Шуруха Артем ИКБО-10-23

**Краткое объяснение работы нейронной сети.**

1. Обучение:
   1. Считывание файла с обучающими примерами
   2. На входе 784 нейрона (размер изображения одного примера)
   3. Заполняем матрицы весов весов и свободных нейронов (нейронов смещения)
   4. С помощью функции прямого распространения и функции поиска максимума ищем максимальное значение среди выходных нейронов
   5. В случае, если ответ неправильный, обучение продолжается, заново запускается функция прямого распространения
   6. Продолжаем обучение в течение указанного количества эпох
2. Тестирование:
   1. Считывание файла с тестовыми примерами
   2. Применение функции прямого распространения и вычисление максимума среди выходных нейронов (на вход также 784 нейрона)
   3. Сравнение полученного значения с правильным ответом
3. Применение к своим значениям:
   1. Открытие фото с написанной цифрой
   2. Преобразование фото в вид, при котором каждый пиксель является числом в диапазоне [0; 1]
   3. Дописывание примера к обучающим

**Математическое введение.**

Перед демонстрацией кода обращу внимание на важность выбора функции активации. Возьмём три самые популярные вариации: выпрямленную линейную (ReLu), тангенс (thx) и сигмоиду. Написав для всех трёх код, сравнил показатели:

|  |  |
| --- | --- |
|  | Сигмоида  График функции активации:    Видно, что график держится в нужном диапазоне от [0, 1]. |

|  |  |
| --- | --- |
|  | ReLu  График функции активации:    График стремится к бесконечности, поэтому необходимо добавить *границу сверху*, чтобы привести к виду [0, 1]. |

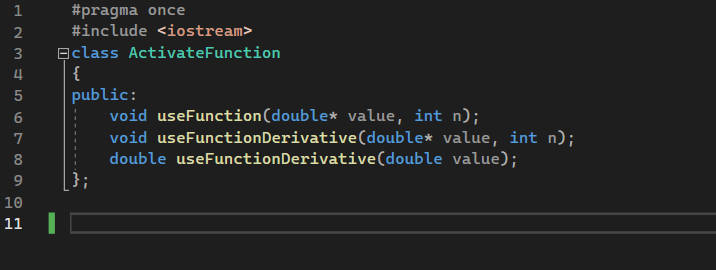
|  |  |
| --- | --- |
|  | Тангенс    Данную функцию нужно *ограничить снизу*, чтобы привести в правильный вид. Но итоговые значения времени обучения слишком велики, поэтому процесс продолжать я не стал. |

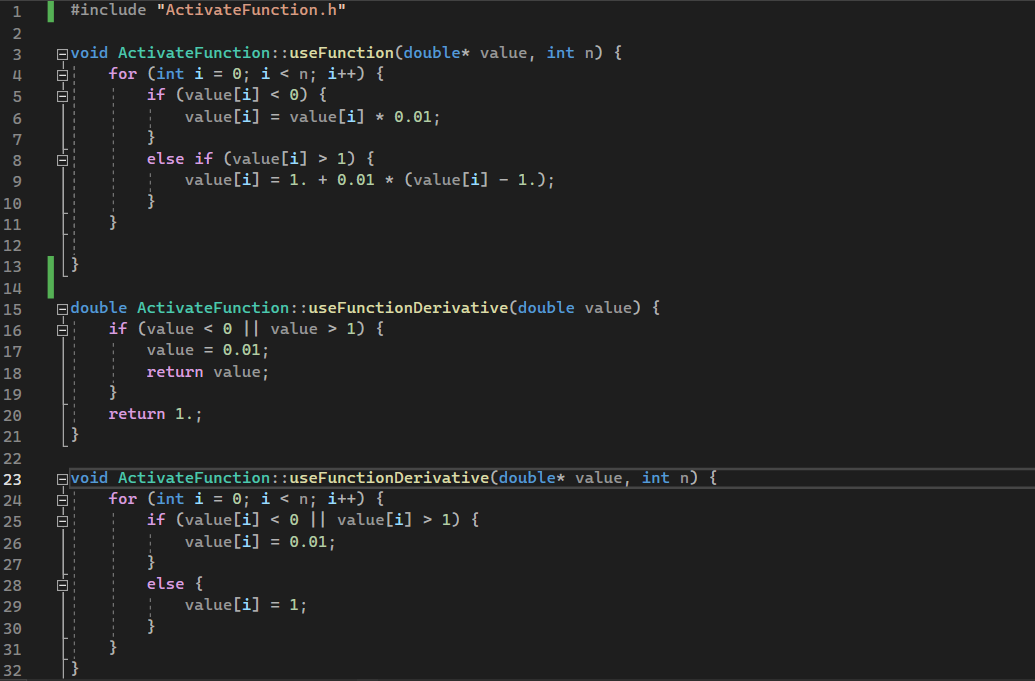
Сравнив показатели, я решил выбрать сигмоиду. Далее в коде будет применяться только она. Стоит сказать, что приведённый выше анализ был обязателен, так как оптимальный выбор функции активации может зависеть от многих факторов, индивидуальных для каждой выборки.

**Почему так произошло?**Функция активации определяет, какой тип нелинейности будет встроен в нейронную сеть. Это позволяет сети выражать сложные нелинейные зависимости в данных. Различные функции активации могут быть более или менее подходящими для определенных типов данных или задач. Выигрыш ReLu объясняет следующие строки из [данной статьи](https://www.g2.com/articles/activation-function): «*Функция ReLU обладает одной из самых высоких показателей точности и используется во всех новейших классификаторах, таких как CNN, R-CNN,* ***многослойных персептронах****, самоорганизующихся картах и генеративных состязательных сетях*». Многослойныq персептрон – это и есть тип нашей нейронной сети.

**Пишем код для функции активации.**

Создадим заголовочный файл и файл cpp расширения для описания функции активации.



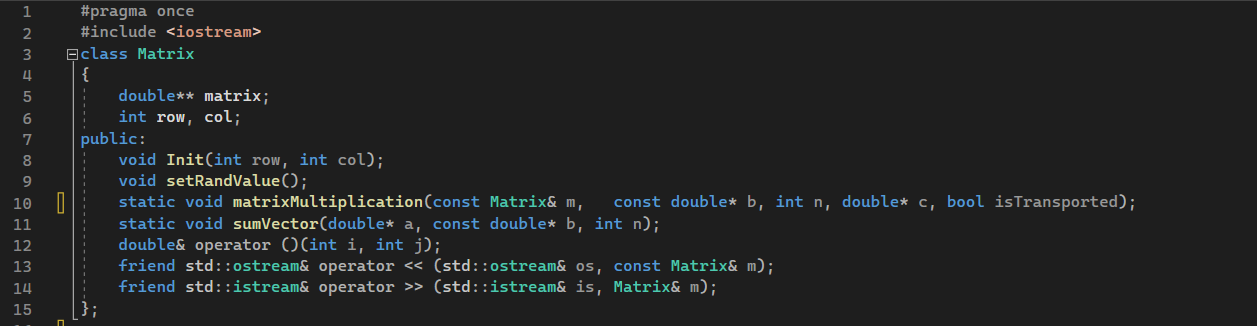


1. useFunction – там записана сама выпрямленная линейная функция активации (ReLU).
2. useFunctionDerivative – там записана производная от функции активации. Производная нужна для последующего вычисления *градиента*. Void-функция нужна для передачи массива с последующим перебором, double (перегрузка) – для числа с плавающей точкой.

**Про датасет.**

База данных образцов рукописного написания цифр **MNIST** содержит 60 000 образцов наборов данных для обучения и тестовый набор из 10 000 образцов (об ограниченности датасета будет в конце). Данные изначально хранятся в бинарном виде, но, так как задача распознавания цифр не является новой, в сети хранятся уже преобразованные в нужный вид файлы (нормированный вид), т.е. каждый пиксель преобразован в double число в диапазоне [0, 1].

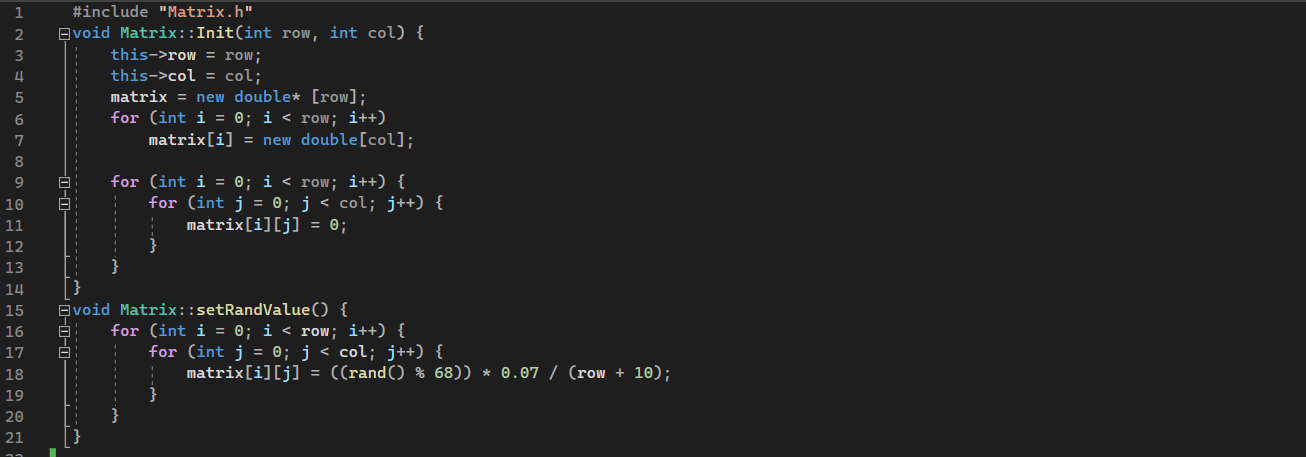
**Класс Matrix.**

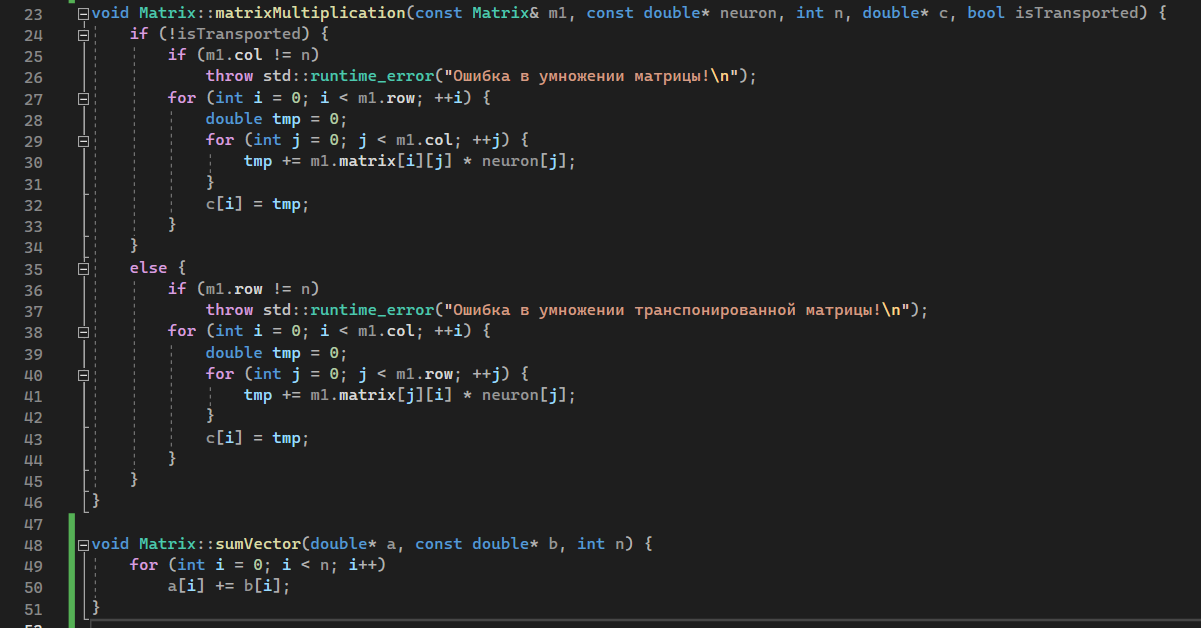


Разберём заголовочный файл Matrix.h:

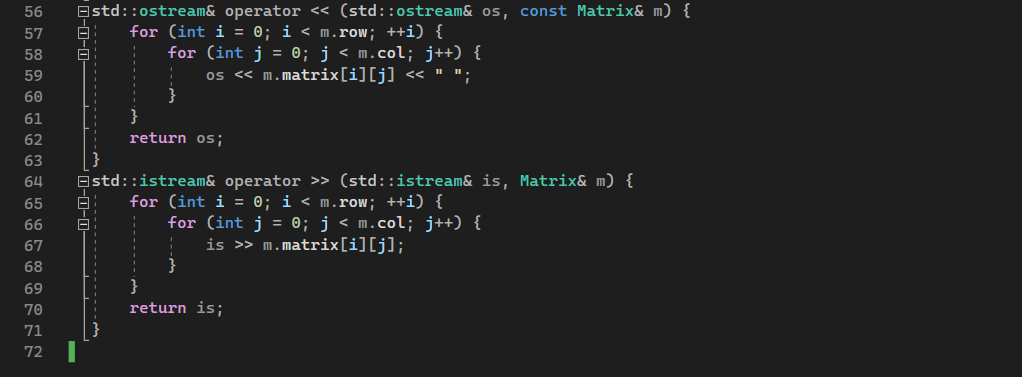
* Init отвечает за создание двумерного массива (выделение памяти)
* matrixMultiplication реализовывает умножение матрицы на вектор-столбец (можно передавать и в транспонированном виде, передав в isTransported true).
* sumVector отвечает за сложение соответственных элементов в векторах
* Последние две функции нужны только для удобства. Можно обойтись без них

Примечание: *setRandValue заполняет массив случайными элементами. Саму «формулу случайности» можно выбрать самому. Значения, получаемые от неё, нужны только на первом шаге, ведь дальше они будут редактироваться в зависимости от правильности ответов при обучении»*.





На данных картинках видно, как работает умножение обычной и транспонированной матрицы на вектор-столбец.

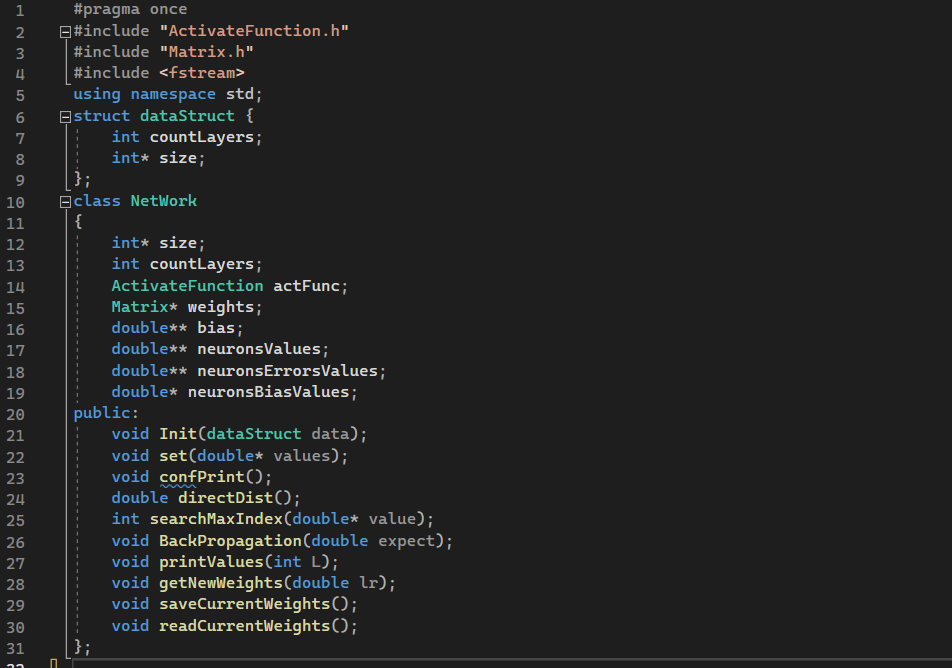


Последние две функции являются служебными и служат для того, чтобы перегрузить потоки ввода и вывода. Возвращается ссылка на поток.

Теперь перейдём к классу NetWork.

**Основной класс NetWork.**

Содержание заголовочного файла:

****

* countLayers – количество слоёв нейронной сети
* size – «хранилище» для нейронов каждого слоя
* actFunc – активационная функция

Далее идёт группа ссылок на ссылки:

* + weights – массива с весами
  + bias – весов смещения
  + neuronsValues – значений нейронов
  + neuronsValuesErrors – значений ошибок нейронов
* neuronsBiasValues – значений нейронов смещения

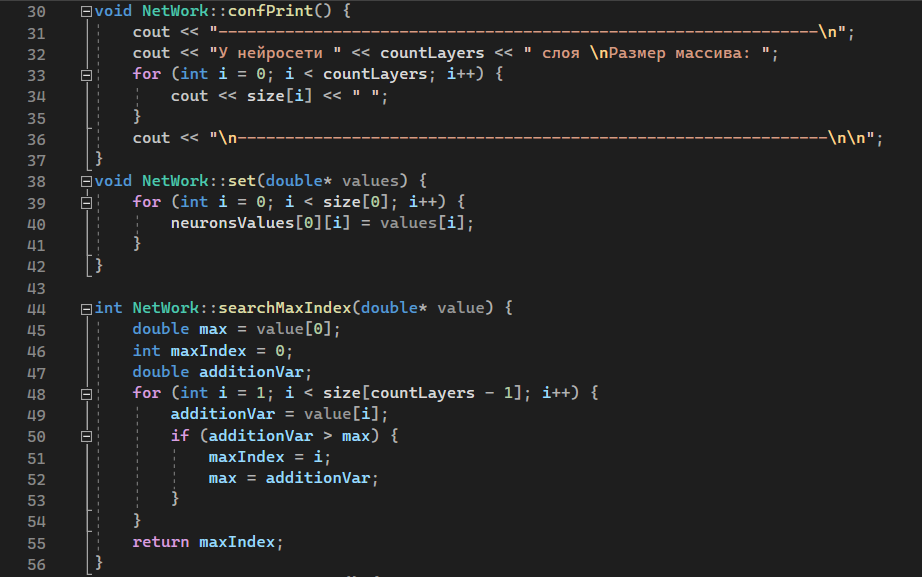
Объявляем функции:

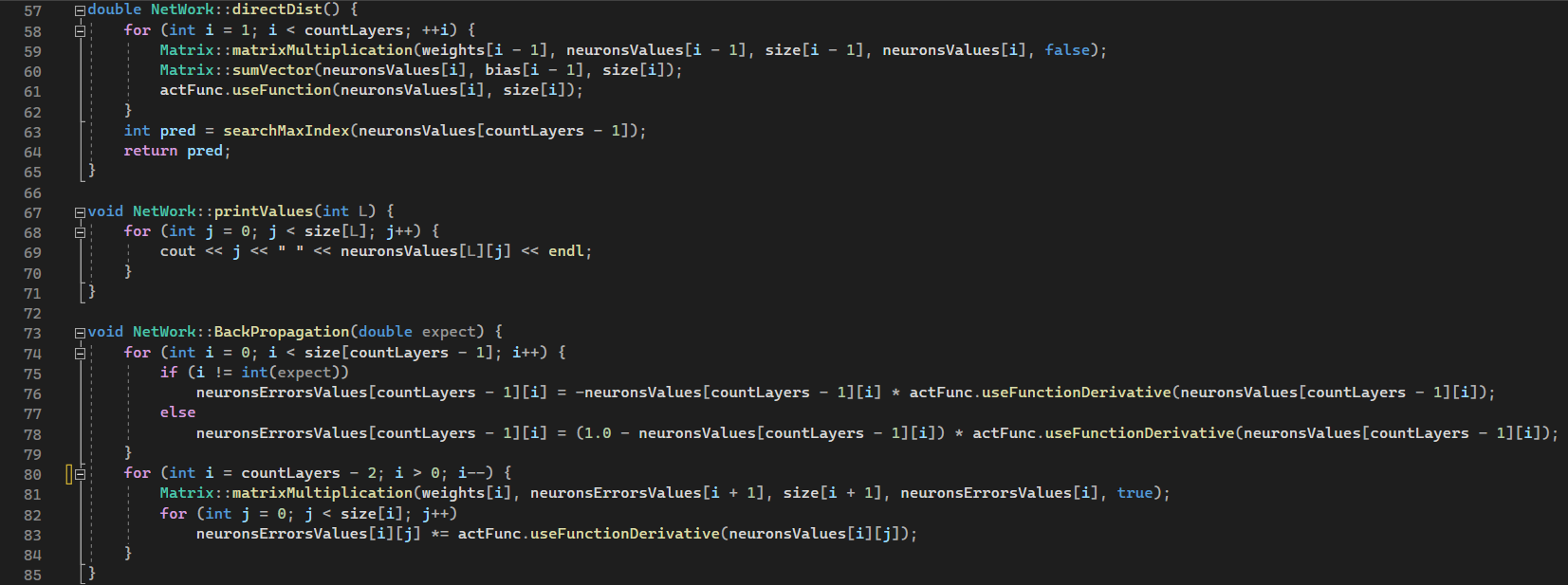
* Init – инициализация/выделение памяти под нейронную сеть
* Set – подача на вход нейросети данных
* confPrint – вывод текста из конфигурационного файла (хранятся сведения о слоях, размерах и т.д.)
* directDist – *функция прямого распространения*
* searchMaxIndex – поиск индекса максимального из переданных значений
* BackPropogation – *функция обратного распространения*
* printValues – вывод значений на экран
* getCurrentWeights – перезапись значений весов
* saveCurrentWeights – сохранение текущих значений весов
* readCurrentWeights – считываний весов из файла (веса хранятся в отдельно txt документе)

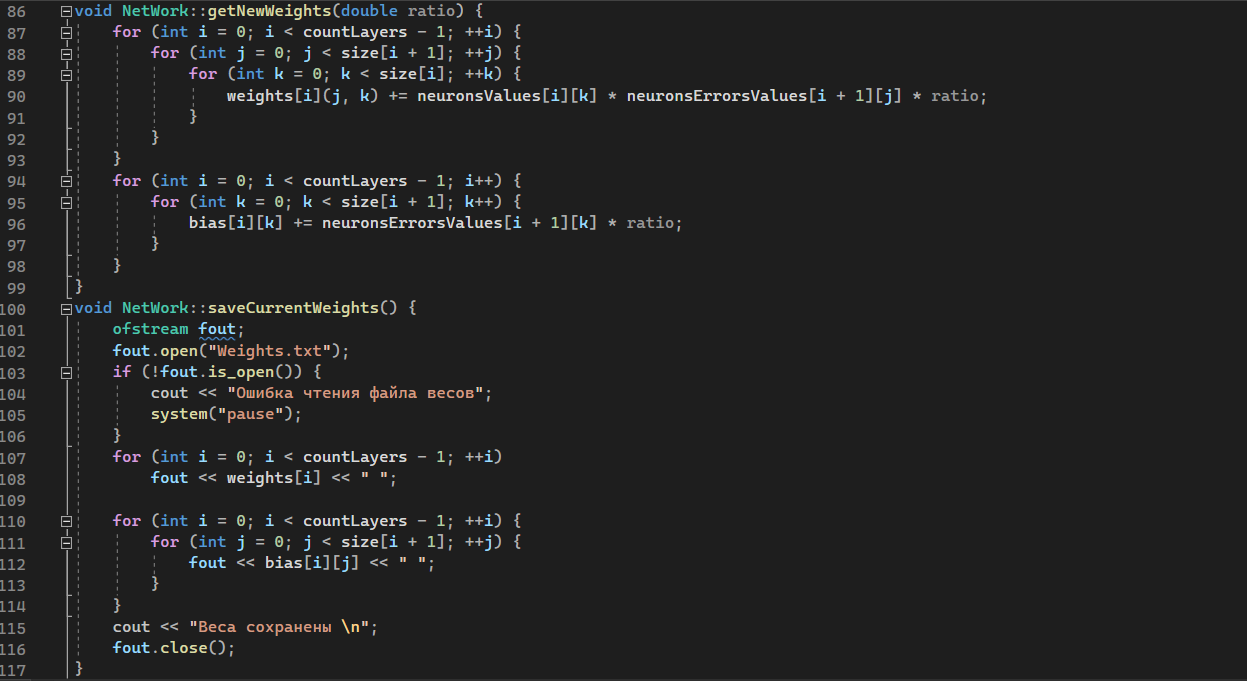
Cpp файл:

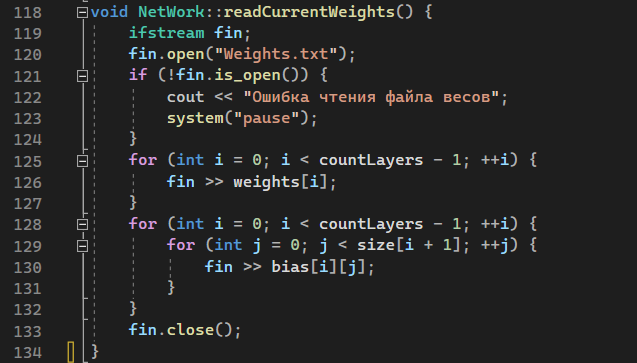


Примечение: *17 строка содержит в себе формулу для случайных чисел, её можно менять*









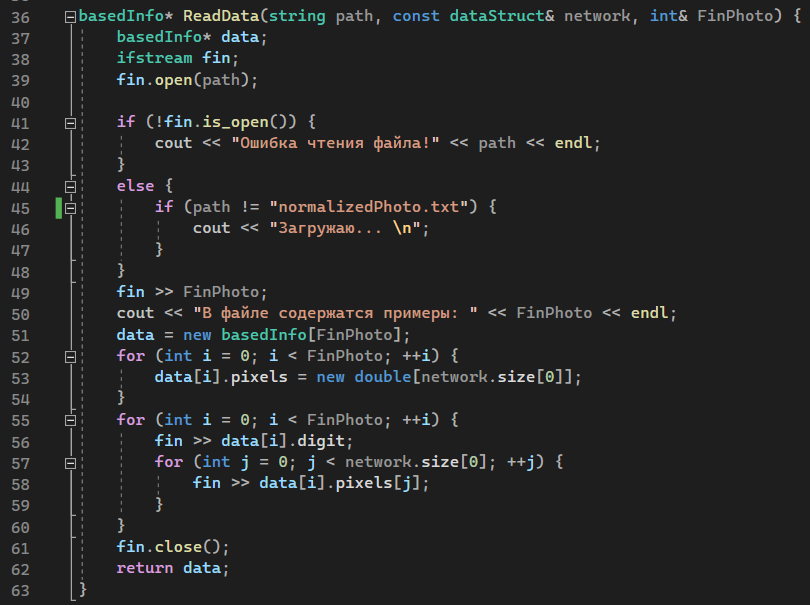
**Поговорим о файле *source.***

Он соединяет в себе работу всех вышеописанных файлов, к тому же обрабатывает пользовательские запросы.

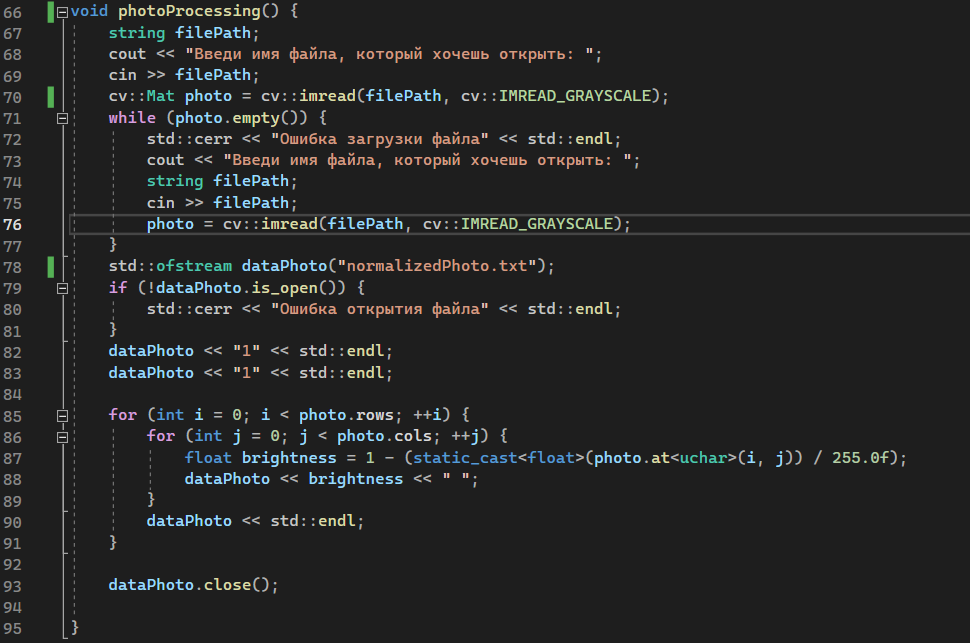


Библиотека «opencv2» нужна для распознавания картинок и преобразования их к необходимому виду. Отмечу, библиотеку сначала нужно установить, а затем уже использовать.

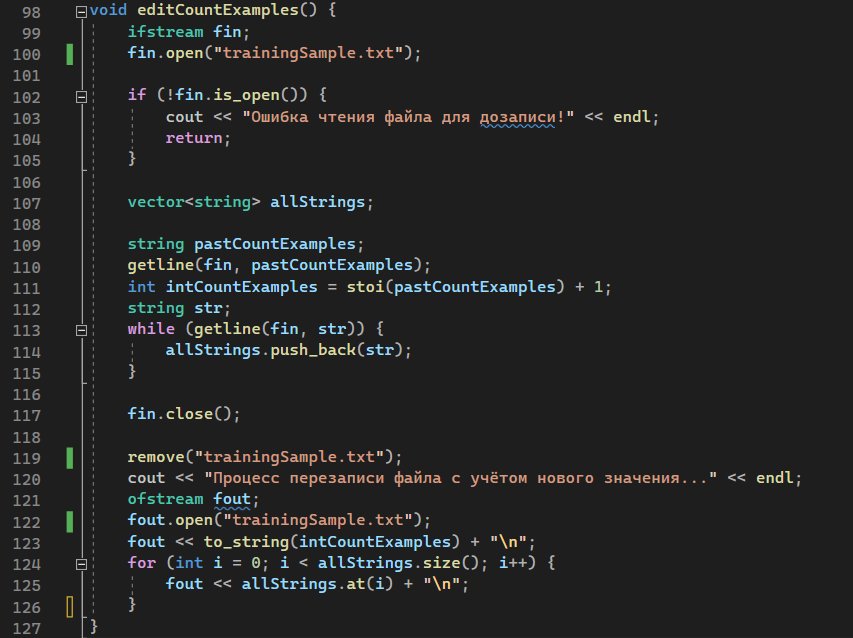
* Функция ReadDataNetWork нужен для чтения данных о переменных конфигурации (количество слоёв, размеры массива и так далее).

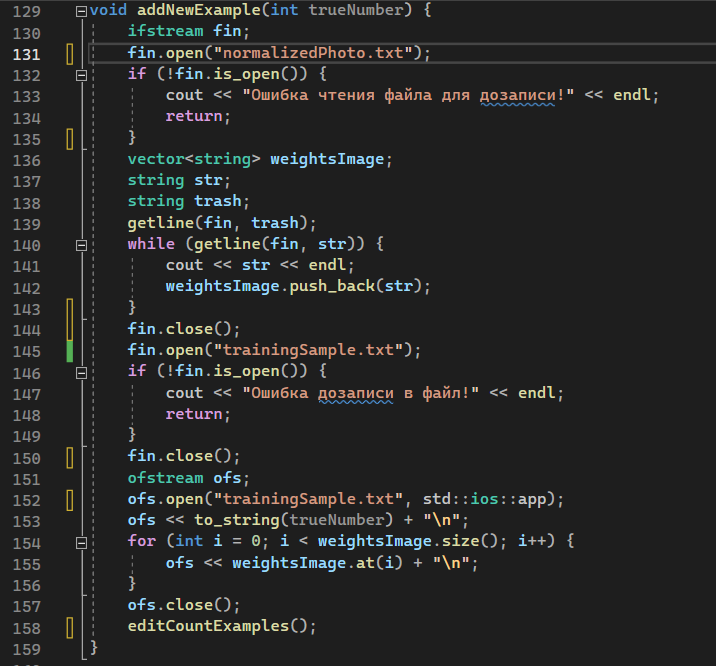


* ReadData – считывание фото в нужном представлении (числа [0;1]). Данные из MNIST, необходимые для обучения и тестов, находятся уже в правильном виде, так что остается их только считать.



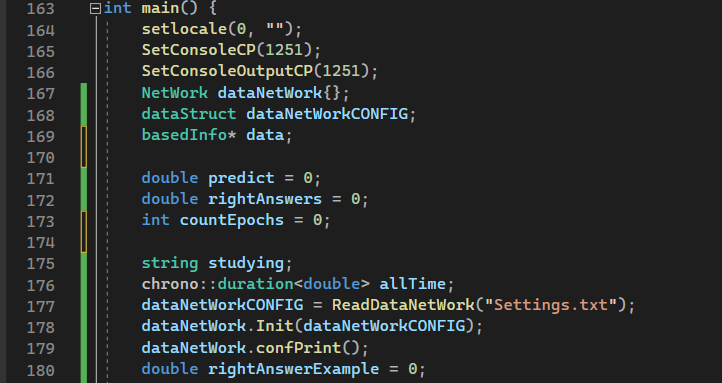
* photoProcessing – функция, который работает с новыми картинками, которые мы создаем, чтобы проверить работоспособность нейронной сети. С помощью библиотеки «opencv2» мы открываем фото (размер 28 \* 28), считываем каждый пиксель и превращаем его в число диапазона [0; 1] в зависимости от того, насколько он светлый/тёмный





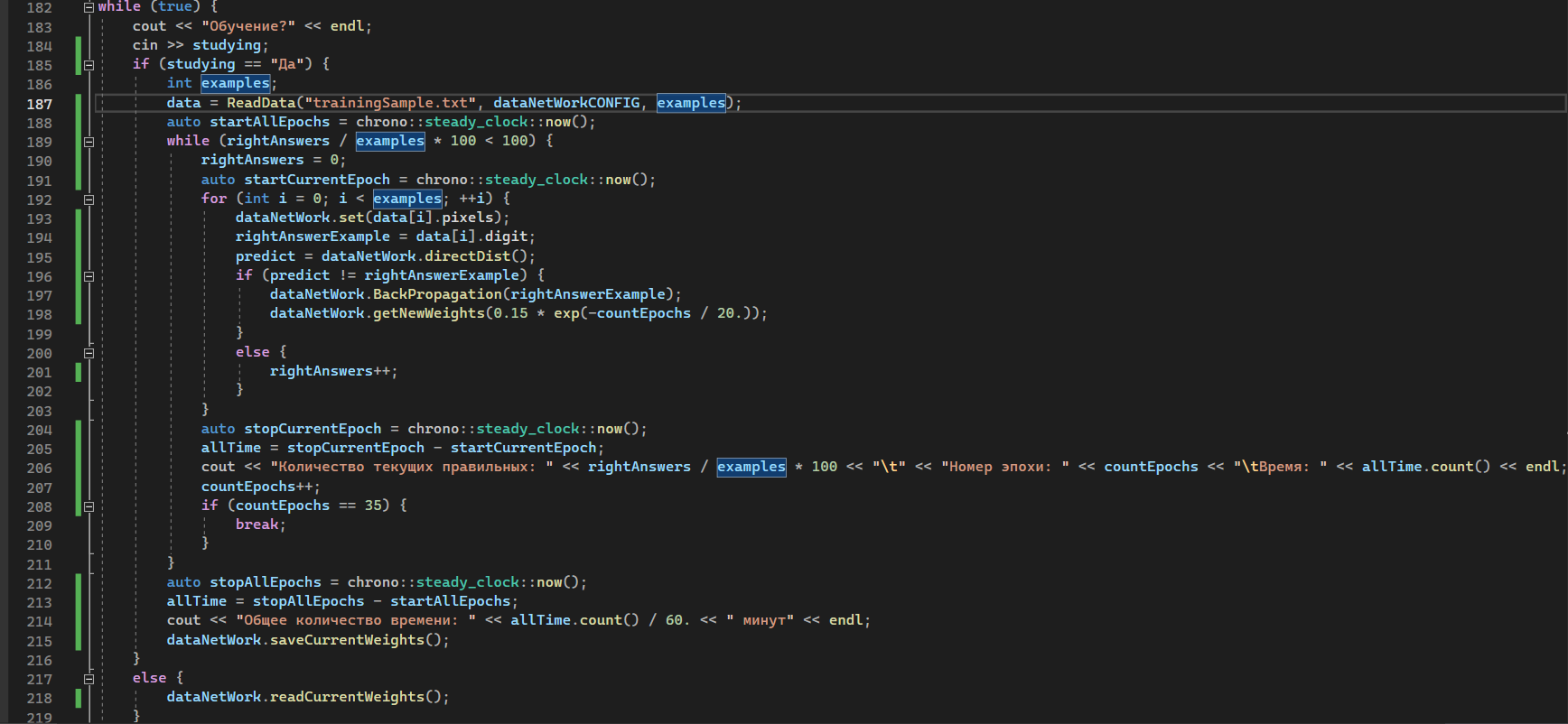
* addNewExample – функция, который добавляет в обучающую выборку нашу нарисованную картинку. Это сделано для того, чтобы можно было переобучать нейронную сеть, основываясь в том числе и на примерах пользователя
* editCountExamples – редактирует значение количества примеров. В файле «trainingSample”, который представляет собой файл со значениями из MNIST, первая строка содержит число, равное количеству примером в файле. Соответственно, при добавлении нового примера, необходимо увеличить значение на единицуа

**Source. main**

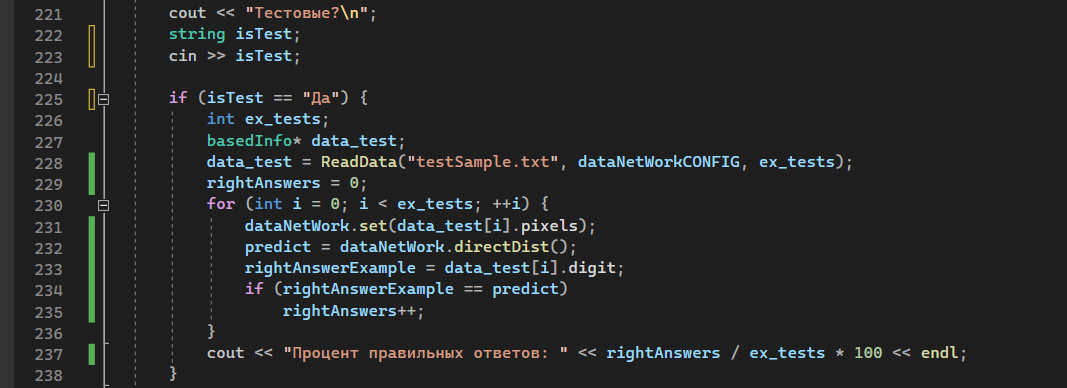


Остаётся написать main-функцию для нейронной сети. Сначала сделаем возможным использование русского языка, объявим необходимые структуры. Затем переменные:

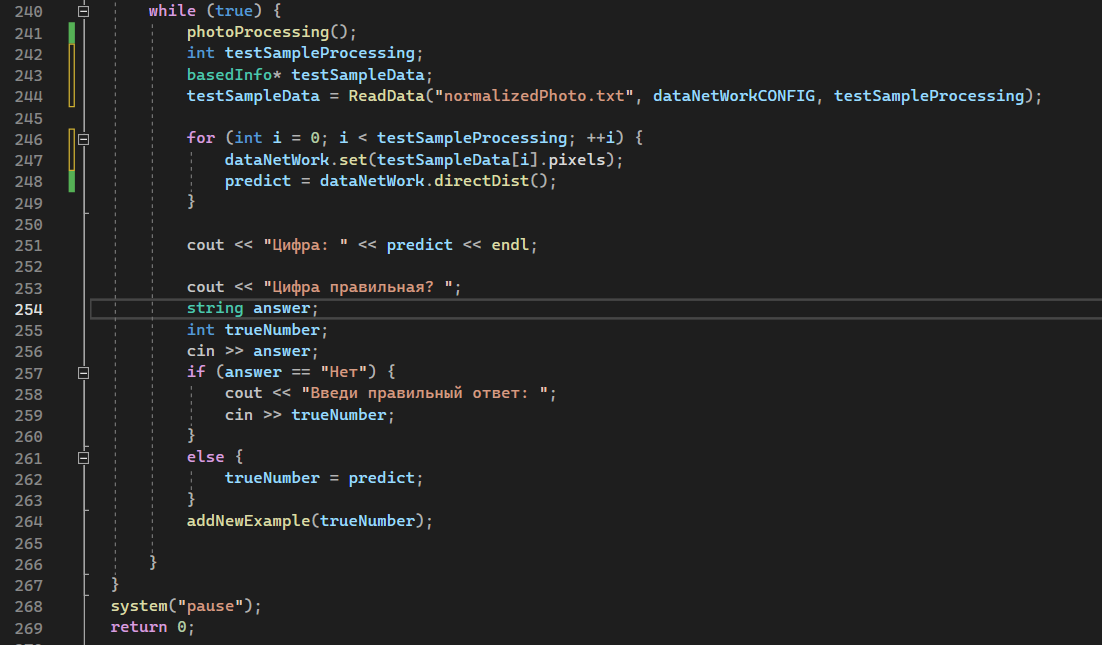
* predict – число, правильный ответ к примеру, который нейронная сеть должна получить. По умолчанию это ноль
* rightAnswers – количество правильных ответов эпохи
* countEpochs – количество эпох, до которого будет идти нейронная сеть, если количество правильных не будет 100%
* studying – вопрос для пользователя, нужно ли обучать нейронную сеть



В данном цикле происходит процесс обучения. Как уже было сказано, это происходит, пока мы не превысим количество эпох (*строка 208*), либо, пока точность не станет 100%. Переменные start и stop отвечают за счёт времени всего процесса обучения. Также есть переменные startCurrentEpoch и stopCurrentEpoch. Они отвечают за время одной эпохи.

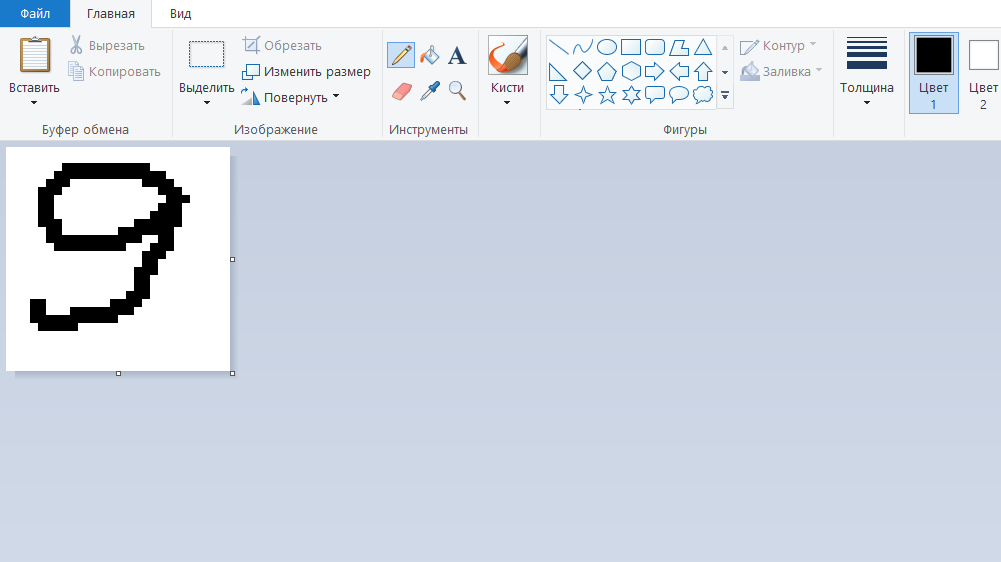


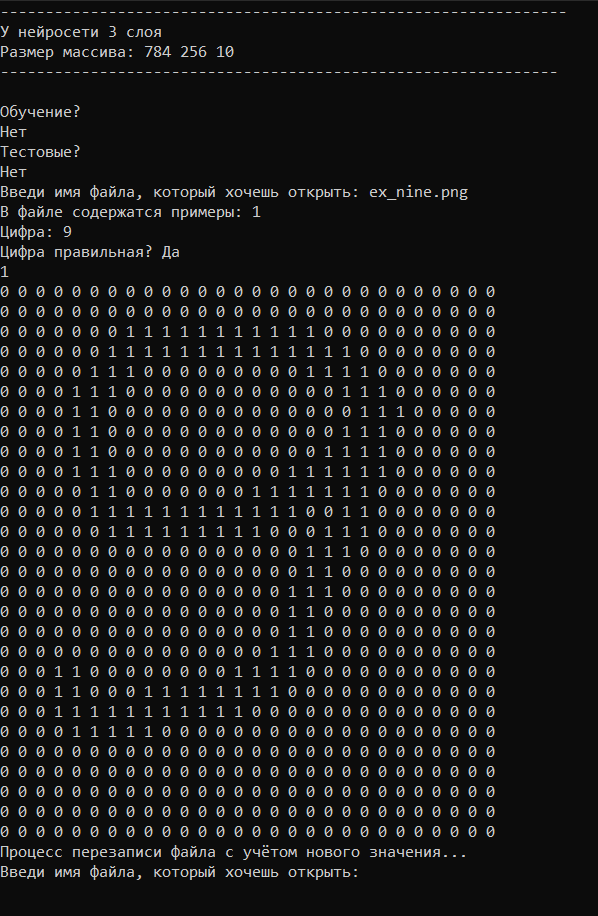
Далее необходимо узнать у пользователя, нужно ли анализировать работу нейронной сети на тестовых примерах библиотеки MNIST.



После обучения и тестов нужно дать возможность пользователю анализировать работу на собственных данных. Пользователь создаёт изображение с нарисованной цифрой, сохраняет её в ту же папку, где находится проект, или вводит полный путь до файла. Программа преобразует изображение к необходимому виду и, учитывая веса после обучения, выдаёт свой прогноз. Если ответ неправильный, необходимо ввести, какой ответ был верный. После этого происходит процесс записи нового примера в файл для обучения.

**Пример использования.**





Так как обучение и тесты уже проходили, отказываемся от предложения программы. Мой пример «ex\_nine.png» находится в той же директории, поэтому просто ввожу имя файла. Программа угадала число. Произошло дописывание примера.

Код на Гитхаб: <https://github.com/galerra/NeuralNetWork.git>