Файл main.cpp содержащий main()

```
#include "mainwindow.h"
#include "PaintScene.h"
#include "NetWork.h"
#include <QtWidgets/QApplication>

int main(int argc, char *argv[]) {
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Файл NetWork.h содержащий объявления класса NetWork

```
#pragma once
#include <iostream>
#include <fstream>
#include "Matrix.h"
#include "ActivateFunctuon.h"
#include "Sigmoid.h"
#include "ReLU.h"
#include "Thx.h"

// информация о сети
struct data_NetWork {
     int L; // количество слоев
     int* size; // количество слоёв на каждом слое
};

class NetWork {
     int L; // колличество слоёв
     int* size; // количество нейронов на каждом слое
     ActivateFunctuon actFunc; // функция активации
     Matrix* weight; // матрица весов
     double** bios; // веса смещений
     double** neuros_val, ** neuros_err; //значение нейронов,
ошибки для нейронов
     double* neuros_bios_val; // значение нейронов смешения
public:
     NetWork(data_NetWork data);
     ~NetWork();
     void PrintConfig();
     void SetInput(double* values);


     double ForwarFeed();
     int SearchMaxIndex(double* values);
     void PrintValue(int L);

     void BackPropogation(double expert);
     void WeightsUpdater(double lr);
```

```
     void SaveWeights();
     void ReadWeights();
};
```

Файл NetWork.cpp содержащий реализацию класса NetWork

```
#include "NetWork.h"
#include <fstream>
#include <chrono>

NetWork::NetWork(data_NetWork data) {
     srand(time(NULL));
     srand(static_cast<unsigned int>(time(0)));
     L = data.L;
     size = new int[L];
     for (int i = 0; i < L;i++)
          size[i] = data.size[i]; // считываем из структуры

     weight = new Matrix[L - 1]; // место для матрицы весов
     bios = new double* [L - 1]; // для весов смешения
     for (int i = 0; i < L - 1; i++) {
          weight[i].Init(size[i + 1], size[i]);
          bios[i] = new double[size[i + 1]];
          weight[i].Rand(); // веса заполнили случайно

          for (int j = 0; j < size[i + 1];j++)
               bios[i][j] = ((rand() % 50) * 0.06 / (size[i] +
15)); // веса смещения заполнили случайно
     }
     neuros_val = new double* [L]; // место для нейронов
     neuros_err = new double* [L]; //
     for (int i = 0; i < L; i++) {
          neuros_val[i] = new double[size[i]];
          neuros_err[i] = new double[size[i]];
     }
     neuros_bios_val = new double[L];
     for (int i = 0; i < L - 1; i++)
          neuros_bios_val[i] = 1;
}

NetWork::~NetWork() {
     for (int i = 0; i < L - 1;i++) {
          delete bios[i];
     }
     delete bios;
     delete weight;

     for (int i = 0;i < L; i++) {
          delete neuros_val[i];
          delete neuros_err[i];
     }
     delete neuros_val;
     delete neuros_err;
```

```
     delete neuros_bios_val;
}

// вывод информации о сети
void NetWork::PrintConfig() {
     std::cout << "_____\n";
     std::cout << "Network has " << L << " layers\nSIZE[]: ";
     for (int i = 0; i < L; i++)
          std::cout << size[i] << " ";
     std::cout <<
"\n_____\n\n";
}

// подаются данные для нейронов(вектора значений)
void NetWork::SetInput(double* values) {
     for (int i = 0; i < size[0]; i++)
          neuros_val[0][i] = values[i];
}

// функция прямого распространения
double NetWork::ForwarFeed() {
     for (int i = 1; i < L; i++) {
          Matrix::Multi(weight[i - 1], neuros_val[i - 1], size[i
- 1], neuros_val[i]); // матрицу весов на вектор столбец
нейронов(матрица весов, значение нейронов, количество нейронов,
желаемый результат)
          Matrix::SumVector(neuros_val[i], bios[i - 1], size[i]);
// суммируем значение с весами смещения
          actFunc.use(neuros_val[i], size[i]);
// используем функцию активации для нейронов
     }
     return SearchMaxIndex(neuros_val[L - 1]); // ищеи MAX
     //int pred = SearchMaxIndex(neuros_val[L - 1]); // ищеи MAX
     //return pred;
}

// возращает индекс MAX из вектора значений(вектор значений)
int NetWork::SearchMaxIndex(double* value) {
     double max = value[0];
     int prediction = 0;
     double temp;
     for (int i = 1; i < size[L - 1]; i++) {
          temp = value[i];
          if (temp > max) {
               prediction = i;
               max = temp;
          }
     }
     return prediction;
}

// выводит индекс и значение нейрона на экран на слое (слой)
void NetWork::PrintValue(int L) {
```

```
     for (int i = 0; i < size[L]; i++)
          std::cout << i << neuros_val[L][i] << std::endl;
}

// считаем дельту(правильный ответ)
void NetWork::BackPropogation(double expert) {
     for (int i = 0; i < size[L - 1]; i++) {
          if (i != int(expert))
               neuros_err[L - 1][i] = -neuros_val[L - 1][i] *
actFunc.useDer(neuros_val[L - 1][i]);
          else
               neuros_err[L - 1][i] = (1.0 - neuros_val[L -
1][i]) * actFunc.useDer(neuros_val[L - 1][i]);
     }
     for (int i = L - 2; i > 0; i--) {
          Matrix::Multi_T(weight[i], neuros_err[i + 1], size[i +
1], neuros_err[i]);
          for (int j = 0; j < size[i]; j++)
               neuros_err[i][j] *=
actFunc.useDer(neuros_val[i][j]);
     }
}

// обновление весов
void NetWork::WeightsUpdater(double lr) {
     for (int i = 0; i < L - 1; i++)
          for (int j = 0; j < size[i + 1]; j++)
               for (int k = 0; k < size[i]; k++)
                    weight[i](j, k) += neuros_val[i][k] *
neuros_err[i + 1][j] * lr; // должен быть +
     for (int i = 0; i < L - 1; i++)
          for (int j = 0; j < size[i + 1]; j++)
               bios[i][j] += neuros_err[i + 1][j] * lr;
}

// сохраняем веса
void NetWork::SaveWeights() {
     std::ofstream fout;
     fout.open("Weight.txt");
     if (!fout.is_open()) {
          std::cout << "Error reading the file";
          system("pause");
     }

     for (int i = 0; i < L - 1; i++)
          fout << weight[i] << " ";

     for (int i = 0; i < L - 1; i++)
          for (int j = 0; j < size[i + 1]; j++)
               fout << bios[i][j] << " ";

     std::cout << "Weghts saved\n";
     fout.close();
```

```
}

// считываем веса
void NetWork::ReadWeights() {
     std::ifstream fin;
     fin.open("Weight.txt");
     if (!fin.is_open()) {
          std::cout << "Error reading the file";
          system("pause");
     }

     for (int i = 0; i < L - 1; i++)
          fin >> weight[i];

     for (int i = 0; i < L - 1; i++)
          for (int j = 0; j < size[i + 1]; j++)
               fin >> bios[i][j];

     std::cout << "Weghts readed\n";
     fin.close();
}
```

Файл Matrix.h содержащий объявления класса Matrix

```
#pragma once
#include <iostream>

class Matrix{
     double** matrix;
     int row, col;
public:
     Matrix();
     ~Matrix();
     void Init(int row, int col);
     void Rand();
     static void Multi(const Matrix& m, const double* b, int n,
double* c);
     static void Multi_T(const Matrix& m, const double* b, int n,
double* c);
     static void SumVector(double* a, const double* b, int n);

     double& operator ()(int i, int j);
     friend std::ostream& operator << (std::ostream& os, const
Matrix& m);
     friend std::istream& operator >> (std::istream& is, Matrix&
m);
};
```

Файл Matrix.cpp содержащий реализацию класса Matrix

```
#include "Matrix.h"

Matrix::Matrix() {
```

```
}

Matrix::~Matrix() {
     for (int i = 0; i < row; i++)
          delete matrix[i];

     delete matrix;
}

// инициализация матрицы(строки, столбцы)
void Matrix::Init(int row, int col) {
     this->row = row;
     this->col = col;
     matrix = new double* [row];
     for (int i = 0; i < row; i++)
          matrix[i] = new double[col];

     for (int i = 0; i < row; i++)
          for (int j = 0; j < col; j++)
               matrix[i][j] = 0;
}

// заполнение случайными числами
void Matrix::Rand() {
     srand(static_cast<unsigned int>(time(0)));
     for (int i = 0; i < row; i++)
          for (int j = 0; j < col; j++)
               matrix[i][j] = ((rand() % 100)) * 0.03 / (row +
35);
}

// умножение на вектор-столбец(матрица, вектор-столбец, размер
вектор столбца, результат??)
void Matrix::Multi(const Matrix& m1, const double* neuron, int
n, double* c) {
     if (m1.col != n)
          throw std::runtime_error("Error Multi\n");

     for (int i = 0; i < m1.row; i++) {
          double temp = 0;
          for (int j = 0; j < m1.col; j++)
               temp += m1.matrix[i][j] * neuron[j];
          c[i] = temp;
     }
}

// умножение на транспонированную матрицу(матрица, вектор-
столбец, размер вектор столбца, результат??)
void Matrix::Multi_T(const Matrix& m1, const double* neuron, int
n, double* c) {
     if (m1.row != n)
          throw std::runtime_error("Error Multi\n");
```

```
     for (int i = 0; i < m1.col; i++) {
          double temp = 0;
          for (int j = 0; j < m1.row; j++)
               temp += m1.matrix[j][i] * neuron[j];
          c[i] = temp;
     }
}

// сложение векторов(вектор(+результат), вектор, размер)
 void Matrix::SumVector(double* a, const double* b, int n){
     for (int i = 0; i < n; i++)
          a[i] += b[i];
}

 // возращает элемент матрицы
double& Matrix::operator()(int i, int j) {
     return matrix[i][j];
}

std::ostream& operator << (std::ostream& os, const Matrix& m) {
     for (int i = 0; i < m.row; i++)
          for (int j = 0; j < m.col; j++)
               os << m.matrix[i][j] << " ";
     return os;
}

std::istream& operator >> (std::istream& is, Matrix& m) {
     for (int i = 0; i < m.row; i++)
          for (int j = 0; j < m.col; j++)
               is >> m.matrix[i][j];
     return is;
}
```

Файл ActivateFunctuon.h содержащий объявления класса ActivateFunctuon

```
#pragma once
#include <iostream>
#include "Sigmoid.h"
#include "ReLU.h"
#include "Thx.h"

enum activateFunc { sigmoid = 1, reLU, thx };

class ActivateFunctuon:public Sigmoid, public ReLU, public Thx{
     activateFunc actFunc;
public:
     ActivateFunctuon();

     void set();
     void use(double* value, int n);
     void useDer(double* value, int n);
     double useDer(double value);
};
```

Файл ActivateFunctuon.cpp содержащий реализацию класса ActivateFunctuon

```
#include "ActivateFunctuon.h"

ActivateFunctuon::ActivateFunctuon() {
     actFunc = sigmoid;
}

void ActivateFunctuon::set() {
     std::cout << "Set actFunc pls\n1 - Sigmoid \n2 - ReLU \n3 -
Th(x) \n";
     int tmp;
     std::cin >> tmp;
     switch (tmp)
     {
     case sigmoid:
          actFunc = sigmoid;
          break;
     case reLU:
          actFunc = reLU;
          break;
     case thx:
          actFunc = thx;
          break;
     default:
          throw std::runtime_error("Error read actFunc");
          break;
     }
}

// применение функции активации(вектор значений, размер)
void ActivateFunctuon::use(double* value, int n) {
     switch (actFunc)
     {
     case sigmoid:
          Sigmoid::use(value, n);
          break;
     case reLU:
          ReLU::use(value, n);
          break;
     case thx:
          Thx::use(value, n);
          break;
     default:
          throw std::runtime_error("Error actFunc \n");
          break;
     }
}

// применение производной  функции активации(вектор значений,
размер)
void ActivateFunctuon::useDer(double* value, int n) {
     switch (actFunc)
```

```
     {
          switch (actFunc)
          {
          case sigmoid:
               Sigmoid::useDer(value, n);
               break;
          case reLU:
               ReLU::useDer(value, n);
               break;
          case thx:
               Thx::useDer(value, n);
               break;
          default:
               throw std::runtime_error("Error actFunc \n");
               break;
          }
     default:
          throw std::runtime_error("Error actFuncDer \n");
          break;
     }
}

// просто перегруженная функция
double ActivateFunctuon::useDer(double value) {
     switch (actFunc)
     {
          switch (actFunc)
          {
          case sigmoid:
               Sigmoid::useDer(value);
               break;
          case reLU:
               ReLU::useDer(value);
               break;
          case thx:
               Thx::useDer(value);
               break;
          default:
               throw std::runtime_error("Error actFunc \n");
               break;
          }
     }
     return value;
}
```

Файл Sigmoid.h содержащий объявления класса Sigmoid

```
#pragma once
#include <cmath>

class Sigmoid
{
public:
     void use(double* value, int n);
```

```
     void useDer(double* value, int n);
     double useDer(double value);
};
```

Файл Sigmoid.cpp содержащий реализацию класса Sigmoid

```
#include "Sigmoid.h"
#include <iostream>

void Sigmoid::use(double* value, int n) {
     for (int i = 0;i < n;i++)
          value[i] = 1 / (1 + exp(-value[i]));
}

void Sigmoid::useDer(double* value, int n) {
     for (int i = 0;i < n;i++)
          value[i] *= (1 + -value[i]);
}


double Sigmoid::useDer(double value) {
     return value = 1 / (1 + exp(-value));
}
```

Файл ReLU.h содержащий объявления класса ReLU

```
#pragma once
class ReLU
{
public:
     void use(double* value, int n);
     void useDer(double* value, int n);
     double useDer(double value);
};
```

Файл ReLU.cpp содержащий реализацию класса ReLU

```
#include "ReLU.h"

void ReLU::use(double* value, int n) {
     for (int i = 0;i < n;i++) {
          if (value[i] < 0)
               value[i] *= 0.01;
          else if (value[i] > 1)
               value[i] = 1. + 0.01 * (value[i] - 1.);
     }
}

void ReLU::useDer(double* value, int n) {
     for (int i = 0; i < n; i++) {
          if (value[i] < 0 || value[i] > 1)
               value[i] = 0.01;
          else
               value[i] = 1;
     }
```

```
}

double ReLU::useDer(double value) {
     if (value < 0 || value > 1)
          value = 0.01;
     return value;
}
```

Файл Thx.h содержащий объявления класса Thx

```
#pragma once
#include <cmath>

class Thx
{
public:
     void use(double* value, int n);
     void useDer(double* value, int n);
     double useDer(double value);
};
```

Файл Thx.cpp содержащий реализацию класса Thx

```
#include "Thx.h"
#include <iostream>

void Thx::use(double* value, int n) {
     for (int i = 0; i < n; i++) {
          if (value[i] < 0)
               value[i] = 0.01 * (exp(value[i]) - exp(-value[i]))
/ (exp(value[i]) + exp(-value[i]));
          else
               value[i] = (exp(value[i]) - exp(-value[i])) /
(exp(value[i]) + exp(-value[i]));
     }
}


void Thx::useDer(double* value, int n) {
     for (int i = 0; i < n; i++) {
          if (value[i] < 0)
               value[i] = 0.01 * (1 - value[i] * value[i]);
          else
               value[i] = 1 - value[i] * value[i];
     }
}

double Thx::useDer(double value) {
     if (value < 0)
          value = 0.01 * (exp(value) - exp(-value)) / (exp(value)
+ exp(-value));
     else
          value = (exp(value) - exp(-value)) / (exp(value) +
exp(-value));
```

```
     return value;
}
```

Файл MainWindow.h содержащий объявления класса MainWindow

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "ui_mainwindow.h"
#include "PaintScene.h"
#include "NetWork.h"


QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT
    PaintScene* PS;
    NetWork NW;
    data_NetWork NW_config;

public:
    MainWindow(QWidget *parent = nullptr);
    data_NetWork ReadDataNetWork(std::string path);
    void ReadTest(double* input, int input_n);

signals:
    void endWriteToFile();
public slots:
    void StartGuess();
    void clearLCD();
private:
    Ui::MainWindow *ui;
};
#endif // MAINWINDOW_H
```

Файл MainWindow.cpp содержащий реализацию класса MainWindow

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"

data_NetWork MainWindow::ReadDataNetWork(std::string path) {
    data_NetWork data{};
    std::ifstream fin;
    fin.open(path);
    if (!fin.is_open()) {
        std::cout << "Error reading the file";
        system("pause");
    }
```

```
    else
        std::cout << path << " loading...\n";
    std::string tmp;
    int n;
    while (!fin.eof()) {
        fin >> tmp;
        if (tmp == "NetWork") {
            fin >> n;
            data.L = n;
            data.size = new int[n];
            for (int i = 0; i < n; i++) {
                fin >> data.size[i];
            }
        }
    }
    fin.close();
    return data;
}


MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    //, ui(new Ui::MainWindow)
{
    ui->setupUi(this);
    setWindowTitle("Мяу - мяу :");
    setWindowIcon(QIcon("D:/Proga/Qt/happy.png"));
    QWidget* MyWidget = new QWidget(this);

    MyWidget->setLayout(ui->gridLayout_2);//ui.gridLayout_2
    setCentralWidget(MyWidget);
    resize(500, 380);
    NW_config = ReadDataNetWork("D:/Proga/Qt/Config.txt");
    NW.Init(NW_config);
    NW.ReadWeights();
    ui->lcdNumber->display("");
}

void MainWindow::clearLCD() {
    ui->lcdNumber->display("");
}

void MainWindow::ReadTest(double* input, int input_n) {

    QFile mFile("D:/Proga/Qt/test.txt");
    mFile.open(QIODevice::ReadOnly);
    QTextStream fin(&mFile);
    if (mFile.isOpen())
        for (int j = 0; j < input_n; j++)
            fin >> input[j];
    else QMessageBox::information(0, "Error", "Error reading the
file");
    mFile.close();
```

```
}

void MainWindow::StartGuess() {
    //double* input = new double[NW_config.size[0]];
    double* input = new double[784];
    //ReadTest(input, NW_config.size[0]);
    ReadTest(input, 784);
    NW.SetInput(input);
    double digit = NW.ForwarFeed();
    ui->lcdNumber->display(QString().setNum(digit));
}
```

Файл PaintScene.h содержащий объявления класса PaintScene

```
#pragma once

#include <QWidget>
#include "ui_PaintScene.h"
#include <QtWidgets>
#include <QPainter>

class PaintScene : public QWidget {
      Q_OBJECT
      bool draw;
      QVector <QPointF> vv;
      QImage pic;
public:
    PaintScene(QWidget *parent = nullptr);
      void paintEvent(QPaintEvent*);
      void mousePressEvent(QMouseEvent*);
      void mouseMoveEvent(QMouseEvent*);
      void mouseReleaseEvent(QMouseEvent*);
public slots:
      void clear();
private:
      Ui::PaintScene ui;
};
```

Файл PaintScene.cpp содержащий реализацию класса PaintScene

```
#include "PaintScene.h"
#include <QWidget>
#include <QtWidgets>
#include <QPainter>
#include <iostream>

PaintScene::PaintScene(QWidget *parent)
    : QWidget(parent)
{
      ui.setupUi(this);
    setFocusPolicy(Qt::StrongFocus);
      draw = false;
}
```

```
void PaintScene::paintEvent(QPaintEvent*) {
    QPainter painter(this);
      QPalette Pal(palette());
    Pal.setColor(QPalette::Background, Qt::black);
    setAutoFillBackground(true);
    setPalette(Pal);
    painter.setRenderHint(QPainter::Antialiasing, true);
    painter.setPen(QPen(Qt::white, 12, Qt::SolidLine));
    for (int i = 0; i < vv.size(); i++)
        if (i > 0)
            painter.drawEllipse(vv[i - 1], 15, 15);
}

void PaintScene::mousePressEvent(QMouseEvent* pe) {
      draw = true;
}

void PaintScene::mouseMoveEvent(QMouseEvent* pe) {
      if (draw) {
        //std::cout<<"dr"<<std::endl;
            vv.push_back(pe->pos());
      }
      update();
}
void PaintScene::mouseReleaseEvent(QMouseEvent* pe) {
    std::cout<<"pic"<<std::endl;
      draw = false;
      QPixmap scr;
      scr = QPixmap::grabWidget(this);

    QFile mFile("D:/Proga/Qt/test.txt");
      mFile.open(QIODevice::WriteOnly);
      QTextStream fin(&mFile);

    std::cout<<"pic2"<<std::endl;
      QImage img;
      img = scr.toImage();
      img = img.scaled(28, 28);
      int w = img.size().width();
      int h = img.size().height();
      for (int i = 0; i < w; i++) {
            for (int j = 0; j < h; j++) {
            double tm = img.pixelColor(j, i).blue() / 255.;
            if(tm!=0)
                tm=1;
            fin << tm << " ";
             }
             fin << endl;
      }


      mFile.close();
}
```

```
void PaintScene::clear() {
      vv.clear();
    update();
}
```

Прочие функции

```
// считывае инфу про сеть (путь)
data_NetWork ReadDataNetWork(std::string path) {
    data_NetWork data{};
    std::ifstream fin;
    fin.open(path);
    if (!fin.is_open()) {
        std::cout << "Error reading the file " << path <<
std::endl;
        system("pause");
    }
    else
        std::cout << path << " loading...\n";
    std::string temp;
    int L;
    while (!fin.eof()) {
        fin >> temp;
        if (temp == "NetWork") {
            fin >> L;
            data.L = L;
            data.size = new int[L];
            for (int i = 0; i < L; i++) {
                fin >> data.size[i];
            }
        }
    }
    fin.close();
    return data;
}

// считываем инфу для обучения (путь, инфа о сети, колличество
примеров)
data_info* ReadData(std::string path, const data_NetWork&
data_NW, int& examples) {
    data_info* data;
    std::ifstream fin;
    fin.open(path);
    if (!fin.is_open()) {
        std::cout << "Error reading the file " << path <<
std::endl;
        system("pause");
    }
    else
        std::cout << path << " loading...\n";

    std::string temp;
```

```
    fin >> temp;
    if (temp == "Examples") {
        fin >> examples;
        std::cout << "Examples: " << examples << std::endl;
        data = new data_info[examples];
        for (int i = 0; i < examples; i++)
            data[i].pixels = new double[data_NW.size[0]];

        for (int i = 0; i < examples; i++) {
            fin >> data[i].digit;
            for (int j = 0; j < data_NW.size[0]; j++)
                fin >> data[i].pixels[j];
        }
        fin.close();
        //NewBase
        std::cout << "NewBase... \n";
        //std::cout << "lib_MNIST loaded... \n";
        return data;
    }
    else {
        std::cout << "Error loading: " << path << std::endl;
        fin.close();
        return nullptr;
    }
}

void Training(NetWork NW,const data_NetWork& NW_config){
    int epoch = 0;
    double right_answer = 0, right, predict, max_right_answer =
0;
    data_info* data;
    bool weights;
    std::chrono::duration<double> time;

    std::cout << "Read Weights? (1/0)" << std::endl;
    std::cin >> weights;
    if (weights)
        NW.ReadWeights();

    int examples;
    data = ReadData("NewBase.txt", NW_config, examples);
    auto begin = std::chrono::steady_clock::now();
    for(;epoch < 20; epoch++){
        right_answer = 0;
        auto t1 = std::chrono::steady_clock::now();
        for (int i = 0; i < examples; i++) {
            NW.SetInput(data[i].pixels);
            right = data[i].digit;
            predict = NW.ForwarFeed();
            if (predict != right) {
                NW.BackPropogation(right);
                NW.WeightsUpdater(0.1);
            }
```

```
            else
                right_answer++;
        }
        auto t2 = std::chrono::steady_clock::now();
        time = t2 - t1;
        if (right_answer > max_right_answer) max_right_answer =
right_answer;
        std::cout << "right answer: " << right_answer / examples
* 100 << "\t" << "max right answer: " << max_right_answer /
examples * 100 << "\t" << "epoch: " << epoch << "\tTIME: " <<
time.count() << std::endl;
    }
    auto end = std::chrono::steady_clock::now();
    time = end - begin;
    std::cout << "TIME: " << time.count() / 60. << " min" <<
std::endl;
    NW.SaveWeights();
}

void Test(NetWork NW, const data_NetWork& NW_config) {
    double right_answer = 0, right, predict;
    int ex_tests;
    data_info* data_test;
    data_test = ReadData("NewBaseTEST.txt", NW_config,
ex_tests);
    right_answer = 0;
    for (int i = 0; i < ex_tests; i++) {
        NW.SetInput(data_test[i].pixels);
        predict = NW.ForwarFeed();
        right = data_test[i].digit;
        if (right == predict)
            right_answer++;
    }
    std::cout << "right answer: " << right_answer / ex_tests *
100 << std::endl;
}
```