Министерство науки и высшего образования РФ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Глазовский государственный педагогический институт имени В.Г. Короленко»

**ОТЧЕТ**

**по производственной практике**

**ПМ.03 Участие в интеграции программных модулей профессионального цикла**

**Специальность 09.02.03 Программирование в компьютерных системах**

|  |
| --- |
| Выполнил(а): |
| студент(ка) ОП СПО группа 31 |
| Мальчиков Н.С. |
|  |
| Руководитель практики: старший преподаватель кафедры математики и информатики Касаткин К.А. |

г. Глазов 2020

Оглавление

[Информация о предприятии 3](#_Toc39856905)

[Практическая деятельность на предприятии. 6](#_Toc39856906)

[Самоанализ пройденной практики 7](#_Toc39856907)

[Индивидуальное задание 8](#_Toc39856908)

[Описание интерфейса 8](#_Toc39856909)

[Структура теста и библиотеки для нейросети. 10](#_Toc39856910)

[Алгоритм работы программы 11](#_Toc39856911)

[Исходный код 12](#_Toc39856912)

[Тестирование 27](#_Toc39856913)

# Информация о предприятии

**Полное наименование предприятия**

Федеральное государственное бюджетное образовательное учреждение высшего образования «Глазовский государственный педагогический институт имени В.Г. Короленко»

**Организационно-правовая форма**

Федеральные государственные бюджетные учреждения

**Юридический адрес**

427621, Республика Удмуртская, г. Глазов, ул. Первомайская, дом 25

**ФИО руководителя предприятия**

Чиговская-Назарова Янина Александровна

**Основная деятельность предприятия**

Основной целью деятельности Управления является организация образовательной деятельности по заочной и очной формам обучения и деятельности по оказанию платных образовательных услуг различным группам населения.

**ФИО руководителя практики**

Касаткин Кирилл Александрович

**Структура ФГБОУ «Глазовский государственный педагогический институт имени В.Г. Короленко» включает в себя следующие подразделения**

Наименование структурного подразделения

Кафедры

* Дошкольного и начального образования
* Иностранных языков и удмуртской филологии
* Истории и социально-гуманитарных дисциплин
* Медико-биологических дисциплин
* Музыкального образования
* Педагогики и психологии
* Русского языка и литературы
* Физики и дидактики физики
* Математики и информатики
* Физической культуры, методики и спорта

Факультеты

* Факультет информатики физики и математики
* Историко-лингвистический факультет
* Социальных коммуникаций и филологии
* Факультет педагогического и художественного образования

Советы

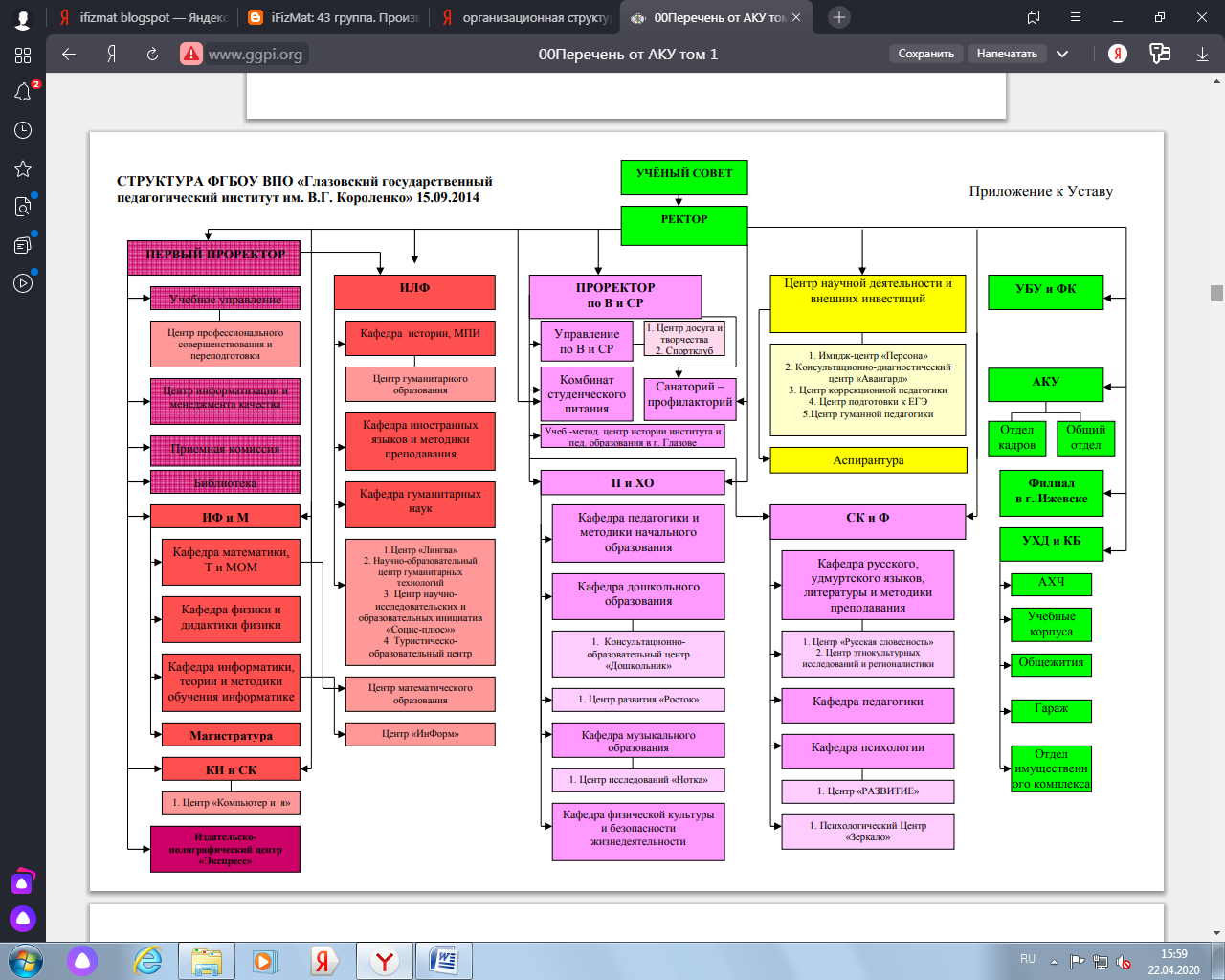
* Совет обучающихся

Другие структурные подразделения

* Административно-кадровое управление
* Учебное управление
* Центр научной и международной деятельности
* Управление по воспитательной и социальной работе
* Управление бухгалтерского учета и финансового контроля
* Управление имущественного комплекса
* Библиотека
* Центр информатизации и дистанционного обучения
* Центр полиграфических услуг ГГПИ "ЭксПресс"
* Центр истории ГГПИ
* Комбинат студенческого питания

Представленная организационная структура

показывает взаимосвязь внутренних подразделений



# Практическая деятельность на предприятии.

**Подготовка к работе, настройка и обслуживание аппаратного обеспечения и операционной системы персонального компьютера**

* Монтаж оперативной памяти в медиатеке 3-го учебного корпуса.
* Снятие оперативной памяти с нерабочих ПК.

**Подготовка к работе, настройка и обслуживание периферийных устройств персонального компьютера и компьютерной оргтехники**

* Копирование данных с жёсткого диска.
* Сборка корпуса проектора для сцены.
* Распаковка и тестирование интерактивного экрана(телевизора).
* Протяжка витой пары.
* Монтаж ЛВС в санатории-профилактории (3 общежитие)

**Создание и управление на персональном компьютере текстовыми документами, таблицами, презентациями и содержанием баз данных**

**Осуществление навигации по ресурсам, поиск, ввод и передача данных с помощью технологий и сервисов Интернета**

**Обеспечение мер по информационной безопасности**

# Самоанализ пройденной практики

Объем выполнения работы был полный.

Задания выполнялись максимально быстро без задержек.

Поставленные задачи и цели были достигнуты.

При отсутствии задач выполнялись самостоятельно найденные задачи или индивидуальное задание.

Алгоритм обслуживания ПК:

* Разборка корпуса
* Продувка от пыли
* Снятие кулера
* Очистка кулера и радиатора от пыли
* Очистка процессора от термопасты
* Нанесение новой термопасты на процессор
* Установка кулера
* Сборка корпуса
* Запуск компьютера для тестирования

# Индивидуальное задание

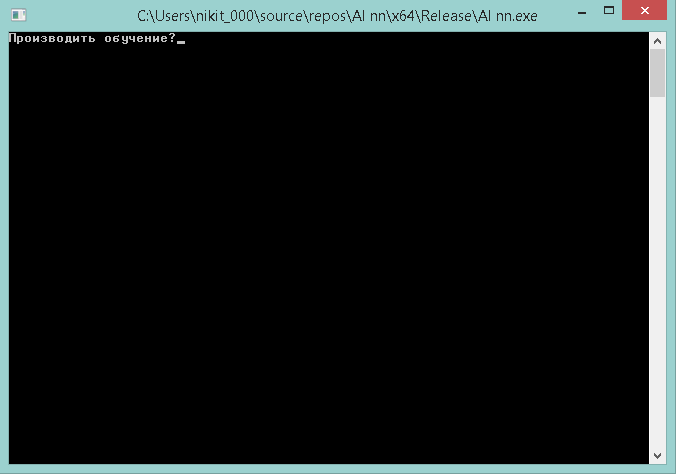
«Большие данные и машинное обучение.»

Во время прохождения практики была разработана программа на языке C++. Программа умеет распознавать английские буквы и обучаться распознанию этих букв.

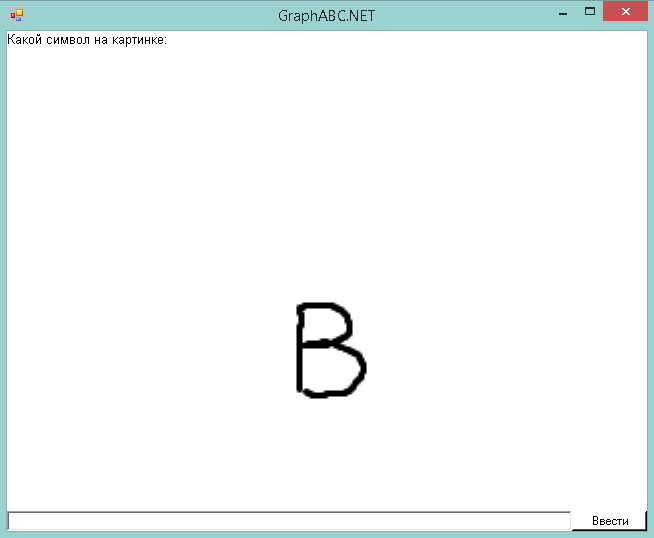
Описание интерфейса

Программа разделена на 2 части:

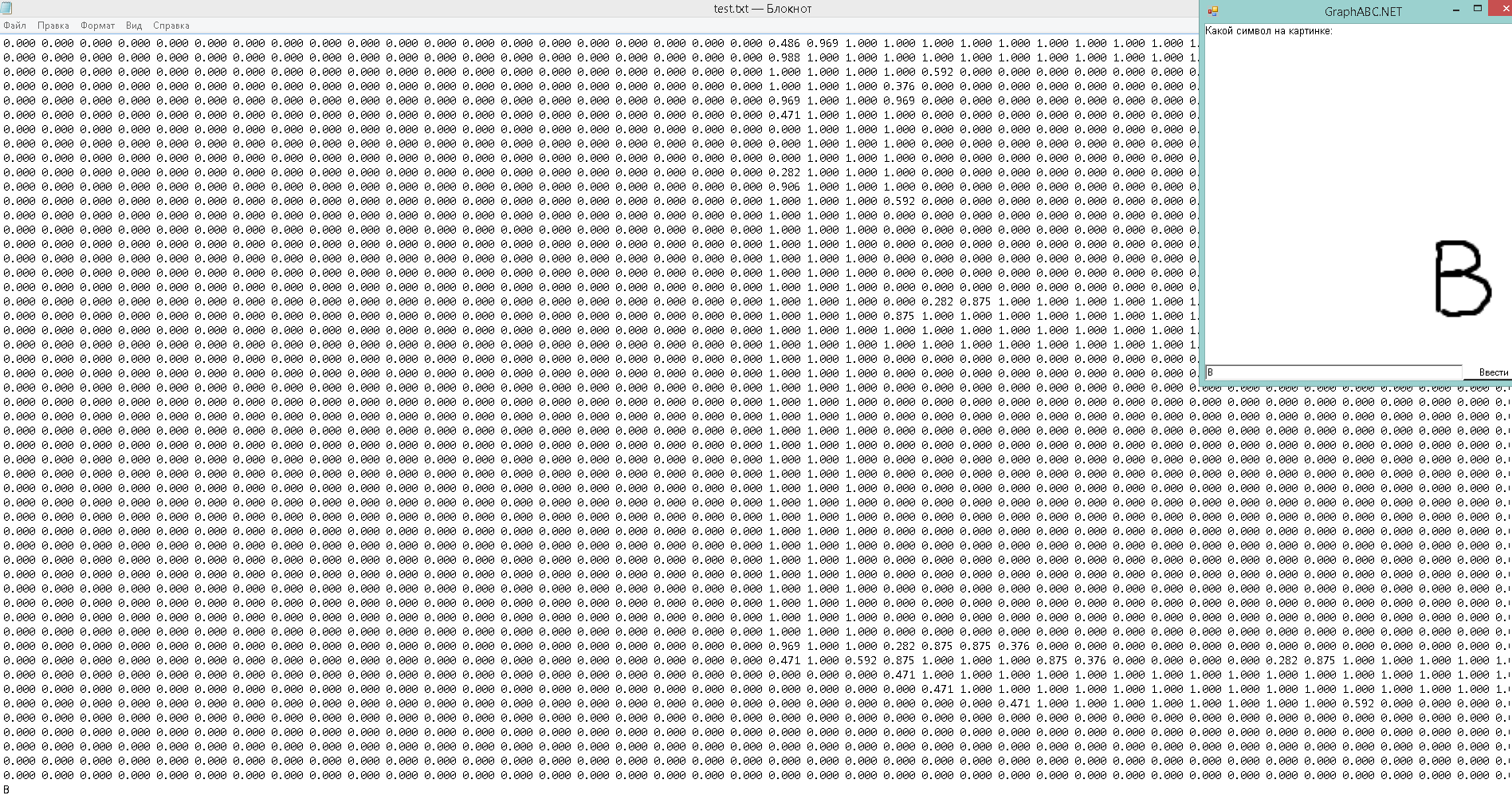
1. Для распознавания букв и обучения этим буквам.



1. Для создания тестов и библиотеки букв.



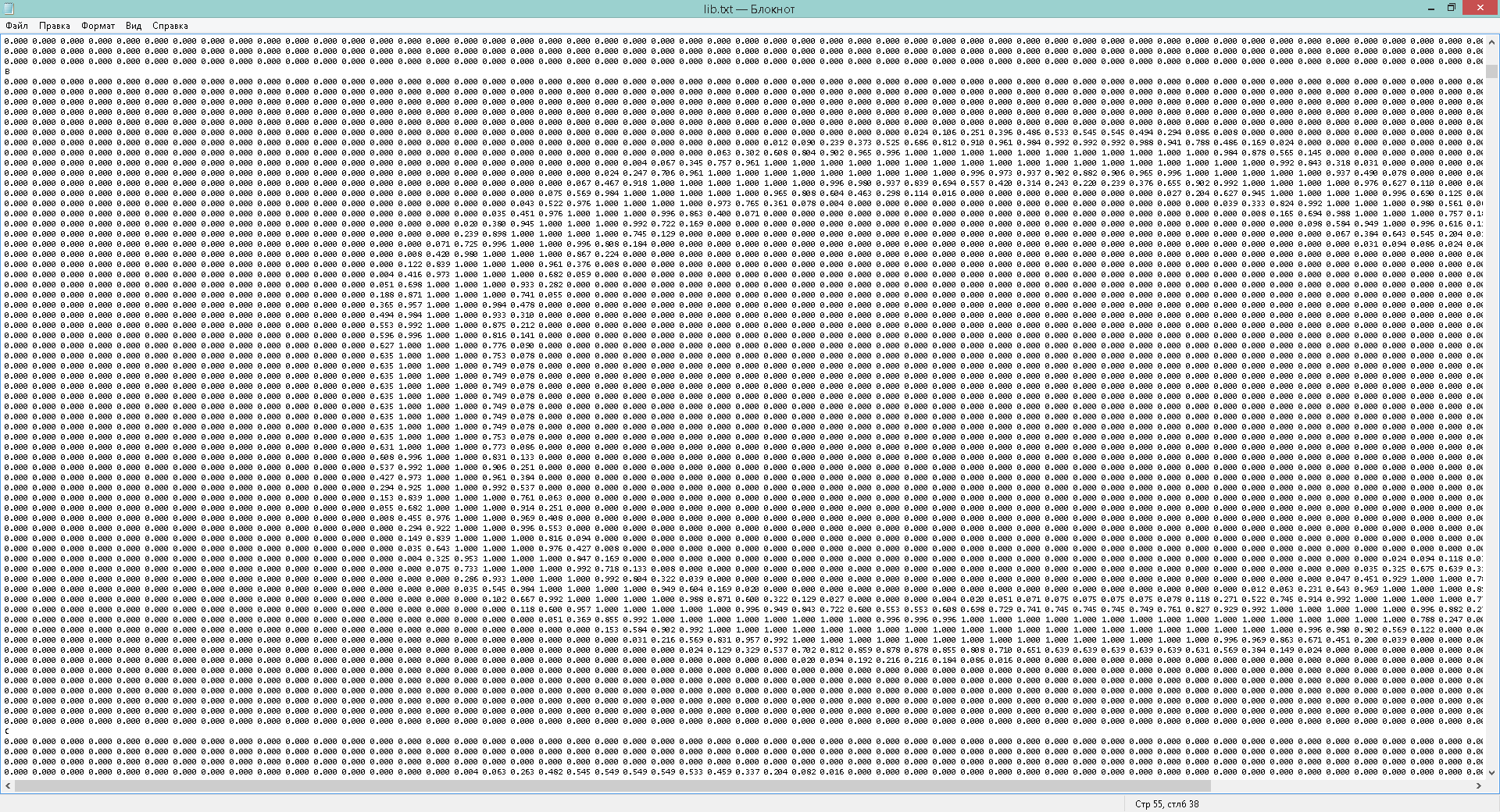
В основной программе производится обучение нейросети английским буквам, которые предварительно были записаны из формата .JPG с разрешением 64x64 в текстовый файл. Для преобразования изображения в текст был специально написан скрипт на языке программирования pascal, который выполнял данную процедуру.



## Структура теста и библиотеки для нейросети.

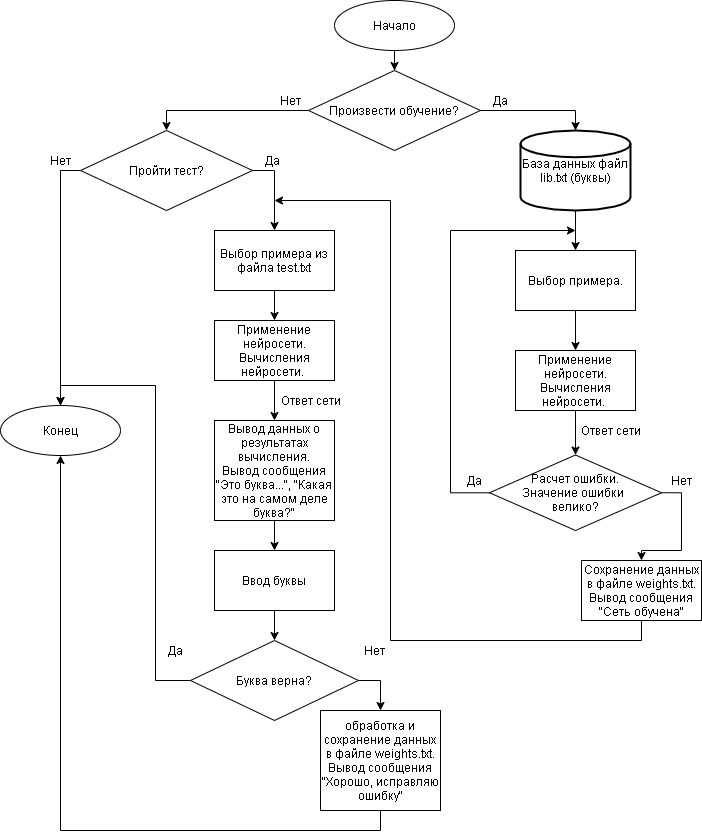
Для правильного обучения и проверки необходимо соблюдать следующую структуру txt файла теста и библиотеки:

1. Буква, переведенная в текс (код).
2. Сама буква.



В тесте должна находиться всего одна буква. Пустых строк в файле быть не должно.

## Алгоритм работы программы



## Исходный код

**AL nn (основная программа)**

#include<fstream>

#include<thread>

#include<random>

#include<time.h>

#include<Windows.h>

#include<iostream>

using namespace std;

struct neuron {

double value;

double error;

void act() {

value = (1 / (1 + pow(2.71828, -value)));

}

};

struct data\_one {

double info[4096];

char rresult;

};

class network {

public:

int layers;

neuron\*\* neurons;

double\*\*\* weights;

int\* size;

int threadsNum = 1;

double sigm\_pro(double x) {

if ((fabs(x - 1) < 1e-9) || (fabs(x) < 1e-9)) return 0.0;

double res = x \* (1.0 - x);

return res;

}

double predict(double x) {

if (x >= 0.9) {

return 1;

}

else {

return 0;

}

}

void setLayersNotStudy(int n, int\* p, string filename) {

ifstream fin;

fin.open(filename);

srand(time(0));

layers = n;

neurons = new neuron \* [n];

weights = new double\*\* [n - 1];

size = new int[n];

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

size[i] = p[i];

neurons[i] = new neuron[p[i]];

if (i < n - 1) {

weights[i] = new double\* [p[i]];

for (int j = 0; j < p[i]; j++) {

weights[i][j] = new double[p[i + 1]];

for (int k = 0; k < p[i + 1]; k++) {

fin >> weights[i][j][k];

}

}

}

}

}

void setLayers(int n, int\* p) {

srand(time(0));

layers = n;

neurons = new neuron \* [n];

weights = new double\*\* [n - 1];

size = new int[n];

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

size[i] = p[i];

neurons[i] = new neuron[p[i]];

if (i < n - 1) {

weights[i] = new double\* [p[i]];

for (int j = 0; j < p[i]; j++) {

weights[i][j] = new double[p[i + 1]];

for (int k = 0; k < p[i + 1]; k++) {

weights[i][j][k] = ((rand() % 100)) \* 0.01 / size[i];

}

}

}

}

}

void setRandomInput() {

for (int i = 0; i < size[0]; i++) {

neurons[0][i].value = (rand() % 256) / 255;

}

}

void set\_input(double p[]) {

for (int i = 0; i < size[0]; i++) {

neurons[0][i].value = p[i];

}

}

void show() {

setlocale(LC\_ALL, "ru");

cout << "Ядер процессора: " << thread::hardware\_concurrency() << endl;

cout << "Нейронная сеть имеет архитектуру: ";

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

cout << size[i];

if (i < layers - 1) {

cout << " - ";

}

}cout << endl;

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

cout << "\n#Слой " << i + 1 << "\n\n";

for (int j = 0; j < size[i]; j++) {

cout << "Нейрон #" << j + 1 << ": \n";

cout << "Значение: " << neurons[i][j].value << endl;

if (i < layers - 1) {

cout << "Веса: \n";

for (int k = 0; k < size[i + 1]; k++) {

cout << "#" << k + 1 << ": ";

cout << weights[i][j][k] << endl;

}

}

}

}

}

void LayersCleaner(int LayerNumber, int start, int stop) {

srand(time(0));

for (int i = start; i < stop; i++) {

neurons[LayerNumber][i].value = 0;

}

}

void ForwardFeeder(int LayerNumber, int start, int stop) {

for (int j = start; j < stop; j++) {

for (int k = 0; k < size[LayerNumber - 1]; k++) {

neurons[LayerNumber][j].value += neurons[LayerNumber - 1][k].value \* weights[LayerNumber - 1][k][j];

}

neurons[LayerNumber][j].act();

}

}

double ForwardFeed() {

setlocale(LC\_ALL, "ru");

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

if (threadsNum == 1) {

thread th1([&]() {

LayersCleaner(i, 0, size[i]);

});

th1.join();

}

if (threadsNum == 2) {

//cout << "Выполняю очистку слоя 2-мя ядрами...\n";

thread th1([&]() {

LayersCleaner(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {

LayersCleaner(i, int(floor(size[i] / 2)), size[i]);

});

th1.join();

th2.join();

}

if (threadsNum == 4) {

if (size[i] == 1) {

cout << "Выполняю очистку слоя 1-м ядром...\n";

thread th1([&]() {

LayersCleaner(i, 0, size[i]);

});

th1.join();

}

if ((size[i] == 2) or (size[i] == 3)) {

cout << "Выполняю очистку слоя 2-мя ядрами...\n";

thread th1([&]() {

LayersCleaner(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {LayersCleaner(i, int(floor(size[i] / 2)), size[i]); });

th1.join();

th2.join();

}

if (size[i] >= 4) {

//cout << "Выполняю очистку слоя 4-мя ядрами...\n";

int start1 = 0; int stop1 = int(size[i] / 4);

int start2 = int(size[i] / 4); int stop2 = int(size[i] / 2);

int start3 = int(size[i] / 2); int stop3 = int(size[i] - floor(size[i] / 4));

int start4 = int(size[i] - floor(size[i] / 4)); int stop4 = size[i];

thread th1([&]() {LayersCleaner(i, start1, stop1); });

thread th2([&]() {LayersCleaner(i, start2, stop2); });

thread th3([&]() {LayersCleaner(i, start3, stop3); });

thread th4([&]() {LayersCleaner(i, start4, stop4); });

th1.join();

th2.join();

th3.join();

th4.join();

}

}

if (threadsNum == 1) {

thread th1([&]() {

ForwardFeeder(i, 0, size[i]);

});

th1.join();

}

if (threadsNum == 2) {

thread th1([&]() {

ForwardFeeder(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {

ForwardFeeder(i, int(floor(size[i] / 2)), size[i]);

});

th1.join();

th2.join();

}

if (threadsNum == 4) {

if ((size[i] == 2) or (size[i] == 3)) {

thread th1([&]() {

ForwardFeeder(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {

ForwardFeeder(i, int(floor(size[i] / 2)), size[i]);

});

th1.join();

th2.join();

}

if (size[i] >= 4) {

int start1 = 0; int stop1 = int(floor(size[i] / 4));

int start2 = int(size[i] / 4); int stop2 = int(size[i] / 2);

int start3 = int(size[i] / 2); int stop3 = int(size[i] - floor(size[i] / 4));

int start4 = int(size[i] - floor(size[i] / 4)); int stop4 = size[i];

thread th1([&]() {ForwardFeeder(i, start1, stop1); });

thread th2([&]() {ForwardFeeder(i, start2, stop2); });

thread th3([&]() {ForwardFeeder(i, start3, stop3); });

thread th4([&]() {ForwardFeeder(i, start4, stop4); });

th1.join();

th2.join();

th3.join();

th4.join();

}

}

}

double max = 0;

double prediction = 0;

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

if (neurons[layers - 1][i].value > max) {

max = neurons[layers - 1][i].value;

prediction = i;

}

}

return prediction;

}

double ForwardFeed(string param) {

setlocale(LC\_ALL, "ru");

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

if (threadsNum == 1) {

//cout << "Выполняю очистку слоя 1-м ядром...\n";

thread th1([&]() {

LayersCleaner(i, 0, size[i]);

});

th1.join();

}

if (threadsNum == 2) {

//cout << "Выполняю очистку слоя 2-мя ядрами...\n";

thread th1([&]() {

LayersCleaner(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {

LayersCleaner(i, int(floor(size[i] / 2)), size[i]);

});

th1.join();

th2.join();

}

if (threadsNum == 4) {

if (size[i] == 1) {

cout << "Выполняю очистку слоя 1-м ядром...\n";

thread th1([&]() {

LayersCleaner(i, 0, size[i]);

});

th1.join();

}

if ((size[i] == 2) or (size[i] == 3)) {

cout << "Выполняю очистку слоя 2-мя ядрами...\n";

thread th1([&]() {

LayersCleaner(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {LayersCleaner(i, int(floor(size[i] / 2)), size[i]); });

th1.join();

th2.join();

}

if (size[i] >= 4) {

//cout << "Выполняю очистку слоя 4-мя ядрами...\n";

int start1 = 0; int stop1 = int(size[i] / 4);

int start2 = int(size[i] / 4); int stop2 = int(size[i] / 2);

int start3 = int(size[i] / 2); int stop3 = int(size[i] - floor(size[i] / 4));

int start4 = int(size[i] - floor(size[i] / 4)); int stop4 = size[i];

thread th1([&]() {LayersCleaner(i, start1, stop1); });

thread th2([&]() {LayersCleaner(i, start2, stop2); });

thread th3([&]() {LayersCleaner(i, start3, stop3); });

thread th4([&]() {LayersCleaner(i, start4, stop4); });

th1.join();

th2.join();

th3.join();

th4.join();

}

}

if (threadsNum == 1) {

thread th1([&]() {

ForwardFeeder(i, 0, size[i]);

});

th1.join();

}

if (threadsNum == 2) {

thread th1([&]() {

ForwardFeeder(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {

ForwardFeeder(i, int(floor(size[i] / 2)), size[i]);

});

th1.join();

th2.join();

}

if (threadsNum == 4) {

if ((size[i] == 2) or (size[i] == 3)) {

thread th1([&]() {

ForwardFeeder(i, 0, int(floor(size[i] / 2)));

});

thread th2([&]() {

ForwardFeeder(i, int(floor(size[i] / 2)), size[i]);

});

th1.join();

th2.join();

}

if (size[i] >= 4) {

int start1 = 0; int stop1 = int(floor(size[i] / 4));

int start2 = int(size[i] / 4); int stop2 = int(size[i] / 2);

int start3 = int(size[i] / 2); int stop3 = int(size[i] - floor(size[i] / 4));

int start4 = int(size[i] - floor(size[i] / 4)); int stop4 = size[i];

thread th1([&]() {ForwardFeeder(i, start1, stop1); });

thread th2([&]() {ForwardFeeder(i, start2, stop2); });

thread th3([&]() {ForwardFeeder(i, start3, stop3); });

thread th4([&]() {ForwardFeeder(i, start4, stop4); });

th1.join();

th2.join();

th3.join();

th4.join();

}

}

}

double max = 0;

double prediction = 0;

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

cout << char(i + 65) << " : " << neurons[layers - 1][i].value << endl;

if (neurons[layers - 1][i].value > max) {

max = neurons[layers - 1][i].value;

prediction = i;

}

}

return prediction;

}

void ErrorCounter(int LayerNumber, int start, int stop, double prediction, double rresult, double lr) {

if (LayerNumber == layers - 1) {

for (int j = start; j < stop; j++) {

if (j != int(rresult)) {

neurons[LayerNumber][j].error = -pow((neurons[LayerNumber][j].value), 2);

}

else {

neurons[LayerNumber][j].error = pow(1.0 - neurons[LayerNumber][j].value, 2);

}

}

}

else {

for (int j = start; j < stop; j++) {

double error = 0.0;

for (int k = 0; k < size[LayerNumber + 1]; k++) {

error += neurons[LayerNumber + 1][k].error \* weights[LayerNumber][j][k];

}

neurons[LayerNumber][j].error = error;

}

}

}

void WeightsUpdater(int start, int stop, int LayerNum, int lr) {

int i = LayerNum;

for (int j = start; j < stop; j++) {

for (int k = 0; k < size[i + 1]; k++) {

weights[i][j][k] += lr \* neurons[i + 1][k].error \* sigm\_pro(neurons[i + 1][k].value) \* neurons[i][j].value;

}

}

}

void BackPropogation(double prediction, double rresult, double lr) {

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

if (threadsNum == 1) {

if (i == layers - 1) {

for (int j = 0; j < size[i]; j++) {

if (j != int(rresult)) {

neurons[i][j].error = -pow((neurons[i][j].value), 2);

}

else {

neurons[i][j].error = pow(1.0 - neurons[i][j].value, 2);

}

}

}

else {

for (int j = 0; j < size[i]; j++) {

double error = 0.0;

for (int k = 0; k < size[i + 1]; k++) {

error += neurons[i + 1][k].error \* weights[i][j][k];

}

neurons[i][j].error = error;

}

}

}

if (threadsNum == 2) {

thread th1([&]() {

ErrorCounter(i, 0, int(size[i] / 2), prediction, rresult, lr);

});

thread th2([&]() {

ErrorCounter(i, int(size[i] / 2), size[i], prediction, rresult, lr);

});

th1.join();

th2.join();

}

if (threadsNum == 4) {

if (size[i] < 4) {

if (i == layers - 1) {

for (int j = 0; j < size[i]; j++) {

if (j != int(rresult)) {

neurons[i][j].error = -(neurons[i][j].value);

}

else {

neurons[i][j].error = 1.0 - neurons[i][j].value;

}

}

}

else {

for (int j = 0; j < size[i]; j++) {

double error = 0.0;

for (int k = 0; k < size[i + 1]; k++) {

error += neurons[i + 1][k].error \* weights[i][j][k];

}

neurons[i][j].error = error;

}

}

}

if (size[i] >= 4) {

int start1 = 0; int stop1 = int(size[i] / 4);

int start2 = int(size[i] / 4); int stop2 = int(size[i] / 2);

int start3 = int(size[i] / 2); int stop3 = int(size[i] - floor(size[i] / 4));

int start4 = int(size[i] - floor(size[i] / 4)); int stop4 = size[i];

thread th1([&]() {

ErrorCounter(i, start1, stop1, prediction, rresult, lr);

});

thread th2([&]() {

ErrorCounter(i, start2, stop2, prediction, rresult, lr);

});

thread th3([&]() {

ErrorCounter(i, start3, stop3, prediction, rresult, lr);

});

thread th4([&]() {

ErrorCounter(i, start4, stop4, prediction, rresult, lr);

});

th1.join();

th2.join();

th3.join();

th4.join();

}

}

}

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

if (threadsNum == 1) {

for (int j = 0; j < size[i]; j++) {

for (int k = 0; k < size[i + 1]; k++) {

weights[i][j][k] += lr \* neurons[i + 1][k].error \* sigm\_pro(neurons[i + 1][k].value) \* neurons[i][j].value;

}

}

}

if (threadsNum == 2) {

thread th1([&]() {

WeightsUpdater(0, int(size[i] / 2), i, lr);

});

thread th2([&]() {

WeightsUpdater(int(size[i] / 2), size[i], i, lr);

});

th1.join();

th2.join();

}

if (threadsNum == 4) {

if (size[i] < 4) {

for (int j = 0; j < size[i]; j++) {

for (int k = 0; k < size[i + 1]; k++) {

weights[i][j][k] += lr \* neurons[i + 1][k].error \* sigm\_pro(neurons[i + 1][k].value) \* neurons[i][j].value;

}

}

}

if (size[i] >= 4) {

int start1 = 0; int stop1 = int(size[i] / 4);

int start2 = int(size[i] / 4); int stop2 = int(size[i] / 2);

int start3 = int(size[i] / 2); int stop3 = int(size[i] - floor(size[i] / 4));

int start4 = int(size[i] - floor(size[i] / 4)); int stop4 = size[i];

thread th1([&]() {

WeightsUpdater(start1, stop1, i, lr);

});

thread th2([&]() {

WeightsUpdater(start2, stop2, i, lr);

});

thread th3([&]() {

WeightsUpdater(start3, stop3, i, lr);

});

thread th4([&]() {

WeightsUpdater(start4, stop4, i, lr);

});

th1.join();

th2.join();

th3.join();

th4.join();

}

}

}

}

bool SaveWeights() {

ofstream fout;

fout.open("weights.txt");

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

if (i < layers - 1) {

for (int j = 0; j < size[i]; j++) {

for (int k = 0; k < size[i + 1]; k++) {

fout << weights[i][j][k] << " ";

}

}

}

}

fout.close();

return 1;

}

};

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(0));

ifstream fin;

ofstream fout;

fout.open("log.txt");

const int l = 4;

const int input\_l = 4096;

int size[l] = { input\_l, 64, 32, 26 };

network nn;

double input[input\_l];

char rresult;

double result;

double ra = 0;

int maxra = 0;

int maxraepoch = 0;

const int n = 56;

bool to\_study = 0;

std::cout << "Производить обучение?";

std::cin >> to\_study;

double time = 0;

data\_one\* data = new data\_one[n];

if (to\_study) {

fin.open("lib.txt");

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

for (int j = 0; j < input\_l; j++) {

fin >> data[i].info[j];

}

fin >> data[i].rresult;

data[i].rresult -= 65;

}

nn.setLayers(l, size);

for (int e = 0; ra / n \* 100 < 100; e++) {

fout << "Epoch # " << e << endl;

double epoch\_start = clock();

ra = 0;

double w\_delta = 0;

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

for (int j = 0; j < 4096; j++) {

input[j] = data[i].info[j];

}

rresult = data[i].rresult;

nn.set\_input(input);

double FF\_start = clock();

result = nn.ForwardFeed();

double FF\_stop = clock();

if (result == rresult) {

cout << "Угадал букву " << char(rresult + 65) << "\t\t\t\*\*\*\*" << endl;

ra++;

}

else {

double BP\_start = clock();

nn.BackPropogation(result, rresult, 1);

double BP\_stop = clock();

}

}

double epoch\_stop = clock();

cout << "Right answers: " << ra / n \* 100 << "% \t Max RA: " << double(maxra) / n \* 100 << "(epoch " << maxraepoch << " )" << endl;

time = 0;

//cout << "W\_Delta: " << w\_delta << endl;

if (ra > maxra) {

maxra = ra;

maxraepoch = e;

}

if (maxraepoch < e - 250) {

maxra = 0;

}

}

//fout.close();

if (nn.SaveWeights()) {

cout << "Веса сохранены!";

}

}

else {

nn.setLayersNotStudy(l, size, "weights.txt");

}

fin.close();

cout << "Начать тест:(1/0) ";

bool to\_start\_test = 0;

cin >> to\_start\_test;

char right\_res;

if (to\_start\_test) {

fin.open("test.txt");

for (int i = 0; i < input\_l; i++) {

fin >> input[i];

}

nn.set\_input(input);

result = nn.ForwardFeed(string("show results"));

cout << "Я считаю, что это буква " << char(result + 65) << "\n\n";

cout << "А какая это буква на самом деле?...";

cin >> right\_res;

if (right\_res != result + 65) {

cout << "Хорошо господин, исправляю ошибку!";

nn.BackPropogation(result, right\_res - 65, 0.15);

nn.SaveWeights();

}

}

return 0;

}

**test\_creator(вспомогательная программа)**

uses GraphABC;

var

p: Picture;

t:text;

c:char;

begin

p:= Picture.Create('image.png');

p.Draw(250,250, 128, 128);

writeln('Какой символ на картинке: ');

read(c);

assign(t, 'test.txt');

//assign(t, 'lib.txt');

t.Append();

for i:integer:= 0 to p.Height-1 do

begin

for j:integer:= 0 to p.Width-1 do

write(t, (1.0 - (p.GetPixel(j,i).R / 255)):0:3, ' ');

writeln(t, '');

end;

writeln(t, c);

t.Close();

writeln('done!');

window.Close();

end.

## Тестирование

Во время тестирования были обнаружены следующие ошибки:

* Нейросеть могла обучаться бесконечно, если количество букв для обучения превышало установленные значения и наоборот.