МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ОМСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра «Прикладная математика и фундаментальная информатика»

Допускается к защите
Зав. кафедрой ПМиФИ,
д-р физ.-мат.наук, проф.
_____А. В. Зыкина
«___»_____ 2015 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

на тему «Восстановление изображений с помощью самоорганизующихся карт Кохонена»
студента Метешова Артема Николаевича группы МО-510

Пояснительная записка
Шифр работы ВКР – 2068998 – 56 – 13
Специальность 010503.65 «Математическое обеспечение и администрирование информационных систем»

Руководитель работы,
к.т.н., доцент
_____ Ю. С. Ракицкий
«___»_____ 2015 г.

Разработал студент
_____ А. Н. Метешов
«___»_____ 2015 г.

Нормоконтролёр:
_____О. Н. Канева
«___»_____ 2015 г.

Омск 2015

МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ОМСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра «Прикладная математика и фундаментальная информатика»

Зав. кафедрой ПМиФИ,
д-р физ.-мат. наук, проф.
___ А. В. Зыкина
«___»_____ 2015 г.

ЗАДАНИЕ

НА ВЫПОЛНЕНИЕ ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

студента Метешова Артема Николаевича

Тема работы: «Восстановление изображений с помощью самоорганизующихся карт Кохонена»

утверждена распоряжением по факультету от «13» февраля 2015 г. № 10

Срок сдачи студентом законченной работы «18» июня 2015 г.

Исходные данные к работе: Кохонен Т. Самоорганизующиеся карты / / Т. Кохонен. – М. Бином: Лаборатория знаний, 2008 – 655 с.

Содержание пояснительной записки (перечень подлежащих разработке разделов):

1. Модель нейросети Кохонена.

2.Разработка программы для восстановления изображений.

Перечень графического материала: не менее 5 слайдов презентации доклада

Дата выдачи задания «02» февраля 2015 г.

Руководитель работы, к.т.н., доцент Ю. С. Ракицкий_____

Задание принял к исполнению студент _____«02» февраля 2015 г.

Реферат

Пояснительная записка 78 с., 18 рис., 1 табл., 15 источников ,3 прил.

ЦИФРОВОЕ ИЗОБРАЖЕНИЕ, ПИКСЕЛЬ, КАРТА КОХОНЕНА, ПОВРЕЖДЕННЫЙ ПИКСЕЛЬ, НЕЙРОН, ВОССТАНОВЛЕНИЕ ИЗОБРАЖЕНИЯ

Объектом выпускной квалификационной работы являются поврежденные цифровые изображения.

Целью выпускной квалификационной работы является разработка приложения для восстановления поврежденных цифровых изображений с использованием самоорганизующейся карты Кохонена.

В ходе работы были изучены карты Кохонена, а именно – математическая модель сети и процесс обучения.

В результате была разработана модель нейронов, подходящих для работы с изображениями, а так же адаптирован алгоритм обучения сети для работы с нейронами, и разработан алгоритм для восстановления изображений, после чего для практического исследования было разработано приложение на языке C#.

Содержание

Введение

Проблема восстановления цифровых изображений возникает в следствии многих причин – неисправность носителя, ошибки при передаче файла по сети, искажение из-за ошибок операционной системы и так далее. Визуально это может проявляться как появление темных областей на изображении, появление зашумленности, отсутствие части изображения. Для избавления изображения от повреждений используются два основных подхода. Первый подход – это использование сглаживающих фильтров, второй – поиск поврежденных пикселей в изображении и их восстановление. Использование фильтров дает хороший визуальный результат, но приводит к искажению изображения. Восстановление изображения по пикселям является более затратным по времени способом, однако потенциал этого метода является большим, так как затрагиваются только поврежденные части изображения, а следовательно и искажения изображения будут меньшими в сравнении с первым подходом.

Существует множество алгоритмов для восстановления поврежденных пикселей, однако практически все они основаны на анализе неповрежденных пикселей, расположенных по соседству с поврежденным. Поиск цвета для поврежденного пикселя происходит с помощью разных методов, например, с помощью арифметического среднего, или с помощью метода иерархий [4].

Однако при большой зашумленности изображений, либо при отсутствии крупной его части (более 30%) результат данных алгоритмов будет иметь визуальные искажения. Большей устойчивостью потенциально обладает метод восстановления пикселей с использованием нейронных сетей.

Нейронные сети – математические модели, а так же их программные или аппаратные реализации, построенные по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма [5]. Они были созданы для описания ситуаций, возникающих в реальном мире, наравне с другими вычислительными формализмами, такими

как: вероятностные рассуждения, теория нечетких множеств, нечеткая логика, генетические алгоритмы.

Существует немало видов нейронных сетей и все они классифицируются по различным характеристикам, к примеру, по количеству слоев нейронов, по характеру обучения сети (с учителем или без), по типу подстройки весов, по типу входной информации, по применяемой модели нейронной сети.

Биологические сети тесно связаны с искусственными нейронными сетями, поэтому искусственные сети должны обладать некоторыми свойствами биологических, а именно – иметь аналоговое представление и обработку информации, иметь способность осуществлять осреднение согласно определенным условиям для набора данных в целом, быть отказоустойчивыми, быть адаптируемыми к изменяющимся внешним условиям.

С помощью искусственных нейронных сетей пытаются моделировать когнитивные функции мозга человека, такие как сенсорные (искусственное восприятие), моторные функции, возможность принятия решений (рассуждение, оценка, решение проблемы), понимание и порождение речи, чтение и письмо, возможность ведения диалога.

Целью данной работы является исследование восстановления изображений при помощи нейронных сетей, а именно – при помощи самоорганизующихся карт Кохонена. Для достижения поставленной цели необходимо решить ряд задач, а именно – изучить теоретические основы карт Кохонена, построить модель нейронной сети для работы с цифровыми изображениями.

1 Модель нейросети Кохонена

1.1 Самоорганизующиеся карты Кохонена

Самоорганизующиеся карты Кохонена(Self-Organaizing Map – SOM) – разновидность нейронных сетей, использующая обучение без учителя. В своем основном варианте, описанном Тойво Кохоненом, SOM создает граф подобия входных данных. Она преобразует нелинейные статистические соотношения между многомерными данными в простые геометрические связи между изображающими их точками на устройстве отображения низкой размерности, обычно в виде регулярной двумерной сетки узлов. Поскольку SOM осуществляет сжатие информации с сохранением в получаемом изображении наиболее важных топологических и (или) метрических связей между первичными элементами данных, можно считать, что с ее помощью порождаются абстракции или обобщения некоторого вида. Формально SOM можно определить как нелинейное, упорядоченное, гладкое отображение многообразий входных данных высокой размерности на элементы регулярного массива низкой размерности[1].

SOM состоит из набора нейронов, количество которых задается аналитиком. Нейрон описывается двумя векторами – вектор веса, размерность которого обычно совпадает с размерностью вектора входных данных, и вектора координат, который определяет местоположение нейрона на карте.

В основном карты Кохонена используют для решения таких задач как моделирование, прогнозирование, поиск закономерностей в больших массивах данных, выявление наборов независимых признаков, сжатие информации, классификация входных данных.

1.2 Модель нейрона

Большинство нейросетевых моделей, особенно сети прямого распространения с передачей сигналов, предполагают использование нейронов, представляющих собой статические элементы с множеством входов и одним выходом. Эти элементы формируют взвешенную сумму входных величин ξ_i,

называемую входной активацией I_i, которую усиливают в нелинейной выходной схеме до величины η_i (рис. 1).

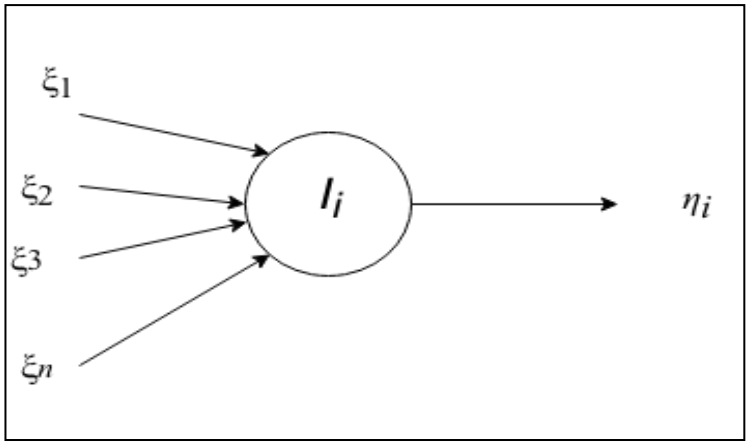

Рисунок 1 – Статическая нелинейная модель нейрона

Такие модели нейрона просты в реализации, и к их выходам относительно несложно присоединить интегратор, как это делается в традиционных аналоговых вычислителях, что дает возможность решать динамические задачи.

Так же существуют модели нейронов, основанные на биофизических теориях, в которых производится попытка описания явления электрохимического триггерного переключения в активных клеточных мембранах биологических нейронов. В связи с такими задачами наиболее часто упоминаются классические уравнения Ходжкина – Хаксли. Наблюдаемые ритмы для входных и выходных импульсов описываются достаточно точно, однако такой уровень анализа очень сложен для описания преобразований сигнала в сложных нейронных сетях.

Модель нейронов, предложенная Тойво Кохоненом, представляет собой упрощенное, практическое уравнение, решение которого достаточно точно отражает типичное нелинейное динамическое поведение для многих видов нейронов. В отличие от других моделей, связывающих нейронную активность с мембранными потенциалами, в этой модели неотрицательные значения сигнала прямо описывают частоты повторения импульсов, или, более точно, обратные значения интервалов между последовательными нейронными импульсами.

На рисунке 1 ξ_i и η_i – неотрицательные скалярные величины, входная активация I_i является функцией от ξ_i и некоторого набора внутренних параметров. Если I_i используется в линеаризованной форме, то входные величины определенным образом ассоциируются с параметрами интенсивности μ_{ij}, которые часто именуют так же синаптическими весами, после чего полученный результат подается на вход модели нейрона. В этом случае нейрон работает подобно квазиинтегратору (интегратору с потерями), а эффект утечки в такой модели будет нелинейным. Входную активацию можно аппроксимировать следующим образом:

$$I_i = \sum_{j=1}^{n} \mu_{ij} \xi_j. \quad (1)$$

Для входной активации могут так же рассматриваться и другие законы, являющиеся нелинейными функциями от ξ_i. Система уравнений для квазиинтегратора может быть записана в виде:

$$\frac{d\eta_i}{dt} = I_i - \gamma_i(\eta_i), \quad (2)$$

при $\eta_i > 0$, где $\gamma_i(\eta_i)$ – член, соответствующий утечке, нелинейная функция выходной активности.

Чтобы гарантировать устойчивость, особенно в сетях с обратными связями, эта функция должна быть выпуклой, по крайней мере при больших значениях η_i, то есть ее вторая производная по аргументу η_i должна быть положительной. Член утечки такого вида достаточно правдоподобно описывает суммарные потери и влияние зон нечувствительности в нейроне, в виде пропорционально увеличивающейся функции активности. Следует отметить, что уравнение (2) справедливо только для $\eta_i \geq 0$. Так как предполагается, что значение η_i ограничено снизу нулем, то производная так же должна быть равна

нулю, если $\eta_i = 0$ и правая часть уравнения отрицательна. Если $\eta_i = 0$ и правая часть уравнения положительна, то производная должна быть положительна. Более точно уравнение (2) можно представить в виде:

$$\frac{d\eta_i}{dt} = \varphi[\eta_i, I_i - \gamma_i(\eta_i)], \tag{3}$$

где $\varphi[a, b] = \max\{H(a)b, b\}$, H – функция Хевисайда: $H(a) = 0$ при $a < 0$ и $H(a) = 1$ при $a \geq 0$.

1.3 Явление самоорганизации в SOM

Для описания процесса самоорганизации Тойво Кохонен приводит практический пример и доказательство утверждения, сформулированного им же в книге [1].

Пусть имеется одномерный, линейный, незамкнутый массив функциональных элементов, с каждым из которых связан скалярный входной сигнал ξ. Элементы массива пронумерованы числами 1,2,…,l и каждому i-му элементу поставлено в соответствие единственное скалярное входное весовое или эталонное значение μ_i. Сходство между ξ и μ_i описывается абсолютным значением их разности, а наилучшее соответствие определяется выражением:

$$|\xi - \mu_c| = \min_i |\xi - \mu_i|. \tag{4}$$

Множество N_c элементов, выбранных для обновления, определяется следующим образом:

$$N_c = \{\max(1, c-1), c, \min(l, c+1)\}. \tag{5}$$

Таким образом, i-й элемент имеет соседей с номерами $i-1$ и $i+1$, исключая края массива, где соседним с первым элементом является второй

элемент, а с l-м элементом соседствует $(l-1)$-й элемент. Тогда N_c есть множество, в которое входит элемент с индексом c и его ближайшие соседи.

Характер процесса обучения в основном не изменяется для различных $\alpha > 0$. Главным образом от α зависит скорость обучения. Для случая непрерывного времени соответствующие уравнения имеют вид:

$$\frac{d\mu_i}{dt} = \alpha(\xi - \mu_i), \tag{6}$$

для $i \in N_c$, и

$$\frac{d\mu_i}{dt} = 0, \tag{7}$$

во всех остальных случаях.

Утверждение 1: пусть ξ – случайная величина. Процесс обучения начинается со случайного выбора начальных значений для величин μ_i, эти значения постепенно будут меняться согласно соотношениям (4) – (7), так что при $t \to \infty$ множество чисел $(\mu_1, \mu_2, \dots, \mu_l)$ станет упорядоченной возрастающей или убывающей последовательностью. После того как данное множество станет упорядоченным, оно будет оставаться таким же для всех t. Кроме того, функция точечной плотности для μ_i будет в конечном счете аппроксимировать некоторую монотонную плотность распределения вероятностей $p(\xi)$ для величины ξ.

Данное утверждение доказывается в два этапа.

Этап 1. Упорядочивание весов

Утверждение 2: в процессе, определяемом выражениями (4) – (7), величины μ_i становятся упорядоченными с вероятностью единица в возрастающем или убывающем порядке при $t \to \infty$.

Используя аргументацию, представленную Гренандером для схожей проблемы [1] , можно определить схему доказательства следующим образом. Пусть $\xi = \xi(t) \in R$ – случайная (скалярная) входная величина, которая имеет плотность распределения вероятностей $p(\xi)$ на конечном носителе, при этом $\xi(t_1)$ и $\xi(t_2)$ независимы для любых $t_1 \neq t_2$.

Доказательство следует из общих свойств марковских процессов, а именно – из факта существования поглощающего состояния, для которого вероятность перехода в себя равна единице. В [1] показано, что если такого рода состояние достигается в ходе процесса, начавшегося из произвольного начального состояния, с помощью некоторой последовательности входов, имеющих положительную вероятность, то для случайной последовательности входов поглощающее состояние достигается с вероятностью равной единице при $t \to \infty$.

Поглощающее состояние определено для любой из упорядоченных последовательностей μ_i, однако, следует отметить, что есть две упорядоченные последовательности, которые вместе и определяют поглощающее состояние.

Далее на оси вещественных чисел выбирается такой интервал, чтобы на нем величина ξ имела положительную вероятность. Повторяя неоднократно выбор значений ξ из этого интервала, можно перенести все μ_i внутрь него за конечное время. После этого становится возможным осуществить выбор значений ξ так, что если к примеру значения μ_i, $\mu_{i\text{-}1}$, $\mu_{i\text{-}2}$ были неупорядоченными, то значение $\mu_{i\text{-}1}$ окажется между значениями $\mu_{i\text{-}2}$ и μ_i. Если с обеих сторон от какого-либо элемента есть нарушение порядка, то можно определить, какая сторона будет упорядочена первой. Например, если:

$$\mu_{i-1} < \mu_{i-2} < \mu_i \,, \tag{8}$$

то выбор ξ из окрестности μ_i перенесет μ_{i-1} между μ_{i-2} и μ_i, при этом μ_{i-2} не изменится. Продолжая сортировку таким образом получим полное

упорядочивание за конечное количество шагов, и поскольку ξ реализуется с положительной вероятностью, утверждение (2) доказано.

Этап 2. Фаза сходимости.

После того, как все μ_i становятся упорядоченными, особый интерес представляет их завершающая сходимость к асимптотическим значениям, поскольку они представляют собой образ входного распределения $p(\xi)$.

Полагается, что μ_i уже упорядочены. Целью является вычисление асимптотических значений μ_i.

Свойства сходимости для значений μ_i автор обсуждает в менее строгом варианте, а именно, анализирует только динамическое поведение математических ожиданий $E\{\mu_i\}$. Автор показывает, что эти величины сходятся к единственным пределам.

1.4 Вектор входных данных

В физической теории информационных процессов совокупности смежных в пространстве или во времени значений сигналов принято считать образами, которые можно трактовать как упорядоченные наборы. Такие наборы обычно описывают с помощью векторов представления.

Основной источник наблюдаемых данных в данной работе – это изображение, которое зачастую имеет прямоугольную форму и может быть представлено в виде двумерной сетки, узлами которой являются пиксели изображения. Для формирования вектора входных данных было решено использовать не последовательный набор пикселей, а совокупность пикселей, которые располагаются рядом с выбранной каким-либо образом точки на изображении. В связи с этим для описания способа построения вектора входных значений необходимо ввести следующие понятия и обозначения:

1) поврежденный пиксель – пиксель, для которого отсутствует цвет в палитре в силу повреждения;

2) W – ширина изображения в пикселях;

3) H – высота изображения в пикселях;

4) $p[\ i, j\ ]$ – пиксель, расположенный в точке (i , j) на изображении, где $i \in W$, а $j\ \in H$;

5) $r[\ i,\ j\ ]$ – восстановленное значение пикселя в точке (i , j) на изображении, где $i\ \in W$, а $j \in H$;

6) S – размер вектора входных данных;

7) N_x – ширина карты нейронов;

8) N_y – высота карты нейронов;

9) N – количество нейронов на карте;

10) $cel(a,b)$ – целая часть деления a на b;

11) $ost(a,b)$ – остаток от деления a на b;

12) $n(x)$ – горизонтальная координата местоположения нейрона n;

13) $n(y)$ – вертикальная координата местоположения нейрона n;

14) «#» - цвет поврежденного пикселя.

В ходе работы было принято решение построения вектора входных значений по следующей формуле:

$$x[i](x,y) = p[\ l\ +\ ost(i,S), t + cel(i,S)], \tag{9}$$

где $l = x - S/2\ ,\ t = y - S/2,\ x$ и y – координаты пикселя, для которого строится вектор, $i\ =\ \overline{1,S}$.

Таким образом, значениями вектора $x[i]$ будут значения цветов пикселей, которые находятся в области рядом с точкой (x,y) на изображении.

Так же необходимо ввести некоторые ограничения:

1) $\frac{S}{2}\ \leq x\ \leq W - \left(\frac{S}{2}\right)$;

2) $\frac{S}{2}\ \leq y\ \leq H - \left(\frac{S}{2}\right)$;

3) S – нечетное.

Условия (1) и (2) вводятся в силу того, что размер блока пикселей для составления вектора входных данных больше, чем один пиксель, а значит и количество таких блоков будет меньше количества пикселей. Последнее

условие необходимо для того, чтобы область вокруг пикселя была более симметричной.

1.5 Мера соседства нейронов и мера сходства

Все наблюдаемые векторы должны быть представлены в пространстве, обладающем некоторой метрикой. Последняя является свойством любого набора элементов, характеризуемой некоторой функцией $d(x, y)$, которую называют расстоянием и которая определена для любой пары (x, y). При выборе функции расстояния необходимо, чтобы она удовлетворяла следующим условиям:

1) $d(x, y) \geq 0$, при $x = y$;

2) $d(x, y) = d(y, x)$;

3) $d(x, y) \leq d(x, z) + d(z, y)$.

Примером функции, определяющей расстояние, которая удовлетворяет вышеперечисленным условиям, является евклидово расстояние на плоскости.

Мерой соседства между нейронами на карте называют функцию от номера итерации и индексов нейронов. Обычно в этих целях используют функцию Гаусса вида:

$$h(k_1, k_2, i) = \alpha(i) \exp\left(-\frac{d(k_1, k_2)^2}{2\delta(i)}\right), \quad (10)$$

где k_1 и k_2 – индексы нейронов, i – номер итерации, $\alpha(i)$ имеет вид (11), а $\delta(i)$ имеет вид (12), $d(k_1, k_2)$ имеет вид (13).

$$\alpha(i) = \begin{cases} 1, \text{если } i < 10 \\ \dfrac{1}{(i-9)^{0,2}}, \text{если } i \geq 10 \end{cases}, \quad (11)$$

$$\delta(i) = 5\frac{\sqrt{N}}{\sqrt{i}}, \quad (12)$$

$$d(k_1, k_2) = min\sqrt{(n_{k_1}(x) - n_{k_2}(x))^2 - (n_{k_1}(y) - n_{k_2}(y))^2}. \quad (13)$$

Функция (11) приведена к такому виду для равномерной инициализации карты.

Мера сходства – это функция, которая показывает насколько один нейрон «похож» на другой. Для сравнения нейронов в классическом варианте SOM Кохоненом приводится несколько вариантов меры сходства.

Сравнение образов часто основывается на их корреляции. Если имеются два упорядоченных набора отсчетов сигнала $x = (\varepsilon_1, \varepsilon_2, \dots, \varepsilon_n)$ и $y = (\gamma_1, \gamma_2, \dots, \gamma_n)$. Тогда их ненормированной корреляцией будет называться величина вида (14).

$$C = \sum_{\gamma=1}^{n} \varepsilon_i \gamma_i \,, \quad (14)$$

Если *x* и *y* как евклидовы векторы, тогда *C* есть ни что иное как скалярное произведение двух векторов.

В случае, когда значимая информация в образах или сигналах содержится в относительных величинах их компонентов, тогда сходство лучше измерять в терминах направляющих косинусов. Пусть $x \in R^n$, $y \in R^n$, и они представляют собой евклидовы векторы, тогда выражение вида (15) есть по определению направляющий косинус угла между ними:

$$\cos\theta = \frac{(x, y)}{\|x\|\|y\|}, \quad (15)$$

где (x, y) – скалярное произведение векторов *x* и *y*; $\|x\|$ – евклидова норма вектора *x*.

Другая мера сходства – евклидово расстояние между векторами x и y, определяется следующим выражением:

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(\varepsilon_i - \gamma_i)^2}\,, \tag{16}$$

В метрике Минковского, которая является обобщением метрики (16), расстояние определяется следующим образом:

$$d(x,y) = (\sum_{i=1}^{n}|\varepsilon_i - \gamma_i|^{\alpha})^{1/\alpha}\,, \tag{17}$$

где $\alpha \in R$, $\alpha > 0$.

Так же в качестве функции меры сходства двух векторов часто используется среднеквадратическое отклонение. В этом случае функция $d(x,y)$ будет выглядеть следующим образом:

$$d(x,y) = \sum_{l}(x[l] - y[l])^2. \tag{18}$$

Все вышеперечисленные способы измерения сходства двух векторов применяются в различных ситуациях. В данной работе в качестве меры сходства было решено использовать среднеквадратическое отклонение, как наиболее простое в реализации.

1.6 Основные правила обучения нейронных сетей

1.6.1 Правило Хебба

Если при моделировании нейросети предполагается, что она должна реализовывать простые эффекты запоминания в случаях, когда используется

ассоциативная или адресуемая память, обычно используется правило Хебба для обучения. Оно основано на следующей гипотезе: если аксон клетки А находится в состоянии, близком к требуемому для возбуждения клетки Б, а также периодически или постоянно участвует в ее активации, имеет место процесс роста или метаболического изменения в одной или обеих клетках таким образом, что эффективность клетки А, как одной из воздействующих на клетку Б, возрастает [1].

Аналитически это означает, что вес μ_{ij} изменяется в соответствии с правилом:

$$\frac{d\mu_{ij}}{dt} = \alpha\eta_i\xi_i\,, \tag{19}$$

где ξ_i – пресинаптическая активность, η_i – постсинаптическая активность, α – скалярный параметр (коэффициент скорости обучения). Это соотношение и есть правило Хебба, которое происходит от некоторых элементарных моделей ассоциативной памяти, называемых матричной корреляционной памятью.

Правило Хебба имеет ряд недостатков. В частности, поскольку величины ξ_i и η_i описывают частоты и, следовательно, неотрицательны, величины μ_{ij} могут изменяться только монотонно. Например, при $\alpha > 0$, величины μ_{ij} могут неограниченно возрастать. Необходимо вводить определенные ограничения, которые будут описывать эффект насыщения. Так же в более сложных нейросистемах должно выполняться условие обратимости изменений весов μ_{ij}.

1.6.2 Правило обучения типа Риккати

Первое изменение, относительно правила Хебба, это введение в функцию обучения члена для пластичности, который описывает следствия активности в окружении некоторого нейрона. Второе изменение – введение члена типа «активного забывания», который гарантирует, что значения вектора веса остаются конечными и весь весовой вектор и его члены будут нормированы.

Для описания каждого из введенных членов автором вводятся следующие обозначения:

1) P – скалярная функция управления пластичностью;
2) Q – скалярная функция забывания.

Поскольку пластичность должна влиять на суммарную скорость обучения, то можно записать уравнение обучения в виде:

$$\frac{d\mu_{ij}}{dt} = P\left(\xi_i - Q\mu_{ij}\right), \tag{20}$$

где ξ_i – обучающий сигнал.

1.7 Общий алгоритм SOM

Общий алгоритм определяет регрессионный рекурсивный процесс специального вида, в котором на каждом шаге осуществляется обработка только части модели. Он определяет отображение входного пространства данных R^n на n-мерную решетку. Так как в ходе данной работы исходными данными являются вектора, построенные на основе изображений, то далее будет использоваться частный случай n-мерной решетки – двумерная решетка. Каждому узлу решетки ставится в соответствие параметрический вектор модели, называемый опорным вектором $m_i = (\mu_{i1}, \mu_{i2}, \dots, \mu_{in})^T \in R^n$. Перед началом процесса обучения все вектора должны быть инициализированы. Если значениями векторов являются случайные числа, то при начале процесса из произвольного исходного состояния векторы распределятся упорядоченно на двумерной решетке при условии, что количество итераций процесса обучения велико. Этот эффект и есть основа самообучения SOM.

Массив нейронов может образовывать решетку прямоугольного, гексагонального или нерегулярного типов.

Пусть $x \in R^n$ – случайный вектор входных данных. Далее этот вектор сравнивается со всеми векторами m_i и ищется наиболее подходящий узел (BMU

– best matching unit). В общем случае пользуются евклидово расстояние. Индекс искомого узла определяется выражением:

$$C = \min_i \|x - m_i\|. \quad (21)$$

В процессе обучения те узлы, которые топографически близки к BMU, будут активировать друг друга, обучаясь в определенной степени за счет этого на одном и том же векторе входных данных. Это является причиной локальной релаксации или эффекта сглаживания для векторов веса нейронов в рассматриваемой окрестности, что при продолжительном обучении приводит к упорядочиванию нейронов по всей карте.

В общем случае обучающий процесс задается функцией:

$$m_i(t+1) = m_i(t) + h_{ci}(t)[x(t) - m_i(t)], \quad (22)$$

где t – целочисленная величина, равная номеру итерации, а начальные значения векторов весов нейронов могут быть случайными числами. Основную роль тут играет функция $h_{ci}(t)$. Ее так же называют сглаживающей функцией. Для сходимости процесса, необходимо чтобы выполнялось условие: при $t \to \infty$, $h_{ci} \to 0$. Чаще всего в качестве функции $h_{ci}(t)$ используется функция Гаусса. В данной работе использовалась функция Гаусса вида (10), но в общем виде эту функцию можно представить в виде:

$$h_{ci}(t) = h(\|r_c - r_i\|, t), \quad (23)$$

где $r_c \in R^2$ и $r_i \in R^2$ – векторы, определяющие размещение узлов c и i, соответственно, в рассматриваемой решетке. С возрастанием $\|r_c - r_i\|$ выполняется условие $h_{ci} \to 0$. Средняя ширина и форма функции $h_{ci}(t)$

определяет «жесткость» поверхности, которая модифицируется так, чтобы наилучшим образом соответствовать обрабатываемым данным.

Пусть N_c – множество индексов точек, которые находятся рядом с узлом c на решетке. Тогда $h_{ci}(t) = \alpha(t)$, если $i \in N_c$ и $h_{ci}(t) = 0$ в противном случае. Значение коэффициента $\alpha(t)$ определяется величиной коэффициента скорости обучения ($0 < \alpha(t) < 1$). Значения $\alpha(t)$ и N_c обычно монотонно уменьшаются в ходе процесса обучения. Следует отметить, что при относительно небольшом размере карты (если количество нейронов в ней примерно несколько сотен) выбор параметров обучения, а именно – скорости обучения, не сильно влияет на ход обучения. Более значимым параметром является выбор размера множества окрестности. Если этот параметр слишком мал, то в ходе обучения не удастся получить глобально упорядоченную карту. В данном случае для карты будут наблюдаться различного рода разбиения, имеющие мозаичный характер, между которыми направление упорядочивания изменяется скачкообразно. Этого явления можно избежать путем задания большого значения размера множества окрестности, которое будет сжиматься со временем.

При задаче параметров векторов весов нейронов случайными числами, то в течение первых тысячи итераций следует задать величину коэффициента скорости обучения близкой к единице, после чего это значение должно монотонно уменьшаться. Коэффициент скорости обучения является функцией, зависящей от времени (или от итераций). Вид зависимости может быть линейной, экспоненциальной либо обратно пропорциональной времени. В основном случае для задания коэффициента скорости обучения используют функцию:

$$\alpha(t) = 0{,}9\left(1 - \frac{t}{1000}\right). \tag{24}$$

Упорядочивание нейронов на карте происходит в течение начального периода работы алгоритма, остальные итерации требуются для более точной настройки карты. После завершения этапа упорядочивания коэффициент скорости обучения должен принимать малые значения, к примеру, $\alpha(t) = 0{,}02$, в течение большого числа итераций. На конечном этапе обучения не имеет значения, по какому закону уменьшается коэффициент скорости обучения.

Для больших карт важно минимизировать время обучения сети. В этом случае выбор функции для определения коэффициента скорости обучения является существенным. Выбор функции и параметров обучения в основном определяется экспериментальным путем.

Так как обучение является случайным процессом, то конечная статистическая точность отображения зависит от количества шагов на завершающем этапе данного процесса, который должен быть достаточно продолжительным. Эвристическое правило получения хорошей статистической точности состоит в том, что количество итераций обучения должно превышать количество нейронов в сети примерно в 500 раз. В то же время размерность вектора входных значений не влияет на число итерационных шагов, поэтому при программной реализации допускается использование высокой размерности входа. Следует отметить, что в силу небольшой вычислительной сложности алгоритма количество итераций для обучения сети лучше всего выбирать экспериментальным путем.

2 Разработка программы для восстановления изображений

2.1 Алгоритм обучения нейронной сети

Алгоритм обучения является итерационным. На нулевой итерации происходит инициализация карты, а именно – присвоение векторам веса нейронов нулевых значений. Это сделано для того, чтобы минимизировать вероятность ошибок в программе, связанных с простейшими операциями на следующих итерациях. Далее следуют итерации, на которых карта равномерно инициализируется для более качественного и равномерного обучения. Общая блок-схема алгоритма показана на рисунке 2.

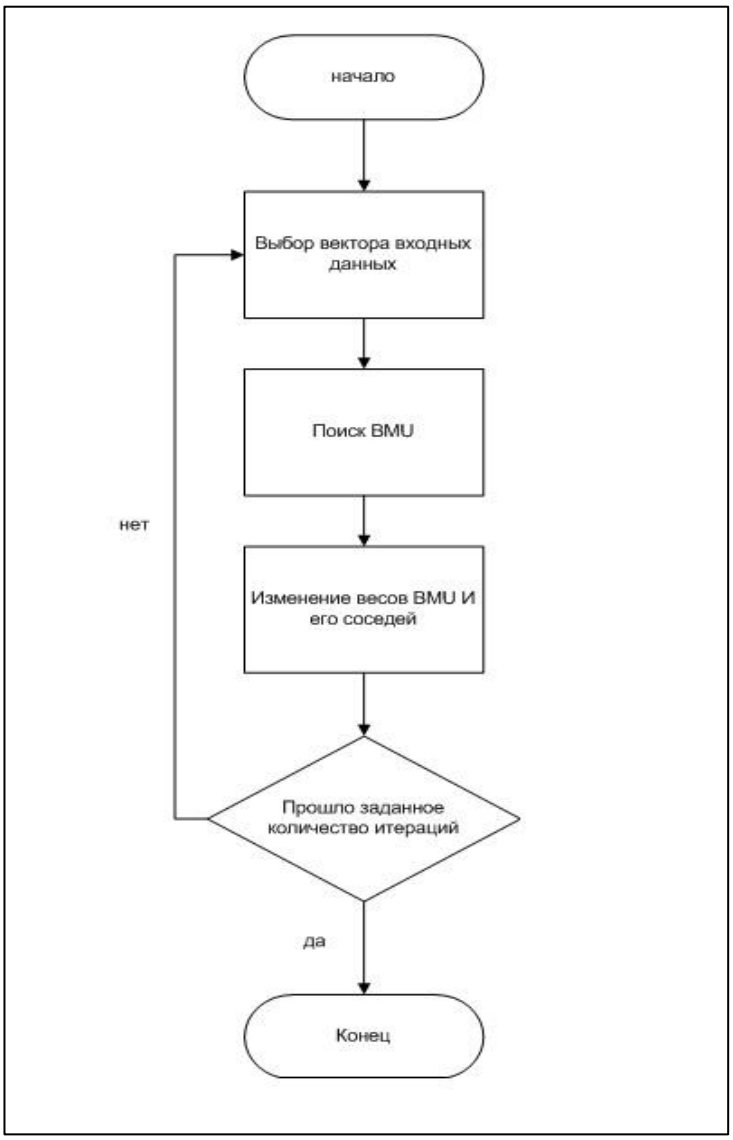

Рисунок 2 – Блок-схема алгоритма обучения

Как видно из рисунка 2, весь алгоритм обучения состоит из нескольких основных этапов.

На первом этапе строится вектор входных данных на основе исходного изображения. Ранее в работе был выбран внутренний формат этого вектора, который будет использоваться в модели.

Следующий этап – это поиск лучшего нейрона. Для этого находятся расстояния до всех векторов весов нейронов карты, после чего выбирается наиболее похожий нейрон. Возможна ситуация, особенно на первых итерациях алгоритма, когда вышеуказанному условию соответствует несколько нейронов. В этом случае BMU выбирается случайным образом.

Далее определяются все нейроны, которые находятся рядом с BMU, в какой-либо определенной заранее окрестности. После этого происходит изменение векторов весов нейронов по формуле:

$$m_i(t) = m_i(t-1) + h_{ci}(t)\big(x(t) - m_i(t-1)\big), \tag{25}$$

где $h_{ci}(t)$ – функция вида (10).

В качестве критерия остановки алгоритма обучения используется количество итераций цикла, которое задается перед началом обучения. Однако на этапе увеличения счетчика итераций так же проверяется количество нейронов, веса которых были изменены за одну итерацию. Если изменения карты незначительно малы, это означает что процесс обучения сети можно остановить.

Другой возможной альтернативой критерия остановки являлась ошибка карты, которая может вычисляться различными способами, к примеру, как среднее арифметическое расстояние между наблюдениями и векторами веса соответствующих ему BMU, но использование такого рода критерия необходимо при использовании более сложных мер соседства и сходства, либо при более сложном формате входных данных. В данном случае вычисления довольно простые, поэтому было решено использовать первый способ.

2.2 Алгоритм восстановления изображения

Этот алгоритм также является итерационным. Количество итераций определяется количеством поврежденных пикселей на изображении. При восстановлении из набора векторов входных данных выбирается такой вектор, чтобы в нем содержался всего один поврежденный элемент.

На следующем этапе ищется BMU среди нейронов обученной ранее карты, однако метрика поиска в этом случае меняется:

$$d\big(x(t), m_i(t)\big) = \sum_{\substack{l \\ x(t)[l] \neq \#}} (m_i(t)[l] - x(t)[l])^2. \tag{26}$$

Другими словами, при поиске наиболее подходящего нейрона не учитывается поврежденная компонента в векторе входных данных, а в векторе веса нейрона не учитывается компонента, которая находится на той же позиции в векторе веса, что и поврежденная в векторе входных данных.

После выбора BMU из его вектора веса берется необходимая компонента и вставляется в изображение вместо поврежденной.

Следует отметить, что после восстановления пикселя изображения, он учитывается при восстановлении других пикселей, которые находятся рядом с ним.

Процесс восстановления заканчивается после восстановления всех пикселей.

2.3 Основные возможности программы

В разрабатываемом приложении должны быть реализованы следующие основные функции:

1) приведение изображения к внутреннему формату программы;
2) обучение нейронной сети;
3) возможность сохранения состояния сети;
4) загрузка обученной сети;

5) обозначение поврежденной области;

6) восстановление изображения;

7) сохранение восстановленного изображения;

8) сбор различных параметров работы для проведения анализа результата.

Для реализации приложения был выбран язык программирования C# и среда разработки Microsoft Visual Studio 2012.

2.4 Внутренний формат изображения

Изображение по сути является матрицей, состоящей из пикселей, но для обучения сети нельзя подавать целое изображение, поэтому было решено составлять вектор входных значений из набора пикселей, находящихся вокруг какой-либо точки, выбранной заранее. Выбор этой точки можно производить разными способами – либо делать это случайным образом, либо путем наложения некоторой сетки на изображение с ячейками, размерность которых равна размерности вектора веса нейрона. В обоих случаях необходимо выполнять проверку на наличие поврежденных пикселей в наборе, и если такие пиксели имеются, то данный набор не подходит для обучения.

2.5 Обозначение поврежденной области

Поврежденная область цифрового изображения может быть различного характера, либо это локализованная область, внутри которой находятся поврежденные пиксели, либо изображение полностью зашумлено, то есть искажено полностью. Для выделения области в первом случае было решено добавить в интерфейс поля, в которые будут водиться координаты четырех точек, ограничивающих поврежденную область, то есть сама область будет иметь прямоугольную форму, это сделано для упрощения реализации и чтобы избежать фактора ошибки конечного пользователя, который будет выделять эту область. В любом случае перед началом обучения поврежденная область будет проверяться внутри программы для выделения точных границ. Если изображение зашумлено полностью, то перед началом работы необходимо будет оставить поля, описанные выше незаполненными.

Следует отметить, что повреждения на цифровых изображениях, которые восстанавливались с помощью проектируемой программы, имитировались путем изменения цвета «поврежденных» пикселей с исходного на цвет, которого нет в палитре цветов исходного изображения. В связи с этим было добавлено поле, в которое вводится цвет поврежденного пикселя в формате Argb.

2.6 Сохранение состояния обученной сети

Поскольку разрабатываемое приложение предназначено для многоразового использования, следует сохранять состояние обученной карты, а так же обеспечить пользователю возможность загрузки ранее сохраненного состояния карты. Для этих целей можно использовать два основных варианта – это сохранение состояния в базу данных, либо сохранение состояния в файл. Первый способ требует проектирования базы данных, создание таблиц и связей между ними, а так же реализации функций внутри программы для работы с это базой данных. Данный способ не является целесообразным, так как объем информации, которую необходимо сохранить, является небольшим. Более простым является второй способ. В ходе работы было решено использовать файлы в формате XML для сохранения состояния карты.

XML – рекомендованный Консорциумом Всемирной паутины (W3C) язык разметки. Спецификация XML описывает XML-документы и частично описывает поведение XML-процессоров (программ, читающих XML-документы и обеспечивающих доступ к их содержимому). XML разрабатывался как язык с простым формальным синтаксисом, удобный для создания и обработки документов программами и одновременно удобный для чтения и создания документов человеком, с подчёркиванием нацеленности на использование в Интернете[3].

В языке C# существует специальный объект для работы с XML-файлами. Этот объект находится в пространстве имен «System.Xml.Schema» и представлен двумя основными классами – «XmlSchemaElement» и «XElement». Первый служит для создания схемы, второй – непосредственно для создания

элементов файла. В рамках концепции XML каждый нейрон будет сохранен в файл как отдельный элемент, и его параметры, такие как вектор веса и координаты будут сохранены как свойства этого элемента. В приложении А показано итоговое содержание созданного программой XML-документа.

2.7 Результаты работы программы

Восстановленное изображение будет являться результатом работы программы, и будет сохраняться как отдельный файл в указанную пользователем директорию. Так же помимо изображения, для оценки результатов следует сохранить все входные данные, такие как:

1) размерность вектора веса нейрона;
2) количество итераций обучения сети;
3) количество нейронов в карте;
4) количество поврежденных пикселей в процентах;
5) время, затраченное на обучение сети;
6) время, затраченное на восстановление изображения;
7) информация об исходном изображении (высота и ширина в пикселях).

Сохранение всех вышеперечисленных параметров позволит производить калибровку программы с целью повышения эффективности. Данные будут сохраняться отдельным файлом в формате XML.

2.8 Интерфейс приложения

Учитывая все возможности и функции проектируемого приложения, описанные выше, был спроектирован интерфейс приложения. Для простоты реализации было решено использовать одно окно для интерфейса, поскольку основных функциональных блоков интерфейса немного.

Условно главное окно можно разделить на несколько блоков:

1) блок настроек нейронной сети;
2) блок выделения поврежденной области;
3) блок добавления поврежденного изображения;
4) блок просмотра результатов работы.

С помощью онлайн – инструмента «moqups.com» был разработан макет главного окна приложения, он представлен на рисунке 3.

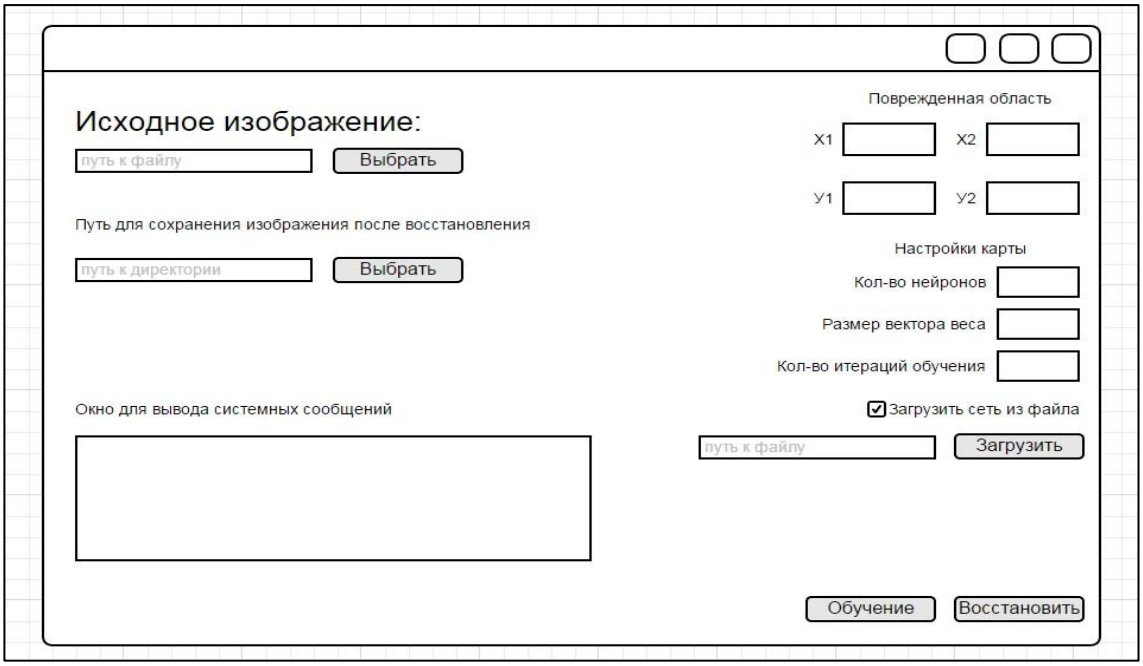

Рисунок 3 – Макет главного окна приложения

Для создания главного окна приложения по макету, Microsoft Visual Studio 2012 обладает встроенным инструментом, который называется «Конструктор форм». Этот инструмент позволяет создавать окна приложения и добавлять на него нужные элементы. Чтобы окно соответствовало макету необходимо использовать следующие элементы управления:

1) Label;
2) TextBox;
3) RichTextBox;
4) Button;
5) CheckBox.

Каждый из этих элементов является классом, обладающий своими свойствами, событиями и методами.

Окно для вывода системных сообщений будет служить для отладки приложения, в конечной версии приложения оно будет выполнять другие функции. На этапе отладки в нем будут выводиться сообщения об ошибках и предупреждения по мере их возникновения. Далее, после окончания процесса

отладки, в нем будут появляться сообщения об окончании обучения сети, сообщения о завершении восстановления изображения.

Возможна ситуация, когда пользователю нужно восстановить изображение по уже обученной сети, либо если пользователю нужно подать другую картинку для обучения уже ранее инициализированной сети. Для этих целей в макете предусмотрен checkbox элемент, который выполняет роль следующей условной конструкции – если возвращаемое значение этого элемента перед началом процесса обучения сети эквивалентно логическому значению «true», то вместо параметров, вводимых в блоке «Настройки карты», будут использоваться параметры, полученные из XML файла, путь к которому должен быть указан пользователем заранее. В ходе написания кода приложения был разработан интерфейс, представленный на рисунке 4.

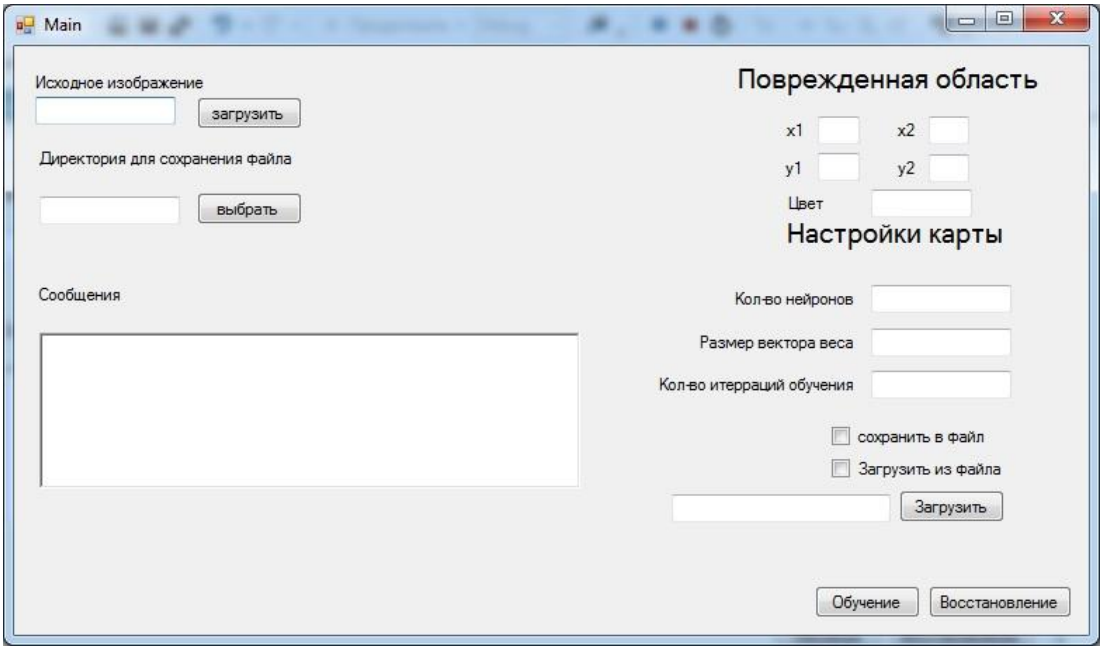

Рисунок 4 – Основное окно приложения

2.9 Классы C#, необходимые для реализации приложения

Язык C# обладает встроенным набором классов для работы с изображениями. Основной класс, который будет использоваться для работы с изображениями – это класс Bitmap. Он позволяет инкапсулировать точечный рисунок GDI+, состоящий из пикселей графического изображения и атрибутов рисунка [4]. Другими словами, он позволяет манипулировать изображением

или получать какие-либо его свойства. Например, можно получить высоту и ширину изображения в пикселях, получить горизонтальное и вертикальное разрешение изображения, манипулировать палитрой цветов, манипулировать пикселями изображения (менять цвет пикселя, получить его координаты и так далее), а так же получить дополнительную информацию об изображении (формат исходной картинки, название, заголовки и так далее). Потребуются следующие свойства класса Bitmap:

1) height;
2) width.

Первое свойство получает высоту изображения в пикселях, второе – ширину. Эти свойства понадобятся при создании экземпляров классов, которые будут написаны для работы с векторами входных данных, а так же при сохранении восстановленного изображения.

Из всей совокупности методов рассматриваемого класса следует выделить основные – это методы SetPixel(), GetPixel(), Save(). Первый метод задает цвет определенного пикселя на изображении, входные параметры этого метода – это координаты нужного пикселя, а так же цвет, который будет записан в него в виде объекта класса Color. Второй метод возвращает цвет пикселя в виде объекта класса Color по заданным координатам, которые передаются в метод в качестве параметров. Последний метод сохраняет объект Bitmap как цифровой графический файл в любом расширении, допустимом в языке C#. Входные параметры этого метода – это название файла и путь к директории, в которую необходимо сохранить файл.

Так же в приложении будет использован класс Color, позволяющий работать с цветами пикселей. Он включает в себя возможность преобразования цветов из различных форматов (например, из формата «rgb») в свой внутренний формат, который используется классом Bitmap для работы с цветами пикселей.

2.10 Архитектура основных классов приложения

В ходе работы были выделены основные объекты приложения – это нейрон, карта нейронов, вектор входных данных. Для каждого объекта в

соответствии с парадигмой объектно-ориентированного программирования, на котором построен язык C#, были реализованные следующие классы – Neuron, NeuronMap, ImgVector.

Класс Neuron является реализацией базового элемента карты Кохонена – нейрона. Он включает в себя следующие свойства:

1) weight;
2) coords;
3) size.

Первое свойство – это вектор веса нейрона, каждый элемент которого имеет тип double. Свойство coords содержит в себе вектор, состоящий из двух элементов типа int, и содержит координаты местоположения нейрона на карте. Последнее свойство является размерностью нейрона.

В данном классе были реализованы следующие методы:

1) конструктор с параметрами;
2) функция getX();
3) функция getY();
4) функция neuronVectorDistance();
5) функция initWeightWithVector();
6) функция drawNeuron().

Конструктор с параметрами является необходимым в данной ситуации, так как свойства класса являются динамическими (за исключением свойства size). Таким образом при инициализации нового объекта класса каждому полю класса будет присваиваться значение, что поможет избежать ошибок, связанных с памятью. Функции getX() и getY() возвращают соответствующие координаты нейрона на карте в виде переменной типа int. Четвертая в вышеприведенном списке функция возвращает расстояние от вектора входных значений до данного нейрона. Следующая функция необходима на этапе инициализации карты. Она копирует значения вектора входных данных в вектор веса нейрона. Данная операция выполняется только во время инициализации, и только для первых десяти нейронов, которые имеют

равномерное распределение по всей карте. Последняя функция используется для калибровки и настройки сети. Она выводит нейрон на карту, которую можно просмотреть и проверить корректность процесса обучения.

Класс ImgVector позволяет сгенерировать вектор входных значений из выбранного пользователем цифрового изображения для обучения. Он содержит в себе следующие поля:

1) size;
2) vector;
3) brokenColor;
4) valid;
5) message.

Поле size хранит размерность вектора и имеет тип int. Поле vector является массивом, элементы которого есть значения соответствующих пикселей. Третье свойство содержит цвет поврежденных пикселей в формате Argb. Свойство valid имеет логический тип и соответственно принимает два значения – true либо false. Если данный вектор входных значений не содержит поврежденных пикселей, то поле принимает значение false, в противном случае – значение true. Последнее поле предназначено для ведения лога. В него будут записываться результаты действий во время операции генерации вектора входных значений, а именно – имеет ли вектор поврежденные пиксели, номер итерации, во время которого был сгенерирован этот вектор. Эта информация поможет понять, насколько корректно реализован функционал выявления поврежденных пикселей.

Методы класса ImgVector:

1) конструктор с параметрами;
2) isValid().

Конструктор принимает следующие аргументы:

1) исходное изображение в формате Bitmap;
2) координаты исходной точки на изображении;
3) размерность вектора входных значений;

4) цвет поврежденного пикселя в формате Argb.

Задача конструктора – создать из исходного изображения вектор входных значений во внутреннем формате программы. Для этого берется исходная точка, и по формуле (9) генерируется вектор входных значений той размерности, которая была задана ранее в настройках.

Функция isValid() данного класса проверяет, есть ли в окрестности исходной точки поврежденные пиксели. Если такие пиксели находятся, то поле valid принимает значение true и в поле message записывается соответствующее сообщение. Таким образом, уже на этапе создания объекта класса ImgVector можно проверить, подходит ли он для обучения нейросети.

Вышеописанные классы являются базовыми и описывают модель нейрона и модель вектора входных значений. Для объединения этих классов в общую модель карты был реализован класс NeuronMap. Он содержит следующие поля:

1) map;
2) neuronCount;
3) neuronSize;
4) color;
5) mapWidth;
6) mapHeight;
7) iterationCount.

Первое поле является двумерным массивом, каждый элемент которого – экземпляр класса Neuron. Размерность данного массива задается при помощи полей mapWidth и mapHeight, которые имеют тип int. Поле neuronSize хранит значение размерности вектора веса нейрона, а в поле neuronCount записывается количество нейронов на карте. Количество итераций алгоритма обучения записывается в поле itertionCount. Свойство color хранит цвет, которым обозначается поврежденный пиксель.

В данном классе были реализованы следующие методы:

1) neuronDistance();
2) neuronSimilarity();
3) learnMap();
4) initializeMap();
5) findBmu();
6) generateCoords();
7) showMap().

Первый метод вычисляет меру соседства двух нейронов на карте используя функцию (9), а сами нейроны передаются в функцию в качестве параметров.

Функция neuronSimilarity() вычисляет меру сходства двух нейронов по формуле (8) и возвращает значение типа double. Нейроны, для которых необходимо вычислить меру сходства, передаются в функцию в качестве параметров.

Методы initializeMap() и learnMap() реализуют процесс обучения. Первый метод необходим для равномерного распределения BMU на карте и инициализации их векторов веса, в зависимости от векторов входных данных на первых десяти итерациях процесса обучения. Метод learnMap() производит дальнейшее обучение карты в течение оставшихся итераций. Оба этих метода имеют лишь один входной параметр – это исходное изображение в виде объекта класса Bitmap.

Так же в рассматриваемом классе были реализованы несколько вспомогательных методов, таких как: findBMU(), generateCoords(), showMap(). Первая функция возвращает координаты BMU-элемента карты Кохонена для вектора входных данных на каждой итерации. Вторая функция возвращает координаты случайно выбранной точки на изображении на этапе генерации вектора входных данных. Последний метод необходим для тестирования и отладки приложения – он выводит в окно приложения картинку, которая отображает состояние обученной сети. По этой картинке можно оценить

качество процесса обучения и корректировать входные параметры для его улучшения.

2.11 Руководство пользователя

После запуска программы появляется главное окно, показанное на рисунке 4.

2.11.1 Обучение карты

Для начала обучения нейронной сети необходимо задать настройки карты, а именно – заполнить поля «Кол-во нейронов», «Размер вектора веса», «Кол-во итераций обучения». Данные поля являются обязательными, если перед началом процесса обучения или восстановления они будут пустыми, то программа отреагирует как показано на рисунке 5.

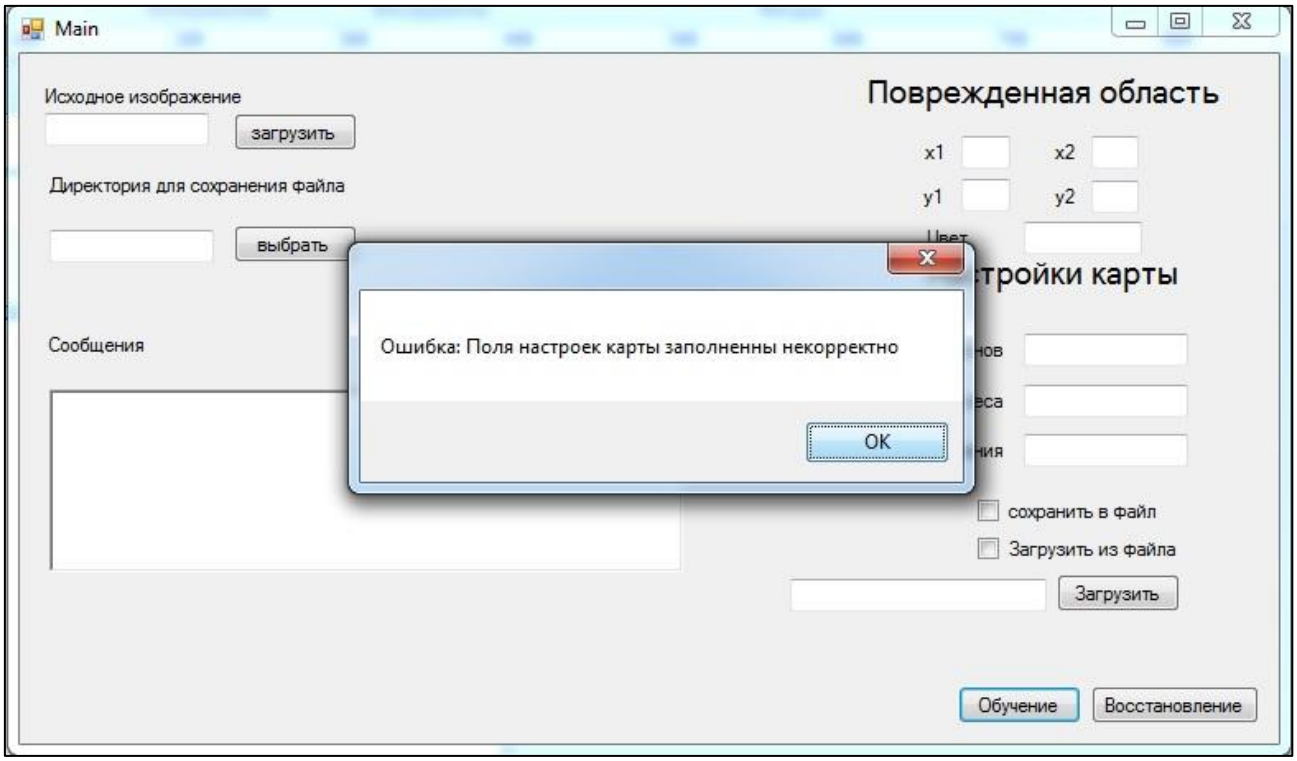

Рисунок 5 – Ошибка заполнения настроек карты

Далее необходимо выбрать изображение для обучения карты путем нажатия на кнопку «загрузить», расположенную в левом верхнем углу. После нажатия откроется окно диалога выбора файла, которое показано на рисунке 6.

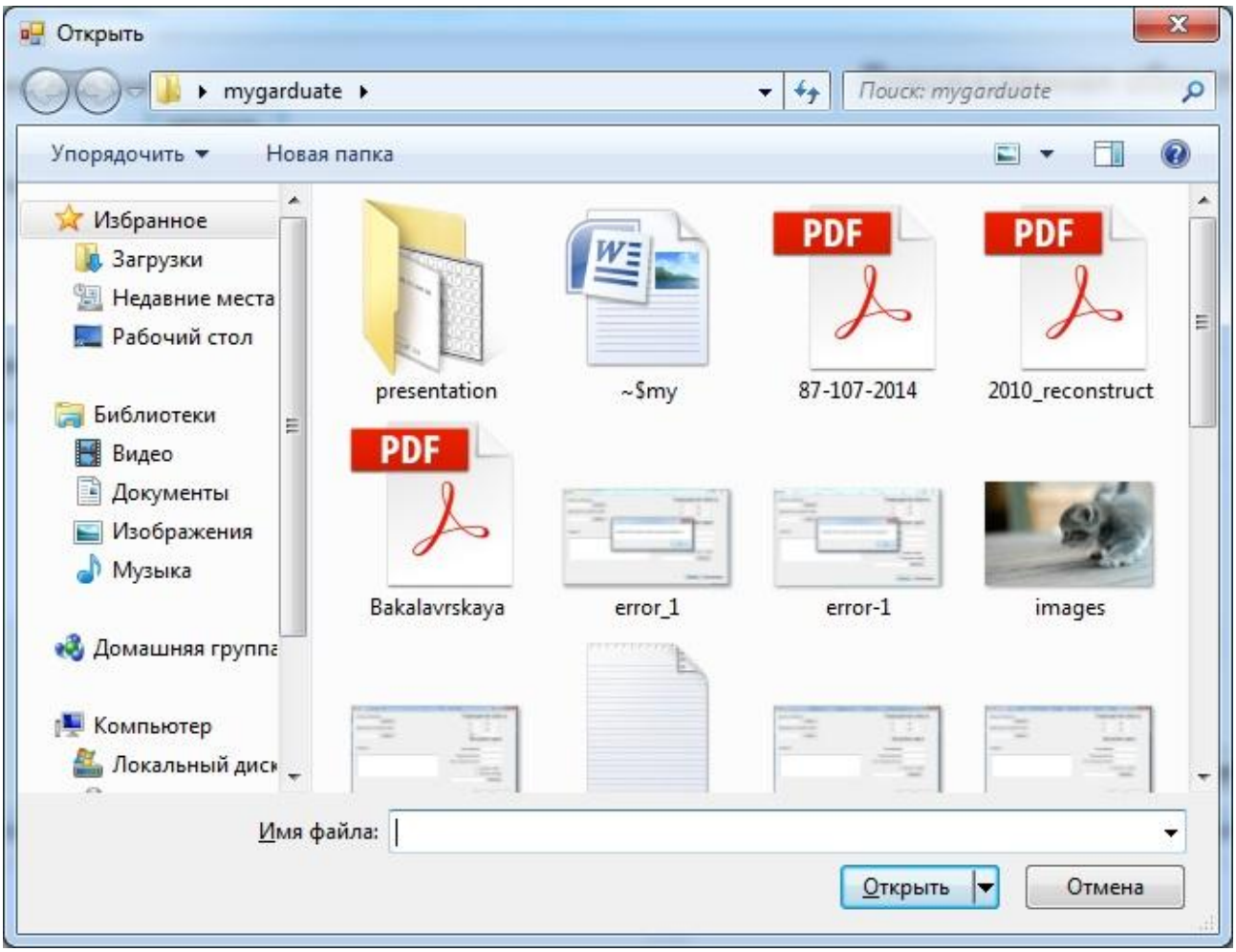

Рисунок 6 – Окно диалога выбора файла

Чтобы обучить карту необходимо нажать кнопку «Обучение». Если необходимо сохранить состояние карты в файл, то перед началом обучения надо поставить отметку в поле «сохранить в файл», которое расположено в блоке настроек карты, а так же задать путь к директории путем нажатия кнопки «выбрать».

После окончания процесса обучения появится окно, показанное на рисунке 7.

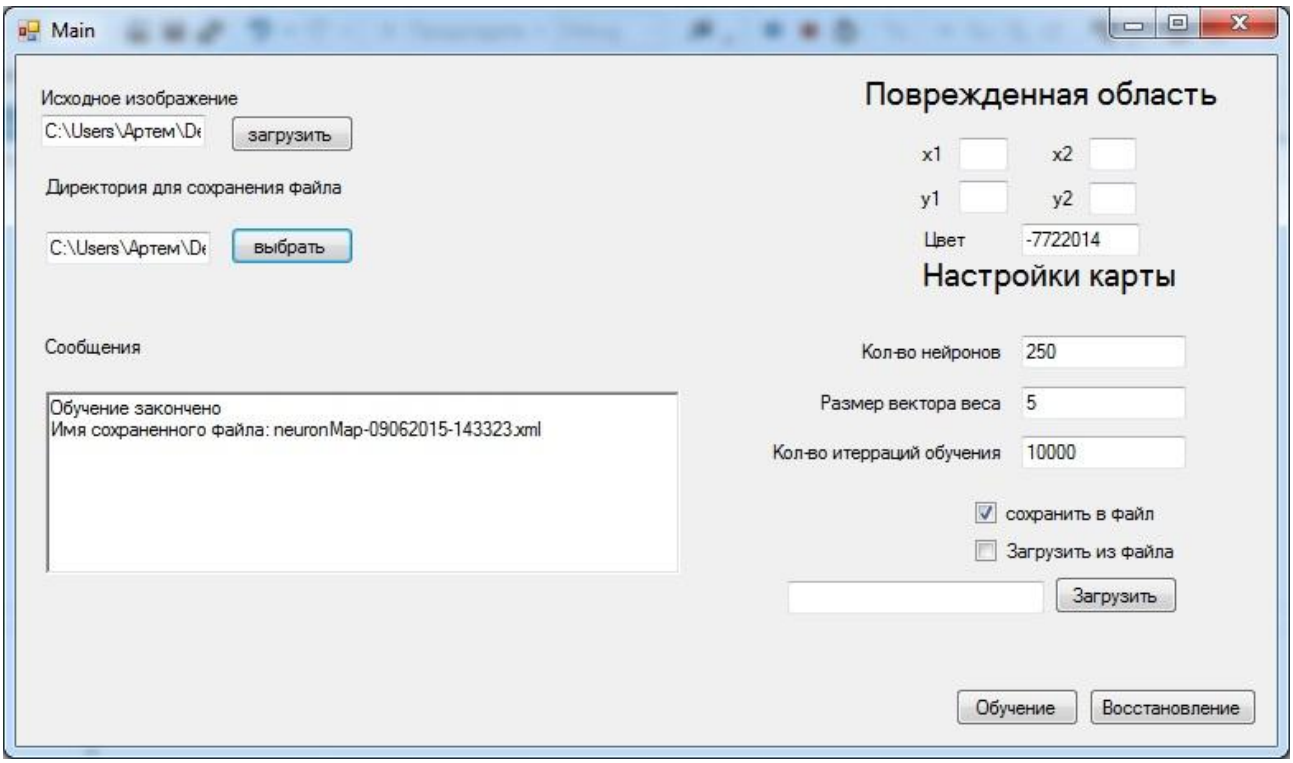

Рисунок 7 – Окно приложения после окончания обучения

2.11.2 Восстановление цифрового изображения

Восстановление картинки является второй основной функцией программы. Для того чтобы начать процесс восстановления необходимо заполнить поля блока «Поврежденная область», где указываются координаты прямоугольника, внутри которого находятся поврежденные пиксели, а так же цвет, с помощью которого проводилась имитация повреждений. Следует отметить, что поля координат не являются обязательными, и если они не будут заполнены, то программа сама выделит нужную область, однако это займет немного больше времени, которое зависит от размеров изображения. Тем не менее, если повреждения цифрового изображения являются аддитивным шумом, то координаты области повреждений необходимо оставить незаполненными. Поле цвета является обязательным для заполнения, на его пустоту программа выдаст ошибку как показано на рисунке 5.

Следующим шагом является выбор обученной нейронной сети для восстановления. Если перед началом обучения проводилось обучение сети, то эта сеть и будет использоваться для восстановления, в противном случае необходимо указать расположение файла, который хранит состояние какой-

либо ранее обученной сети путем нажатия кнопки «Загрузить», которая находится под блоком «Настройки карты». После нажатия на эту кнопку откроется окно диалога выбора файла, которое показано на рисунке 6.

Если файл имеет формат отличный от XML, или внутренняя структура XML-файла является неправильной, то программа выдаст ошибку, которая показана на рисунке 8.

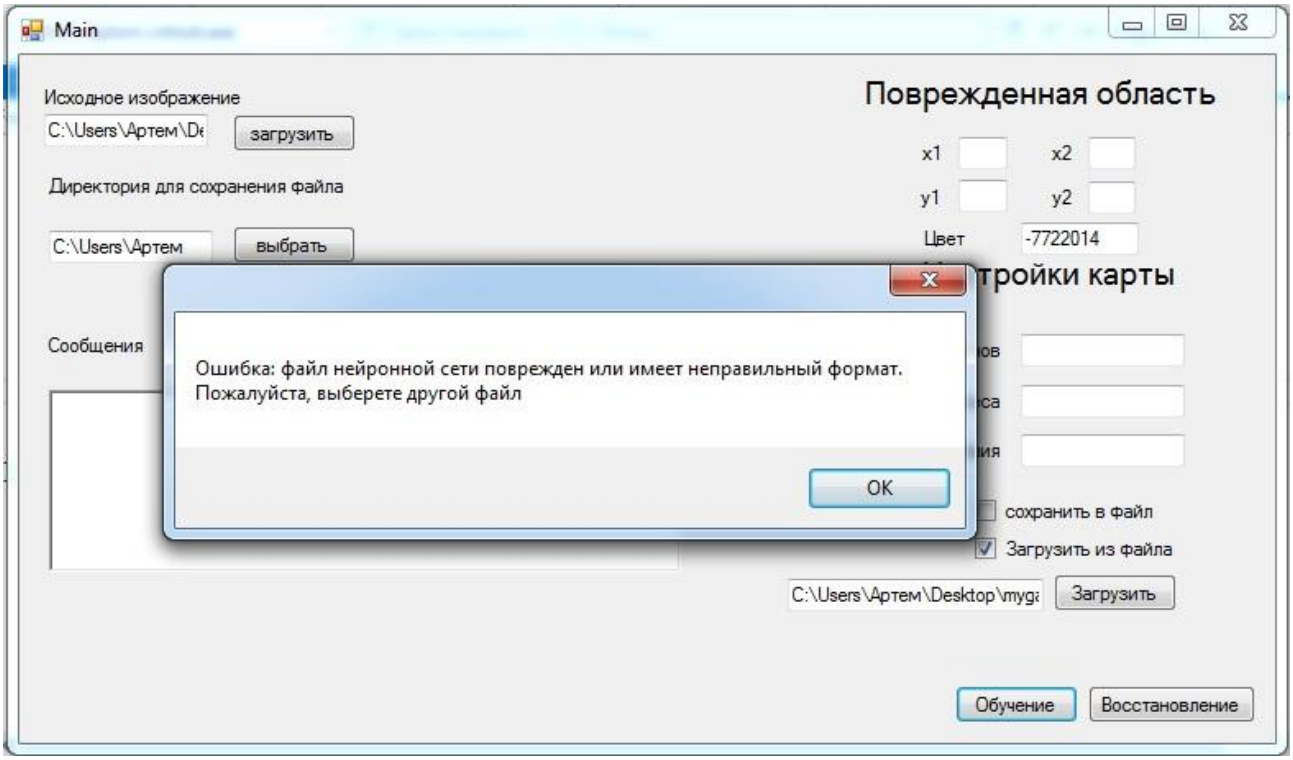

Рисунок 8 – Ошибка загрузки XML-файла

Чтобы указать директорию для сохранения восстановленного изображения, необходимо нажать кнопку «выбрать» в левом верхнем углу основного окна приложения. После нажатия откроется окно диалога выбора директории, показанное на рисунке 8.

При нажатии кнопки «Восстановление» начнется процесс восстановления изображения, по окончании которого программа показывает окно с описанием результата выполнения операции – было ли изображение восстановлено успешно (как показано на рисунке 9), или произошла ошибка при выполнении операции. Возможные ошибки при восстановлении – это несоответствие цвета поврежденных пикселей на изображении и в настроенной карте, либо системная ошибка.

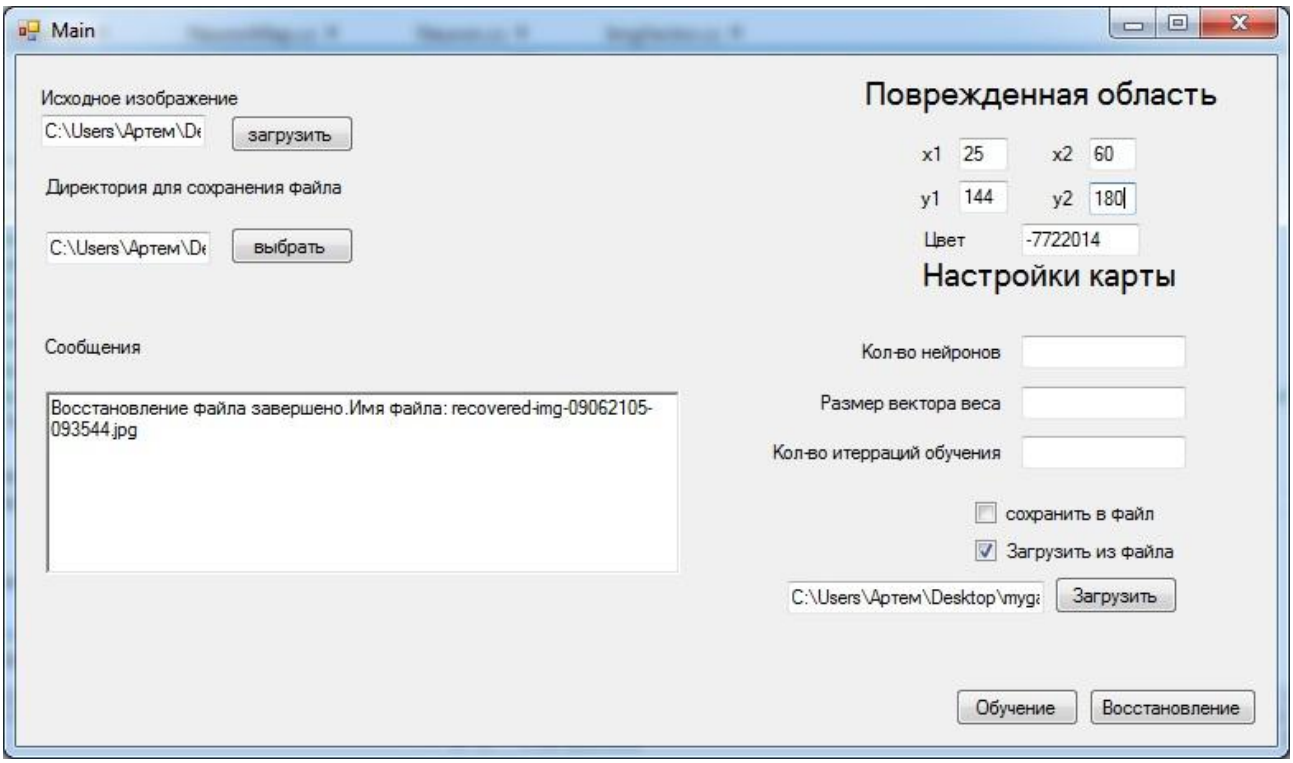

Рисунок 9 – Окно программы после завершения восстановления изображения

2.12 Анализ результатов

Во время тестирования программы было произведено большое количество unit-тестов для обнаружения и устранения программных ошибок.

Далее было проведено большое количество экспериментов для оценки качества восстановления изображений и для поиска оптимальных настроек нейронной сети.

Для того чтобы в полной мере оценить потенциал разработанного приложения по восстановлению изображений эксперименты были проведены следующим образом – с помощью исходного изображения сеть обучалась, после чего на изображении имитировались повреждения различного характера, и ранее обученной сетью картинка восстанавливалась. Далее в работе приведены результаты нескольких экспериментов, которые отображены на рисунках 10 – 18.

Рисунок 10 – Поврежденное изображение для первого эксперимента

Рисунок 11 – Результат первого эксперимента

Рисунок 12 – Исходное изображение для первого эксперимента

Рисунок 13 – Поврежденное изображение для второго эксперимента

Рисунок 14 – Результат второго эксперимента

Рисунок 15 – Исходное изображение для второго эксперимента

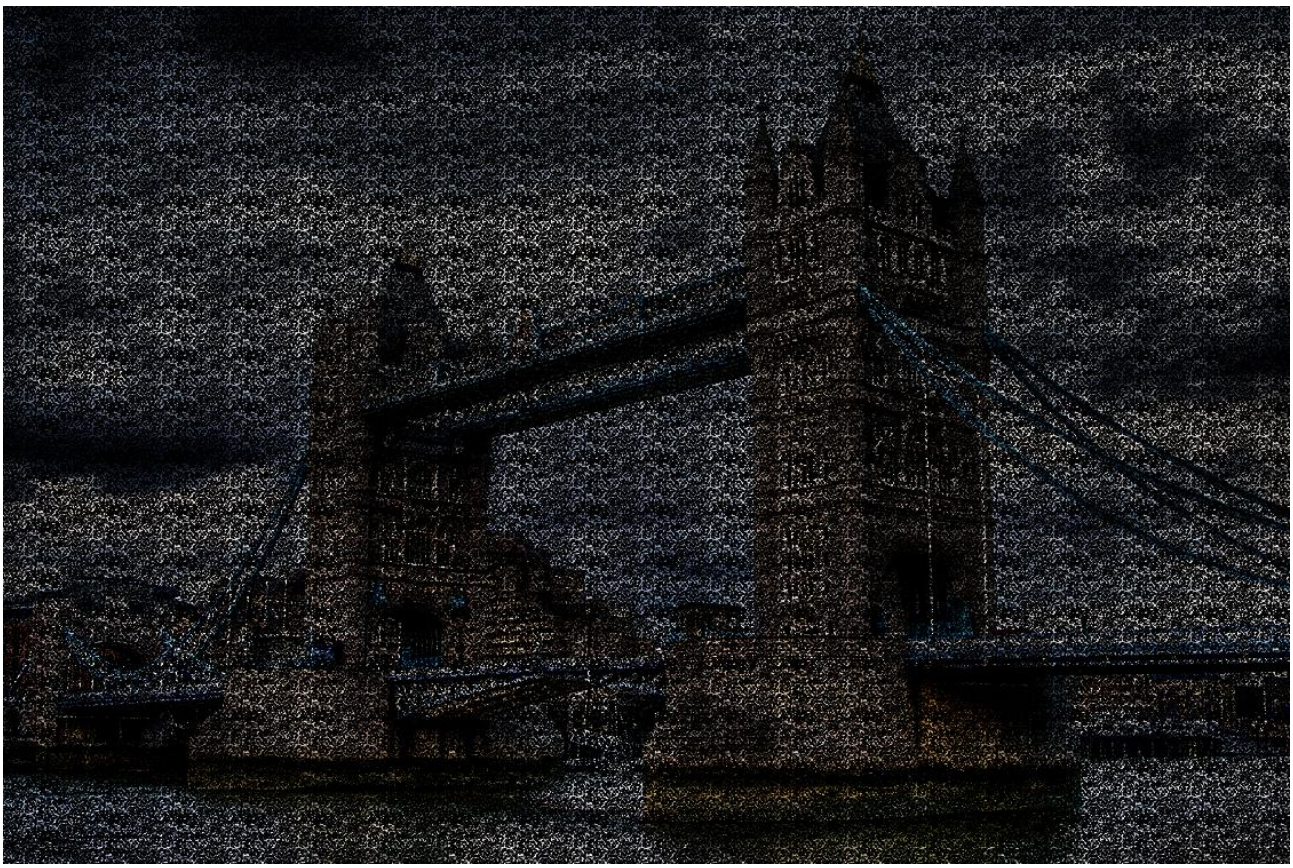

Рисунок 16 – Поврежденное изображение для третьего эксперимента

Рисунок 17 – Результат третьего эксперимента

Рисунок 18 – Исходное изображение для третьего эксперимента

Для каждого эксперимента были выбраны изображения с разной степенью повреждения. Первая поврежденная картинка имеет небольшую область повреждения прямоугольной формы, как показано на рисунке 10. Из рисунка 11 видно, что такие изображения при качественном обучении сети восстанавливаются хорошо. На рисунке 13 показано изображение, которое зашумлено слабым аддитивным шумом. Такой тип повреждения восстанавливается хуже, чем в первом случае – видимые искажения остаются, но они не сильно заметны, что проиллюстрировано на рисунке 14. В последнем эксперименте использовалась картинка, на которой повреждено более семидесяти процентов пикселей, как показано на рисунке 16. В данном случае восстановленное изображение имеет явные сильные искажения, по сравнению с оригиналом, что продемонстрировано на рисунках 17 и 18.

В ходе экспериментов было установлено, что оптимальное количество итераций для обучения должно быть более тысячи, но менее десяти тысяч, так как после этого числа качество восстановленных изображений увеличивается

очень мало, но количество времени, потраченного на обучение сети, существенно растет. Так же была обнаружена явная зависимость качества восстановленного изображения от размеров вектора веса нейрона и от количества нейронов в сети – чем меньше размерность вектора веса и больше количество нейронов на карте, тем более резким получается восстановленное изображение.

Для оценки качества восстановления был использован коэффициент *H*, который вычисляется по формуле (27):

$$H = \frac{\sum_{n=1}^{K} |p_n - q_n|}{K}, \tag{27}$$

где p_n – исходный цвет поврежденного пикселя, q_n – цвет соответствующего восстановленного пикселя, K – общее количество восстановленных пикселей. Все данные по приведенным ранее экспериментам представлены в таблице 1.

Таблица 1 – Результаты экспериментов

№ эксперимента	1	2	3
Ширина карты в нейронах	500	500	500
Высота карты в нейронах	500	500	500
Количество итераций обучения	10000	10000	10000
Размерность вектора веса нейрона	5	5	5
Количество поврежденных пикселей, %	5	18	72
H	4	56	816

Как видно из таблицы 1, чем меньше поврежденных пикселей, тем выше качество восстановленного изображения.

Таким образом, в ходе данной работы был сделан вывод, что карты Кохонена обладают высоким потенциалом как средство восстановления поврежденных изображений.

Заключение

В ходе выполнения работы были проведены теоретические исследования самоорганизующихся карт Кохонена. Были построены модель нейрона и модель карты нейросети Кохонена для восстановления поврежденных изображений, а именно – были выбраны функции меры сходства и меры соседства нейронов для работы с цифровыми изображениями, были адаптированы алгоритмы обучения сети и восстановления изображений. Используя построенные модели было разработано приложение на языке C#, которое позволяет восстанавливать поврежденные изображения при помощи карты Кохонена. Для реализации приложения была построена модель внутреннего формата вектора входных значений, который используется во время обучения сети.

Был произведен ряд экспериментов, целью которых являлся поиск оптимальных настроек для нейронной карты, в результате было выяснено, что для качественного обучения сети размер вектора веса нейрона должен принимать значения от трех до семи в зависимости от области повреждений изображения, которое необходимо восстановить.

Далее были проведены эксперименты по восстановлению изображений с различным количеством поврежденных пикселей. Для оценки качества был выбран критерий, который вычисляет среднее значение разностей между поврежденными и соответствующими им исходными пикселями. Другими словами, он показывает, насколько восстановленная часть изображения соответствует исходной. В результате была выделена явная зависимость качества восстановления от количества поврежденных пикселей и качества обучения сети.

В результате работы был сделан вывод, что метод восстановления поврежденных изображений при помощи карт Кохонена качественно восстанавливает небольшие локальные повреждения и слабые зашумления.

Список использованных источников

1. Кохонен Т. Самоорганизующиеся карты [Текст] / Т. Кохонен. – М. Бином: Лаборатория знаний, 2008 – 655 с.

2. Интуит [Электронный ресурс] // Карты Кохонена. – Режим доступа: http://www.intuit.ru/studies/courses/6/6/lecture/180?page=3.

3. Лопес – Мартинес Х. Л. Восстановление изображений с помощью микросканирующей изображающей системы [Текст] / В.И. Кобер, В. Н. Корноухов // Информационные процессы. – 2014. – Т. XIV, № 1. – С. 87 – 107.

4. Грузман И.С. Цифровая обработка изображений в информационных системах: учебное пособие [Текст] / Грузман И.С., Киричук В.С., Косых В.П., Перетягин Г.И., Спектор А.А. – Новосибирск: Изд-во НГТУ, 2008. – 352 с.

5. BaseGroup Labs [Электронный ресурс] // Самоорганизующиеся карты Кохонена – Математический аппарат. – Режим доступа: http://www.basegroup.ru/library/analysis/clusterization/som/.

6. Гагарина Л. Г. Технология разработки программного обеспечения: учеб. пособие [Текст] / Л. Г. Гагарина, Е. В. Кокарева, Б. Д. Виснадул. – М.: Форум, 2008. – 300 с.

7. Академик [Электронный ресурс] // Самоорганизующиеся карты Кохонена. – Режим доступа: http://dic.academic.ru/dic.nsf/ruwiki/1136560.

8. Академик [Электронный ресурс] // Искусственная нейронная сеть. – Режим доступа: http://dic.academic.ru/dic.nsf/ruwiki/13889.

9. Википедия [Электронный ресурс] // Искусственная нейронная сеть. – Режим доступа: https://ru.wikipedia.org/wiki/%D0%98%D1%81%D0%BA%D1%83%D1%81%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D0%B0%D1%8F_%D0%BD%D0%B5%D0%B9%D1%80%D0%BE%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D0%B5%D1%82%D1%8C.

10. Хабрахабр [Электронный ресурс] // Восстановление при помощи нейросетей. – Режим доступа: http://habrahabr.ru/post/120473/.

11. Microsoft Developer Network [Электронный ресурс] // Создание XML-деревьев в C# (LINQ to XML). – Режим доступа: https://msdn.microsoft.com/ru-ru/library/bb387089.aspx.

12. Microsoft Developer Network [Электронный ресурс] // Bitmap – класс.– Режим доступа: https://msdn.microsoft.com/ru-ru/library/system.drawing.bitmap(v=vs.110).aspx.

13. Microsoft Developer Network [Электронный ресурс] // XML-документы и данные. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/2bcctyt8(v=vs.110).aspx .

14. Microsoft Developer Network [Электронный ресурс] // Color – структура. – Режим доступа: https://msdn.microsoft.com/ru-ru/library/system.drawing.color(v=vs.110).aspx

15. Шилдт, Г. C# 4.0: полное руководство [Текст] / Г. Шилдт. – М. : ООО «И. Д. Вильямс», 2011. – 1054 с.

Приложение А

(справочное)

Файл состояния нейронной сети


```
<xml>
<map>
<iterrations>10000</iterrations>
<width>9</width>
<height>9</height>
</map>
<neurons>
<neuron>
<x>0</x>
<y>0</y>
<weightVec>
<value>9157897</value>
<value>8507781</value>
<value>6129516</value>
</weightVec>
<x>0</x>
<y>1</y>
<weightVec>
<value>2695465</value>
<value>6243225</value>
<value>8620941</value>
</weightVec>
<x>0</x>
<y>2</y>
<weightVec>
<value>1026367</value>
<value>2764678</value>
<value>9850036</value>
</weightVec>
<x>0</x>
<y>3</y>
<weightVec>
<value>8575897</value>
<value>5679077</value>
```

<value>4222564</value>
</weightVec>
<x>0</x>
<y>4</y>
<weightVec>
<value>2822082</value>
<value>1341400</value>
<value>9681396</value>
</weightVec>
<x>0</x>
<y>5</y>
<weightVec>
<value>6787872</value>
<value>8503112</value>
<value>2386199</value>
</weightVec>
<x>0</x>
<y>6</y>
<weightVec>
<value>7127075</value>
<value>5679351</value>
<value>9736877</value>
</weightVec>
<x>0</x>
<y>7</y>
<weightVec>
<value>4679046</value>
<value>5609863</value>
<value>9615753</value>
</weightVec>
<x>0</x>
<y>8</y>
<weightVec>
<value>2867126</value>
<value>5688690</value>
<value>8223510</value>
</weightVec>
<x>1</x>
<y>0</y>

<weightVec>
<value>4815826</value>
<value>6737609</value>
<value>7383880</value>
</weightVec>
<x>1</x>
<y>1</y>
<weightVec>
<value>4561767</value>
<value>8998321</value>
<value>8692077</value>
</weightVec>
<x>1</x>
<y>2</y>
<weightVec>
<value>7233367</value>
<value>6718383</value>
<value>8381988</value>
</weightVec>
<x>1</x>
<y>3</y>
<weightVec>
<value>9574279</value>
<value>7205902</value>
<value>4287109</value>
</weightVec>
<x>1</x>
<y>4</y>
<weightVec>
<value>2685577</value>
<value>5727966</value>
<value>4770233</value>
</weightVec>
<x>1</x>
<y>5</y>
<weightVec>
<value>9361694</value>
<value>5127838</value>
<value>6996887</value>

</weightVec>
<x>1</x>
<y>6</y>
<weightVec>
<value>1895111</value>
<value>2535888</value>
<value>6427520</value>
</weightVec>
<x>1</x>
<y>7</y>
<weightVec>
<value>9724792</value>
<value>5049285</value>
<value>4903442</value>
</weightVec>
<x>1</x>
<y>8</y>
<weightVec>
<value>1803375</value>
<value>5755432</value>
<value>7201507</value>
</weightVec>
<x>2</x>
<y>0</y>
<weightVec>
<value>6058105</value>
<value>8974151</value>
<value>9432556</value>
</weightVec>
<x>2</x>
<y>1</y>
<weightVec>
<value>5820526</value>
<value>9093627</value>
<value>5158599</value>
</weightVec>
<x>2</x>
<y>2</y>
<weightVec>

<value>3599914</value>
<value>2875091</value>
<value>3603759</value>
</weightVec>
<x>2</x>
<y>3</y>
<weightVec>
<value>8075469</value>
<value>2601257</value>
<value>4833953</value>
</weightVec>
<x>2</x>
<y>4</y>
<weightVec>
<value>1682250</value>
<value>3943511</value>
<value>7280334</value>
</weightVec>
<x>2</x>
<y>5</y>
<weightVec>
<value>4665863</value>
<value>1922851</value>
<value>1481475</value>
</weightVec>
<x>2</x>
<y>6</y>
<weightVec>
<value>4980895</value>
<value>8839569</value>
<value>7419311</value>
</weightVec>
<x>2</x>
<y>7</y>
<weightVec>
<value>4908111</value>
<value>5546691</value>
<value>1323822</value>
</weightVec>

<x>2</x>
<y>8</y>
<weightVec>
<value>7765380</value>
<value>4942169</value>
<value>5321472</value>
</weightVec>
<x>3</x>
<y>0</y>
<weightVec>
<value>6587371</value>
<value>6054809</value>
<value>5802398</value>
</weightVec>
<x>3</x>
<y>1</y>
<weightVec>
<value>5148986</value>
<value>8598968</value>
<value>9881347</value>
</weightVec>
<x>3</x>
<y>2</y>
<weightVec>
<value>7207550</value>
<value>1372985</value>
<value>4827362</value>
</weightVec>
<x>3</x>
<y>3</y>
<weightVec>
<value>4338745</value>
<value>4376373</value>
<value>3837219</value>
</weightVec>
<x>3</x>
<y>4</y>
<weightVec>
<value>6241302</value>

<value>6020751</value>
<value>6027618</value>
</weightVec>
<x>3</x>
<y>5</y>
<weightVec>
<value>8885437</value>
<value>1309539</value>
<value>9021118</value>
</weightVec>
<x>3</x>
<y>6</y>
<weightVec>
<value>7380035</value>
<value>8361389</value>
<value>1000823</value>
</weightVec>
<x>3</x>
<y>7</y>
<weightVec>
<value>2933593</value>
<value>8152374</value>
<value>7608825</value>
</weightVec>
<x>3</x>
<y>8</y>
<weightVec>
<value>2783905</value>
<value>8851928</value>
<value>3563385</value>
</weightVec>
<x>4</x>
<y>0</y>
<weightVec>
<value>7471496</value>
<value>8627532</value>
<value>7369873</value>
</weightVec>
<x>4</x>

<y>1</y>
<weightVec>
<value>2331817</value>
<value>4293151</value>
<value>7000457</value>
</weightVec>
<x>4</x>
<y>2</y>
<weightVec>
<value>1581176</value>
<value>8676422</value>
<value>7917541</value>
</weightVec>
<x>4</x>
<y>3</y>
<weightVec>
<value>8871429</value>
<value>8079589</value>
<value>4315948</value>
</weightVec>
<x>4</x>
<y>4</y>
<weightVec>
<value>5688415</value>
<value>2709197</value>
<value>2958862</value>
</weightVec>
<x>4</x>
<y>5</y>
<weightVec>
<value>2469146</value>
<value>7449523</value>
<value>8482513</value>
</weightVec>
<x>4</x>
<y>6</y>
<weightVec>
<value>2812744</value>
<value>2854766</value>

<value>9544616</value>
</weightVec>
<x>4</x>
<y>7</y>
<weightVec>
<value>5660125</value>
<value>1734985</value>
<value>9691558</value>
</weightVec>
<x>4</x>
<y>8</y>
<weightVec>
<value>3817443</value>
<value>5210784</value>
<value>7319335</value>
</weightVec>
<x>5</x>
<y>0</y>
<weightVec>
<value>4698272</value>
<value>4611755</value>
<value>4603240</value>
</weightVec>
<x>5</x>
<y>1</y>
<weightVec>
<value>4659545</value>
<value>1093658</value>
<value>3771301</value>
</weightVec>
<x>5</x>
<y>2</y>
<weightVec>
<value>5806243</value>
<value>1349365</value>
<value>7596466</value>
</weightVec>
<x>5</x>
<y>3</y>

<weightVec>
<value>6639831</value>
<value>6288543</value>
<value>9482543</value>
</weightVec>
<x>5</x>
<y>4</y>
<weightVec>
<value>1425445</value>
<value>5061096</value>
<value>8018890</value>
</weightVec>
<x>5</x>
<y>5</y>
<weightVec>
<value>9652832</value>
<value>9299346</value>
<value>4378845</value>
</weightVec>
<x>5</x>
<y>6</y>
<weightVec>
<value>8466033</value>
<value>4953979</value>
<value>8436920</value>
</weightVec>
<x>5</x>
<y>7</y>
<weightVec>
<value>5436828</value>
<value>9598724</value>
<value>2979736</value>
</weightVec>
<x>5</x>
<y>8</y>
<weightVec>
<value>7556915</value>
<value>4578796</value>
<value>8885711</value>

</weightVec>
<x>6</x>
<y>0</y>
<weightVec>
<value>6823852</value>
<value>1878082</value>
<value>2648498</value>
</weightVec>
<x>6</x>
<y>1</y>
<weightVec>
<value>8295745</value>
<value>6080078</value>
<value>9119171</value>
</weightVec>
<x>6</x>
<y>2</y>
<weightVec>
<value>4989685</value>
<value>5297576</value>
<value>3842163</value>
</weightVec>
<x>6</x>
<y>3</y>
<weightVec>
<value>6498931</value>
<value>3446105</value>
<value>1859954</value>
</weightVec>
<x>6</x>
<y>4</y>
<weightVec>
<value>7703857</value>
<value>2736114</value>
<value>3361511</value>
</weightVec>
<x>6</x>
<y>5</y>
<weightVec>

<value>4451629</value>
<value>2758911</value>
<value>2049468</value>
</weightVec>
<x>6</x>
<y>6</y>
<weightVec>
<value>5579650</value>
<value>6041351</value>
<value>5601074</value>
</weightVec>
<x>6</x>
<y>7</y>
<weightVec>
<value>4157745</value>
<value>6444274</value>
<value>4097869</value>
</weightVec>
<x>6</x>
<y>8</y>
<weightVec>
<value>1324096</value>
<value>4279693</value>
<value>6799133</value>
</weightVec>
<x>7</x>
<y>0</y>
<weightVec>
<value>9589935</value>
<value>6521728</value>
<value>2134063</value>
</weightVec>
<x>7</x>
<y>1</y>
<weightVec>
<value>2987976</value>
<value>1986297</value>
<value>6287719</value>
</weightVec>

<x>7</x>
<y>2</y>
<weightVec>
<value>1939605</value>
<value>4854553</value>
<value>1255706</value>
</weightVec>
<x>7</x>
<y>3</y>
<weightVec>
<value>8215820</value>
<value>3415069</value>
<value>8742614</value>
</weightVec>
<x>7</x>
<y>4</y>
<weightVec>
<value>4866912</value>
<value>6399780</value>
<value>1779205</value>
</weightVec>
<x>7</x>
<y>5</y>
<weightVec>
<value>6495910</value>
<value>5788665</value>
<value>8433349</value>
</weightVec>
<x>7</x>
<y>6</y>
<weightVec>
<value>5750762</value>
<value>3458190</value>
<value>1489715</value>
</weightVec>
<x>7</x>
<y>7</y>
<weightVec>
<value>8410278</value>

<value>1548492</value>
<value>9473205</value>
</weightVec>
<x>7</x>
<y>8</y>
<weightVec>
<value>2938812</value>
<value>4924316</value>
<value>6891143</value>
</weightVec>
<x>8</x>
<y>0</y>
<weightVec>
<value>2884704</value>
<value>7604705</value>
<value>1069213</value>
</weightVec>
<x>8</x>
<y>1</y>
<weightVec>
<value>7997467</value>
<value>2536437</value>
<value>8456146</value>
</weightVec>
<x>8</x>
<y>2</y>
<weightVec>
<value>4438720</value>
<value>4586212</value>
<value>4772155</value>
</weightVec>
<x>8</x>
<y>3</y>
<weightVec>
<value>2961883</value>
<value>9126586</value>
<value>9876129</value>
</weightVec>
<x>8</x>

<y>4</y>
<weightVec>
<value>1435607</value>
<value>2090667</value>
<value>4726562</value>
</weightVec>
<x>8</x>
<y>5</y>
<weightVec>
<value>5585968</value>
<value>6870544</value>
<value>3311248</value>
</weightVec>
<x>8</x>
<y>6</y>
<weightVec>
<value>3332397</value>
<value>4934478</value>
<value>3920715</value>
</weightVec>
<x>8</x>
<y>7</y>
<weightVec>
<value>8592376</value>
<value>3537841</value>
<value>7640411</value>
</weightVec>
<x>8</x>
<y>8</y>
<weightVec>
<value>6929870</value>
<value>6402801</value>
<value>8436645</value>
</weightVec>
</neuron>
</neurons>
</xml>

Приложение Б

(обязательное)

Код разработанных классов

1) Класс ImgVector.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
namespace diplom
{
   class ImgVector
   {
      //поля класса
      private int size;
      public int[,] vector;
      public bool valid = true;
      int brokenColor;
      string message;
      //функции класса

      public ImgVector(Bitmap imgSource,int[] coords,int size,int brokenColor)
      {
         this.size = size;
         this.vector = new int[size,size];
         this.brokenColor = brokenColor;
         this.message = "";
         int offset = Convert.ToInt32((size -1 )/2);
         int a = 0;
         int b = 0;
         for (int i = (Convert.ToInt32(coords[0]) - offset); i <=
(Convert.ToInt32(coords[0]) + offset); i++)
            {
               b = 0;
```

```
                for (int j = (Convert.ToInt32(coords[1]) - offset); j <=
(Convert.ToInt32(coords[1]) + offset); j++)
                {
                    this.vector[a, b] = imgSource.GetPixel(i, j).ToArgb();
                    b++;
                }
                a++;
            }
        }
        public bool isValid(Bitmap imgSource,int[] coords)
        {
            bool result = false;
            int offset = Convert.ToInt32((this.size - 1) / 2);
            for (int i = (Convert.ToInt32(coords[0]) - offset); i < offset + this.size; i++)
            {
                for (int j = (Convert.ToInt32(coords[1]) - offset); j < offset + this.size; j++)
                {
                    if (this.brokenColor == imgSource.GetPixel(i, j).ToArgb())
                    {
                        result = true;
                        break;
                    }
                }
            }
            return result;
        }
    }
}
```

2) Класс Neuron.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
namespace diplom
```

```
{
    class Neuron
    {
        /*поля класса*/
        public double[,] weight;
        private int[] coords;
        public int size;
        /*конструкторы и методы класса*/
        public Neuron(int size,int x,int y)
        {
            this.size = size;
            this.coords = new int[2];
            this.weight = new double[size,size];
        }
        public int getX()
        {
            return this.coords[0];
        }
        public int getY()
        {
            return this.coords[1];
        }
        public double neuronVectorDistance(ImgVector img)
        {
            double result = 0;
            for (int i = 0; i < this.size; i++)
            {
                for (int j = 0; j < this.size; j++)
                {
                    result += (Math.Pow((this.weight[i, j] - img.vector[i, j]), 2));
                }
            }
            return result;
        }
        public void initWeightWithVector(ImgVector img)
        {
            for (int i = 0; i < this.size; i++)
            {
                for (int j = 0; j < this.size; j++)
```

```
            {
                this.weight[i, j] = Convert.ToDouble(img.vector[i, j]);
            }
        }
    }
    public void drawNeuron(Bitmap img, int x, int y)
    {
        for (int i = 0; i < this.size; i++)
        {
            for (int j = 0; j < this.size; j++)
            {
                if (Convert.ToInt32(this.weight[i, j]) == 0)
                {
                    img.SetPixel(x, y, Color.FromName("white"));
                }
                else
                {
                    img.SetPixel(x, y, Color.FromArgb(Convert.ToInt32(this.weight[i,
j])));
                }
                y++;
            }
            x++;
        }
    }
  }
}
```

3) Класс neuronMap.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Drawing;
namespace diplom
{
```

```
class NeuronMap
{
    private Neuron[,] map;
    private double neuronCount = 0;
    public int neuronSize;
    public int color;
    private int mapWidth = 0;
    private int mapHeight = 0;
    private int iterationCount = 0;
    public NeuronMap(int size,int vectorSize)
    {
        this.mapWidth = size;
        this.mapHeight = size;
        this.map = new Neuron[size,size];
        for (int i = 0; i < size; i++)
        {
            for (int j = 0; j < size; j++)
            {
                this.map[i,j] = new Neuron(vectorSize,i,j);
            }
        }
        this.neuronCount = Math.Pow(Convert.ToDouble(size),2);
        this.neuronSize = vectorSize;
    }
    public double neuronDistance(Neuron fst,Neuron scnd,int stepNum)
    {
        double hcit = 0;
        double a = 1;
        double dci = 0;
        double sigma = 0;
        if (stepNum > 10)
        {
            a = 1 / (Math.Pow((stepNum - 9),0.2));
        }
        sigma = 5 * (Math.Sqrt(this.neuronCount) / Math.Sqrt(stepNum));
        dci = Math.Sqrt(Math.Pow((fst.getX() - scnd.getX()),2) +
Math.Pow((fst.getY() - scnd.getY()),2));
        hcit = a * Math.Exp(-(Math.Pow(dci, 2) / (2 * sigma)));
        return hcit;
```

```
        }
        public double neuronVectorDistance(Neuron fst,ImgVector img)
        {
            double result = 0;
            for (int i = 0; i < fst.size; i++)
            {
                for (int j = 0; j < fst.size; j++)
                {
                    result += (Math.Pow((fst.weight[i, j] - img.vector[i, j]), 2));
                }
            }
            return result;
        }

        public bool learnMap(Bitmap imgSource)
        {
            int[] coords = this.generateCoords(imgSource);
            ImgVector vector = new ImgVector(imgSource, coords, this.neuronSize,
this.color);
            Neuron Bmu = this.findBMU(vector);
            return true;
        }
        public bool initializeMap(Bitmap imgSource)
        {
            int step = this.mapWidth / 10;
            for (int i = step; i < this.mapWidth; i += step)
            {
                for (int j = step; j < this.mapHeight; j += step)
                {
                    int[] coords = this.generateCoords(imgSource);
                    ImgVector vector = new ImgVector(imgSource, coords, this.neuronSize,
this.color);
                    this.map[i, j].initWeightWithVector(vector);
                }
            }
            return true;
        }
        public Neuron findBMU(ImgVector vector)
        {
```

```
            int[] coords = { 0, 0 };
            double distance = 100000;
            for (int i = 0; i < this.mapWidth; i++)
            {
                for (int j = 0; j < this.mapHeight; j++)
                {
                    double tempDistance = this.map[i, j].neuronVectorDistance(vector);
                    if (tempDistance <= distance)
                    {
                        distance = tempDistance;
                        coords[0] = this.map[i, j].getX();
                        coords[1] = this.map[i, j].getY();
                    }
                }
            }
            return this.map[coords[0], coords[1]];
        }
        public int[] generateCoords(Bitmap imgSource)
        {
            Random generator = new Random();
            int[] returned = new int[2];
            returned[0] = generator.Next(0, imgSource.Width);
            returned[1] = generator.Next(0, imgSource.Height);
            return returned;
        }
        public Bitmap showMap()
        {
            int a=0;
            int b=0;
            Bitmap mapImg = new Bitmap(this.mapWidth *
this.neuronSize,this.mapHeight * this.neuronSize);
            for (int i = 0; i < this.mapWidth; i++)
            {
                a = 0;
                for (int j = 0; j < this.mapHeight; j++)
                {
                    this.map[i, j].drawNeuron(mapImg,a,b);
                    a=a+this.neuronSize-1;
                }
```

```
                b = b + this.neuronSize-1;
            }
                return mapImg;
        }
    }
}
```

Приложение В

(обязательное)

Графические материалы

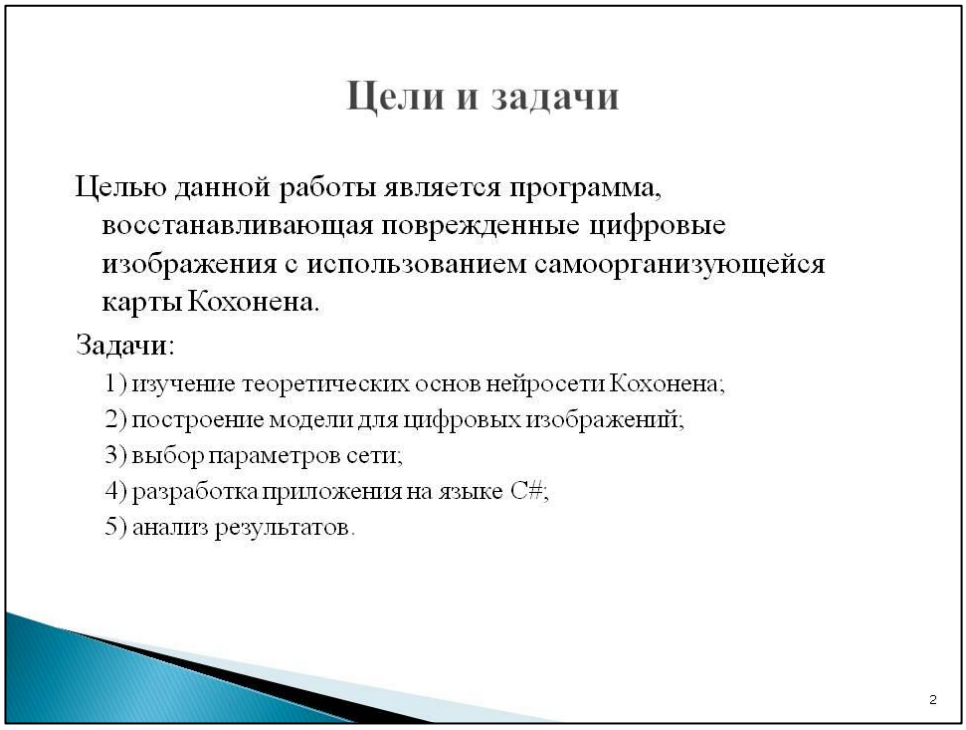

Рисунок В.1 – Цели и задачи

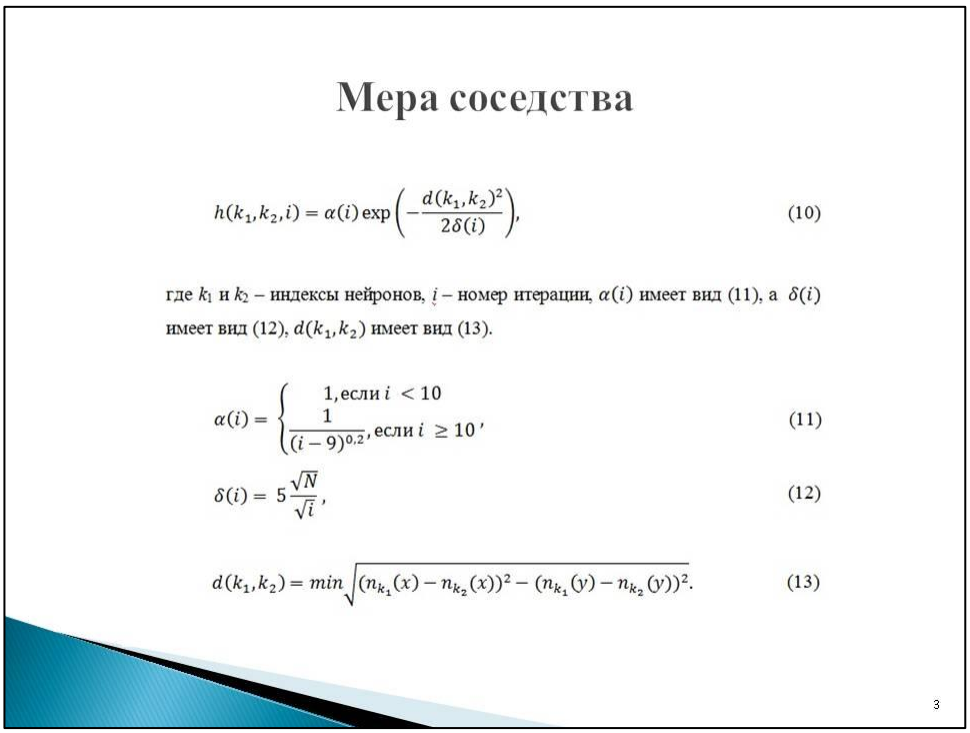

Рисунок Б.2 – Мера соседства нейронов

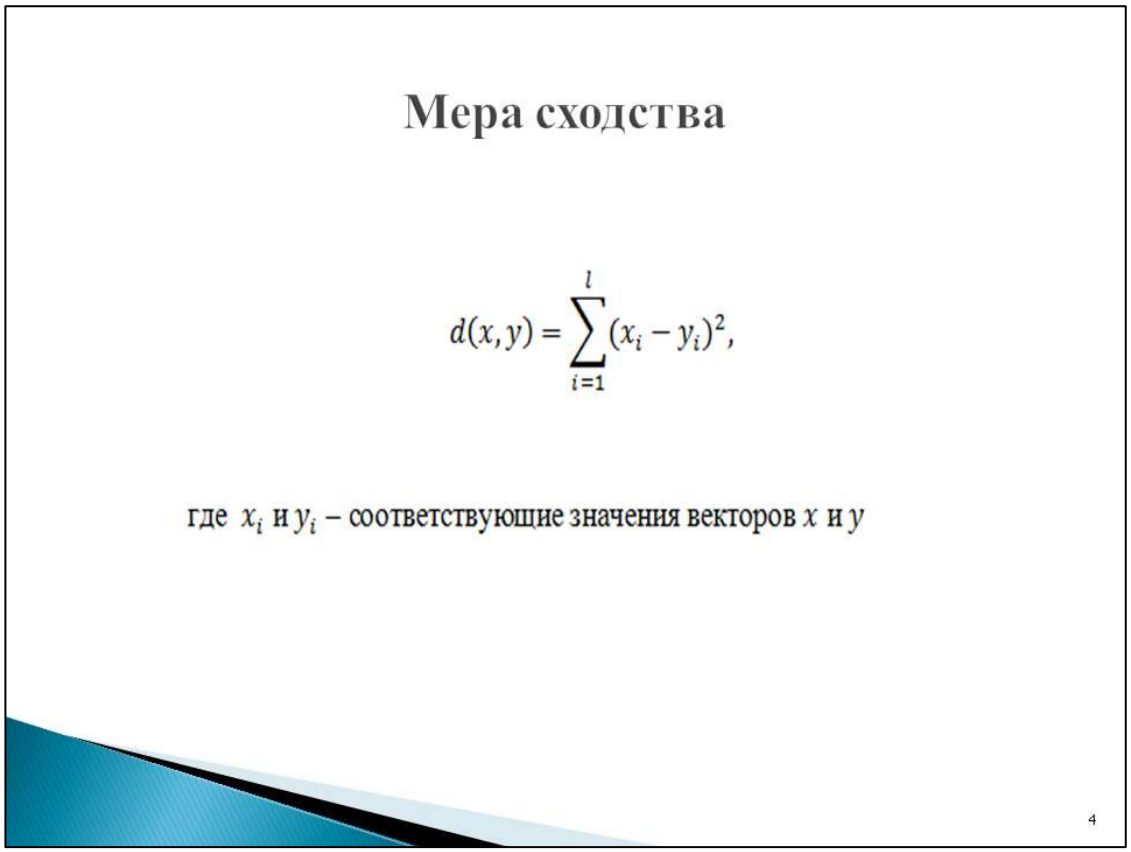

Рисунок Б.3 – Мера сходства нейронов

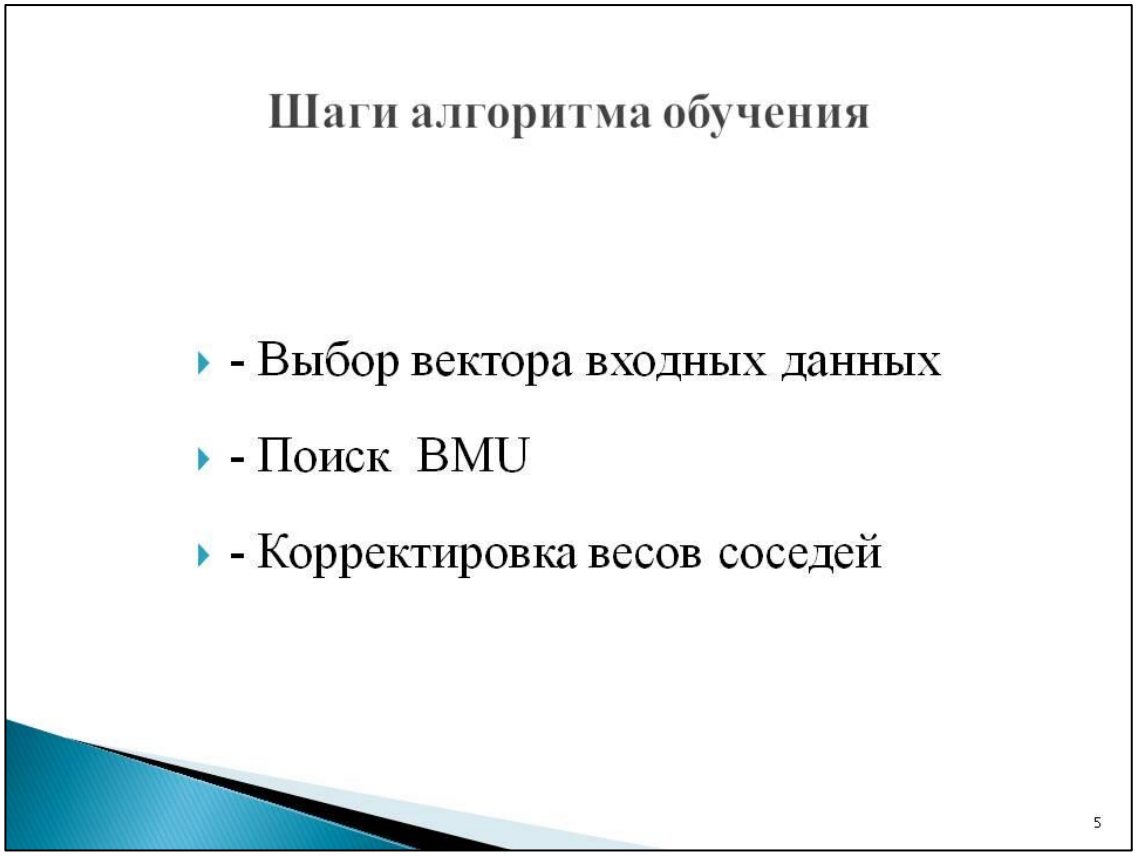

Рисунок Б.4 – Шаги алгоритма обучения

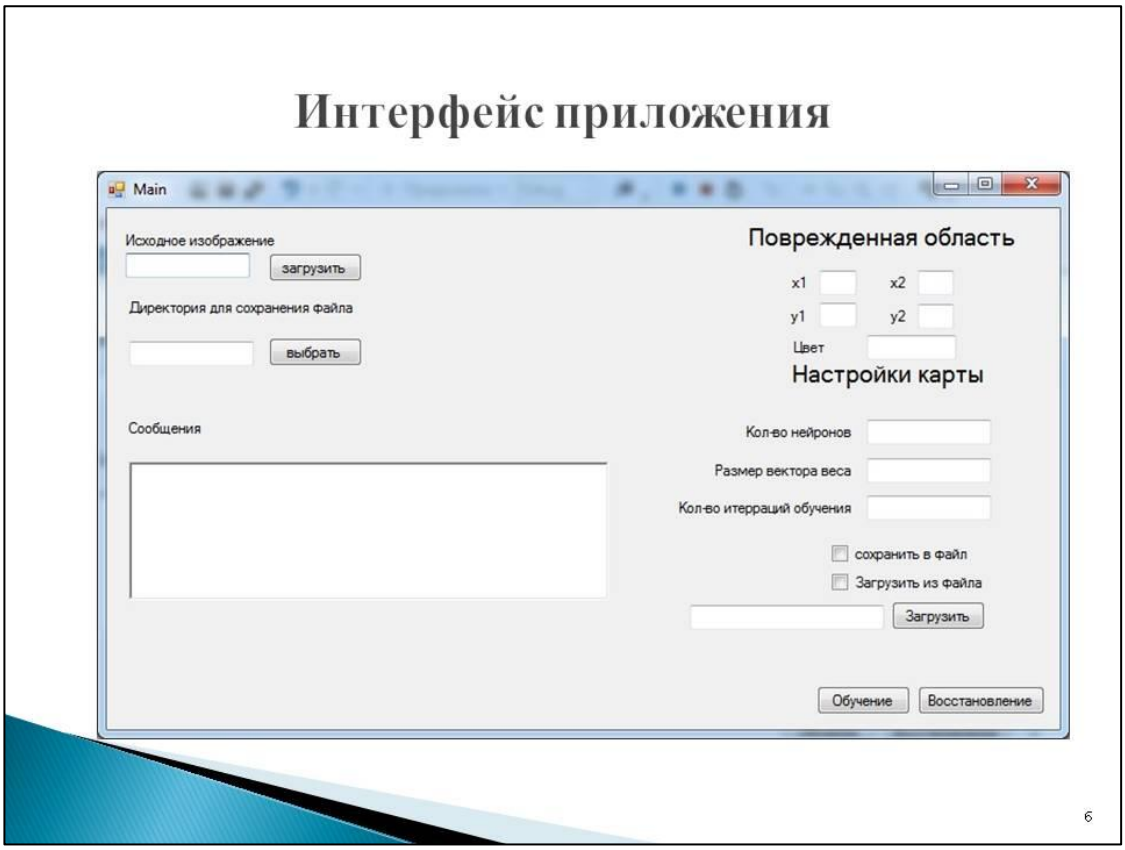

Рисунок Б.5 – Интерфейс приложения

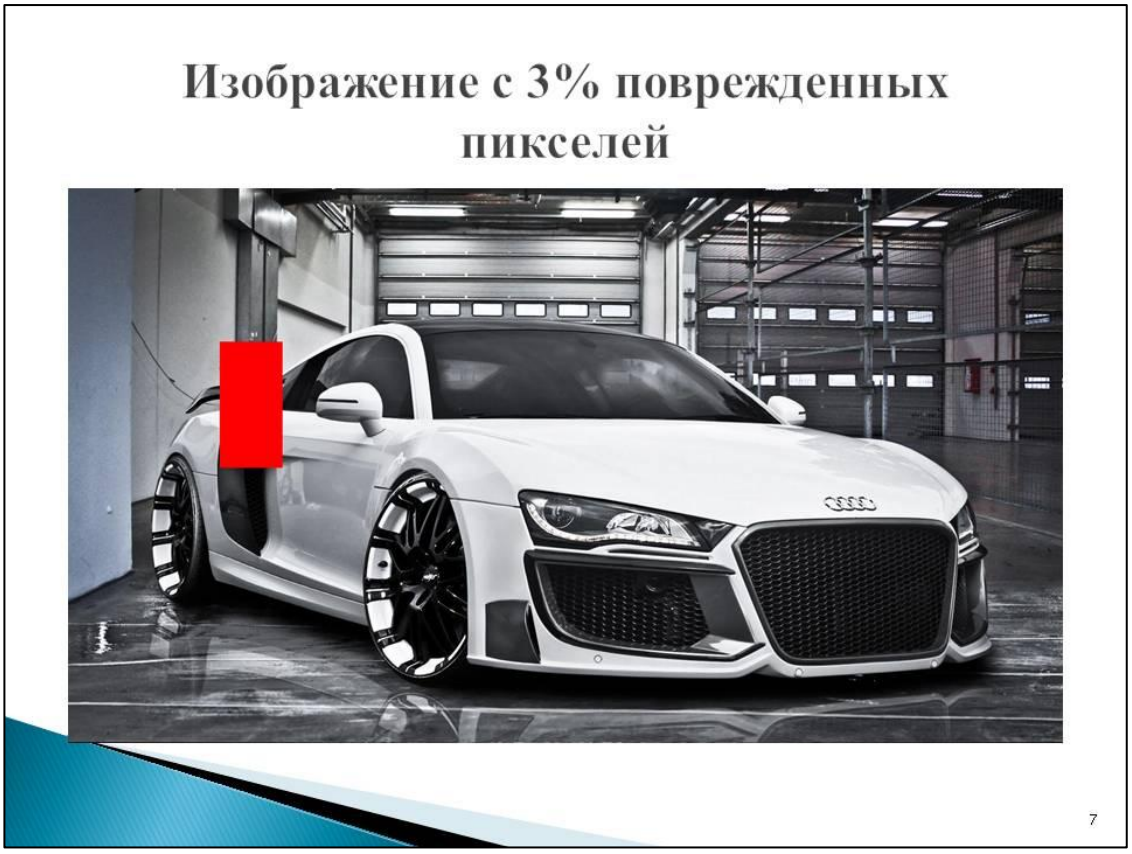

Рисунок Б.6 – Изображение с 3% поврежденных пикселей

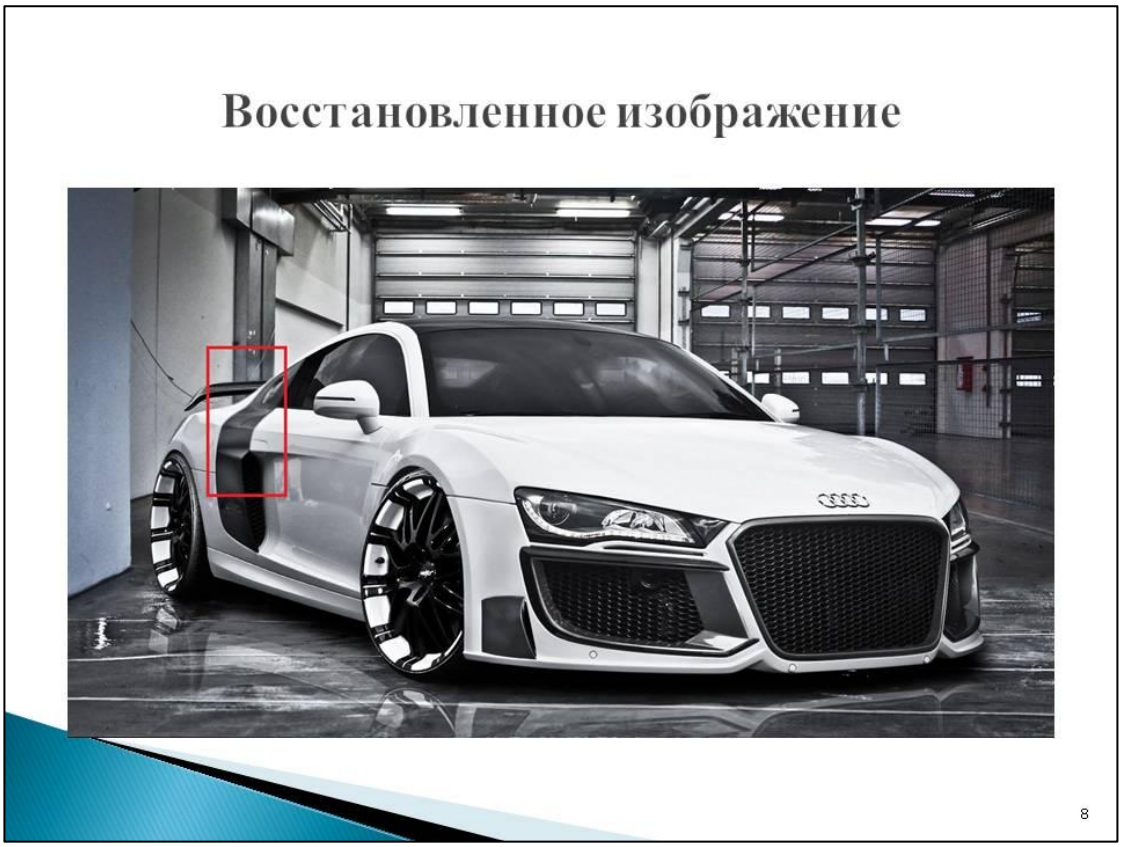

Рисунок Б.7 – Восстановленное изображение №1

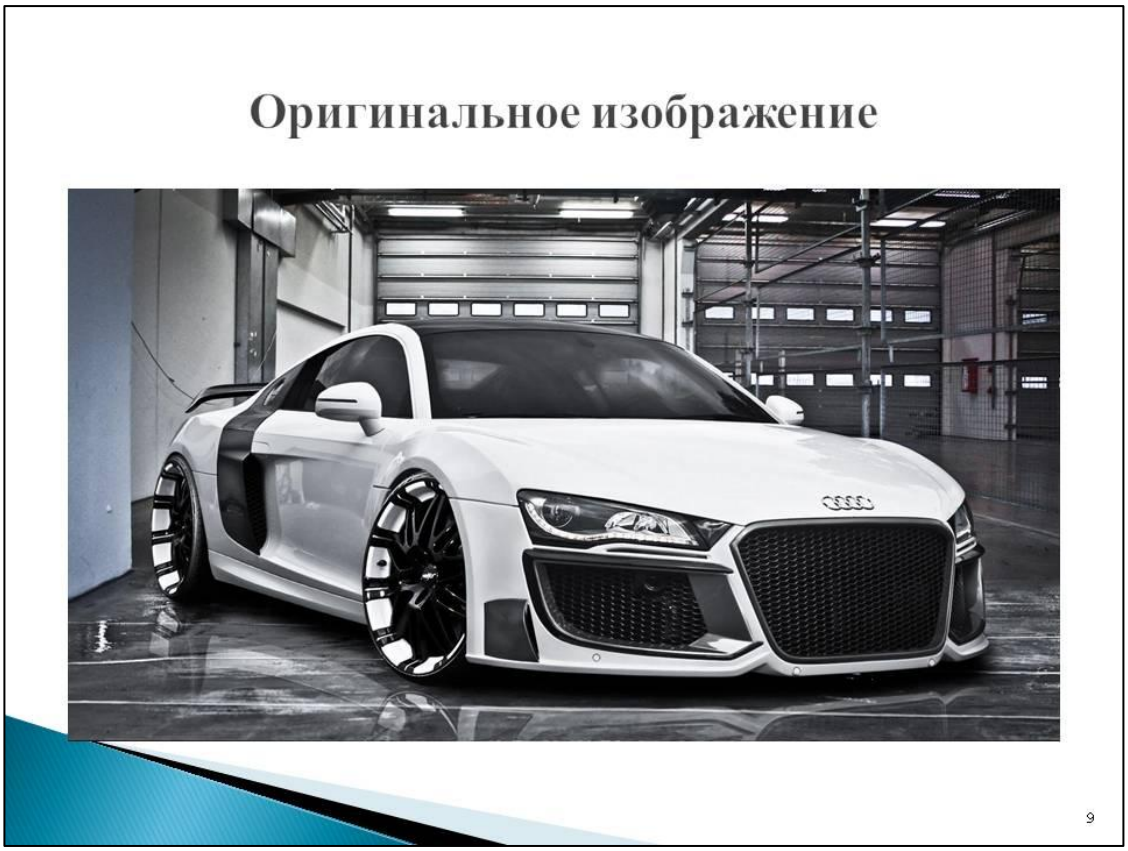

Рисунок Б.8 – Оригинальное изображение №1

Рисунок Б.9 – Изображение с более чем 70% поврежденных пикселей

Рисунок Б.10 – Восстановленное изображение №2