Министерство науки и высшего образования

Российской федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

«К защите»

Заведующий кафедрой КТ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.И. Гусев

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2022 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ**

**РАБОТА**

**(магистратура)**

на тему

«Алгоритмы и программное обеспечение распаковки спутниковых данных и анализа их целостности»

Направление подготовки: *09.04.01, Информатика и вычислительная техника*

Наименование ОПОП: *Космические информационные системы и технологии*

Руководитель ОПОП \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Таганов А.И. )

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Москвитин А.Э. )

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Фоломеев К.Д. )

Рязань 2022

**Аннотация**

Выпускная квалификационная работа изложена на 96 страницах и состоит из введения, пяти глав и заключения. Работа включает 30 рисунков, 9 таблиц, список использованных источников из 26 элементов, 2 приложения на 22 страницах.

Объектом выпускной квалификационной работы является разработка программного обеспечения для нейросетевого анализа спутниковых изображений со сбойными столбцами пикселей.

Предметом работы являются современные методы распаковки спутниковых данных и анализа их целостности.

Цель выпускной квалификационной работы – изучение существующих алгоритмов распаковки спутниковых данных и анализа их целостности, а также разработка программного обеспечения для нейросетевого анализа спутниковой визуальной информации.

При разработке программного обеспечения использовались следующие средства:

* языки программирования *C++14*, *Python 3.8*;
* набор сторонних библиотек;
* интегрированная среда разработки *Microsoft Visual Studio 2022*;
* система контроля версий *git*.

Результаты выпускной квалификационной работы могут быть использованы в дальнейших работах и исследованиях.

**Annotation**

The graduation work consists of 96 pages and contains an introduction, five chapters and a conclusion. It includes 30 figures, 9 tables, a reference list with 26 elements, 2 annexes on 22 pages.

The object of the graduation work is development of a software for neural analysis of corrupted pixel columns found in satellite imagery.

The subject of the work is modern methods for decompression and integrity analysis of satellite data.

The purpose of the work is to explore existing methods for satellite data decompression and analysis of satellite data integrity and to develop a software for neural analysis of satellite imagery.

The following tools were used for software development:

* *C++14* and *Python 3.8* programming languages;
* a set of third-party libraries;
* *Microsoft Visual Studio 2022* integrated development environment;
* an open-source distributed version control system named *git*.

The results of the graduation work may be used in future works and research.

содержание

[ВВЕДЕНИЕ 7](#_Toc106704204)

[1 ОБЗОР МЕТОДОВ РАСПАКОВКИ И АНАЛИЗА ЦЕЛОСТНОСТИ СПУТНИКОВЫХ ДАННЫХ 10](#_Toc106704205)

[1.1 Структура информационного потока 10](#_Toc106704206)

[1.2 Методы анализа целостности спутниковой информации 12](#_Toc106704207)

[1.2.1 Статистический анализ спутниковой информации 13](#_Toc106704208)

[1.2.2 Нейросетевой анализ спутниковой информации 15](#_Toc106704209)

[2 МАТЕМАТИЧЕСКИЙ АППАРАТ МОДЕЛИ ГЛУБОКОГО ОБУЧЕНИЯ 17](#_Toc106704210)

[2.1 Нейронные сети 17](#_Toc106704211)

[2.2 Распараллеливание обучения нейронной сети 20](#_Toc106704212)

[3 СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 25](#_Toc106704213)

[3.1 Обзор языка программирования *Python* 25](#_Toc106704214)

[3.2 Обзор языка программирования *C++* 27](#_Toc106704215)

[3.3 Матричная библиотека *Eigen* 31](#_Toc106704216)

[3.4 Библиотека интеграции *C++* программ *Pybind11* 35](#_Toc106704217)

[3.5 *Pybind11* и *Eigen* 40](#_Toc106704218)

[3.6 Технология распараллеливания *OpenMP* 42](#_Toc106704219)

[3.6.1 Модель Исполнения 42](#_Toc106704220)

[3.6.2 Модель Памяти 43](#_Toc106704221)

[3.6.3 Директивы 45](#_Toc106704222)

[3.6.4 Опции 47](#_Toc106704223)

[3.6.5 Библиотечный Функционал и Переменные Окружения 50](#_Toc106704224)

[3.7 Система контроля версий *git* 52](#_Toc106704225)

[3.8 Интегрированная среда разработки *Microsoft Visual Studio* 53](#_Toc106704226)

[4 ОБЗОР РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ 56](#_Toc106704227)

[4.1 Руководство оператора 56](#_Toc106704228)

[4.1.1 Назначение программы 56](#_Toc106704229)

[4.1.2 Условия выполнения программы 57](#_Toc106704230)

[4.1.3 Выполнение программы 57](#_Toc106704231)

[4.1.4 Сообщения оператору 65](#_Toc106704232)

[4.2 Пример сеанса работы с программой 68](#_Toc106704233)

[4.2.1 Подготовка к работе 68](#_Toc106704234)

[4.2.2 Фрагментация исходного изображения 69](#_Toc106704235)

[4.2.3 Синтез шума 70](#_Toc106704236)

[4.2.4 Инициализация нейронной сети 71](#_Toc106704237)

[4.2.5 Вывод конфигурации нейронной сети в консоль 72](#_Toc106704238)

[4.2.6 Обучение нейронной сети 73](#_Toc106704239)

[4.2.7 Тестирование нейронной сети 77](#_Toc106704240)

[4.2.8 Получение ответа модели на входное изображение 79](#_Toc106704241)

[4.2.9 Сведения о системе 80](#_Toc106704242)

[5 НЕЙРОСЕТЕВОЙ АНАЛИЗ ИЗОБРАЖЕНИЙ 82](#_Toc106704243)

[5.1 Фрагментация исходного изображения 83](#_Toc106704244)

[5.2 Анализ исходного изображения и зашумление фрагментов 84](#_Toc106704245)

[5.3 Архитектура нейронной сети 87](#_Toc106704246)

[5.4 Результаты обучения и тестирования моделей 88](#_Toc106704247)

[ЗАКЛЮЧЕНИЕ 93](#_Toc106704248)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 94](#_Toc106704249)

[ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД НЕЙРОСЕТЕВОГО МОДУЛЯ 97](#_Toc106704250)

[Заголовочный файл нейронной сети и вспомогательных функций *NeuralNetwork.h* 97](#_Toc106704251)

[Заголовочный файл генератора случайных чисел *Rnd.h* 98](#_Toc106704252)

[Исходный код нейронной сети *NeuralNetwork.cpp* 99](#_Toc106704253)

[Исходный код вспомогательных функций *Tools.cpp* 103](#_Toc106704254)

[Исходный код сериализации конфигурации нейронной сети *Serialize.cpp* 105](#_Toc106704255)

[Исходный код распараллеливания обучения нейронной сети *Parallel.cpp* 106](#_Toc106704256)

[Исходный код взаимодействия копий нейронной сети *State.cpp* 108](#_Toc106704257)

[Исходный код инициализации терминатора строки *VTMode.cpp* 108](#_Toc106704258)

[Исходный код сборки *Python*-модуля *Bind.cpp* 109](#_Toc106704259)

[ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД МОДУЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ И ИНТЕРФЕЙСА КОМАНДНОЙ СТРОКИ 110](#_Toc106704260)

[Исходный код интерфейса командной строки *CLI.py* 110](#_Toc106704261)

[Исходный код загрузки из файла конфигурации параметров по умолчанию *Defaults.py* 113](#_Toc106704262)

[Исходный код взаимодействия с файлом конфигурации *HandleConfig.py* 114](#_Toc106704263)

[Исходный код приведения данных для нейронной сети к векторному и матричному виду *GetData.py* 116](#_Toc106704264)

[Исходный код модуля обработки изображений *Imagery.py* 117](#_Toc106704265)

# ВВЕДЕНИЕ

**Тема выпускной квалификационной работы** – исследование алгоритмов распаковки и анализа целостности спутниковых данных.

*Искусственный спутник Земли* – объект, намеренно помещённый на орбиту Земли. Объекты называются *искусственными*, чтобы можно было их отличить от *естественных спутников*, таких как Луна. Первый искусственный спутник был запущен с территории СССР 4 октября 1957 года и имел название *Спутник 1*. С тех пор более 40 стран запустили около 8900 спутников. На 2022 год на орбите Земли находятся около 5000 спутников.

Спутники могут использоваться для выполнения различных задач, таких как картография, спутниковая съёмка, наблюдение в реальном времени, коммуникация, навигация, метеорология, дистанционное зондирование. Спутники могут функционировать как самостоятельные единицы или элементы более глобальных систем, например спутниковых созвездий или группировок.

Орбиты спутника варьируются в зависимости от выполняемых задач и имеют несколько классификаций, к наиболее известной можно отнести классификацию по орбитам: низкая, полярная, геостационарная.

Для запуска спутников на орбиту используют ракеты. Запуск осуществляется со стартовых площадок, располагающихся в основном на суше. Площадки могут также оборудоваться на мобильных наводных платформах или на борту самолёта.

Обычно спутники представляют собой полуавтономные системы, контролируемые компьютером. Подсистемы спутника выполняют множество функций, например генерация энергии, контроль температуры, телеметрия, контроль положения в пространстве, управление научной аппаратурой, коммуникация и т. п.

В данной работе используются данные (изображения), полученные спутником наблюдения за Землёй, для решения задач дистанционного зондирования.

*Дистанционное зондирование* – процесс сбора информации об объекте или явлении без непосредственного контакта с предметом интереса. Термин особенно распространён в отношении наблюдения за Землёй и другими планетами. Дистанционное зондирование применяется во множестве сфер, таких как география, топографическая съёмка, гидрология, экология, метеорология, океанография, геология и др. Такой подход сбора данных находит применение в разведке, экономике, планировании, а также гуманитарной, военной и коммерческой деятельности.

Термин «дистанционное зондирование» обычно применяется к спутниковым и воздушным датчикам, используемым для определения и классификации наземных объектов. Область интереса таких датчиков обычно включает атмосферу, океаны и сушу, в зависимости от свойств распространяемого сигнала. Дистанционное зондирование может быть «активным», при котором излучаемый аппаратом сигнал отражается от объекта и принимается датчиком, и «пассивным», при котором источником отражаемого излучения является какой-либо другой объект, например Солнце (рисунок 1).

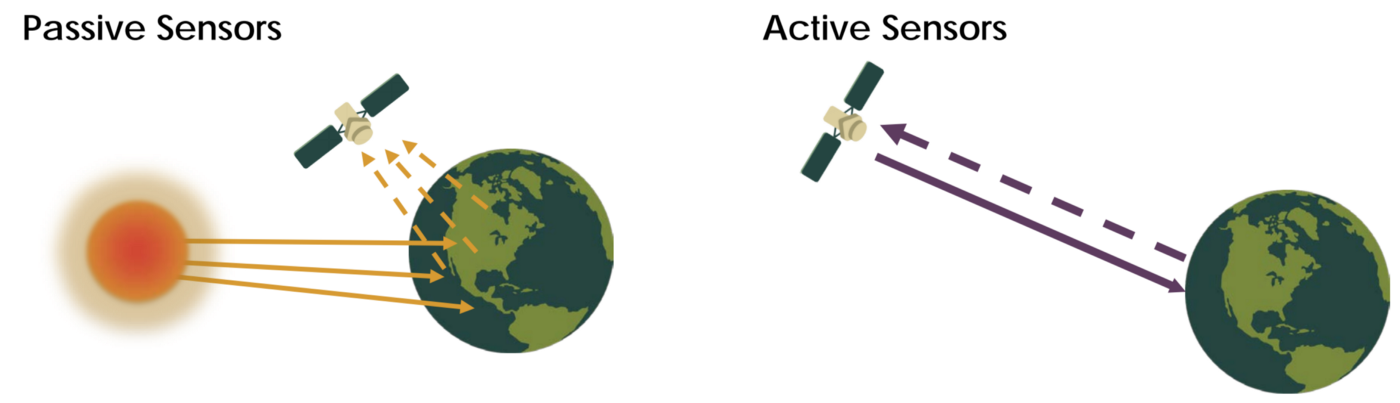


Рисунок 1. Пассивное (слева) и активное (справа) дистанционное зондирование

Одной из задач при работе с данными дистанционного зондирования является распаковка спутниковых данных и анализ их целостности. В рамках данной работы проводится исследование существующих и перспективных алгоритмов анализа целостности спутниковых данных, а также программная реализация этих алгоритмов.

Таким образом, **цель выпускной квалификационной работы** – изучить существующие алгоритмы распаковки спутниковых данных и анализа их целостности, а также разработать программное обеспечение для реализации анализа спутниковой визуальной информации.

Пояснительная записка к дипломному проекту состоит из введения, пяти разделов, заключения и приложений. Первый раздел содержит описание структуры информационного потока и обзор существующих методов анализа целостности спутниковых данных. Во втором разделе приведено описание математического аппарата модели глубокого обучения и методов распараллеливания обучения нейронной сети. Третий раздел содержит перечень использованных при разработке программного обеспечения средств. В четвёртом разделе представлено руководство оператора разработанного программного обеспечения и приведён пример возможного сеанса работы с программой. В пятом разделе описан ход решения поставленной задачи анализа целостности спутниковой информации, а также представлены результаты выполненной работы. В заключении сделаны выводы о проделанной работе и предложены варианты дальнейшего развития достигнутых результатов.

# ОБЗОР МЕТОДОВ РАСПАКОВКИ И АНАЛИЗА ЦЕЛОСТНОСТИ СПУТНИКОВЫХ ДАННЫХ

## Структура информационного потока

Информационный поток, поступающий со спутника, включает видеоданные нескольких спектральных каналов, служебные и телеметрические данные. Структура спектральных каналов представлена в таблицах 1.1 – 1.3.

Таблица 1.1. Структура первого спектрального канала

|  |  |  |  |
| --- | --- | --- | --- |
| Число строк | № строк | Коэф. усиления | Вид информации |
| 512 | 0…511 | – | Тест 1[[1]](#footnote-1) |
| 286 | 512…797 | К1 | Тест 2[[2]](#footnote-2) |
| 12400 | 798…13197 | К1 | Видеоинформация |
| 4002 | 13198…17199 | К10 |

Таблица 1.2. Структура второго спектрального канала

|  |  |  |  |
| --- | --- | --- | --- |
| Число строк | № строк | Коэф. усиления | Вид информации |
| 512 | 0…511 | – | Тест 1 |
| 286 | 512…797 | К1 | Тест 2 |
| 4002 | 798…4799 | К10 | Видеоинформация |
| 12400 | 4800..17199 | К1 |

В таблице 1.4 представлена структура строки на выходе модуля видимого диапазона.

Таблица 1.3. Структура третьего спектрального канала

|  |  |  |  |
| --- | --- | --- | --- |
| Число строк | № строк | Коэф. усиления | Вид информации |
| 512 | 0…511 | – | Тест 1 |
| 286 | 512…797 | К1 | Тест 2 |
| 16402 | 798…17199 | К1 | Видеоинформация |

Таблица 1.4. Структура строки на выходе модуля видимого диапазона

|  |  |  |
| --- | --- | --- |
| № элемента | Значения | Информация |
| 0…170 | 0…255 | Служебная информация |
| 171…12170 | 0…1023 | Видеоинформация |
| 12171…12176 | 0…255 | Целевой массив информации |
| 12177 | 255 | Наличие ИНС |
| 0 | Отсутствие ИНС |
| 12178 | 255 | Наличие Готов |
| 0 | Отсутствие Готов |
| 12179 | 0…255 | Телеметрия сигнала |
| 12180 | 255 | Съёмка |
| 0 | Калибровка |

Информация в модуль инфракрасного канала поступает одновременно с семи каналов и формируется в общий информационный поток, строки которого состоят из служебной и видеоинформации, передаваемой поочерёдно с каждого из семи каналов. Информационный поток состоит из 4 кадров. Структура кадров информационного потока в модуле инфракрасного канала представлена в таблице 1.5. Структура строки на выходе модуля инфракрасного канала отражена в таблице 1.6.

Таблица 1.5. Структура кадров информационного потока в модуле ИК

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Число строк | № строки | Вид информации | № скана | № кадра |
| 1700 | 0…1699 | Тест 1 | 0-й скан | 1…4 |
| 1700 | 1700…3399 | Тест 2 |
| 3400 | 0…3399 | Видеоинформация | сканы 1…35 |

Таблица 1.6. Структура строки на выходе модуля ИК

|  |  |  |
| --- | --- | --- |
| № элемента | Значения | Информация |
| 1…48 | 0…255 | Служебная информация |
| 49…140 | 0…255 | Резерв |
| 141…1484 | 0…1023 | Тест/видеоинформация |

## Методы анализа целостности спутниковой информации

Для проверки информационного потока необходимо решить следующие задачи:

* приём, декодирование и восстановление транспортного потока;
* разделение целевой информации по видам;
* контроль полноты и целостности информации;
* просмотр видеоинформации по отдельным спектральным каналам;
* изменение кодов яркости отдельных пикселей изображений;
* выделение областей интереса различной формы;
* определение статистических параметров изображения или фрагмента;
* просмотр яркостных срезов по строке или столбцу;
* контрастирование, масштабирование и скроллирование изображений;
* табличное представление служебной информации;
* вывод фрагментов данных и сохранение файлов графического формата.

Полнота данных анализируется путём контроля присутствия всех видов информации и её объёма. Для изображения каждого спектрального канала определяются значения таких параметров, как имя файла, размер файла, смещение на данные, номер канала, общее число строк, число строк в тестовых наборах и видеоданных, общий размер строки, количество элементов видеоданных в строке, число сканов, число строк телеметрической и служебной информации и т. п. Вывод о полноте данных осуществляется путём сопоставления значений с протокольными.

### Статистический анализ спутниковой информации

При статистическом подходе целостность информации определяется путём оценки вероятности сбоев параметров служебной информации, значения которых заранее известны, и определения среднего значения, среднеквадратического отклонения и вероятности выхода за допустимый диапазон динамически изменяющихся служебных данных.

Оценка вероятности сбоев выполняется для параметров трёх групп.

* 1-я группа.

Параметры, принимающие только одно заранее известное значение (*Xconst*). Для таких параметров вероятность сбоев оценивается как:

|  |  |
| --- | --- |
| ,  где , , ,  – значения анализируемого параметра. | (.) |

* 2-я группа.

Параметры, принимающие несколько заранее известных значений . Здесь сначала определяется истинное значение параметра как приближение к его среднему значению:

|  |  |
| --- | --- |
| ,  где , , . | (1.2) |

Далее оценка вероятности сбоя происходит также, как и для параметра 1-й группы.

* 3-я группа.

Параметры, для которых известен только закон изменения, например номер строки. При анализе номера строки полагается, что при сбоях возможны как пропуск строк, так и запись ложных значений. Работа алгоритма начинается с поиска индекса, от которого идёт корректный рост номера строки:

|  |  |
| --- | --- |
| , | (.) |

– размер участка с корректным ростом. Далее выполняется подсчёт верных значений параметра:

|  |  |
| --- | --- |
| ,  где , . | (.) |

Искомая вероятность сбоя:

|  |  |
| --- | --- |
| . | (.) |

Анализ динамики сигналов служебной информации осуществляется путём вычисления:

* среднего значения , где () – значение анализируемого сигнала;
* среднеквадратического отклонения ;
* вероятности выхода за допустимый диапазон , где , .

В качестве оценки *математического ожидания* случайной величины используется выборочное среднее. Пусть для случайной величины *x* дана случайная выборка . Выборочным средним значением величины *x* называется . Если случайная величина *x* имеет дисперсию, то значение становится близким к математическому ожиданию случайной величины *x*.

В теории вероятностей и статистики *среднеквадратическое отклонение* – наиболее распространённый показатель рассеивания значений случайной величины относительно её математического ожидания (аналога среднего арифметического с бесконечным числом исходов). Обычно он приравнивается к квадратному корню из дисперсии случайной величины, но иногда может выражаться другими способами. В научной литературе обычно обозначается греческой буквой **σ** (сигма). Математически среднеквадратическое отклонение можно записать следующим образом:

|  |  |
| --- | --- |
| . | (.) |

*Доверительный интервал* – термин, используемый в математической статистике при интервальной оценке статистических параметров, более предпочтительной при небольшом объёме выборки, чем точечная. Доверительным называют интервал, покрывающий неизвестный параметр с заданной точностью. Доверительным также называют интервал, в который попадают измеренные в эксперименте значения, соответствующие доверительной вероятности.

Доверительным интервалом параметра **ϴ** распределения случайной величины *X* с уровнем доверия *p*, порождённым выборкой , называется интервал с границами и , которые являются реализациями случайных величин и , таких, что .

### Нейросетевой анализ спутниковой информации

По мере развития технологий искусственного интеллекта в работе со спутниковыми данными всё чаще применяются различные архитектуры нейронных сетей. При наличии большой выборки данных нейронные сети можно обучить анализировать спутниковую информацию и проводить корректировку сбоев в контрольной информации согласно протокольным данным. Кроме того, в настоящее время нейронные сети набирают актуальность при работе с визуальной информацией (изображения и видео): поиск сбойных строк и столбцов, восстановление изображений и т. п.

Основной проблемой такого подхода является необходимость в наличии большого объёма данных для формирования выборки. Сбор достаточного количества информации может быть особенно трудоёмким, если речь идёт об изображениях и видеоинформации. Решением подобной проблемы может послужить фрагментация изображений или видеофрагментов, однако в таком случае разрешение графических фрагментов, с которыми работает нейронная сеть, становится значительно ниже.

В данной работе анализ целостности спутниковой информации проводится с помощью нейронной сети, а выборка данных собирается путём фрагментации спутникового изображения высокого разрешения и искусственного синтеза сбойных столбцов пикселей.

# МАТЕМАТИЧЕСКИЙ АППАРАТ МОДЕЛИ ГЛУБОКОГО ОБУЧЕНИЯ

## Нейронные сети

Нейро́нная сеть (также искусственная нейронная сеть, ИНС) – математическая модель, а также её программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. После разработки алгоритмов обучения получаемые модели стали использовать в практических целях: в задачах прогнозирования, для распознавания образов, в задачах управления и др.

ИНС представляет собой систему соединённых и взаимодействующих между собой элементарных процессоров (искусственных нейронов). Такие процессоры обычно довольно просты, особенно в сравнении с процессорами, используемыми в персональных компьютерах. Каждый процессор такой сети имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим процессорам. И, тем не менее, будучи соединёнными в достаточно большую сеть с управляемым взаимодействием, такие по отдельности простые процессоры вместе способны выполнять довольно сложные задачи.

С точки зрения машинного обучения, нейронная сеть представляет собой частный случай методов распознавания образов, дискриминантного анализа, методов кластеризации и т.п. С точки зрения искусственного интеллекта, ИНС является основой философского течения коннекционизма и основным направлением в структурном подходе по изучению возможности построения естественного интеллекта с помощью компьютерных алгоритмов.

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/или «зашумленных», частично искажённых данных.

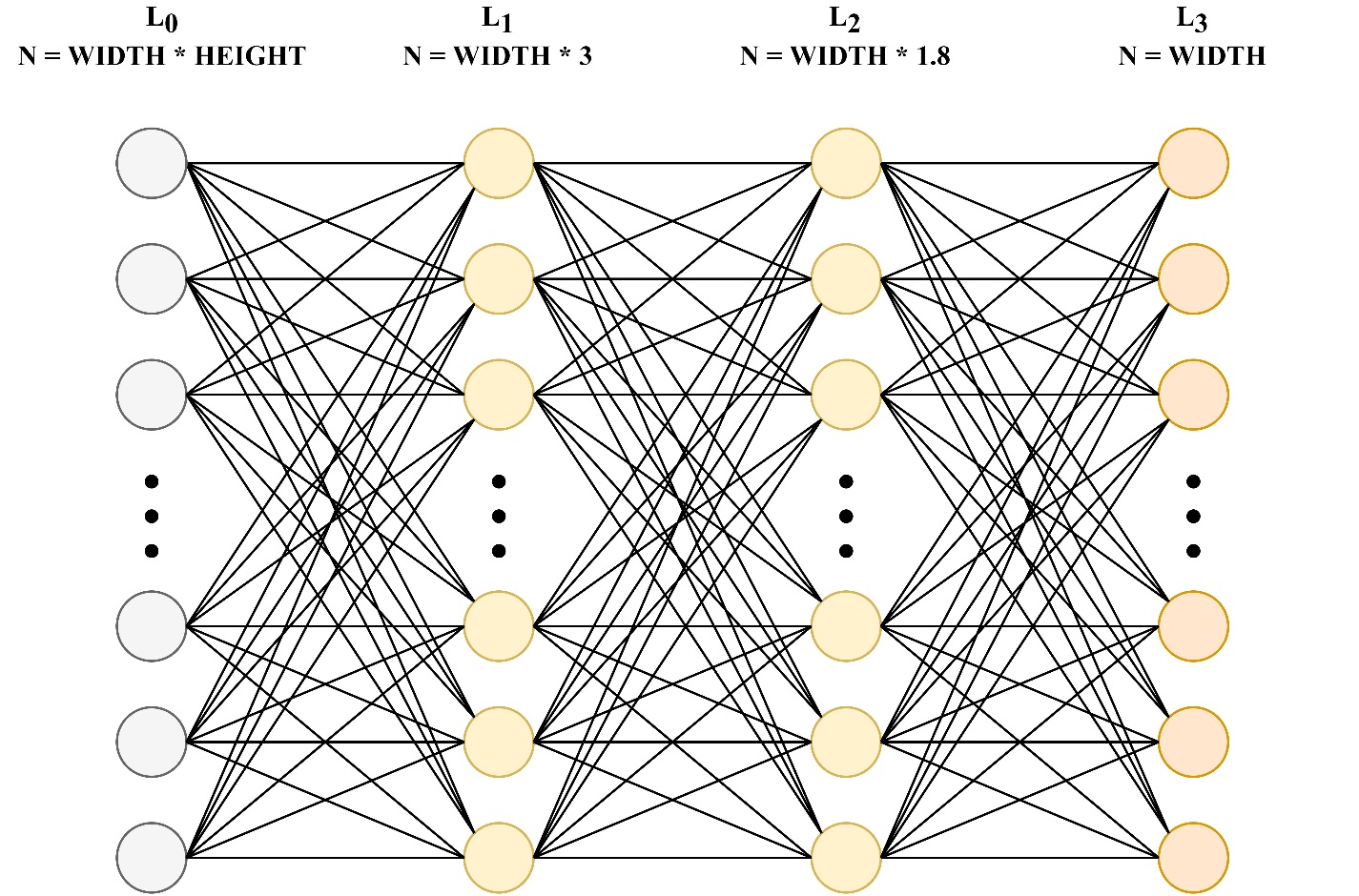


Рисунок 2.1. Пример структуры НС с входным, выходным и двумя скрытыми слоями

Стандартная архитектура нейронной сети (рисунок 2.1) подразумевает наличие входного, выходного и, возможно, скрытых слоёв, каждый из которых состоит из определённого числа нейронов, при этом, как правило, нейроны одного слоя не связаны друг с другом. Классический пример нейронной сети – *сеть прямого распространения*, значения на каждом нейроне которой вычисляются следующим образом:

|  |  |
| --- | --- |
| , | (.) |

где – весовой коэффициент связи между *j*-м нейроном слоя *l* и *k*-м нейроном слоя *l – 1*, – значение активационной функции в *j*-м нейроне *l­*-го слоя, – смещение в *j*-м нейроне *l­*-го слоя, – активационная функция. Для реализации вычислений в программном коде удобно использовать матричную форму записи выражения:

|  |  |
| --- | --- |
| . | (.) |

Наиболее типичным примером активационной функции является *сигмоида*:

|  |  |
| --- | --- |
| . | (2.3) |

Классический метод обучения нейронной сети с учителем подразумевает использование так называемого *алгоритма обратного распространения ошибки*, состоящего из нескольких шагов.

1. Нахождение ошибки на выходе нейронной сети, определяемой как адамарово произведение градиента стоимостной функции и вектора, компонентами которого являются производные соответствующей активационной функции по значению взвешенной суммы на соответствующем нейроне выходного слоя:

|  |  |
| --- | --- |
| , | (.) |

где стоимостная функция *C* может определяться как , *Y* – эталонный вектор для текущего примера обучающей выборки.

1. Последовательное нахождение ошибки в скрытых слоях в соответствии с выражением:

|  |  |
| --- | --- |
| . | (.) |

1. Вычисление частных производных для определения шага градиентного спуска по в соответствии с выражением:

|  |  |
| --- | --- |
| . | (.) |

1. Вычисление частных производных для определения шага градиентного спуска по согласно выражению:

|  |  |
| --- | --- |
| . | (.) |

Шаг градиентного спуска определяет величину, на которую надо изменить соответствующий параметр текущей конфигурации нейронной сети, чтобы уменьшить значение стоимостной функции для конкретного эталонного вектора обучающей выборки. При *онлайн*-обучении параметры сети регулируются под каждую пару векторов *входной вектор – эталонный вектор* отдельно. «Чувствительность» алгоритма обучения сети к каждому элементу обучающей выборки можно регулировать соответствующим коэффициентом *η*, определяющим, как сильно отдельный элемент выборки влияет на конфигурацию нейронной сети. При *оффлайн­*-обучении конфигурация сети регулируется после прогона всей или части обучающей выборки, а шаг градиентного спуска вычисляется как среднее арифметическое частных производных для каждого из примеров всей или части обучающей выборки.

Таким образом, при *онлайн*-обучении значения весов и смещений вычисляются многократно в соответствии с выражениями:

|  |  |
| --- | --- |
|  | (.) |
| . | (2.9) |

При *оффлайн*-обучении регулировка параметров сети происходит единожды и соответствует выражениям:

|  |  |
| --- | --- |
|  | (2.10) |
| . | (2.11) |

## Распараллеливание обучения нейронной сети

**Параллелизм модели** – разбиение модели между устройствами или узлами. Увеличение размера модели глубокого обучения (слои и параметры) способствует повышению точности результата. Однако размеры нейронной сети ограничены максимальным объёмом данных, умещаемым на одном устройстве. Подобные ограничения объёма могут являться узким местом при обучении.

* Ограниченный размер модели. Один из основных способов повышения точности при обучении – увеличение глубины и ширины нейронной сети. При решении сложных задач необходимый размер модели может оказаться слишком большим, чтобы уместиться на одном вычислительном узле.
* Ограничение размера партии элементов обучающей выборки, что может замедлить процесс обучения.

Распараллеливание модели позволяет распределить нейронную сеть между вычислительными узлами и проводить обучение на множестве устройств (рисунок 2.2).

Основной проблемой при параллелизме модели является простаивание вычислительных узлов. Так, например, устройство 1 после передачи данных на устройство 2 должно ждать, пока все остальные устройства не закончат вычисления. Для решения проблемы можно прибегнуть к *асинхронному параллелизму модели* с помощью конвейеризации вычислительных операций (рисунок 2.3). При таком подходе время простоя на каждом из устройств сводится к минимуму. После передачи данных на следующее устройство узел тут же получает новую порцию данных для обработки.

Параллелизм модели целесообразно использовать не столько для ускорения обучения, сколько для обеспечения возможности обучения очень больших нейронных сетей. Подобная концепция подразумевает высокую интенсивность обмена данными между вычислительными узлами, что может существенно замедлить прогресс при обучении модели.

При **параллелизме данных** происходит одновременное обучение сразу нескольких идентичных моделей (копий) c периодической или единовременной синхронизацией (рисунок 2.4). Каждая из копий модели может обучаться на своём фрагменте обучающей выборки или на всей выборке, доступной каждой из копий. Синхронизация моделей может быть осуществлена одним из нескольких способов:

* усреднение параметров, например весовых коэффициентов или шагов градиентного спуска;
* «голосование», при котором конечные выходные значения подбираются исходя из выходных значений на каждой из независимых моделей.

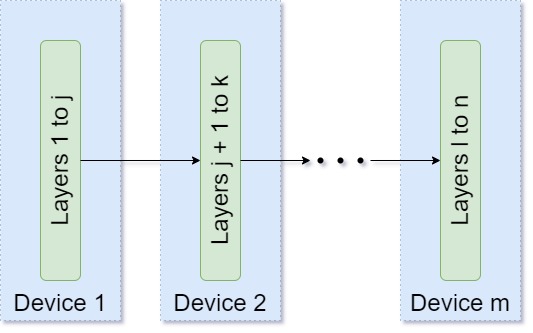


Рисунок 2.2. Параллелизм модели. На схеме n слоёв нейронной сети распределены на m вычислительных устройствах. На каждом устройстве можно разместить один или несколько слоёв нейронной сети.

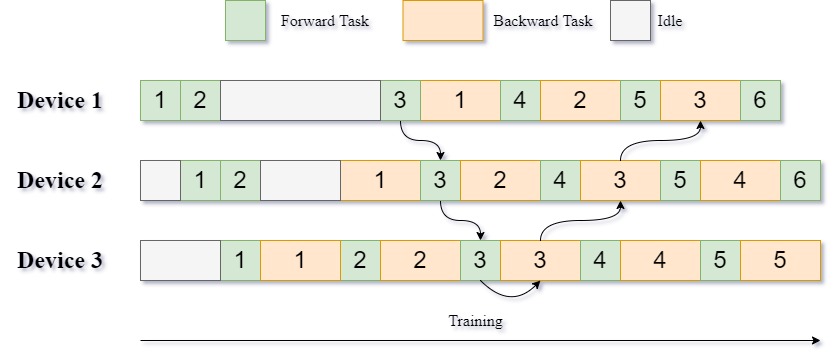


Рисунок 2.3. Асинхронный параллелизм модели, осуществлённый с помощью конвейеризации вычислительных операций

При усреднении весовых коэффициентов одна из моделей суммирует свои параметры со значениями других моделей и делит результат на количество копий (включая себя). При регулярном повторении процедуры по ходу обучения остальные копии заменяют свои весовые параметры на полученные усреднённые значения, после чего обучение продолжается до следующей синхронизации.

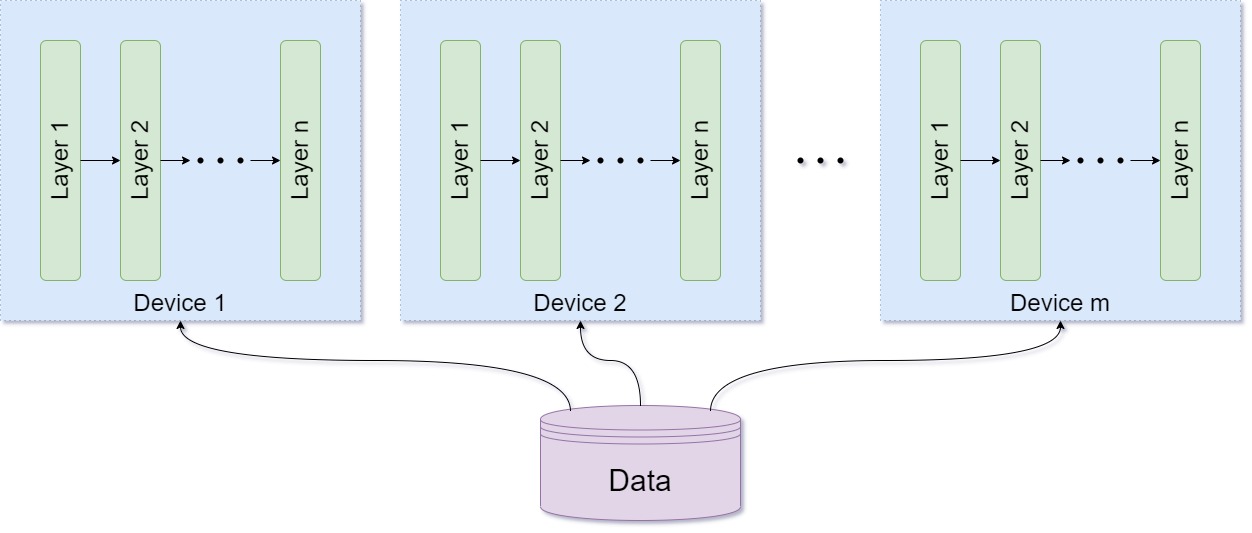


Рисунок 2.4. Параллелизм данных. Несколько идентичных копий одной и той же модели обучаются на своей части данных или на всей выборке целиком.

При таком подходе можно добиться существенного ускорения обучения за счёт:

* обучения копий с разными весовыми коэффициентами до первой синхронизации;
* увеличения параметра скорости обучения .

Поскольку изначально копии модели инициализируются собственными случайными значениями весов, до первой синхронизации каждая из копий обучается независимо от других копий, что обеспечивает градиентный спуск к глобальному экстремуму сразу с нескольких направлений. Кроме того, экспериментальным путём установлено, что при параллельном обучении нескольких моделей возможно установление бо́льшего значения параметра , чем при однопоточном обучении одной модели. В последнем случае высокая скорость обучения существенно ограничивает качество результата, поскольку из-за слишком большого значения при градиентном спуске алгоритм обучения просто «перепрыгивает» искомый экстремум. При параллельном обучении каждая из моделей может иметь обучаться с бо́льшим шагом без потери точности, что достигается за счёт усреднения весовых коэффициентов и параметров смещения .

Концепция «голосования» также подходит для нейронных сетей классификации. При таком подходе не происходит промежуточной синхронизации моделей между собой, копии никогда не взаимодействуют друг с другом и обучаются независимо на протяжении всего сеанса обучения. Ответ на входной вектор, поданный на каждую из копий, зависит от количества моделей, давших идентичный ответ, при этом требуемое число «голосов» при классификации данных может быть опциональным. Основной недостаток такого подхода – необходимость хранить в памяти компьютера все копии модели, в то время как при усреднении параметров по завершении обучения остаётся только одна копия.

В рамках данной работы выбрана наиболее оптимальная концепция – *параллелизм данных с усреднением конфигурации моделей*. Такой подход позволяет свести к минимуму число сеансов обмена данными между устройствами и количество хранимых копий модели в памяти компьютера. Экспериментально установлено, что выбранная концепция по сравнению с однопоточным обучением одной модели обеспечивает ускорение обучения в среднем на несколько порядков.

# СРЕДСТВА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Обзор языка программирования *Python*

*Python* (далее «Пайтон») представляет собой *объектно-ориентированный интерпретируемый* язык программирования. Будучи *языком общего назначения*, Пайтон включает пакет библиотек для различных целей: выполнение математических и матричных операций (*math*, *numpy*), построение графиков (*matplotlib*), работа с базами данных (*sqlite3*), работа с сетевыми протоколами (*socket*) и др. Существует и множество сторонних библиотек, расширяющих функционал языка: создание GUI (*wx*), работа с изображениями (*Pillow*), создание анимации и игр (*pygame*), разработка веб-приложений (*webapp2*) и др.

Среди основных черт языка Пайтон можно выделить следующие.

1. *Динамическая типизация*. При такой типизации переменная привязывается к типу данных не во время её объявления (что в принципе отсутствует в языке как таковое), а во время связывания переменной с некоторым значением. Данная особенность, во-первых, избавляет от необходимости объявлять переменную и тип присваиваемых данных перед включением в код, а во-вторых позволяет одну и ту же переменную в рамках одной программы связывать со значениями различных типов.
2. *Автоматическое управление памятью* (сборщик мусора). В Пайтоне реализован так называемый «сборщик мусора»: переменные автоматически удаляются из памяти, как только они перестают участвовать в выполнении программы.
3. *Интроспекция*. Возможность получения типа и структуры данных объекта в процессе выполнения программы. Под объектом подразумеваются в том числе и любые переменные, привязанные к значению конкретного типа, поскольку типы данных на языке Пайтон также представляют собой классы объектов, а данные – представителей этих классов. Например, целое число 5 является объектом класса *int* с соответствующей структурой (методами и атрибутами). Один из способов реализации интроспекции – функции *type* и *dir*.
4. *Обработка исключений*. Данная особенность позволяет предписывать поведение программы в случае возникновения ошибок (исключений). Осуществляется с помощью операторов *try* – *except* – *else* – *finally*.
5. *Поддержка многопоточных вычислений*. Позволяет разбивать выполнение программы на несколько потоков, обрабатываемых одновременно или в заранее описанном порядке. Инструменты реализации предоставляются, например, библиотекой *threading*.
6. *Структуры данных высокого уровня*. Будучи высокоуровневым языком программирования, Пайтон реализует высокоуровневые структуры данных – абстракции, описание которых на низкоуровневых языках или машинном коде требует больших усилий. В Пайтоне примерами таких структур являются списки и словари.
7. *Перегрузка операторов*. В отличие от таких популярных языков программирования, как *Java* и *JavaScript*, Пайтон поддерживает перегрузку операторов. Любой встроенный функционал (операторы арифметических операций, встроенные функции и т.п.) может быть описан заново. Например, при описании некоторого класса можно задать метод *\_\_str\_\_*, отвечающий за значение, возвращаемое функцией *print* в стандартный поток вывода при взятии объекта данного класса в качестве аргумента.
8. *Синтаксические особенности*. Отличительной чертой Пайтона, заметной с первого знакомства с языком, являются синтаксические особенности. Отсутствуют, например, операторные и фигурные скобки, определяющие блоки кода, а вместо них используются отступы.

Как и многие другие языки программирования, Пайтон поддерживает *полиморфизм*, *методы управления жизненным циклом объекта*, *метапрограммирование* и многое другое.

Как объектно-ориентированный язык программирования, Пайтон поддерживает механизмы *одиночного и множественного наследования*. При описании классов можно задавать «родительские» классы. В таком случае объект «дочернего» класса кроме собственных атрибутов и методов также «наследует» атрибуты и методы «родительских» классов.

Одним из главных недостатков языка Пайтон, как и других интерпретируемых языков, является низкое в сравнении с компилируемыми языками быстродействие. Компилятор обрабатывает весь код сразу и переводит его в машинный код, интерпретатор же делает это «на ходу», что, с одной стороны, даёт такие преимущества, как динамическая типизация, а с другой значительно уменьшает скорость выполнения программы.

Другим недостатком, в отличие от таких языков, как *Ruby*, является отсутствие возможности модификации или изменения встроенных классов: *object*, *int*, *list*. Но и у такого подхода есть свои положительные стороны, например уменьшение ресурсоёмкости программ.

В данную работу включены следующие сторонние библиотеки *Python*:

* *numpy* – инструмент для работы с матрицами,
* *Pillow* – библиотека для работы с изображениями,
* *psutil* – инструмент извлечения информации о текущих процессах и используемых системой ресурсах (ЦП, память, диски, сеть),
* *typer* – библиотека для построения интерфейса командной строки,
* *PyInstaller* – библиотека для сборки программного пакета в единый исполняемый файл.

## Обзор языка программирования *C++*

Поскольку вычисления, необходимые для выполнения алгоритмов нейронной сети, могут оказаться достаточно требовательными к вычислительным ресурсам, реализовать нейронную сеть было решено с помощью средств языка *C++*.

*С++* – *язык программирования общего назначения*, разработанный датским учёным Бьёрном Стра́уструпом в 1985 году в качестве расширения языка *C* поддержкой *объектно-ориентированного* подхода. Современный *C++* поддерживает также концепции *обобщённого* и *функционального* программирования и позволяет манипулировать памятью на низком уровне. Чаще всего реализуется как *компилируемый* язык программирования; компиляторы поставляются многими организациями, такими как *Microsoft*, *Intel*, *IBM* и др.

*C++* разрабатывался с акцентом на системное программирование и разработку интегрированных программных пакетов с ограниченными ресурсами. Основными приоритетами разработки языка являлись производительность, эффективность и гибкость использования. *C++* нашёл применение и во многих других сферах, таких как настольные приложения, видеоигры, серверное программное обеспечение и ориентированные на производительность приложения.

Для языка регулярно публикуются стандарты, разрабатываемые Международной Организацией по Стандартизации (англ. *ISO – International Organization for Standardization*). На 2022 год наиболее актуальным является стандарт *ISO/IEC 14882:2020*, более известный как *C++20*. Вместе с обновлением стандартов, как правило, расширяется и функционал языка. Так, стандарт *C++20* добавил в *C++* атрибуты *[[likely]]* и *[[unlikely]]*, позволяющие компилятору оптимизировать код для наиболее и наименее вероятных ветвей исполнения программы.

Язык *C++* заключает в себе две основных компоненты: унаследованное от *C* непосредственное взаимодействие с аппаратурой и использование при таком взаимодействии абстракций с нулевыми издержками производительности. Стра́уструп описывает *C++* как «легковесный язык программирования с возможностью абстрагирования данных для создания и использования эффективных и изящных абстракций». Эффективное построение абстракций при сохранении преимуществ низкоуровневых языков программирования – то, что выделяет *C++* среди других языков.

Ниже перечислены основные особенности современной реализации языка *C++*.

* *Хранение данных и продолжительность существования объектов*. В *C++* возможны четыре варианта хранения данных:
  + *статическое*, при котором переменная определяется и инициализируется до исполнения программы (до входа в функцию *main*), существует на протяжении всего хода выполнения программы и удаляется после исполнения (когда программа выходит из функции *main*);
  + *поточное*, аналогичное статическому, однако вместо времени работы программы рассматривается время существования потока;
  + *автоматическое*, при котором переменные определяются как локальные для конкретного блока функции, создаются на *стэке* при входе программы в блок функции, существуют на протяжении исполнения этого блока и удаляются по завершении исполнения кода функции;
  + *динамическое*, при котором выделение памяти под объект и удаление объекта осуществляется программистом вручную с помощью операторов *new* и *delete* (аналоги *malloc* и *free* в *C*) или «умных» указателей, доступных начиная со стандарта *C++11*.
* *Шаблонное метапрограммирование*. *C++* позволяет создавать шаблоны функций, классов, переменных и псевдонимов типов. Шаблоны заполняются во время компиляции кода и могут быть параметризованы типами данных, известными на момент компиляции константами и другими шаблонными аргументами. С помощью шаблонов можно описывать структуры с разными известными на момент компиляции кода параметрами, не переписывая при этом одну и ту же структуру под каждое значение параметра.
* *Поддержка объектно-ориентированное подхода*. *C++* позволяет создавать *классы*, определять для них *атрибуты* и *методы*, создавать *объекты классов*. ООП в *C++* реализует *абстракцию* и *полиморфизм*, а также присущие только ООП концепции:
  + *инкапсуляция* позволяет ограничивать доступ к элементам структуры данных, что обеспечивает дополнительную безопасность при работе с объектом извне и делает интерфейс класса более очевидным;
  + *наследование* позволяет одному классу включать методы и атрибуты другого класса, при этом первый класс становится «дочерним», а второй – «родительским».
* *Полиморфизм*. Подразумевает возможность одного интерфейса иметь множество реализаций. Так, полиморфные функции обладают одним и тем же именем, но разными списками аргументов, и при вызове различаются передаваемыми параметрами. В объектно-ориентированном подходе полиморфизм проявляется при обращении к дочерним классам через родительский класс; в данном случае объект родительского класса полиморфен в том плане, что он может принимать форму как объекта своего класса, так и любого дочернего класса. Шаблоны также могут быть использованы для создания полиморфных структур.
* *Операторы и их перегрузка*. Язык *C++* поддерживает более 35 операторов, среди которых есть базовые арифметические, побитовые, косвенного обращения, сравнения, логические и другие. Практически все операторы могут быть перегружены для определённых пользователем типов данных.
* *Лямбда-выражения*, также известные как *анонимные функции*. Такие функции не привязаны к идентификатору и исполняются в том же месте, где они определены. Обычно используются в качестве аргументов функций высшего порядка.
* *Обработка исключений*. С помощью операторов *try* и *catch* можно предписывать поведение программы при возникновении ошибок времени исполнения. Кроме того, оператор *throw* позволяет генерировать исключения намеренно.

Стандартные реализации *C++* включают ряд стандартных библиотек для доступа к потокам ввода/вывода и файловым системам, инструментам работы со временем, генераторам случайных чисел, функциям для работы с памятью, а также библиотеки с более высокоуровневым функционалом (англ. *STL – Standard Template Library* – Стандартная Библиотека Шаблонов): векторы, списки, массивы, кортежи, алгоритмы над контейнерами и др.

Будучи кросс-платформенным мультипарадигменным языком программирования, позволяющим создавать высокопроизводительные программы, *C++*, тем не менее, обладает рядом недостатков:

* работа с указателями и низкоуровневый доступ к памяти, требующие особой осторожности и внимательности;
* отсутствие «сборщика мусора», вся ответственность за очистку памяти лежит на программисте;
* необходимость в ручной работе с памятью приводит к потенциально небезопасным программам, поскольку в случае ошибки программиста могут возникнуть такие осложнения, как утечка памяти, несанкционированный доступ к памяти, ошибка сегментации и др.;
* относительная сложность изучения по сравнению с большинством других языков программирования.

При разработке программного обеспечения планируется включение следующих сторонних библиотек *C++*:

* *Eigen* – шаблонная библиотека линейной алгебры;
* *pybind11 –* библиотека интеграции *C++* модулейв *Python*.

## Матричная библиотека *Eigen*

*Eigen* – шаблонная библиотека для осуществления операций линейной алгебры. В библиотеке описываются матричные и векторные типы данных, реализуемые как с помощью шаблонов, так и с помощью динамического выделения памяти. Будучи описанными в виде классов, матрицы и векторы поддерживают множество основных операций: умножение, сложение, транспонирование и т. д. В *Eigen* матрица определяется следующим образом:

*Matrix<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>*,

где в качестве параметров шаблона выступают тип данных и число строк и столбцов на момент компиляции соответственно.

Если размер матрицы или вектора известен в момент компиляции кода, *Eigen* предоставляет специальные типы данных с заданными по умолчанию параметрами шаблона, например:

* *typedef Matrix<float, 4, 4> Matrix4f;* – квадратная матрица 4 × 4 с компонентами типа *float*;
* *typedef Matrix<int, 3, 1> Vector3i;* – вектор из 3 компонент типа *int*.

Представленные типы определены в заголовочном файле. Конечно, если пользователю нужны матрицы или векторы иных размерностей, он может задать их вручную:

*Matrix<double, 3, 2> M;*

Для инициализации матрицы или вектора можно использовать оператор побитового сдвига влево, например:

*M << 1, 2, 3, 4, 5, 6;*

Существует и множество других способов инициализации матрицы или вектора, например случайными значениями, нулями, единицами и т. д.

Для случаев, когда размер матрицы или вектора на момент компиляции кода неизвестен, существует параметр шаблона *Dynamic*, обозначающий, что память для матрицы или вектора будет выделена динамически:

*typedef Matrix<int, Dynamic, Dynamic> MatrixXi;*

*MatrixXi M(3, 5); // матрица с динамически выделенной памятью*

Доступ к компонентам матрицы или вектора осуществляется с помощью перегруженного оператора вызова, аргументами которого выступают координаты компоненты:

*M(0, 0) = 1;*

*M(3, 2) = 4;*

*M(1, 1) = M(0, 0) + M(3, 2);*

С помощью методов *.col()* и *.row()* можно осуществлять доступ к отдельным столбцам или строкам матрицы.

Размер динамических матриц можно изменять с помощью метода *.resize()*. Однако важно отметить, что нет гарантии сохранения текущих значений компонент, то есть данный метод является деструктивным.

Матрицы и векторы можно складывать и вычитать. При этом операнды должны быть одинаковой размерности и скалярного типа данных:

*Matrix2i a = Matrix2i::Random(); // заполнение случайными значениями*

*Matrix2i b = Matrix2i::Random();*

*std::cout << a + b << std::endl;*

Перегруженный оператор умножения позволяет выполнение умножения матриц, однако число столбцов матрицы слева должно быть равно числу строк матрицы справа:

*Matrix2i c = a \* b;*

Поскольку векторы являются частным случаем матриц, умножение матрицы на вектор также допускается, однако число столбцов матрицы должно быть равно длине вектора. Векторы одинаковой длины также могут быть перемножены.

К сожалению, матричные и векторные типы не поддерживают многие поэлементные операции. Для этих целей *Eigen* предоставляет класс массивов общего назначения:

*Array<typename Scalar, int RowsAtCompileTime, int ColsAtCompileTime>*

Аналогично матрицам, для удобства имеются заранее объявленные типы массивов, например:

*typedef* *Array<float, Dynamic, 1> ArrayXf;*

*typedef Array<int, 3, 1> Array3i;*

*typedef Array<double, Dynamic, Dynamic> ArrayXXd;*

*typedef Array<double, 3, 3> Array33d;*

При объявлении массива пользователь имеет возможность задавать параметры шаблона напрямую:

*Array<int, 3, 2> A;*

Инициализация массивов осуществляется таким же образом, как и инициализация матриц/векторов.

Для массивов перегруженные операторы умножения, сложения, вычитания и т. п. выполняют соответствующие поэлементные операции. При этом необходимо, чтобы операнды были одинаковых размерностей.

*Array XXi a = ArrayXXi::Ones(3, 4); // инициализация массива размера*

*// 3 × 4 единицами*

*ArrayXXi b = ArrayXXi::Random(3, 4);*

*std::cout << a \* b << std::endl; // адамарово произведение матриц*

Кроме того, массивы, в отличие от матриц, поддерживают бинарные операции вида *массив + скаляр*:

*const int value = 5;*

*std::cout << a + value << std::endl;*

Массивы поддерживают множество других поэлементных операций:

* *.abs()* – вычисление модуля каждого компонента;
* *.sqrt()* – вычисление квадратного корня каждого компонента;
* *.min(.)* – построение массива, чьи компоненты есть минимальные из пар компонент двух переданных массивов;
* и т. п.

Для выполнения над матрицами операций, не поддерживаемых этим типом данных, необходимо конвертировать матрицу в массив. Соответственно, для выполнения над массивами операций, поддерживаемых только матрицами, необходимо конвертировать массив в матрицу.

* *Matrix<…>(…).array()* – конвертация матрицы в массив;
* *Array<…>(…).matrix()* – конвертация массива в матрицу.

Поскольку необходимость в вычислении адамарова произведения возникает достаточно часто, *Eigen* предоставляет для матриц метод *.cwiseProduct(.)*, позволяющий избежать конвертации в массив для выполнения поэлементного умножения.

Инструментарий библиотеки *Eigen* очень обширен. Для матриц существует также поддержка блочных операций, срезов, изменения формы матрицы (*reshape*) и множества других функций. Библиотека может использоваться при решении задач линейной алгебры и геометрии.

## Библиотека интеграции *C++* программ *Pybind11*

Поскольку взаимодействие с нейросетью планируется осуществлять через скрипты интерпретируемого языка *Python*, необходимо интегрировать собранный на *C++* модуль в *Python*. Существует несколько способов «обёртывания» написанного на *C++* кода.

* *Python/C API* – представляет собой программный интерфейс приложения, предоставляющий доступ к интерпретатору *Python* непосредственно через *C/C++* код. Используется в основном для написания *Python*-расширений на *C/C++*, а также для включения компонент языка *Python* в более крупные *C/C++* приложения. Основным недостатком является относительная сложность написания «обёртывающего» кода и невозможность интеграции *C++* классов в *Python* напрямую.
* *Cython* – язык программирования, упрощающий написание модулей *C/C++* для *Python*. Код *Cython* преобразуется в *C/C++* код для последующей компиляции и впоследствии может использоваться как расширение стандартного *Python* или как независимое приложение со встроенной библиотекой выполнения *Cython*. Язык обладает синтаксисом, очень близким к *Python*. Основной недостаток – невозможность использования кода независимо от *Python*. Для сравнения, *Python/C API* является лишь инструментом обёртки *C/C++* кода, который, в свою очередь, может быть использован независимо от *Python/C API*.
* *Boost.Python* – *С++* библиотека, обеспечивающая бесшовное взаимодействие между *C++* и *Python*. Позволяет быстро экспонировать *C++* классы, функции и объекты в *Python* и наоборот без использования специальных инструментов, достаточно наличие *C++* компилятора. Написание обёртывающего кода заметно проще, чем у стандартного *Python/C API*. К недостаткам можно отнести размер библиотеки.
* *Pybind11* – легковесная библиотека, состоящая только из заголовочных файлов. Позволяет экспонировать типы *C++* в *Python* и наоборот. Используется в основном для связывания *Python* с существующим *C++* кодом. Синтаксис схож с *Boost.Python*, при этом главное преимущество над последним – легковесность.

*Pybind11* представляет собой легковесную автономную библиотеку из одних заголовочных файлов. Не включая комментарии, заголовочные файлы насчитывают лишь около 4 000 линий кода. Библиотека также зависит от *Python* (2.7 или 3.5+) и стандартной библиотеки *C++*. Подобная компактность стала возможной благодаря расширенному функционалу *C++11*, например добавление кортежей, лямбда-функций и вариативных шаблонов. *Pybind11* позволяет экспонировать следующие *C++* конструкции в *Python*:

* функции, принимающие и возвращающие пользовательские структуры данных по значению, ссылке или адресу;
* методы экземпляра и статические методы;
* перегруженные функции;
* атрибуты экземпляра и статические атрибуты;
* произвольные типы исключений;
* перечисления;
* обратные вызовы;
* итераторы и диапазоны;
* пользовательские операторы;
* одиночное и множественное наследование;
* структуры данных библиотеки стандартных шаблонов (STL);
* умные указатели с подсчётом ссылок;
* классы с виртуальными методами, расширяемыми в *Python*.

*Pybind11* поддерживается множеством компиляторов, таких как *Clang/LLVM 3.3+*, *GCC 4.8+*, *Microsoft Visual Studio 2015 Update 3+*, *NVCC (CUDA 11.0)* и др.

Файлы библиотеки доступны для установки из каталога пакетов *Python* (*PyPl*)с помощью менеджера пакетов *pip*. Необходимые для обёртки кода макросы содержатся в заголовочном файле *pybind11/pybind11.h*. Традиционно также используется пространство имён *py*:

*#include <pybind11/pybind11.h>*

*namespace py = pybind11;*

Основным инструментом обёртки *C++* кода и построения *Python* модуля является макрос *PYBIND11\_MODULE*:

*PYBIND11\_MODULE(name, variable)*

Макрос создаёт входную точку, к которой обращается интерпретатор *Python* с помощью ключевого слова *import*. В качестве первого аргумента макроса выступает имя модуля без кавычек, второй параметр – переменная типа *py::module\_*, используемая для инициализации модуля. Методы *py::module\_*:

* *module\_::def()* – генерация связующего кода, экспонирующего функцию в *Python*;
* *module\_::attr()* – регистрация *C++* значения в модуле *Python*;
* *module\_::doc()* – доступ к строкам документации *Python* (*obj.\_\_doc\_\_*);
* и др.

Пример кода, экспонирующего простую функцию в *Python*:

*#include <iostream>*

*#include <pybind11/pybind11.h>*

*namespace py = pybind11;*

*void foo() {*

*std::cout << “Hello World!” << std::endl;*

*}*

*PYBIND11\_MODULE(MyModule, m) {*

*m.def(“SayHelloWorld”, &foo, “A function saying ‘Hello World!’”);*

*m.doc() = “This is a sample module”;*

*}*

Для обёртывания классов в библиотеке имеется шаблон *py::class\_<T>(variable, className)*. Шаблон создаёт привязки класса или структуры *C++*. В качестве параметра шаблона выступает имя *C++* класса, аргумент *variable* есть переменная типа *py::module\_*, *className* – строка имени класса, через которое к классу обращается интерпретатор *Python*. Метод *class\_::def()* описывает метод класса. Доступ к атрибутам можно реализовать с помощью метода *class\_::def\_readwrite()* или, если нужно открыть доступ только для чтения, с помощью *class\_::def\_readonly()*. Очевидно, экспонируемые методы и атрибуты должны быть публичными. Обязательно присутствие описания конструктора класса, определяемого с помощью передаваемой методу *def* шаблонной функции *py::init<init\_parameters>()*. Примечательно, что параметры инициализации объекта в данном случае передаются как параметры шаблона, а не самой функции.

Пример кода, экспонирующего простой класс в *Python*:

*#include <iostream>*

*#include <string>*

*#include <pybind11/pybind11.h>*

*namespace py = pybind11;*

*class Person {*

*private:*

*std::string name;*

*public:*

*unsigned age;*

*Person(const std::string& name) : name(name) {}*

*void getName() const { std::cout << name << std::endl; }*

*void setName(const std::string& name) { this->name = name; }*

*int getAge() const { return age; }*

*};*

*PYBIND11\_MODULE(MyModule, m) {*

*py::class\_<Person>(m, “Person”)*

*.def(py::init<const std::string&>())*

*.def\_readwrite(“age”, &Person::age)*

*.def(“getName”, &Person::getName)*

*.def(“setName”, &Person::setName)*

*.def(“getAge”, &Person::getAge);*

*}*

Синтаксически на стороне *Python* принцип работы с обёрнутыми функциями и классами *C++* не отличается от принципа работы с «родными» функциями и классами *Python*.

К сожалению, концепция *C++* модулей для *Python* не допускает передачу параметров *С++* шаблонов со стороны *Python*, поскольку заполнение шаблонов осуществляется в момент компиляции кода.

Для построения *Python* модуля можно использовать встроенный набор утилит для создания пакетов (*setuptools, distutils* и т. п.) путём написания соответствующих скриптов сбора пакета. При использовании интегрированной среды разработки *Microsoft Visual Studio* необходимость в написании дополнительных скриптов отсутствует.

Модуль в *Microsoft Visual Studio* может быть представлен в виде отдельного проекта со следующими параметрами:

* тип конфигурации – динамическая библиотека (*.dll*);
* расширение целевого файла – *.pyd*;
* отладка – отключена (установлено препроцессорное определение *NDEBUG*, библиотека среды выполнения – *Multi-threaded DLL*).

Также необходимо убедиться, что в дополнительных каталогах включения и библиотек указаны пути на соответствующие исходные файлы. Необходимые дополнительные каталоги включения доступны по команде в терминале *Windows*:

*py -m pybind11 --includes*

Дополнительный каталог библиотек должен содержать файл *python<версия>.lib*.

На выходе проекта генерируется *.pyd* файл, который в дальнейшем может быть использован в качестве импортируемого *Python* модуля.

## *Pybind11* и *Eigen*

В связи с популярностью библиотеки *Eigen* *pybind11* допускает непосредственную конвертацию между типами данных *Eigen* и матричными объектами *Python*. Для обеспечения поддержки *Eigen* при обёртывании кода необходимо включить заголовочный файл *pybind11/eigen.h*.

При передаче параметра по значению *pybind11* на стороне *Python* принимает аргумент типа *numpy.ndarray*, копирует данные и конвертирует в соответствующий тип *Eigen*. Со стороны программиста никаких дополнительных действий для обеспечения конвертации не требуется.

При передаче параметра по ссылке существует ряд ограничений, поскольку по умолчанию *numpy* и *Eigen* матрицы в отношении хранения данных в памяти не совместимы. Параметр *C++* функции, принимающей матрицу по ссылке, должен быть представлен с помощью шаблона *Ref<T>*, например:

*void foo (Ref<MatrixXi> m);*

Кроме того, необходимо, чтобы скалярные типы данных матриц на стороне *Python* и *C++* совпадали, то есть если матрица *numpy* состоит из компонент типа *float64*, то матрица *Eigen* должна быть скалярного типа *double*. Также модификация значений матрицы на стороне *C++* будет невозможна, если на стороне *Python* матрица имеет доступ только для чтения (то есть если атрибут *flags.writeable* имеет значение *False*). Матрицы должны быть организованы одинаковым образом: матрицы *Eigen* по умолчанию организованы в стиле языка *Fortran* (элементы распределены в постолбцовом порядке), поэтому матрицы *numpy* должны иметь такую же организацию компонент. Добиться этого можно с помощью функции *numpy.asfortranarray(array\_like)*.

Если *C++* функция возвращает ссылку на матрицу или любой другой аргумент, необходимо гарантировать сохранность данных и доступа к ним по завершении выполнения функции. Для подобных случаев *pybind11* позволяет трактовать *политику возврата значений*, передаваемую в качестве параметра метода *py::module\_::def()*. По умолчанию решение о том, как обращаться с возвращаемым значением, принимается автоматически, чему соответствует параметр *return\_value\_policy::automatic*. Иные политики возвращаемых значений:

* *return\_value\_policy::take\_ownership* – завладеть объектом без копирования;
* *return\_value\_policy::copy* – копировать объект;
* *return\_value\_policy::move* – переместить объект в памяти и завладеть им;
* *return\_value\_policy::reference* – сослаться на существующий объект, но не завладевать им;
* и др.

В случае, если доступ к возвращаемым данным каким-либо образом теряется, политику возвращаемых значений может потребоваться задать вручную. Например, если функция возвращает ссылку на матрицу, целесообразно обозначить политику *take\_ownership*, что не только сохранит доступ к данным, но и предотвратит их перезапись.

## Технология распараллеливания *OpenMP*

### Модель Исполнения

**OpenMP** (англ. *Open Multi-Processing*) – Программный Интерфейс Приложения (англ. *API – Application Programming Interface*) для реализации многопоточных вычислений на языках программирования *C/C++* и *Fortran*. Концепция стандарта OpenMP при распараллеливании вычислений реализует так называемую модель *fork-join* (разветвление и объединение). С помощью OpenMP API можно создавать программы, пригодные к выполнению как множеством, так и одним потоком. Тем не менее, стандарт допускает написание программ, требующих для выполнения не менее двух потоков, а также генерирующих разные результаты при выполнении одного и того же кода параллельно и последовательно. Кроме того, программа, задействующая разное число потоков, может давать разные численные результаты.

Программа, написанная с применением OpenMP, начинает своё выполнение последовательно с включением одного потока, именуемого первоначальным. Как только какой-либо из потоков (не обязательно первоначальный) доходит до распараллеленной области программы, поток создаёт *команду* потоков (англ. *team*), включающую порождающий поток и, возможно, один или несколько дополнительных потоков. Порождающий поток при этом становится потоком-*мастером* (англ. *master*). Для каждого из потоков в команде неявным образом генерируется *задание* (англ. *task*), определяемое заключённым в параллельную область кодом. Каждое задание привязывается к потоку, для которого оно было сгенерировано, то есть не подлежит исполнению другими потоками. В конце параллельной области устанавливается *неявный барьер* (англ. *implicit barrier*), препятствующий выходу из распараллеленного сегмента и выполнению последующего кода до тех пор, пока все потоки команды не закончат работу. Право на снятие барьера при этом имеет только поток-мастер. Количество распараллеленных областей для одной программы не ограничено.

OpenMP позволяет создавать *вложенные* параллельные области программы. Для этого необходимо, чтобы конкретная реализация OpenMP поддерживала вложенность параллельных регионов. Поддержка вложенности также должна быть включена вручную с помощью соответствующих инструментов OpenMP. Если поддержка вложенности включена, поток из команды внутри распараллеленной области при столкновении с вложенной распараллеленной областью порождает новую команду потоков, в которой порождающий поток становится потоком-мастером. Если поддержка вложенности выключена или отсутствует, для вложенного распараллеленного региона новой команды потоков не создаётся, и весь включённый в область код выполняется одним потоком.

OpenMP API включает средства синхронизации работы потоков и доступа к данным в распараллеленных областях. Также имеются библиотечные функции и переменные окружения, позволяющие контролировать среду исполнения программ, написанных с использованием стандарта OpenMP.

Спецификация OpenMP не гарантирует синхронность ввода/вывода данных одного файла, обрабатываемого распараллеленным кодом. В данном случае ответственность за синхронизацию ввода/вывода лежит на программисте. Для этого OpenMP предоставляет соответствующие инструменты синхронизации и библиотечные функции. Если каждый из потоков работает с отдельным файлом, необходимость в синхронизации со стороны программиста отсутствует.

### Модель Памяти

OpenMP API реализует *модель ослабленной согласованности* (англ. *relaxed-consistency*) с *общей памятью* (англ. *shared-memory*), все потоки имеют доступ к памяти. Кроме того, для каждого из потоков может быть доступно *временное представление* (англ. *temporary view*) памяти, при этом оно не является необходимым для модели памяти OpenMP. Временное представление памяти позволяет потокам кэшировать переменные общей памяти, что избавляет от необходимости читать общую память каждый раз, когда нужно получить доступ к данным переменной. Для каждого потока также доступна *приватная для потока память* (англ. *threadprivate memory*), к которой имеет доcтуп только привязанный поток.

Директива, для которой обозначен атрибут совместного использования данных, определяет два вида доступа к используемым в распараллеленном блоке кода переменным: *общий* (англ. *shared*) и *приватный* (англ. *private*). Переменная, обозначенная как общая или приватная, должна быть определена до входа в распараллеленную область. Каждая ссылка на общую переменную в распараллеленном регионе является ссылкой к первоначальной переменной, объявленной до распараллеленного блока кода. Для каждой приватной переменной создаётся новая версия того же типа данных и размера для каждого задания в распараллеленной области. При этом создание новой версии переменной гарантированно не изменяет значение изначальной переменной. Связь между изначальной переменной и её приватными версиями определяется конкретным атрибутами директивы.

Если несколько потоков одновременно записывают данные в один и тот же адрес памяти без синхронизации, возникает *конкуренция* (англ. *data race* или *race condition*). Аналогично, конкуренция имеет место, если по крайней мере один поток читает из участка памяти и по крайней мере один поток пишет в тот же участок памяти, при этом чтение и запись происходят одновременно и без синхронизации. В условиях конкуренции результат работы программы не определён.

Если несколько потоков пытаются получить доступ к небольшим фрагментам данных, размещённых на одной кэш-строке, возникает *ложное разделение* (англ. *false sharing*) данных. Поскольку процессор загружает данные в кэш блоками, называемыми также *строками* и обычно имеющими объём 64 байта, процессоры, работающие с данными в пределах одной кэш-строки, не могут получать доступ к данным одновременно, вследствие чего производительность программы существенно падает. Поэтому необходимо, чтобы предоставляемые кэшам процессоров данные не лежали в одной кэш-строке. Одно из возможных решений проблемы ложного разделения данных при работе с небольшими фрагментами данных – использование двумерных массивов, в которых длина одной строки приравнивается к длине кэш-строки. Необходимые данные при этом размещаются в начале каждой строки массива.

### Директивы

Для распараллеливания работы программы OpenMP API предоставляет набор препроцессорных директив. Директивы обрабатываются препроцессором до компиляции кода и позволяют изменять поведение программы. Директивы чувствительны к регистру, то есть одна и та же директива не может быть написана с разным регистром текста.

В языке *C/C++* директивы OpenMP указываются после препроцессорной директивы **pragma**:

**#pragma omp** *имя-директивы [опция[ [,] опция]…] новая-строка*

Директивы OpenMP применяются не более чем к одному нижестоящему выражению, которое может быть также оформлено в виде блока из нескольких выражений.

Ниже перечислены основные имена директив, доступных в OpenMP API.

* *Директивы распараллеливания работы программы*
  + **parallel** – распараллеливание кода в соответствии с моделью SPMD (англ. *Single Program Multiple Data* – Одна Программа и Множество Данных). Каждый поток команды выполняет один и тот же алгоритм, но при этом может работать с разными участками памяти.
  + **for** – распараллеливание итераций цикла между потоками, обычно помещается внутри распараллеленной области программы. Может быть также указана в качестве опции директивы **parallel**.
  + **sections** – директива, позволяющая распределять разные блоки кода между потоками. Код, принадлежащий директиве **sections**, разбивается на фрагменты, и каждый поток команды работает со своим фрагментом кода. Обычно директива помещается внутри распараллеленного блока программы. Может быть указана в качестве опции директивы **parallel**.
  + **section** – директива, определяющая фрагмент кода для выполнения одним потоком. Помещается внутри блока кода, принадлежащего директиве **sections**. Одной директиве **sections** могут принадлежать сразу несколько директив **section**. Таким образом, обозначенные данной директивой фрагменты программы распределяются между потоками команды, и каждый поток работает со своим блоком кода.
* *Директивы синхронизации*
  + **barrier** – установка барьера явным образом. Запрещает выполнение последующего кода до тех пор, пока все потоки в данной распараллеленной области не достигнут барьера.
  + **master** – команда, разрешающая выполнение кода только потоком-мастером, все остальные потоки указанный фрагмент программы игнорируют.
  + **single** – команда, разрешающая выполнение кода только одним потоком. В отличии от директивы **master**, данная директива указывает на то, что любой поток имеет право на исполнение фрагмента программы. Кроме того, на выходе из блока кода, принадлежащего директиве **single**, существует неявный барьер, в то время как для директивы **master** такой барьер отсутствует.
  + **critical** – директива синхронизации, запрещающая одновременное выполнение программы несколькими потоками команды. Код, принадлежащий директиве **critical**, выполняется потоками поочерёдно.
  + **atomic** – директива синхронизации для выполнения атомарных операций (инкремент, декремент и т. п.). Запрещает одновременное выполнение кода несколькими потоками команды. Код, принадлежащий директиве **atomic**, выполняется поочерёдно.
* *Прочие директивы*
  + **threadprivate** – директива, обозначающая ранее определённую переменную как приватную для каждого из потоков. Когда объявляется первая распараллеленная область, для каждого из потоков создаётся своя копия переменной, доступная только данному потоку. Кроме того, созданные приватные копии сохраняются между параллельными областями программы, и при входе в новый распараллеленный блок каждому потоку будет вновь доступна своя приватная копия, причём с теми значениями, которые были записаны в данной копии на выходе из предыдущей распараллеленной области программы. Только глобальные и статические переменные могут быть обозначены как **threadprivate**.

### Опции

Опции (англ. *clauses*) позволяют уточнять поведение программы в распараллеленных участках: задавать число потоков, определять общие и приватные переменные и т. п. Опции указываются сразу после имени директивы, порядок опций не имеет значения. Опции могут принимать один или несколько аргументов. Ниже представлен список основных опций директив OpenMP API.

* **num\_threads(n)** – опция, задающая **n** потоков для соответствующего распараллеленного фрагмента кода. При этом система не гарантирует предоставление запрошенного числа потоков. Если опция не указана, и если количество потоков не установлено в программе другими способами, для распараллеленной области создаётся число потоков, указанное в переменной окружения **OMP\_NUM\_THREADS**.
* **shared(var\_1 [, var\_2 [, … ] ])** – опция, определяющая общие для всех потоков переменные. Переменные должны быть определены до входа в распараллеленную область. Обычно все переменные, определённые до распараллеленного фрагмента программы, являются общими по умолчанию, поэтому часто необходимость в данной опции отсутствует.
* **private(var\_1[, var\_2[, … ] ])** – опция, определяющая приватные для каждого потока версии переменных. Переменные должны быть определены до входа в распараллеленную область. Приватные копии переменных не инициализируются.
* **firstprivate(var\_1[, var\_2[, … ] ])** – опция, определяющая приватные для каждого потока переменные. В отличии от **private**, данная опция инициализирует каждую копию значением, которое было у оригинальной переменной до входа в распараллеленную область.
* **lastprivate(var\_1[, var\_2[, … ] ])** – опция, определяющая приватные для каждого потока переменные. Кроме того, при выходе из распараллеленной области оригинальной переменной присваивается значение, которое было записано в приватную копию потока, прогонявшего последнюю итерацию цикла или последнюю секцию многосекционного блока. Обычно применяется с директивами **for** или **sections**.
* **default(shared | none)** – опция, определяющая поведение переменных в распараллеленной области по умолчанию. Если в качестве аргумента выбрано **none**, для всех определённых переменных поведение должно быть указано явным образом. Если выбран аргумент **shared** (обычно опция **default** с данным аргументом включена по умолчанию), то все определённые переменные по умолчанию становятся общими для всех потоков.
* **nowait** – опция, отключающая неявный барьер к конце принадлежащего директиве блока.
* **schedule(static | dynamic | guided[, n])** – опция директивы/опции **for**, позволяющая задать распределение итераций распараллеленного цикла между потоками. Статическое распределение (**static**) каждому потоку отводит одинаковое количество итераций, причем распределение блоков итераций между потоками циклично. Переменная **n** определяет размер блока итераций, отводимого одному потоку, и по умолчанию обычно равна числу итераций, делёному на число потоков. Динамическое распределение (**dynamic**) позволяет распределять итерации между потоками в зависимости от того, какой поток в конкретный момент времени свободен. При динамическом распределении создаётся очередь из итераций, и каждый последующий элемент в очереди достаётся первому освободившемуся потоку. Размер блока итераций **n** по умолчанию равен 1. Управляемое распределение (**guided**) аналогично динамическому, однако размер блока итераций не фиксирован и уменьшается по мере продвижения программы по циклу. Переменная **n** определяет минимальный размер блока итераций и по умолчанию приравнивается к 1.
* **reduction(operator: list)** – опция, обозначающая операцию сокращения в соответствии с оператором **operator** для указанных переменных списка **list**. Для каждого потока команды создаётся приватная копия переменной из списка **list** и инициализируется в зависимости от указанного оператора **operator**. Так, для оператора **+** копии получают значение 0, для оператора **\*** – 1. На выходе из распараллеленной области над приватными копиями выполняется операция сокращения в соответствии с указанным оператором **operator**, и результат операции записывается в оригинальные переменные списка. Например, для операции **+** все приватные копии складываются, для операции **\*** – умножаются.

### Библиотечный Функционал и Переменные Окружения

Библиотечные функции OpenMP API позволяют управлять поведением программы во время исполнения. Функции доступны из заголовочного файла **omp.h**. Ниже перечислены основные библиотечные функции OpenMP API.

* **void omp\_set\_num\_threads(int n)** – установить количество потоков для распараллеленных областей по умолчанию. Функция приравнивает переменную окружения **OMP\_NUM\_THREADS** к значению **n**, после чего при каждом вхождении в распараллеленную область порождается **n** потоков, если число потоков для конкретной распараллеленной области не указано явным образом (с помощью опции **num\_threads**), и если система может предоставить затребованное число потоков.
* **int omp\_get\_num\_threads(void)** – запросить количество потоков в текущей области программы. При вызове из распараллеленной области функция возвращает значение, равное реальному числу потоков в данной области. При вызове из последовательной области программы функция возвращает единицу.
* **int omp\_get\_max\_threads(void)** – запросить максимальное число потоков, доступное по умолчанию для создания команды потоков распараллеленной области программы. Возвращает текущее значение переменной окружения **OMP\_NUM\_THREADS**. При вхождении в распараллеленную область, обозначенную после вызова данной функции, максимальное число порождаемых потоков равно возвращаемому функцией значению, если число потоков не указано явным образом (с помощью опции **num\_threads**), либо если значение переменной **OMP\_NUM\_THREADS** не было изменено после вызова функции **omp\_get\_max\_threads** (например с помощью функции **omp\_set\_num\_threads**).
* **int omp\_get\_thread\_num(void)** – запросить порядковый номер потока. Возвращает *id* потока, из которого эта функция была вызвана. Значение *id* лежит в диапазоне [0, **omp\_get\_num\_threads()** – 1].
* **int omp\_in\_parallel(void)** – возвращает ненулевое значение, если вызвана из распараллеленной области программы. В противном случае возвращает 0.
* **void omp\_set\_dynamic(int dynamic)** – позволяет разрешать или запрещать системе динамически выбирать число потоков для распараллеленных областей программы. Приравнивает переменную окружения **OMP\_DYNAMIC** к значению аргумента **dynamic**. Ненулевое значение аргумента функции даёт разрешение на динамический выбор числа потоков, нулевое – запрещает.
* **int omp\_get\_dynamic(void)** – запросить значение переменной окружения **OMP\_DYNAMIC**. Если возвращаемое значение не равно нулю, система имеет разрешение на динамический выбор числа потоков для распараллеленных областей. Если возвращаемое значение равно нулю, у системы такого разрешения нет.
* **int omp\_get\_num\_procs(void)** – запросить количество процессоров, доступных для выполнения программы.

Помимо перечисленных, заголовочный файл **omp.h** также содержит функции для разрешения или запрета вложенного параллелизма, установки максимального числа уровней вложенного параллелизма, установки распределения итераций распараллеленного цикла во время исполнения, временны́е функции, а также соответствующие типы данных. Кроме того, библиотека содержит функции и типы, позволяющие реализовывать синхронизацию потоков с помощью *замко́в* (англ. *lock*).

Переменные окружения OpenMP описывают поведение программы, написанной с использованием OpenMP API. К основным переменным окружениям можно отнести:

* **OMP\_NUM\_THREADS** – максимальное количество потоков по умолчанию;
* **OMP\_DYNAMIC** – разрешает или запрещает системе динамически выбирать число потоков для каждой распараллеленной области;
* **OMP\_NESTED** – разрешает или запрещает вложенный параллелизм;
* **OMP\_PROC\_BIND** – контролирует привязку потока к процессору;
* **OMP\_THREAD\_LIMIT** – контролирует максимальное число доступных для программы потоков.

Среди переменных окружения OpenMP также имеются переменные контроля размера стека для каждого потока, описания политики ожидания для простаивающих потоков и контроля максимального числа уровней вложенного параллелизма.

## Система контроля версий *git*

Поскольку разрабатываемое программное обеспечение достаточно объёмно и состоит из множества компонент, необходимо обеспечить удобство контроля хода разработки и отслеживания изменений. Одним из возможных инструментов для решения этой задачи является система контроля версий *git*.

*Git* (произносится «гит») – программное обеспечение для отслеживания изменений в любом наборе файлов. Как правило, используется для координирования работы между разработчиками программного обеспечения. Основные цели *git* включают ускорение разработки, контроль целостности данных проекта и поддержку распределённых и нелинейных рабочих процессов. *Git* впервые появился в 2005 году под авторством финско-американского программиста Ли́нуса То́рвальдса для поддержки разработки ядра операционной системы *Linux*.

Наиболее традиционным пользовательским интерфейсом *git* является интерфейс командной строки. Взаимодействие с *репозиторием* осуществляется с помощью соответствующих команд. После *инициализации* репозитория пользователь может сохранять промежуточные результаты разработки с помощью *коммитов* (англ. *commit*) и при необходимости производить *откаты*. А команды сравнения коммитов позволяют наглядно отобразить изменения, привнесённые конкретным коммитом. Кроме того, *git* позволяет распараллелить разработку путём *ветвления*, при котором параллельные *ветви* не зависят друг от друга и содержат свой набор коммитов. Ветви впоследствии можно *объединять* в одну ветвь.

Помимо *локальных репозиториев*, *git* также позволяет работать с *удалёнными репозиториями*, например предоставляемыми веб-сервисом *GitHub*. С помощью *пуша* (англ. *push* – толкать) можно выгружать содержимое локального репозитория в удалённый, *пулл* (англ. *pull* – тянуть) позволяет загружать содержимое удалённого репозитория в локальный. Команда запроса на принятие изменений (англ. *pull request*) позволяет другим разработчикам ознакомиться с предлагаемыми изменениями, после чего предложенные изменения могут быть добавлены в проект или отклонены. Функционал *git* также включает отображение истории разработки, отслеживание авторства коммитов и т. п.

Современные интегрированные среды разработки могут включать функционал *git* и позволять взаимодействовать с локальными и удалёнными репозиториями с помощью графических интерфейсов. Подобная интеграция *git* может существенно упростить разработку приложений.

## Интегрированная среда разработки *Microsoft Visual Studio*

*Microsoft Visual Studio* – интегрированная среда разработки (англ. *IDE – Integrated Development Environment*), поддерживаемая и разрабатываемая корпорацией *Microsoft*. Используется для создания компьютерных программ, веб-сайтов, веб-приложений, веб-сервисов и мобильных приложений.

Редактор кода *Visual Studio* поддерживает технологию автодополнения кода *IntelliSense* и *рефакторинг* кода – изменение структуры кода без влияния на поведение программы. Интегрированный отладчик позволяет работать как с исходным кодом, так и с кодом машинного уровня. К прочим встроенным инструментам относятся средства *профилирования* кода (исследования характеристик программы), конструктор приложений с графическим интерфейсом, веб-конструктор, конструктор классов, конструктор схем баз данных и др. Функционал *Visual Studio* может быть расширен с помощью сторонних плагинов.

*Visual Studio* поддерживает более 30 различных языков программирования. Редактор кода и отладчик при наличии соответствующих расширений способны в той или иной степени поддерживать практически любой язык программирования. К встроенным языкам *Visual Studio* относятся *C*, *C++*, *Visual Basic*, *C#*, *F#*, *JavaScript*, *TypeScript*, *XML*, *HTML*, *CSS*. Для поддержки других языков, таких как *Python*, необходимо установить соответствующие расширения.

Базовая версия *Visual Studio*, известная как *Community Edition*, доступна бесплатно. *Visual Studio Community* распространяется под слоганом: «Бесплатная, полноценная IDE для студентов, разработчиков открытого исходного кода и самостоятельных разработчиков».

На 2022 год самой последней версией среды разработки является *Visual Studio 2022*, однако поддержка предыдущих версий (2013, 2015, 2017, 2019) продолжается. В рамках данной работы используется наиболее актуальная версия *Visual Studio* (рисунок 3.1).

Среда *Visual Studio* состоит из нескольких ключевых окон, к которым можно отнести *окно* *редактора* и *обозреватель решений*. В окне редактора проводится непосредственная работа с содержимым файла, обозреватель решений представляет собой браузер *решений* и *проектов*. В *Visual Studio* файлы объединяются в *проекты*, которые, в свою очередь, объединяются в *решения*. Каждый из проектов может быть настроен индивидуально, проекты также можно связывать между собой. Среда *Visual Studio* может быть настроена согласно предпочтениям пользователя: изменение содержимого панели инструментов, добавление в среду новых окон и их закрытие и т. д.

Одной из основных причин выбора в качестве среды разработки *Microsoft Visual Studio* послужило наличие поставляемого со средой компилятора *Microsoft Visual C++*. Встроенный компилятор избавляет от необходимости установки компилятора отдельно. Кроме того, проекты на *C++* в *Visual Studio* имеют широкий спектр настроек компилятора, которые при работе с компилятором через интерфейс командной строки приходилось бы прописывать вручную при каждой сборке программы.

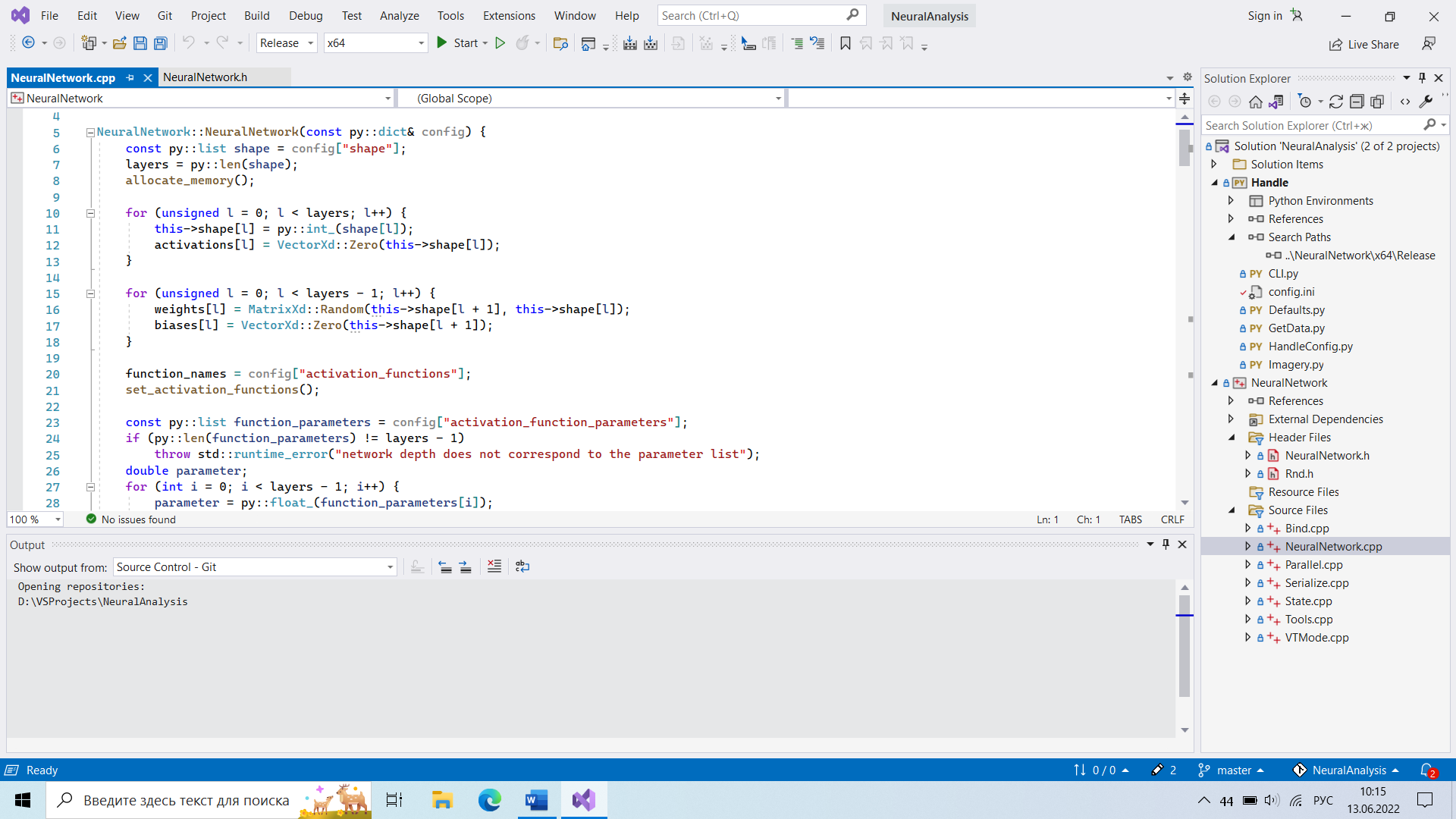


Рисунок 3.1. Среда Visual Studio 2022

Ещё одним преимуществом *Visual Studio* является интеграция системы контроля версий *git*. С помощью удобного графического интерфейса можно создавать новые репозитории и ветви, совершать коммиты, отслеживать историю разработки, взаимодействовать с удалёнными репозиториями и т. п.

# ОБЗОР РАЗРАБОТАННОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

## Руководство оператора

### Назначение программы

**Интерфейс Командной Строки для работы с Нейронной Сетью** (*NCLI*, англ. *Neural Network Command Line Interface*) – программное обеспечение, с помощью которого осуществляется взаимодействие с нейронной сетью и обработка изображений для формирования выборки для модели глубокого обучения. Архитектура программы представлена на рисунке 4.1.

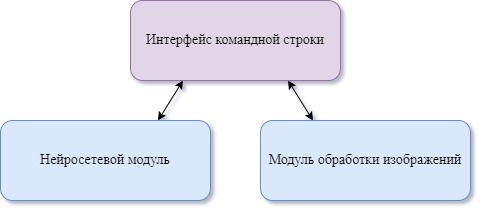


Рисунок 4.1. Архитектура программного обеспечения

Программа состоит из двух ключевых файлов:

* **ncli.exe** – исполняемый файл, запускаемый из командной строки;
* **config.ini** – файл конфигурации программы.

С помощью исполняемого файла осуществляется выполнение команд на инициализацию, обучение, тестирование нейросети, а также фрагментацию и зашумление изображений. Файл конфигурации позволяет задать параметры нейросети и обучения, параметры зашумления изображений, директорию выборки данных.

### Условия выполнения программы

Минимальные системные требования для выполнения программы:

* операционная система *Windows 7 x64*,
* объём оперативной памяти 512 Мб,
* процессор с тактовой частотой 2 ГГц.

Поскольку при обучении нейронной сети на центральный процессор возлагается большая нагрузка, скорость выполнения операций процессором напрямую влияет на скорость обучения нейронной сети. Для параллельного обучения модели желательно, чтобы процессор имел хотя бы два ядра. Кроме того, при увеличении размера нейронной сети может также и увеличиться объём необходимой оперативной памяти. Поэтому при инициализации новой модели глубокого обучения необходимо учитывать возможности компьютера.

### Выполнение программы

*Для запуска программы необходимо перейти в директорию расположения программы и открыть в ней командную строку, после чего в появившемся окне ввести имя исполняемого файла* ***ncli****.*

Работа в программе осуществляется с помощью набора команд и опций:

**ncli [ОПЦИИ] КОМАНДА [АРГУМЕНТЫ] [ОПЦИИ]**

Для отображения списка команд программы и для вывода справочной информации необходимо в командной строке ввести:

**ncli --help**

Для отображения списка опций и аргументов команды и вывода справочной информации по команде необходимо в командной строке ввести:

**ncli КОМАНДА --help**

Ниже представлен список команд программы и соответствующих опций.

* **init** – команда инициализации нейронной сети. Создаёт новую нейронную сеть в соответствии с указанными в файле конфигурации параметрами и сохраняет результат в бинарный файл с расширением *.bin*. Опции команды:
  + **-c, --config-file ДИРЕКТОРИЯ** – путь к файлу конфигурации, значение по умолчанию *config.ini*;
  + **-d, --dumpfile ДИРЕКТОРИЯ** – путь к бинарному файлу, в который сохраняется нейронная сеть, значение по умолчанию указывается в файле конфигурации.
* **inspect** – команда вывода конфигурации нейронной сети в консоль. Выводит такие данные, как форма (ширина и глубина сети), функции активации, коэффициенты функций активации, общее число прошедших эпох обучения. Опции команды:
  + **-d, --dumpfile ДИРЕКТОРИЯ** – путь к бинарному файлу, из которого считывается конфигурация нейронной сети, значение по умолчанию указывается в файле конфигурации.
* **feed** – команда подачи входных данных на нейросеть и вывода ответа в виде координат вертикалей. Аргументы и опции:
  + **ИМЯ\_ФАЙЛА** (обязательный аргумент) – имя файла, подаваемого на вход сети, файл должен быть изображением с соответствующим входному слою разрешением;
  + **-d, --dumpfile ДИРЕКТОРИЯ** – путь к бинарному файлу, из которого считывается конфигурация нейронной сети, значение по умолчанию указывается в файле конфигурации;
  + **-p, --path ДИРЕКТОРИЯ** – директория, в которой происходит поиск указанного файла изображения, значение по умолчанию берётся из файла конфигурации;
  + **-t, --threshold T** – пороговое значение активационной функции на выходном нейроне, при превышении которого ответ нейронной сети на конкретном нейроне считается положительным, значение **T** должно лежать в диапазоне (0, 1), значение по умолчанию указывается в файле конфигурации;
  + **-s, --show-image** – флаг, при передаче которого после вывода результата программа открывает поданное на вход изображение.
* **train** – команда запуска сеанса обучения нейронной сети. При обучении в консоль выводятся результаты промежуточных тестов, проводимых на выборке для обучения. По результатам каждого промежуточного теста в консоли отображается эпоха, в которую проводилось тестирование, полученное значение точности и наилучшее значение точности, полученное за все проведённые тесты в рамках текущего сеанса обучения. Опции команды:
  + **-p, --data-path** **ДИРЕКТОРИЯ** – путь к данным для обучения, значение по умолчанию берётся из файла конфигурации;
  + **-ch, --channel КАНАЛ** – один из диапазонов RGB, данные которого используются при обучении, имя диапазона также соответствует имени папки, расположенной в директории данных для обучения, значение по умолчанию берётся из файла конфигурации;
  + **-r, --reverse-dataset** – флаг инвертирования порядка расположения элементов в обучающей выборке;
  + **-s, --dataset-size** **S** – доля указанной выборки, используемая для обучения, значение должно лежать в диапазоне (0, 1] и применяется после инвертирования порядка расположения элементов в выборке при указании соответствующего флага, значение по умолчанию указывается в файле конфигурации;
  + **-c, --config-file** **ДИРЕКТОРИЯ** – путь к файлу конфигурации, значение по умолчанию *config.ini*;
  + **-d, --dumpfile ДИРЕКТОРИЯ** – путь к бинарному файлу, из которого считывается и в который сохраняется текущая конфигурация нейронной сети, значение по умолчанию указывается в файле конфигурации;
  + **--no-dump** – флаг, запрещающий сохранение конфигурации нейронной сети в бинарный файл по окончании сеанса обучения.
* **test** – команда запуска тестирования нейронной сети. По результатам тестирования в консоль выводится рассчитанная точность. Опции команды:
  + **-p, --data-path ДИРЕКТОРИЯ** – путь к данным для тестирования, значение по умолчанию берётся из файла конфигурации;
  + **-ch, --channel КАНАЛ** – один из диапазонов RGB, данные которого используются при тестировании, имя диапазона также соответствует имени папки, расположенной в директории данных для обучения, значение по умолчанию берётся из файла конфигурации;
  + **-nr, --no-reverse-dataset** – флаг запрета инвертирования порядка расположения элементов в тестовой выборке;
  + **-s, --dataset-size S** – доля указанной выборки, используемая для тестирования, значение должно лежать в диапазоне (0, 1] и применяется после инвертирования порядка расположения элементов в выборке, если не был указан флаг запрета на инвертирование, значение по умолчанию указывается в файле конфигурации;
  + **-d, --dumpfile** **ДИРЕКТОРИЯ** – путь к бинарному файлу, из которого считывается конфигурация нейронной сети, значение по умолчанию указывается в файле конфигурации;
* **crop** – команда фрагментации исходного изображения. Полученные фрагменты сохраняются в указанную папку, при отсутствии директории она создаётся. Если в выходной директории уже есть файлы, программа запросит подтверждение на очистку директории, после которой происходит фрагментация изображения. При ошибке права доступа на удаление файла в директории этот файл пропускается. Ход очистки и фрагментации выводится в консоль. Опции команды:
  + **-i, --input-path ДИРЕКТОРИЯ** – путь к исходному изображению, подлежащему фрагментации, значение по умолчанию берётся из файла конфигурации;
  + **-o, --output-path ДИРЕКТОРИЯ** – имя директории, в которую сохраняются полученные фрагменты, значение по умолчанию берётся из файла конфигурации;
  + **-s, --patch-size ШИРИНА\*ВЫСОТА** – размер фрагмента, если значение превышает разрешение исходного изображения, фрагменты не генерируются, значение по умолчанию берётся из файла конфигурации.
* **corrupt** – команда синтеза шума на изображениях. Зашумлённые изображения сохраняются в указанную папку, при отсутствии директории она создаётся. Если в выходной директории уже есть файлы, программа запросит подтверждение на очистку директории, после которой происходит зашумление изображений. При ошибке права доступа на удаление файла в директории этот файл пропускается. Ход очистки директории и синтеза шума выводится в консоль. Аргументы и опции команды:
  + **КАНАЛ** (обязательный аргумент) – один из каналов диапазона RGB, для которого синтезируется шум, представляет собой аппендикс к выходной директории в виде одноимённой папки;
  + **-i, --input-path** **ДИРЕКТОРИЯ** – путь к изображениям, подлежащим зашумлению, значение по умолчанию берётся из файла конфигурации;
  + **-o, --output-path** **ДИРЕКТОРИЯ** – имя директории, в которую сохраняются полученные зашумлённые изображения, значение по умолчанию берётся из файла конфигурации;
  + **-f, --csv-filename ИМЯ\_ФАЙЛА** – шаблон для имени файла с расширением *.csv*, в который сохраняются координаты полос, в конце имени файла приписывается имя канала, значение по умолчанию берётся из файла конфигурации;
  + **-c, --config-file ДИРЕКТОРИЯ** – путь к файлу конфигурации, значение по умолчанию *config.ini*.

Для установки конфигурации нейронной сети, параметров обучения, параметров фрагментации изображения и синтеза шума существует файл конфигурации с расширением *.ini*. По умолчанию файл носит имя *config.ini*. Файл конфигурации должен находиться в одной папке с исполнительным файлом **ncli.exe**.

Файл конфигурации состоит из тематических секций, каждая из которых содержит ряд параметров. Ниже представлены секции файла конфигурации и соответствующие параметры.

* **[meta]** – секция настройки поведения файла конфигурации. Содержит один параметр:
  + **split\_char** – сепаратор значений, принадлежащих одному параметру, заключается в кавычки.
* **[image\_resolution]** – секция настройки размера изображения, с которым работает программа. Определяет конфигурацию нейронной сети и размер фрагмента изображения по умолчанию. Содержит один одноимённый параметр:
  + **image\_resolution** – размер изображения в виде двух разделённых сепаратором **split\_char** значений, определяет размер фрагмента по умолчанию, произведение значений определяет ширину входного слоя нейронной сети, первое значение определяет ширину выходного слоя нейросети.
* **[network\_configuration]** – секция конфигурации нейронной сети. Определяет форму (глубину и ширину), активационные функции и коэффициенты для активационных функций. Содержит параметры:
  + **shape\_modifiers** – модификаторы формы нейронной сети, являются множителями количества нейронов на выходном слое нейросети и определяют ширину скрытых слоёв, значения отделяются сепаратором **split\_char**, количество параметров определяет глубину нейронной сети и не может быть меньше 1 (сеть должна иметь хотя бы один скрытый слой);
  + **activation\_functions** – активационные функции для всех слоёв нейронной сети (кроме входного), на данный момент доступны функции *sigmoid* и *ReLU*, значения отделяются сепаратором **split\_char**, количество значений должно соответствовать глубине нейронной сети;
  + **activation\_function\_parameters** – коэффициенты для активационных функций, значения отделяются сепаратором **split\_char**, количество значений должно соответствовать глубине нейронной сети.
* **[train\_parameters]** – секция конфигурации обучения нейронной сети. Содержит параметры:
  + **epochs** – количество эпох на один сеанс обучения;
  + **rate** – шаг обучения;
  + **test\_frequency** – число эпох, через которое проводится промежуточное тестирование и обновление сообщения в консоли.
* **[dynamic\_rate]** – секция конфигурации динамического шага обучения нейросети. Содержит параметры:
  + **dynamic\_rate** – флаг включения или отключения динамического шага обучения, может принимать значения **enable | true | disable | false**;
  + **rate\_delta** – значение, на которое делится текущий шаг обучения, должно быть отличным от нуля;
  + **accuracy\_stuck\_limit** – значение, определяющее количество эпох, по прошествии которых при условии отсутствии роста наилучшего значения точности шаг обучения **rate** сокращается в **rate\_delta** раз;
  + **accuracy\_stuck\_limit\_delta** – значение, на которое умножается параметр **accuracy\_stuck\_limit** после каждого сокращения шага обучения.
* **[parallel\_training]** – секция конфигурация распараллеленного обучения нейронной сети, реализующего концепцию параллелизма данных. Содержит параметры:
  + **parallel\_training** – флаг включения или отключения параллелизма обучения, может принимать значения **enable | true | disable | false**;
  + **threads** – количество потоков (копий), может принимать значения **n | cores | auto**, где **n** – натуральное число, **cores** – автоподбор числа потоков в соответствии с числом физических процессоров, **auto** – автоподбор числа потоков в соответствии с числом логических процессоров.
* **[corruption]** – секция конфигурации синтеза шума на изображениях. Содержит параметры:
  + **strip\_frequency** – интенсивность шума, значение должно лежать в диапазоне (0, 1);
  + **min\_shift** – минимальная величина модуля изменения значения яркости в столбце пикселей, при которой генерируется полоса, значение должно лежать в диапазоне [0, 255];
  + **shift\_direction** – допускает генерацию тёмных, светлых или любых полос, может принимать значения **darker | brighter | any**.
* **[default\_path]** – секция конфигурации директорий по умолчанию. Содержит параметры:
  + **relative** – флаг, определяющий интерпретацию указанных директорий как относительных или абсолютных;
  + **items\_path** – путь ко всем данным, с которыми работает программа;
  + **image\_file** – имя исходного изображения, располагающегося в директории **items\_path**;
  + **patches** – директория фрагментов исходного изображения, расположенная в директории **items\_path**;
  + **dataset\_path** – директория выборки нейросети, расположенная в директории **items\_path**;
  + **channel** – канал из диапазона RGB по умолчанию, представляет собой папку внутри директории **dataset\_path**;
  + **csv\_filename\_template** – шаблон имени *.csv* файла, к которому при создании файла *.csv* дописывается имя канала, файл располагается в директории **dataset\_path**.
* **[defaults]** – секция прочих настроек. Содержит параметры:
  + **dumpfile** – бинарный файл по умолчанию для сохранения и загрузки конфигурации нейронной сети;
  + **train\_dataset\_size** – доля набора данных для обучения и тестирования, причём в последнем случае коэффициент определяется как 1 – **train\_dataset\_size**, значение должно быть в диапазоне (0, 1];
  + **output\_threshold** – пороговое значение активационной функции на выходном нейроне, при превышении которого ответ нейронной сети на конкретном нейроне считается положительным, значение должно лежать в диапазоне (0, 1).

### Сообщения оператору

При работе с программой в консоль выводятся сообщения, информирующие пользователя о выполняемых или выполненных операциях. Ниже представлен список возможных сообщений:

* *Cropping…* – отображение прогресса при фрагментации изображения;
* *The image has been successfully cropped.* – сообщение, информирующее пользователя об успешном завершении фрагментации изображения;
* *Corrupting images…* – отображение прогресса при зашумлении изображений;
* *Comprising data…* – отображение прогресса при записи эталонных данных (координат сгенерированных полос) в файл;
* *The patches have been successfully corrupted.* – сообщение, информирующее пользователя об успешном завершении синтеза шума на изображениях;
* *The chosen directory already contains files. All the files in the directory will be deleted. Type in Y/y to proceed* – сообщение, информирующее пользователя о том, что в указанной директории для сохранения чистых или зашумлённых фрагментов изображения уже имеются файлы, и что для продолжения выполнения программы директорию необходимо очистить вводом символов *Y* или *y*;
* *Clearing…* – отображение прогресса при очистке директории;
* *Aborted!* – сообщение, информирующее пользователя о прерывании выполнения программы;
* *A new neural network has been initialised.* – сообщение, информирующее пользователя об успешной инициализации новой модели глубокого обучения;
* *Initialising...* – сообщение, информирующее пользователя об инициализации выполнения команды;
* *Training...* – отображение прогресса при обучении нейронной сети;
* *Testing...* – сообщение, информирующее пользователя о том, что программа выполняет тестирование модели.

Выполнение программы также предусматривает вызов исключений с выводом сообщения об ошибке в консоль. Выводимые сообщения достаточно конкретно информируют пользователя о возникшей проблеме. Ниже представлен список возможных ошибок:

* *FileNotFoundError: No such file or directory* – ошибка имени файла и/или директории, информирует пользователя о некорректности введённых данных касательно имени файла и/или его расположения;
* *FileNotFoundError: Системе не удаётся найти указанный путь* – ошибка имени директории, указывает на некорректность введённых данных касательно расположения файлов;
* *PIL.UnidentifiedImageError: cannot identify image file* – ошибка идентификации файла изображения, информирует пользователя о взаимодействии программы с некорректным файлом, возможной причиной возникновения ошибки может быть нераспознаваемый файл исходного изображения или наличие посторонних файлов в директориях с чистыми или зашумлёнными фрагментами;
* *ValueError: not enough values to unpack (expected 2, got 1) / too many values to unpack (expected 2)* – ошибка фрагментации изображения или инициализации модели глубокого обучения, указывающая на некорректность переданных данных о размере фрагмента изображения;
* *RuntimeError: network depth does not correspond to the function list* – ошибка инициализации модели глубокого обучения, указывающая на несоответствие количества значений параметра **activation\_functions** секции **[network\_configuration]** в файле конфигурации глубине нейронной сети;
* *RuntimeError: network depth does not correspond to the parameter list* – ошибка инициализации модели глубокого обучения, указывающая на несоответствие количества значений параметра **activation\_function\_parameters** секции **[network\_configuration]** в файле конфигурации глубине нейронной сети;
* *RuntimeError: invalid function name has been passed* – ошибка инициализации модели глубокого обучения, указывающая на некорректное значение параметра **activation\_functions** секции **[network\_configuration]** в файле конфигурации;
* *RuntimeError: failed to open a dump file* – ошибка загрузки или сохранения конфигурации нейронной сети, возникающая чаще всего при некорректном имени бинарного файла;
* *MemoryError: bad allocation* – ошибка инициализации модели глубокого обучения, наиболее вероятной причиной может быть слишком большой размер нейронной сети;
* *RuntimeError: vector size does not fit the network shape* – ошибка, указывающая на несоответствие размера подаваемого на вход модели вектора ширине входного слоя нейронной сети или размера эталонного вектора ширине выходного слоя нейросети, возможной причиной может быть несоответствие входного изображения или эталонного вектора форме нейронной сети;
* *IndexError: index is out of bounds for axis with size* – ошибка, указывающая на некорректность эталонных данных, возможной причиной может быть несоответствие файла с эталонными данными выборке зашумлённых изображений, проблему можно решить повторным синтезом шума на изображениях;
* *RuntimeError: rate\_delta must be a non-zero value* – ошибка, указывающая на приравнивание к нулю параметра **rate\_delta** секции **[dynamic\_rate]** файла конфигурации;
* *ValueError: could not convert string to float* – ошибка, указывающая на некорректное значение параметра файла конфигурации, ожидающего в качестве значения число;
* *ValueError: invalid literal for int() with base 10* – ошибка, указывающая на некорректное значение параметра файла конфигурации, ожидающего в качестве значения целое число.
* *Error: Got unexpected extra argument* – ошибка работы с интерфейсом командной строки, указывающая на ввод лишнего аргумента команды;
* *Error: Missing argument* – ошибка работы с интерфейсом командной строки, указывающая на отсутствие необходимого аргумента команды;
* *Error: No such option* – ошибка работы с интерфейсом командной строки, указывающая на ввод не предусмотренной программой опции;
* *Error: Invalid value for: is not a valid float.* – ошибка работы с интерфейсом командной строки, указывающая на некорректное значение опции, ожидающей число.

## Пример сеанса работы с программой

### Подготовка к работе

Прежде чем начать работу в программе, желательно поместить файлы **ncli.exe** и **config.ini** в отдельную папку. Далее в этой же директории следует создать новую папку и поместить в неё файл с исходным изображением. Имя созданной папки и имя помещённого в неё изображения должно быть указано в файле конфигурации в разделе **[default\_path]** для параметров **items\_path** и **image\_file** соответственно. Параметру **relative** в этой же секции желательно присвоить значение **true**, это позволит указывать относительные директории вместо абсолютных. На рисунке 4.2 представлено примерное содержание директории с исполнительным файлом, конфигурационным файлом и папкой с исходным изображением.

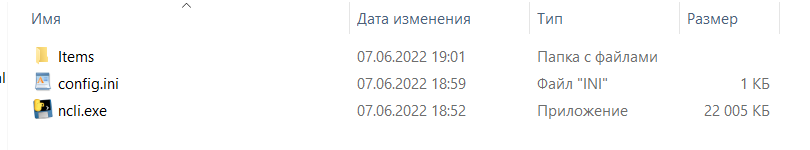


Рисунок 4.2. Примерное представление содержимого директории программы

### Фрагментация исходного изображения

Перед формированием выборки для нейронной сети, то есть перед синтезом шума на изображениях, необходимо разбить исходное изображение на фрагменты небольшого размера. В рамках данного примера размер фрагмента составляет 28 × 28, что позволяет сократить время обучения нейронной сети. Сначала в файле конфигурации в секции **[image\_resolution]** для одноимённого параметра нужно указать ширину и высоту изображения, значения разделить указанным в секции **[meta]** для параметра **split\_char** сепаратором, саму строку сепаратора при желании можно изменить. В данном примере параметру **image\_resolution** присваивается значение **28 28**. В секции **[default\_path]** для параметра **patches** указать директорию, в которую будут сохранены полученные фрагменты. Директория для фрагментов располагается внутри указанной для параметра **items\_path** директории, в случае отсутствия директории она создаётся.

Далее необходимо запустить программу. Прежде всего для этого необходимо открыть командную строку операционной системы *Windows* в директории программы. Для этого проще всего открыть директорию программы с помощью проводника *Windows*, в адресной строке ввести *cmd* и нажать *Enter*.

Чтобы начать фрагментацию изображения, нужно в открывшейся командной строке ввести следующую команду:

**ncli crop**

При этом с помощью опций можно также указать другой размер фрагмента или путь к изображению. В данном случае все значения остаются по умолчанию, то есть согласно указанным в файле конфигурации параметрам. Ход фрагментации изображения отображается в консоли (рисунок 4.3). По окончании фрагментации в консоль выводится соответствующее сообщение, а в указанную для фрагментов директорию сохраняются полученные изображения.

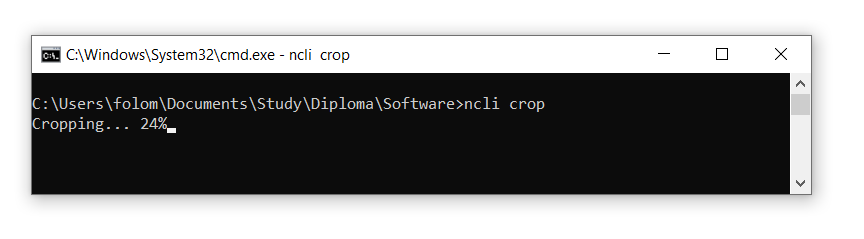


Рисунок 4.3. Мониторинг фрагментации изображения

### Синтез шума

Теперь, когда получены фрагменты исходного изображения, можно собирать выборку для нейронной сети. Прежде всего необходимо настроить конфигурацию зашумления изображений. Для данного примера параметры зашумления установлены в соответствии с рисунком 4.4.

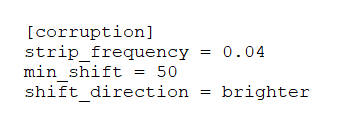


Рисунок 4.4. Параметры синтеза шума

В секции **[default\_path]** для параметра **dataset\_path** должна быть указана директория, в которую сохраняются зашумлённые изображения и файл с координатами зашумлённых полос. Для параметра **csv\_filename\_template** должен быть указан шаблон имени файла *.csv*, в который записываются координаты зашумлённых полос. Директория **dataset\_path** размещается внутри папки **items\_path**.

Для инициализации синтеза шума в канале R в командной строке необходимо ввести следующую команду:

**ncli corrupt R**

Ход синтеза шума отображается в консоли, как показано на рисунке 4.5. По завершении зашумления изображений в консоль выводится соответствующее сообщение, в указанной для зашумлённых изображений директории создаётся одноимённая каналу папка (в данном случае *R*) с зашумлёнными фрагментами и соответствующий каналу *.csv* файл с координатами зашумлённых полос.

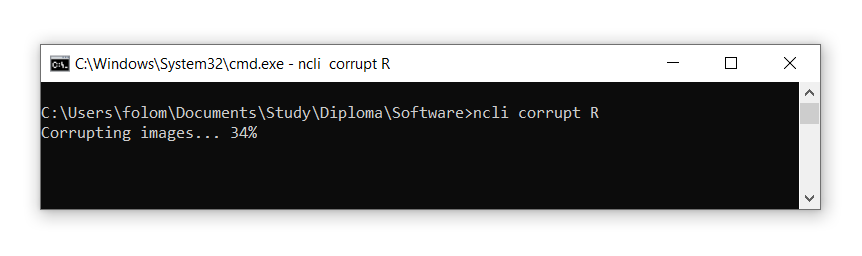


Рисунок 4.5. Мониторинг синтеза шума

### Инициализация нейронной сети

Теперь можно инициализировать новую нейронную сеть. Для начала в файле конфигурации необходимо задать конфигурацию нейросети. Размеры входных и выходных слоёв определяются исходя из размеров фрагмента изображения, указанных в секции **[image\_resolution]**. Для данного примера конфигурация сети установлена в соответствии с рисунком 4.6.

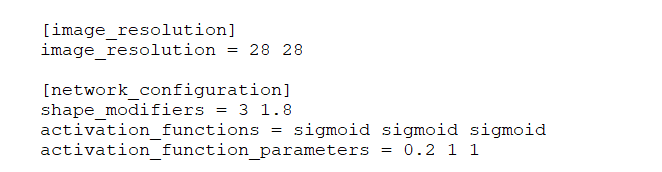


Рисунок 4.6. Конфигурация нейронной сети

Кроме того, в секции **[defaults]** для параметра **dumpfile** должно быть указано имя бинарного файла, в который сохраняется инициализированная конфигурация нейронной сети.

Для инициализации новой нейросети необходимо в командной строке ввести следующую команду:

**ncli init**

При успешной инициализации в консоль выводится соответствующее сообщение, а в директорию программы добавляется файл с расширением *.bin*.

### Вывод конфигурации нейронной сети в консоль

Чтобы убедиться, что нейронная сеть была инициализирована правильно, можно вывести текущую конфигурацию нейросети в консоль. Сеть загружается из бинарного файла, указанного в секции **[defaults]** для параметра **dumpfile**, если при вводе команды не указана соответствующая опция. Для вывода конфигурации модели в консоль необходимо ввести следующую команду:

**ncli inspect -d dump.bin**

В демонстрационных целях бинарный файл с конфигурацией сети был указан с помощью соответствующей опции. Результат работы команды представлен на рисунке 4.7.

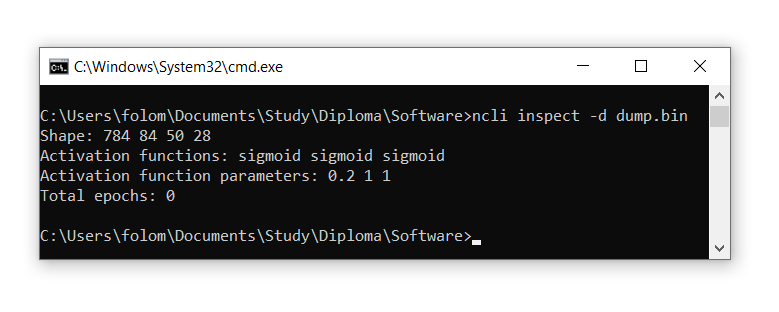


Рисунок 4.7. Вывод конфигурации нейронной сети в консоль

Как видно из рисунка 4.7, нейронная сеть была инициализирована правильно. Количество нейронов во входном слое соответствует числу пикселей на фрагменте изображения, число нейронов в выходном слое равно ширине фрагмента. Первый скрытый слой имеет 28 × 3 = 84 нейрона, второй – 28 × 1,8 ≈ 50 нейронов. Функции активации и их коэффициенты также соответствуют указанным в файле конфигурации на момент инициализации сети параметрам. Общее число эпох, равное 0, говорит о том, что нейронная сеть ещё не обучалась.

### Обучение нейронной сети

После инициализации нейронной сети можно начать её обучение. Стоит отметить, что обучение сети не ограничивается одним сеансом, и каждый новый сеанс можно проводить с совершенно иными параметрами обучения сети, например с уменьшенным шагом обучения, отключенным параллелизмом или другим числом потоков. Кроме того, разные сеансы можно проводить на разных выборках данных, существующие выборки данных можно пересобирать между сеансами обучения. Важно отметить, что в текущей версии программы *сеанс обучения нельзя прервать внутри командной строки*, а при закрытии командной строки весь прогресс, полученный в текущем сеансе, *будет потерян*. Поэтому не рекомендуется устанавливать слишком большое число эпох обучения для одного сеанса.

Перед началом обучения необходимо задать конфигурацию сеанса обучения. Параметры обучения описываются в файле конфигурации в секциях **[train\_parameters]**, **[dynamic\_rate]** и **[parallel\_training]**. Первый сеанс обучения в данном примере проводится с конфигурацией, представленной на рисунке 4.8.

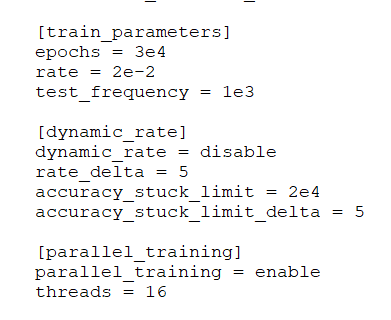
**

Рисунок 4.8. Конфигурация первого сеанса обучения

Также в секции **[default\_path]** для параметров **dataset\_path**, **channel** и **csv\_filename\_template** должны быть указаны путь к выборке данных, имя канала и шаблон имени *.csv* файла соответственно. В данном примере в качестве обучающей выборки выступают данные канала **R**. В секции **[defaults]** для параметра **train\_dataset\_size** должен быть указан коэффициент в диапазоне (0, 1), обозначающий используемую для обучения долю выборки данных, в данном примере значение этого параметра установлено равным **0,9**. Параметр **dumpfile** должен содержать имя бинарного файла с инициализированной конфигурацией нейронной сети, в этот же файл сохраняется обновлённая после обучения нейросеть, если при запуске сеанса обучения не был указан флаг запрета на сохранение конфигурации. Многие из перечисленных параметров могут также быть изменены с помощью опций команды обучения.

Для запуска сеанса обучения необходимо в командной строке ввести следующую команду:

**ncli train**

Через некоторое время после запуска команды в консоль выводятся результаты промежуточных тестов нейронной сети (рисунок 4.9). При правильно подобранных конфигурации нейронной сети и сеанса обучения, а также при правильно собранной выборке данных точность должна показывать стабильный рост. По завершении сеанса обучения в консоль также выводится время, затраченное на проведение сеанса, а обновлённая конфигурация нейросети записывается в исходный бинарный файл.

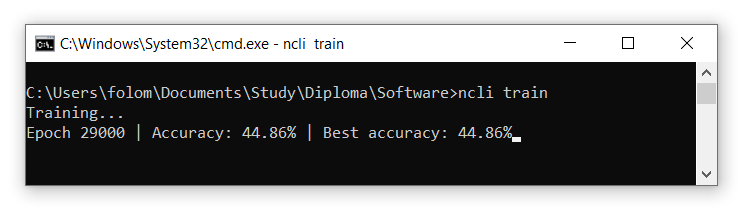


Рисунок 4.9. Мониторинг обучения нейронной сети

По мере обучения нейросети показатели точности могут перестать расти и начать колебаться или даже стремительно падать. Как правило, такое поведение встречается при обучении с большим шагом, так как в определённый момент конфигурация нейросети начинает «перепрыгивать» искомый экстремум. Распараллеливание обучения позволяет существенно уменьшить колебания точности при большом шаге обучения, однако это не значит, что при параллельном обучении для достижения желаемого результата можно не ограничивать темп обучения. Здесь для некоторого постоянного шага обучения и при прочих равных закономерность выглядит следующим образом: чем больше потоков задействовано в обучении, то есть чем больше копий участвует в одном сеансе обучения, тем меньше амплитуда колебаний точности и, как следствие, тем выше показатель точности по окончании сеанса обучения. Таким образом, шаг обучения следует выбирать исходя из задаваемого числа потоков, при этом не стоит забывать, что доступное количество потоков ограничено, а при числе потоков, большем числа физических процессоров, время работы программы существенно увеличивается.

В данном примере ближе к концу первого сеанса обучения точность стала демонстрировать проседания, что говорит о том, что шаг обучения необходимо уменьшить. Вообще, шаг обучения следует подбирать опытным путём. Для этого следует для сеанса обучения указать небольшое число эпох, а при инициализации обучения использовать флаг **--no-dump** для запрета сохранения результата обучения в бинарный файл. Таким образом можно подобрать шаг обучения, при котором рост точности модели будет наиболее быстрым и стабильным.

Для следующего сеанса обучения шаг **rate** установлен равным **10-2**, а число эпох **epochs** увеличено до **105**. Запуск следующего сеанса обучения осуществляется аналогичной командой на обучение. Результаты второго сеанса обучения представлены на рисунке 4.10. Можно заметить, что счёт эпох учитывает все предыдущие сеансы обучения.

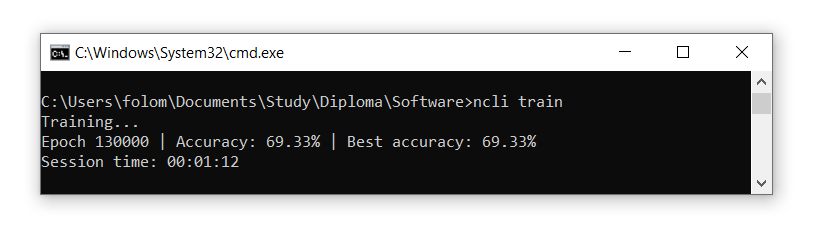


Рисунок 4.10. Результаты второго сеанса обучения

Поскольку второй сеанс обучения не демонстрировал существенных просадок в точности модели, для третьего сеанса параметры остаются прежними. К концу третьего сеанса обучения наилучшая точность составила *87,5%*. Для четвёртого сеанса параметры обучения также не менялись, и по результатам сеанса наилучший показатель точности составил *92,9%*. Пятый сеанс обучения при той же конфигурации увеличил точность модели до *94,66%*.

Для шестого сеанса обучения шаг **rate** был уменьшен до **5 × 10-3**. В секции **[dynamic\_rate]** файла конфигурации можно также задать динамическое изменение шага в зависимости от того, как долго в рамках сеанса обучения наилучшее значение точности не увеличивалось. В данном примере динамический шаг обучения не используется.

Одним из основных признаков того, что искомый экстремум при текущих конфигурации нейросети и выборке данных близко, является сильное замедление темпа роста точности модели. Таким образом, в данном примере точность наилучшая нейронной сети после обучения составила **95,34%** (рисунок 4.11).

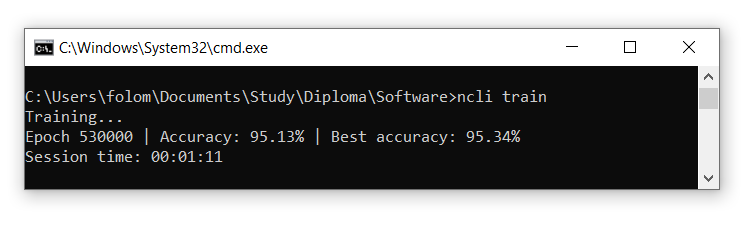


Рисунок 4.11. Результаты шестого сеанса обучения

### Тестирование нейронной сети

Во время обучения промежуточные тесты проводятся на обучающей выборке. Однако после обучения нейросеть должна быть способна работать с любыми данными, имеющими отношение к поставленной задаче. Поэтому имеет смысл тестировать нейросеть на данных, не вошедших в обучающую выборку.

Параметр **train\_dataset\_size** секции **[defaults]** файла конфигурации определяет, какая доля предоставленных данных отходит в обучающую выборку, а какая в тестовую. Например, при значении параметра 0,85 85% данных отводится на обучение, оставшиеся 15% – на тестирование.

В данном примере при значении параметра **train\_dataset\_size**, равном **0,9**, тестирование по умолчанию проводится на не попавшей в обучающую выборку доле данных, составляющей 10% от всей выборки. Для того, чтобы запустить тестирование на выделенной для него доле выборки, необходимо в командной строке ввести следующую команду:

**ncli test**

Результат тестирования выводится в консоль, как показано на рисунке 4.12. В данном случае точность на тестовой выборке составила **93,93%**.

Проводить тестирование можно на любой другой выборке и с любой другой долей выборки. В данном примере будет проведено тестирование на каналах G и B.

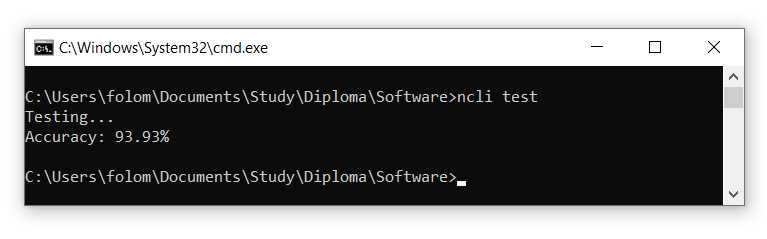


Рисунок 4.12. Вывод результатов тестирования на тестовой выборке

Сначала необходимо синтезировать шум для изображений в каналах G и B. Конфигурация синтеза шума для изображений зелёного канала остаётся такой же, какая была при зашумлении изображений в канале R (при сборке обучающей выборки). Для инициализации синтеза шума в канале G необходимо в командной строке ввести следующую команду:

**ncli corrupt G**

Конфигурация синтеза шума в канале B показана на рисунке 4.13. Для изображений в этом диапазоне увеличена интенсивность шума и уменьшено минимальное изменение значения яркости при генерации полосы. На изображениях этой выборки полосы встречаются чаще и могут быть менее чёткими.

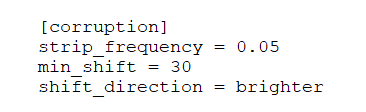


Рисунок 4.13. Конфигурация синтеза шума в канале B

Инициализация синтеза шума в канале B осуществляется с помощью команды:

**ncli corrupt B**

Теперь можно проводить тестирование на новых выборках данных. Для того, чтобы протестировать нейронную сеть на всей выборке канала G, необходимо в командной строке ввести следующую команду:

**ncli test -ch G -s 1**

Команда инициализирует тест на данных канала G, причём для тестирования используется вся выборка данных. По результатам тестирования точность модели при работе с этим диапазоном составила **94,51%**.

Команда инициализации тестирования на всей выборке канала B выглядит следующим образом:

**ncli test -ch B -s 1**

На рисунке 4.14 показан вывод результатов тестирования на канале B в консоль. Точность в этом диапазоне несколько ниже, чем в каналах R и G. Это связано с тем, что конфигурация синтеза шума для этого канала отличалась от параметров зашумления в других каналах. Тем не менее, по результатам теста показатель точности в диапазоне B составил **90,12%**.

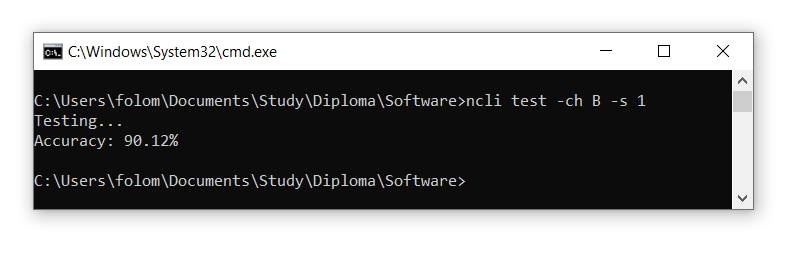


Рисунок 4.14. Результаты тестирования на данных канала B

### Получение ответа модели на входное изображение

В зависимости от поданной на вход матрицы изображения нейронная сеть отвечает активностью выходных нейронов. Уровень активности выходного нейрона, превышающий пороговое значение, говорит о том, что соответствующая нейрону вертикаль на изображении искажена.

Прежде чем подавать на вход нейросети изображение, нужно убедиться, что соответствующим параметрам в секции **[default\_path]** файла конфигурации присвоены директории с зашумлёнными изображениями. Для параметра **channel** в этой же секции необходимо указать имя канала, с которым планируется работать. Параметру **output\_threshold** секции **[defaults]** должно быть присвоено значение из диапазона (0, 1), соответствующее минимальному уровню активности нейрона, при котором полоса воспринимается как зашумлённая. В данном примере параметр **channel** имеет значение **R**, параметру **output\_threshold** присвоено значение **0,2**.

Матрица изображения подаётся на нейросеть с помощью имени соответствующего файла. Фрагменты изображения проименованы как *“i, j.bmp”*, где значения *i* и *j* соответствуют координатам фрагмента на исходном изображении. Так, при размере фрагмента 28 × 28 изображения проименованы как *“0, 0.bmp”, “0, 28.bmp”, “56, 84.bmp”* и т. д.

Для того, чтобы подать на вход нейросети изображение под именем *“28, 56.bmp”*, необходимо ввести следующую команду:

**ncli feed “28, 56.bmp”**

После обработки входных данных модель выдаёт ответ в консоль, как показано на рисунке 4.15. Изображение, поданное на входной слой нейросети, показано на рисунке 4.16. Как видно из представленных рисунков, модель в ответ на изображение выдала ожидаемый результат: полосы 4 и 23 (счёт начинается с 0) действительно искажены. Ответ можно также проверить в *.csv* файле соответствующего канала.

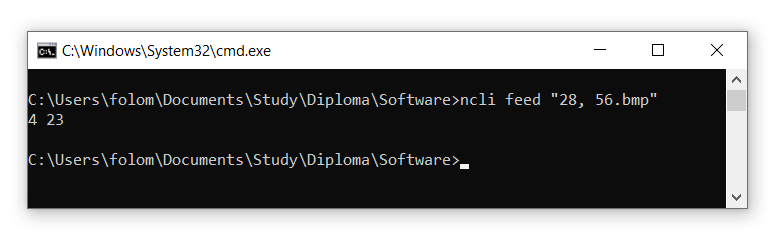


Рисунок 4.15. Вывод в консоль ответа на поданное на вход изображение



Рисунок 4.16. Поданное на вход изображение

### Сведения о системе

Описанный сеанс работы с интерфейсом командной строки для нейронной сети проводился на персональном компьютере со следующими техническими характеристиками:

* Операционная система *Windows 10 x64*,
* Процессор *Intel Core i5-10300H 2.50 GHz*,
* Объём оперативной памяти – 8 Гб.

Суммарное время обучения модели на ЭВМ с представленными характеристиками составило примерно 6 минут 17 секунд.

# НЕЙРОСЕТЕВОЙ АНАЛИЗ ИЗОБРАЖЕНИЙ

Одной из наиболее распространённых проблем спутниковой фотосъёмки является зашумление изображений. Ввиду различных факторов, таких как разность в температуре датчиков на спутнике, различия в конструкции датчиков и конструкции спутника в целом, изображения, полученные спутником, могут получаться неоднородными. Шумы на снимке могут затруднять извлечение, обработку и анализ спутниковых данных. Во многих случаях шумы на снимке могут проявляться в виде полос (рисунок 5.1).

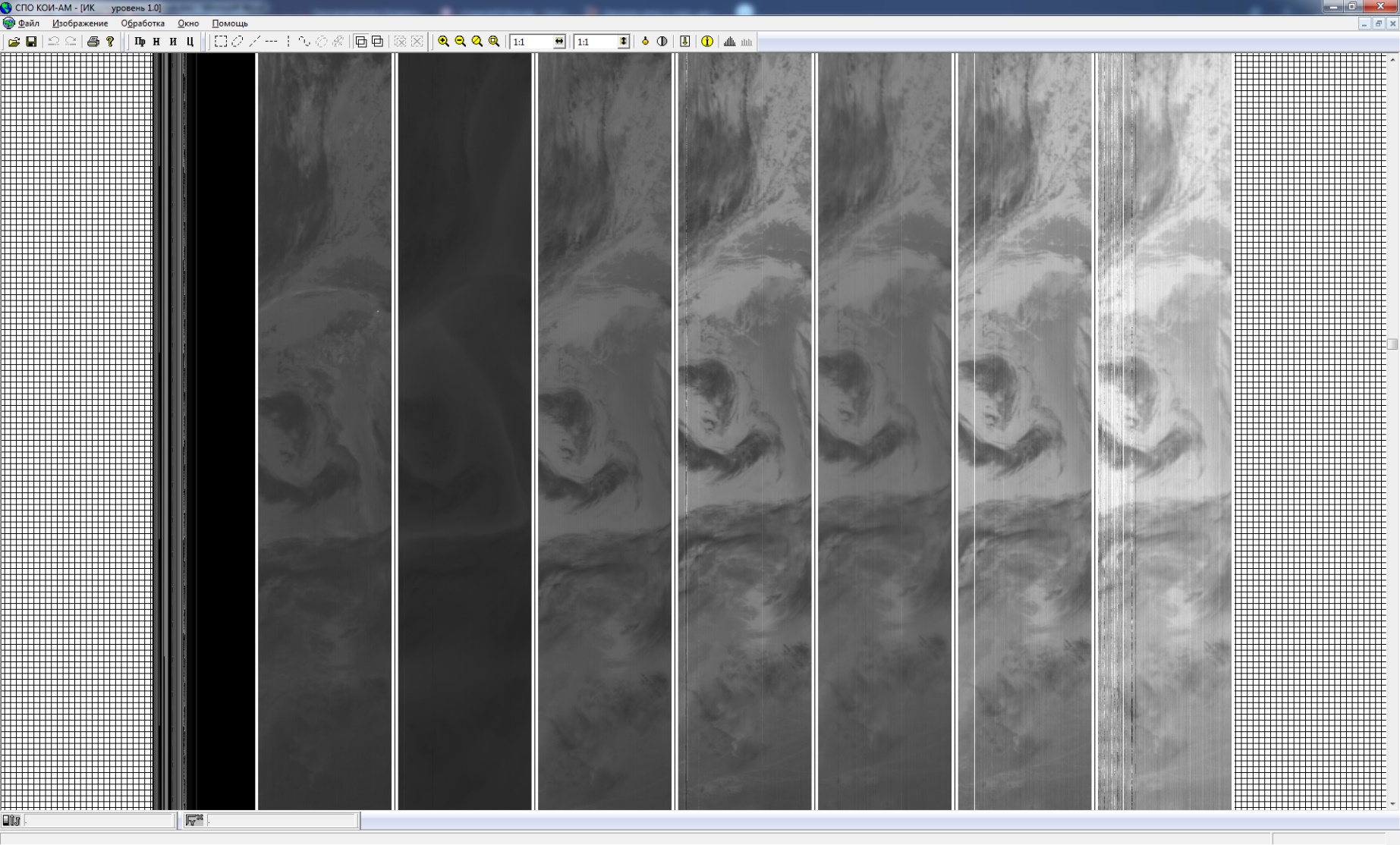


Рисунок 5.1. Спутниковые снимки со сбойными столбцами

Поскольку для обучения нейронной сети необходим большой объём данных, а также ввиду отсутствия доступа к большому количеству реальных зашумлённых спутниковых снимков, в рамках дипломного проекта нейронная сеть обучается на изображениях с синтезированным шумом, как показано на схеме (рисунок 5.2).

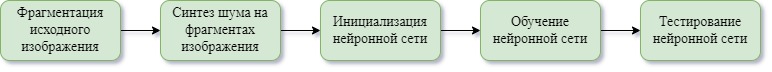


Рисунок 5.2. Алгоритм нейросетевого анализа изображений

## Фрагментация исходного изображения

В качестве исходного спутникового снимка был выбран фрагмент изображения поверхности Земли с разрешением 3841 × 3841 пикселей (рисунок 5.3). Изображение разбивается на небольшие фрагменты (рисунок 5.4), размеры которых определяются исходя из мощностных ресурсов ЭВМ, на которой выполняется обучение нейронной сети, а также использованных технологий разработки программного обеспечения. Каждый фрагмент проименован в соответствии с его координатами на исходном изображении. Фрагменты, как и целевое изображение, имеют расширение *.bmp*.

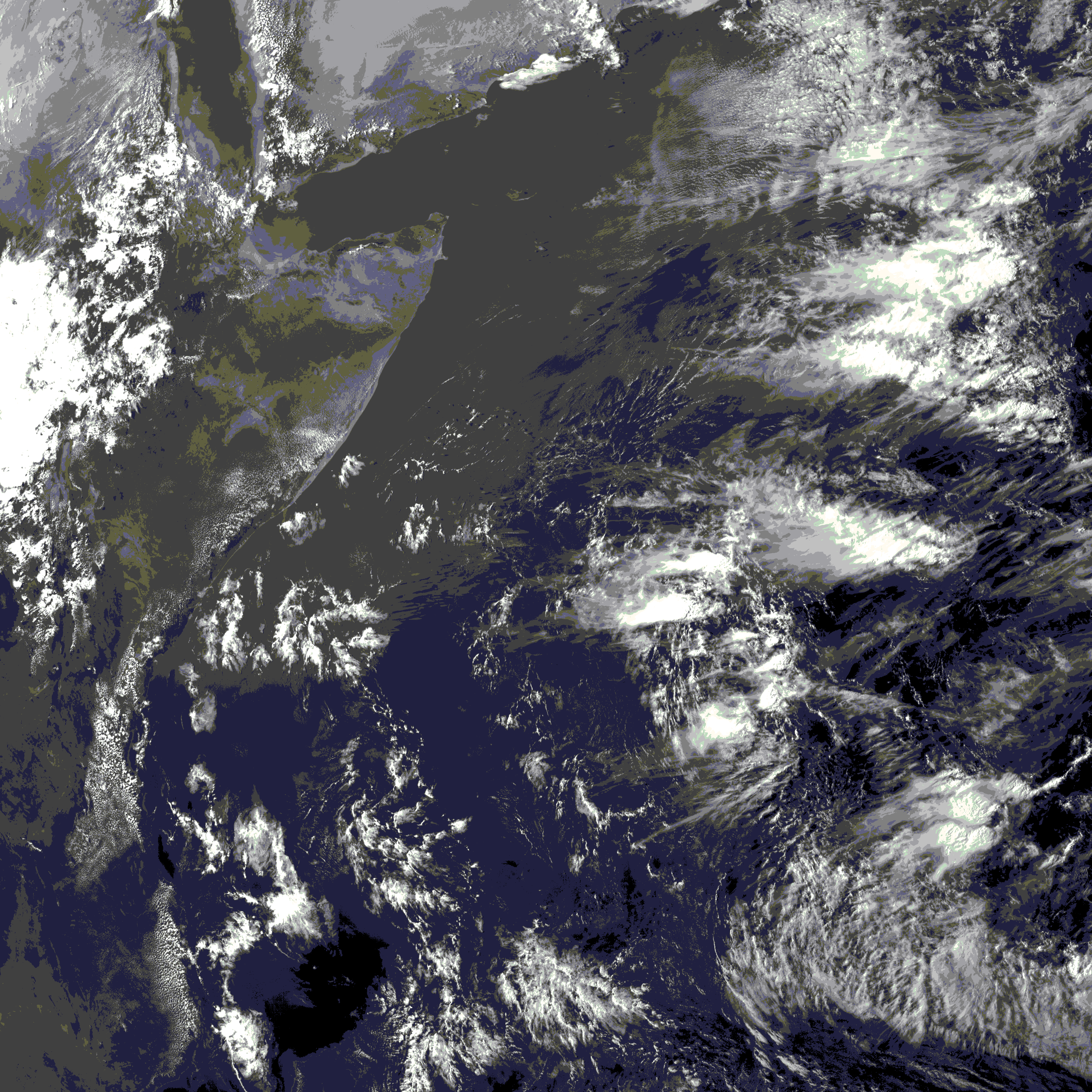


Рисунок 5.3. Исходное изображение – фрагмент спутникового снимка Земли

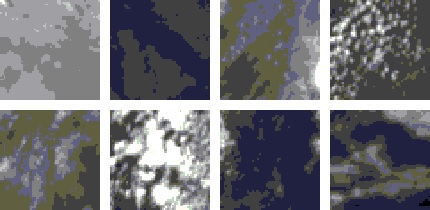


Рисунок 5.4. Фрагменты исходного изображения размером 56 × 56 пикселей

## Анализ исходного изображения и зашумление фрагментов

Каждый из фрагментов искусственным образом зашумляется: случайным образом на каждое изображение наносятся вертикальные полосы. При этом значение каждого пикселя конкретной полосы вычисляется в соответствии с линейным уравнением:

|  |  |
| --- | --- |
| , | (.) |

где *x* – текущее значение пикселя, *xnoise* – значение зашумлённого пикселя, *a* и *b* – параметры, вычисляемые случайным образом и константные для пикселей одной полосы.

Поскольку значение пикселя в каналах *R*, *G* и *B* с глубиной цвета 8 бит может варьироваться в диапазоне от 0 до 255, важно, чтобы значение *xnoise* не вышло за пределы этого диапазона. Следовательно, параметры *a* и *b* выбираются случайным образом в соответствии с условиями:

|  |  |
| --- | --- |
| , | (5.2) |

где *X* – множество значений пикселей в конкретном столбце.

Прежде чем собирать выборку для нейронной сети, следует провести анализ исходного изображения (рисунок 5.3). Важно отметить, что на спутниковом снимке преобладают тёмные цвета, в то время как светлые тона встречаются лишь на отдельных фрагментах изображения. На рисунке 5.5 представлена гистограмма, отображающая результаты анализа исходного изображения. Как видно из гистограммы, во всех трёх диапазонах RGB основная доля пикселей имеет значения меньше 128.

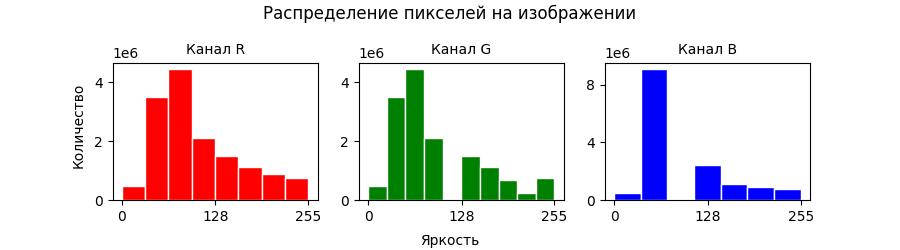


Рисунок 5.5. Гистограмма распределения пикселей на исходном спутниковом снимке

Поскольку на изображении преобладают тёмные цвета, при зашумлении фрагментов в соответствии с уравнением (5.1) и условиями (5.2) в полученной выборке данных будут преобладать фрагменты со светлыми полосами, тёмные же полосы будут встречаться значительно реже. Такое неравномерное распределение данных в выборке может привести к существенному ухудшению результата обучения нейронной сети. Поэтому при зашумлении изображений допускается генерация только светлых полос в соответствии с условиями:

|  |  |
| --- | --- |
| , | (.) |

Кроме того, чтобы избежать синтеза слишком тусклых полос, с трудом различимых даже человеческим глазом, и поднять потенциальную точность модели, при зашумлении изображений для обучающей и тестовой выборок было установлено ограничение на минимальное изменение яркости пикселя в сбойном столбце, равное 50. Таким образом, если в соответствии с выражениями (5.1) и (5.3) , где *X* – множество пикселей одной сбойной строки, то шум для данного столбца не синтезируется. Примеры зашумлённых фрагментов представлены на рисунке 5.6.

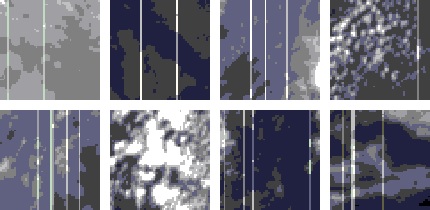


Рисунок 5.6. Результат синтеза шума на фрагментах изображения

В качестве эталонных данных при обучении нейронной сети выступают координаты сгенерированных полос. При зашумлении изображений порядковые номера изменённых столбцов пикселей, начиная от 0, записываются в файл с расширением *.csv* (рисунок 5.7).

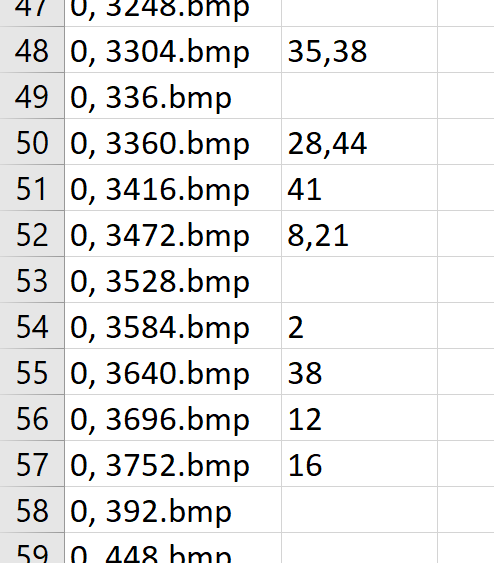


Рисунок 5.7. Фрагмент возможного содержимого файла .csv

## Архитектура нейронной сети

Поскольку на вход нейронной сети поступает изображение, входной слой нейронной сети был настроен таким образом, чтобы каждый входной нейрон соответствовал конкретному пикселю на изображении. Число выходных нейронов сети соответствует числу вертикальных рядов пикселей на изображении. Что касается скрытых слоёв нейронной сети, не существует какой-либо единой концепции, позволяющей чётко установить необходимое количество скрытых слоёв, называемое *глубиной* нейронной сети, а также количество нейронов на каждый скрытый слой. Однако существует корреляция между глубиной сети, числом нейронов на скрытый слой и результативностью нейронной сети: чем больше скрытых слоёв и чем больше нейронов на каждый скрытый слой, тем выше потенциал нейронной сети, т.е. тем более сложные задачи сеть способна решать. Однако вместе с тем слишком большие значения этих двух параметров могут привести к необучаемости сети. Выбор числа скрытых слоёв и нейронов также должен исходить из мощностных ресурсов ЭВМ и средств реализации нейронной сети. Таким образом, экспериментальным путём количество скрытых слоёв выбрано равным 2, при этом число нейронов в первом скрытом слое в три раза больше числа нейронов в выходном слое, а во втором – в 1,8 раз. Структура нейронной сети представлена на рисунке 5.8.

В качестве активационной функции во всех, кроме входного, слоях нейронной сети используется сигмоида; на входном слое какая-либо активационная функция отсутствует, так как данный слой служит лишь инструментом сбора данных. Поскольку данные на входном нейроне варьируются в диапазоне от 0 до 255, для слоя *L1* значение крутизны сигмоиды было установлено равным 0,02, что теоретически обеспечивает надлежащую чувствительность к изменению входных параметров. Сигмоида для слоёв *L2* и *L3* имеет коэффициент крутизны равный 1.

На входе нейронной сети матрица изображения размера *WIDTH* × *HEIGHT* преобразуется в одномерный массив. На выходе нейронной сети каждый из *WIDTH* нейронов соответствует вертикальному ряду пикселей на изображении. «Активность» выходного нейрона, то есть значение активационной функции на этом нейроне, свидетельствует о зашумлённости соответствующего ряда пикселей на входном изображении. Предполагается, что вывод о зашумлённости той или иной полосы делается на основе факта превышения значения активационной функции выходного нейрона некоторого порогового значения.

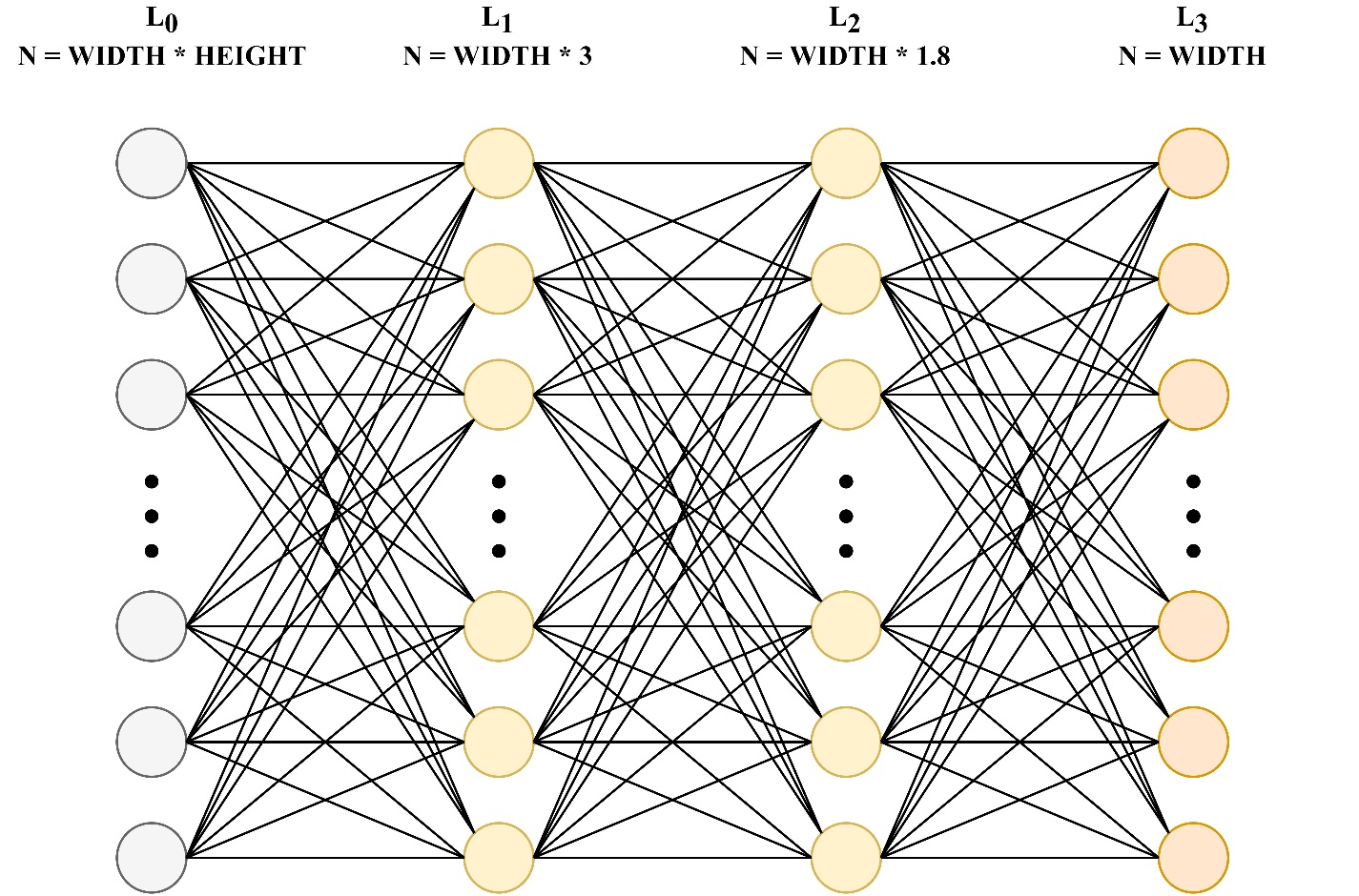


Рисунок 5.8. Архитектура нейронной сети

## Результаты обучения и тестирования моделей

В качестве обучающей выборки было использовано около 90% зашумлённых фрагментов канала *R*. Каналы *G* и *B* в обучающую выборку не вошли. Эталонными данными выступили координаты зашумлённых столбцов (рисунок 5.7). Начальный показатель скорости обучения *η* устанавливался равным *2 × 10-2* и уменьшался от сеанса к сеансу в зависимости от темпа роста точности предсказаний модели.

Ход обучения оценивался по показателям точности совершаемых сетью предсказаний. Точность предсказаний оценивалась в соответствии с выражением:

|  |  |
| --- | --- |
| , | (.) |

где *ci* – количество найденных сетью реальных полос на *i*-м изображении, *pi* – общее число полос на *i*-м изображении, *N* – число изображений в обучающей выборке. Стоит отметить, что такой подход оценки точности не принимает во внимание ложные предсказания, однако в целом его достаточно для общей оценки хода обучения и текущего состояния нейронной сети. Кроме того, ввиду малого числа полос по отношению к ширине изображения (обычно не больше 5 на один фрагмент), как показали дальнейшие эксперименты, сеть оказалась несклонной к ложным срабатываниям. Оценка точности происходила через каждую 1000 эпох обучения.

Для решения поставленной задачи было инициализировано сразу несколько моделей нейронной сети, каждая соответствовала заданному разрешению фрагмента изображения. Данные для обучающей и тестовой выборки взяты из канала *R* в следующей пропорции: 90% фрагментов канала *R* берутся для обучения нейронной сети, оставшиеся 10% – для тестирования.

При синтезе шума на изображениях обучающей и тестовой выборки был установлен параметр минимально допустимой видимости полосы, равный 50. Это значит, что значение яркости пикселей при зашумлении должно быть изменено не меньше, чем на 50 единиц. Это позволяет предотвратить синтез слабозаметных и неразличимых полос и повысить точность модели. Результаты обучения моделей представлены в таблице 5.1.

После обучения модели тестируются на данных канала *G* и *B*. При синтезе шума для изображений канала *G* параметр минимально допустимой видимости полосы остаётся прежним. Каждая из моделей тестируется дважды, каждое тестирование проводится на выборках изображений с разным значением параметра интенсивности полос. Результаты тестирования на канале *G* представлены в таблице 5.2.

При синтезе шума для изображений канала *B* интенсивность полос постоянна и составляет 0,03. Каждая из моделей тестируется два раза, каждый тест модели проводится на выборках зашумлённых изображений с разным значением минимальной видимости полосы. Результаты тестирования на канале *B* представлены в таблице 5.3.

Таблица 5.1. Результаты обучения и тестирования моделей

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер фрагмента | Число эпох обучения | Интенсивность полос | Точность на обучающей выборке, % | Точность на тестовой выборке, % |
| 28 × 28 | 5,3 × 105 | 0,04 | 95,34 | 93,93 |
| 36 × 36 | 7 × 105 | 93,53 | 89,57 |
| 42 × 42 | 8 × 105 | 0,03 | 96,26 | 91,33 |
| 48 × 48 | 106 | 96,72 | 93,62 |
| 56 × 56 | 1,2 × 106 | 94,63 | 89,35 |

Таблица 5.2. Результаты тестирования моделей на данных канала G

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер фрагмента | Интенсивность полос | Точность, % | Интенсивность полос | Точность, % |
| 28 × 28 | 0,07 | 90,21 | 0,02 | 97,4 |
| 36 × 36 | 85,45 | 95,56 |
| 42 × 42 | 0,05 | 87,85 | 0,015 | 95,83 |
| 48 × 48 | 87,95 | 95,93 |
| 56 × 56 | 82,83 | 92,56 |

Как видно из результатов обучения и тестирования, с ростом размера фрагмента точность модели падает. Это можно объяснить в первую очередь тем, что при увеличении размера изображения растёт и поле поиска полосы, что усложняет задачу. Кроме того, чем больше размер фрагмента, тем больше полос может быть сгенерировано на одном изображении, по этой же причине для обучающей и тестовой выборок с фрагментами большего размера интенсивность полос была уменьшена. С ростом размера изображения падает и количество фрагментов для обучения сети: при размерах одного фрагмента 28 × 28 пикселей число элементов в выборке составляет 18 769, а для изображений с разрешением 56 × 56 пикселей – 4 624, что более чем в 4 раза меньше.

Таблица 5.3. Результаты тестирования моделей на данных канала B

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер фрагмента | Минимальная видимость полосы | Точность, % | Минимальная видимость полосы | Точность, % |
| 28 × 28 | 20 | 92,39 | 80 | 96,95 |
| 36 × 36 | 86,54 | 94,7 |
| 42 × 42 | 85,38 | 95,12 |
| 48 × 48 | 84,35 | 95,02 |
| 56 × 56 | 78,09 | 91,52 |

Таким образом, после обучения моделей средняя точность предсказаний координат зашумлённых полос в рамках проведённых тестов составила примерно **90,8%**, максимальная точность предсказаний достигала **97,4%**. При этом нейронные сети после обучения на данных одного канала видимого диапазона способны без потери точности работать с данными других каналов видимого диапазона. Конечно, показатели точности сильно зависят от выборки данных, на которой проводится тестирование. Поэтому перед обучением нейронной сети необходимо чётко определиться, с какими данными модель должна быть способна работать по завершении обучения, и подобрать обучающую выборку в соответствии с поставленной задачей. В данном случае выборки для тестирования формировались таким образом, чтобы продемонстрировать вариативность поведения моделей в зависимости от предоставленных данных, но при этом не отойти от поставленной при обучении нейросети задачи слишком далеко.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были поставлены и выполнены следующие задачи:

* изучить существующие методы распаковки спутниковых данных и анализа их целостности;
* разработать программное обеспечение для анализа сбоев на спутниковых снимках с помощью модели глубокого обучения;
* провести обучение нейронной сети на искусственно зашумленных изображениях;
* провести анализ целостности спутниковых изображений путём распознавания сбойных столбцов на зашумлённых фрагментах с помощью построенной нейронной сети.

Разработанное программное обеспечение было многократно протестировано на персональных компьютерах с различными характеристиками и продемонстрировало надёжность и устойчивость к сбоям и ошибкам, не предусмотренным ожидаемыми сценариями использования программы.

Таким образом, можно считать, что все поставленные в рамках выпускной квалификационной работы задачи были выполнены успешно.

Возможные направления развития концепций, разработанных в ходе выполнения выпускной квалификационной работы, включают:

* разработка архитектуры модели глубокого обучения для восстановления зашумлённых изображений;
* реализация иных методов распараллеливания обучения нейронной сети;
* внедрение графических ускорителей и соответствующих программных средств в разработанную программную реализацию модели нейронной сети.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гонсалес Р., Вудс Р. Цифровая обработка изображений. Москва, Техно-сфера, 2005, 1072 с.
2. Chavez P.S. Jr. Image-based atmospheric corrections – revisited and improved // Photogrammetric Engineering & Remote Sensing. Vol. 62 (9). 1996.
3. Gao Y.N., Zhang W.C. Simplification and modification of a physical topographic correction algorithm for remotely sensed data // Acta Geodaet. Cartogr. Sin. Vol. 37. 2008.
4. Huang W., Zhang L.P., Li. P.X. An improved topographic correction approach for satellite image // J. ImageGraph. Vol 10. 2005.
5. Лупян Е.А., Прошин А.А., Бурцев М.А., Балашов И.В., Барталев С.А., Ефремов. А.В., Кашницкий В.Ю., Мазуров А.А., Матвеев А.М., Суднева О.А., Сычугов И.Г., Толпин В.А., Уваров И.А. Центр коллективного пользования системами архивации, обработки и анализа спутниковых данных ИКИ РАН для решения задач изучения и мониторинга окружающей среды // Современные проблемы дистанционного зондирования Земли из космоса. 2015. Т. 12. No 5.
6. Андреев Р. В., Гектин Ю. М., Зайцев А. А. и др. Практическая реализация методов радиометрической коррекции ИК-изображений, получаемых с использованием многоэлементных фотоприемников // Ракетно-космическое приборостроение и информационные системы. 2015, т. 2, вып. 4.
7. Майорова В.И., Банников А.М., Зайцев К.И. Математическое моделирование процесса радиометрической коррекции снимков ДЗЗ. Инженерный журнал: наука и инновации, 2013, вып. 3. URL:http://engjournal.ru/catalog/mathmodel/hidden/641.html
8. Злобин В.К., Еремеев В.В. Обработка аэрокосмических изображений. Москва, ФИЗМАЛИТ, 2006, 288с.
9. Интеллектуальные системы и нечеткая логика: Учебник / В.П. Корячко, М.А. Бакулева, В.И. Орешков – М.: КУРС, 2017 – 352 с.
10. Барский А.Б. Нейронные сети: распознавание, управление, принятие решений. – М.: Финансы и статистика, 2004.– 176 с.
11. Калан Роберт. Основные концепции нейронных сетей: Пер. с англ. – М.: Издательский дом «Вильямс», 2001. – 287 с.
12. Michael Nielsen. Neural Networks and Deep Learning [Электронный ресурс]: a free online book / Michael Nielsen – 2019. – Режим доступа: http://neuralnetworksanddeeplearning.com
13. Data parallelism vs. model parallelism – How do they differ in distributed training? [Электронный ресурс]: a science-oriented article / analyticsindiamag.com – Научно-популярная статья – 2022 – Режим доступа: https://analyticsindiamag.com/data-parallelism-vs-model-parallelism-how-do-they-differ-in-distributed-training/
14. Vishakh Hegde, Sheema Usmani. Parallel and Distributed Deep Learning [Электронный ресурс]: a scientific article / Stanford University. – Научная публикация – Режим доступа: https://web.stanford.edu/~rezab/classes/cme323/S16/projects\_reports/hedge\_usmani.pdf
15. Introduction to Model Parallelism [Электронный ресурс]: a science-oriented article / Amazon Web Services. – Научно-популярная статья – Режим доступа: https://docs.aws.amazon.com/sagemaker/latest/dg/model-parallel-intro.html
16. Christopher J. Shallue, Jaehoon Lee, Joseph Antognini, Jascha Sohl-Dickstein, Roy Frostig, George E. Dahl. Measuring the Effects of Data Parallelism on Neural Network Training [Электронный ресурс]: a scientific article / Journal of Machine Learning Research. – Научная публикация – Журнал исследований в области машинного обучения, 2019 – Режим доступа: https://www.jmlr.org/papers/volume20/18-789/18-789.pdf
17. Python Software Foundation. Python 3.8.11 documentation [Электронный ресурс]: documentation for Python 3.8.11 / Python Software Foundation. – Электронная документация – Режим доступа: https://docs.python.org/3.8/
18. NumPy project and community. NumPy v1.18 Manual [Электронный ресурс]: the documentation for NumPy 1.18.4 / NumPy project. – Электронная документация – Режим доступа: https://numpy.org/doc/1.18/
19. Pillow (PIL Fork) 9.2.0.dev0 documentation [Электронный ресурс]: the documentation for Pillow 9.2.0.dev0 / Fredrik Lundh, Alex Clark and Contributors. – Электронная документация – Режим доступа: https://pillow.readthedocs.io/en/latest/
20. Typer – a library for building CLI applications [Электронный ресурс]: Tutorial / Sebastián Ramírez – Электронное руководство – Режим доступа: https://typer.tiangolo.com/
21. PyInstaller Manual [Электронный ресурс]: Manual for PyInstaller 5.1 / PyInstaller contributors – Электронное руководство – Режим доступа: https://pyinstaller.org/en/latest/
22. C/C++ reference [Электронный ресурс]: C++ documentation / Created and maintained by a group of C++ enthusiasts from around the world – Электронное руководство – Режим доступа: https://en.cppreference.com/w/
23. Документация по языку C++ [Электронный ресурс]: документация / Microsoft Corporation – Электронная документация – Режим доступа: https://docs.microsoft.com/ru-ru/cpp/cpp/?view=msvc-170
24. Eigen 3.4.0 documentation [Электронный ресурс]: documentation for Eigen 3 / Волонтёрский проект. – Электронная документация – Режим доступа: https://eigen.tuxfamily.org/dox
25. Wenzel Jakob, Jason Rhinelander, Dean Moldovan. Pybind11 – Seamless operability between C++11 and Python [Электронный ресурс]: documentation for pybind11 / Wenzel Jakob – 2017. – Электронная документация – Режим доступа: https://pybind11.readthedocs.io/en/stable/
26. Вигерс Карл. Разработка требований к программному обеспечению / Пер. с англ. – М.: Издательско-торговый дом «Русская Редакция», 2004. —576с.: ил.

# ПРИЛОЖЕНИЕ А. ИСХОДНЫЙ КОД НЕЙРОСЕТЕВОГО МОДУЛЯ

## Заголовочный файл нейронной сети и вспомогательных функций *NeuralNetwork.h*

#pragma once

#include <pybind11/pybind11.h>

#include <pybind11/Eigen.h>

#include "Rnd.h"

namespace py = pybind11;

using namespace Eigen;

typedef const VectorXd(\*function\_)(const VectorXd&, double);

class NeuralNetwork

{

private:

// network configuration

size\_t layers;

unsigned\* shape;

VectorXd\* activations;

MatrixXd\* weights;

VectorXd\* weighted\_sums;

VectorXd\* biases;

VectorXd\* deltas;

py::list function\_names;

function\_\* actfuncs;

function\_\* actfunc\_ders;

double\* func\_params;

// train parameters

Rnd<Index> rng;

static MatrixXd \*input, \*target;

VectorXd X, Y;

Index i;

unsigned epochs, test\_freq;

static double lr;

static unsigned total\_epochs;

// dynamic learing rate

static double delta\_lr;

static unsigned accuracy\_stuck\_limit;

static unsigned delta\_accuracy\_stuck\_limit;

// private methods

NeuralNetwork(const NeuralNetwork\* src);

void allocate\_memory();

void set\_activation\_functions();

void backprop(const VectorXd& Y, const double lr) noexcept;

void init\_train(MatrixXd\* input, MatrixXd\* target, const py::dict& config);

void init\_train(const int threads, const int thread\_num, const unsigned epochs);

void train(void) noexcept;

void monitor(const unsigned epoch) noexcept;

void load(const std::string& filename);

void copy\_state(const NeuralNetwork\* src);

void average\_state(NeuralNetwork\* family[], const unsigned count);

void train\_in\_parallel\_with\_averaging(int threads);

public:

NeuralNetwork(const py::dict& config);

NeuralNetwork(const std::string& dumpfile);

void inspect(void) const noexcept;

const VectorXd& forwardprop(const VectorXd& X);

const float test(const MatrixXd \*input, const MatrixXd \*target);

void dump(const std::string& filename) const;

~NeuralNetwork();

friend void launch\_train(NeuralNetwork& network, MatrixXd\* input, MatrixXd\* target, const py::dict& config);

};

const VectorXd sigmoid(const VectorXd& X, const double width);

const VectorXd sigmoid\_der(const VectorXd& X, const double width);

const VectorXd ReLU(const VectorXd& X, const double angle);

const VectorXd ReLU\_der(const VectorXd& X, const double angle);

function\_ get\_function\_by\_name(const std::string& name);

function\_ get\_function\_der\_by\_name(const std::string& name);

const VectorXi argsort(VectorXd X, const Index count);

const Index intersect1d\_len(VectorXi X, VectorXi Y);

const std::string get\_terminator(void) noexcept;

## Заголовочный файл генератора случайных чисел *Rnd.h*

#pragma once

#include <random>

template <typename T>

class Rnd

{

private:

std::mt19937 rng;

std::uniform\_int\_distribution<T> uni;

public:

Rnd(void) {};

Rnd(const T distribution\_start, const T distribution\_end);

const T operator()(void);

};

template <typename T>

Rnd<T>::Rnd(const T distribution\_start, const T distribution\_end) {

rng = std::mt19937(std::random\_device{}());

uni = std::uniform\_int\_distribution<T>(distribution\_start, distribution\_end);

}

template <typename T>

inline const T Rnd<T>::operator()(void) {

return uni(rng);

}

## Исходный код нейронной сети *NeuralNetwork.cpp*

#include <iostream>

#include "NeuralNetwork.h"

NeuralNetwork::NeuralNetwork(const py::dict& config) {

const py::list shape = config["shape"];

layers = py::len(shape);

allocate\_memory();

for (unsigned l = 0; l < layers; l++) {

this->shape[l] = py::int\_(shape[l]);

activations[l] = VectorXd::Zero(this->shape[l]);

}

for (unsigned l = 0; l < layers - 1; l++) {

weights[l] = MatrixXd::Random(this->shape[l + 1], this->shape[l]);

biases[l] = VectorXd::Zero(this->shape[l + 1]);

}

function\_names = config["activation\_functions"];

set\_activation\_functions();

const py::list function\_parameters = config["activation\_function\_parameters"];

if (py::len(function\_parameters) != layers - 1)

throw std::runtime\_error("network depth does not correspond to the parameter list");

double parameter;

for (int i = 0; i < layers - 1; i++) {

parameter = py::float\_(function\_parameters[i]);

func\_params[i] = parameter;

}

}

void NeuralNetwork::allocate\_memory() {

shape = new unsigned[layers];

activations = new VectorXd[layers];

weights = new MatrixXd[layers - 1];

weighted\_sums = new VectorXd[layers - 1];

biases = new VectorXd[layers - 1];

deltas = new VectorXd[layers - 1];

actfuncs = new function\_[layers - 1];

actfunc\_ders = new function\_[layers - 1];

func\_params = new double[layers - 1];

}

void NeuralNetwork::set\_activation\_functions() {

if (py::len(function\_names) != layers - 1)

throw std::runtime\_error("network depth does not correspond to the function list");

std::string function\_name;

for (int i = 0; i < layers - 1; i++) {

function\_name = py::str(function\_names[i]);

actfuncs[i] = get\_function\_by\_name(function\_name);

actfunc\_ders[i] = get\_function\_der\_by\_name(function\_name);

if (!actfuncs[i] || !actfunc\_ders[i]) throw std::runtime\_error("invalid function name has been passed");

}

}

void NeuralNetwork::inspect(void) const noexcept {

using namespace std;

// cout << "=============NeuralNetwork===============\n";

size\_t l;

cout << "Shape:";

for (l = 0; l < layers; l++) cout << " " << shape[l];

/\*

cout << "\n\nWeights:\n";

cout << "----------------------------------\n";

for (l = 0; l < layers - 1; l++) {

cout << weights[l];

cout << "\n----------------------------------\n";

}

cout << "\nActivations:\n";

cout << "----------------------------------\n";

for (l = 0; l < layers; l++) {

cout << activations[l].transpose() << '\n';

}

cout << "----------------------------------" << endl;

\*/

cout << "\nActivation functions:";

for (auto function : function\_names) cout << " " << static\_cast<string>(py::str(function));

cout << "\nActivation function parameters:";

for (l = 0; l < layers - 1; l++) cout << " " << func\_params[l];

cout << "\nTotal epochs: ";

cout << total\_epochs << endl;

}

const VectorXd& NeuralNetwork::forwardprop(const VectorXd& X) {

activations[0] = X;

for (unsigned l = 0; l < layers - 1; l++) {

// Eigen matrix multiplication requires optimization

weighted\_sums[l] = weights[l] \* activations[l] + biases[l];

activations[l + 1] = actfuncs[l](weighted\_sums[l], func\_params[l]);

}

return activations[layers - 1];

}

void NeuralNetwork::backprop(const VectorXd& Y, const double lr) noexcept {

deltas[layers - 2] = (Y - activations[layers - 1]).cwiseProduct(actfunc\_ders[layers - 2](weighted\_sums[layers - 2], func\_params[layers - 2]));

for (size\_t l = layers - 2; l > 0; l--) {

deltas[l - 1] = (weights[l].transpose() \* deltas[l]).cwiseProduct(actfunc\_ders[l - 1](weighted\_sums[l - 1], func\_params[l - 1]));

}

for (unsigned l = 0; l < layers - 1; l++) {

for (unsigned j = 0; j < shape[l + 1]; j++)

weights[l].row(j) += lr \* activations[l] \* deltas[l](j);

biases[l] += deltas[l];

}

}

MatrixXd\* NeuralNetwork::input = nullptr; MatrixXd\* NeuralNetwork::target = nullptr;

double NeuralNetwork::lr = 0;

double NeuralNetwork::delta\_lr = 1;

unsigned NeuralNetwork::accuracy\_stuck\_limit = UINT\_MAX;

unsigned NeuralNetwork::delta\_accuracy\_stuck\_limit = 1;

unsigned NeuralNetwork::total\_epochs = 0;

void NeuralNetwork::init\_train(MatrixXd \*input, MatrixXd \*target, const py::dict& config) {

this->input = input;

this->target = target;

if (input->cols() != shape[0] || target->cols() != shape[layers - 1])

throw std::runtime\_error("vector size does not fit the network shape");

epochs = py::int\_(config["epochs"]);

test\_freq = py::int\_(config["test\_frequency"]);

lr = py::float\_(config["rate"]);

if (py::bool\_(config["dynamic\_rate"])){

delta\_lr = py::float\_(config["rate\_delta"]);

accuracy\_stuck\_limit = py::int\_(config["accuracy\_stuck\_limit"]);

delta\_accuracy\_stuck\_limit = py::int\_(config["accuracy\_stuck\_limit\_delta"]);

if (!delta\_lr) throw std::runtime\_error("rate\_delta must be a non-zero value");

}

if (!py::bool\_(config["parallel\_training"]))

rng = Rnd<Index>(0, input->rows() - 1);

}

void NeuralNetwork::train(void) noexcept {

for (unsigned epoch = 1; epoch < epochs + 1; epoch++) {

if (!(epoch % test\_freq)) [[unlikely]] monitor(epoch);

i = rng();

X = input->row(i);

Y = target->row(i);

forwardprop(X);

backprop(Y, lr);

}

}

const float NeuralNetwork::test(const MatrixXd \*input, const MatrixXd \*target) {

Index expected\_predictions = 0, correct\_predictions = 0, ones;

VectorXd network\_output;

VectorXi real\_output\_indices, expected\_output\_indices;

float accuracy;

if (input->cols() != shape[0] || target->cols() != shape[layers - 1])

throw std::runtime\_error("vector size does not fit the network shape");

for (Index i = 0; i < input->rows(); i++) {

network\_output = forwardprop(input->row(i));

ones = target->row(i).count();

expected\_predictions += ones;

real\_output\_indices = argsort(network\_output, ones);

expected\_output\_indices = argsort(target->row(i), ones);

if (ones) correct\_predictions += intersect1d\_len(real\_output\_indices, expected\_output\_indices);

}

accuracy = static\_cast<float>(correct\_predictions) / static\_cast<float>(expected\_predictions);

accuracy = std::round(accuracy \* 1e4f) / 100;

return accuracy;

}

void NeuralNetwork::monitor(const unsigned epoch) noexcept {

static float accuracy, best\_accuracy = 0;

static unsigned accuracy\_not\_increased\_for = 0;

accuracy = test(input, target);

if (accuracy > best\_accuracy) {

best\_accuracy = accuracy;

accuracy\_not\_increased\_for = 0;

}

else accuracy\_not\_increased\_for += test\_freq;

if (accuracy\_not\_increased\_for > accuracy\_stuck\_limit) [[unlikely]] {

lr /= delta\_lr;

accuracy\_stuck\_limit \*= delta\_accuracy\_stuck\_limit;

accuracy\_not\_increased\_for = 0;

}

static const std::string terminator = get\_terminator();

std::cout << "\rEpoch " << epoch + total\_epochs << " | "

<< "Accuracy: " << accuracy << "% | "

<< "Best accuracy: " << best\_accuracy << "%" << terminator;

}

NeuralNetwork::~NeuralNetwork() {

delete[] shape;

delete[] activations;

delete[] weights;

delete[] weighted\_sums;

delete[] biases;

delete[] deltas;

delete[] actfuncs;

delete[] actfunc\_ders;

delete[] func\_params;

}

## Исходный код вспомогательных функций *Tools.cpp*

#include "NeuralNetwork.h"

function\_ get\_function\_by\_name(const std::string& name) {

if (name == "sigmoid") return &sigmoid;

else if (name == "ReLU") return &ReLU;

else return nullptr;

}

function\_ get\_function\_der\_by\_name(const std::string& name) {

if (name == "sigmoid") return &sigmoid\_der;

else if (name == "ReLU") return &ReLU\_der;

else return nullptr;

}

const VectorXd sigmoid(const VectorXd& X, const double width) {

VectorXd result(X.size());

for (unsigned i = 0; i < result.size(); i++)

result(i) = 1 / (1 + exp(-X(i) \* width));

return result;

}

const VectorXd sigmoid\_der(const VectorXd& X, const double width) {

VectorXd result(X.size());

result = sigmoid(X, width).cwiseProduct((-sigmoid(X, width).array() + 1).matrix());

return result;

}

const VectorXd ReLU(const VectorXd& X, const double angle) {

VectorXd result(X.size());

for (unsigned i = 0; i < result.size(); i++)

result(i) = fmax(0, X(i) \* angle);

return result;

}

const VectorXd ReLU\_der(const VectorXd& X, const double angle) {

VectorXd result(X.size());

for (unsigned i = 0; i < result.size(); i++)

result(i) = 0 ? X(i) < 0 : angle;

return result;

}

const VectorXi argsort(VectorXd X, const Index count) {

VectorXi indices = VectorXi::Zero(count);

int max\_ind;

for (Index i = 0; i < count; i++) {

X.maxCoeff(&max\_ind);

X(max\_ind) = -DBL\_MAX;

indices(i) = max\_ind;

}

return indices;

}

const Index intersect1d\_len(VectorXi X, VectorXi Y) {

std::vector<int> intersection;

std::sort(X.begin(), X.end()); std::sort(Y.begin(), Y.end());

std::set\_intersection(X.data(), X.data() + X.size(), Y.data(), Y.data() + Y.size(), std::back\_inserter(intersection));

return static\_cast<Index>(intersection.size());

}

## Исходный код сериализации конфигурации нейронной сети *Serialize.cpp*

#include "NeuralNetwork.h"

#include <fstream>

template <class Container> static void writeVector(const Container& V, std::ofstream& file);

static void writeMatrix(const MatrixXd& M, std::ofstream& file);

template <class Container> static const Container readVector(std::ifstream& file);

static const MatrixXd readMatrix(std::ifstream& file);

NeuralNetwork::NeuralNetwork(const std::string& dumpfile) {

load(dumpfile);

set\_activation\_functions();

}

void NeuralNetwork::dump(const std::string& filename) const {

std::ofstream file(filename, std::ios::binary);

if (!file.is\_open()) throw std::runtime\_error("failed to open a dump file");

file.write((const char\*)&layers, sizeof layers);

file.write((const char\*)shape, sizeof \*shape \* layers);

for (unsigned i = 0; i < layers; i++) writeVector(activations[i], file);

for (unsigned i = 0; i < layers - 1; i++) {

writeMatrix(weights[i], file);

writeVector(weighted\_sums[i], file);

writeVector(biases[i], file);

writeVector(deltas[i], file);

writeVector((const std::string)py::str(function\_names[i]), file);

}

file.write((const char\*)func\_params, sizeof \*func\_params \* (layers - 1));

file.write((const char\*)&total\_epochs, sizeof total\_epochs);

file.close();

}

void NeuralNetwork::load(const std::string& filename) {

std::ifstream file(filename, std::ios::binary);

if (!file.is\_open()) throw std::runtime\_error("failed to open a dump file");

file.read((char\*)&layers, sizeof layers);

this->allocate\_memory();

file.read((char\*)shape, sizeof \*shape \* layers);

for (unsigned i = 0; i < layers; i++) activations[i] = readVector<VectorXd>(file);

for (unsigned i = 0; i < layers - 1; i++) {

weights[i] = readMatrix(file);

weighted\_sums[i] = readVector<VectorXd>(file);

biases[i] = readVector<VectorXd>(file);

deltas[i] = readVector<VectorXd>(file);

function\_names.append(readVector<std::string>(file));

}

file.read((char\*)func\_params, sizeof \*func\_params \* (layers - 1));

file.read((char\*)&total\_epochs, sizeof total\_epochs);

file.close();

}

template <class Container>

static void writeVector(const Container& V, std::ofstream& file) {

size\_t size = V.size();

file.write((const char\*)&size, sizeof size);

file.write((const char\*)V.data(), sizeof \*V.data() \* size);

}

static void writeMatrix(const MatrixXd& M, std::ofstream& file) {

Index size[2] = { M.rows(), M.cols() };

file.write((const char\*)&size[0], sizeof size[0]);

file.write((const char\*)&size[1], sizeof size[1]);

file.write((const char\*)M.data(), sizeof \*M.data() \* M.size());

}

template <class Container>

static const Container readVector(std::ifstream& file) {

size\_t size;

file.read((char\*)&size, sizeof size);

Container V; V.resize(size);

file.read((char\*)V.data(), sizeof \*V.data() \* size);

return V;

}

static const MatrixXd readMatrix(std::ifstream& file) {

Index size[2];

file.read((char\*)&size[0], sizeof size[0]);

file.read((char\*)&size[1], sizeof size[1]);

MatrixXd M(size[0], size[1]);

file.read((char\*)M.data(), sizeof \*M.data() \* M.size());

return M;

}

## Исходный код распараллеливания обучения нейронной сети *Parallel.cpp*

#include <omp.h>

#include "NeuralNetwork.h"

NeuralNetwork::NeuralNetwork(const NeuralNetwork\* src) {

this->layers = src->layers;

allocate\_memory();

std::copy(src->shape, src->shape + layers, this->shape);

std::copy(src->actfuncs, src->actfuncs + layers - 1, this->actfuncs);

std::copy(src->actfunc\_ders, src->actfunc\_ders + layers - 1, this->actfunc\_ders);

std::copy(src->func\_params, src->func\_params + layers - 1, this->func\_params);

for (unsigned l = 0; l < layers - 1; l++) {

weights[l] = MatrixXd::Random(this->shape[l + 1], this->shape[l]);

biases[l] = VectorXd::Zero(this->shape[l + 1]);

}

}

void NeuralNetwork::init\_train(const int threads, const int thread\_num, const unsigned epochs) {

this->epochs = epochs;

this->test\_freq = epochs + 1;

static const Index block\_size = input->rows() / static\_cast<Index>(threads);

const Index block\_start = block\_size \* static\_cast<Index>(thread\_num);

const Index block\_end = block\_start + block\_size - 1;

rng = Rnd<Index>(block\_start, block\_end);

}

void NeuralNetwork::train\_in\_parallel\_with\_averaging(int threads) {

NeuralNetwork\*\* family = new NeuralNetwork\*[threads - 1];

for (int i = 0; i < threads - 1; i++) family[i] = new NeuralNetwork(this);

const unsigned global\_epochs = epochs;

# pragma omp parallel num\_threads(threads)

{

const int id = omp\_get\_thread\_num();

# pragma omp single

threads = omp\_get\_num\_threads();

if (id && total\_epochs) family[id - 1]->copy\_state(this);

if (id) family[id - 1]->init\_train(threads, id, test\_freq);

else this->init\_train(threads, id, test\_freq);

for (unsigned epoch = 1; epoch <= global\_epochs / epochs; epoch++) {

# pragma omp barrier

if (id) family[id - 1]->train();

else this->train();

# pragma omp barrier

# pragma omp master

{

this->average\_state(family, threads);

this->monitor(epoch \* epochs);

}

# pragma omp barrier

if (id) family[id - 1]->copy\_state(this);

}

}

for (int i = 0; i < threads - 1; i++) delete family[i];

delete[] family;

}

## Исходный код взаимодействия копий нейронной сети *State.cpp*

#include "NeuralNetwork.h"

void NeuralNetwork::copy\_state(const NeuralNetwork\* src) {

std::copy(src->weights, src->weights + layers - 1, this->weights);

std::copy(src->biases, src->biases + layers - 1, this->biases);

}

void NeuralNetwork::average\_state(NeuralNetwork\* family[], const unsigned count) {

size\_t l;

for (unsigned id = 0; id < count - 1; id++) {

for (l = 0; l < layers - 1; l++) {

this->weights[l] += family[id]->weights[l];

this->biases[l] += family[id]->biases[l];

}

}

for (l = 0; l < layers - 1; l++) {

this->weights[l] /= (double)count;

this->biases[l] /= (double)count;

}

}

## Исходный код инициализации терминатора строки *VTMode.cpp*

#ifndef WIN32\_LEAN\_AND\_MEAN

#define WIN32\_LEAN\_AND\_MEAN

#endif

#include <windows.h>

#include <string>

static const bool enable\_VT\_mode(void) {

HANDLE hOut = GetStdHandle(STD\_OUTPUT\_HANDLE);

if (hOut == INVALID\_HANDLE\_VALUE) return false;

DWORD dwMode = 0;

if (!GetConsoleMode(hOut, &dwMode)) return false;

dwMode |= ENABLE\_VIRTUAL\_TERMINAL\_PROCESSING;

if (!SetConsoleMode(hOut, dwMode)) return false;

return true;

}

const std::string get\_terminator(void) noexcept {

if (enable\_VT\_mode()) return "\x1b[0K";

else return " ";

}

## Исходный код сборки *Python*-модуля *Bind.cpp*

#include "NeuralNetwork.h"

static void launch\_train(NeuralNetwork& network, MatrixXd\* input, MatrixXd\* target, const py::dict& config) {

network.init\_train(input, target, config);

if (py::bool\_(config["parallel\_training"])) {

const int threads = py::int\_(config["threads"]);

network.train\_in\_parallel\_with\_averaging(threads);

}

else network.train();

network.total\_epochs += py::int\_(config["epochs"]);

}

PYBIND11\_MODULE(NeuralNetwork, mod) {

py::class\_<NeuralNetwork>(mod, "NeuralNetwork")

.def(py::init<const py::dict&>())

.def(py::init<const std::string&>())

.def("inspect", &NeuralNetwork::inspect)

.def("\_\_call\_\_", [](NeuralNetwork& network, const VectorXd& X) { return network.forwardprop(X); })

.def("train", &launch\_train)

.def("test", &NeuralNetwork::test)

.def("dump", &NeuralNetwork::dump);

}

# ПРИЛОЖЕНИЕ Б. ИСХОДНЫЙ КОД МОДУЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ И ИНТЕРФЕЙСА КОМАНДНОЙ СТРОКИ

## Исходный код интерфейса командной строки *CLI.py*

import time

import typer

from numpy import argwhere

from HandleConfig import getConfigInit, getConfigTrain, getConfigCorrupt

import Imagery

from GetData import getDataset, getVector

import Defaults

import NeuralNetwork as nn

app = typer.Typer(help = "CLI for a fully connected feedforward neural network")

@app.command("init", help = "Initialise a new network")

def init\_network(

config\_file: str = typer.Option(Defaults.\_CONFIG\_FILE, "-c", "--config-file", help = "Configuration file"),

dumpfile: str = typer.Option(Defaults.DUMPFILE, "-d", "--dumpfile", help = "A binary file to store the configuration")

):

config\_init = getConfigInit(config\_file)

network = nn.NeuralNetwork(config\_init)

network.dump(dumpfile)

typer.echo("A new neural network has been initialised.")

@app.command(help = "Show network configuration")

def inspect(

dumpfile: str = typer.Option(Defaults.DUMPFILE, "-d", "--dumpfile", help = "A binary file to load the configuration from")

):

network = nn.NeuralNetwork(dumpfile)

network.inspect()

@app.command("feed", help = "Get the output for a given input")

def feedforward(

dumpfile: str = typer.Option(Defaults.DUMPFILE, "-d", "--dumpfile", help = "A binary file to load the configuration"),

path: str = typer.Option(Defaults.DATASET + Defaults.CHANNEL + '/', "-p", "--path", help = "Path to data"),

filename: str = typer.Argument(..., help = "Name of the file of interest"),

threshold: float = typer.Option(Defaults.OUTPUT\_THRESHOLD, "-t", "--threshold", \

help = "Minimum value on the output neuron to consider the answer positive"),

show\_image: bool = typer.Option(False, "-s", "--show-image", help = "Open the input image")

):

if not 0 < threshold < 1: raise ValueError("Threshold must be in range (0, 1)")

img, img\_vector = getVector(path, filename)

network = nn.NeuralNetwork(dumpfile)

output = network(img\_vector)

output = argwhere(output > threshold).T[0]

print(\*output)

if show\_image: img.show()

img.close()

def \_getModel(dumpfile, data\_path, channel, reverse\_dataset, dataset\_size):

if channel not in Defaults.\_IMAGERY\_RANGE: raise ValueError("Channel must be R, G or B")

input\_data = data\_path + channel + '/'

ground\_truth = data\_path + Defaults.CSV\_FILENAME(channel)

X, Y = getDataset(input\_data, ground\_truth)

if reverse\_dataset: X = X[::-1]; Y = Y[::-1]

if not 0 < dataset\_size <= 1: raise ValueError("dataset\_size must be in the interval (0, 1]")

dataset\_size = int(X.shape[0] \* dataset\_size)

network = nn.NeuralNetwork(dumpfile)

return network, X[:dataset\_size], Y[:dataset\_size]

@app.command(help = "Run training")

def train(

data\_path: str = typer.Option(Defaults.DATASET, "-p", "--data-path", help = "Data for training"),

channel: str = typer.Option(Defaults.CHANNEL, "-ch", "--channel", help = f"A channel from the {Defaults.\_IMAGERY\_RANGE} range"),

reverse\_dataset: bool = typer.Option(False, "-r", "--reverse-dataset", help = "Invert the order of the elements in the dataset"),

dataset\_size: float = typer.Option(Defaults.TRAIN\_DATASET\_SIZE, "-s", "--dataset-size", \

help = "Multiply the size of the dataset by the value"),

config\_file: str = typer.Option(Defaults.\_CONFIG\_FILE, "-c", "--config-file", help = "Configuration file"),

dumpfile: str = typer.Option(Defaults.DUMPFILE, "-d", "--dumpfile", help = "A binary file to store and load the configuration"),

dump: bool = typer.Option(True, "--no-dump", help = "Do not dump the configuration", show\_default = False)

):

typer.echo("Initialising...", nl = False)

config\_train = getConfigTrain(config\_file)

network, X, Y = \_getModel(dumpfile, data\_path, channel, reverse\_dataset, dataset\_size)

typer.echo("\rTraining...\33[0K")

train\_time = time.time()

network.train(X, Y, config\_train)

train\_time = round(time.time() - train\_time)

m, s = divmod(train\_time, 60)

h, m = divmod(m, 60)

typer.echo(f"\nSession time: {str(h).zfill(2)}:{str(m).zfill(2)}:{str(s).zfill(2)}")

if dump: network.dump(dumpfile)

@app.command(help = "Run test")

def test(

data\_path: str = typer.Option(Defaults.DATASET, "-p", "--data-path", help = "Data for testing"),

channel: str = typer.Option(Defaults.CHANNEL, "-ch", "--channel", help = f"A channel from the {Defaults.\_IMAGERY\_RANGE} range"),

reverse\_dataset: bool = typer.Option(True, "-nr", "--no-reverse-dataset", show\_default = False, \

help = "Do not invert the order of the elements in the dataset"),

dataset\_size: float = typer.Option(round(1 - Defaults.TRAIN\_DATASET\_SIZE, 2), "-s", "--dataset-size", \

help = "Multiply the size of the dataset by the value"),

dumpfile: str = typer.Option(Defaults.DUMPFILE, "-d", "--dumpfile", help = "A binary file to load the configuration")

):

typer.echo("Initialising...", nl = False)

network, X, Y = \_getModel(dumpfile, data\_path, channel, reverse\_dataset, dataset\_size)

typer.echo("\rTesting...\33[0K")

accuracy = network.test(X, Y)

typer.echo(f"Accuracy: {round(accuracy, 2)}%")

@app.command(help = "Crop an image into patches")

def crop(

input\_file: str = typer.Option(Defaults.IMAGE\_FILE, "-i", "--input-path", help = "Path to an image to crop"),

output\_path: str = typer.Option(Defaults.PATCHES, "-o", "--output-path", help = "Output path"),

patch\_size: str = typer.Option(Defaults.PATCH\_SIZE, "-s", "--patch-size", \

help = f"Size of each patch, values must be separated with {Defaults.\_CLI\_SPLITTER}")

):

patch\_size = [int(value) for value in patch\_size.split(Defaults.\_CLI\_SPLITTER)]

Imagery.crop(input\_file, output\_path, patch\_size)

@app.command(help = "Corrupt the images with vertical strips")

def corrupt(

input\_path: str = typer.Option(Defaults.PATCHES, "-i", "--input-path", help = "Path to images to corrupt"),

output\_path: str = typer.Option(Defaults.DATASET, "-o", "--output-path", help = "Path to store the output to"),

channel: str = typer.Argument(..., help = "A channel from the RGB range to provide the result"),

csv\_filename\_template: str = typer.Option(Defaults.CSV\_FILENAME\_TEMPLATE, "-f", "--csv-filename", help = "CSV filename template"),

config\_file: str = typer.Option(Defaults.\_CONFIG\_FILE, "-c", "--config-file", help = "Configuration file")

):

csv\_filename = Defaults.CSV\_FILENAME(channel)

config\_corrupt = getConfigCorrupt(config\_file)

Imagery.corrupt(input\_path, output\_path, channel, csv\_filename, config\_corrupt)

if \_\_name\_\_ == "\_\_main\_\_":

app()

## Исходный код загрузки из файла конфигурации параметров по умолчанию *Defaults.py*

import os, sys

from HandleConfig import getConfigDefaults

if getattr(sys, "frozen", False): \_\_file\_\_ = sys.executable

\_CONFIG\_FILE = "config.ini"

\_IMAGERY\_RANGE = "RGB"

\_MAX\_PIX\_VAL = 255

\_SHIFT\_DIRECTION\_OPTIONS = "darker", "brighter", "any"

\_CLI\_SPLITTER = "\*"

config\_defaults = getConfigDefaults(\_CONFIG\_FILE)

relative\_path = True if config\_defaults["relative"].lower() == "true" else False

try:

\_NETWORK\_PATH = os.path.dirname(\_\_file\_\_).replace("\\", "/") + '/' if relative\_path else ""

\_NETWORK\_PATH += config\_defaults["network\_path"]

sys.path.append(\_NETWORK\_PATH)

except KeyError: pass

ITEMS\_PATH = os.path.dirname(\_\_file\_\_).replace("\\", "/") + '/' if relative\_path else ""

ITEMS\_PATH += config\_defaults["items\_path"]

if not ITEMS\_PATH.endswith('/'): ITEMS\_PATH += '/'

IMAGE\_FILE = ITEMS\_PATH + config\_defaults["image\_file"]

PATCHES = ITEMS\_PATH + config\_defaults["patches"]

if not PATCHES.endswith('/'): PATCHES += '/'

PATCH\_SIZE = config\_defaults['image\_resolution']

PATCH\_SIZE = PATCH\_SIZE.replace(config\_defaults["split\_char"][1:-1], \_CLI\_SPLITTER)

DATASET = ITEMS\_PATH + config\_defaults["dataset\_path"]

if not DATASET.endswith('/'): DATASET += '/'

CHANNEL = config\_defaults["channel"].upper()

CSV\_FILENAME\_TEMPLATE = config\_defaults["csv\_filename\_template"]

CSV\_FILENAME = lambda channel: f"{CSV\_FILENAME\_TEMPLATE}{channel}.csv"

DUMPFILE = config\_defaults["dumpfile"]

TRAIN\_DATASET\_SIZE = float(config\_defaults["train\_dataset\_size"])

OUTPUT\_THRESHOLD = float(config\_defaults["output\_threshold"])

## Исходный код взаимодействия с файлом конфигурации *HandleConfig.py*

from configparser import ConfigParser

from psutil import cpu\_count

def \_getConfig(config\_file: str) -> ConfigParser:

...

def getConfigInit(config\_file: str) -> dict:

config = \_getConfig(config\_file)

split\_char = config["meta"]["split\_char"][1:-1]

config\_init = dict(config.items("image\_resolution") + config.items("network\_configuration"))

for key in config\_init: config\_init[key] = config\_init[key].split(split\_char)

WIDTH, HEIGHT = [abs(int(val)) for val in config\_init["image\_resolution"]]

shape\_modifiers = [abs(float(val)) for val in config\_init["shape\_modifiers"]]

del config\_init["shape\_modifiers"]

config\_init["shape"] = [WIDTH \* HEIGHT]

for modifier in shape\_modifiers:

layer\_width = round(modifier \* WIDTH)

config\_init["shape"].append(layer\_width)

config\_init["shape"].append(WIDTH)

config\_init["activation\_function\_parameters"] = [float(val) for val in config\_init["activation\_function\_parameters"]]

return config\_init

def getConfigTrain(config\_file: str) -> dict:

config = \_getConfig(config\_file)

train\_parameters = dict(config.items("train\_parameters"))

for key in train\_parameters: train\_parameters[key] = float(train\_parameters[key])

dynamic\_rate = dict(config.items("dynamic\_rate"))

for key in "rate\_delta", "accuracy\_stuck\_limit", "accuracy\_stuck\_limit\_delta":

dynamic\_rate[key] = float(dynamic\_rate[key])

parallel\_training = dict(config.items("parallel\_training"))

threads = parallel\_training["threads"]

parallel\_training["threads"] = \

cpu\_count(logical = True) if threads.lower() == "auto" \

else cpu\_count(logical = False) if threads.lower() == "cores" \

else int(threads)

config\_train = dict(train\_parameters, \*\*dynamic\_rate, \*\*parallel\_training)

for key in "parallel\_training", "dynamic\_rate":

config\_train[key] = True if config\_train[key].lower() in ("true", "enable") else False

return config\_train

def getConfigCorrupt(config\_file: str) -> dict:

config = \_getConfig(config\_file)

config\_corrupt = dict(config.items("corruption"))

return config\_corrupt

def getConfigDefaults(config\_file: str) -> dict:

config = \_getConfig(config\_file)

config\_defaults = dict(config.items("meta") \

+ config.items("image\_resolution") \

+ config.items("default\_path") \

+ config.items("defaults") \

)

return config\_defaults

def \_getConfig(config\_file: str) -> ConfigParser:

config = ConfigParser()

config.read(config\_file)

return config

## Исходный код приведения данных для нейронной сети к векторному и матричному виду *GetData.py*

# A script to comprise all the necessary data

# For a neural network

import os, csv

import numpy as np

from PIL import Image

def getVector(path: str, filename: str):

img = Image.open(path + filename)

img\_vector = np.asarray(img).reshape(img.width \* img.height)

return img, img\_vector

def getDataset(input\_path: str, ground\_truth\_path: str):

X = []

for file in os.listdir(input\_path):

img, img\_vector = getVector(input\_path, file)

img.close()

X.append(img\_vector)

X = np.array(X)

Y = []

with open(ground\_truth\_path, "r", newline = "\n") as csvfile:

reader = csv.DictReader(csvfile, delimiter = ";")

for row in reader:

y = row["Ids"]

if len(y): y = np.array(y.split(","), dtype = "int8")

else: y = np.array([])

Y.append(y)

for i in range(len(Y)):

y = np.zeros(img.width)

for j in Y[i]:

y[j] = 1

Y[i] = y

Y = np.array(Y)

return X, Y

## Исходный код модуля обработки изображений *Imagery.py*

# Create artificial noise on each image

# The resulting images are in the chosen channel

import os, random, sys

import csv

from numpy import array

from typer import echo

from PIL import Image

from Defaults import \_MAX\_PIX\_VAL, \_SHIFT\_DIRECTION\_OPTIONS, \_IMAGERY\_RANGE

def crop(input\_file: str, output\_path: str, patch\_size: tuple):

\_makeDir(output\_path)

scene = Image.open(input\_file)

width, height = patch\_size

for i in range(0, scene.width - width, width):

for j in range(0, scene.height - height, height):

patch = scene.crop((i, j, i + width, j + height))

patch.save(output\_path + f"{i}, {j}.bmp", 'bmp')

patch.close()

percent = round(i / (scene.width - width) \* 100)

echo(f"\rCropping... {percent}%\33[0K", nl = False)

echo(f"\rThe image has been successfully cropped.\33[0K")

scene.close()

def corrupt(input\_path: str, output\_path: str, channel: str, csv\_filename: str, config: dict):

if channel not in \_IMAGERY\_RANGE: raise ValueError(f"Channel must be in range {\_IMAGERY\_RANGE}")

strip\_freq = float(config["strip\_frequency"])

if not 0 < strip\_freq < 1: raise ValueError("Corruption probability must be in range (0, 1)")

min\_shift = int(config["min\_shift"])

if not 0 <= min\_shift <= \_MAX\_PIX\_VAL: raise ValueError(f"Minimal deviation must be in range [0, {\_MAX\_PIX\_VAL}]")

shift\_direction = config["shift\_direction"]

if shift\_direction not in \_SHIFT\_DIRECTION\_OPTIONS: raise ValueError(f"Shift direction must be in {\_SHIFT\_DIRECTION\_OPTIONS}")

images = os.listdir(input\_path)

corrupted\_image\_path = output\_path + channel + '/'

csv\_file\_path = output\_path + csv\_filename

channel = \_IMAGERY\_RANGE.index(channel)

\_makeDir(corrupted\_image\_path)

Y = []

for file in images:

patch = Image.open(input\_path + file)

patch = patch.convert(\_IMAGERY\_RANGE).split()[channel]

strip\_ids = \_makeNoise(patch, strip\_freq, min\_shift, shift\_direction)

Y.append(strip\_ids)

patch.save(corrupted\_image\_path + file, 'bmp')

patch.close()

ind = images.index(file)

if ind % 100 == 0:

percent = round(ind / len(images) \* 100)

echo(f"\rCorrupting images... {percent}%\33[0K", nl = False)

# The ordinary number of each strip in each image gets stored into a file

# The data is used as ground truth for a neural network

with open(csv\_file\_path, "w", newline = "\n") as csvfile:

fields = "Image", "Ids"

writer = csv.DictWriter(csvfile, fieldnames = fields, delimiter = ";")

writer.writeheader()

for i, \_ in enumerate(Y):

writer.writerow({"Image": images[i], "Ids": Y[i]})

if i % 100 == 0:

percent = round(i / len(Y) \* 100)

echo (f"\rComprising data... {percent}%\33[0K", nl = False)

echo(f"\rThe patches have been successfully corrupted.\33[0K")

def \_makeDir(path: str):

try: os.makedirs(path)

except FileExistsError:

items = os.listdir(path)

if items:

echo("The chosen directory already contains files. All the files in the directory will be deleted.")

prompt = input("Type in Y/y to proceed\n").lower()

if prompt == "y":

for item in items:

try: os.remove(path + item)

except PermissionError: continue

percent = round(items.index(item) / len(items) \* 100)

echo(f"\rClearing... {percent}%\33[0K", nl = False)

else:

echo("Aborted!")

sys.exit(0)

def \_makeNoise(img, strip\_freq: float, min\_shift: int, shift\_direction: str):

imgMatrix = img.load()

strip\_ids = []

for i in range(img.width):

if random.random() < strip\_freq:

column = array([imgMatrix[i, j] for j in range(img.height)])

maxPix = column.max(); minPix = column.min()

a = random.uniform(0 if shift\_direction != "brighter" else 1, \

((\_MAX\_PIX\_VAL / maxPix) if (maxPix and shift\_direction != "darker") else 1))

b = random.randint(-int(a \* minPix) if shift\_direction != "brighter" else 0, \

(\_MAX\_PIX\_VAL - int(a \* maxPix)) if shift\_direction != "darker" else 0)

if any(abs(column - (a \* column + b)) < min\_shift): continue

for j in range(img.height): imgMatrix[i, j] = int(a \* imgMatrix[i, j] + b)

strip\_ids.append(str(i))

strip\_ids = ",".join(strip\_ids)

return strip\_ids

1. Электронный тестовый сигнал [↑](#footnote-ref-1)
2. Тестовый видеосигнал, информирующий о работоспособности видеотракта модуля видимого диапазона [↑](#footnote-ref-2)