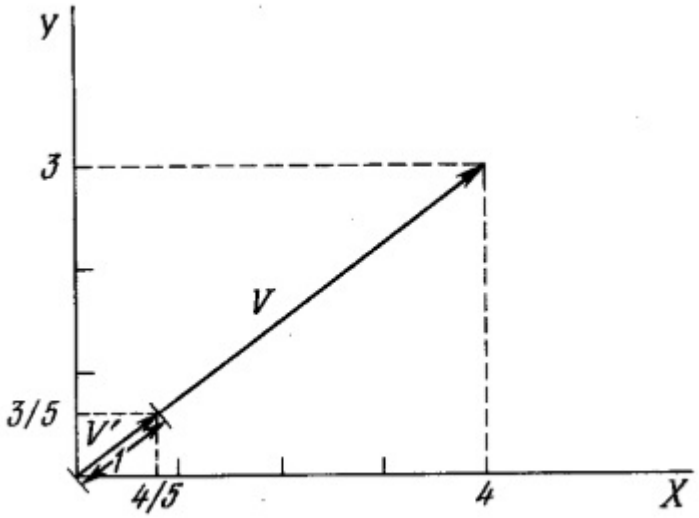

Рис. 6: Единичный входной вектор

Уменьшение резкости возрастания a, при не достаточном числе обучающих выборок, может быть достигнуто путем увеличения обучающих процессов по выборкам. В программе, при каждом вызове метода Study, значение a увеличивается на такое значение сигма, что к концу процесса обучения сети, a принимает значение близкое к единице. Нормализация обучающих векторов реализована в функции NormaizeStudyVector.

3.6 Организация текстовой информации

3.6.1 Пространственное расположение

При помощи нейронной сети Кохонена входные символы были распределены по классам, однако в разделенном состоянии они не несут никакой информационной нагрузки. Поэтому необходимо составить текст в соответствии с расположением символов на обрабатываемом изображении.

В процессе сегментации для каждого вырезанного символа были сохранены его границы. Используя эти значения можно найти координаты центра символа на исходном изображении следующим образом:

x = (Right + Left)/2; y = (Bottom + Top)/2;

Символы в списке сортируются по их пространственному расположению слева направо, сверху вниз[7].

3.6.2 Переходы к новой строке

Так как предполагается, что предложения текста расположены горизонтально и не создают пересечений друг с другом. То, для того, чтобы определить переход к новой строке, можно использовать следующий алгоритм:

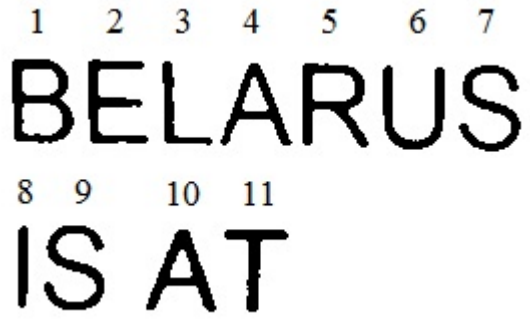

Рис. 7: Пример нумерации символов

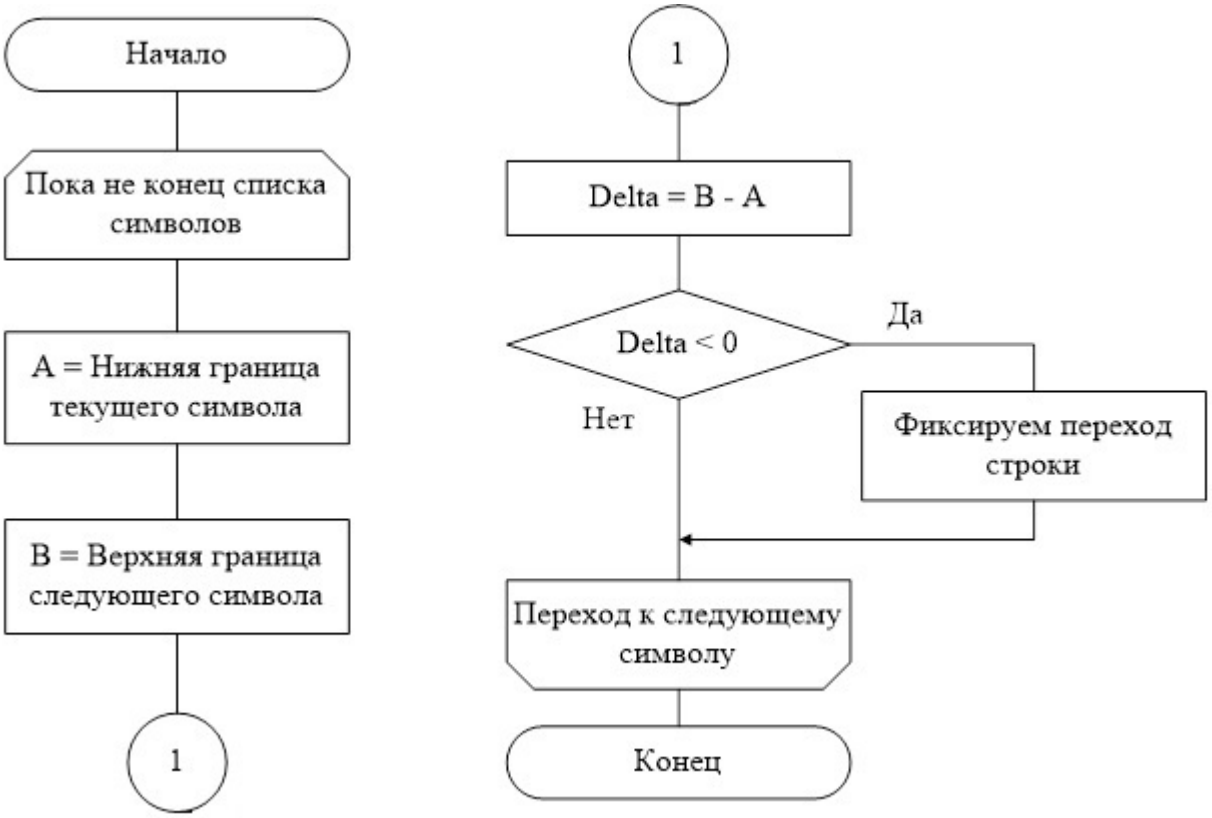

Рис. 8: Алгоритм определения переходов к новой строке

3.6.3 Поиск пробелов

Поиск пробелов в строке текста на удивление сложная задача. Хотя у человека при чтении текста даже на неизвестном языке практически никогда не возникает проблемы найти пробелы в тексте, при автоматической классификации все время возникают разные проблемы. Базовая идея заключается в следующем – ищутся на изображении строки вертикальные белые просветы, а дальше классифицируются по ширине: широкие просветы – это пробелы между словами, узкие – между символами[2].

Идея замечательная, но в реальной жизни ширина пробелов может

Рис. 9: Пример вертикальных просветов

быть очень неоднозначным показателем.

4 ТЕСТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

Использовалось модульное тестирование с помощью библиотеки, встроенной в Visual Studio 2013 - Microsoft.VisualStudio.TestTools.UnitTesting.

Был разработан класс модульных тестов MatTest, который покрыл класс Mat, выполняющий важные вычислительные операции в программном средстве, на 100%. Пройденные тесты представлены на рисунке 10.

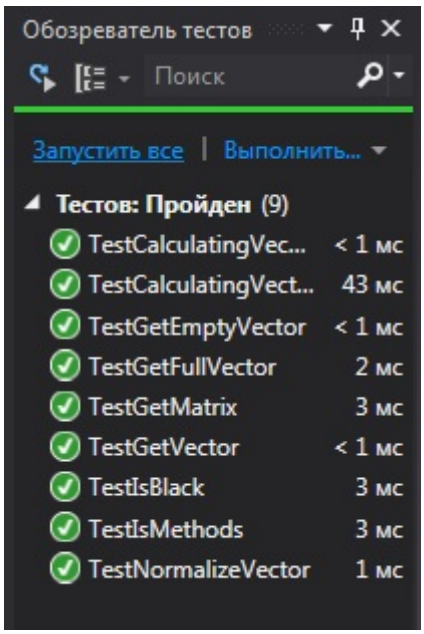

Рис. 10: Тесты

Класс MatTest содержит девять методов:

- TestCalculatingVectorLengthInt() и TestCalculatingVectorLengthDouble() - проверяют, совпадает ли ожидаемое значение длины вектора с фактическим;

- TestIsBlack() - проверяет, отличает ли приложение пиксели черного цвета от остальных цветов;

- TestNormalizeVector() - проверяет,правильность вычисления нормальной формы вектора и совпадает ли ожидаемое значение длины нормализованного вектора с фактическим;

- TestGetMatrix() - сравнивает получаемую из изображения матрицу с фактической;

- TestGetFullVector() и TestGetEmptyVector() - проверяют правильность создания единичных и нулевых векторов соответственно;

- TestGetVector() - проверяет правильность создания вектора графического изображения.

- TestIsMethods() - проверяет корректность работы методов обработки единичных и нулевых векторов.

5 ПРИМЕР РАБОТЫ

1) Загрузка исходного изображения;

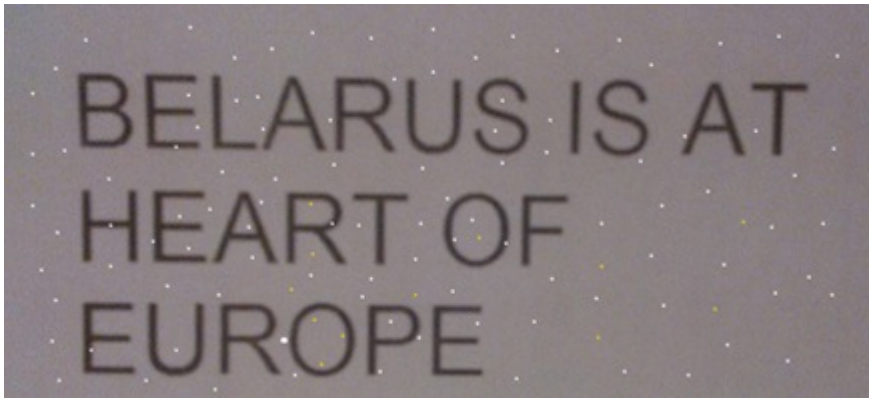

Рис. 11: Исходное изображение

2) Медианная фильтрация изображения;

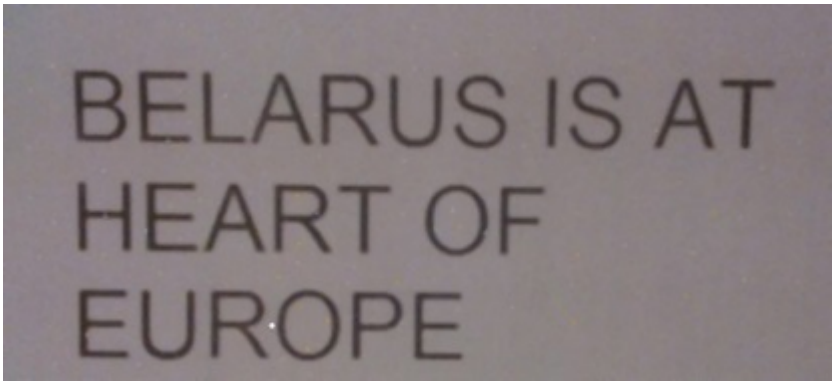

Рис. 12: Результат применения медианного фильтра

3) Бинаризация изображения;

Рис. 13: Результат бинаризации изображения

4) Сегментация;
5) Распознавание и организация текстовой информации;

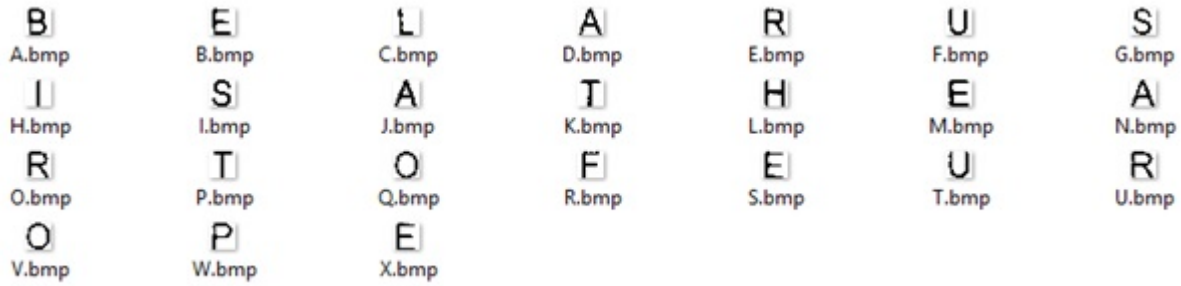

Рис. 14: Результат сегментации

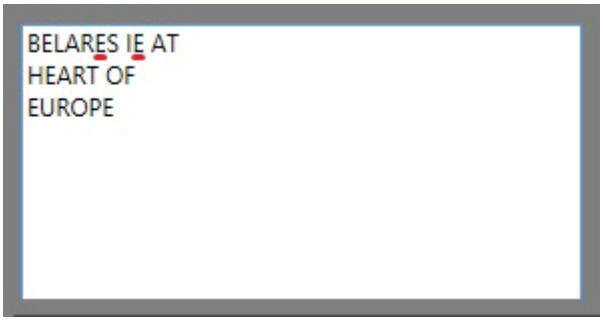

Рис. 15: Результат распознавания

В результате распознавания были допущены две ошибки. Данное отклонение обусловлено не достаточным количеством и качеством обучающих выборок созданных для обучения нейронной сети. А так же на основе этого можно утверждать о том, что качественное распознавание не возможно при использовании единственного классификатора – нейронной сети Кохонена.

ЗАКЛЮЧЕНИЕ

В результате работы была создана программа, позволяющая распознавать текстовую информацию.

Созданная программа позволяет пользователю, распознавать текст, при использовании изображений высокого либо среднего качества, со слабым шумом либо без него.

В качестве направления для дальнейшей работы с программой распознавания можно выделить:

1)Выбор более сложных иерархических моделей нейросетей организованных аналогично зрительной коре (Когнитрон, Неокогнитрон).

2)Комбинированное использование стандартных классификаторов

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

[1] Уоссермен, Ф. Нейрокомпьютерная техника: Теория и практика: пер. с англ. Ю. А. Зуев / В. А. Точенов – М. : Феникс, 1992. – 184 с.

[2] Распознавание текста в ABBYY FineReader [Электронный ресурс]. – Электронные данные. – Режим доступа: http://habrahabr.ru/company/abbyy/blog/225215.

[3] Цифровая обработка изображений [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.sibsauktf.ru/courses/fulleren/g3.htm.

[4] Научная библиотека избранных естественно-научных изданий [Электронный ресурс]. – Электронные данные. – Режим доступа: http://sernam.ru/book$_k ir.php?id = 25$.

[5] Бинаризация изображений a [Электронный ресурс]. – Электронные данные. – Режим доступа: http://recog.ru/blog/applied/15.html.

[6] Нейронные сети [Электронный ресурс]. – Электронные данные. – Режим доступа: http://www.aiportal.ru/articles/neural-networks/1/.

[7] Методы распознавания текста [Электронный ресурс]. – Электронные данные. – Режим доступа: http://habrahabr.ru/post/220077.

ПРИЛОЖЕНИЕ А
(обязательное)
Исходный код программы

```
class Program
{
        static void Main(string[] args)
        {
            var bmp = new Bitmap(
Bitmap.FromFile(@"C:\Users\Andrew\Desktop\belphoto.jpg")
     );

            var kn = new KohonenNetwork();
            var td = new TextDetector();
            var recognizer = new Recognizer(kn, td);
            var Message = recognizer.Parse(bmp);

            Console.WriteLine(Message);
        }
}

class Neuron
{
        public Link[] IncomingLinks;
        public double Power { get; set; }
        private double _es = 0.79;
        private double _alpha = 0.001;
        public double Alpha { get { _alpha *= 1.28; return _alpha; } }
        public double EducationSpeed { get { _es *= 0.9; return _es; }

        public Neuron(int LinksCount)
        {
            IncomingLinks = new Link[LinksCount];
            Power = 0;
        }
}


class Link
{
        public Neuron Neuron;
```

```
        public double Weight;

        public Link(Neuron neuron)
        {
            Neuron = neuron;
            Weight = 0;
        }
}


class Input
{
        public Link[] OutgoingLinks;

        public Input(int LinksCount)
        {
            OutgoingLinks = new Link[LinksCount];
        }
}


class KohonenNetwork
{
        private readonly Input[] _inputs;
        private readonly Neuron[] _neurons;
        public static readonly int _alphabetLen = 26;
        private readonly int _pixelsNumber;

        public KohonenNetwork()
        {
            _pixelsNumber = 16 * 16;
            _inputs = new Input[_pixelsNumber];
            _neurons = new Neuron[_alphabetLen];
            FieldInputs();
            FieldNeurons();
            CreateNetwork();
            SetZeroExcitations();
        }

        private void FieldInputs()
        {
            for (var i = 0; i < _pixelsNumber; i++)
                _inputs[i] = new Input(_alphabetLen);
```

```
}

private void FieldNeurons()
{
    for (var i = 0; i < _alphabetLen; i++)
        _neurons[i] = new Neuron(_pixelsNumber);
}

private void CreateNetwork()
{
    for (var j = 0; j < _pixelsNumber; j++)
    {
        for (var i = 0; i < _alphabetLen; i++)
        {
            var link = new Link(_neurons[i]);
            link.Weight = 1 / Math.Sqrt(_pixelsNumber);
            _inputs[j].OutgoingLinks[i] = link;
            _neurons[i].IncomingLinks[j] = link;
        }
    }
}

public int  Parse(int[] input)
{
    var normalizedVector = Mat.NormaizeVector(input);
    for (var i = 0; i < _inputs.Length; i++)
    {
        var inputNeuron = _inputs[i];
        foreach (var outgoingLink in inputNeuron.OutgoingLinks
        {
            outgoingLink.Neuron.Power += outgoingLink.Weight *
        }
    }
    var maxIndex = FindNeuronWithMaxExcitation();
    SetZeroExcitations();

    return maxIndex;
}

private int FindNeuronWithMaxExcitation()
{
    var maxIndex = 0;
    for (var i = 1; i < _neurons.Length; i++)
```

```
            {
                if (_neurons[i].Power > _neurons[maxIndex].Power)
                    maxIndex = i;
            }
            return maxIndex;
        }

        private void SetZeroExcitations()
        {
            foreach (var outputNeuron in _neurons)
            {
                outputNeuron.Power = 0;
            }
        }

        public void Study(int[] input, int correctAnswer)
        {
            var neuron = _neurons[correctAnswer];
            var educationSpeed = neuron.EducationSpeed;
            var alpha = neuron.Alpha;
            var normalizedVector = NormaizeStudyVector(input, alpha);

            for (var i = 0; i < neuron.IncomingLinks.Length; i++)
            {
                var incomingLink = neuron.IncomingLinks[i];
                incomingLink.Weight = incomingLink.Weight + educationS

            }
        }

        private double[] NormaizeStudyVector(int[] input, double alpha
        {
            double[] normalizedVector = new double[input.Length];
            int n = input.Length;

            for (var i = 0; i < input.Length; i++)
            {
                normalizedVector[i] = alpha * input[i] + (1 - alpha) /
            }
            return normalizedVector;
        }
    }
```

```
class Filter
{
        public static void Median3x3(ref Bitmap image)
        {
            int[] dx = {0, 1, 1, 0, -1, -1, 1, -1, 0};
            int[] dy = {1, 0, 1, -1, 0, 1, -1, -1, 0};
            var arrR = new int[9];
            var arrG = new int[9];
            var arrB = new int[9];
            var outImage = new Bitmap(image);

            for (int i = 1; i < image.Width - 1; i++)
                for (int j = 1; j < image.Height - 1; j++)
                {
                    for (int i1 = 0; i1 < 9; i1++)
                    {
                        var point = image.GetPixel(i + dx[i1], j + dy[

                        arrR[i1] = point.R;
                        arrG[i1] = point.G;
                        arrB[i1] = point.B;
                    }
                    Array.Sort(arrR);
                    Array.Sort(arrG);
                    Array.Sort(arrB);

                    outImage.SetPixel(i, j, Color.FromArgb(arrR[4], ar
                }

            image = outImage;
        }

        public static void Monochrome(ref Bitmap image, int level)
        {
            for (int j = 0; j < image.Height; j++)
            {
                for (int i = 0; i < image.Width; i++)
                {
                    var color = image.GetPixel(i, j);
                    int sr = (color.R + color.G + color.B) / 3;
                    image.SetPixel(i, j, (sr < level ? Color.Black : C
                }
```

```
            }
        }
}


class Symbol
{
        private int _x;
        private int _y;
        private int _top = int.MaxValue;
        private int _buttom = 0;
        private int _left = int.MaxValue;
        private int _right = 0;
        public bool isEmpty = true;

        public int CenterX { get { return _x; } }
        public int CenterY { get { return _y; } }
        public int Width { get { return _right - _left; } }
        public int Height { get { return _buttom - _top; } }
        public int Top { get { return _top; } }
        public int Buttom { get { return _buttom; } }
        public int Left { get { return _left; } }
        public int Rigt { get { return _right; } }

        public Bitmap Pattern { get; set; }

        public Symbol(){}

        public int[] GetVector()
        {
            return Mat.GetVector(Pattern);
        }

        private Bitmap LeadToThePattern(Bitmap image)
        {
            if (image.Height < 5 && image.Width < 5)
                return new Bitmap(16, 16);

            int newWidth = 16;
            int newHeight = 16;

            if (image.Height > image.Width)
                newWidth = (int)(image.Width / ((float)image.Height /
```

```
        if (image.Height < image.Width)
            newHeight = (int)(image.Height / ((float)image.Width /

        Bitmap newImage = new Bitmap(image, new Size(newWidth, new

        Bitmap tmpFile = new Bitmap(16, 16);
        Graphics gr = Graphics.FromImage(tmpFile);
        gr.Clear(Color.White);
        gr.DrawImage(newImage, new Point((16 - newWidth) / 2, 0));

        return tmpFile;
    }

    public void CutSymbol(ref Bitmap image)
    {
        RectangleF cloneRect = new RectangleF(_left, _top, _right
        System.Drawing.Imaging.PixelFormat format =
            image.PixelFormat;
        Bitmap cloneBitmap = image.Clone(cloneRect, format);

        var img = LeadToThePattern(cloneBitmap);
        Filter.Monochrome(ref img, 200);
        Pattern = img;
    }

    public void AddBlackPoint(int x, int y)
    {
        if (y > _buttom)
            _buttom = y;
        if (y < _top)
            _top = y;
        if (x < _left)
            _left = x;
        if (x > _right)
            _right = x;
        isEmpty = false;
        CountCenter();
    }

    private void CountCenter()
    {
        _x = (_left + _right) / 2;
```

```
            _y = (_top + _buttom) / 2;
        }
}


class TextDetector
{
        private Bitmap _image;
        private int[][] _matrix;
        private List<Symbol> _symbols;

        private int[] dx = {0, 1, 1, 0, -1, -1, 1, -1};
        private int[] dy = {1, 0, 1, -1, 0, 1, -1, -1};

        public List<Symbol> Symbols { get { return _symbols; } }

        public TextDetector() { }

        public void Detect(Bitmap image)
        {
            _image = image;
            _symbols = new List<Symbol>();
            _matrix = Mat.GetMatrix(_image);
            FindSymbols();
            SortSymbols();
        }

        public List<int[]> GetVectors()
        {
            List<int[]> vectors = new List<int[]>();
            for (int i = 0; i < _symbols.Count; i++)
                vectors.Add(_symbols[i].GetVector());

            CompleteTextOrganization(vectors);

            return vectors;
        }

        public void CompleteTextOrganization(List<int[]> vectors)
        {
            var enterVector = Mat.GetEnterVector();
            var spaceVector = Mat.GetSpaceVector();
            var deltaI = 0;
```

```
        int[] spaces = FindSpaces();
        int[] enters = FindLineCompletions();

        for (var i = 1; i < _symbols.Count; i++)
        {
            if (spaces[i] == 1)
            {
                vectors.Insert(i + deltaI, spaceVector);
                deltaI++;
            }

            if (enters[i] == 1)
            {
                vectors.Insert(i + deltaI, enterVector);
                deltaI++;
            }
        }
    }

    private int[] FindLineCompletions()
    {
        var result = new int[_symbols.Count];
        var deltaY = 0;

        for (int i = 1; i < _symbols.Count; i++)
        {
            deltaY = (_symbols[i - 1].Buttom - _symbols[i].Top);
            if (deltaY < 0)
                result[i] = 1;
            else
                result[i] = 0;
        }

        return result;
    }

    private int[] FindSpaces()
    {
        var result = new int[_symbols.Count];
        var spaces = new List<int>(){-1};
        var deltaY = 0;
        var deltaX = 0;
```

```
        for (var i = 1; i < _symbols.Count; i++)
        {
            deltaY = (_symbols[i - 1].Buttom - _symbols[i].Top);
            deltaX = _symbols[i].Left - _symbols[i - 1].Rigt;
            if (deltaY > 0)
                spaces.Add(deltaX);
            else
                spaces.Add(-1);
        }
        double avSpaceWidth = spaces.Max() / 2;

        for (var i = 1; i < _symbols.Count; i++)
        {
            deltaY = (_symbols[i - 1].Buttom - _symbols[i].Top);
            if (spaces[i] > avSpaceWidth && deltaY > 0)
                result[i] = 1;
            else
                result[i] = 0;
        }

        return result;
    }

    private void SortSymbols()
    {
        _symbols.Sort(delegate(Symbol x, Symbol y)
        {
            if (x.CenterY - y.CenterY > 10) return 1;
            if (x.CenterY - y.CenterY < -10) return -1;
            if (x.CenterX >= y.CenterX) return 1;
            if (x.CenterX < y.CenterX) return -1;
                return 0;
        }
        );
    }

    private void FindSymbols()
    {
        Symbol newSymbol;
        for (int i = 0; i < _image.Height; i++)
            for (int j = 0; j < _image.Width; j++)
                if (_matrix[i][j] == 1)
                {
```

```
                    newSymbol  = new Symbol();
                    AbsorbSymbol(j, i, newSymbol);
                    if (!newSymbol.isEmpty)
                    _symbols.Add(newSymbol);
                }
            CutSymblos();
        }

        private void AbsorbSymbol(int x, int y, Symbol symbol)
        {
            if (x < 0 || x >= _image.Width || y < 0 || y > _image.Heig
                return;
            if (_matrix[y][x] == 0)
                return;

            symbol.AddBlackPoint(x, y);
            _matrix[y][x] = 0;
            for (int i = 0; i < 8; i++)
                for (int j = 0; j < 8; j++)
                    AbsorbSymbol(x + dx[i], y + dy[j], symbol);
        }

        private void CutSymblos()
        {
            foreach (var symbol in _symbols)
                symbol.CutSymbol(ref _image);
        }
}


static class Mat
{
        static public int[][] GetMatrix(Bitmap image)
        {
            int[][] matrix = new int[image.Height][];
            for (var i = 0; i < image.Height; i++)
            {
                matrix[i] = new int[image.Width];
                for (var j = 0; j < image.Width; j++)
                    if (isBlack(image.GetPixel(j, i)))
                        matrix[i][j] = 1;
                    else
                        matrix[i][j] = 0;
```

```
    }

    return matrix;
}

static public int[] GetVector(Bitmap image)
{
    var vector = new int[16 * 16];
    for (var i = 0; i < 16; i++)
        for (var j = 0; j < 16; j++)
            if (isBlack(image.GetPixel(j, i)))
                vector[i * 16 + j] = 1;
            else
                vector[i * 16 + j] = 0;

    return vector;
}


static public int[] GetEnterVector()
{
    return new int[1];
}

static public int[] GetSpaceVector()
{
    return new int[2];
}

static public bool isSpaceVector(int[] vector)
{
    return vector.Length == 2;
}

static public bool isEnterVector(int[] vector)
{
    return vector.Length == 1;
}

static public bool isBlack(Color color)
{
    if (color.R == 0 && color.G == 0 && color.B == 0)
        return true;
```

```
            else
                return false;
        }

        static public double CalculateVectorLength(int[] input)
        {
            double sumOfSquares = 0;
            for (var i = 0; i < input.Length; i++)
            {
                sumOfSquares += input[i] * input[i];
            }

            return Math.Sqrt(sumOfSquares);
        }

        static public double CalculateVectorLength(double[] input)
        {
            double sumOfSquares = 0;
            for (var i = 0; i < input.Length; i++)
            {
                sumOfSquares += input[i] * input[i];
            }

            return Math.Sqrt(sumOfSquares);
        }

        static public double[] NormaizeVector(int[] input)
        {
            double[] normalizedVector = new double[input.Length];
            var vectorLength = CalculateVectorLength(input);

            for (var i = 0; i < input.Length; i++)
            {
                normalizedVector[i] = input[i] / vectorLength;
            }
            return normalizedVector;
        }
}


class Recognizer
{
        private KohonenNetwork _kn;
```

```
private TextDetector _td;

private double[][] _matchingPercents;

public Recognizer(KohonenNetwork kn, TextDetector td)
{
    _kn = kn;
    _td = td;
    StudyNetwork();
}

public List<Symbol> GetSymbols()
{
    return _td.Symbols;
}

public double[][] GetMatchingPercents()
{
    return _matchingPercents;
}

public string Parse(Bitmap image)
{
    Filter.Median3x3(ref image);
    Filter.Monochrome(ref image, 100);

    _td.Detect(image);
    var vectros = _td.GetVectors();
    var message = String.Empty;

    int i = 0;
    foreach (var vector in vectros)
        if (Mat.isSpaceVector(vector))
            message += ' ';
        else if (Mat.isEnterVector(vector))
            message += '\n';
        else
        {
            message += (char)(_kn.Parse(vector) + 'A');
            _matchingPercents[i++] = _kn.MatchingPercents;
        }

    return message;
```

```
        }

        private void StudyNetwork()
        {
            for (var i = 0; i < 10; i++)
            {
                LearnFont("Arial16x16(photo)changed");
                LearnFont("Arial16x16(50)");
                LearnFont("Arial16x16photo");
                LearnFont("Arial16x16(photo)");
                LearnFont("Arial16x16");
            }
        }

        private void LearnFont(string Font)
        {
            Bitmap bmp;
            int[] vector;
            char symbol;

            _kn._fonts++;

            for (var i = 0; i < 26; i++)
            {
                symbol = (char)('A' + i);
                bmp = new Bitmap(Bitmap.FromFile(@"C:\Users\Alexander\
                    + Font + @"\" + symbol + ".bmp"));
                vector = Mat.GetVector(bmp);
                _kn.Study(vector, i);
            }
        }

        public void SetKohonenNetwork(KohonenNetwork kn)
        {
            _kn = kn;
        }

        public void SetTextDetector(TextDetector td)
        {
            _td = td;
        }
}
```

ПРИЛОЖЕНИЕ В
Unit-тесты

```
using System;
using ManyWindows;
using System.Drawing;
using Microsoft.VisualStudio.TestTools.UnitTesting;

namespace UnitTestProject
{
    [TestClass]
    public class MatTest
    {
        [TestMethod]
        public void TestCalculatingVectorLengthInt()
        {
            int[] vector=new int[3];
            for (int i=0;i<3;i++)
            {
                vector[i] = i+1;
            }
            double d;
            double answer = Math.Sqrt(14);
            d=Mat.CalculateVectorLength(vector);
            Assert.AreEqual(answer, d, 0.001, "Length not calculeted c
        }
        [TestMethod]
        public void TestCalculatingVectorLengthDouble()
        {
            double[] vector=new double[3];
            for (int i=0;i<3;i++)
            {
                vector[i] = i+1;
            }
            double d;
            double answer = Math.Sqrt(14);
            d=Mat.CalculateVectorLength(vector);
            Assert.AreEqual(answer, d, 0.001, "Length not calculeted c
        }
        [TestMethod]
        public void TestIsBlack()
```

```
{
    Color color = Color.Black;
    bool answer=true;
    bool b;
    b=Mat.isBlack(color);
    Assert.AreEqual(answer, b,"Length not calculeted correctly
}
[TestMethod]
public void TestNormalizeVector()
{
    int[] vector = new int[3];
    for (int i = 0; i < 3; i++)
    {
        vector[i] = i + 1;
    }
    double[] d = Mat.NormaizeVector(vector);
    double[] answer = new double[3];
    for (int i = 0; i < 3; i++)
    {
        answer[i] = (i + 1) / Math.Sqrt(14);
    }
    for (int i = 0; i < 3; i++)
    {
        Assert.AreEqual(answer[i], d[i], "Vector not normalize
    }
}
[TestMethod]
public void TestGetMatrix()
{
    Bitmap bmp = new Bitmap(3, 3);
    int[][] resMatrix = Mat.GetMatrix(bmp);
    int[][] matrix = new int[3][];
    for (var i = 0; i < 3; i++)
    {
        matrix[i] = new int[3];
        for (var j = 0; j < 3; j++)
            matrix[i][j] = 1;
    }
    for (var i = 0; i < 3; i++)
    {
        for (var j = 0; j < 3; j++)
        {
            Assert.AreEqual(resMatrix[i][j], matrix[i][j], "Ma
```

```
        }
    }

}
[TestMethod]
public void TestGetFullVector()
{
    var vector = Mat.GetFullVector();
    for (int i = 0; i < vector.Length; i++)
    {
        Assert.AreEqual(vector[i], 1, "Vector not got correctl
    }
}
[TestMethod]
public void TestGetEmptyVector()
{
    var vector = Mat.GetEmptyVector();
    for (int i = 0; i < vector.Length; i++)
    {
        Assert.AreEqual(vector[i], 0, "Vector not got correctl
    }
}
[TestMethod]
public void TestGetVector()
{
    Bitmap bmp = new Bitmap(16,16);
    var vector = Mat.GetFullVector();
    var fullvector = Mat.GetVector(bmp);
    for (int i = 0; i < vector.Length; i++)
    {
        Assert.AreEqual(vector[i], fullvector[i], "Vector not
    }
}
[TestMethod]
public void TestIsMethods()
{
    bool[] b = new bool[2];
    var fullvector = Mat.GetFullVector();
    var emptyvector = Mat.GetEmptyVector();
    b[0] = Mat.isEmptyVector(fullvector);
    b[1] = Mat.isFullVector(emptyvector);
    for (int i = 0; i < 2; i++)
    {
```

```
                Assert.AreEqual(b[i], false, "Is methods work not corr
            }
        }
    }
}
```