Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет Компьютерных систем и сетей
Кафедра Информатики

К защите допустить:
Заведующий кафедрой информатики
_____ Н. А. Волорова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломной работе
на тему:

МНОГОСЛОЙНЫЕ ПЕРЦЕПТРОНЫ ДЛЯ СЖАТИЯ ИЗОБРАЖЕНИЙ

БГУИР ДР **1-31 03 04 07 072** ПЗ

Студент: А. А. Сафонов
Руководитель: Е. В. Кукар
Консультанты:
от кафедры информатики Е. В. Кукар
по экономической части Т. Л. Слюсарь
по охране труда Т. В. Гордейчук
Нормоконтролёр: В. В. Шиманский

Рецензент:

Минск 2015

Министерство образования Республики Беларусь
Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет	КСиС	Кафедра	Информатики
Специальность	1-31 03 04	Специализация	07

УТВЕРЖДАЮ

_____ Н. А. Волорова
«___» _____ 2015 г.

ЗАДАНИЕ
по дипломной работе студента
Сафонова Анатолия Анатольевича
(фамилия, имя, отчество)

1. Тема проекта (работы): Многослойные перцептроны для сжатия изображений

утверждена приказом по университету от 25.03.2015 г. №502-с

2. Срок сдачи студентом законченной работы: 1 июня 2015 года

3. Исходные данные к проекту (работе): Платформа .NET Framework 4.5; план проведения исследования; язык программирования C#; литература по нейронным сетям; литература по методам сжатия данных с потерями.

4. Содержание пояснительной записки (перечень подлежащих разработке вопросов): 1 Обзор предметной области
2 Используемые технологии
3 Постановка задачи
4 Разработка алгоритма
5 Архитектура прототипа
6 Анализ полученных данных
7 Обеспечение безопасных условий труда инженера-программиста при проведении исследования многослойных перцептронов для сжатия изображений
8 Технико-экономическое обоснование эффективности разработки и использования многослойных перцептронов для сжатия изображений
Заключение
Список используемых источников
Приложение А Листинг программного средства

5. Перечень графического материала (с точным указанием обязательных чертежей):

6. Содержание задания по технико-экономическому обоснованию:
Технико-экономическое обоснование эффективности разработки и использования многослойных перцептронов для сжатия изображений
Задание выдал: / Т. Л. Слюсарь /

7. Содержание задания по охране труда и экологической безопасности, ресурсо- и энергосбережению:
Обеспечение безопасных условий труда инженера-программиста при проведении исследования многослойных перцептронов для сжатия изображений
Задание выдал: / Т. В. Гордейчук /

КАЛЕНДАРНЫЙ ПЛАН

№№ п/п	Наименование этапов дипломного проекта (работы)	Объем этапа, %	Срок выполнения этапов	Примечание
1	Анализ предметной области,			
	разработка плана исследования	5	03.02 – 11.02	
2	Разработка алгоритма	10	12.02 – 23.02	
3	Проектирование структуры			
	прототипа	15	24.02 – 10.03	
4	Разработка прототипа	30	11.03 – 13.04	
5	Сбор и анализ полученных			
	материалов	30	14.04 – 27.04	
6	Оформление пояснительной			
	записки и графического			
	материала	10	28.04 – 30.05	

Дата выдачи задания: Руководитель / Е. В. Кукар /

Задание принял к исполнению / А. А. Сафонов /

РЕФЕРАТ

Ключевые слова: НЕЙРОННЫЕ СЕТИ; СЖАТИЕ С ПОТЕРЯМИ; ЧАСТОТА ДИСКРЕТИЗАЦИИ; КРИТЕРИИ СРАВНЕНИЯ ИЗОБРАЖЕНИЙ.

Дипломная работа выполнена на 6 листах формата А1 с пояснительной запиской на 76 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 8 глав, 32 рисунков, 2 таблиц, 44 формулы, 21 литературный источник.

Целью дипломной работы является разработка алгоритма, пригодного для решения практических задач, возникающих в реальных проектах, связанных с обработкой и передачи изображений и другой мультимедийной информации.

Для достижения цели дипломной работы был разработан алгоритм, предназначенный для сжатия графической или любой другой мультимедийной информации. Созданные с использованием данного алгоритма библиотеки могут быть использованы в реальных проектах, использующих методы сжатия с потерями для мультемидийной информации. В работе приведена библиотека реализующая алгоритм на платформе .NET.

В разделе технико-экономического обоснования был произведён расчёт затрат на разработку данного алгоритма, а также прибыли от разработки, получаемой разработчиком. Проведённые расчёты показали экономическую целесообразность проведения исследования.

Пояснительная записка включает раздел по охране труда, в котором была произведена оценка безопасных условий труда на предприятии, где частично разрабатывалась данная дипломная работа.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В последнее десятилетие наблюдается возросший интерес к нейронным сетям, которые успешно применяются в самых различных областях: медицина, физика, химия, системы безопасности.

Нейронные сети вошли в практику везде, где требуется решать задачи прогнозирования, классификации или управления. Такие направления использования во многом обусловлены несколькими причинами: богатыми возможностями и простота в использовании. Нейронные сети – исключительно мощный метод моделирования, позволяющий воспроизводить чрезвычайно сложные зависимости. В частности, нейронные сети являются нелинейными по своей природе. На протяжении длительного времени линейное программирование было основным методом моделирования в большинстве областей, потому что для него хорошо разработаны процедуры оптимизации. Следовательно, нейронные сети могут решать некоторые типы задач, с которыми возникают сложности при применении методов линейного программирования.

Нейронные сети учатся на примерах. Пользователь нейронной сети подбирает представительные данные, а затем запускает алгоритм обучения, который автоматически воспринимает структуру данных. При этом от пользователя, требуется набор эвристических знания о том, как следует отбирать и подготавливать данные для обучения, подбирать архитектуру сети и правильно интерпретировать результаты.

Нейронные сети привлекательны с интуитивной точки зрения, ибо они основаны на примитивной биологической модели нервных систем. В будущем развитие таких нейробиологических моделей может привести к созданию действительно мыслящих компьютеров.

Целью дипломной работы является исследования возможностей использования нейронных сетей для сжатия графической информации, оценка качества полученных изображений, степени сжатия и скорости обработки данных.

1 ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

В данном разделе будет произведён обзор предметной области задачи, решаемой в рамках дипломной работы; рассмотрены вопросы о сущности нейронных сетей и принципе их работ.

1.1 Нейронные сети

Нейронные сети – это одно из направлений исследований в области искусственного интеллекта, основанное на попытках воспроизвести нервную систему человека, а именно: способность нервной системы обучаться и исправлять ошибки, что должно позволить смоделировать, хотя и достаточно грубо, работу человеческого мозга [1].

Биологический нейрон – это специальная клетка, которая структурно состоит из ядра, тела клетки и отростков. Одной из ключевых задач нейрона является передача электрохимического импульса по всей нейронной сети через доступные связи с другими нейронами. Притом, каждая связь характеризуется некоторой величиной, называемой силой синаптической связи. Эта величина определяет, что произойдет с электрохимическим импульсом при передаче его другому нейрону: либо он усилится, либо он ослабится, либо останется неизменным.

Биологическая нейронная сеть обладает высокой степенью связности: на один нейрон может приходиться несколько тысяч связей с другими нейронами. Но, это приблизительное значение и в каждом конкретном случае оно разное. Передача импульсов от одного нейрона к другому порождает определенное возбуждение всей нейронной сети. Величина этого возбуждения определяет реакцию нейронной сети на какие-то входные сигналы. Например, встреча человека со старым знакомым может привести к сильному возбуждению нейронной сети, если с этим знакомым связаны какие-то яркие и приятные жизненные воспоминания. В свою очередь сильное возбуждение нейронной сети может привести к учащению сердцебиения, более частому морганию глаз и к другим реакциям. Встреча же с незнакомым человеком для нейронной сети пройдет практически незаметной, а значит и не вызовет каких-либо сильных реакций.

Искусственная нейронная сеть представляет из себя математическую модель, построенную по принципу организации и функционирования биологических нейронных сетей. Сеть состоит из системы соединенных и взаимодействующих между собой простых процессоров. Эти процессоры обрабатывают входные сигналы и передает остальным. Обычно сложных опе-

раций они не производят, однако соединенные в большую сеть, могут выполнять сложные задачи.

Каждый нейрон выполняет небольшой объем работ — например, суммирует пришедшие на него сигналы с некоторыми весовыми коэффициентами и дополнительно нелинейно преобразует эту взвешенную сумму входных данных. Другим распространённым вариантом является нейрон-детектор, выдающий высокий выходной сигнал при малых отличиях своих входных сигналов от некоторого запомненного эталона, и низкий выходной сигнал при существенных отличиях.

Нейроны группируются в последовательность слоёв; входные сигналы поступают на первый слой и последовательно проходят через все слои до последнего, выходного слоя сети. Но бывают и рекуррентные структуры, обеспечивающие циркуляцию некоторого набора внутренних сигналов.

Нейроны, составляющие некоторый слой сети, работают в параллельном режиме. Изменение числа слоёв и количества нейронов в слоях позволяет гибко или максимально эффективно настроить общий параллельно-последовательный объем вычислений на особенности применяемой вычислительной техники.

Процесс работы с сетью представляет собой движение потока внешних сенсорных данных от входных нейронов к выходным и преобразование этих данных. В общем случае поток сигналов может формировать и перекрёстные, и обратные связи [2].

Нейронная сеть способна обучаться решению задач, для которых у человека не существует формализованных, быстрых или работающих с приемлемой точностью теоретических или эмпирических алгоритмов. Наряду с обучающими данными требуется лишь задать некоторый критерий качества решения задачи, который нейронная сеть при своём обучении должна будет минимизировать или оптимизировать.

Структура сети может быть адаптирована к задаче. В сеть могут быть включены дополнительные нейроны и даже слои нейронов, если исходно она была не способна обеспечить нужную точность решения. Из нейронной сети могут быть исключены лишние нейроны и связи между ними, если исходная сеть была избыточна. Сеть может сама выделить наиболее информативные для задачи входные сигналы, отбросить неинформативные, шумовые сигналы и в итоге повысить надежность решения. При этом коррекция размеров нейронной сети не приводит к полному забыванию ранее сформированных при обучении навыков, что ускоряет последующий процесс обучения [2].

Нейронные сети не программируются в привычном смысле этого слова, они обучаются. Возможность обучения — одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Технически обучение заключается в нахождении коэффициентов связей между нейронами. В процессе обучения нейронная сеть способна выявлять сложные зависимости между входными данными и выходными, а также выполнять обобщение. Это значит, что в случае успешного обучения сеть сможет вернуть верный результат на основании данных, которые отсутствовали в обучающей выборке, а также неполных и повреждённых, частично искаженных данных.

1.2 Типы нейронных сетей

Существует несколько типов нейронных сетей: многослойный перцептрон, рекуррентный перцептрон, ассоциативная память, сверточные нейронные сети, сети Кохонена и спайковые сети.

Многослойный перцептрон самая известная и архитектура, в которой идут подряд несколько слоев нейронов — входной, один или несколько скрытых слоев, и выходной слой. Почти всегда обучается методом обратного распространения ошибки — что автоматически означает, что мы должны предоставить для обучения набор пар «входной вектор — правильный выход»[3]. Тогда входной вектор отправится на вход сети, последовательно будут рассчитаны состояния всех промежуточных нейронов, и на выходе образуется выходной вектор, который мы и сравним с правильным. Расхождение даст нам ошибку, которую можно распространить обратно по связям сети, вычислить вклад в итоговую ошибку каждого нейрона, и скорректировать его веса, чтобы ее исправить. Повторив эту процедуру много тысяч раз, возможно выйдет обучить сеть.

Сеть такого типа очень хорошо справляется с задачами, где ответ действительно зависит только от того, что мы даем на вход сети, и никак не зависит от истории входов (т.е. это не последовательность информации со скрытой закономерностью; ответ не зависит или слабо зависит от высоких степеней или произведений параметров — функции этого типа сеть строить почти не умеет; в наличии есть достаточно много примеров (желательно иметь не менее сотни примеров на каждую связь сети), или у вас есть большой опыт борьбы с эффектом специализации. Это связано с тем, что имея много коэффициентов, сеть может банально запомнить много конкретных примеров, и выдавать на них отличный результат — но ее прогнозы не будут иметь ничего общего с реальностью в случае, если дать на вход примеры

не из обучающей выборки.

Слабые стороны — неумение работать с динамическими процессами, необходимость большой обучающей выборки. Рекуррентный перцептрон на первый взгляд поход на обычный перцептрон, единственное существенное отличие состоит в том, что его выходные сигналы попадают ему же на входы, и участвуют в обработке уже следующего входного вектора сигналов.

То есть, в случае рекуррентного перцептрона имеет место не набор отдельных, ничем не связанных образов, а некоторый процесс, и значение имеют не только сами входы, но и то, в какой последовательности они поступают. Из-за этого возникают отличия в методе обучения — используется то же самое обратное распространение ошибки, но для того, чтобы ошибка попала по рекуррентной связи в прошлое, используются разные ухищрения. В остальном же ситуация похожа на обычный перцептрон — для обучения нужно иметь достаточно длинную последовательность пар вход-выход, которую нужно много раз прогнать через сеть, чтобы ее обучить или же иметь под рукой математическую модель искомого процесса, которую можно гонять во всевозможных условиях, и в режиме реального времени давать результаты сети для обучения. Сеть такого типа обычно хорошо решает задачи управления динамическими процессами начиная от классической задачи стабилизации перевернутого маятника, и до любых систем, которыми вообще хоть как-то получается управлять, предсказания динамических процессов, и вообще всего, где помимо явно наблюдаемого входа у системы есть некоторое внутреннее состояние, которое не совсем понятно, как использовать.

Ассоциативная память — это широкий класс сетей, которые в той или иной степени напоминают архитектуру Хопфилда, которая состоит из одного слоя нейронов, выходы которого поступают на его входы в следующий момент времени. Этот слой служит и входом сети. В начальный момент выходы нейронов принимаются равными входному вектору, и ее выходом — значения на нейронах, образовавшиеся в конце работы, считаются ответом сети. Данный тип сетей меняет свои состояния с течением времени до тех пор, пока состояние не перестанет меняться. Свойства весовой матрицы выбраны таким образом, чтобы устойчивое состояние всегда гарантированно достигалось и обычно это происходит за несколько шагов. Такая сеть помнит некоторое количество векторов, и при подаче на вход любого вектора, может определить, на какой из запомненных он более всего похож — отсюда и название [4].

Двухслойная модификация этой сети, а именно гетероассоциативная

память, может запоминать вектора не по одному, а по парам разной размерности. Сети такого типа хорошо справляются с задачами, где нужно определить похожесть вектора на один из стандартных запомненных. Собственно, это единственный класс задач, где они хороши. Также конкретно сеть Хопфилда может использоваться для решения задач оптимизации, например, задачи коммивояжёра, однако ее эффективность в этой области под вопросом. К сильным сторонам можно отнести — очень быстрое обучение, возможность удалить образ из памяти или добавить в память, не затронув остальные, некоторые свойства такой памяти напоминают свойства мозга, и их изучение интересно с такой позиции [4]. Слабые стороны — очень узкий класс решаемых задач, неумение обобщать примеры, максимальный объем памяти жестко связан с размерностью запоминаемого вектора, ввиду особенностей построения. Перспективы:

– разработана ядерная ассоциативная память, которая способна к обобщению образов, и имеет неограниченный объем памяти, потому что сеть растет по мере заполнения;

– разработана динамическая ассоциативная память, которая запоминает не отдельные образы, а определенные последовательности образов, и поэтому может применяться для распознавания элементов динамических процессов;

– динамическая ассоциативная память демонстрирует способность к генерации отклика, содержащего разные элементы запомненных последовательностей при подаче входного сигнала, соответствующего одновременно разным последовательностям, что, возможно, является некоторой грубой моделью творчества человека;

– гибрид ядерной и динамической ассоциативной памяти может дать новое качество в распознавании последовательностей — например, в распознавании речи.

Нейронные сети Кохонена — класс нейронных сетей, основным элементом которых является слой Кохонена. Слой Кохонена состоит из адаптивных линейных сумматоров. Как правило, выходные сигналы слоя обрабатываются по правилу «победитель получает всё»: наибольший сигнал превращается в единичный, остальные обращаются в ноль [5].

По способам настройки входных весов сумматоров и по решаемым задачам различают много разновидностей сетей Кохонена. Наиболее известные из них:

– сети векторного квантования сигналов, тесно связанные с простейшим базовым алгоритмом кластерного анализа (метод динамических ядер

или K-средних);

– самоорганизующиеся карты Кохонена;

– сети векторного квантования, обучаемые с учителем.

Самоорганизующаяся карта Кохонена — нейронная сеть с обучением без учителя, выполняющая задачу визуализации и кластеризации. Идея сети предложена финским учёным Т. Кохоненом. Является методом проецирования многомерного пространства в пространство с более низкой размерностью (чаще всего, двумерное), применяется также для решения задач моделирования, прогнозирования и др. Является одной из версий нейронных сетей Кохонена. Самоорганизующиеся карты Кохонена используются для решения таких задач, как моделирование, прогнозирование, выявление наборов независимых признаков, сжатие информации, а также для поиска закономерностей в больших массивах данных. Наиболее часто описываемый алгоритм применяется для кластеризации данных.

Самоорганизующаяся карта состоит из компонентов, называемых узлами или нейронами. Их количество задаётся аналитиком. Каждый из узлов описывается двумя векторами. Первый — т. н. вектор веса m, имеющий такую же размерность, что и входные данные. Второй — вектор r, представляющий собой координаты узла на карте. Карта Кохонена визуально отображается с помощью ячеек прямоугольной или шестиугольной формы; последняя применяется чаще, поскольку в этом случае расстояния между центрами смежных ячеек одинаковы, что повышает корректность визуализации карты [5].

Изначально известна размерность входных данных, по ней некоторым образом строится первоначальный вариант карты. В процессе обучения векторы веса узлов приближаются к входным данным. Для каждого наблюдения выбирается наиболее похожий по вектору веса узел, и значение его вектора веса приближается к наблюдению. Также к наблюдению приближаются векторы веса нескольких узлов, расположенных рядом, таким образом если в множестве входных данных два наблюдения были схожи, на карте им будут соответствовать близкие узлы. Циклический процесс обучения, перебирающий входные данные, заканчивается по достижении картой допустимой (заранее заданной аналитиком) погрешности, или по совершении заданного количества итераций. Таким образом, в результате обучения карта Кохонена классифицирует входные данные на кластеры и визуально отображает многомерные входные данные в двумерной плоскости, распределяя векторы близких признаков в соседние ячейки и раскрашивая их в зависимости от анализируемых параметров нейронов. В результате работы

алгоритма получаются следующие карты: карта входов нейронов — визуализирует внутреннюю структуру входных данных путем подстройки весов нейронов карты. Обычно используется несколько карт входов, каждая из которых отображает один из них и раскрашивается в зависимости от веса нейрона [5]. На одной из карт определенным цветом обозначают область, в которую включаются приблизительно одинаковые входы для анализируемых примеров; карта выходов нейронов — визуализирует модель взаимного расположения входных примеров.

Очерченные области на карте представляют собой кластеры, состоящие из нейронов со схожими значениями выходов. Специальные карты — это карта кластеров, полученных в результате применения алгоритма самоорганизующейся карты Кохонена, а также другие карты, которые их характеризуют.

1.3 Многослойные перцептроны

Многослойными персептронами называют нейронные сети прямого распространения. Входной сигнал в таких сетях распространяется в прямом направлении, от слоя к слою. Многослойный персептрон в общем представлении состоит из следующих элементов:

- множества входных узлов, которые образуют входной слой;
- одного или нескольких скрытых слоев вычислительных нейронов;
- одного выходного слоя нейронов.

Искуусственный нейрон — узел искусственной нейронной сети, являющийся упрощённой моделью естественного нейрона. Математически, искусственный нейрон обычно представляют как некоторую нелинейную функцию от единственного аргумента — линейной комбинации всех входных сигналов. Данную функцию называют функцией активации. Полученный результат посылается на единственный выход. Такие искусственные нейроны объединяют в сети — соединяют выходы одних нейронов с входами других. Искусственные нейроны и сети являются основными элементами идеального нейрокомпьютера Многослойный персептрон представляет собой обобщение однослойного персептрона Розенблатта [6].

Примером многослойного персептрона является следующая модель нейронной сети(рисунок 1.1).

Многослойные персептроны успешно применяются для решения разнообразных сложных задач и имеют следующих три отличительных признака.

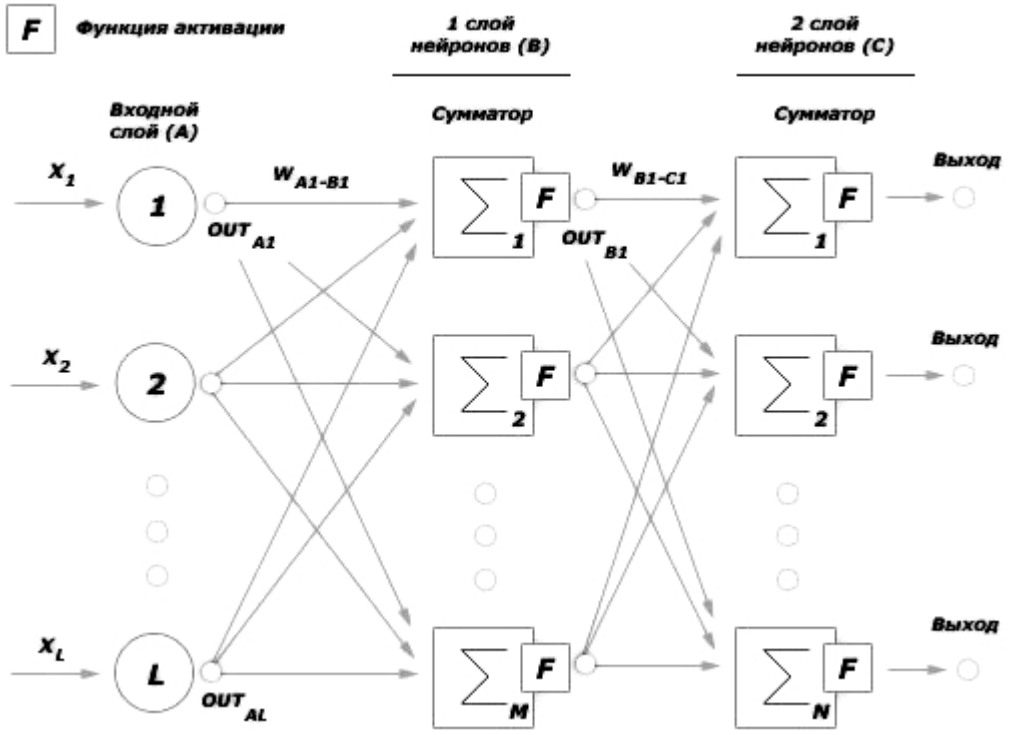

Рисунок *1.1* — Схема двухслойного перцептрона

Свойство 1. Каждый нейрон сети имеет нелинейную функцию активации

Важно подчеркнуть, что такая нелинейная функция должна быть гладкой (т.е. всюду дифференцируемой), в отличие от жесткой пороговой функции, используемой в персептроне Розенблатта. Самой популярной формой функции, удовлетворяющей этому требованию, является сигмоидальная. Примером сигмоидальной функции может служить логистическая функция, задаваемая следующим выражением:

$$f(x) = \frac{2}{1 + e^{ax}} - 1 \tag{1.1}$$

где a — параметр наклона сигмоидальной функции. Изменяя этот параметр, можно построить функции с различной крутизной.

Наличие нелинейности играет очень важную роль(рисунок 1.1), так как в противном случае отображение «вход-выход» сети можно свести к обычному однослойному персептрону Монотонно возрастающая всюду дифференцируемая S-образная нелинейная функция с насыщением. Сигмоид

позволяет усиливать слабые сигналы и не насыщаться от сильных сигналов. Гроссберг (1973 год) обнаружил, что подобная нелинейная функция активации решает поставленную им дилемму шумового насыщения. Слабые сигналы нуждаются в большом сетевом усилении, чтобы дать пригодный к использованию выходной сигнал. Однако усилительные каскады с большими коэффициентами усиления могут привести к насыщению выхода шумами усилителей, которые присутствуют в любой физически реализованной сети. Сильные входные сигналы в свою очередь также будут приводить к насыщению усилительных каскадов, исключая возможность полезного использования выхода. Таким образом одна и та же сеть может обрабатывать как слабые, так и сильные сигналы.

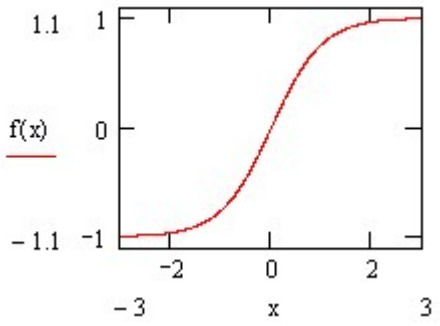

Рисунок *1.2* — Биполярная сигмоидальная функция

Свойство 2. Несколько скрытых слоев

Многослойный персептрон содержит один или несколько слоев скрытых нейронов, не являющихся частью входа или выхода сети. Эти нейроны позволяют сети обучаться решению сложных задач, последовательно извлекая наиболее важные признаки из входного образа.

Свойство 3. Высокая связность

Многослойный персептрон обладает высокой степенью связности, реализуемой посредством синаптических соединений. Изменение уровня связности сети требует изменения множества синаптических соединений или их весовых коэффициентов.

Комбинация всех этих свойств наряду со способностью к обучению на собственном опыте обеспечивает вычислительную мощность многослойного персептрона [6]. Однако эти же качества являются причиной неполноты современных знаний о поведении такого рода сетей: распределенная форма нелинейности и высокая связность сети существенно усложняют теоретический анализ многослойного персептрона.

2 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

Выбор технологий является важным предварительным этапом разработки сложных информационных систем. Платформа и язык программирования, на котором будет реализована система, заслуживает большого внимания, так как исследования показали, что выбор языка программирования влияет на производительность труда программистов и качество создаваемого ими кода [7, с. 59].

Основываясь на опыте работы имеющихся программистов разрабатывать ПО целесообразно на платформе .NET. Приняв во внимание необходимость обеспечения доступности дальнейшей поддержки ПО, возможно, другой командой программистов, целесообразно не использовать малоизвестные и сложные языки программирования. С учетом этого фактора выбор языков программирования сужается до четырех официально поддерживаемых Microsoft и имеющих изначальную поддержку в Visual Studio 2013: Visual C++/CLI, C#, Visual Basic .NET и F#. Необходимость использования низкоуровневых возможностей Visual C++/CLI в разрабатываемом ПО отсутствует, следовательно данный язык можно исключить из списка кандидатов. Visual Basic .NET уступает по удобству использования двум другим кандидатам из нашего списка. Оставшиеся два языка программирования C# и F# являются первостепенным, элегантными, мультипарадигменными языками программирования для платформы .NET. Так как на данном этапе нет необходимости выполнять много операций по обработке данных, F#тоже не подходит. Таким образом, с учетом вышеперечисленных факторов, целесообразно остановить выбор на следующих технологиях:

– операционная система Windows 7;
– платформа разработки .NET;
– язык программирования C#.

Поддержка платформой .NET различных языков программирования позволяет использовать язык, который наиболее просто и «красиво» позволяет решить возникающую задачу. Разрабатываемое программное обеспечение в некоторой степени использует данное преимущество платформы. Далее проводится характеристика используемых технологий и языков программирования более подробно.

2.1 Программная платформа .NET

Программная платформа .NET является одной из реализаций стандарта ECMA-335 [8] и является современным инструментом создания клиент-

ских и серверных приложений для операционной системы Windows. Платформа .NET — это один из компонентов системы Windows. Он позволяет создавать и использовать приложения нового поколения. Назначение платформы .NET:

– создание целостной объектно-ориентированной среды программирования допускающей различные варианты реализации: код может храниться и выполняться локально; выполняться локально, а распространяться через Интернет; или выполняться удаленно;

– предоставление среды выполнения кода, в которой число конфликтов при развертывании программного обеспечения и управлении версиями будет сведено к минимуму;

– обеспечение безопасности выполнения кода в среде - в том числе кода, созданного неизвестным разработчиком или разработчиком с частичным доверием;

– предоставление среды выполнения кода, позволяющей устранить проблемы, связанные с производительностью сред на основе сценариев или интерпретации;

– унификация работы разработчиков в совершенно разных приложениях: как в приложениях Windows, так и в веб-приложениях;

– использование промышленных стандартов во всех областях обмена данными и, как следствие, обеспечения совместимости кода, созданного в .NET Framework, с другими программами.

Первая общедоступная версия .NET Framework вышла в феврале 2002 года. С тех пор платформа активно эволюционировала и на данный момент было выпущено десять версии данного продукта. На данный момент номер последней версии .NET Framework — 4.5.3. Платформа .NET была призвана решить некоторые наболевшие проблемы, скопившиеся на момент её выхода, в средствах разработки приложений под Windows. Ниже перечислены некоторые из них [9, с. XIV – XVII]:

– сложность создания надежных приложений;

– сложность развертывания и управления версиями приложений и библиотек;

– сложность создания переносимого ПО;

– отсутствие единой целевой платформы для создателей компиляторов;

– проблемы с безопасным исполнением непроверенного кода;

– великое множество различных технологий и языков программирования, которые не совместимы между собой.

Многие из этих проблем были решены. Далее более подробно рассматривается внутреннее устройство .NET.

Основными составляющими компонентами .NET являются общая языковая исполняющая среда (Common Language Runtime) и стандартная библиотека классов (Framework Class Library). CLR представляет из себя виртуальную машину и набор сервисов обслуживающих исполнение программ написанных для .NET. Ниже приводится перечень задач, возлагаемых на CLR [10]:

– загрузка и исполнение управляемого кода;
– управление памятью при размещении объектов;
– изоляция памяти приложений;
– проверка безопасности кода;
– преобразование промежуточного языка в машинный код;
– доступ к расширенной информации от типах — метаданным;
– обработка исключений, включая межъязыковые исключения;
– взаимодействие между управляемым и неуправляемым кодом (в том числе и COM-объектами);
– поддержка сервисов для разработки (профилирование, отладка и т. д.).

Программы написанные для .NET представляют из себя набор типов взаимодействующих между собой. .NET имеет общую систему типов (Common Type System, CTS). Данная спецификация описывает определения и поведение типов создаваемых для .NET [11]. В частности в данной спецификации описаны возможные члены типов, механизмы сокрытия реализации, правила наследования, типы-значения и ссылочные типы, особенности параметрического полиморфизма и другие возможности предоставляемые CLI. Общая языковая спецификация (Common Language Specification, CLS) — подмножество общей системы типов. Это набор конструкций и ограничений, которые являются руководством для создателей библиотек и компиляторов в среде .NET Framework. Библиотеки, построенные в соответствии с CLS, могут быть использованы из любого языка программирования, поддерживающего CLS. Языки, соответствующие CLS (к их числу относятся языки C#, Visual Basic .NET, Visual C++/CLI), могут интегрироваться друг с другом. CLS — это основа межъязыкового взаимодействия в рамках платформы .NET [10].

Некоторые из возможностей, предоставляемых .NET: верификация кода, расширенная информация о типах во время исполнения, сборка мусора, безопасность типов, — невозможны без наличия подробных метаданных

о типах из которых состоит исполняемая программа. Подробные метаданные о типах генерируются компиляторами и сохраняются в результирующих сборках. Сборка – это логическая группировка одного или нескольких управляемых модулей или файлов ресурсов, является минимальной единицей с точки зрения повторного использования, безопасности и управлениями версиями [11, с. 6].

Одной из особенностей .NET, обеспечивающей переносимость программ без необходимости повторной компиляции, является представление исполняемого кода приложений на общем промежуточном языке (Common Intermediate Language, CIL). Промежуточный язык является бестиповым, стековым, объекто-ориентированным ассемблером [11, с. 16 – 17]. Данный язык очень удобен в качестве целевого языка для создателей компиляторов и средств автоматической проверки кода для платформы .NET, также язык довольно удобен для чтения людьми. Наличие промежуточного языка и необходимость создания производительных программ подразумевают наличие преобразования промежуточного кода в машинный код во время исполнения программы. Одним из компонентов общей языковой исполняющей среды, выполняющим данное преобразование, является компилятор времени исполнения (Just-in-time compiler) транслирующий промежуточный язык в машинные инструкции, специфические для архитектуры компьютера на котором исполняется программа.

Ручное управление памятью всегда являлось очень кропотливой и подверженной ошибкам работой. Ошибки в управлении памятью являются одними из наиболее сложных в устранении типами программных ошибок, также эти ошибки обычно приводят к непредсказуемому поведению программы, поэтому в .NET управление памятью происходит автоматически [11, с. 505 – 506]. Автоматическое управление памятью является механизмом поддержания иллюзии бесконечности памяти. Когда объект данных перестает быть нужным, занятая под него память автоматически освобождается и используется для построения новых объектов данных. Имеются различные методы реализации такого автоматического распределения памяти [12, с. 489]. В .NET для автоматического управления памятью используется механизм сборки мусора (garbage collection). Существуют различные алгоритмы сборки мусора со своими достоинствами и недостатками. В .NET используется алгоритм пометок (mark and sweep) в сочетании с различными оптимизациями, такими как, например, разбиение всех объектов по поколениям и использование различных куч для больших и малых объектов.

Ниже перечислены, без приведения подробностей, некоторые важные функции исполняемые общей языковой исполняющей средой:

– обеспечение многопоточного исполнения программы;
– поддержание модели памяти, принятой в CLR;
– поддержка двоичной сериализации;
– управление вводом и выводом;
– структурная обработка исключений;
– возможность размещения исполняющей среды внутри других процессов.

Как уже упоминалось выше, большую ценностью для .NET представляет библиотека стандартных классов — соответствующая CLS-спецификации объектно-ориентированная библиотека классов, интерфейсов и системы типов (типов-значений), которые включаются в состав платформы .NET. Эта библиотека обеспечивает доступ к функциональным возможностям системы и предназначена служить основой при разработке .NET-приложений, компонент, элементов управления [10].

2.2 Язык программирования C#

C# — объектно-ориентированный, типо-безопасный язык программирования общего назначения. Язык создавался с целью повысить продуктивность программистов. Для достижения этой цели в языке гармонично сочетаются простота, выразительность и производительность промежуточного кода, получаемого после компиляции. Главным архитектором и идеологом языка с первой версии является Андрес Хейлсберг (создатель Turbo Pascal и архитектор Delphi). Язык C# является платформенно нейтральным, но создавался для хорошей работы с .NET [13].

Язык имеет богатую поддержку парадигмы объекто-ориентированного программирования, включающую поддержку инкапсуляции, наследования и полиморфизма. Отличительными чертами C# с точки зрения ОО парадигмы являются:

– Унифицированная система типов. В C# сущность, содержащая данные и методы их обработки, называется типом. В C# все типы, являются ли они пользовательскими типами, или примитивами, такими как число, производны от одного базового класса.

– Классы и интерфейсы. В классической объекто-ориентированной парадигме существуют только классы. В C# дополнительно существуют и другие типы, например, интерфейсы. Интерфейс — это сущность напоми-

нающая классы, но содержащая только определения членов. Конкретная реализация указанных членов интерфейса происходит в типах, реализующих данный интерфейс. В частности интерфейсы могут быть использованы при необходимости проведения множественного наследования (в отличие от языков C++ и Eiffel, C# не поддерживает множественное наследование классов).

– Свойства, методы и события. В чистой объекто-ориентированной парадигме все функции являются методами. В C# методы являются лишь одной из возможных разновидностей членов типа, в C# типы также могут содержать свойства, события и другие члены. Свойство — это такая разновидность функций, которая инкапсулирует часть состояния объекта. Событие — это разновидность функций, которые реагируют на изменение состояния объекта [13].

В большинстве случаев C# обеспечивает безопасность типов в том смысле, что компилятор контролирует чтобы взаимодействие с экземпляром типа происходило согласно контракту, который он определяют. Например, компилятор C# не скомпилирует код который обращается со строками, как если бы они были целыми числами. Говоря более точно, C# поддерживает статическую типизацию, в том смысле что большинство ошибок типов обнаруживаются на стадии компиляции. За соблюдение более строгих правил безопасности типов следит исполняющая среда. Статическая типизация позволяет избавиться от широкого круга ошибок, возникающих из-за ошибок типов. Она делает написание и изменение программ более предсказуемыми и надежными, кроме того, статическая типизация позволяет существовать таким средствам как автоматическое дополнение кода и его предсказуемый статический анализ. Еще одним аспектом типизации в C# является её строгость. Строгая типизация означает, что правила типизации в языке очень «сильные». Например, язык не позволяет совершать вызов метода, принимающего целые числа, передавая в него вещественное число [13]. Такие требования спасают от некоторых ошибок.

C# полагается на автоматическое управление памятью со стороны исполняющей среды, предоставляя совсем немного средств для управления жизненным циклом объектов. Не смотря на это, в языке все же присутствует поддержка работы с указателями. Данная возможность предусмотрена для случаев, когда критически важна производительность приложения или необходимо обеспечить взаимодействие с неуправляемым кодом [13].

Как уже упоминалось C# не является платформенно зависимым языком. Благодаря усилиям компании Xamarin возможно писать программы на

языке C# не только для операционных систем Microsoft, но и ряда других ОС. Существуют инструменты создания приложений на C# для серверных и мобильных платформ, например: iOS, Android, Linux и других.

Создатели языка C# не являются противниками привнесения в язык новых идей и возможностей, в отличии от создателей одного из конкурирующих языков. Каждая новая версия компилятора языка привносит различные полезные возможности, которые отчаются требованиям индустрии. Далее приводится краткий обзор развития языка.

Первая версия C# была похожа по своим возможностям на Java 1.4, несколько их расширяя: так, в C# имелись свойства (выглядящие в коде как поля объекта, но на деле вызывающие при обращении к ним методы класса), индексаторы (подобные свойствам, но принимающие параметр как индекс массива), события, делегаты, циклы foreach, структуры, передаваемые по значению, автоматическое преобразование встроенных типов в объекты при необходимости (boxing), атрибуты, встроенные средства взаимодействия с неуправляемым кодом (DLL, COM) и прочее [14].

Версия .NET 2.0 привнесла много новых возможностей в сравнении с предыдущей версией, что отразилось и на языках под эту платформу. Проект спецификации C# 2.0 впервые был опубликован Microsoft в октябре 2003 года; в 2004 году выходили бета-версии (проект с кодовым названием Whidbey), C# 2.0 окончательно вышел 7 ноября 2005 года вместе с Visual Studio 2005 и .NET 2.0. Ниже перечислены новые возможности в версии 2.0

– Частичные типы (разделение реализации класса более чем на один файл).

– Обобщённые, или параметризованные типы (generics). В отличие от шаблонов C++, они поддерживают некоторые дополнительные возможности и работают на уровне виртуальной машины. Вместе с тем, параметрами обобщённого типа не могут быть выражения, они не могут быть полностью или частично специализированы, не поддерживают шаблонных параметров по умолчанию, от шаблонного параметра нельзя наследоваться.

– Новая форма итератора, позволяющая создавать сопрограммы с помощью ключевого слова **yield**, подобно Python и Ruby.

– Анонимные методы, обеспечивающие функциональность замыканий.

– Оператор ??: return obj1 ?? obj2; означает (в нотации C# 1.0) `return obj1!=null ? obj1 : obj2;`.

– Обнуляемые (nullable) типы-значения (обозначаемые вопросительным знаком, например, `int? i = null;`), представляющие собой те же самые

типы-значения, способные принимать также значение null. Такие типы позволяют улучшить взаимодействие с базами данных через язык SQL.

– Поддержка 64-разрядных вычислений позволяет увеличить адресное пространство и использовать 64-разрядные примитивные типы данных [14].

Третья версия языка имела одно большое нововведение — Language Integrated Query (LINQ), для реализации которого в языке дополнительно появилось множество дополнительных возможностей. Ниже приведены некоторые из них:

– Ключевые слова **select**, **from**, **where**, позволяющие делать запросы из SQL, XML, коллекций и т. п.

– Инициализацию объекта вместе с его свойствами:

```
Customer c = new Customer(); c.Name = "James"; c.Age=30;
```

можно записать как

```
Customer c = new Customer { Name = "James", Age = 30 };
```

– Лямбда-выражения:

```
listOfFoo.Where(delegate(Foo x) { return x.size > 10; });
```

теперь можно записать как

```
listOfFoo.Where(x => x.size > 10);
```

– Деревья выражений — лямбда-выражения теперь могут быть представлены в виде структуры данных, доступной для обхода во время выполнения, тем самым позволяя транслировать строго типизированные C#-выражения в другие домены (например, выражения SQL).

– Вывод типов локальной переменной: `var x = "hello";` вместо `string x = "hello";`

– Безымянные типы: `var x = new { Name = "James"};`

– Методы-расширения — добавление метода в существующий класс с помощью ключевого слова **this** при первом параметре статической функции.

– Автоматические свойства: компилятор сгенерирует закрытое поле и соответствующие аксессор и мутатор для кода вида

```
public string Name { get; private set; }
```

C# 3.0 совместим с C# 2.0 по генерируемому MSIL-коду; улучшения в языке — чисто синтаксические и реализуются на этапе компиляции [14].

Visual Basic .NET 10.0 и C# 4.0 были выпущены в апреле 2010 года, одновременно с выпуском Visual Studio 2010. Новые возможности в версии 4.0:

– Возможность использования позднего связывания.
– Именованные и опциональные параметры.
– Новые возможности COM interop.
– Ковариантность и контрвариантность интерфейсов и делегатов.
– Контракты в коде (Code Contracts) [14].

В C# 5.0 было немного нововведений, но они носят большую практическую ценность. В новой версии появилась упрощенная поддержка выполнения асинхронных функций с помощью двух новых слов — **async** и **await**. Ключевым словом **async** помечаются методы и лямбда-выражения, которые внутри содержат ожидание выполнения асинхронных операций с помощью оператора **await**, который отвечает за преобразования кода метода во время компиляции.

На данный момент разрабатывается версия C# 6.0. Новые возможности в версии 6.0 [15]:

– инициализация свойств со значениями:

```
public int Value { get; set; } = 100500;
```

– интерполяция строк:

```
obj.FirstName = "Anatoly";
obj.LastName = "Safonov";
name = $"First name {obj.FirstName}, last name {obj.LastName}";
```

– свойства и методы можно определять через лямбда-выражения:

```
public string[] GetCountryList() => new string[] { "Russia", "Belarus", "
    Poland" };
```

– импорт статических классов:

```
using System.Math;
double powerValue = Pow(2, 3);
double roundedValue = Round(10.6);
```

Это можно использовать не только внутри класса, но и при выполнении метода;

– null-условный оператор:

```
string location = obj?.Location;
```

– nameof оператор:

```
Console.WriteLine(emp.Location);
Console.WriteLine(nameof(Employee.Location));
```

– await в catch и finally блоках:

```
public async Task StartAnalyzingData()
{
  try
  {
```

```
    // code
  }
  catch
  {
    await LogExceptionDetailsAsync();
  }
  finally
  {
    await CloseResourcesAsync();
  }
}
```

– фильтры исключений:

```
try
{
  //throw exception
}
catch (ArgumentNullException ex) if (ex.Source == "EmployeeCreation")
{
  //catch block
}
catch (InvalidOperationException ex) if (ex.InnerException != null)
{
  //catch block
}
catch (Exception ex) if (ex.InnerException != null)
{
  //catch block
}
```

– инициализация Dictionary:

```
var country = new Dictionary<int, string>
{
  [0] = "Russia",
  [1] = "USA",
  [2] = "UK",
  [3] = "Japan"
};
```

Новая версия языка будет доступна в Visual Studio 2015. Microsoft выпустила предварительную версию Visual studio 2015 и .Net 4.6 для разработчиков, в которой можно изучить нововведения.

3 ПОСТАНОВКА ЗАДАЧИ

Сжатие данных — одна из задач, решаемых с использованием нейронных сетей. Как и любое сжатие, решение данной задачи основано на устранении избыточности.

Способность нейронных сетей к выявлению взаимосвязей между различными параметрами дает возможность выразить данные большой размерности более компактно, если данные тесно взаимосвязаны друг с другом.

Обратный процесс – восстановление исходного набора данных из части информации — называется ассоциативной памятью. Ассоциативная память позволяет также восстанавливать исходный сигнал или образ из поврежденных входных данных.

Исследования и разработки, проводимые в рамках дипломной работы, являются основой для анализа и проектирования новых способов использования нейронных сетей в области сжатия данных. Данная работа предназначена для оценки эффективности использования многослойных перцептронов для сжатия графической информации, учитывая такие факторы как степень сжатия, качество полученного изображения на выходе, а также вопрос производительности и нахождение оптимальных параметров нейронной сети.

3.1 Постановка задачи

В рамках дипломной работы поставлена цель найти способ сжимать графические данные с использованием многослойных перцептронов. Графическая информация представляется на компьютере как набор точек. Каждая точка имеет такие характеристики как позиция относительно границ и цвет. Процесс сжатия называется сжимающим кодированием. Сжатие основано на устранении избыточности, содержащейся в исходных данных. Иными словами, для сжатия данных используются некоторые априорные сведения о том, какого рода данные сжимаются. Не обладая такими сведениями об источнике, невозможно сделать никаких предположений о преобразовании, которое позволило бы уменьшить объём данных. Модель избыточности может быть статической, неизменной для всего сжимаемого сообщения, либо строиться на этапе сжатия и восстановления. Методы, позволяющие на основе входных данных изменять модель избыточности информации, называются адаптивными. Неадаптивными являются обычно узкоспециализированные алгоритмы, применяемые для работы с данными, обладающими хорошо определёнными и неизменными характеристиками. Нейронная сеть

позволяет реализовать алгоритм сжатия графической информации методом сжатия с потерями. Для решения поставленной задачи с помощью нейронных сетей необходимо:

– собрать данные для обучения;
– подготовить и нормализовать данные;
– выбрать топологию сети;
– эксперементально подобрать характеристики сети;
– эксперементально подобрать параметры для обучения;
– обучить сеть;
– проверить адекватность обучения сети.

3.2 Определение границ исследования

В рамках исследования необходимо получить численные данные о результатах использования нейронных сетей для сжатия данных. Для этого необходимо:

– выбрать тестовые данные;
– выбрать тип и топологию нейронной сети;
– составить поэтапный алгоритм преобразования изображения.

На следующем этапе необходимо разработать тестовое приложение. Приложение будет работать по составленному алгоритму и собирать статистику. Полученные данные можно представить в виде таблицы или графиков. Будет производен полный анализ всех факторов влияющий на итоговое изображение:

– частота дискретизации;
– размер входного блока для нейронной сети;
– соотношение размера входного и выходного блока кодера изображения;
– количество тестов для обучения нейронной сети.

После разработки прототипа необходимо подготовить подробное описание его структуры и сделать выводы о его эффективности. На основе проделанной работы сделать заключение о возможности использования нейронный сетей для сжатия графической информации.

4 РАЗРАБОТКА АЛГОРИТМА

Идея, лежащая в основе всех алгоритмов сжатия с потерями, довольно проста:

– на первом этапе удалить несущественную информацию;

– на втором этапе к оставшимся данным применить наиболее подходящий алгоритм сжатия без потерь.

Основные сложности заключаются в выделении этой несущественной информации. Подходы здесь существенно различаются в зависимости от типа сжимаемых данных. Для звука чаще всего удаляют частоты, которые человек просто не способен воспринять, уменьшают частоту дискретизации, а также некоторые алгоритмы удаляют тихие звуки, следующие сразу за громкими, для видеоданных кодируют только движущиеся объекты, а незначительные изменения на неподвижных объектах просто отбрасывают. Для одиночных изображений, как и для звука, отсеиваются элементы которые человек не способен различить. Методы выделения несущественной информации на изображениях будут подробно рассмотрены далее.

4.1 Критерии качества сжатого изображения

Прежде чем говорить об алгоритмах сжатиях с потерями, необходимо договориться о том, что считать приемлемыми потерями. Понятно, что главным критерием остаётся визуальная оценка изображения, но также изменения в сжатом изображении могут быть оценены количественно. Самый простой способ оценки — это вычисление непосредственной разности сжатого и исходного изображений.

$$error = \sum_{i=1}^{N} \sum_{j=1}^{M} |s_{i,j} - a_{i,j}| \tag{4.1}$$

где $s_{i,j}$ — значение пикселя в позиции i,j исходного изображения;
$a_{i,j}$ — значение пикселя в позиции i,j сжатого изображения изображения;
N — высота изображения в пикселях;
M — ширина изображения в пикселях.

Очевидно, что, чем больше суммарная ошибка, тем сильнее искажения на сжатом изображении. Тем не менее, эту величину крайне редко используют на практике, т.к. она никак не учитывает размеры изображения,

а значит не поможет оценить метод сжатия изображений разной величины. Гораздо шире применяется оценка с использованием среднеквадратичного отклонения [16]:

$$error = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{M}(s_{i,j} - a_{i,j})^2}{N * M}} \tag{4.2}$$

где $s_{i,j}$ — значение пикселя в позиции i,j исходного изображения;
$a_{i,j}$ — значение пикселя в позиции i,j сжатого изображения изображения;
N — высота изображения в пикселях;
M — ширина изображения в пикселях.

Другой подход заключается в следующем: пиксели итогового изображения рассматриваются как сумма пикселей исходного изображения и шума. Критерием качества при таком подходе называют величину отношения сигнал-шум (SNR), вычисляемую следующим образом:

$$SNR = \sqrt{\frac{\sum_{i=1}^{N}\sum_{j=1}^{M} a_{i,j}^2}{\sum_{i=1}^{N}\sum_{j=1}^{M}(s_{i,j} - a_{i,j})^2}} \tag{4.3}$$

где $s_{i,j}$ — значение пикселя в позиции i,j исходного изображения;
$a_{i,j}$ — значение пикселя в позиции i,j сжатого изображения изображения;
N — высота изображения в пикселях;
M — ширина изображения в пикселях.

Две последние формулы будут использоваться для оценки качества полученного изображения.

4.2 Выбор нейронной сети

В отличие от традиционных методов сжатия — математического вычисления и удаления избыточности — нейронная сеть при решении задачи сжатия исходит из соображений нехватки ресурсов. Топология сети и ее алгоритм обучения таковы, что данные большой размерности требуется передать со входа нейронной сети на ее выходы через сравнительно небольших размеров канал. Для реализации сжатия такого рода может использоваться многослойный перцептрон следующей архитектуры:

– количество нейронов во входном и выходном слое одинаково и равно размерности сжимаемых данных;

– между этими слоями располагаются один или более промежуточных слоев меньшего размера.

Число промежуточных слоев определяет степень сложности преобразования данных. Например сеть с тремя промежуточными слоями может выполнять лучшее сжатие на обучающих данных, но может дать худший результат в реальных ситуациях. Это связано с тем, что в исходных данных может случайно образоваться некая зависимость, которая не имеет никакого отношения к реальности. Исходные данные для сети составляются таким образом, чтобы на выходах был всегда тот же набор сигналов, что и на входе. В процессе работы алгоритм обратного распространения ошибки минимизирует ошибку. Это означает, что веса связей от входного слоя нейронов и, примерно, до серединного слоя будут работать на компрессию сигнала, а остальные — на его декомпрессию. При практическом использовании полученную сеть разбивают на две. Вывод первой сети передают по каналу связи и подают на вход второй, которая осуществляет декомпрессию(рисунок 4.1).

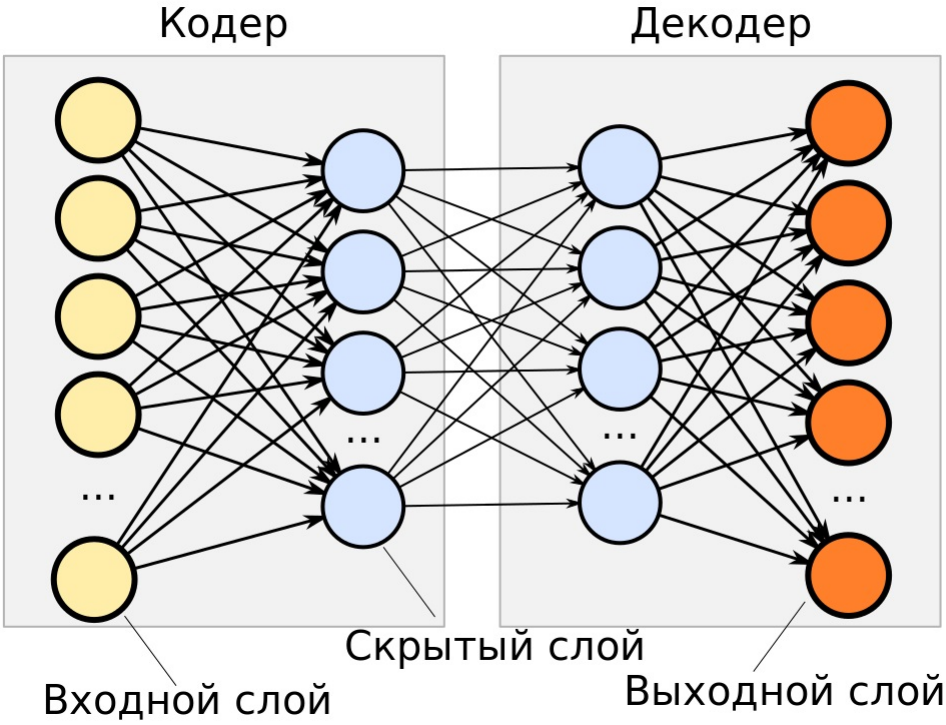

Рисунок *4.1* — «Бутылочное горлышко» — одна из возможных топологий нейросетей

4.3 Построение схемы алгоритма

Для построения прототипа необходимо сформировать последовательность операций на исходным изображением. Распишем этапы сжатия последовательно:

– загрузка матрицы пикселей;

– преобразование значений пикселей в сигналы для нейронной сети;

– разбиение матрицы сигналов на блоки;

– выбор соотношения количества входных нейронов к количеству нейроннов в срединном слое;

– выбор необходимого количества блоков для обучения сети;

– обучение нейронный сети на выбранных блоках;

– разделение сети на кодер и декодер;

– кодирование изображение при помощи кодера;

– сохранение декодера и полученных данных;

– сжатие методом без потерь сохраненных данных;

Загрузка изображений будет производится в двухмерный массив, где каждому пикселю исходного изображения будет соответствовать число от 0 до 255. На данном этапе будем сжимать только черно-белые изображения.

Далее нам необходимо преобразовать исходную матрицу в матрицу сигналов. Сигнал — это вещественное число от -1 до 1.

$$signal = (pixel + 1)/128 - 1 \tag{4.4}$$

где $pixel$ — значение пикселя, лежащее в диапазоне 0..255.

Следующий этап — разбиени матрицы сигналов на блоки. Размер блока будет влиять на размер нейронной сети и соответственно на скорость ее обучения. В реализации алгоритма попробуем использовать блоки разного размера.

На этапе выбора соотношения количества входных нейронов к количеству нейроннов в срединном слое, фактически идет выбор степени сжатия изображения. Если количество входных нейроннов будет 2 раза больше, чем количество нейроннов в срединном слое, то данные сожмуться в 2 раза.

На этапе подготовки выборки, следует учитывать несколько моментов:

– большое количество тестов ведет к повышению качества изображе-

ния;

– большое количесво тестов замедляет обучение сети, а соответственно замедляет процесс сжатия;

– тестовые данные должны удолетворять критериям репрезентативности и непротиворечивости.

Обучение сети будет производидится алгоритмом обратного распространения ошибки. На каждой итерации алгоритма обратного распространения весовые коэффициенты нейронной сети модифицируются так, чтобы улучшить решение одного примера. Следовательно будем применять алгоритм к каждому выбранному блоку на предыдущем этапе, пока не достигнем необходимой точности(разницы между входным и выходными сигналами).

Основная идея этого метода состоит в распространении сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы. Для возможности применения метода обратного распространения ошибки активационная функция нейронов должна быть дифференцируема. Метод является модификацией классического метода градиентного спуска. На каждой итерации алгоритма обратного распространения весовые коэффициенты нейронной сети модифицируются так, чтобы улучшить решение одного примера. Таким образом, в процессе обучения циклически решаются однокритериальные задачи оптимизации.

Следующий этап — разделение нейронной сети на кодер и декодер. Это необходимо для того, чтобы кодером сжать исходные сигнялы в новую матрицу сигналов. Декодер вместе с новой матрицей сигналов сохраняется с применением метода сжатия без потерь.

Этапы открытия изображения для просмотра:

– загрузить сохраненный данные;

– декодировать сжатые данные;

– из полученных данных выделить декодер и матрицу сигналов;

– с помощью декодера преобразовать матрицу сигналов;

– выполнить обратное преобразование матрицы сигналов в матрицу пикселей.

В результате мы получили алгоритм преобразования изображения для сжатия и сохранения, а так же алгоритм для открытия сжатого изображения. Реализация данных алгоритмов будет описана в следующей главе.

5 АРХИТЕКТУРА ПРОТОТИПА

Разработанное программное обеспечение предстваляет из себя библиотеку кода, написанную на языке C#. Библиотека предназначена для демонстрирования использования нейронных сетей для сжатия графической информации.

Одним из важных решений, которое было принято в начале проектирования модуля работы с нейронными сетями, было разделение реализации от абстрактной модели. Это решение позволило сначало реализовать сам алгоритм без реализации его компонентов. Так же это позволило в дальнейшем протестировать составные части.

В ходе разработки активно использовались паттерны Стратегия, Абстрактная Фабрика, Строитель. Стратегия — поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения их взаимозаменяемости. Это позволяет выбирать алгоритм путем определения соответствующего класса. Шаблон позволяет менять выбранный алгоритм независимо от объектов-клиентов, которые его используют [17].

Абстрактная фабрика — порождающий шаблон проектирования, позволяющий изменять поведение системы, варьируя создаваемыми объектами, при этом сохраняя интерфейсы. Он позволяет создавать целые группы взаимосвязанных объектов, которые, будучи созданными одной фабрикой, реализуют общее поведение. Шаблон реализуется созданием абстрактного класса Factory, который представляет собой интерфейс для создания компонентов системы. Затем пишутся классы, реализующие этот интерфейс [17].

Строитель — отделяет конструирование сложного объекта от его представления, так что в результате одного и того же процесса конструирования могут получаться разные представления [17].

На рисунке 5.1 представлена схема модулей системы. В данной секции будет приведено описение секции Interfaces. В ней содержится абстрактное представление системы.

В системе присутствует 8 интерфейсов(рисунок 5.2):

– IPixelConverter — преобразователь одного пикселя в сигнал;

– IImageConverter — преобразователь матрицы пикселей в матрицу сигналов;

– IEncoder — интерфейс объекта кодирования сигналов;

– IDecoder — интерфейс объекта декодирования сигналов;

– IConverterFactory — интерфейс фабрики, которая возвращает пару

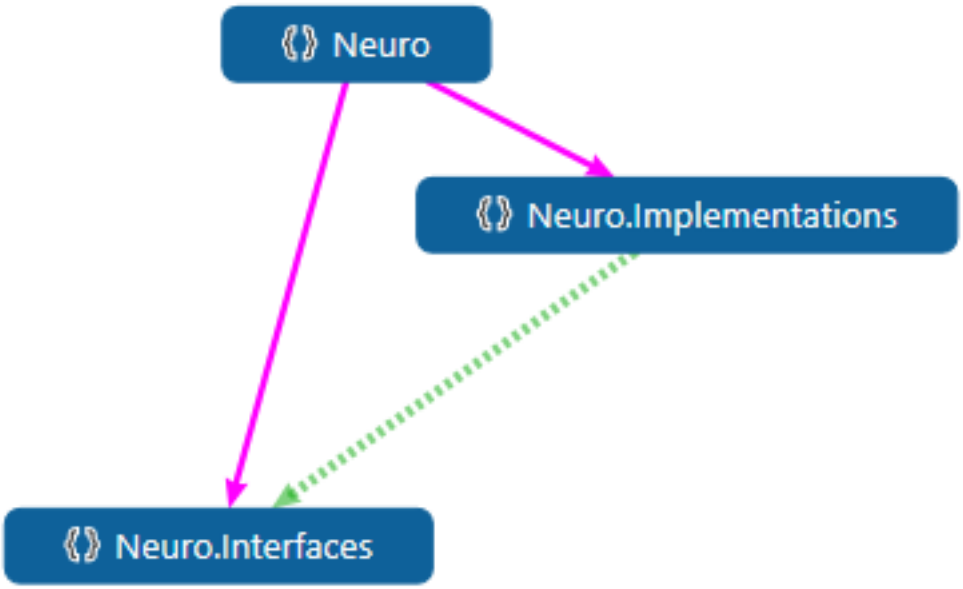

Рисунок *5.1* — Схема модулей программы

кодер/декодер;

– IConverterBuilder — интерфейс строителя, который по исходным данным создает фабрику кодеров/декодеров;

– IFileManager — интерфейс для загрузки/сохранения исходных и сжатых изображений;

– IImageComparer — интекфейс для алгоритмов оценки качества полученных изображений.

5.1 Интерфейс IPixelConverter

IPixelConverter — представляет интерфейс для конвертации сигналов в пиксели и наоборот(рисунок 5.3).

В нем представлены методы:

– ToPixel() — конвертация сигнала в пиксель;

– ToReal() — конфертация пикселя в сигнал.

В прототипе представлены 3 реализации:

– PixelConverterLow — преобразует 256-цветовой пиксель с глубиной дискретизации 8;

– PixelConverterMedium — преобразует 256-цветовой пиксель с глубиной дискретизации 32;

– PixelConverterHigh — преобразует 256-цветовой пиксель с глубиной дискретизации 256.

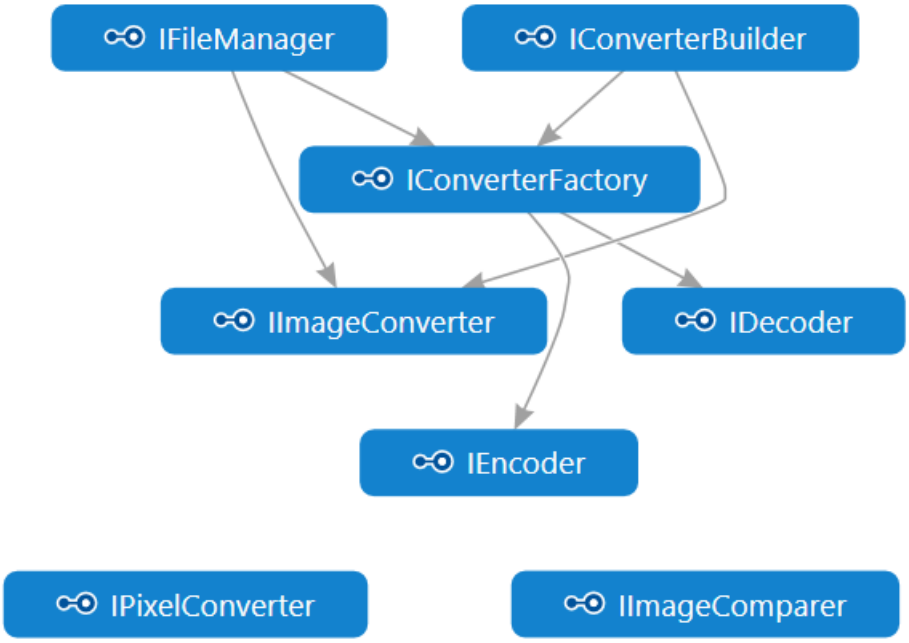

Рисунок *5.2* — Схема интерфейсов программы

5.2 Интерфейс IImageConverter

IImageConverter — представляет интерфейс для преобразования матрицы пикселей в матрицу сигналов и преобразования матрицы сигналов в матрицу пикселей, при помощи указанного перобразвателя который наследует интерфейс IPixelConverter(рисунок 5.4).

В нем представлены методы и свойства:

– BlockSize — размер стороны блока;

– Epsilon — шаг дискретизации;

– ToBitmap() — метод преобразования матрицы сигналов в матрицу пикселей;

– ToSignals() — метод преобразования матрицы пикселей в матрицу сигналов.

В прототипе представлены 3 реализации:

– ImageConverter — преобразует матрицу пикселей в матрицу сигналов соответвующего размера;

– ImageConverterRandom — преобразует матрицу пикселей в матрицу сигналов и выделяет только необходимое количество блоков для использования в алгоритме обучения нейронной сети.

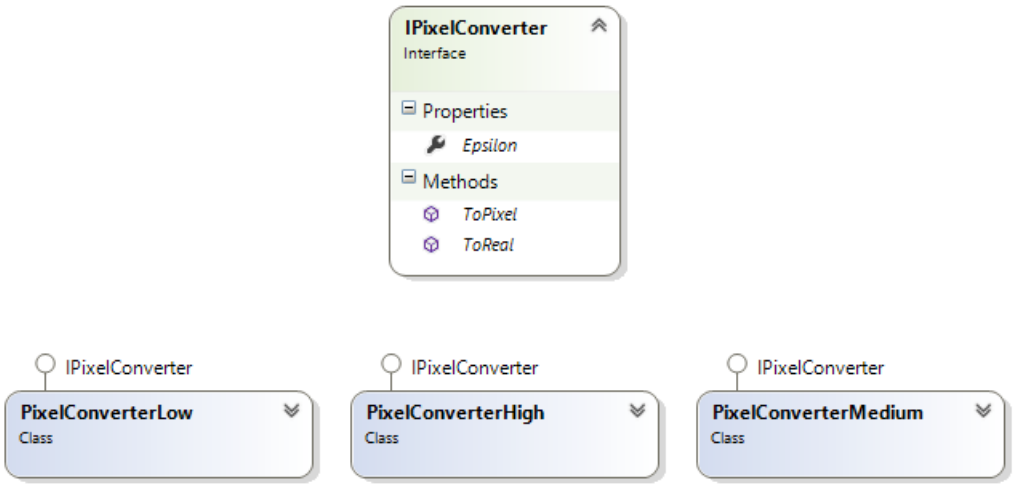

Рисунок *5.3* — Интерфейс IPixelConveter

5.3 Интерфейс IEncoder

IEncoder — представляет интерфейс для преобразования матрицы сигналов в матрицу сигналов меньшей размерности (рисунок 5.5).

В нем представлены методы:

– Encode() — метод преобразования матрицы сигналов одной размерности в матрицу сигналов другой размерности.

В прототипе представлена одна реализация Encoder, которая создается на основе полученной нейронной сети.

5.4 Интерфейс IDecoder

IDecoder — представляет интерфейс для преобразования матрицы сигналов малой размерности в матрицу сигналов исходной размерности (рисунок 5.6).

В нем представлены методы:

– Decode() — метод преобразования матрицы сигналов малой размерности в матрицу сигналов исходной размерности.

– Save() — метод сохраняющий объект IDecoder в Stream(класс представляющий поток данных).

В прототипе представлена одна реализация Decoder, которая создается на основе полученной нейронной сети или загружается из Stream.

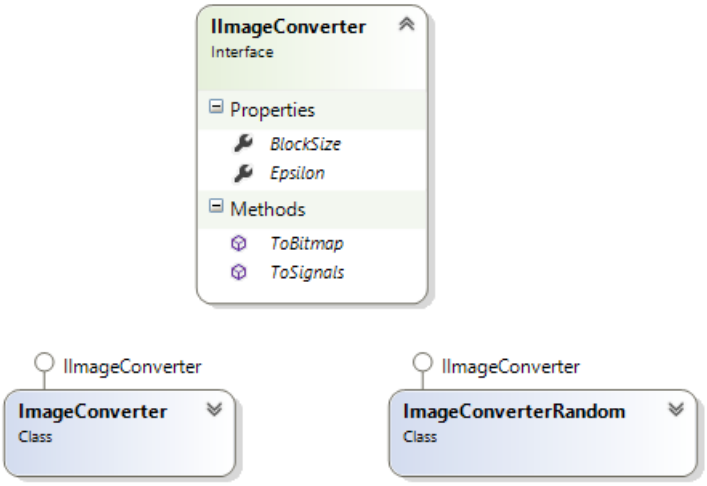

Рисунок *5.4* — Интерфейс IImageConverter

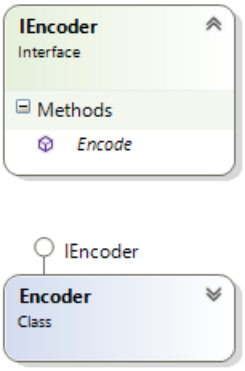

Рисунок *5.5* — Интерфейс IEncoder

5.5 Интерфейс IConverterFactory

IConverterFactory — представляет интерфейс, с использованием которого можно получить объекты кодера и декодера, а также узнать время обучения нейронной сети для их создания (рисунок 5.7).

В нем представлены методы и свойства:

– BuildTime — время обучения нейронной сети;

– GetEncoder() — метод, который предоставляет объект IEncoder;

– GetDecoder() — метод, который предоставляет объект IDecoder.

В прототипе представлена одна реализация ConverterFactory, которая создается объектом строителя ConverterBuilder.

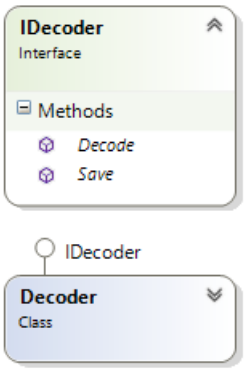

Рисунок *5.6* — Интерфейс IDecoder

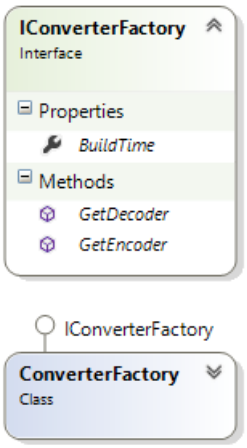

Рисунок *5.7* — Интерфейс IConverterFactory

5.6 Интерфейс IConverterBuilder

IConverterBuilder — представляет интерфейс, с помощью которого можно создавать объекты фабрик кодеров и декодеров.(рисунок 5.8). Использование паттерна строитель, позволяет манипулировать характеристиками нейронной сети, глубиной дискретизации. Интерфейс позволяет изменять степень сжатия, что позволяет манипулировать размером конечного файла и качеством конечного изображения.

В нем представлены методы и свойства:

– Build() — метод котрый принимает на вход IImageConverter, Bitmap, размер исходного блока, размер выходного блока и строит фабрику кодеров/декодеров.

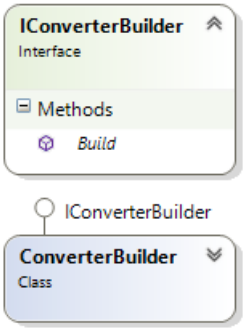

Рисунок *5.8* — Интерфейс IConverterBuilder

В прототипе представлена одна реализация ConverterBuilder, которая создает фабрику кодеров/декодеров для проведения исследования.

5.7 Интерфейс IFileManager

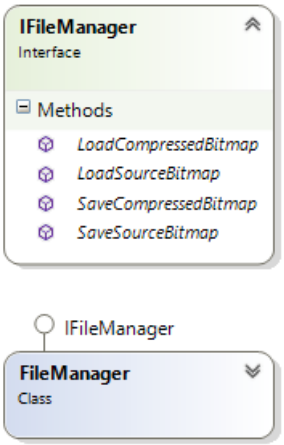

Рисунок *5.9* — Интерфейс IFileManager

IFileManager — представляет интерфейс, который инкапсулирует работу с файловой системой(рисунок 5.9). Данный интерфейс был выделен для того, чтобы уменьшить зависимость алгоритма сжатия данных от способа хранения. Следовательно, в будущем можно подобрать другие способы представления данных сжатых при помощи данного приложения.

В нем представлены методы:

– LoadSourceBitmap() — загрузка матрицы пикселей исходного изображения;

– SaveSourceBitmap() — сохранение матрицы пикселей;

– LoadCompressedBitmap() — загрузка матрицы пикселей из файла, содержащего сжатые данные нейронной сети;

– SaveCompressedBitmap() — сохранение матрицы пикселей используя сжатие.

В прототипе представлена одна реализация FileManager, которая читает матрицы пикселей из файлов. Также класс может FileManager сохранять и читать файлы с сжатыми изображниями.

5.8 Интерфейс IImageComparer

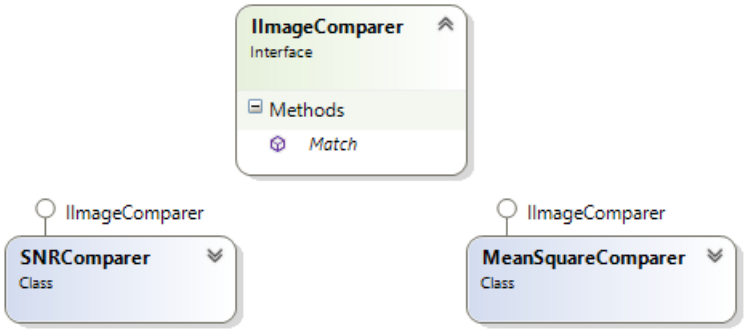

Рисунок *5.10* — Интерфейс IImageComparer

IImageComparer — представляет интерфейс для сбора статистики по уровню искажений в полученных изображениях(рисунок 5.10).

В нем представлены методы:

– Match() — метод сравнивает исходное изображение и получившееся, вычисляя погрешность;

В прототипе представлены 2 реализации:

– SNRComparer — вычисляет отношение сигнал-шум;

– MeanSquareComparer — вычисляет среднеквадратическое отклонение между изображениями.

6 АНАЛИЗ ПОЛУЧЕННЫХ ДАННЫХ

В данной описывается влияние параметров системы на сжатие изображений. Параметры влияние которых исследовалось:

– частота дискретизации;
– количество входных нейронов сети;
– количество нейронов на самом узком участке нейронной сети;
– количество тестовых примеров, подобранных для обучения сети.

В итоге был сформирован такой набор тестов:

– 3 глубины дискретизации изображения(8,32 и 256 уровней);
– 4 множества входных нейронов сети(16, 25, 64 и 100 нейронов);
– 6 отношений количества нейронов на самом узком участке сети к количеству входных нейронов(2 к 8, 3 к 8, 4 к 8, 5 к 8, 6 к 8 и 7 к 8);
– 15 наборов тестов к каждому обучению сети в количестве от 10 до 150 с шагом 10.

Было проведено 7560 тестов. Процесс был расспараллен на все вычислительные потоки центрального процессора. Каждый поток обрабатывал тесты для одного изображения и сохранял данные о результатах тестирования в тектовый файл, но вычисления все равно заняли 5 суток процессорного времени. Программа собирала данные о времени затраченном на сжатие изображения в конкретном тесте, степень сжатия, соотношение сигнал-шум и среднеквадратическое отклонение для исходного и полученного изображения. Так же программа сохраняла изображение в сжатом виде и его декодированную версию.

6.1 Частота дискретизации

С процессом квантования и дискретизации связано понятие визуальной избыточности. Значительная часть информации на изображении не может быть воспринята человеком: например, человек способен замечать незначительные перепады яркости, но гораздо менее чувствителен к цветности. Также, начиная с определённого момента, повышение точности дискретизации не влияет на визуальное восприятие изображения. Таким образом, некоторая часть информации может быть удалена без ухудшения визуального качества. Такую информацию называют визуально избыточной. Самым простым способом удаления визуальной избыточности является уменьшение точности дискретизации, но на практике этот способ можно применять только для изображений с простой структурой, т.к. искажения, возникающие на сложных изображениях, слишком заметны.

Как видно на рисунке 6.1 и рисунке 6.2, увеличение глубины дескритезации положительно сказывается на качестве изображения. Это связано с тем, что нейронная сеть обучается на примерах, более приближенных к исходному изображению. В противовес к этому нейронная должна запомнить больше количество комбинаций и построить более сложные связи.

Внимательно изучив полученные изображения, можно заметить, что на сжатых изображениях возникают отчётливые ложные контуры, которые значительно ухудшают визуальное восприятие. Существуют методы, основанные на переносе ошибки квантования в следующий пиксел, позволяющие значительно уменьшить или даже совсем удалить эти контуры, но они приводят к зашумлению изображения и появлению зернистости. Перечисленные недостатки сильно ограничивают прямое применение квантования для сжатия изображений.

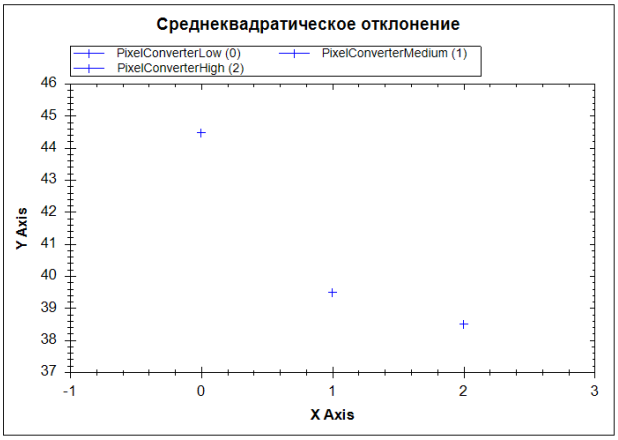

Рисунок *6.1* — зависимость значений среднеквадратичного отклонения от глубины дискретизации

Если посмотреть на рисунок 6.3, то можно заметить переходный этап, когда при достижении определенной глубины дикретизации, нейронная сеть обучается быстрее. Возможно это связано с тем, что на малой глубине дискретизации, нейронная сеть запоминает наборы сигналов поступивших на вход. На большой глубине дискретизации, нейронная сеть строит более сложные зависимости между нейронами и лучше анализирует значение сигнала в зависимости от его положения. А на переходном этапе, нейронная сеть обучается с большим количеством ошибок. В связи с этим алгоритм снижает скорость обучения сети, чтобы допускать меньше ошибок, приближение к оптиманьной ошибке затягивается. В дальнейшем планируется провести больше тестов в данном направлении, потому что из-за ограничений по времени, не удалось охватить больше уровней глубины дискретизации.

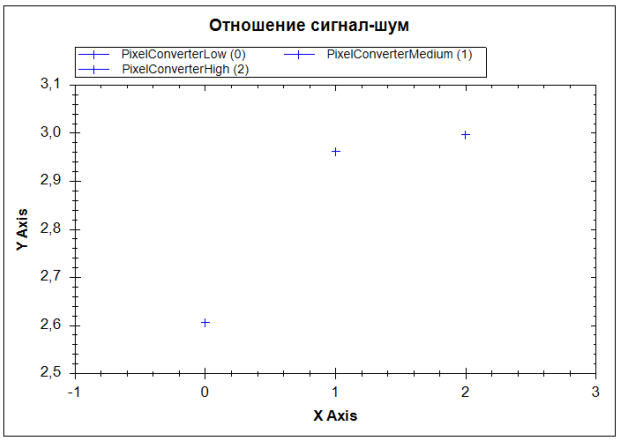

Рисунок *6.2* — зависимость значений отношения сигнал-шум от глубины дискретизации

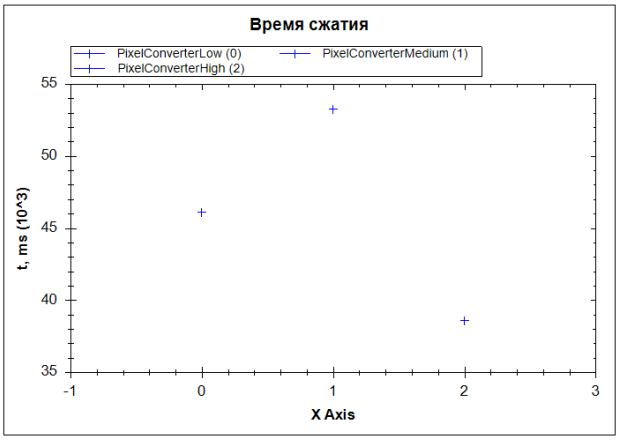

Рисунок *6.3* — зависимость времени сжатия от глубины дискретизации

На рисунке 6.4 видно, что с увеличением глубины дискретизации увеличивается и размер сжатой информации. Это объясняется тем что после обработки изображения нейронной сетью, получается набор сигналов, который с повышением глубины дискретизации принимает более детальные значение. То есть повышение глубины дискретизации приводит к увеличению количества комбинаций выходных данных, а так как на последнем этапе, сигналы сжимаются методом сжатия без потерь, то это приводит к увеличению энтропии, а следовательно и к увеличению размера выходного набора данных.

В данном случае использовался алгоритм Хаффмана. Это адаптивный жадный алгоритм оптимального префиксного кодирования алфавита с минимальной избыточностью. Был разработан в 1952 году аспирантом Массачусетского технологического института Дэвидом Хаффманом при на-

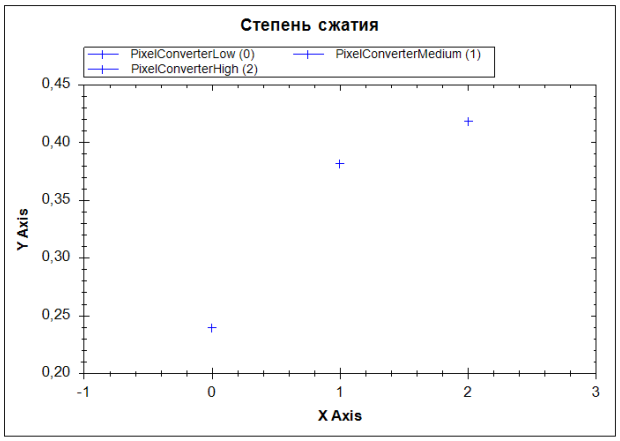

Рисунок *6.4* — зависимость отношения итогового размера к исходному размеру от глубины дискретизации

писании им курсовой работы. В настоящее время используется во многих программах сжатия данных [18].

6.2 Размер входного блока

Если посмотреть на рисунок 6.5 видно, что разброс результатов не превышает 2 пунктов. Пункты отражают насколько один пиксель текущего изображения отличается от пикселя сжатого изображения. Отношение сигнал-шум тоже имеет отклонение не более 0.2 пункта(рисунок 6.6). Это приводит к выводу, что размер блоков слабо влияет на итоговое изображение. Основной критерии выбора размера блока можно вывести следующие:

– размер блока должет позволять построить оптимальный набор тестов для обучения;

– размер блока не должен быть вырожденным(размеров в 1-4 сигнала);

– размер блока должен позволять разбить исходное изображение на оптимальное количество участков, по которым, можно будет восстановить исходное изображение.

Фактически сеть с большим количеством нейронов сможет построить более сложные зависимости, между участками изображения, но подбор тестовых данных и обучение такой сети может занять продолжительное время. Следовательно необходимо найти оптимальное соотношение времени-сложности сжатия. На рисунке 6.7 прослеживается линейная зависимость времени сжатия изобраения от величины входного слоя. Следовательно оптимальным решением будет брать размер блока меньше, но не слишком ма-

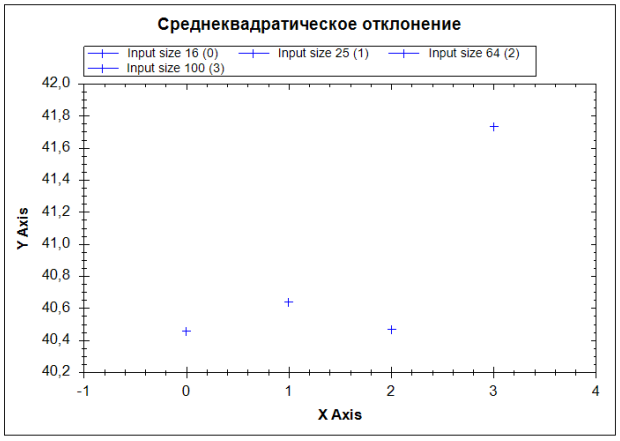

Рисунок *6.5* — зависимость значений среднеквадратичного отклонения от количества нейроннов на входном слое

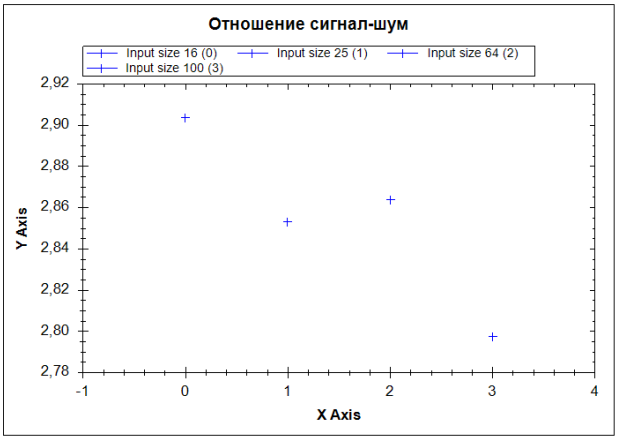

Рисунок *6.6* — зависимость значений отношения сигнал-шум от количества нейроннов на входном слое

леньким, чтобы нейронная сеть смогла объективно обучится на выбранных тестах.

На рисунке 6.8 видно, что размер входного блока имеет сильное влияние на размер выходных данных. Очевидно что блок большего размера имеет большее количество возможных представлений и количество их растет экспонинциально. Это является еще одной причиной, почему следует использовать блоки меньшего размера.

Одна из наиболее серьезных трудностей изложенного подхода в том, что таким образом мы минимизируем не ту ошибку, которую на самом деле нужно минимизировать. В конечном итоге требуется уменьшать ошибку, которую можно ожидать от сети, когда ей будут подаваться совершенно новые наблюдения. Иначе говоря, чтобы нейронная сеть обладала способностью

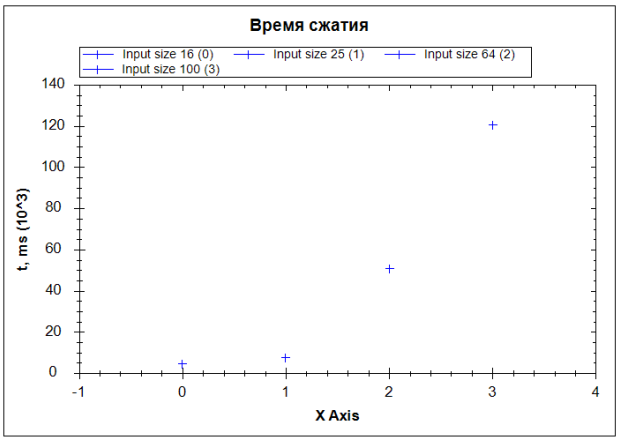

Рисунок *6.7* — зависимость времени сжатия от количества нейроннов на входном слое

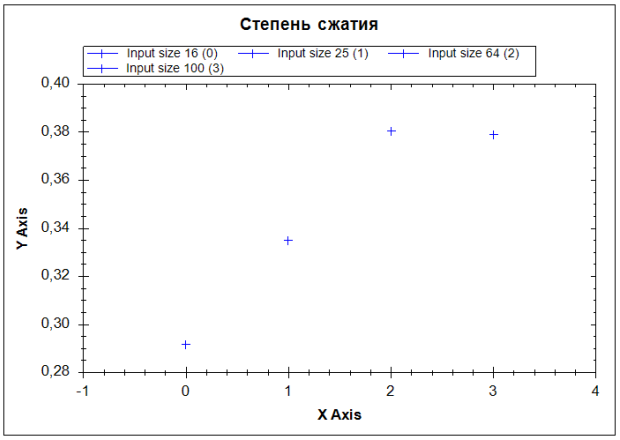

Рисунок *6.8* — зависимость отношения итогового размера к исходному размеру от количества нейроннов на входном слое

обобщать результат на новые наблюдения. В действительности сеть обучается минимизировать ошибку на обучающем множестве, и в отсутствие идеального и бесконечно большого обучающего множества это совсем не то же самое, что минимизировать требуемую ошибку на поверхности ошибок в заранее неизвестной модели явления. То есть обучая сеть на заданной выборке блоков, неинзвестно, уменьшается ли ошибка для блоков, которые будут кодироваться в конечном итоге, для получения сжатого изображения.

6.3 Соотношение количества нейронов между слоями

Если посмотреть на рисунок 6.9 видно, что разброс результатов не превышает 3 пунктов. Это значит, что соотношение количества нейронов

сильнее влияет на качество итогового изображения, чем размер входного слоя нейроном. Отношение сигнал-шум тоже имеет отклонение не более 0.1 пункта(рисунок 6.10). Из этого можно сделать вывод, что увеличение количества нейроном промежуточного слоя, повышает качество изображения, а из этого следует, что данное оотношение влияет на:

– качество изображения;
– скорость сжатия;
– размер сжатых данных.

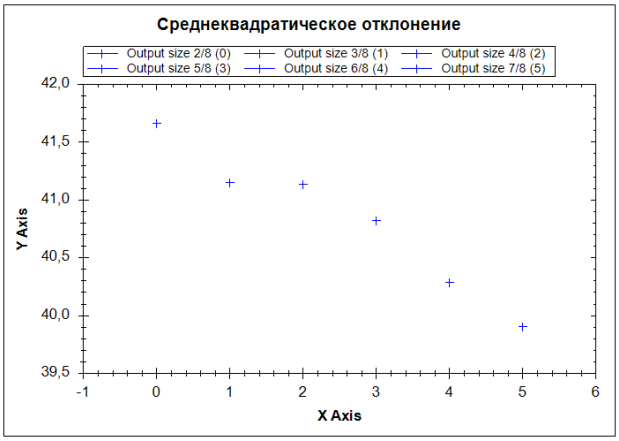

Рисунок *6.9* — зависимость значений среднеквадратичного отклонения от отношений количества нейронов на слоях

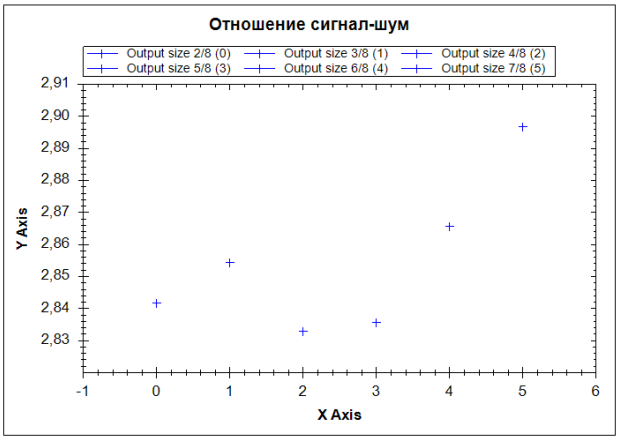

Рисунок *6.10* — зависимость значений отношения сигнал-шум от отношений количества нейронов на слоях

На рисунке 6.11 прослеживается сильная зависимость скорости сжатия от данного соотношения. Это связано с тем, что увеличение количества нейронов в слое приводит к увеличению связей между соседними слоями

нейронной сети. При добавлении одного нейрона в слой, происходит формирование потоков сигналов от всех нейронов предыдущего слоя к новому нейрону. И для нового нейрона создаются потоки, которые отправляют сигнал от него ко всем нейронам следующего слоя.

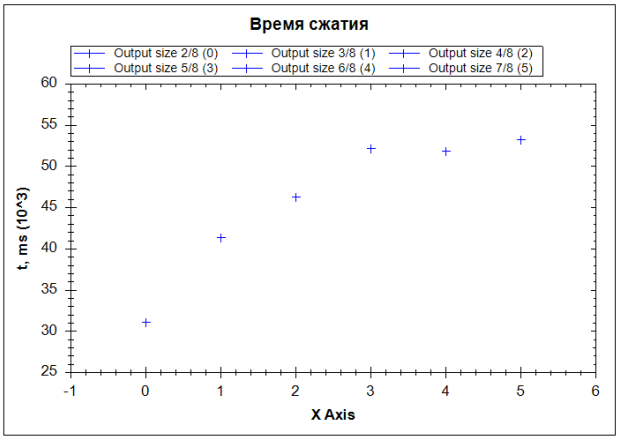

Рисунок *6.11* — зависимость времени сжатия от отношений количества нейронов на слоях

На рисунке 6.12 видно, что соотношение имеет заметное влияние на рамер выходного файла. Это объясняется тем что после обработки изображения нейронной сетью, получается набор сигналов, который с позволяет запомнить большее количество комбинаций, полученных из исходного изображения. Так же при увеличении количества нейронов увеличивается размер нейронной сети. Вместе с выходными сигналами хранятся данные для востановления декодера, а декодер является половиной исходной нейронной сети, полученной в процессе обучения на текущем изображении. Следовательно, чем больше нейронов и слоёв содержит нейронная сеть, тем больше будет занимать место декодер в выходных данных.

Если весовые коэффициенты к элементам скрытого слоя фиксированы, то необходимо, чтобы количество элементов скрытого слоя экспоненциально возрастало с ростом размерности задачи. Тем самым, теряется их основное преимущество — способность решать задачи произвольной сложности при помощи простых элементов. В данном случае это проявляется в том, что при увеличении размеров входного слоя увеличивается и размер скрытого слоя.

В отличие от традиционных методов сжатия — математического вычисления и удаления избыточности — данная нейронная сеть при решении задачи сжатия исходит из соображений нехватки ресурсов. Топология сети и

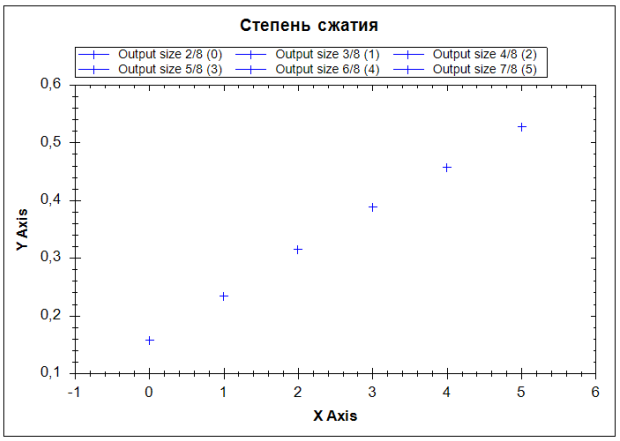

Рисунок *6.12* — зависимость отношения итогового размера к исходному размеру от отношений количества нейронов на слоях

ее алгоритм обучения таковы, что данные большой размерности передаются на входной слой нейронов. Далее сигналы передаются на скрытый слой, который меньше по размеру чем входной слой. Сигналы с скрытого слоя направляются к выходному слою, который копирует размер входного слоя. В итоге нейронная сеть, состоящая из 3 слоев, напоминает по форме песочные часы. Слой входных нейронов называется кодером, а слой выходных нейронов называется декодером. Число промежуточных слоев определяет степень сложности преобразования данных. апример сеть с тремя промежуточными слоями может выполнять лучшее сжатие на обучающих блоках, но может дать худший результат в реальных ситуациях. Это связано с тем, что в исходных данных может случайно образоваться некая зависимость, которая не имеет никакого отношения к реальности.

Веса нейронов были представлены вещественными числами двойной точности, а следовательно занимали 8 байт информации. Максимальный размер сети, который использовался в данном проекте содержал 880 000 отдельнях весов. Для декодера было необходима только половина этих весов. Следовательно имеелось 440 000 чисел, занимающих в памяти по 8 байт. При сохранении число байт под отдельное значение веса сокращалось, по причине того, что в данном случае не требуется точность в 15-16 знаков после запятой, которую дает вещественное число двойной точности. Общий размер декодера занимал значительное количество места в итоговых данных.

6.4 Количество тестов для обучения нейронной сети

Анализируя данные, полученные на рисунке 6.13 и рисунке 6.14, делаются выводы о том, что увеличение тестов ведет к увеличению величины ошибки. Это связано с тем, что нейронная сеть начинает специализироваться на конкретных примерах, что дает ошибку на реальных данных.

Переобучение в машинном обучении и статистике — явление, когда построенная модель хорошо объясняет примеры из обучающей выборки, но относительно плохо работает на примерах, не участвовавших в обучении (блоки выбранные из исходного изображения). Это связано с тем, что при построении модели в обучающей выборке обнаруживаются некоторые случайные закономерности, которые отсутствуют в генеральной совокупности. Даже тогда, когда обученная модель не имеет чрезмерного количества параметров, можно ожидать, что эффективность её на новых данных будет ниже, чем на данных, использовавшихся для обучения. В частности, значение коэффициента детерминации будет сокращаться по сравнению с исходными данными обучения.

С переобучением в данном случае, можно бороться увеличением размера нейронной сети. То есть необходимо увеличить количество нейронов на входном слое, увеличить количество скрытых слоев и число нейронов в них. Тогда нейронная сеть будет способна запомнить большее количество тестов и остаться при этом не переобученной и способной находить и поддерживать необходимый уровень качества обработки сигналов. Еще один способ борьбы с переобучением, это подбирать другие данные для обучения, когда достигается определенный порог точности, потому что мы будем использовать обученную нейронную сеть, на данных, которые нам заранее известны.

Очевидно, что увеличение количества блоков в тестовом наборе, приводит к увеличению времени обучения(рисунок 6.15). Это объясняется тем, что нейронной сети требуется сформировать большее количество связей между нейронами. Так же в процессе обучения могут возникать ошибки, которые задерживают процесс схождения обучения нейронной сети. Даже если процесс обучения не сойдется за обозримое время, в алгоритме было предусмотрено, что при достижении определенного количества поколений обучения, прерывать процесс. Так как проводилось тестирование параметров сети в разнообразных ситуациях, получиваяся нейронная сеть, даже не достигнувшая необходимой точности, в любом случае включалась в процесс сжатия изображения. В любом случае требуется следить за состоянием

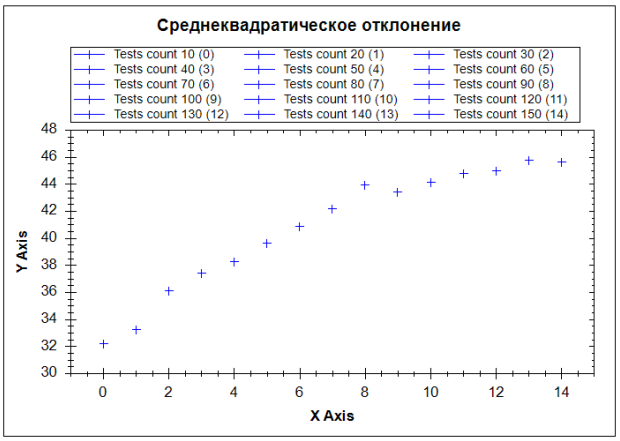

Рисунок *6.13* — зависимость значений среднеквадратичного отклонения от размера данных для обучения нейронной сети

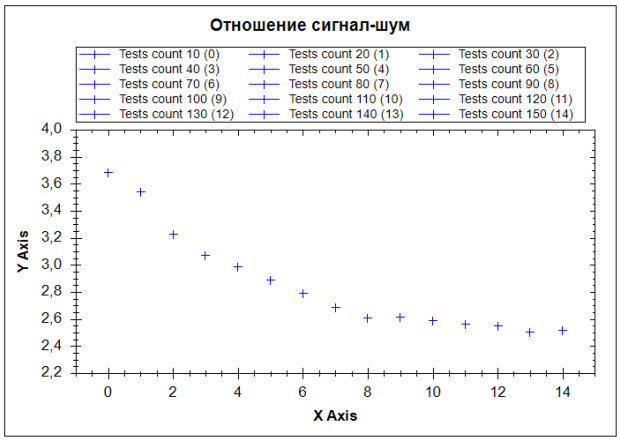

Рисунок *6.14* — зависимость значений отношения сигнал-шум от размера данных для обучения нейронной сети

нейронной сети, тщательно подбирать ее параметры.

После выбора общей структуры нужно экспериментально подобрать параметры сети. Для сетей, подобных перцептрону, это будет число слоев, наличие или отсутствие обходных соединений, активационные функции нейронов. При выборе количества слоев и нейронов в них следует исходить из того, что способности сети к обобщению тем выше, чем больше суммарное число связей между нейронами. С другой стороны, число связей ограничено сверху количеством записей в обучающих данных.

В процессе обучения сеть в определенном порядке просматривает обучающую выборку блоков. Порядок просмотра может быть последовательным, случайным и т. д. Основной принцип обучения в данном случае заключался в том, что на вход подавались данные, которые и ожидались на

выходе. Этот факт значительно облегчает процесс обучения.

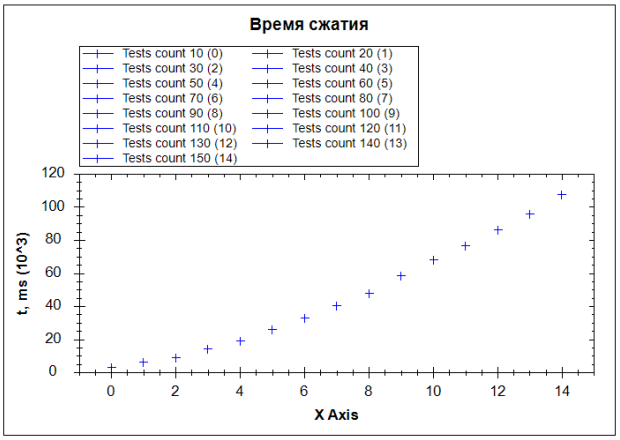

Рисунок *6.15* — зависимость времени сжатия от размера данных для обучения нейронной сети

Размер выходных файлов так же зависел от количества блоков, которые были выбраны для обучения(рисунок 6.16).

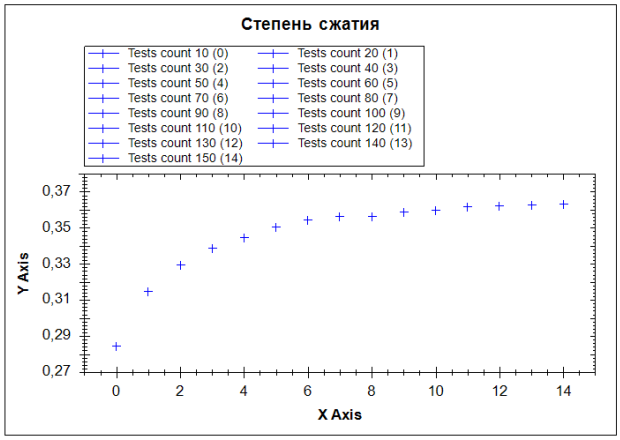

Рисунок *6.16* — зависимость отношения итогового размера к исходному размеру от размера данных для обучения нейронной сети

При увеличении количества тестов происходило увеличение размера выходных данных. Это объясняется увеличением количества комблинаций выходных блоков, образованных нейронной сетью под влиянием теста с большим количеством блоков.

Таким образом были перечислены основные факторы влияющие на результать кодирования исходных изображений.

7 ОБЕСПЕЧЕНИЕ БЕЗОПАСНЫХ УСЛОВИЙ ТРУДА ИНЖЕНЕРА-ПРОГРАММИСТА ПРИ ПРОВЕДЕНИИ ИССЛЕДОВАНИЯ МНОГОСЛОЙНЫХ ПЕРЦЕПТРОНОВ ДЛЯ СЖАТИЯ ИЗОБРАЖЕНИЙ

При проведении исследований в области сжатия данных и изучении свойств нейронных сетей используется электронно-вычислительная техника. Исследование включает в себя поиск и анализ материалов по использованию нейронных сетей в качестве способа сжатия графической информации. Процесс исследования тесно связан с необходимостью восприятия изображений на экране, наблюдением за изменениями в системе, ведением записей, подготовкой чтением рукописных и печатных материалов.

Зрительное восприятие человека является одним из основных в системе анализаторных систем. В процессе работы с персональной электронной вычислительной машиной(ПЭВМ) можно столкнуться с такими вредными факторами: несоответствие визуальных эргономических параметров дисплея (по яркости, контрасту, цветовой гамме и т. д.); неправильное расположение рабочих мест; неправильно спроектированное освещение в помещении.

Рабочее место необходимо располагать таким образом, чтобы в поле зрения не попадали оконные проемы или осветительные приборы. Следует добиваться уменьшения отражений на экране от различных источников света.

При низком уровне освещенности ухудшается видимость, при слишком высоком - уменьшается контраст изображения на экране. Освещенность на поверхности стола в зоне размещения рабочего документа должна быть 300-500 лк. Возможна установка светильников местного освещения для подсветки документов. При этом не следует допускать появления бликов на экране и увеличения его освещенности - более 300 лк [19].

В зависимости от источников света производственное освещение может быть естественным, искусственным и совмещенным. Естественное освещение создается источниками света природного характера. Освещение обеспечивается через оконные проемы с коэффициентом естественного освещения КЕО не ниже 1,2% в зонах с устойчивым снежным покровом и не ниже 1,5% на остальной территории [19].

Искусственное освещение создается лампами накаливания, люминес-

центными лампами или источниками, использующие светодиоды. Искусственное освещение в помещениях эксплуатации компьютеров должно осуществляться системой общего равномерного освещения. Совмещенное освещение представляет собой дополнение естественного освещения искусственным в темное и светлое время суток при недостаточном естественном освещении. Искусственное освещение делится на несколько разновидностей: общее; местное; комбинированное. При общем освещении происходит равномерное распределение света по все площади. Это достигается соблюдением одинакового расстояния между светильниками, которые равномерно рассеянны.

При источнике света, локализованном в одной точке, будет наблюдаться разница в яркости света, но резкие перепады будут отсутствовать. Примером может послужить расположенная посередине потолка люстра.

Чтобы выделить необходимые объекты или зоны используют местное освещение. Источник света при этом располагают на определенном участке: рабочем столе или части стены. В нашем случае это настольные лампы, расположенные на каждом рабочем столе.

Естественное освещение желательно осуществлять через светопроемы, ориентированные преимущественно через север и северо-восток.

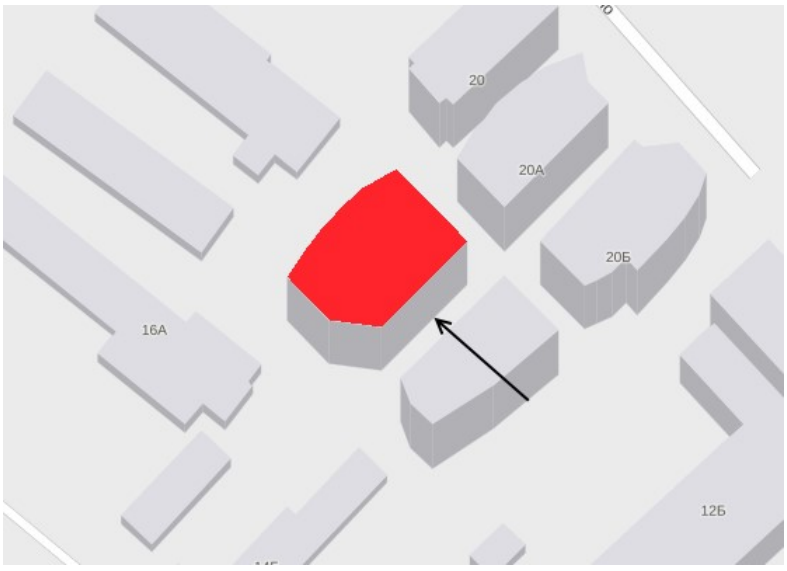

Рисунок *7.1* — Расположение окон офиса относительно сторон света

На (рисунок 7.1) видно, что светопроемы направлены на юго-восток, поэтому могут возникать проблемы, когда в дневное время суток, прямые солнечные лучи будут попадать на рабочее место, и вызывать блики на поверхности дисплея. Рекомендуется установить жалюзи, чтобы контроли-

ровать поток солнечного света.

Рисунок *7.2* — Офисный светодиодный светильник

В качестве источников света при искусственном освещении должны применяться преимущественно люминесцентные лампы. В данном случае используется освещение общее освещение, создаваемое светодиодными лампами (рисунок 7.2). В сравнение с обычными лампами накаливания, а также люминесцентными лампами светодиодные источники света обладают многими преимуществами [20].

Экономично используют энергию по сравнению с предшествующими поколениями электрических источников света — дуговыми, накальными и газоразрядными лампами. Так, световая отдача светодиодных систем уличного освещения с резонансным источником питания достигает 120 люмен на ватт, что сравнимо с отдачей люминесцентных ламп — 60-100 люмен на ватт. Для сравнения, световая отдача ламп накаливания, включая галогенные, составляет 10–24 люмен на ватт [20].

При оптимальном подключении источников питания, применении качественных компонентов и обеспечении надлежащего теплового режима срок службы светодиодных систем освещения при сохранении приемлемых для общего освещения показателей может достигнуть 36–72 тысяч часов, что в среднем в 50 раз больше по сравнению с номинальным сроком службы ламп накаливания общего назначения и в 4–16 раз больше, чем у большинства люминесцентных ламп.

Отсутствие инерционности при включении и выключении, что важно для светодинамических установок. Возможность диммирования по сравнению с большинством типов люминесцентных ламп. Отсутствие в составе соединений ртути (в отличие от газоразрядных люминесцентных ламп и других приборов), что исключает отравление ртутью при переработке и при эксплуатации. Практически полное отсутствие ультрафиолетового и инфракрасного излучения [21].

К недостаткам светодиодного освещения можно отнести: Спектр светодиодной лампы отличается от солнечного, поэтому приходится идти на компромисс между световой мощностью и качеством. Вместе с тем, он зачастую, при правильно подобранных люминофорах лучше по сравнению с люминесцентными лампами.

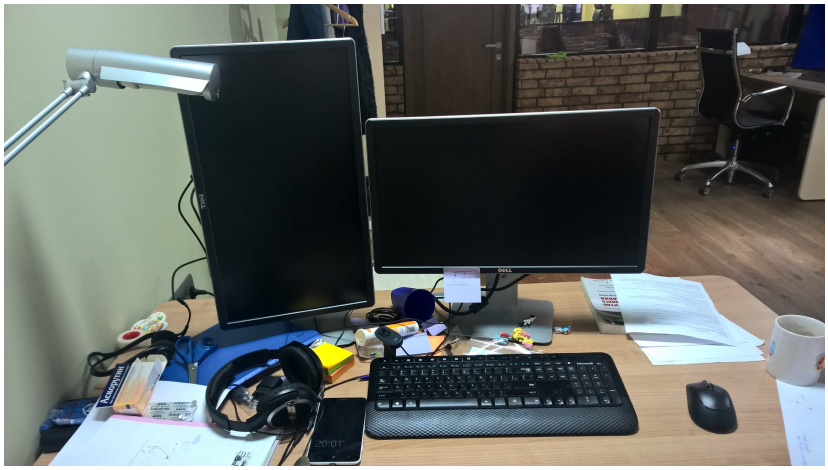

Рисунок *7.3* — Рабочее место разработчика программного обеспечения

Рабочее место также оборудовано местным освещением в виде светильника с установленной компактной люминесцентной лампой мощность 20 ватт(рисунок 7.3). Лампа имеет цветовую температуру дневного света 4200 К. Местное освещении требуется при длительной работе с печатными источниками информации. Рабочее место размещается таким образом, чтобы естественный свет падает сзади, что создает неудобства при работе ранним утром. Светильники общего освещения создают нормальные условия освещенности и соответствующий контраст между экраном и окружающей обстановкой с учетом вида работы и требований видимости со стороны работника. На рабочем месте используются мониторы с матовым покрытием, поэтому даже с ярким искусственным или естественным освещением работа с ним не доставляет дискомфорта. Возможные мешающие отражения и отблески на экране монитора и другом оборудовании устраняются путем соответствующего размещения экрана, оборудования, расположения светильников местного освещения.

Проблемы связанные с бликами на экране ранним утром решаются закрытием жалюзей или поворотом рабочего места боком к световым проемам. Таким образом, изложенные выше предложения обеспечивают безопасность условий труда инженера программиста при проведении исследования по теме «Многослойные перцептроны для сжатия изображений».

8 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ ЭФФЕКТИВНОСТИ РАЗРАБОТКИ И ИСПОЛЬЗОВАНИЯ МНОГОСЛОЙНЫХ ПЕРЦЕПТРОНОВ ДЛЯ СЖАТИЯ ИЗОБРАЖЕНИЙ

8.1 Характеристика программного продукта

В рамках дипломной работы поставлена цель найти способ сжимать данные с использованием многослойных перцептронов. В данной работе проведено исследование в области сжатия графической информации. Произведены оценки скорости, степени сжатия и качества полученных изображений. Программный продукт позволит продемонстрировать результаты использования нейронных сетей для сжатия аналоговой информации и оценить качество сжатия данных методом «сжатия с потерями».

В данном разделе будут рассмотрены затраты на разработку и использование приложения основанное на использовании нейронных сетей. Данное прилолжение позволит:

а) эффективно сжимать изображения

б) изучить возможности использования нейронных сетей для сжатия графической информации

Экономическая целесообразность инвестиций в разработку и использование программного продукта осуществляется на основе расчета и оценки следующих показателей:

– чистая дисконтированная стоимость ($Ч_{дд}$);
– срок окупаемости инвестиций ($Т_{ок}$);
– рентабельность инвестиций ($Р_{и}$).

Полученные результаты позволят строить программные комплексы, использование которых, улучшит эффективность сжатия данных методом «сжатия с потерями».

Для оценки экономической эффективности инвестиционного проекта по разработке и внедрению программного продукта необходимо рассчитать:

а) Результат (Р), получаемый от использования программного продукта;

б) Затраты (инвестиции), необходимые для разработки программного продукта;

в) Показатели эффективности инвестиционного проекта по производ-

ству программного продукта.

8.2 Расчет стоимостной оценки затрат

Общие капитальные вложения ($К_о$) заказчика (потребителя), связанные с приобретением, внедрением и использованием ПС, рассчитываются по формуле:

$$К_о = К_{пр} + К_{ос}. \quad (8.1)$$

где $К_{пр}$ — затраты пользователя на приобретение ПС по отпускной цене разработчика с учетом стоимости услуг по эксплуатации и сопровождению (тыс.руб.);
$К_{ос}$ — затраты пользователя на освоение ПС (тыс. руб.).

8.2.1 Расчет стоимостной оценки затрат

Основная заработная плата исполнителей на наш программный продукт рассчитывается по формуле:

$$З_о = \sum_{i=1}^{n} Т_{чi} Т_ч Ф_п К. \quad (8.2)$$

где n — количество исполнителей, занятых разработкой программного продукта;
$Т_{чi}$ — часовая тарифная ставка i-го исполнителя (тыс. руб.);
$Ф_п$ — плановый фонд рабочего времени i-го исполнителя (дн.);
$Т_ч$ — количество часов работы в день (ч).
К — коэффициэнт премирования.

Коэффициент премирования 1,6. Для расчета заработной платы месячная тарифная ставка 1-го разряда на предприятии установлено на уровне одного миллиона восьмиста шестидесяти тысяч белорусских рублей.

Дополнительная заработная плата на наш программный продукт ($З_д$) включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по нормативу в процентах к основной заработной

Таблица *8.1* — Расчёт заработной платы

Категория исполнителя	Разряд	Тарифный коэффициент	Часовая тарифная ставка, тыс. руб.	Трудоёмкость, дн.	Основная заработная плата, тыс. руб.
программист 1-ой категории	14	3.66	28.5	30	6760
руководитель проекта	16	3	32.1	30	7704
Итого с премией (60%)	-	-	-	-	14544

плате:

$$З_д = \frac{З_о Н_д}{100\%}, \tag{8.3}$$

где $З_д$ — дополнительная заработная плата исполнителей (тыс. руб.)

$Н_д$ — норматив дополнительной заработной платы равный 20%.

$$З_д = \frac{14\,544 \cdot 20\%}{100\%} = 2908 \text{ тыс.руб.}, \tag{8.4}$$

Отчисления в фонд социальной защиты населения и обязательное страхование ($З_{сз}$) определяются в соответствии с действующими законодательными актами по нормативу в процентном отношении к фонду основной и дополнительной зарплаты исполнителей, определенной по нормативу, установленному в целом по организации:

$$З_{сз} = \frac{(З_о + З_д)Н_{сз}}{100\%}, \tag{8.5}$$

где $Н_{сз}$ — норматив отчислений в фонд социальной защиты населения и на обязательное страхование (34 + 0,6%)

$$З_{сз} = \frac{(14\,544 + 2908) \cdot 34{,}6\%}{100\%} = 6038{,}7 \text{ тыс.руб.}, \tag{8.6}$$

Расходы по статье «Машинное время» ($Р_м$) включают оплату машинного времени, необходимого для разработки и отладки программного про-

дукта, которое определяется по нормативам (в машино-часах) на 100 строк исходного кода ($Н_{мв}$) машинного времени, и определяются по формуле:

$$Р_м = Ц_м + Т_{пр}, \quad (8.7)$$

где $Ц_м$ — цена одного машино-часа. Рыночная стоимость машино-часа компьютера со всеми необходимым оборудованием (10 тыс. руб. / ч);

$Т_{пр}$ — время работы над программным продуктом (25дн*8ч = 200ч).

$$Р_м = 10 \cdot 200 = 2000 \text{ тыс.руб.}, \quad (8.8)$$

Расходы по статье «Научные командировки» ($Р_{нк}$) на програмнное средство определяются по формуле:

$$Р_{нк} = \frac{З_о Н_{рнк}}{100\%}, \quad (8.9)$$

где $Н_{рнк}$ — норматив расходов на командировки в целом по организации (%). Норматив на командировки - 12 % от основной заработной платы.

$$Р_{нк} = \frac{14\,544 \cdot 12}{100\%} = 1745{,}3 \text{ тыс.руб.}, \quad (8.10)$$

Расходы по статье «Прочие затраты» ($П_з$) на программное средство включают затраты на приобретение и подготовку специальной научной и технической информации и специальной литературы. И определяются по формуле:

$$П_з = \frac{З_о Н_{пз}}{100\%}, \quad (8.11)$$

где $Н_{пз}$ — норматив прочих затрат в целом по организации равен 50%.

$$П_з = \frac{14\,544 \cdot 22\%}{100\%} = 3199{,}7 \text{ тыс.руб.}, \quad (8.12)$$

Затраты по статье «Накладные расходы» ($Р_н$), связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и

опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды ($Р_н$), и определяют по формуле:

$$Р_н = \frac{З_о Н_{рн}}{100\%}, \tag{8.13}$$

где $Н_{рн}$ — накладные расходы на программный продукт (тыс. руб.);
$Н_{рн}$ — норматив накладных расходов в целом по организации, 50%.

$$Р_н = \frac{14\,544 \cdot 50\%}{100\%} = 7272 \text{ тыс.руб.}, \tag{8.14}$$

Общая сумма расходов по смете ($С_р$) на программный продукт рассчитывается по формуле:

$$С_р = З_о + З_д + З_{сз} + Р_м + Р_{нк} + П_з + Р_н, \tag{8.15}$$

$$\begin{gathered} С_р = 14\,544 + 2908 + 6038{,}7 + 2000 + 1745{,}3 + \\ + 3199{,}7 + 7272 = 23\,163{,}7 \text{ тыс.руб.}, \end{gathered} \tag{8.16}$$

Кроме того, организация-разработчик осуществляет затраты на сопровождение и адаптацию программного продукта ($Р_{са}$), которые определяются по формуле:

$$Р_{са} = \frac{С_р Н_{рса}}{100\%}, \tag{8.17}$$

где $Н_{рса}$ — норматив расходов на сопровождение и адаптацию 20%.

$$Р_{са} = \frac{23\,163{,}7 \cdot 20\%}{100\%} = 4632{,}8, \tag{8.18}$$

Общая сумма расходов на разработку (с затратами на сопровождение и адаптацию) как полная себестоимость программно продукта ($С_п$) определяется по формуле:

$$С_п = С_р + Р_{са}, \tag{8.19}$$

$$С_п = 23\,163{,}7 + 4632{,}8 = 27\,796{,}5, \tag{8.20}$$

Прибыль рассчитывается по формуле:

$$П_о = \frac{С_п У_{рп}}{100\%}, \tag{8.21}$$

где $П_о$ — прибыль от реализации программного продукта заказчику (тыс. руб.);
$У_{рп}$ — уровень рентабельности программного продукта 20%;
$С_п$ — себестоимость програмнного продукта (тыс. руб.).

$$П_о = \frac{27\,796{,}5 \cdot 20\%}{100\%} = 5559{,}3 \text{ тыс.руб.}, \tag{8.22}$$

Прогнозируемая цена нашего программного продукта без налогов ($Ц_п$):

$$Ц_п = С_р + П_о, \tag{8.23}$$

$$Ц_п = 23\,163{,}7 + 5559{,}3 = 28\,723 \text{ тыс.руб.}, \tag{8.24}$$

8.3 Расчет стоимостной оценки результата

Результатом (Р) в сфере использования нашего программного продукта является прирост чистой прибыли и амортизационных отчислений.

8.3.1 Расчет прироста чистой прибыли

Прирост чистой прибыли представляет собой экономию затрат на заработную плату и начислений на заработную плату, полученную в результате внедрения программного продукта, составит:

$$Э_з = К_{пр}(t_с Т_с - t_н Т_н) N_п (1 + \frac{Н_{др}}{100\%})(1 + \frac{Н_{нпо}}{100\%}), \tag{8.25}$$

где $N_п$ — плановый объем работ по анализу и обработки результатов, сколько раз выполнялись в году ((24) раз);

$t_с$ — трудоемкость выполнения работы до внедрения программного продукта; (24 нормочасов);

$t_п$ — трудоемкость выполнения работы после внедрения программного продукта; (4 нормочаса);

$Т_с$ — часовая тарифная ставка, соответствующая разряду выполняемых работ до внедрения программного продукта; (35 тыс. руб. /ч)

$Т_п$ — часовая тарифная ставка, соответствующая разряду выполняемых работ после внедрения программного продукта; (50 тыс. руб. /ч)

$К_{пр}$ — коэффициент премий (1.6);

$Н_д$ — норматив дополнительной заработной платы (10%);

$Н_{по}$ — ставка отчислений в ФСЗН и обязательное страхование (34+0,6%).

$$Э_з = 1{,}6 \cdot (24 \cdot 35 - 4 \cdot 50) \cdot 24 \cdot (1 + \frac{10\%}{100\%})(1 + \frac{34{,}6}{100\%}) = 36\,387{,}3 \text{ тыс.руб.}, \quad (8.26)$$

Прирост чистой прибыли рассчитывается по формуле:

$$П_ч = \sum_{i=1}^{n} Э_i (1 - \frac{Н_п}{100\%}), \quad (8.27)$$

где n — виды затрат, по которым получена экономия;

Э — сумма экономии, полученная за счет снижения i-ых затрат, тыс. руб.

Н — ставка налога на прибыль, 18%.

$$П_ч = 36\,387{,}3 \cdot (1 - \frac{18\%}{100\%}) = (6549.7) \text{ тыс.руб.}, \quad (8.28)$$

8.3.2 Расчет прироста амортизационных отчислений

Амортизационные отчисления являются источником погашения инвестиций в приобретение программного продукта. Расчет амортизационных

отчислений осуществляется по формуле:

$$A = \frac{Н_а И_{об}}{100\%}, \tag{8.29}$$

где $Н_а$ — норма амортизации программного продукта 20%;
$И_{об}$ — стоимость программного продукта, тыс. руб.

$$A = \frac{20\% \cdot 28\,723}{100\%} = 5744{,}6, \tag{8.30}$$

8.4 Расчет показателей экономической эффективности проекта

При оценке эффективности инвестиционных проектов необходимо осуществить приведение затрат и результатов, полученных в разные периоды времени, к расчетному году, путем умножения затрат и результатов на коэффициент дисконтирования a_t, который определяется следующим образом:

$$a_t = \frac{1}{(1 + Е_н)^{(t - t_р)}}, \tag{8.31}$$

где $Е_н$ — требуемая норма дисконта, 30%;
t — порядковый номер года, затраты и результаты которого приводятся к расчетному году;
$t_р$ — расчетный год, в качестве расчетного года принимается год вложения инвестиций, равный 1.

$$a_1 = \frac{1}{(1 + 0{,}3)^{1\text{-}1}} = 1; \tag{8.32}$$

$$a_2 = \frac{1}{(1 + 0{,}30)^{2\text{-}1}} = 0{,}769; \tag{8.33}$$

$$a_3 = \frac{1}{(1 + 0{,}30)^{3\text{-}1}} = 0{,}591; \tag{8.34}$$

$$a_4 = \frac{1}{(1 + 0{,}30)^{4\text{-}1}} = 0{,}455; \tag{8.35}$$

Рассчитаем рентабельность инвестиций ($Р_и$) по формуле:

$$Р_и = \frac{П_{чср}}{З} \cdot 100\%, \tag{8.36}$$

где З — затраты на приобретения нашего программного продукта;

$П_{чср}$ — среднегодовая величина чистой прибыли за расчетный период, тыс. руб., которая определяется по формуле:

$$П_{чср} = \frac{\sum_{i=1}^{n} П_{чi}}{n}, \tag{8.37}$$

где $П_{чi}$ — чистая прибыль, полученная в году i, тыс. руб.

$$П_{чср} = \frac{2729 + 6549{,}7 + 6549{,}7 + 6549{,}7}{4} = 5594{,}5 \text{ тыс.руб.}, \tag{8.38}$$

$$Р_{и} = \frac{5594{,}5}{28\,723} \cdot 100 = 19{,}5\%, \tag{8.39}$$

В результате технико-экономического обоснования инвестиций по производству нового изделия были получены следующие значения показателей их эффективности:

а) Чистый дисконтированный доход за четыре года производства продукции составит 6549.7 тыс. руб.

б) Все инвестиции окупаются на 4 год

в) Рентабельность инвестиций составляет 19.5%

Таким образом проведение исследования и разработка программного продукта являются эффективными. Следовательно, инвестирование в разработку и внедрение программы сжатия данных основанной на многослойных перцептронах является целесообразным.

Таблица *8.2* — Экономические результаты работы предприятия

Наименование показателей	Един. изм-мер.	Усл. обоз.	По годам использования ПП			
			1-ый	2-ой	3-ий	4-ый
1. Прирост чистой прибыли	Тыс. руб.	$\Delta П_ч$	2729	6549.7	6549.7	6549.7
2. Прирост автоматизированных отчислений	Тыс. руб.	ΔA	5744.6	5744.6	5744.6	5744.6
3. Прирост результата	Тыс. руб.	ΔP_t	8473.6	12294.3	12294.3	12294.3
4. Коэфициент дисконтирования		a_t	1	0.769	0.591	0.455
5. Результат с учётом фактора времени	Тыс. руб.	$P_t a_t$	8473.6	9454.3	7265.9	5593.9
6. Инвестиции	Тыс. руб.	$И_{об}$	28723			
7. Инвестиции с учётом фактора времени	Тыс. руб.	$И_т a_t$	28723			
8. Чистый дисконтированный доход по годам	Тыс. руб.	$ЧДД_t$	-20294.4	9454.3	7265.9	5593.9
9. ЧДД нарастающим итогом	Тыс. руб.	ЧДД	-20249.4	-10795.1	-3529.2	2064.8

ЗАКЛЮЧЕНИЕ

В данном дипломной работе был рассмотрен вопрос использования многослойных перцептронов для сжатия изображений методом сжатия с потерями. В рамках дипломной работы был разработан поэтапный алгоритм преобразования графических данных с разными степенями сжатия. Был разработан прототип, который наглядно демонстрировал работоспособность данного метода. Были собраны многочисленные статистические данные, произведена качественная оценка получившихся результатов.

В целом были получены удовлетворительные результаты проверенные с помощью методов сравнения изображений с оригиналом, такими как соотношения сигнал-шум и среднеквадратического отклонения. Результаты работы реализованных в прототипе функций помогли выделить основные факторы, влияющие на степень сжатия изобращения, скорость обработки и степень отличия от исходного изображения. Данный способ удовлетворительно зарекомендовал себя в проведённых тестах.

В результате цель дипломной работы была достигнута. Было разработан алгоритм и проверена его работоспособность на реальных данных. Но за рамками рассматриваемой темы осталось еще много других алгоритмов, например использование других типов сетей и методов их обучения. Алгоритм имеет большой потенциал для его оптимизации путем распараллеливания процессов, переноса вычислений на графические процессора или возможность задействования многоядерной архитектуры центрального процессора. Данным алгоритм можно применить для сжатия другой мультимедийной информации, для полного восприятия которой, человек не имеет достаточного количества биологических ресурсов. Эти задачи также являются нетривиальными и требуют детального изучения и проработки, они не рассматривались в данной работе из-за временных ограничений на их исследвание.

В дальнейшем планируется развивать и довести существующее ПО до полноценной библиотеки, способной решать более широкий класс задач, возникающих в области применения нейронных сетей и сжатия мультимедийной информации.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

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

[2] В.Г., Царегородцев. Нейронные сети, методы обработки и анализа данных [Электронный ресурс]. — Электронные данные. — 2015. — Режим доступа: http://www.neuropro.ru/. http://www.neuropro.ru/.

[3] В.Д., Романов. Интеллектуальные информационные системы в экономике. Учебное пособие / Романов В.Д. — Московский государственный университет печати, 2003.

[4] Вассерман., Ф. Нейрокомпьютерная техника: Теория и практика. / Ф. Вассерман. — М.: «Мир», 1992.

[5] Kohonen, T. Self-Organizing Maps (Third Extended Edition) / T. Kohonen. — New York, 2001.

[6] Розенблатт. Principles of Neurodynamic: Perceptrons and the Theory of Brain Mechanisms. / Розенблатт. — М.: Мир, 1965.

[7] Макконнелл, С. Совершенный код. Мастер-класс / Пер. с англ. / С. Макконнелл. — СПб. : Издательско-торговый дом «Русская редакция», 2005. — 896 с.

[8] Common Language Infrastructure (CLI). Partitions I to VI. — 2012. — June. http://www.ecma-international.org/publications/standards/Ecma-335.htm.

[9] Рихтер, Джеффри. CLR via C#. Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. — 2-е изд. — СПб. : Питер, Русская Редакция, 2007. — 656 с.

[10] Марченко, А. Л. Основы программирования на C# 2.0 / А. Л. Марченко. — БИНОМ. Лаборатория знаний, Интернет-университет информационных технологий — ИНТУИТ.ру, 2007. — 552 с.

[11] Richter, Jeffrey. CLR via C# / Jeffrey Richter. Microsoft, Developer Reference. — 4-th edition. — One Microsoft Way, Redmond, Washington 98052-6399 : Microsoft Press, 2012. — 896 P.

[12] Абельсон, Харольд. Структура и интерпретация компьютерных программ / Харольд Абельсон, Джеральд Джей Сассман, Джули Сассман. — Добросвет, 2006. — 608 с.

[13] Albahari, Joseph. C# 5.0 in a Nutshell / Joseph Albahari, Ben Albahari. — 5-th edition. — O'Reilly Media, Inc, 2012. — June. — 1062 P.

[14] C Sharp [Электронный ресурс]. — Электронные данные. — Режим доступа: http://ru.wikipedia.org/wiki/C_Sharp. — Дата доступа: 22.03.2013.

[15] Michaelis, Mark. The New and Improved C 6.0 / Mark Michaelis // MSDN Magazine. — 2014.

[16] Rektorys, K. Applicable mathematics / K. Rektorys. — Iliffe, 1969.

[17] Эрих Гамма Ричард Хелм, Ральф Джонсон Джон Влиссидес. Приемы объектно-ориентированного проектирования. Паттерны проектирования / Ральф Джонсон Джон Влиссидес Эрих Гамма, Ричард Хелм. — Питер, 2012.

[18] Томас Х. Кормен Чарльз И. Лейзерсон, Рональд Л. Ривест Клиффорд Штайн. Алгоритмы: построение и анализ. 2-е изд / Рональд Л. Ривест Клиффорд Штайн Томас Х. Кормен, Чарльз И. Лейзерсон. — М.: Вильямс, 2006.

[19] Козловская В.Б. Радкевич В.Н., Сацукевич В.Н. Электрическое освещение. Справочник. / Сацукевич В.Н. Козловская В.Б., Радкевич В.Н. — Минск, 2007.

[20] СВЕТОДИОДНАЯ ЛАМПА [Элетронный ресурс]. — Электронные данные. — Режим доступа: http://radioskot.ru/publ/svetodiody/svetodiodna-lampa/3-1-0-141. http://radioskot.ru/publ/svetodiody/svetodiodna_lampa/3-1-0-141.

[21] Российская энциклопедия по охране труда. — Изд-во НЦ ЭНАС, 2007.

ПРИЛОЖЕНИЕ А ЛИСТИНГ ПРОГРАММНОГО СРЕДСТВА

Листинг 8.1 — Дискретизатор глубиной 256 уровней

```
using System.Drawing;
using Neuro.Interfaces;

namespace Neuro.Implementations
{
    public class PixelConverterHigh : IPixelConverter
    {
        public Color ToPixel(double real)
        {
            int value = (int)((real + 1) * 128);
            value = value > 255 ? 255 : value;
            value = value < 0 ? 0 : value;
            return Color.FromArgb(255, value, value, value);
        }

        public double ToReal(Color pixel)
        {
            return ((pixel.B + 1)) * (1.0/128) - 1;
        }

        public double Epsilon
        {
            get { return 0.1; }
        }
    }
}
```

Листинг 8.2 — Преобразователь матрицы изображения в матрицу сигналов

```
using System;
using System.Drawing;
using Neuro.Interfaces;

namespace Neuro.Implementations
{
    public class ImageConverter : IImageConverter
    {
        private readonly int _width;
        private readonly int _height;
        private readonly int _blockSize;
        private readonly IPixelConverter _converter;

        public ImageConverter(int width, int height, int blockSize,
            IPixelConverter converter)
        {
            if (width%blockSize + height%blockSize != 0)
            {
```

```
            throw new ArgumentException("invalid width, height and
                blockSize");
        }

        _width = width;
        _height = height;
        _blockSize = blockSize;

        if (converter == null)
        {
            throw new ArgumentNullException("converter");
        }

        _converter = converter;
    }

    public Bitmap ToBitmap(double[][] signals)
    {
        int n = _width / _blockSize;
        int m = _height / _blockSize;
        int size = _blockSize * _blockSize;
        Bitmap result = new Bitmap(_width,_height);

        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                double[] row = signals[i * m + j];

                for (int ii = 0; ii < _blockSize; ii++)
                {
                    for (int jj = 0; jj < _blockSize; jj++)
                    {
                        result.SetPixel(i*_blockSize + ii, j
                            *_blockSize + jj, _converter.
                            ToPixel(row[ii*_blockSize + jj])
                            );
                    }
                }
            }
        }

        return result;
    }

    public double[][] ToSignals(Bitmap image)
    {
        int n = _width/_blockSize;
        int m = _height/_blockSize;
        int size = _blockSize*_blockSize;
        double[][] result = new double[n*m][];

        for (int i = 0; i < n; i++)
```

```
            {
                for (int j = 0; j < m; j++)
                {
                    double[] row = new double[size];

                    for (int ii = 0; ii < _blockSize; ii++)
                    {
                        for (int jj = 0; jj < _blockSize; jj++)
                        {
                            row[ii*_blockSize + jj] = _converter
                                .ToReal(image.GetPixel(i*
                                _blockSize + ii, j*_blockSize +
                                jj));
                        }
                    }

                    result[i*m+j] = row;
                }

            }
            return result;
        }

        public double Epsilon
        {
            get { return _converter.Epsilon; }
        }

        public int BlockSize
        {
            get { return _blockSize; }
        }
    }
}
```

Листинг 8.3 — Обучение нейронной сети

```
using System;
using AForge.Neuro;
using AForge.Neuro.Learning;
using Neuro.Exceptions;
using Neuro.Interfaces;
using Neuro.Implementations;

namespace Neuro.Helpers
{
    public static class NeuroHelper
    {
        private const double Alpha = 0.4;
        private const int IterationsLimit = 100000;
        private const int FailsLimit = 1000;

        public static ActivationNetwork GetActivationNetwork(
            int inputBlockSize,
```

```
        int outputBlockSize,
        double epsilon,
        double[][] input,
        double[][] output)
{
        ActivationNetwork net = new ActivationNetwork(
                new BipolarSigmoidFunction(Alpha),
                inputBlockSize,
                outputBlockSize,
                inputBlockSize);

        BackPropagationLearning trainer = new
           BackPropagationLearning(net);

        double error = double.MaxValue;
        trainer.LearningRate = 1;
        int iterations = 0;
        double prevError = error;
        int failsCount = 0;

        while (error > epsilon)
        {
                prevError = error;
                error = trainer.RunEpoch(input, output);
                iterations++;

                if (iterations > IterationsLimit)
                {
                        return net;
                }

                if (prevError <= error)
                {
                        failsCount++;
                        trainer.LearningRate *= 0.95;
                        if (trainer.LearningRate < 0.1)
                        {
                                trainer.LearningRate = 1;
                        }
                }

                if (failsCount > FailsLimit)
                {
                        return net;
                }
        }
        return net;
}

public static IDecoder ExtractDecoder(ActivationNetwork net)
{
        Layer layer = net.Layers[1];
        int inputCount = layer.InputsCount;
```

```
            int outputCount = layer.Neurons.Length;

            ActivationNetwork decoderNetwork = new ActivationNetwork(
                  new BipolarSigmoidFunction(Alpha),
                  inputCount,
                  outputCount);

            Layer decoderLayer = decoderNetwork.Layers[0];
            for (int i = 0; i < outputCount; i++)
            {
                  Neuron decoderNeuron = decoderLayer.Neurons[i];
                  Neuron neuron = layer.Neurons[i];

                  for (int j = 0; j < inputCount; j++)
                  {
                        decoderNeuron.Weights[j] = neuron.Weights[j];
                  }
            }

            return new Decoder(decoderNetwork);
      }

      public static IEncoder ExtractEncoder(ActivationNetwork net)
      {
            Layer layer = net.Layers[0];
            int inputCount = layer.InputsCount;
            int outputCount = layer.Neurons.Length;

            ActivationNetwork encoderNetwork = new ActivationNetwork(
                  new BipolarSigmoidFunction(Alpha),
                  inputCount,
                  outputCount);

            Layer encoderLayer = encoderNetwork.Layers[0];
            for (int i = 0; i < outputCount; i++)
            {
                  Neuron decoderNeuron = encoderLayer.Neurons[i];
                  Neuron neuron = layer.Neurons[i];

                  for (int j = 0; j < inputCount; j++)
                  {
                        decoderNeuron.Weights[j] = neuron.Weights[j];
                  }
            }

            return new Encoder(encoderNetwork);
      }
   }
}
```

АННОТАЦИЯ

на дипломную работу «Многослойные перцептроны для сжатия изображений» студента УО «Белорусский государственный университет информатики и радиоэлектроники» Сафонова А. А.

Ключевые слова: НЕЙРОННЫЕ СЕТИ; СЖАТИЕ С ПОТЕРЯМИ; ЧАСТОТА ДИСКРЕТИЗАЦИИ; КРИТЕРИИ СРАВНЕНИЯ ИЗОБРАЖЕНИЙ.

Дипломная работа выполнена на 6 листах формата А1 с пояснительной запиской на 76 страницах, без приложений справочного или информационного характера. Пояснительная записка включает 8 глав, 32 рисунков, 2 таблиц, 44 формулы, 21 литературный источник.

Целью дипломной работы является разработка алгоритма, пригодного для решения практических задач, возникающих в реальных проектах.

Для достижения цели дипломной работы был разработан алгоритм, предназначенный для сжатия графической или любой другой мультимедийной информации.

Во введении производится ознакомление с проблемой, решаемой в дипломной работе.

В первой главе производится обзор предметной области проблемы решаемой в данной дипломной работе.

Во второй главе производится краткий обзор технологий, использованных для реализации прототипа ПО в рамках дипломной работы.

В третьей главе производится постановка задачи для данного исследования.

В четвертой главе производится поэтапное описание создания алгоритма, созданного в рамках данной дипломной работы.

В пятой главе производится описание созданного прототипа и архитектурные решения используемые при его создании.

В шестой главе производится оценка полученных результатов, приводятся графики и примеры.

В седьмой главе производится оценка обеспечения безопасных условий труда при проведении исследования.

В восьмой главе производится технико-экономическое обоснование эффективности данной разработки.

В заключении подводятся итоги и делаются выводы по дипломной работе, а также описывается дальнейший план развития проекта.