МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Нижегородский государственный университет   
им. Н.И. Лобачевского»

**Институт информационных технологий, математики и механики**

**Кафедра информатики и автоматизации научных исследований**

Направление: Прикладная информатика

**МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ**

Тема:

**«Нейросетевой модуль гибридной интеллектуальной системы»**

**Выполнил:**

студент группы 381207 - 1

Алявдин Александр Владимирович

Подпись\_\_\_\_\_\_\_\_\_\_

**Научный руководитель:**

к.т.н., доцент Басалин П.Д.

Подпись\_\_\_\_\_\_\_\_\_\_

**Зав. Кафедры**:

д.т.н, профессор Прилуцкий М.Х.

Подпись\_\_\_\_\_\_\_\_\_

Нижний Новгород  
2016

Оглавление

[Введение 3](#_Toc453286817)

[1. Сжатие изображений 4](#_Toc453286818)

[1.1. Классификация изображений 4](#_Toc453286819)

[1.2. Требования к алгоритмам компрессии изображений 5](#_Toc453286820)

[1.3. Критерии оценки алгоритмов сжатия 5](#_Toc453286821)

[2. Основные алгоритмы сжатия изображений 7](#_Toc453286822)

[2.1. Сжатие изображений без потерь 7](#_Toc453286823)

[2.2. Cжатие изображений с потерями 8](#_Toc453286824)

[2.2.1. JPEG 8](#_Toc453286825)

[2.2.2. Фрактальный алгоритм 11](#_Toc453286826)

[2.2.3. Вейвлет сжатие 12](#_Toc453286827)

[3. Применение нейронных сетей к задаче сжатия изображений 14](#_Toc453286828)

[3.1. Концепция искусственной нейронной сети. 14](#_Toc453286829)

[3.2. Многослойный персептрон 17](#_Toc453286830)

[3.3. Алгоритм обратного распространения ошибки 18](#_Toc453286831)

[4. Многослойный персептрон типа «бутылочное горлышко» 22](#_Toc453286832)

[4.1. Структура персептрона 22](#_Toc453286833)

[4.2. Поэтапный алгоритм компрессии и декомпрессии 23](#_Toc453286834)

[5. Программная реализация алгоритма 24](#_Toc453286835)

[5.1. Полученные результаты 25](#_Toc453286836)

[5.2. Визуальная оценка полученных результатов: 27](#_Toc453286837)

[Заключение 30](#_Toc453286838)

[Список используемой литературы 31](#_Toc453286839)

[Приложение: Исходные коды разработанного программного обеспечения 3](#_Toc453286840)2

Введение

Во многих областях своей деятельности человеку приходится сталкиваться с задачами трудноформализуемого и неформального плана, алгоритмы (сценарии) решения которых заранее не известны.

К таким задачам относятся задачи, математически формулируемые, но не имеющие априори известного алгоритма решения, гарантирующего получение приемлемого результата в реальном масштабе времени, особенно в случае их высокой размерности (NP-трудные проблемы). К таким задачам относятся, в частности, задачи принятия решений в предметных областях, связанных с диагностикой, прогнозированием процессов, управлением сложными системами.

Помочь человеку в решении этих задач способна *система интеллектуальной поддержки*, объединяющая возможности компьютера со знаниями и опытом эксперта в такой форме, что она способна предложить «разумный» совет или осуществить «разумное» решение поставленной задачи. Такая система должна уметь рассуждать при сомнительных, неполных данных, объяснять ход своих рассуждений понятным пользователю способом, самообучаться и адаптироваться к конкретным условиям применения.

Концепция системы, основанной на знаниях (СОЗ), позволяет добиться этой цели, обеспечивает открытость и масштабируемость системы, однако аналитический механизм получения решения в таких системах достаточно громоздкий и не всегда срабатывает в реальном масштабе времени.

Для решения данной проблемы предлагается использовать нейронную сеть, что значительно повышает производительность системы, кроме того, появляется возможность интегрировать аналитический модуль системы с нейросетевым, используя результаты работы аналитического модуля для дообучения нейронной сети.

*В первом разделе* данной работы поставлены основные цели и задачи.

*Во втором разделе* рассмотрены концепции представления знаний на инфологическом уровне, их преимущества и недостатки.

*В третьем разделе* описана концепция гибридной системы интеллектуальной поддержки, концепция системы, основанной на знаниях, которая является базовой для гибридной системы, а также основные компоненты гибридной системы.

*В четвертом разделе* рассмотрен многослойный персептрон - структура нейронной сети, на базе которой предлагается создать нейронную сеть для нейросетевого модуля гибридной системы. Также в этом разделе рассмотрены основные типы активационных функций нейронов.

*В пятом разделе* описан подход создания нейросетевого модуля гибридной системы интеллектуальной поддержки, его основные компоненты, а также механизм построения и структура нейронной сети, которые лежат в его основе. Также в этом разделе рассмотрены концепции дообучения сети.

*В шестом разделе* описана программная реализация модуля, его архитектура, диаграммы классов и детали реализации.

*В седьмом разделе* приведены результаты работы системы на тестовом базисе и сравнение результатов работы нейросетевого модуля с аналитическим модулем на тестовых примерах.

1. Постановка задачи

Основная цель данной работы - создать модуль, позволяющий принимать решения в различных предметных областях на основании нейросетевого механизма вывода и, при необходимости, использовать результаты работы аналитического модуля для дообучения нейронной сети.

Для достижения данной цели поставлены следующие задачи:

1. **Разработать подход к созданию нейросетевого модуля гибридной системы интеллектуальной поддержки.** В рамках данной задачи, необходимо рассмотреть концепцию создания системы, основанной на знаниях и гибридной интеллектуальной системы. Разработать алгоритм преобразования знаний в эквивалентную по «рассуждениям» нейронную сеть, а также сформировать структуру нейронной сети и способ дообучения нейронной сети
2. **Разработать архитектуру нейросетевого модуля.** В рамках данной задачи необходимо описать основные компоненты нейросетевого модуля гибридной системы интеллектуальной поддержки, разработать программную архитектуру, диаграммы классов.
3. **Создать программную реализацию.** В рамках данной задачи необходимо определиться со средой разработки, языком, используемыми форматами файлов и библиотеками, написать и протестировать программу, создать тестовую инфраструктуру.
4. **Сравнить результаты работы нейросетевого и аналитического модулей гибридной системы.** В рамках данной задачи нужно сравнить полученные результаты двух модулей и проанализировать их.

2. Представление знаний о предметной области на инфологическом уровне

2.1. Дерево решений как формализм описания знаний на инфологическом уровне

Анализ действий экспертов, связанных с диагностикой состояний сложных объектов и систем, показывает, что профессионал проводит экспертизу вполне целенаправленно, придерживаясь определенной стратегии. При этом он руководствуется множеством правил (эвристик), которые могут быть представлены в форме

ЕСЛИ *<посылка>* ТО  *<заключение>,*

называемой *продукционным правилом*.

Правило может быть задано с *коэффициентом уверенности*, определяющим положительным значением меньше или равном единице.

Однако в продукционной системе (системе, основанной на продукционных правилах) для представления знаний удобно используется *дерево решений*, которое к тому же существенно компенсирует недостаток продукционных правил, связанный с отсутствием наглядности представления иерархии понятий.

2.2. Cжатие изображений с потерями

2.2.1. JPEG

Оперирует областями 8х8, на которых яркость и цвет меняются сравнительно плавно. Вследствие этого, при разложении матрицы такой области в двойной ряд по косинусам значимыми оказываются только первые коэффициенты. Таким образом, сжатие в JPEG осуществляется за счет плавности изменения цветов в изображении.

Алгоритм основан на дискретном косинусоидальном преобразовании (в дальнейшем ДКП), применяемом к матрице изображения для получения некоторой новой матрицы коэффициентов. Для получения исходного изображения применяется обратное преобразование. ДКП раскладывает изображение по амплитудам некоторых частот. Таким образом, при преобразовании мы получаем матрицу, в которой многие коэффициенты либо близки, либо равны нулю. Кроме того, благодаря несовершенству человеческого зрения, можно аппроксимировать коэффициенты более грубо без заметной потери качества изображения. Для этого используется квантование коэффициентов (quantization). При этом преобразовании теряется часть информации, но могут достигаться большие коэффициенты сжатия.

Шаг 1.Переводим изображение из пространства RGB в пространство YCbCr.

RGB - цветовое пространство, наиболее широко используется в компьютерной графике. Красный, зеленый и голубой - главные компоненты цветов и представляют три размерности данного пространства.

YCbCr – цветовое пространство, позволят отобразить цветовое изображение более эффективно, отделив светимость от цветовой информации и представив ее с большим разрешением, чем цвет. Буква Y в этом цветовом пространстве обозначает компоненту светимость, которая вычисляется как взвешенное усреднение компонент R, G и B по следующей формуле:

http://sernam.ru/htm/lect_cod/files/cod_7.files/image002.gif,

где http://sernam.ru/htm/lect_cod/files/cod_7.files/image003.gif обозначает соответствующий весовой множитель. Остальные цветовые компоненты определяются в виде разностей между светимостью Y и компонентами R, G и B:

Cb=B-Y, Cr =R-Y, Cg=G-Y.

При этом получаются четыре компоненты нового пространства вместо трех RGB. Однако число Cb+Cr+Cg является постоянным, поэтому только две из трех хроматических компонент необходимо хранить, а третью вычислять на основе них. Чаще всего в качестве две искомых цветовых компонент используют Cb и Cr.  Преимущество пространства YCbCr по сравнению с RGB заключается в том, что Cb и Cr можно представлять с меньшим разрешением, чем Y, т.к. глаз человека менее чувствителен к цвету предметов, чем к их яркости. Это позволяет сократить объем информации, требуемый для представления хроматических компонент. [5]

Шаг 2. Разбиваем исходное изображение на матрицы 8х8. Формируем из каждой три рабочие матрицы ДКП – по 8 бит отдельно для каждой компоненты. При больших степенях сжатия блок 8х8 раскладывается на компоненты YCbCr в формате 4:2:0, т.е. компоненты для Cb и Cr берутся через точку по строкам и столбцам.

Шаг 3. Применение ДКП к блокам изображения 8х8 пикселей. В результате ДКП получаем матрицу http://sernam.ru/htm/lect_cod/files/cod_15.files/image011.gif, в которой коэффициенты в левом верхнем углу соответствуют низкочастотной составляющей изображения, а в правом нижнем – высокочастотной.

Шаг 4. Квантование. На этом шаге происходит отбрасывание части информации. Здесь каждое число из матрицы http://sernam.ru/htm/lect_cod/files/cod_15.files/image011.gif делится на специальное число из «таблицы квантования», а результат округляется до ближайшего целого:

http://sernam.ru/htm/lect_cod/files/cod_15.files/image014.gif.

Причем для каждой матрицы Y, Cb и Cr можно задавать свои таблицы квантования. Стандарт JPEG даже допускает использование собственных таблиц квантования, которые, однако, необходимо будет передавать декодеру вместе со сжатыми данными, что увеличит общий размер файла. [5]

Понятно, что пользователю сложно самостоятельно подобрать 64 коэффициента, поэтому стандарт JPEG использует два подхода для матриц квантования.

Первый заключается в том, что в стандарт JPEG включены две рекомендуемые таблицы квантования: одна для яркости, вторая для цветности.

На этапе квантования осуществляется управление степенью сжатия, и происходят самые большие потери. Понятно, что задавая таблицы квантования с большими коэффициентами, мы получим больше нулей и, следовательно, большую степень сжатия.

Матрицы, используемые для квантования коэффициентов ДКП, хранятся в заголовочной части JPEG-файла. Обычно они строятся так, что высокочастотные коэффициенты подвергаются более сильному квантованию, чем низкочастотные. Это приводит к огрублению мелких деталей на изображении. Чем выше степень сжатия, тем более сильному квантованию подвергаются все коэффициенты. [5]

Шаг 5. Матрица 8х8 переводится в 64-элементный вектор при помощи «зигзаг»-сканирования и данный вектор преобразовывается с помощью модифицированного алгоритма RLE, на выходе которого получаются пары типа (пропустить, число), где «пропустить» является счетчиком пропускаемых нулей, а «число» - значение, которое необходимо поставить в следующую ячейку. Далее пары кодируются методом Хаффмана.

Коэффициенты компрессии: 2-200 (Задается пользователем).

Алгоритм JPEG в наибольшей степени пригоден для сжатия фотографий и картин, содержащих реалистичные сцены с плавными переходами яркости и цвета. Наибольшее распространение JPEG получил в [цифровой фотографии](https://ru.wikipedia.org/wiki/%D0%A6%D0%B8%D1%84%D1%80%D0%BE%D0%B2%D0%B0%D1%8F_%D1%84%D0%BE%D1%82%D0%BE%D0%B3%D1%80%D0%B0%D1%84%D0%B8%D1%8F) и для хранения и передачи изображений с использованием сети [Интернет](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BD%D0%B5%D1%82). Формат JPEG в режиме сжатия с потерями малопригоден для сжатия чертежей, текстовой и знаковой графики, где резкий контраст между соседними пикселями приводит к появлению заметных [артефактов](https://ru.wikipedia.org/wiki/%D0%90%D1%80%D1%82%D0%B5%D1%84%D0%B0%D0%BA%D1%82%D1%8B_%D1%81%D0%B6%D0%B0%D1%82%D0%B8%D1%8F). С квантованием связаны и специфические эффекты алгоритма.

При больших значениях шага квантования потери могут быть настолько велики, что изображение распадется на однотонные квадраты 8х8.

2.2.2. Фрактальный алгоритм

Фрактальная архивация основана на том, что мы представляем изображение в более компактной форме — с помощью коэффициентов системы итерируемых функций (Iterated Function System — далее по тексту как IFS).

IFS представляет собой набор трехмерных аффинных преобразований, в данном случае переводящих одно изображение в другое.

Шаг 1.

Изображение разбивается на ранговые области ri (непересекающиеся области, покрывающие все изображение).

Шаг 2.

Далее для каждой области ri находят область di и преобразование wi такие, что выполняются следующие условия:

di по размерам больше ri.

wi (ri) имеет ту же форму, размеры и положение, что и ri.

Коэффициент u преобразования wi должен быть меньше единицы.

Значение должно быть как можно меньше.

Шаг 3.

Запомнить коэффициенты аффинных преобразований W, положения доменных областей di, а также разбиение изображения на домены[[1]](#footnote-1)

Коэффициенты компрессии: 2-2000 (Задается пользователем).

Сравнение с JPEG: Фрактальный алгоритм, в зависимости от оптимизации, сжимает дольше, чем JPEG. Распаковка изображения, наоборот, происходит быстрее.

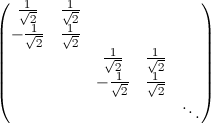
JPEG использует разложение изображения по косинусоидальным функциям, поэтому потери в нем (даже при заданных минимальных потерях) проявляются в волнах и ореолах на границе резких переходов цветов. [[2]](#footnote-2)

2.2.3. Вейвлет сжатие

Идея алгоритма заключается в том, что мы сохраняем в файл разницу — число между средними значениями соседних блоков в изображении, которая обычно принимает значения, близкие к 0.

Шаг 1.

К матрице изображения применяется преобразование Хаара. Степень сжатия можно увеличить, применяя преобразование Хаара многократно. Преобразование описывается матрицей вида



Преобразование Хаара — пара фильтров, разделяющих сигнал на низкочастотную и высокочастотную составляющие. Низкочастотная составляющая несёт в себе информацию об общей форме лица, о плавных перепадах яркости. Высокочастотная — это шум и мелкие детали.

Шаг 2.

Квантование. Меняя число обнуляемых коэффициентов, можно регулировать степень сжатия.

Кроме преобразования Хаара используется преобразование Добеши, оперирующее четверками чисел.

Коэффициенты компрессии: 2-200 (Задается пользователем).

Характерные особенности: Ориентирован алгоритм на цветные и черно-белые изображения с плавными переходами. При попытке задать большой коэффициент на резких границах, особенно проходящих по диагонали, проявляется “лестничный эффект” — ступеньки разной яркости размером в несколько пикселов.

Алгоритм JPEG, в отличие от вейвлетного, сжимает по отдельности каждый блок исходного изображения размером 8 на 8 пикселов. В результате при высоких степенях сжатия на восстановленном изображении может быть заметна блочная структура. При вейвлетном сжатии такой проблемы не возникает, но могут появляться искажения другого типа, имеющие вид «призрачной» ряби вблизи резких границ.

Преимущество метода сжатия Wavelet перед JPEG состоит в том, что Wavelet преобразует полное изображение, а не его отдельные фрагменты, и позволяет получить качественное изображение при больших (до 100) коэффициентах сжатия. [3]

|  |  |  |  |
| --- | --- | --- | --- |
|  | Коэф. сжатия | Отношение времени компрессии к времени декомпрессии | Класс изображений |
| JPEG | 2-200 раз | ~1 | 24-битные, серые |
| Фрактальное | 2-200 раз | 1.5 | 24-битные, серые |
| Вейвлетт | 2-2000 раз | 1000-10000 | 24-битные, серые |

3. Применение нейронных сетей к задаче сжатия изображений

3.1. Концепция искусственной нейронной сети.

*Искусственная нейронная сеть* (ИНС) – это существенно параллельно распределенный процессор, обладающий способностью к приобретению, сохранению и репрезентации опытного знания и сходный с мозгом в двух аспектах:

знание приобретается сетью в процессе обучения;

для сохранения знания используются силы межнейронных связей (*синаптические веса*).

Составные элементы ИНС:

*нейроны*, суммирующие поступающие на их входы сигналы и преобразующие смещенную на величину порога сумму в соответствии с заданной активационной функцией нейрона;

*связи между нейронами*, реализующие межнейронные взаимодействия в виде сигналов, умножаемых на синаптические веса связей.

Схема работы нейрона представлена на рисунке 1.

 — сигналы, поступающие на входные связи нейрона;

 — синаптические веса входных связей;

 — порог срабатывания нейрона;

— Сумма взвешенных входных сигналов, смещенная на величину порога

 — активационная функция нейрона, реализующая тот или иной вид преобразования суммы взвешенных входных сигналов, смещенной на величину порога активационной функции .[8]

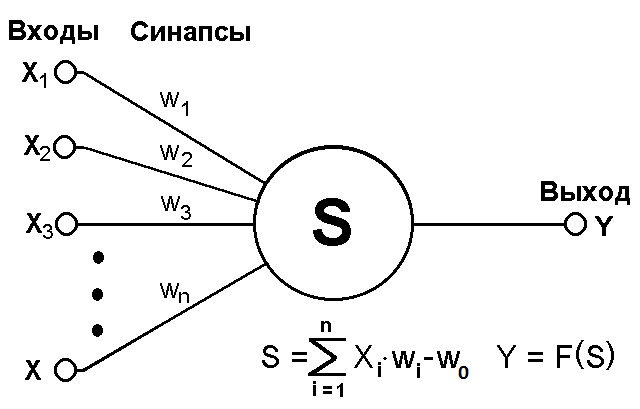


Рис. 1 Схема топологии искусственного нейрона

Активационная функция нейрона определяет нелинейное преобразование, осуществляемое нейроном.

Существует много видов активационных функций, но более всего распространены следующие:

а) *линейная* активационная функция

.

б) *линейная с насыщением*

,

Имеет настроечный параметр .

в) *ступенчатая* активационная функция

.

г) сигмоидальная активационная функция



С параметром крутизны а.

д) радиально - симметричная



С параметром **.**

е) *пороговая* активационная функция



*f*(*s*)

*s*

1

*Рис.5*  Сигмоидальная активационная функция

0,5

*-σ*

1

*f*(*s*)

*s*

*a*

*Рис. 6*  Радиально-симметричная активационная функция

0

0

*K*

*s*

*f*(*s*)

*Рис. 7*  Пороговая активационная функция

*σ*

*a*

*s*

*f*(*s*)

- *a*

- *a*

*a*

*Рис. 3*  Линейная с насыщением активационная функция

0

*s*

*f*(*s*)

0

1

*s*

*f*(*s*)

*Рис.4*  Ступенчатая активационная функция

*Рис.2*  Линейная активационная функция

0

Один из самых распространенных видов активационной функции, используемый, в первую очередь, для персептронов – это сигмоидальная функция. Следует отметить, что сигмоидальная функция дифференцируема на всей оси абсцисс. Непрерывность первой производной позволяет обучать сеть градиентными методами (например, метод обратного распространения ошибки). Введение функций сигмоидального типа было обусловлено ограниченностью нейронных сетей с пороговой функцией активации нейронов — при такой функции активации любой из выходов сети равен либо нулю, либо единице, что ограничивает использование сетей не в задачах классификации. Сигмоид сужает диапазон изменения так, что значение выхода нейрона лежит между нулем и единицей. Его дополнительное преимущество состоит в автоматическом контроле усиления. Для слабых сигналов кривая вход-выход имеет сильный наклон, дающий большое усиление. Когда величина сигнала становится больше, усиление падает. Таким образом, большие сигналы воспринимаются сетью без насыщения, а слабые сигналы проходят по сети без чрезмерного ослабления. [9]

3.2. Многослойный персептрон

*Многослойный персептрон* представляет собой сеть прямого распространения с одним *входным* слоем, одним *выходным* слоем и одним или несколькими *внутренними* слоями нейронов. Характерной его чертой является *прямонаправленность*: информация, преобразуясь, передается от входного слоя через скрытых слоев к выходному слою.

В стандартной (регулярной) топологии персептрона каждый нейрон   слоя   непосредственно воздействует с синаптическими весами  на все нейроны   следующего слоя  и ни на какие другие (– соответственно, число нейронов в слое  и в слое ;  и  обозначают, соответственно, входной и выходной слои). [[3]](#footnote-3)

Схема многослойного персептрона стандартной топологии представлена на рис. 8.

**..................**

**.......................**

**................**

**.......................**

*х1*

*х2*

*хN*

*y1*

*y2*

*yM*

Входной

слой

Скрытые слои

Выходной слой

Рис. 8 Многослойный персептрон стандартной топологии

**.................................**

Модифицированные версии многослойного персептрона могут иметь нерегулярные связи между слоями (какие-то связи могут отсутствовать) и прямые (непосредственные) связи между нейронами несмежных слоев.

Входной слой персептрона выполняет функции приема и ретрансляции входных сигналов  на нейроны первого скрытого слоя. Основное нелинейное преобразование сигналов происходит в скрытых слоях. Нейроны выходного слоя осуществляют суперпозицию взвешенных сигналов последнего скрытого слоя (имеют линейную активационную функцию) или выполняют нелинейное преобразование, как и нейроны скрытых слоев.

*Обучение персептрона* – это итеративный целенаправленный процесс изменения значений весов синаптических связей (и, возможно, порогов активационных функций нейронов), реализуемый «внешней» процедурой (*алгоритмом обучения*) с использованием тренировочных шаблонов (*обучение с супервизором*) до тех пор, пока сеть не приобретет желаемые свойства.

3.3. Алгоритм обратного распространения ошибки

Алгоритм обратного распространения ошибки является одним из методов обучения многослойных нейронных сетей прямого распространения, называемых также многослойными персептронами. Многослойные персептроны успешно применяются для решения многих сложных задач. Обучение алгоритмом обратного распространения ошибки предполагает два прохода по всем слоям сети: прямого и обратного. При прямом проходе входной вектор подается на входной слой нейронной сети, после чего распространяется по сети от слоя к слою. В результате генерируется набор выходных сигналов, который и является фактической реакцией сети на данный входной образ. Во время прямого прохода все синаптические веса сети фиксированы. Во время обратного прохода все синаптические веса настраиваются в соответствии с правилом коррекции ошибок, а именно: фактический выход сети вычитается из желаемого, в результате чего формируется сигнал ошибки. Этот сигнал впоследствии распространяется по сети в направлении, обратном направлению синаптических связей. Отсюда и название – алгоритм обратного распространения ошибки. Синаптические веса настраиваются с целью максимального приближения выходного сигнала сети к желаемому. [8]

*Тренировочный шаблон* представляет собой пару векторов , один из которых  – вектор известных входных сигналов, другой  – вектор желаемых выходных сигналов. В процессе обучения векторы  из тренировочного набора  последовательно подаются на вход ИНС и для каждого из них оценивается ошибка между фактическим  и желаемым  откликом (выходом) сети . Затем определяется общая ошибка , на основании которой алгоритм обучения осуществляет модификацию значений настроечных параметров сети, направленную на уменьшение ошибки. Как вариант, модификация значений варьируемых параметров сети может осуществляться после оценки действия каждого очередного шаблона, т.е. по «локальной» ошибке .

Процесс обучения повторяется до тех пор, пока сеть не приобретет способность выполнять желаемый тип преобразования, заданный тренировочным набором шаблонов .

Один из самых распространенных и эффективных методов обучения многослойных персептронов - *алгоритм обратного распространения ошибки* (АОР, или BPE – Back-Propagation Error).

В основу алгоритма BPE положен метод градиентного спуска, согласно которому на каждой итерации поиска значений синаптических весов и порогов активационных функций нейронов, направленной на уменьшение ошибки , их приращения определяются по формулам:

, , ; ; ,

где  — синаптический вес связи *j-*го нейрона (*k–*1)-го слоя с *i-*м нейроном *k*-го слоя;

— порог активационной функции *i*-го нейрона *k*-го слоя;

*Hk* – число нейронов в *k*-м слое;

*η*∈(0,1] — настроечный параметр обучения.

Согласно правилу дифференцирования сложной функции, производная от функции ошибки по определяется: ,

где  — сигнал на выходе *i*-го нейрона *k*-го слоя;

 — сдвинутая на величину порога сумма взвешенных входных сигналов *i*-го нейрона *k*-го слоя; .

Все выражения приведены для случая оценки локальной ошибки  (для одного шаблона), поэтому индекс шаблона везде опущен.

Представление производной  через выходы нейронов (*k+*1)-го слоя



позволяет получить рекурсивную формулу 

для пересчета величин  со слоя *k*+1 на слой *k*  (то, что называют обратным распространением ошибки).

Очевидно, для слоя *K+*1(выходного слоя), с которого начинается обратное распространение ошибки, .

Таким образом, итерационная формула для коррекции значений синаптических весов принимает следующий вид:



Аналогично получается итерационная формула коррекции значений порогов активационных функций нейронов:



Выбор той или иной активационной функции определяется многими факторами, среди которых в первую очередь следует выделить существование производной на всей оси абсцисс и удобство организации вычислений. С учетом этих факторов в алгоритме BPE чаще используют сигмоидальную активационную функцию. Для нее

.

 — сигнал на выходе *j*-го нейрона выходного слоя персептрона. [[4]](#footnote-4)

4. Многослойный персептрон типа «бутылочное горлышко»

4.1. Структура персептрона

Многослойный персептрон в особой конфигурации может быть использован для сжатия и кодирования данных, в частности, изображений. Для этой цели можно использовать персептрон со следующей топологией: (Рис.4)

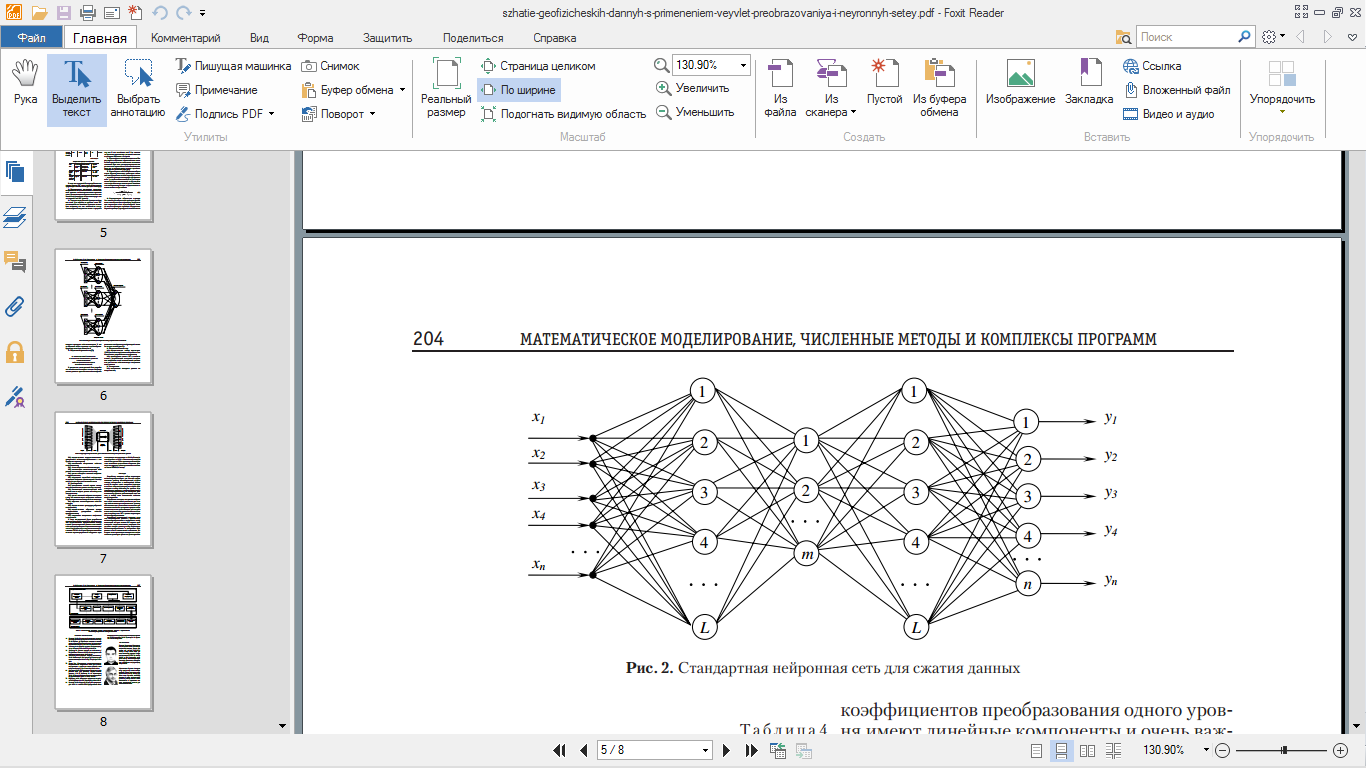


Рис. 9. Концептуальная топология персептрона для сжатия данных.

Особенности конфигурации такой сети:

Количество нейронов входного слоя равно количеству нейронов выходного слоя

Количество скрытых слоёв больше или равно 1.

Количество нейронов в каждом скрытом слое меньше количества нейронов во входном слое и регулируется коэффициентом сжатия.

Каждый слой имеет связи только со следующим и предыдущим, причем каждый нейрон одного слоя связан с каждым нейроном следующего слоя.

Идея алгоритма работы данной сети состоит в том, чтобы обучить сеть на шаблонах с одинаковыми значениями обучающих векторов Х и У,  – вектор известных входных сигналов,  – вектор желаемых выходных сигналов. Далее, обученная сеть разделяется на две составляющие – кодер («левая» часть сети, включающая в себя входной и несколько (от 1 до n) скрытых слоёв исходной нейронной сети) и декодер (включающий в себя оставшиеся скрытые слои и выходной слой исходной нейронной сети).

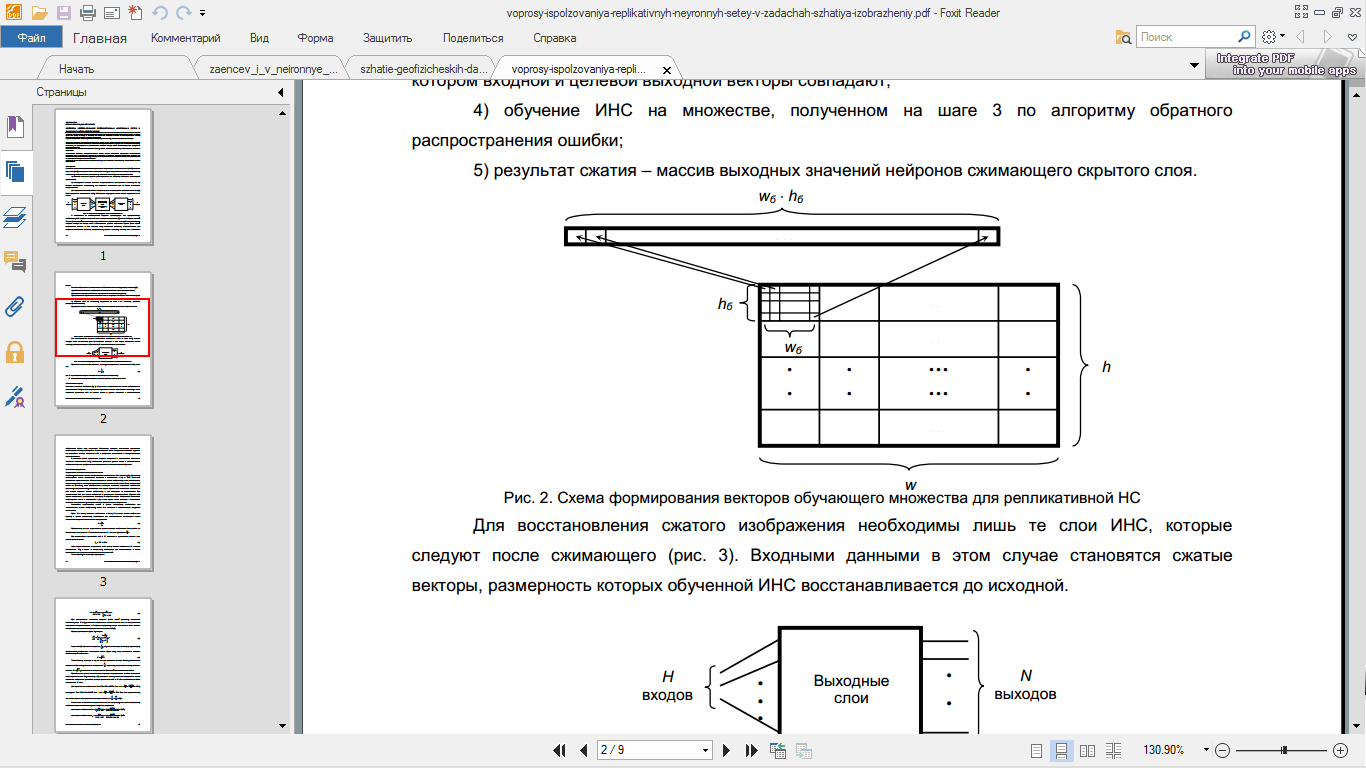
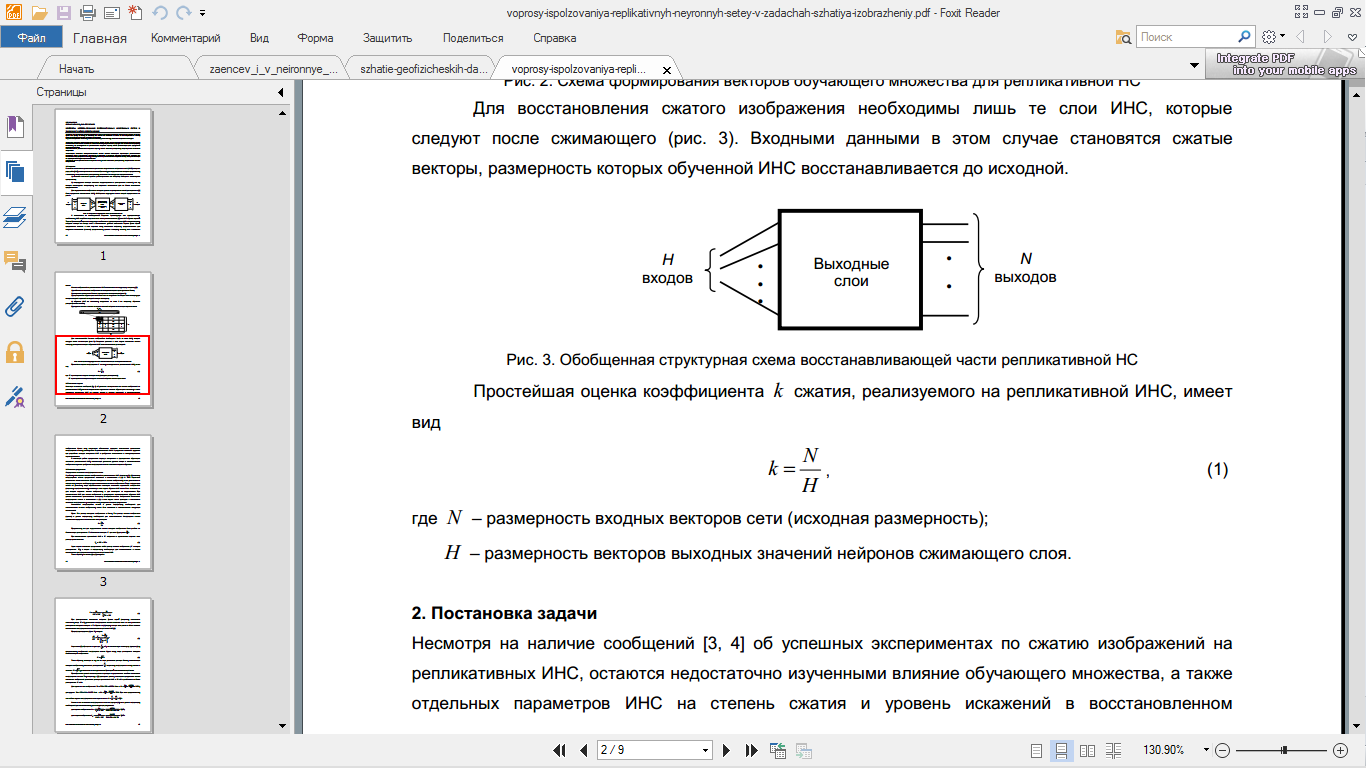


Рис. 10. Разбиение входного изображения на одинаковые блоки

4.2. Поэтапный алгоритм компрессии и декомпрессии

Поэтапно алгоритм работы можно сформулировать следующим образом:

1. Определение коэффициента сжатия сети – отношения количества входных нейронов к наименьшему количеству нейронов скрытого слоя. Определение алгоритма обучения и активационной функции нейронов
2. В общем случае, подготовка изображения к сжатию – нормирование значений яркости пикселов, т.е. деление значения яркости на 256 (в общем случае, по каждому RGB каналу) для подачи на вход сети.
3. Разбиение входных изображений на непересекающиеся блоки одинакового размера (w\*h) (Рис. 5)
4. Построчное разложение блоков – формирование входных векторов-строк
5. Подготовка обучающих шаблонов – различных блоков, размер которых совпадает с входными блоками сети (w\*h). Обучающие шаблоны можно как подготовить заранее, так и получить из изображения исходной выборки
6. Обучение ИНС на множестве, полученном на этапе 5
7. Разбиение сети на 2 составляющие – кодер и декодер
8. Результатом сжатия сети будет массив выходов последнего скрытого слоя кодера
9. Для восстановления сжатого изображения, массив значений выходов кодера подается на вход декодера
10. Изображение восстанавливается из массива векторов-строк в двумерный массив значений яркости пикселов

Простейшая оценка коэффициента k сжатия данной ИНС, имеет вид:

где N – размерность входных векторов сети (исходная размерность);

H – размерность векторов выходных значений нейронов сжимающего слоя.

5. Программная реализация алгоритма

На данный момент создана программная реализация алгоритма со следующими характеристиками:

1. Структура сети N-H-H-N. N-кол-во нейронов в входном\выходном слое. H – кол-во нейронов сжимающего слоя
2. Сеть работает с изображениями в 256 градациях серого
3. Есть возможность выбора размера входного блока и возможность выбора коэффициента сжатия
4. Активационная функция нейронов – сигмоидальная с настраиваемым параметром (без интерфейса)
5. Алгоритм обучения сети – Алгоритм обратного распространения ошибки
6. Добавлено улучшение обучения сети за счёт регулировки коэффициента скорости обучения в зависимости от изменения ошибки сети
7. Обучающие шаблоны генерируются с помощью входного изображения, случайным образом(выбор случайных блоков изображения)

Программная реализация представляет собой интегрированный кодер и декодер для оценки работы и обучения нейронной сети. С помощью диалогового окна пользователь выбирает тестовое изображение и задает коэффициент сжатия. Из блоков изображения генерируются обучающие шаблоны, сеть обучается по алгоритму обратного распространения ошибки, изображение разбивается на блоки и с помощью первой части нейронной сети(Coder) кодируется в массив байт. Далее происходит обратный процесс декодирования из массива байт с помощью второй части нейронной сети(Decoder). Декодированное изображение отображается на экране для визуальной оценки результата.

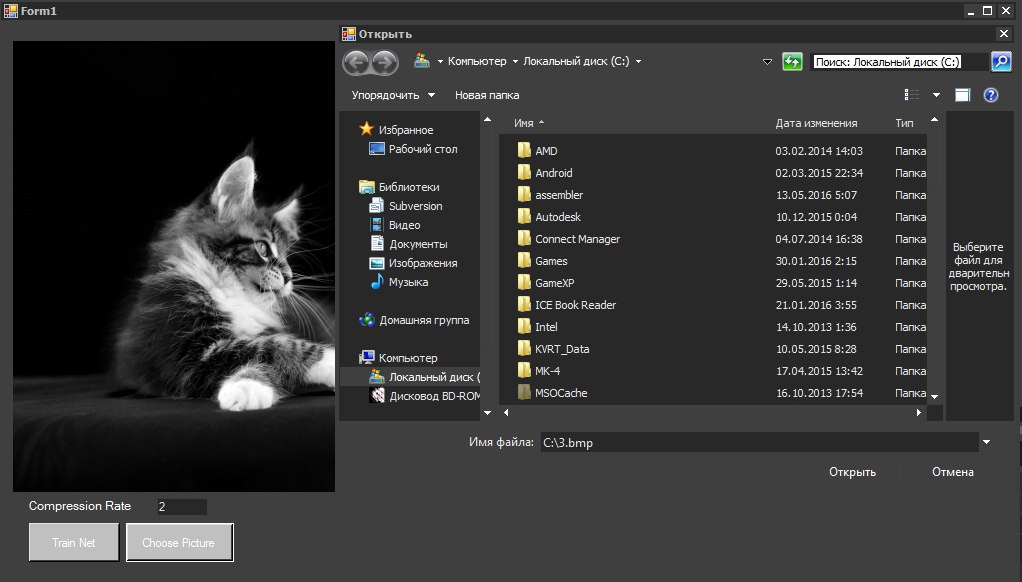


Рис 11. Интерфейс главного окна программы

5.1. Полученные результаты

Сеть обучалась в 89 из 100 тестов, результат кодирования сильно зависит от выбора обучающих шаблонов, необходим алгоритм для выбора шаблонов, т.к. случайный выбор шаблонов не всегда дает корректный результат, изображение часто сильно искажается из-за неудачного обучения сети.

Также обучаемость сети зависит от количества входных слоев(размерности шаблонов). Самая оптимальная размерность сети, полученная на данный момент – 64-n-n-64, при такой размерности удается получить относительно корректные результаты сжатия и времени обучения сети.

Удалось добиться сжатия данных в 32 раза, сеть обучается, но потери изображения существенные.

Также сравнивались сжатые декодированные изображения с их исходными образами по метрике [PSNR](http://sernam.ru/cod_11.php) (пиковое отношение сигнал/шум). Тесты производились на восьмибитных изображениях 1024\*1024 пикселя. При четырехкратном сжатии нейросетевым подходом PSNR~21-25Дб. При шестнадцатикратном сжатии нейросетевым подходом PSNR~20-23 Дб. Для алгоритма JPEG при аналогичных коэффициентах сжатия PSNR~61-64Дб



Рис. 12. Сравнение по метрике пиковое отношение сигнал\шум для декодированного изображения. Коэф. Сжатия = 16

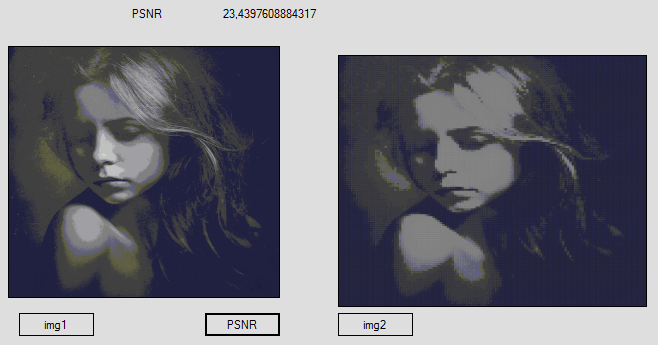


Рис. 12. Сравнение по метрике пиковое отношение сигнал\шум для декодированного изображения. Коэф. Сжатия = 8

5.2. Визуальная оценка полученных результатов:

1. Цветность изображения: двухбитное ч\б

Коэффициент сжатия: 4 раза

Количество входных шаблонов: 4

Размерность входных шаблонов: 4\*4 пикселя

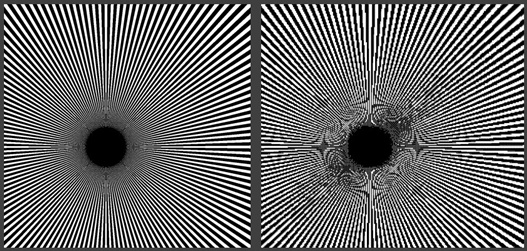


Рис 13. Исходное изображение слева. Справа результат после операции компрессии и декомпрессии

1. Цветность изображения: двухбитное ч\б

Коэффициент сжатия: 8 раза

Количество входных шаблонов: 4

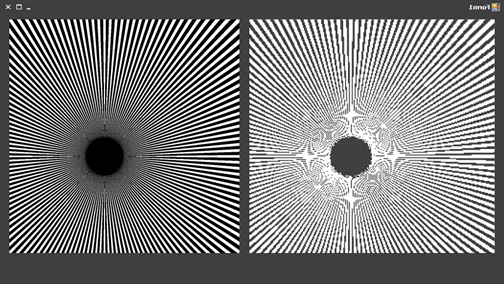
Размерность входных шаблонов: 4\*4 пикселя

Рис 14. Исходное изображение слева. Справа результат после операции компрессии и декомпрессии. Особенность – наблюдается эффект дробления изображения по сетке с ячейкой 4\*4 пикселя.

1. Цветность изображения: двухбитное ч\б

Коэффициент сжатия: 4 раза

Количество входных шаблонов: 4

Размерность входных шаблонов: 4\*4 пикселя

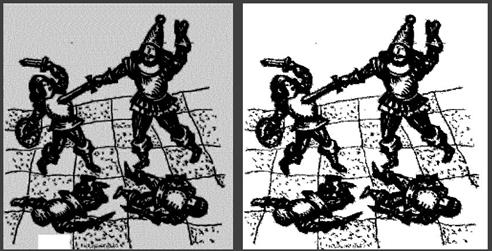


Рис 15. Справа результат после операции компрессии и декомпрессии. Особенность – наблюдается эффект шумоподавления серого равномерного шума с исходного изображения.

1. Цветность изображения: 8-битное ч\б

Коэффициент сжатия: 16 раз

Количество входных шаблонов: 6

Размерность входных шаблонов: 8\*8 пикселей



Рис 16. Исходное изображение слева. Справа результат после операции компрессии и декомпрессии. Доработанная версия программы для 8-битных изображений. Увеличен размер входного блока, увеличена обучающая выборка.

1. Цветность изображения: 8-битное ч\б

Коэффициент сжатия: 32 раза

Количество входных шаблонов: 6

Размерность входных шаблонов: 8\*8 пикселей



Рис 17. Исходное изображение слева. Справа результат после операции компрессии и декомпрессии.

# 

Заключение

В ходе работы рассмотрены основные существующие алгоритмы сжатия изображений, а также проработан метод, альтернативный существующим на данный момент подходам, связанный с использованием многослойного персептрона для сжатия и кодирования изображений. Также создана программная реализация, в которой тестируется не только разработанный алгоритм применения многослойного персептрона, но и реализована сама структура и процесс обучения нейронной сети. В результате, благодаря программной реализации, удалось убедиться, что метод сжатия с помощью многослойного персептрона работоспособен. Однако получить результаты, хотя бы визуально сравнимые с существующими алгоритмами сжатия при тех же коэффициентах сжатия, не удалось.

Алгоритм сжатия, лежащий в основе программной реализации, во многом основывается на удачном выборе обучающих шаблонов нейронной сети, которые должны быть разными для каждого изображения. Другой вариант – использование одних и тех же обучающих шаблонов для сжатия изображений, имеющих похожий визуальный характер (напр. Топографических карт).

Из-за отсутствия жестких теоретических обоснований, область нейронных сетей, во многом базируется на практике и исследовании посредством экспериментов.

Поэтому остается открытой задача улучшения алгоритма и варьирования множества параметров, начиная от параметра активационной функции, заканчивая множеством тренировочных шаблонов.

Список используемой литературы

1. Ватолин Д.С. Алгоритмы сжатия изображений. Методическое пособие / Издательский отдел факультета Вычислительной Математики и Кибернетики МГУ им. М.В.Ломоносова, 1999 г. — 76 с.
2. Гонсалес Р., Вудс Р. Цифровая обработка изображений; Монография / Москва: Техносфера, 2005. – 1072 с.
3. Тропченко А.Ю., Тропченко А.А. Методы сжатия изображений, аудиосигналов и видео: Учебное пособие / СПб: СПбГУ ИТМО, 2009. – 108 с.
4. Д.Сэломон. Сжатие данных, изображений и звука; Учебное пособие / Москва: Техносфера, 2004. - 368с.
5. <http://sernam.ru/lect_cod.php> Методы кодирования информации; Ульяновский Государственный технический университет
6. Просис Д. Фракталы и сжатие данных; статья / PC Magazine, November 8, 1994, p. 289
7. Ватолин Д. Фрактальное сжатие изображений; статья / Журнал «Открытые системы» 2011
8. Басалин, П.Д. Модели и методы интеллектуальной поддержки процессов принятия решений: Учебное пособие / Басалин П.Д., Безрук К.В., Радаева М.В. – Нижний Новгород: Нижегородский госуниверситет, 2011. – 129 с.
9. Заенцев И.В. Нейронные сети, основные модели: Учебное пособие к курсу «Нейронные сети» для студентов 5 курса магистратуры / И.В. Заенцев, Воронеж: Воронежский Государственный Университет, 1999 – 76 с

Приложение: Исходные коды разработанного программного обеспечения

public class BackPropagationLearning

{

private Network network;

private double learningRate = 0.1;/// скорость обучения

private double momentum = 0.0;

private double[][] neuronErrors = null;

private double[][][] weightsUpdates = null;

private double[][] thresholdsUpdates = null;

public double LearningRate

{

get { return learningRate; }

set

{

learningRate = Math.Max( 0.0, Math.Min( 1.0, value ) );

}

}

/// обозначает часть предыдущего обновления веса на текущей итерации [0,1]

public double Momentum

{

get { return momentum; }

set

{

momentum = Math.Max( 0.0, Math.Min( 1.0, value ) );

}

}

public BackPropagationLearning( Network network )

{

this.network = network;

neuronErrors = new double[network.Layers.Length][];

weightsUpdates = new double[network.Layers.Length][][];

thresholdsUpdates = new double[network.Layers.Length][];

// инициализировать изменения и ошибки

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

{

Layer layer = network.Layers[i];

neuronErrors[i] = new double[layer.Neurons.Length];

weightsUpdates[i] = new double[layer.Neurons.Length][];

thresholdsUpdates[i] = new double[layer.Neurons.Length];

// для каждого нейрона

for ( int j = 0; j < weightsUpdates[i].Length; j++ )

{

weightsUpdates[i][j] = new double[layer.InputsCount];

}

}

}

//итерация для данного шаблона

public double Run( double[] input, double[] output )

{

network.Compute( input );

double error = CalculateError( output );

CalculateUpdates( input );

UpdateNetwork( );

return error;

}

//эпоха обучения для всех шаблонов

public double RunEpoch( double[][] input, double[][] output )

{

double error = 0.0;

// run learning procedure for all samples

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

{

error += Run( input[i], output[i] );

}

// return summary error

return error;

}

private double CalculateError( double[] desiredOutput )

{

Layer layer, layerNext;

double[] errors, errorsNext;

double error = 0, e, sum;

double output;

int layersCount = network.Layers.Length;

// ошибка для последнего слоя

layer = network.Layers[layersCount - 1];

errors = neuronErrors[layersCount - 1];

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

{

output = layer.Neurons[i].Output;

e = desiredOutput[i] - output;

// ошибка на производную активационной функции

errors[i] = e \* SigmoidFunction.DerivativeY( output );

// квадрат и сумма ошибки

error += ( e \* e );

}

// посчитать ошибку для остальных слоев

for ( int j = layersCount - 2; j >= 0; j-- )

{

layer = network.Layers[j];

layerNext = network.Layers[j + 1];

errors = neuronErrors[j];

errorsNext = neuronErrors[j + 1];

// для всех нейронов в слое

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

{

sum = 0.0;

// для всех нейронов в след. слое

for ( int k = 0; k < layerNext.Neurons.Length; k++ )

{

sum += errorsNext[k] \* layerNext.Neurons[k].Weights[i];

}

errors[i] = sum \* SigmoidFunction.DerivativeY( layer.Neurons[i].Output );

}

}

// среднеквадратичная ошибка

return error / 2.0;

}

private void CalculateUpdates( double[] input )

{

Neuron neuron;

Layer layer, layerPrev;

double[][] layerWeightsUpdates;

double[] layerThresholdUpdates;

double[] errors;

double[] neuronWeightUpdates;

// 1 - вычислить обновления для 1 слоя

layer = network.Layers[0];

errors = neuronErrors[0];

layerWeightsUpdates = weightsUpdates[0];

layerThresholdUpdates = thresholdsUpdates[0];

// кэш часто используемых значений

double cachedMomentum = learningRate \* momentum;

double cached1mMomentum = learningRate \* ( 1 - momentum );

double cachedError;

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

{

neuron = layer.Neurons[i];

cachedError = errors[i] \* cached1mMomentum;

neuronWeightUpdates = layerWeightsUpdates[i];

for ( int j = 0; j < neuronWeightUpdates.Length; j++ )

{

neuronWeightUpdates[j] = cachedMomentum \* neuronWeightUpdates[j] + cachedError \* input[j];

}

layerThresholdUpdates[i] = cachedMomentum \* layerThresholdUpdates[i] + cachedError;

}

// 2 - для всех остальных слоев

for ( int k = 1; k < network.Layers.Length; k++ )

{

layerPrev = network.Layers[k - 1];

layer = network.Layers[k];

errors = neuronErrors[k];

layerWeightsUpdates = weightsUpdates[k];

layerThresholdUpdates = thresholdsUpdates[k];

// для всех нейронов слоя

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

{

neuron = layer.Neurons[i];

cachedError = errors[i] \* cached1mMomentum;

neuronWeightUpdates = layerWeightsUpdates[i];

// для всех связей нейрона

for ( int j = 0; j < neuronWeightUpdates.Length; j++ )

{

// вычислить обновление веса

neuronWeightUpdates[j] = cachedMomentum \* neuronWeightUpdates[j] + cachedError \* layerPrev.Neurons[j].Output;

}

// вычислить обновление порога

layerThresholdUpdates[i] = cachedMomentum \* layerThresholdUpdates[i] + cachedError;

}

}

}

private void UpdateNetwork( )

{

// current neuron

Neuron neuron;

Layer layer;

double[][] layerWeightsUpdates;

double[] layerThresholdUpdates;

double[] neuronWeightUpdates;

// для всех слоев

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

{

layer = network.Layers[i];

layerWeightsUpdates = weightsUpdates[i];

layerThresholdUpdates = thresholdsUpdates[i];

// для всех нейронов слоя

for ( int j = 0; j < layer.Neurons.Length; j++ )

{

neuron = layer.Neurons[j];

neuronWeightUpdates = layerWeightsUpdates[j];

for ( int k = 0; k < neuron.Weights.Length-1; k++ )

{

// веса

neuron.Weights[k] += neuronWeightUpdates[k];

}

// пороги

neuron.Threshold += layerThresholdUpdates[j];

}

}

}

}

}

**Neuron.cs**

public class Neuron

{

private double threshold = 0.0;

public double Threshold

{

get

{ return threshold; }

set

{ threshold = value; }

}

private int inputsCount = 0;

public int InputsCount { get { return inputsCount; } set { inputsCount = value; } }

private double[] weights;

public double[] Weights { get { return weights; } }

private double output = 0;

public double Output { get { return output; } }

public Neuron(int inputs)

{

inputsCount = Math.Max(1, inputs);

weights = new double[inputsCount];

Randomize();

}

public void Randomize()

{

for (int i = 0; i < inputsCount; i++) weights[i] = rand.NextDouble();

threshold = (rand.NextDouble())/3;

}

private static Random rand = new Random();

public double Compute(double[] input)

{

if (input.Length != inputsCount) throw new ArgumentException("Wrong length of the input vector.");

double sum = 0.0;

for (int i = 0; i < weights.Length; i++) sum += weights[i] \* input[i];

sum += threshold;

output = SigmoidFunction.Calculate(sum);

return output;

}

}

**Layer.cs**

public class Layer

{

private int inputsCount = 0;

public int InputsCount { get { return inputsCount; } set { inputsCount = value; } }

private int neuronsCount = 0;

private Neuron[] neurons;

public Neuron[] Neurons { get { return neurons; } }

private double[] output;

public double[] Output { get { return output; } }

public Layer( int neuronsCount, int inputsCount)

{

this.inputsCount = Math.Max( 1, inputsCount );

this.neuronsCount = Math.Max( 1, neuronsCount );

neurons = new Neuron[this.neuronsCount];

for (int i = 0; i < this.neuronsCount; i++)

{

neurons[i] = new Neuron(inputsCount);

}

}

public virtual double[] Compute(double[] input)

{

output = new double[neuronsCount];

for ( int i = 0; i < neurons.Length; i++ ) output[i] = neurons[i].Compute( input );

return output;

}

public virtual void Randomize( )

{

foreach ( Neuron neuron in neurons ) neuron.Randomize( );

}

}

**Network.cs**

public class Network

{

private int inputsCount;

public int InputsCount { get { return inputsCount; } }

private int layersCount;

private Layer[] layers;

private double[] output;

public Layer[] Layers { get { return layers; } }

public double[] Output { get { return output; } }

public Network( int inputsCount, params int[] InputLayers )

{

this.inputsCount = Math.Max( 1, inputsCount );

this.layersCount = InputLayers.Length;

layers = new Layer[layersCount];

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

{

this.layers[i] = new Layer(InputLayers[i], (i == 0) ? inputsCount : InputLayers[i - 1]);

}

this.Randomize();

}

public virtual double[] Compute( double[] input )

{

output = input;

for ( int i = 0; i < layers.Length; i++ ) { output = layers[i].Compute( output ); }

return output;

}

public virtual void Randomize( )

{

foreach ( Layer layer in layers ) layer.Randomize( );

}

}

**SigmoidFunction.cs**

public static class SigmoidFunction

{

private static double alpha = 0.5;

static SigmoidFunction() { }

public static void SetAlpha(double alphaIn)

{

alpha = alphaIn;

}

public static double Calculate(double x)

{

return (1 / (1 + Math.Exp(-alpha \* x)));

}

public static double DerivativeY(double y)

{

return (alpha \* y \* (1 - y));

}

}

**Form1.cs**

public partial class Form1 : Form

{

static int ExampleCount = 6;//количество шаблонов

static public int BlockSize = 8; //размеры пиксельного блока

static int CompressingRate=2;//Коэффициент сжатия

static int InputLayerSize = BlockSize \* BlockSize;

static int SmallLayerSize = InputLayerSize / CompressingRate;

static Network net;

static double[][][] InputImage;

static string filepath;

static Network decoder = new Network(SmallLayerSize, SmallLayerSize, InputLayerSize);

Network coder = new Network(InputLayerSize, SmallLayerSize);

public Form1()

{

InitializeComponent();

pictureBox1.SizeMode = PictureBoxSizeMode.Zoom;

pictureBox2.SizeMode = PictureBoxSizeMode.Zoom;

pictureBox1.Size = pictureBox2.Size;

}

public void TrainNet()

{

BackPropagationLearning trainer = new BackPropagationLearning(net);

double[][] input = GetTrainExample();

double[][] output = input;

// Переменная, сохраняющая значение ошибки сети на предыдущем шаге

double prErr = 10000000;

// Ошибка сети

double error = 100;

// Сначала скорость обучения должна быть высока

trainer.LearningRate = 1;

// Обучаем сеть пока ошибка сети станет небольшой

while (error > 1)

{

// Получаем ошибку сети

error = trainer.RunEpoch(input, output);

// Если ошибка сети изменилась на небольшое значения, в сравнении ошибкой предыдущей эпохи

if (Math.Abs(error - prErr) < 0.000000001)

{

// Уменьшаем коэффициент скорости обучения на 2

trainer.LearningRate /= 2;

if (trainer.LearningRate < 0.0001)

trainer.LearningRate = 0.0001;

}

prErr = error;

}

}

public double[][][] Decompress(byte[][][] bmpbyte)

{

for (int i = 0; i < decoder.Layers.Count(); i++) decoder.Layers[i] = net.Layers[i + (net.Layers.Count()-decoder.Layers.Count())];

int TaskSize = bmpbyte.Length;

double[][][] bmpbig = new double[TaskSize][][];

for (int y = 0; y < TaskSize; y++)

{

bmpbig[y] = new double[TaskSize][];

for (int x = 0; x < TaskSize; x++)

{

bmpbig[y][x] = new double[InputLayerSize];

double[] tempDoubleVector = new double[SmallLayerSize];

for (int i = 0; i < SmallLayerSize; i++) tempDoubleVector[i] = Convert.ToDouble(bmpbyte[y][x][i])/255;

bmpbig[y][x] = decoder.Compute(tempDoubleVector);

}

}

return bmpbig;

}//декодер и декомпрессия

public byte[][][] Compress(double[][][] bmp)

{

for (int i = 0; i < coder.Layers.Count(); i++) coder.Layers[i] = net.Layers[i];

int TaskSize = bmp.Length;

double[][][] bmpsmall = new double[TaskSize][][];

byte[][][] bmpbyte = new byte[TaskSize][][];

for (int y = 0; y < TaskSize; y++)

{

bmpbyte[y] = new byte[TaskSize][];

bmpsmall[y] = new double[TaskSize][];

for (int x = 0; x < TaskSize; x++)

{

bmpsmall[y][x] = new double[SmallLayerSize];

bmpbyte[y][x] = new byte[SmallLayerSize];

bmpsmall[y][x] = coder.Compute(bmp[y][x]);

for (int i = 0; i < SmallLayerSize; i++) bmpbyte[y][x][i] = Porog(bmpsmall[y][x][i]);

//for (int i = 0; i < SmallLayerSize; i++) bmpsmall[y][x][i] /= 255;

}

}

return bmpbyte;

}//создание кодера и компрессия

public byte Porog(double input)

{

return Convert.ToByte(input\*255);

}//пороговая функция, если input >0.5 то на выход 1, иначе 0

public double[][] GetTrainExample()

{

Bitmap bmp = new Bitmap(filepath);

Random random = new Random();

double[][] Example = new double[ExampleCount][];

for (int k = 0; k < ExampleCount; k++)

{

Example[k] = new double[BlockSize\*BlockSize];

int x = random.Next(0, bmp.Height / BlockSize); //случайный блок изображения на обучение

int y = random.Next(0, bmp.Height / BlockSize);

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

{

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

{

Color pixel = bmp.GetPixel(x \* BlockSize + j, y \* BlockSize + i);

Example[k][j + i \* BlockSize] = (pixel.R \* 77 + pixel.G \* 150 + pixel.B \* 28) / 255;

Example[k][j + i \* BlockSize] /= 255;

}

}

}

return Example;

}

public double[][][] BMPInput()

{

Bitmap bmp = new Bitmap(filepath);

//PixelFormat pxf = bmp.PixelFormat;

//Rectangle rect = new Rectangle(0, 0, bmp.Width, bmp.Height);

//BitmapData bmpData = bmp.LockBits(rect, ImageLockMode.ReadWrite, pxf);

//int numBytes = bmpData.Stride \* bmp.Height;

//byte[] rgbValues = new byte[numBytes];

//IntPtr ptr = bmpData.Scan0;

//Marshal.Copy(ptr, rgbValues, 0, numBytes);

double[][][] Example = new double[bmp.Width/BlockSize][][];

for (int y = 0; y < bmp.Width / BlockSize; y++)

{

Example[y] = new double[bmp.Width / BlockSize][];

for (int x = 0; x < bmp.Width / BlockSize; x++)

{

Example[y][x] = new double[BlockSize \* BlockSize];

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

{

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

{

Color pixel = bmp.GetPixel(x \* BlockSize + j, y \* BlockSize + i);

Example[y][x][j + i \* BlockSize] = (pixel.R \* 77 + pixel.G \* 150 + pixel.B \* 28) / 255;

Example[y][x][j + i \* BlockSize] /= 255;

}

}

}

}

return Example;

}//Ввод из BMP файла

public Bitmap GetBMPFromJuggedArray(double[][][] Example)

{

int BmpPartSize = Example.Length;

Bitmap bmp = new Bitmap(BmpPartSize \* BlockSize, BmpPartSize \* BlockSize);

for (int y = 0; y < BmpPartSize; y++)

{

for (int x = 0; x < BmpPartSize; x++)

{

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

{

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

{

int color = Convert.ToByte(Example[y][x][j + i \* BlockSize]\*255);

bmp.SetPixel(j + x \* BlockSize, i + y \* BlockSize, Color.FromArgb(color,color,color));

}

}

}

}

Bitmap bmp2 = CopyToBpp(bmp, 8);

return bmp2;

}//получить BMP из массива

private void Start\_Click(object sender, EventArgs e)

{

CompressingRate = Convert.ToInt32(CompRate.Text);

InputLayerSize = BlockSize \* BlockSize;

SmallLayerSize = InputLayerSize / CompressingRate;

net = new Network(InputLayerSize, SmallLayerSize, SmallLayerSize, InputLayerSize);

TrainNet();

pictureBox1.Image = GetBMPFromJuggedArray(Decompress(Compress(InputImage)));

}

private void button1\_Click(object sender, EventArgs e)

{

openFileDialog1.ShowDialog();

filepath = openFileDialog1.FileName;

pictureBox2.Image = new Bitmap(filepath);

InputImage = BMPInput();

}

private void saveBtn\_Click(object sender, EventArgs e)

{

saveFileDialog1.ShowDialog();

if(saveFileDialog1.FileName != "")

{

System.IO.FileStream fs = (System.IO.FileStream)saveFileDialog1.OpenFile();

this.pictureBox1.Image.Save(fs,System.Drawing.Imaging.ImageFormat.Bmp);

fs.Close();

}

}

}

1. Просис Д. Фракталы и сжатие данных; статья / PC Magazine, November 8, 1994, p. 289 [↑](#footnote-ref-1)
2. Ватолин Д. Фрактальное сжатие изображений; статья / Журнал «Открытые системы» 2011 [↑](#footnote-ref-2)
3. Басалин, П.Д. Модели и методы интеллектуальной поддержки процессов принятия решений: Учебное пособие / Басалин П.Д., Безрук К.В., Радаева М.В. – Нижний Новгород: Нижегородский госуниверситет, 2011. – 129 с. [↑](#footnote-ref-3)
4. Басалин, П.Д. Модели и методы интеллектуальной поддержки процессов принятия решений: Учебное пособие / Басалин П.Д., Безрук К.В., Радаева М.В. – Нижний Новгород: Нижегородский госуниверситет, 2011. – 129 с. [↑](#footnote-ref-4)