МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
Учреждение образования
«Витебский государственный технологический университет»

Е. Б. Дунина, А. С. Соколова, А. А. Корниенко

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебно-методическое пособие

Рекомендовано
Учебно-методическим объединением по образованию в области информатики и радиоэлектроники в качестве учебно-методического пособия для специальности 1-40 05 01 «Информационные системы и технологии (по направлениям)» направления специальности 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)»

Витебск
2024

УДК 004.8
ББК 32.813
И86

Рецензенты:

доцент кафедры прикладного и системного программирования УО «Витебский государственный университет им. П.М. Машерова» Ермоченко С.А.;

директор ООО «ДримСофт», г. Витебск Балюцкий Д.Л.

Одобрено кафедрой «Информационные системы и технологии» УО «ВГТУ», протокол № 6 от 07.12.2023.

Рекомендовано к изданию редакционно-издательским советом УО «ВГТУ», протокол № 4 от 27.12.2023.

И86 Искусственный интеллект. Лабораторный практикум : учебно-методическое пособие / Е. Б. Дунина, А. С. Соколова, А. А. Корниенко – Витебск : УО «ВГТУ», 2024. – 89 с.
ISBN 978-985-481-761-3

В учебно-методическом пособии изложены теоретические сведения и задания к лабораторным работам по дисциплине «Искусственный интеллект». Издание предназначено для студентов специальности 1-40 05 01-01 «Информационные системы и технологии (в проектировании и производстве)» дневной и заочной на базе ссуза форм обучения.

УДК 004.8
ББК 32.813

ISBN 978-985-481-761-3

СОДЕРЖАНИЕ

ЛАБОРАТОРНАЯ РАБОТА 1. ЭЛЕМЕНТЫ МАТЕМАТИЧЕСКОЙ ЛОГИКИ. АЛГЕБРА ВЫСКАЗЫВАНИЙ

Цель работы: изучение основных понятий алгебры высказываний.

Краткие теоретические сведения

Высказывание в математической логике – повествовательное предложение, о котором мы можем сказать, что оно либо истинно, либо ложно.

Высказывания обычно обозначаются большими латинскими буквами: *A, B, C,* .... Логические действия над ними записывают с помощью значков операторов. Поскольку высказывание рассматривается как величина, которая принимает одно из двух значений: истина или ложь, то эти значения обычно обозначают через 1 и 0.

Рассмотрим логические операции для двух произвольных высказываний *A* и *B*.

Определение 1.1. *Отрицанием* высказывания *A* называется новое высказывание, истинное тогда и только тогда, когда *A* ложно. Отрицание обозначается $\neg A$ или $\overline{A}$. Операция отрицания определяется следующей таблицей истинности:

A	$\overline{A}$
1	0
0	1

Определение 1.2. *Конъюнкция* высказывания (логическое умножение, оператор И) – высказывание, которое будет истинным только в случае, когда истинны все его составляющие. Конъюнкция обычно обозначается значком «$\wedge$».

A	B	$A \wedge B$
1	1	1
1	0	0
0	1	0
0	0	0

Определение 1.3. *Дизъюнкция* (логическое сложение, оператор ИЛИ) – высказывание, которое будет ложным, только в случае, когда ложны все составляющие, и истинно во всех остальных случаях. Дизъюнкция обозначается значком «$\vee$».

A	B	$A \vee B$
1	1	1
1	0	1
0	1	1
0	0	0

Определение 1.4. Высказывание, обозначаемое $A \to B$ («если A то B»), ложно только в том случае, когда A истинно, а B ложно. И называется *импликацией* с посылкой A и заключением B.

A	B	$A \to B$
1	1	1
1	0	0
0	1	1
0	0	1

Определение 1.5. Высказывание, обозначаемое $A \leftrightarrow B$ «A эквивалентно B», истинное в том случае, когда A и B имеют одно и то же истинное значение, называется *эквиваленцией*.

A	B	$A \leftrightarrow B$
1	1	1
1	0	0
0	1	0
0	0	1

Замечание. Операции по силе своего действия можно расположить в порядке убывания: $\neg$, $\wedge$, $\vee$, $\to$, $\leftrightarrow$.

Пример: в формуле $B \leftrightarrow \neg C \vee D \wedge A$ мы можем скобки расставить следующим образом:

$$B \leftrightarrow (\neg C) \vee D \wedge A,$$
$$B \leftrightarrow (\neg C) \vee (D \wedge A),$$
$$B \leftrightarrow ((\neg C) \vee (D \wedge A)),$$
$$(B \leftrightarrow ((\neg C) \vee (D \wedge A))).$$

Определение 1.6. *Тавтологией, тождественно истинной формулой или законом логики* называется формула логики высказываний, принимающая значение «истина» при любом распределении значений входящих в эту формулу элементов.

Определение 1.7. Если формула логики высказываний принимает значение «ложь» при любом распределении значений входящих в эту формулу элементов, то она называется *противоречием или тождественно ложной формулой*.

Пример: докажем, что формула $(P \to Q) \wedge (Q \to R) \to (P \to R)$ является тавтологией. Для этого составим таблицу истинности:

P	Q	R	$P \to Q$	$Q \to R$	$P \to R$	$(P \to Q) \wedge (Q \to R) \to (P \to R)$
1	1	1	1	1	1	1
1	1	0	1	0	0	1
1	0	1	0	1	1	1
1	0	0	0	1	0	1
0	1	1	1	1	1	1
0	1	0	1	0	1	1
0	0	1	1	1	1	1
0	0	0	1	1	1	1

Задания для самостоятельной работы

1. Для каждой из формул составьте таблицу истинности:

а) $p \to \neg(q \wedge r)$;

б) $p \to q \leftrightarrow \neg p \vee q$;

в) $r \to (p \to q)$;

г) $(p \to q) \to (s \wedge \neg s) \to p \vee s)$.

2. Восстановив предварительно скобки в логических формулах, составьте таблицу истинности:

а) $A \vee B \wedge C \vee \neg A \wedge B$;

б) $A \vee \neg B \wedge C \to A \wedge \neg B$;

в) $A \vee \neg B \wedge C \to B \leftrightarrow \neg C$.

3. Докажите, что следующие формулы являются тавтологиями:

а) $(A \wedge B) \to C \leftrightarrow A \to (B \to C)$;

б) $(A \wedge B) \to C \leftrightarrow (A \wedge \neg C) \to \neg B$;

в) $\neg(A \to B) \leftrightarrow A \wedge \neg B$;

г) $(A \to B) \wedge \neg B \to \neg A$;

д) $A \to (\neg A \to B)$;

е) $A \to (B \to A)$;

ж) $(\neg A \to A) \to A$;

з) $(A \to B) \to (A \wedge C \to B \wedge C)$;

и) $(A \to B) \wedge (C \to D) \to (A \wedge C \to B \wedge D)$;

к) $(A \to B) \wedge (C \to D) \to (A \vee C \to B \vee D)$;

л) $\neg(A \leftrightarrow B) \leftrightarrow (\neg(A \to B) \vee \neg(B \to A))$.

ЛАБОРАТОРНАЯ РАБОТА 2. РАБОТА В NEURAL NETWORK PATTERN RECOGNITION TOOL

Цель работы: изучение основ работы с GUI-интерфейсом пакета Neural Networks Toolbox в программной среде MatLab R2021a.

Краткие теоретические сведения

Рассмотрим основные принципы работы с GUI-интерфейсом пакета Neural Networks Toolbox. MATLAB создает двухслойную сеть прямого распространения с одним скрытым слоем и выходным слоем. В качестве алгоритма обучения используется метод обратного распространения ошибки. В скрытом слое используется сигмоидальная функция активации.

Для запуска указанного инструмента в командном окне MatLab необходимо выполнить команду:

```
>> nprtool
```

В результате появится окно, представленное на рис. 2.1.

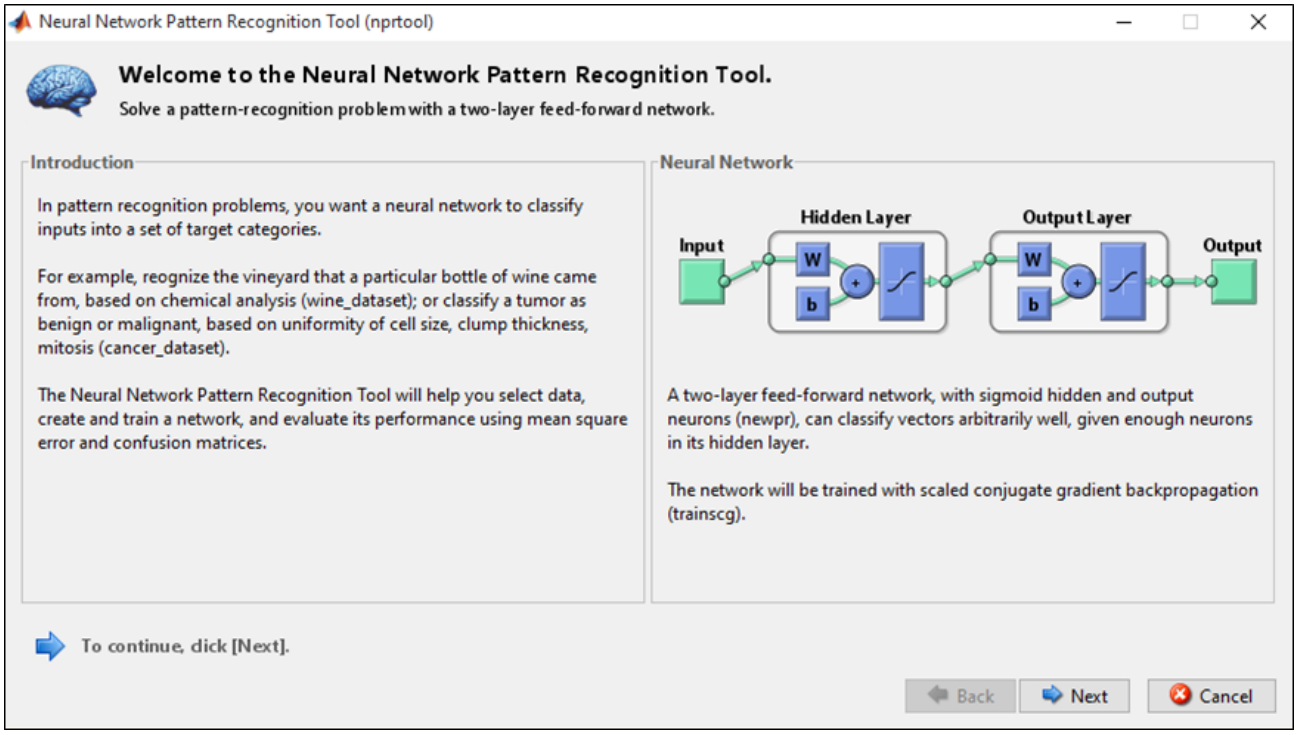


Рисунок 2.1 – Окно приглашения в инструмент распознавания образов нейронной сети

Если нажать кнопку «Next», мы перейдем к окну Select Data для выбора данных (рис. 2.2).

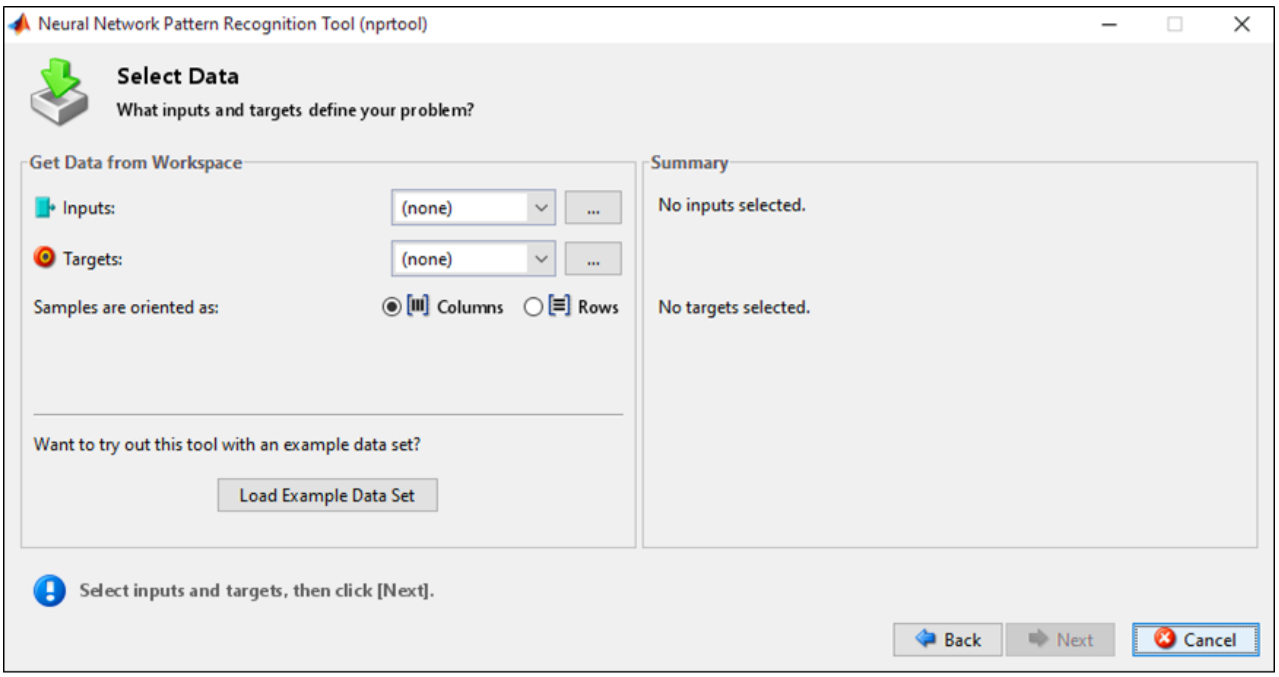


Рисунок 2.2 – Окно для выбора данных

С помощью кнопки ... мы можем сделать выбор входных (Inputs) и целевых (Targets) данных. Готовые демонстрационные примеры мы можем выбрать, используя кнопку Load Example Data Set. В этом случае мы перейдем к окну Pattern Recognition Data Set Chooser (рис. 2.3).

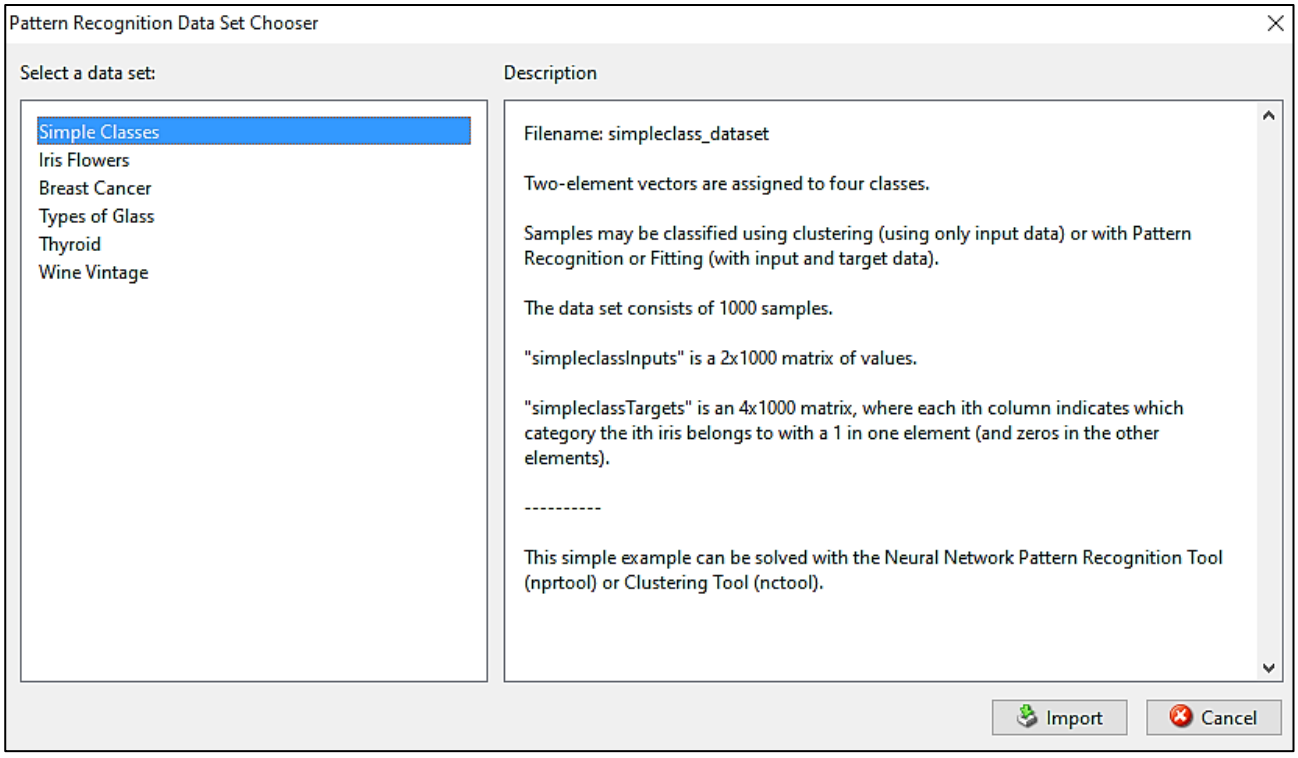


Рисунок 2.3 – Окно выбора демонстрационных примеров

В демонстрационных примерах доступны следующие наборы данных:

1. Simple Classes – простые классы. Имя файла: simpleclass_dataset. Набор данных включает 1000 образцов.

2. Iris Flowers – распознавание сорта цветка ириса по четырем признакам: длина и ширина чашелистика, длина и ширина лепестка. Имя файла: iris_dataset. Набор содержит 150 образцов.

3. Breast Cancer – классификация опухоли молочной железы, на основе девяти характеристик образцов биопсии. Набор содержит 699 образцов. Имя файла: Cancer_dataset.

4. Types of Glass – классификация стекла (оконное или не оконное) на основе изучения химического состава. Набор состоит из 214 образцов. Имя файла: glass_dataset.

5. Thyroid – классификация пациентов по состоянию щитовидной железы. Набор данных включает показатели исследований 21 пациента.

6. Wine Vintage – классификация вин с помощью химического анализа, на основе 13 компонентов. Набор включает 178 образцов.

После выбора нужного класса, в правой половине окна (Description), можно увидеть описание каждого класса и источник.

Порядок выполнения работы

Выбираем, например, набор данных Types of Glass и нажимаем кнопку Import и Next. В окне Validation and Test Data (рис. 2.4) можно увидеть распределение данных между обучающим (Training), проверочным (Validation) и тестовым набором (Testing).

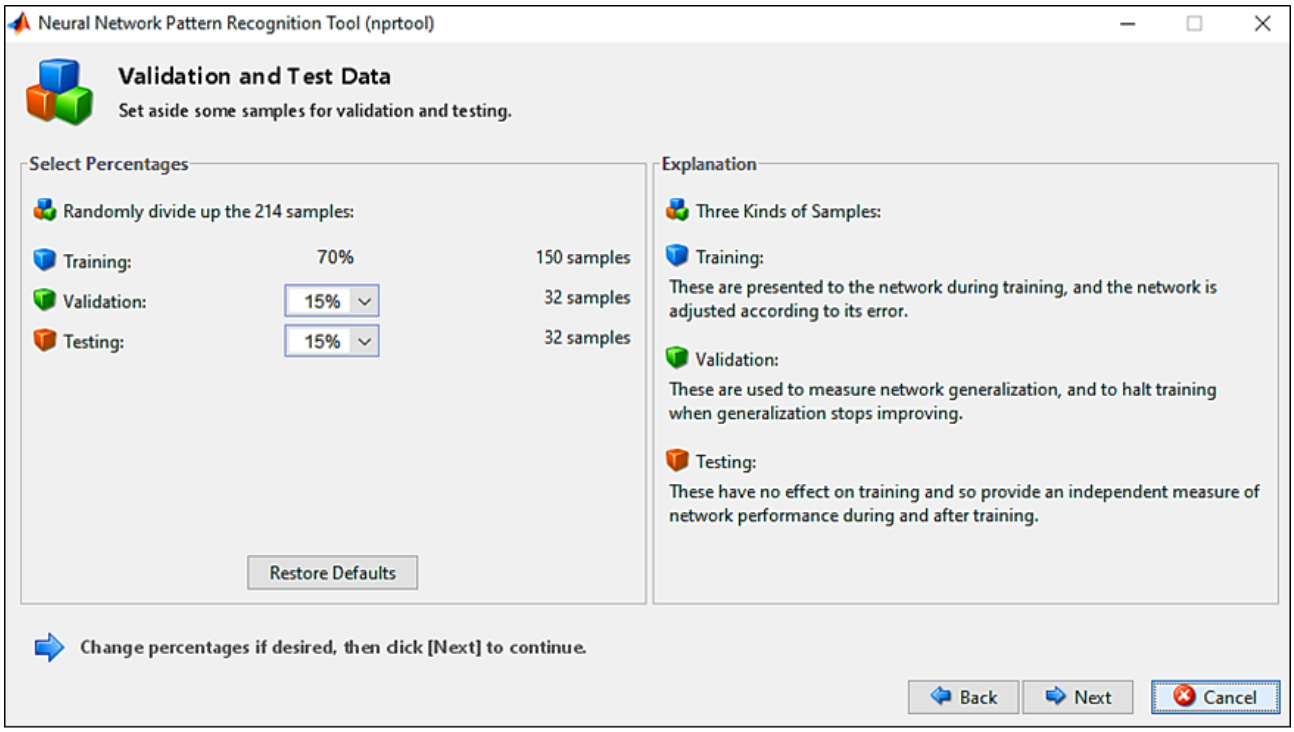


Рисунок 2.4 – Окно распределения набора данных

Если нажать кнопку Next, то распределение данных останется без изменения и мы перейдем в окно Network Size (рис. 2.5).

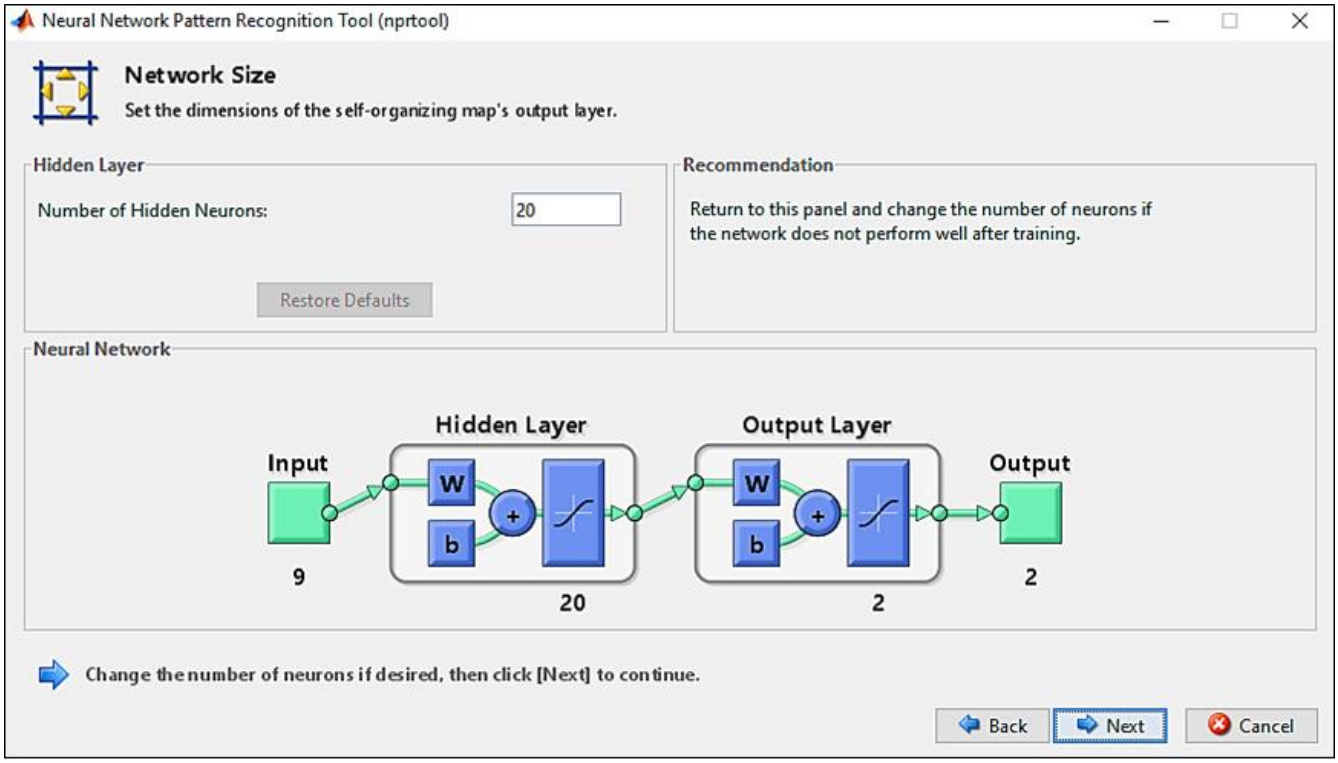


Рисунок 2.5 – Окно архитектура сети

При необходимости мы можем изменить число скрытых нейронов, по умолчанию оно равно 20. Число нейронов выходного слоя равно двум. Поскольку входные данные должны быть разделены на 2 типа: оконное или не оконное стекло. GlassInputs представляет собой матрицу 9×214, строки которой:

– показатель преломления;

– натрий (единица измерения: весовой процент в соответствующем оксиде);

– магний;

– алюминий;

– кремний;

– калий;

– кальций;

– барий;

– железо.

GlassTargets – это матрица 2×214, где каждый столбец указывает правильную категорию с 1 в одном элементе и 0 в другом: оконное стекло и неоконные стекло.

После нажатия кнопки Next осуществляется переход к окну Train Network (рис. 2.6).

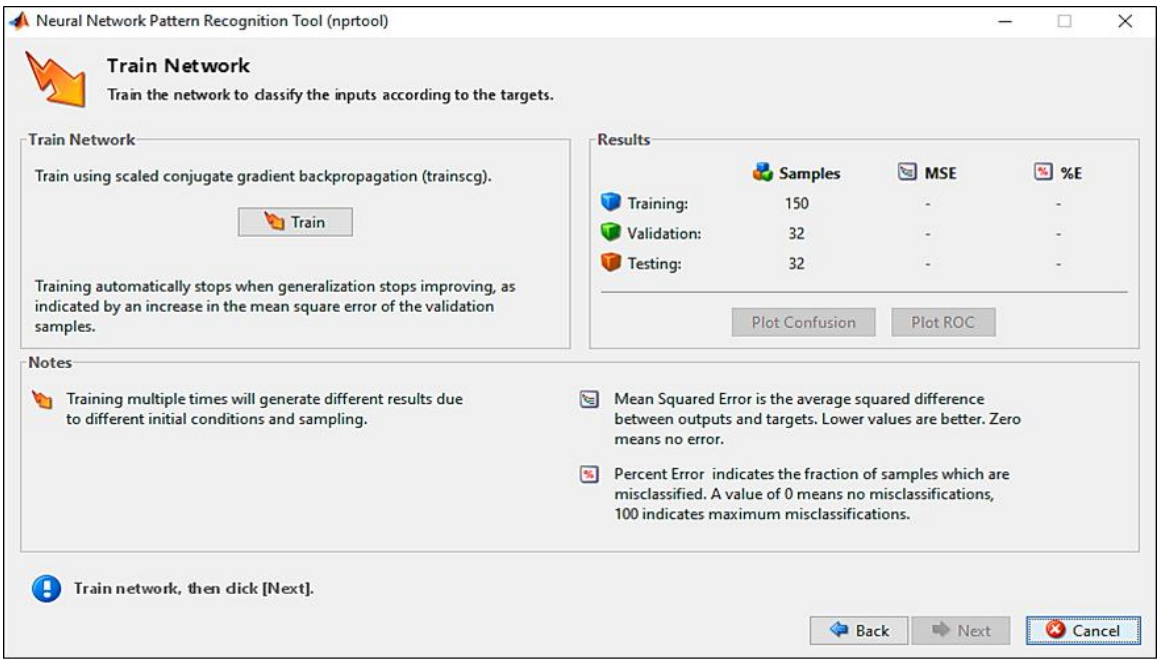


Рисунок 2.6 – Окно обучения сети

По кнопке Train – переход в окно обучения сети (рис. 2.7). Одновременно появляется окно, позволяющее повторить обучение (рис. 2.8).

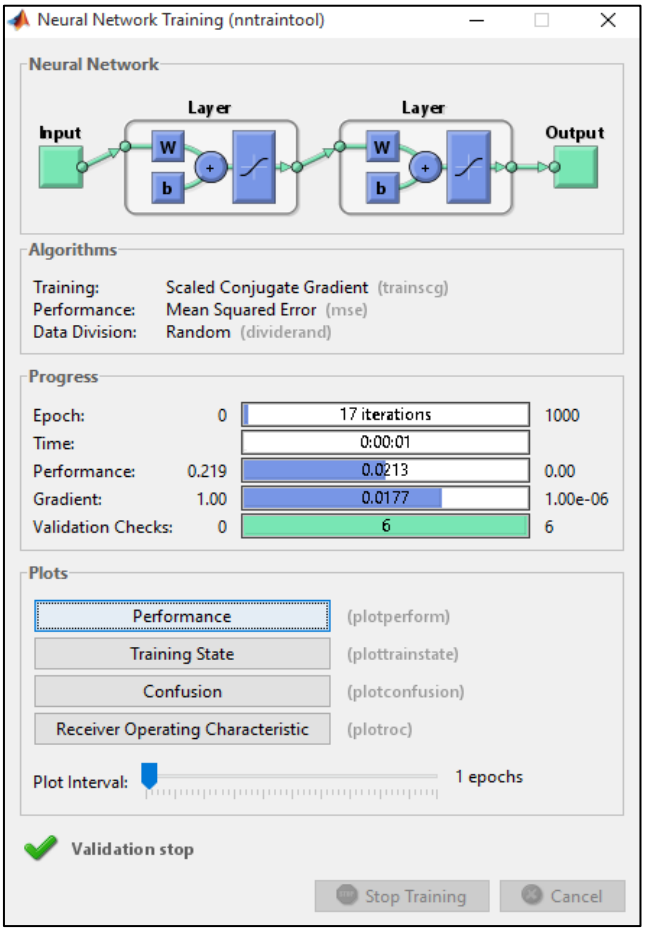


Рисунок 2.7 – Окно процесса обучения

В окне процесса обучения, нажав на кнопку Performance, мы перейдем к графику обучения сети (рис. 2.9), который показывает поведение ошибки обучения на обучающем, контрольном и тестовом множествах.

Графики Training State (рис. 2.10) показывают изменение градиента функционала ошибки и величины ошибки на проверочном множестве в процессе обучения.

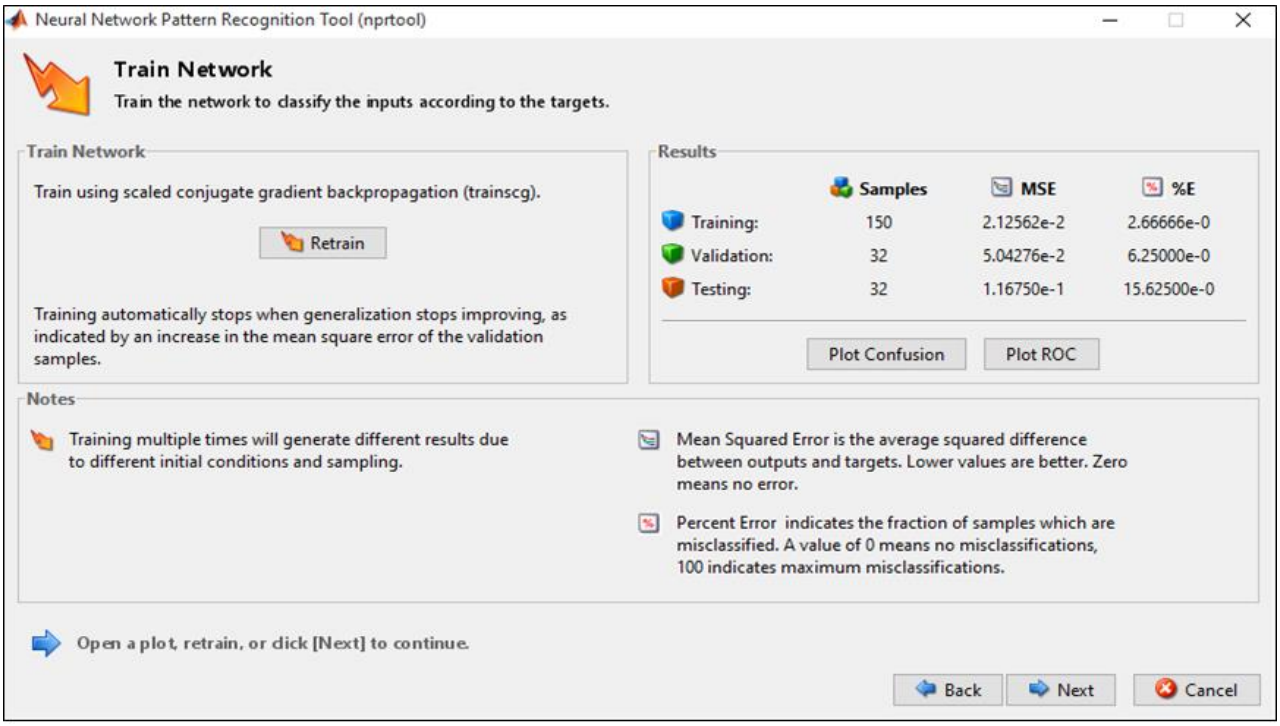


Рисунок 2.8 – Окно обучения сети

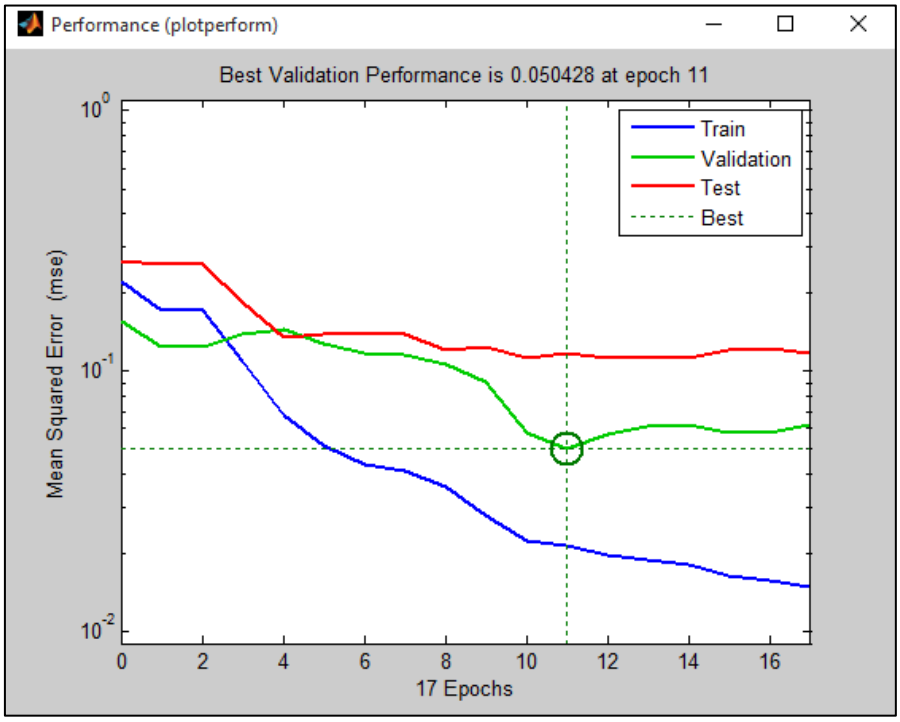


Рисунок 2.9 – График обучения сети распознавания

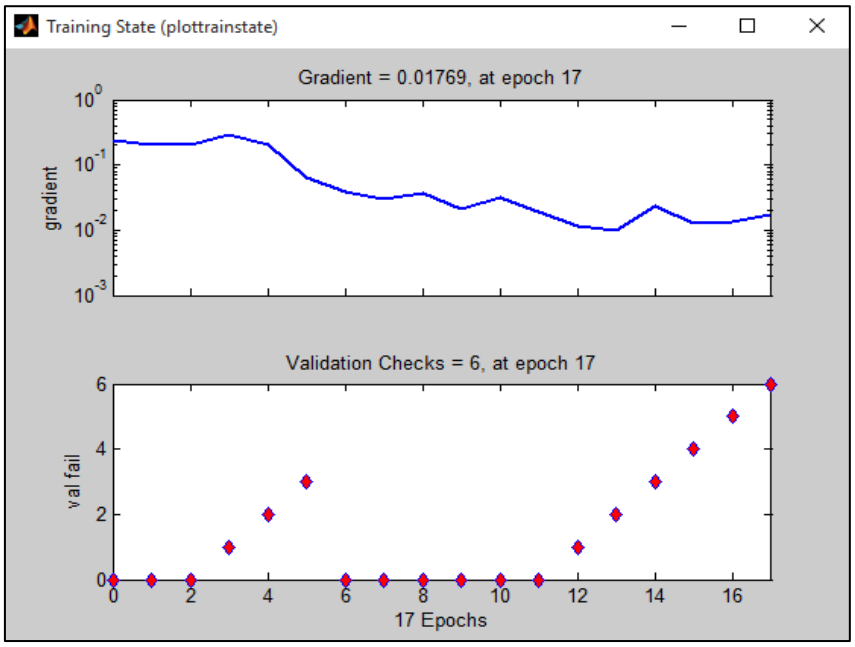


Рисунок 2.10 – Графики состояния обучения

Используя кнопку Plot Confusion (рис. 2.8) можно получить матрицу несоответствий сети (Confusion) (рис. 2.11). Столбцы матрицы соответствуют предсказанным классам, а строки – фактическим классам.

Рисунок 2.11 – Матрица ошибок сети

Приводятся матрицы несоответствия: Training Confusion Matrix (Матрица ошибок обучения), Validation Confusion Matrix (Матрица ошибок контрольного

множества), Test Confusion Matrix (Матрица ошибок тестового множества) и All Confusion Matrix (Матрица ошибок суммарного множества). Если, в Training Confusion Matrix (рис.2.11) 32 объекта от общего объема обучающей выборки были распознаны как объекты первого типа и 1 объект как второго типа. Тогда процент правильно распознанных объектов первого типа равен 97 %, а ошибка распознавания объектов второго типа равна 3,0 %. Из последней строки для данной матрицы можно увидеть, что из всех объектов, классифицированных как объекты первого класса, 91,4 % составляют объекты первого класса и ошибка классификации составила 8,6 %. Неправильно классифицировано в целом 2,7 % объектов.

Кроме этого, в окне процесса обучения (рис. 2.7) нажав кнопку Receiver Operating Characteristic, мы получим характеристические кривые обнаружения (рис. 2.12).

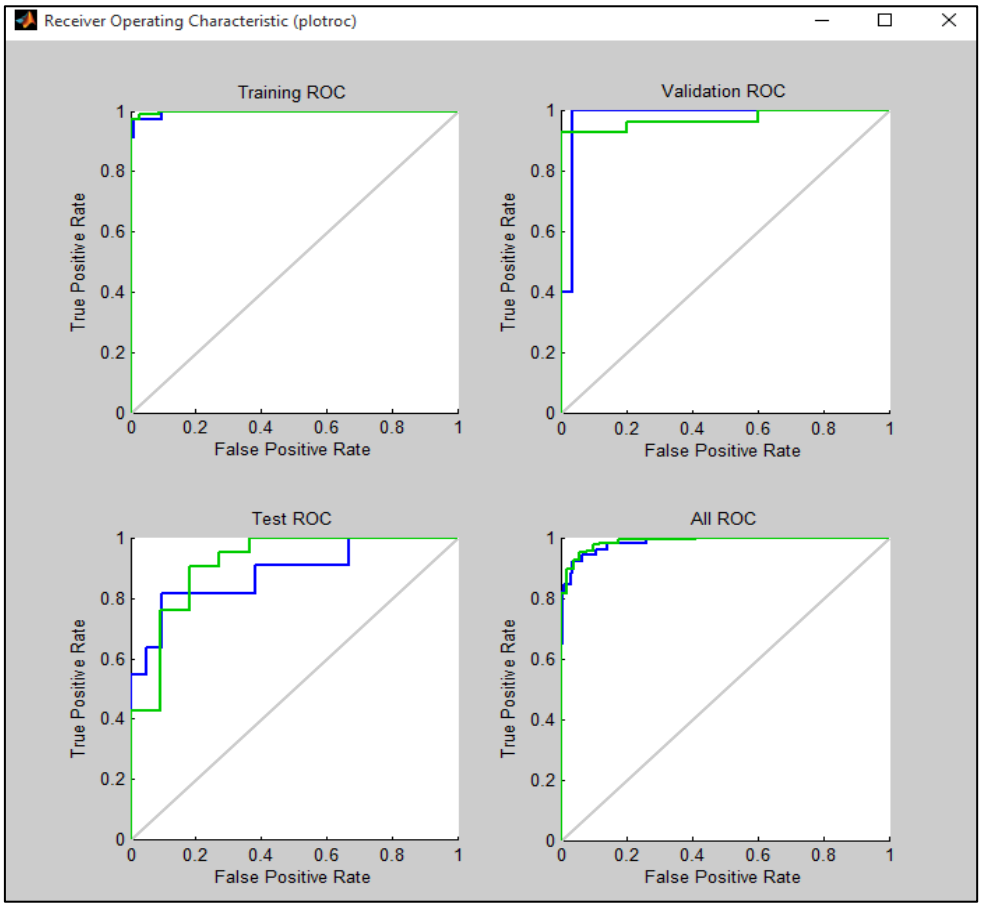


Рисунок 2.12 – Характеристические кривые обнаружения

Классификатор нейронной сети относит выход сети к одному из классов в зависимости от того, превышает ли выходной сигнал пороговое значение – Cut-Off Point (точка отсечения).

Для построения ROC-кривой, нужно изменять порог отсечения в интервале от 0 (или -1) до 1 с заданным шагом. Для каждого класса классификатора с некоторым шагом изменяются пороговые значения. Для каждого порогового значения вычисляются: доля истинно положительных

примеров (True Positives Rate – TPR) – колличество выходов, больших или равных порогу, разделенное на количество единичных целевых значений; доля ложно положительных примеров (False Positives Rate – FPR) – количество выходов, меньших чем порог, разделенное на количество нулевых целевых значений. Затем строится график зависимости TPR от FPR при разных значениях порога и дополняется прямой $y = x$, которая соответствует «бесполезному» классификатору, то есть когда результат получается случайным угадыванием. В случае идеального классификатора график ROC-кривой проходит через верхний левый угол. При этом доля положительных случаев будет составлять 100 %.

Если показатели классификатора получились неудовлетворительные, то можно повторить обучение сети при помощи кнопки Retrain в окне Train Network (рис. 2.8).

Чтобы оценить работу сети, необходимо нажать кнопку Next в окне Train Network (рис. 2.8) после завершения обучения. Появится окно Evaluate Network – оценка сети.

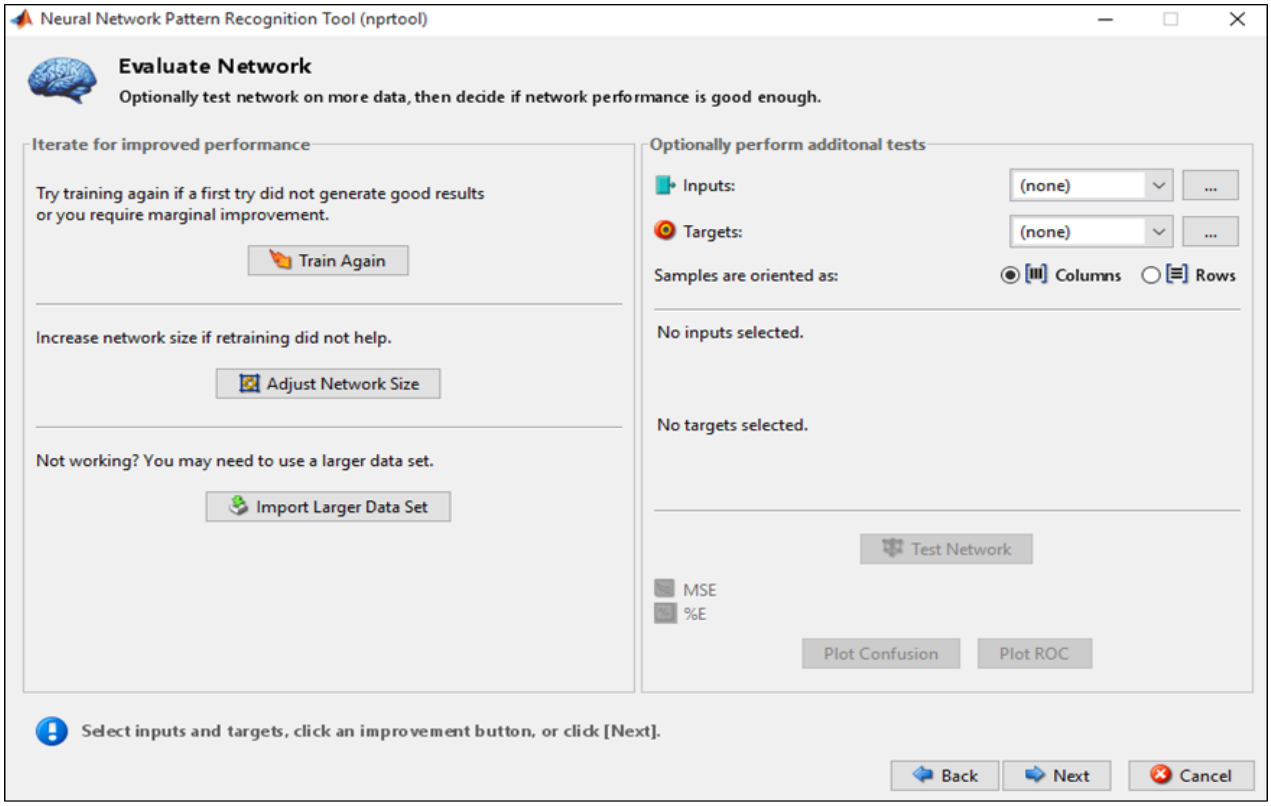


Рисунок 2.13 – Окно оценки сети

Кнопка Train Again предлагает обучить сеть, кнопка Import Larger Data Set позволяет использовать больший набор данных, а кнопка Adjust Network Size позволяет увеличить размер сети.

При нажатии Next мы перейдем в окно Save Results для размещения решения (рис. 2.14). В нем можно сохранить результаты – сгенерировать m-файл.

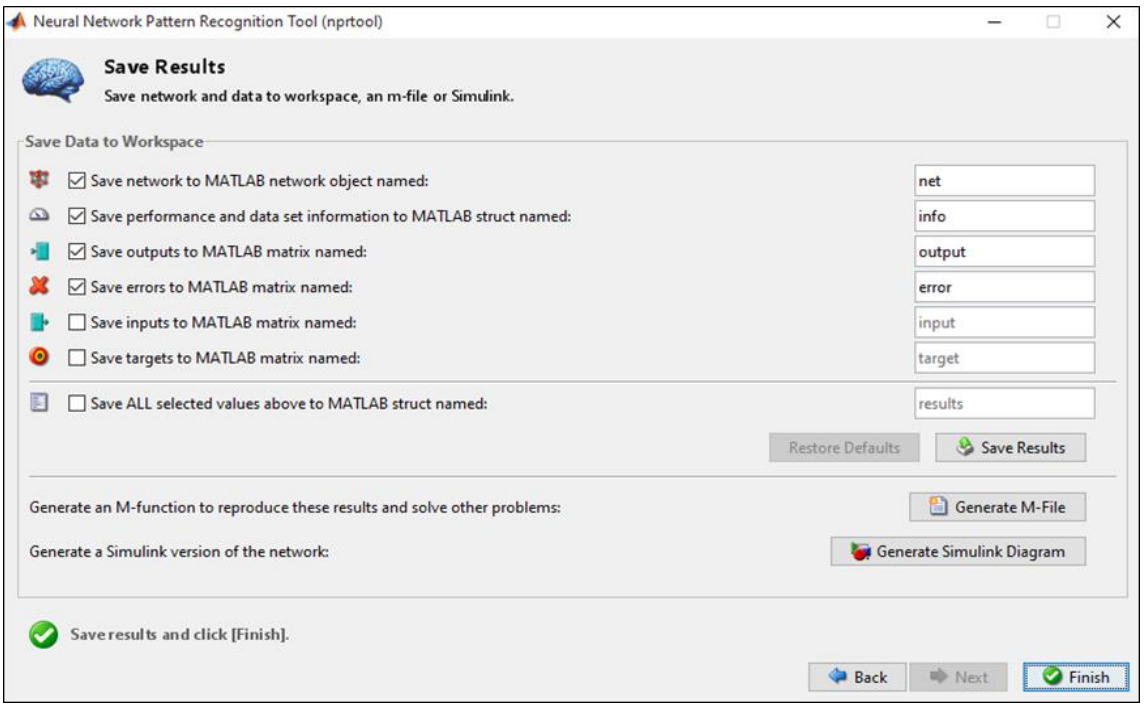


Рисунок 2.14 – Окно развертывания решения

Для завершения работы необходимо нажать кнопку Finish.

Задания для самостоятельной работы

1. Выполнить обучение для всех наборов данных, описанных выше.
2. Исследовать как влияет на точность распознавания структура сети.

ЛАБОРАТОРНАЯ РАБОТА 3. КЛАССИФИКАЦИЯ С ПОМОЩЬЮ ПЕРСЕПТРОНА

Цель работы: создание и исследование моделей нейронных сетей в программной среде MatLab R2021a.

Краткие теоретические сведения

3.1 Классификация однослойным персептроном на два класса

Построим персептронную нейронную сеть, которая производит разделение следующих векторов на два класса: $A_1[-0,7;-0,6]$ – 1 класс; $A_2[-0,5;0,2]$ – 1 класс; $A_3[-0,6;0,5]$ – 1 класс; $A_4[0,4;0,5]$ – 2 класс; $A_5[0,6;1]$ – 2 класс. Вектор T определяет целевые классы для этих векторов. Его можно задать в виде $T=[0\ 0\ 0\ 1\ 1]$. Визуализировать векторы в виде обучающих точек в MatLab можно при помощи команды plotpv (рис. 3.1).

Листинг 3.1 – Подготовка данных для обучения персептрона

```
% X - входные вектора
X = [-0.7 -0.5 -0.6 0.4 0.6; -0.6 0.2 0.5 0.5 1.0];
% T - целевые классы
T = [0 0 0 1 1];
plotpv (X, T);
```

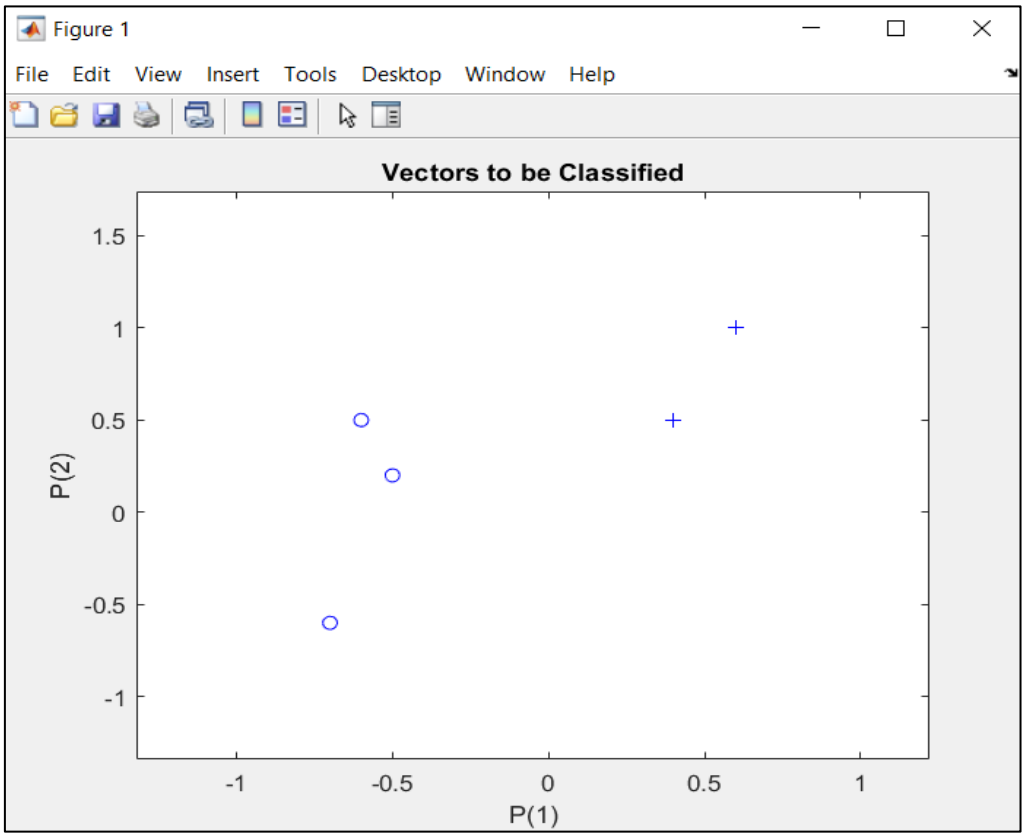


Рисунок 3.1 – Обучающие точки

Количество классов равно 2, поэтому сеть выберем из одного нейрона. Входами будут координаты X и Y. Выход – номер класса (рис.3.2).

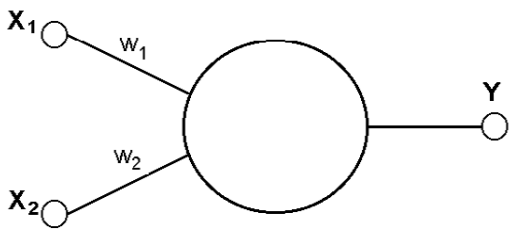


Рисунок 3.2 – Однонейронный персептрон с одним нейроном:
X_1 – входная координата X; X_2 – входная координата Y;
Y – выход (номер класса)

Листинг 3.2 – Решение задачи классификации

```
X = [-0.7 -0.5 -0.6 0.4 0.6;-0.6 0.2 0.5 0.5 1.0];
T = [0 0 0 1 1];
```

```
plotpv (X, T);
% создание нейросети
net = perceptron;
net = configure(net,X,T);
% создание серии для входных данных
view(net)
inputweights = net.inputweights{1,1}
XX = repmat(con2seq(X),1,3);
% создание серии для выходных данных
TT = repmat(con2seq(T),1,3);
% настройка нейросети
net = adapt(net,XX,TT);
% визуализация разделяющей линии
plotpc(net.IW{1},net.b{1});
w = net.IW{1};
b = net.b{1};
```

Команда perceptron создает новую нейронную сеть с одним нейроном. Команда net = configure (net,X,T) принимает входные данные X и целевые данные T и настраивает входы и выходы сети в соответствии с ними.

В приведенном примере входные и целевые данные преобразуются в последовательности (массив ячеек, где каждый столбец указывает временной шаг) и копируются три раза при помощи команды repmat, чтобы сформировать серии XX и TT. Для каждого временного шага в серии команда adapt обновляет сеть и возвращает новый сетевой объект. Последний выполняет функции классификатора для данных.

Из рис. 3.3 видно, что в результате всего трех тактов обучения нейросети, содержащая всего один персептрон в одном слое, удалось классифицировать два класса в двумерном пространстве входов.

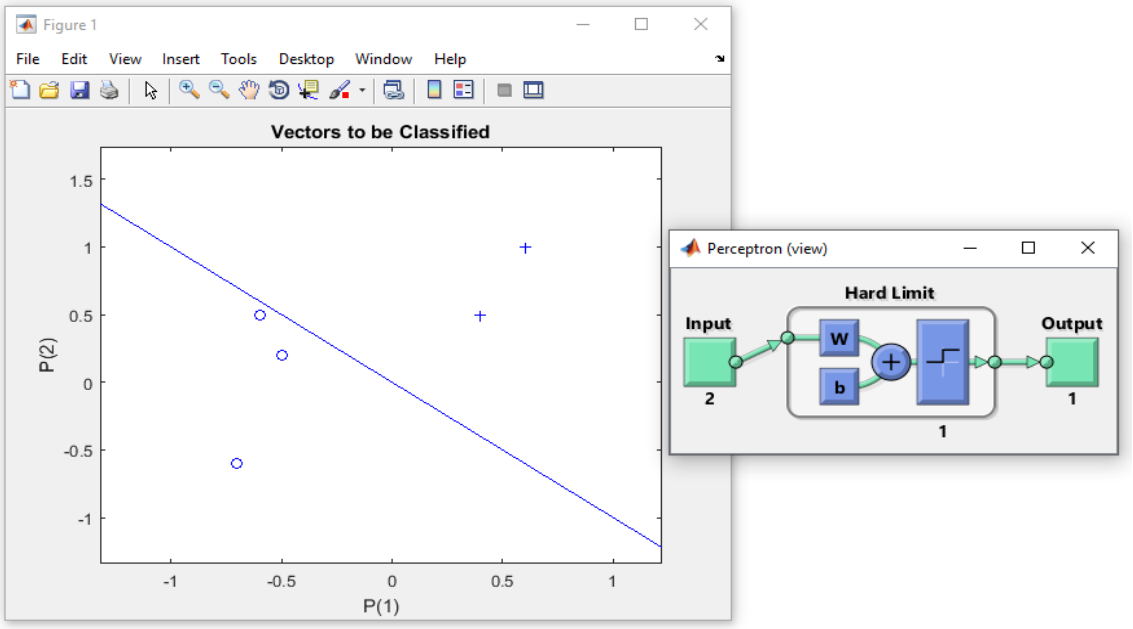


Рисунок 3.3 – Разделения обучающих входных векторов

Возьмем произвольный вектор, не входящий в обучающую выборку, например [0,7; 1,5]. Посмотрим, правильно ли обученная сеть отнесет вектор к нужному классу (листинг 3.3). Чтобы тестовая точка отличалась от точек обучающей выборки, закрасим ее, например, красным цветом.

Листинг 3.3 – Тестирование нейронной сети

```
x = [0.7; 1.5];
y = net(x);
plotpv(x,y);
point = findobj(gca,'type','line');
point.Color = 'red';
hold on;
plotpv(X,T);
plotpc(net.IW{1},net.b{1});
hold off;
```

Функция Findobj осуществляет поиск графических объектов по заданной паре свойство-значение, gca возвращает текущее графическое окно. Команда hold on позволяет добавлять новые графики к уже существующим (по умолчанию hold off – графики заменяются).

На рис. 3.4 видно, что наша новая точка выделена красным плюсом, а не кружком. Из этого следует, что персептрон выполнил правильную классификацию.

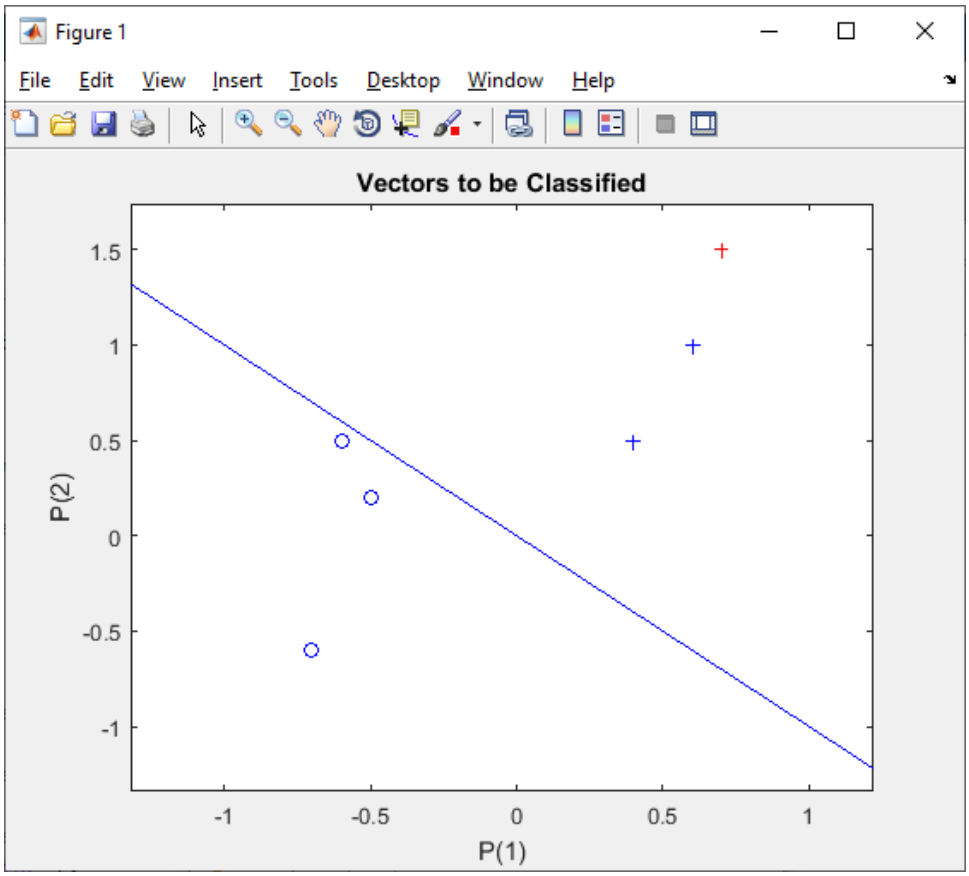


Рисунок 3.4 – Результат классификации тестового примера

3.2 Классификация однослойным персептроном на четыре класса

Для решения задачи классификации в двумерном пространстве на 4 класса необходимо иметь 2 нейрона с пороговой функцией активации. Случайным образом создаются обучающие выборки по 10 примеров для 4 классов, которые в дальнейшем будут разделяться (листинг 3.4).

Листинг 3.4 – Генерация 4 классов обучающей выборки

```
clc, format compact
K = 10;
q = .5; % смещение классов относительно друг друга
A = [rand(1,K)-q; rand(1,K)+q];
B = [rand(1,K)+q; rand(1,K)+q];
C = [rand(1,K)+q; rand(1,K)-q];
D = [rand(1,K)-q; rand(1,K)-q];
plot(A(1,:),A(2,:),'bs') % визуализация обучающих выборок
hold on
grid on
plot(B(1,:),B(2,:),'r+')
plot(C(1,:),C(2,:),'go')
plot(D(1,:),D(2,:),'m*')
text(.5-q,.5+2*q,'Класс A')
text(.5+q,.5+2*q,'Класс B')
text(.5+q,.5-2*q,'Класс C')
text(.5-q,.5-2*q,'Класс D')
% определение выходных кодировок классов
a = [0 1]';
b = [1 1]';
c = [1 0]';
d = [0 0]';
```

Обозначения для типа линий, цветов и маркеров при построении графиков в MatLab приведены в таблице 3.1.

Таблица 3.1 – Обозначения для типа линий, цветов и маркеров

Цвет		Маркер	
y	Желтый	.	Точка
m	Манджента	o	Кружок
c	Голубой	x	Крестик
r	Красный	+	Знак «плюс»
g	Зеленый	*	Звездочка
b	Синий	s	Квадрат
w	Белый	d	Ромб
k	Черный	v	Треугольник вершиной вниз
Линия		^	Треугольник вершиной вверх
-	Сплошная	<	Треугольник вершиной влево
:	Пунктирная	>	Треугольник вершиной вправо
-.	Штрих-пунктирная	P	Пятиконечная звезда
--	Штриховая	H	Шестиконечная звезда

Персептрон создается в переменной net при помощи функции perceptron (net = perceptron) и затем обучается в цикле while, повторяя процедуру adapt не более 1000 раз или пока ошибка E не станет равна 0 (листинг 3.5).

Листинг 3.5 – Обучение нейросети

```
P = [A B C D];
% массив целевых векторов P
T = [repmat(a,1,length(A)) repmat(b,1,length(B)) ...
repmat(c,1,length(C)) repmat(d,1,length(D))];
net = perceptron;
E = 1;
net.adaptParam.passes = 1;
linehandle = plotpc(net.IW{1},net.b{1});
n = 0;
while (sse(E) & n<1000)
n = n+1;
[net,Y,E] = adapt(net,P,T);
linehandle = plotpc(net.IW{1},net.b{1},linehandle);
drawnow;
end
view(net);
```

Функция sse определяет производительности сети. Измерение происходит по сумме квадратов ошибок.

В каждом цикле обучения при помощи команды plotpc рисуются разделительные линии.

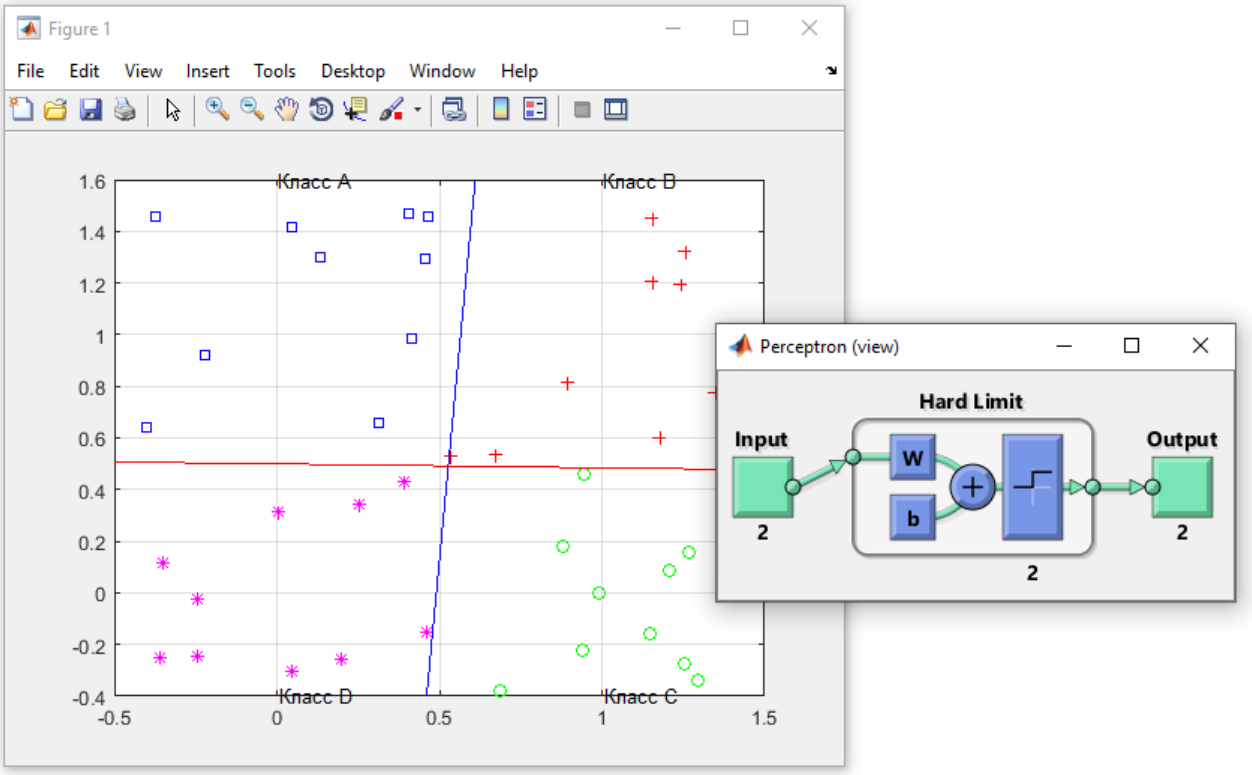


Рисунок 3.5 – Классификация однослойным персептроном на 4 класса

Задания для самостоятельной работы

1. Изучить возможности нейрона выполнять логические функции двух переменных: И (AND), И-НЕ (NAND), ИЛИ (OR), ИЛИ-НЕ (NOR), представленные в табл. 3.2.

Таблица 3.2 – Логические функции

Входы		Логические функции			
x1	x2	AND	NAND	OR	NOR
0	0	0	1	0	1
0	1	0	1	1	0
1	0	0	1	1	0
1	1	1	0	1	0

2. В вариантах заданий (табл. 3.3) приведена проверочная выборка. Координаты точек обучающей выборки должны подобрать самостоятельно. Для этого выполните следующие действия:

a. Точки проверочного множества нанесите на график.

b. Постройте вокруг каждой из этих точек не менее 3 точек обучающей выборки.

c. Координаты этих точек запишите в массив обучающей выборки.

Таблица 3.3 – Варианты заданий

№ варианта	Координаты точек проверочного множества и номер класса, к которому принадлежит каждая точка
1	$A_1[0;0]$ – 1 класс; $A_2[1;1]$ – 1 класс; $A_3[-2;-1]$ – 2 класс
2	$A_1[2;1]$ – 1 класс; $A_2[1;0]$ – 1 класс; $A_3[-1;-1]$ – 2 класс
3	$A_1[0;0]$ – 1 класс; $A_2[2;0]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
4	$A_1[0;0]$ – 1 класс; $A_2[1;3]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
5	$A_1[-1;0]$ – 1 класс; $A_2[1;1]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
6	$A_1[0;-1]$ – 1 класс; $A_2[1;1]$ – 2 класс; $A_3[-2;-1]$ – 1 класс
7	$A_1[0;0]$ – 1 класс; $A_2[2;1]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
8	$A_1[0;-1]$ – 1 класс; $A_2[1;1]$ – 2 класс; $A_3[-2;-2]$ – 1 класс
9	$A_1[0;0]$ – 1 класс; $A_2[1;1]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
10	$A_1[-2;0]$ – 1 класс; $A_2[1;1]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
11	$A_1[2;0]$ – 1 класс; $A_2[-1;0]$ – 2 класс; $A_3[1;2]$ – 1 класс
12	$A_1[-1;1]$ – 1 класс; $A_2[2;1]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
13	$A_1[0;-2]$ – 1 класс; $A_2[1;2]$ – 2 класс; $A_3[-1;-1]$ – 1 класс
14	$A_1[0;0]$ – 1 класс; $A_2[1;1]$ – 2 класс; $A_3[-2;-1]$ – 1 класс
15	$A_1[-1;-2]$ – 1 класс; $A_2[1;2]$ – 2 класс; $A_3[-1;-1]$ – 1 класс

3. Применив функцию plotpv, продемонстрировать обучающую выборку.

4. С помощью функции perceptron построить нейронную сеть, которая производит классификацию на заданное количество классов. На составленной обучающей выборке для обучения нейронной сети воспользоваться функцией adapt. Показывать изменение линий разбиения на классы с помощью функции plotpc.

5. Для обученной нейронной сети, произвести классификацию проверочного множества. Привести график классификации точек (plotpc, plotpv).

6. Создать обучающие выборки по 15 точек для 4 классов. Решить задачу классификации в двумерном пространстве на 4 класса.

ЛАБОРАТОРНАЯ РАБОТА 4. НЕЙРОСЕТЕВОЕ РАСПОЗНАВАНИЕ ИЗОБРАЖЕНИЙ

Цель работы: исследование возможностей распознавания изображений с помощью нейронных сетей в программной среде MatLab R2021a.

Краткие теоретические сведения

Рассмотрим пример решения задачи распознавания образов с использованием многослойного персептрона. В качестве набора эталонных образов будем брать последовательность из десяти цифр от 0 до 9. Число образов N = 10 и число классов также будет равно 10.

4.1 Подготовка эталонных образов

Каждый образ формируется в виде графического файла в битовом формате (рис. 4.1). Для создания графических файлов используем редактор Paint. При создании каждого файла необходимо выполнить следующие действия:

– задать высоту N2 и ширину N1 рисунка (например, N1 = 10 и N2 = 12 пикселей);

– используя инструменты графического редактора, нарисовать требуемый образ символа;

– сохранить образ, в виде файла типа *tif* (*n.tif*, где *n*-номер класса образов).

Пример программы, создающей зашумленные изображения приведен в листинге 4.1.

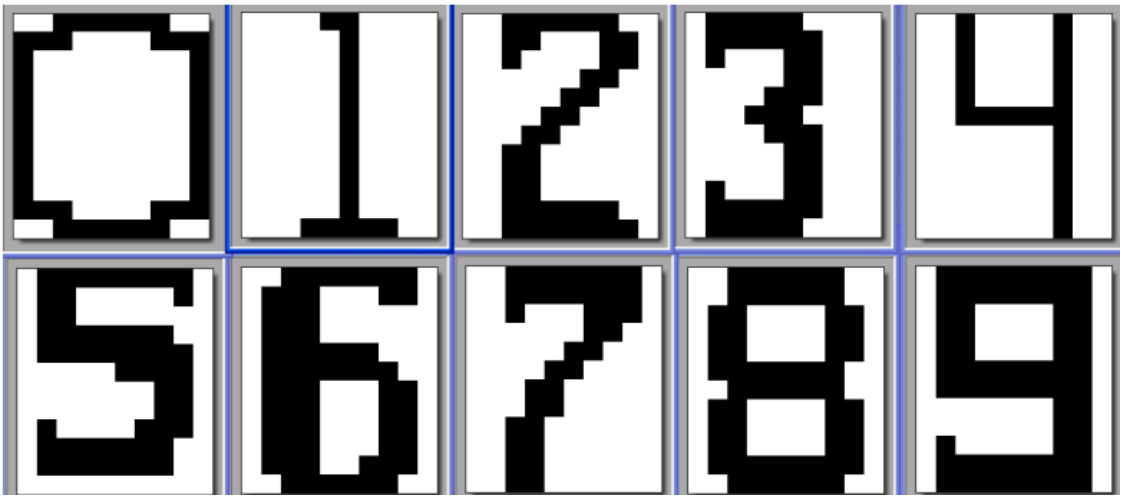


Рисунок 4.1 – Примеры графических символов цифр

Листинг 4.1 – Создание зашумленных изображений

```
CharsetNames=['0_char.tif';'1_char.tif';'2_char.tif';'3_char.tif';
...
'4_char.tif';'5_char.tif';'6_char.tif';'7_char.tif';'8_char.tif';.
..
'9_char.tif'];
for i=1:size(CharsetNames)
A=CharsetNames(i,:);
X=imread(A)
for j=1:50
B=rand(size(X));
C=X+((B>0.7));
NewName=strcat(int2str(j),'_',A) ;
imwrite(C,NewName);
end
end
```

Вероятность зашумления в одном пикселе задается строкой C = X + (B > 0.7). Причем, чем больше число в скобке, тем ниже вероятность шума.

4.2 Создание обучающей выборки и обучение нейросети

Эталонный образ каждого символа представляется в виде вектора столбца, размерности N (листинг 4.2). Он создается при помощи команды imread. при считывании графического файла из двумерного массива-изображения [N1, N2]. Команда A = imread(имя_файла) считывает изображение из файла.

Процедура reshape используется для преобразования двумерного массива в одномерный вектор-столбец. В общем случае, B = reshape(A,sz) изменяет форму A, используя вектор размера sz, чтобы определить размер B. Например, reshape(A,[2,3]) преобразует A в матрицу размером 2 на 3.

Контрастность изображения обеспечивает использование векторов и

матриц типа logical.

Листинг 4.2 – Считывание обучающих образов

```
% Чтение данных
CharsetNames=['0_char.tif';'1_char.tif';'2_char.tif';'3_char.tif';
...
'4_char.tif';'5_char.tif';'6_char.tif';'7_char.tif';'8_char.tif';.
..
'9_char.tif'];
k=0;
N1=size(CharsetNames)
N2=N1(1)
for i=1:N2
A=CharsetNames(i,:);
for j=1:20
k=k+1;
ReadName=strcat(int2str(j),'_',A);
X=logical(imread(ReadName));
L=size(X);
N=L(1)*L(2);
XR(k,:)=reshape(X,[N,1])';
Y0=zeros(N2,1);
Y0(i)=1;
YR(k,:)=logical(Y0')
end
end
```

Чтобы создать и обучить нейронную сеть (листинг 4.3) необходимо:

– сформировать двумерный массив обучающих образов, каждый столбец которого представляет собой набор N признаков одного образа, число столбцов K равно числу обучающих образов;

– сформировать двумерный массив целевых откликов YR;

– использовать для создания нейронной сети команду feedforwardnet;

– обучить нейросеть, используя команду train.

Выходной слой содержит число нейронов равное числу классов классификации.

Листинг 4.3 – Создание и обучение нейронной сети

```
InData=double(XR');
OutData=double(YR');
nHidden=50;
Net = feedforwardnet(nHidden);
Net.performFcn = 'sse'; % sse квадратичная ошибка
Net.trainParam.goal = 1e-3;
Net.trainParam.epochs = 5000;
Net.trainParam.mc = 0.95; % Momentum constant.
Net.trainParam.lr = 0.0001;
Net.layers{1}.transferFcn = 'tansig';
Net.layers{2}.transferFcn = 'logsig';
Net.trainFcn = 'traingda';
```

```
Net=init(Net);
Net.IW{1,1}=Net.IW{1,1}+1e-2*randn;
Net.b{1}=Net.b{1}+1e-3*randn;
Net.LW{2,1}=Net.LW{2,1}+1e-2*randn;
Net.b{2}=Net.b{2}+1e-2*randn;
Net = train(Net,XR',YR');
```

В данном коде используется трехслойная нейросеть с одним скрытым слоем с числом нейронов nHidden равным 50.

4.3 Распознавание символов обученной нейросетью

В листинге 4.4 приводится код тестирования работы нейросети при помощи команды YTest = sim(Net,XTest)'. При помощи команды C = X + ((B > 0.9)); можно варьировать степень искажения.

Листинг 4.4 – Распознавание символов

```
for i=1:size(CharsetNames)
for j=1:10
A=CharsetNames(i,:);
X=logical(imread(A));
B=rand(size(X));
C=X+((B>0.9));
L=size(X);
N=L(1)*L(2);
XTest=double(reshape(X,[N,1]));
YTest=sim(Net,XTest)'
end
end
```

Задания для самостоятельной работы

1. Нарисовать графические файлы образов.
2. Создать и обучить нейронную сеть, используя программную среду MatLab.
3. Исследовать, как зависит качество работы сети от искажения символов и изменения числа нейронов в скрытом слое. Оценить работу сети, используя формулу:

$$P(i) = \frac{N_{TRUE}}{N},$$

где N_{TRUE} – число правильных распознаваний образа для i-го класса;

N – общее число распознаваний образов i-го класса.

ЛАБОРАТОРНАЯ РАБОТА 5. НЕЙРОННАЯ СЕТЬ ХОПФИЛДА

Цель работы: создание приложения для распознавания и восстановления графических образов с помощью сети Хопфилда.

Краткие теоретические сведения

5.1 Сеть Хопфилда

Сеть Хопфилда является однослойной сетью, так как в ней используется лишь один слой нейронов, каждый из которых связан со всеми остальными. Архитектура сети изображена на рисунке 5.1.

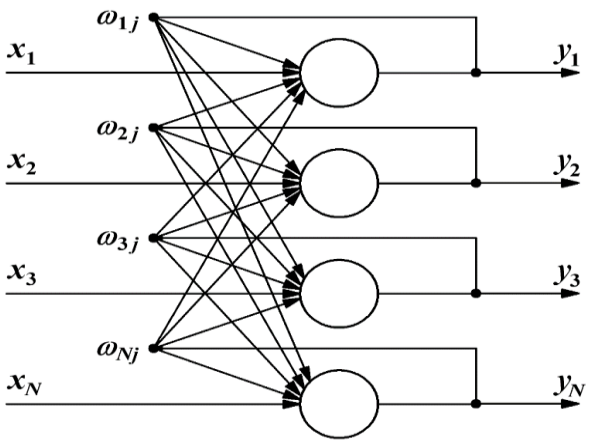


Рисунок 5.1 – Архитектура нейросети Хопфилда

Сеть Хопфилда – это искусственная нейронная сеть с шаговой функцией активации (наиболее часто используется сигнум-функция (5.1)).

$$sign(u) = f(u) = \begin{cases} 1, & \text{если } u \geq 0 \\ -1, & \text{если } u < 0 \end{cases}. \qquad (5.1)$$

Особенностью сети Хопфилда является то, что она может распознать только те образы, которые ей представлены в процессе обучения в качестве тестовых данных. В сети Хопфилда после представления объектов обучения, матрица весов формируется сразу.

В начале обучения всем весам присваивают нулевые значения. Для всех i и j, которые меняются от 1 до N, выполняются следующие действия. Для всех образов выполняем корректировку весов:

$$W_{ij} = W_{ij} + x_i x_j .$$

Далее диагональные элементы зануляют

$$W_{ii} = 0$$

и выполняют нормировку

$$W_{ij} = \frac{W_{ij}}{N}.$$

Наша сеть обучена.

Далее матрица весов W не изменяется. Следует отметить, что для сети Хопфилда матрица весовых коэффициентов симметрична $W_{ij} = W_{ji}$ и имеет нулевую главную диагональ $W_{ii} = 0$, т. е. отсутствует обратная связь нейрона на себя.

Процедуру распознавания образов можно продемонстрировать следующим алгоритмом. Допустим, мы ищем образ:

$$y = (y_1; y_2; ...; y_N).$$

1. Цикл по j от 1 до N:
2. Положим $d = 0$.
3. Вложенный цикл по i от 1 до N.
4. $d = d + W_{ij} \cdot y_i$.
5. Конец вложенного цикла.
6. Если $d > 0$, то положить $z_j = 1$, иначе $z_j = -1$.
7. Конец внешнего цикла.
8. Получен вектор $Z = [Z_1, Z_2, ..., Z_N]$. Если вектор Z содержится во множестве исходных образов, то считаем, что алгоритм нашел образ Z, который соответствует исходному образу Y.
9. Положим $Y = Z$.
10. Переход к шагу 1.

Пусть имеется нейронная сеть размерностью $N = n \times n$ нейронов, которые будут рассматриваться как квадратная матрица из n строк и n столбцов. Одним из недостатков сети Хопфилда – это относительно небольшой объем памяти. Величину которой можно оценить выражением:

$$M \approx \frac{N}{2 \ln N}. \qquad (5.2)$$

При записи большего числа образов нейронная сеть перестанет их распознавать.

5.2 Разработка приложения

Приложение должно обеспечивать выполнение перечисленных ниже функций:

– обеспечение ввода «правильных» образов, с помощью которых должно происходить обучение системы;

– обеспечение возможности просмотра введённых «правильных» образов;

– построение матрицы весовых коэффициентов для сети Хопфилда;

– обеспечение ввода «испорченного» образа;

– нахождение «правильного» образа.

Разработка программного продукта для создания пользовательского интерфейса осуществляется на языке программирования C#. Средой разработки выбран «Visual Studio 2019».

1. Размещаем на форме необходимые элементы: 6 кнопок Button (Name: btnAdd, btnShow, btnRestore, btnClearTable, btnClearImages, Exit) и pictureBox.

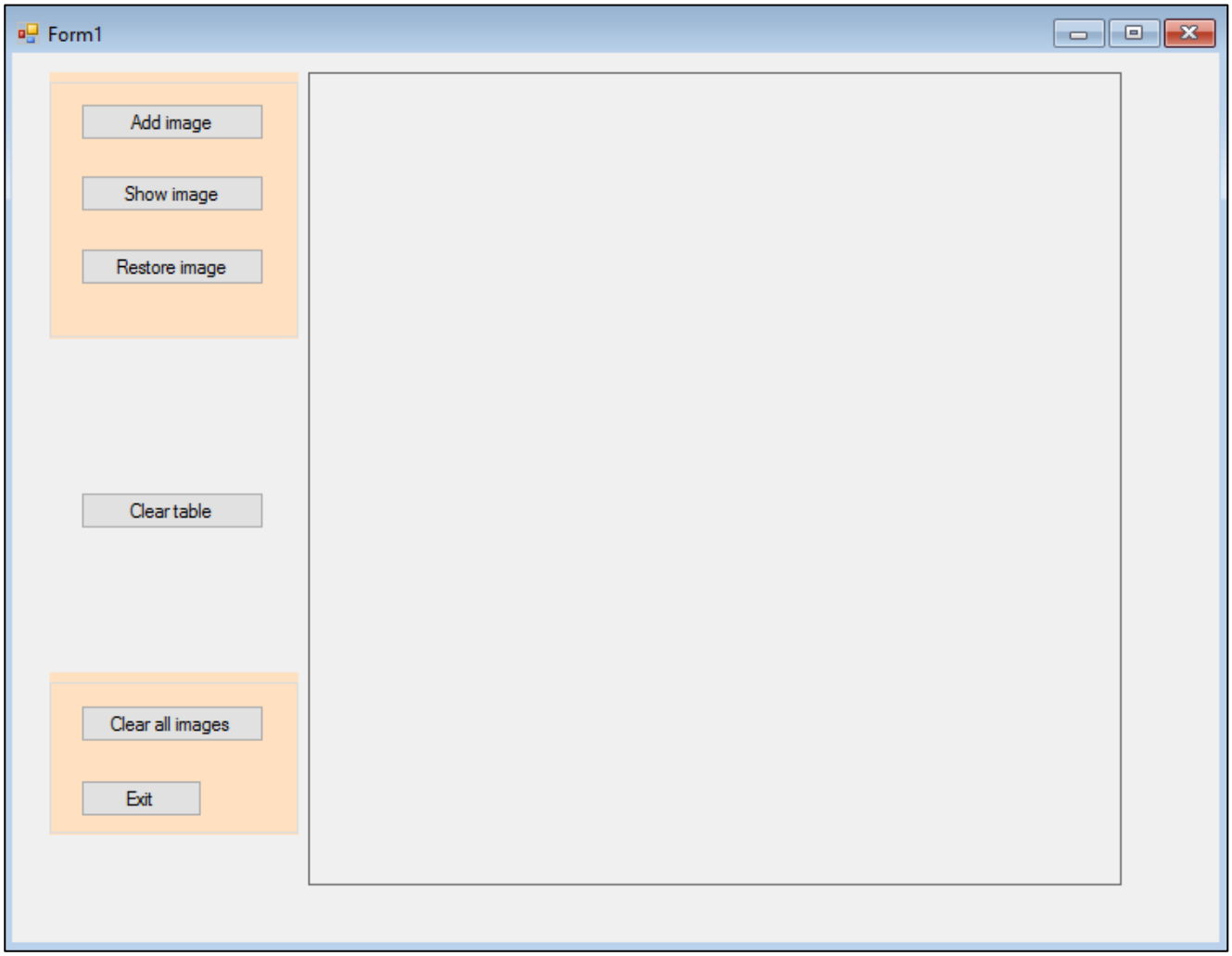


Рисунок 5.2 – Окно формы

2. Алгоритм работы сети Хопфилда реализует класс CalcHopfield (листинг 5.1). Создаем данный класс. Для этого перейдем в пункт меню Проект → Добавить класс (рис. 5.3).

Вводим имя класса и нажимаем кнопку «Добавить». На первом этапе сеть обучается, на предъявляемых ей различных эталонных образах, и формируется матрица весов W. Для этого служит метод void Learning(). Процедура

распознавания образов осуществляется функцией public Shape Find(Shape BS, int T = 1000).

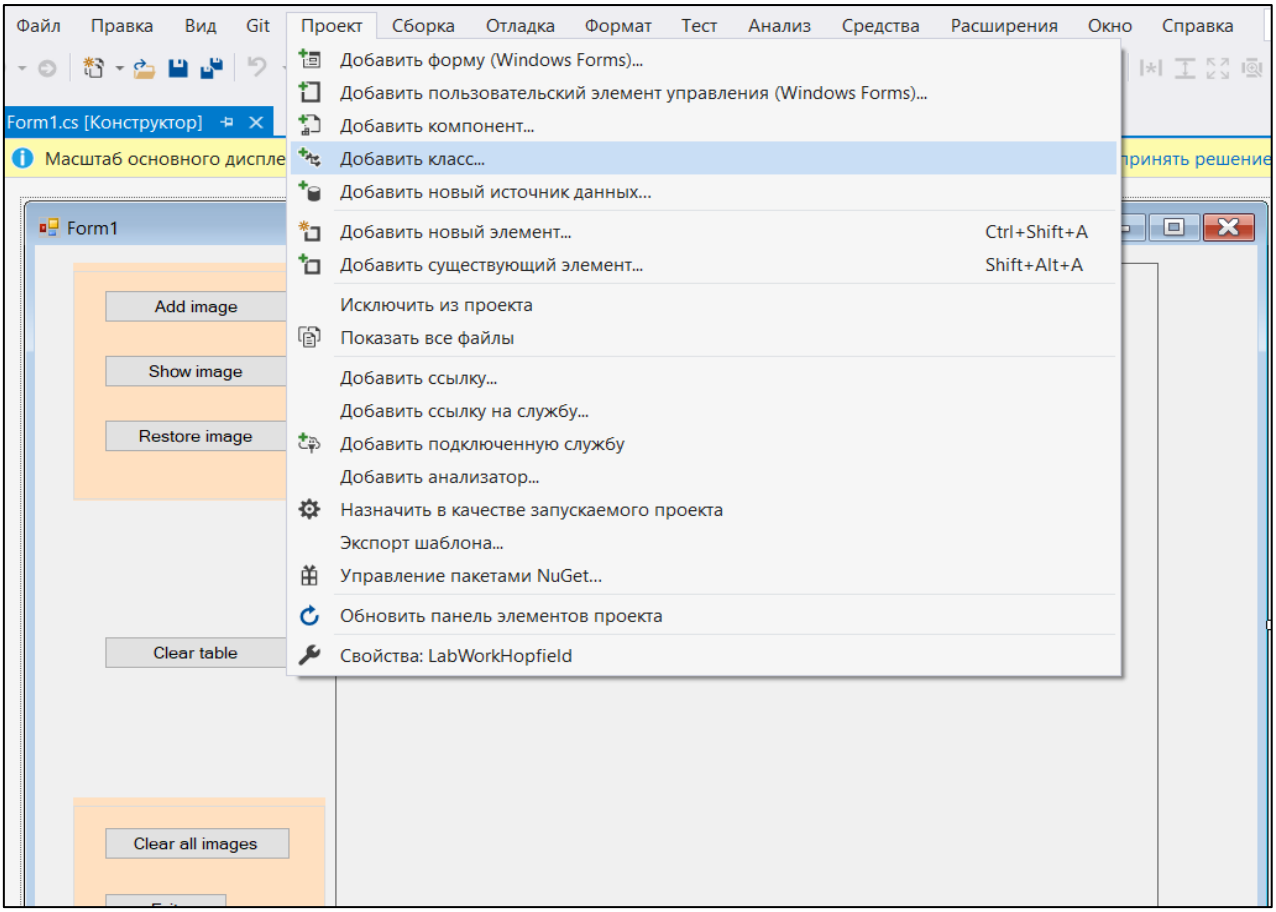


Рисунок 5.3 –Добавление класса

Листинг 5.1 – Класс CalcHopfield

```
namespace LabWorkHopfield
{
    class CalcHopfield
    {
        ShapeList Boxes;
        int N;
        int M;
        double[,] w;
        public CalcHopfield(ShapeList Boxes)
        {
            this.Boxes = Boxes;
            N = Boxes.N;
            M = Boxes.Count;
            w = new double[N, N];

            for (int i = 0; i < N; i++)
            {
                for (int j = 0; j < N; j++)
                {
```

```
                w[i, j] = 0;
            }
        }
        Learning();
    }
    // Обучение сети Хопфилда
    void Learning()
    {
        for (int i = 0; i < N; i++)
        {
            for (int j = 0; j < N; j++)
            {
                if (i == j)
                {
                    continue;
                }

                for (int m = 0; m < M; m++)
                {
                    // корректировка весов
                    w[i, j] += Boxes[m][i] * Boxes[m][j];
                }
                w[i, j] /= N;
            }
        }
    }

    // восстановление образа
    public Shape Find(Shape BS, int T = 1000)
    {
        Shape B = new Shape(BS);
        Shape B2 = new Shape(B);

        for (int t = 0; t < T; t++)
        {
            for (int j = 0; j < N; j++)
            {
                double d = 0;

                for (int i = 0; i < N; i++)
                {
                    d += w[j, i] * B[i];
                }

                if (d > 0)
                {
                    B2[j] = 1;
                }
                else
                {
                    B2[j] = -1;
```

```
                    }
                }
                B = new Shape(B2);

                if (Boxes.Find(B))
                {
                    return B;
                }
            }
            return null;
        }
    }
}
```

3. Создадим класс Shape (листинг 5.2). Данный класс используется для хранения образов. Метод public bool IsEq(Shape B) проверяет на совпадение два образа. Он используется при нахождении оригинала для «испорченного» образа.

Листинг 5.2 – Класс Shape

```
namespace LabWorkHopfield
{
    [Serializable]
    class Shape
    {
        public int NN; // размерность поля для сетки
        public int N; // количество элементов

        public int[] B;

        //костыль чтобы работать с JsonConvert
        private Shape() { }

        public Shape(int NN, int d = -1)
        {
            this.NN = NN;
            N = NN * NN;

            B = new int[N];

            for (int n = 0; n < N; n++)
            {
                B[n] = d;
            }
        }

        public Shape(Shape TB)
        {
            NN = TB.NN;
            N = TB.N;
```

```
            B = new int[N];

            for (int n = 0; n < N; n++)
            {
                B[n] = TB[n];
            }
        }
        //проверяем на совпадение два образа
        public bool IsEq(Shape B)
        {
            for (int i = 0; i < N; i++)
            {
                if (this[i] != B[i])
                {
                    return false;
                }
            }

            return true;
        }
        //индексатор
        public int this[int n]
        {
            get
            {
                return B[n];
            }
            set
            {
                B[n] = value;
            }
        }

        public int this[int i, int j]
        {
            get
            {
                return this[i * NN + j];
            }
            set
            {
                this[i * NN + j] = value;
            }
        }
    }
}
```

4. Создадим класс GridManager (листинг 5.3). Класс GridManager предназначен для изображения образа на компоненте и изменения образа. Образ формируется из закрашенных или не закрашенных ячеек. Данный класс содержит следующие методы:

– public void DrawGrid(Graphics g) рисует сетку;

– public void Check(int x, int y) выделяет ячейку по адресу x, y или снимает с нее выделение, если она выделена. При этом изменяется одномерный массив образа;

– public void DrawRectangles(Graphics g) закрашивает выделенные ячейки;

– public void Draw(Graphics g) рисует сетку и выделенные ячейки (образ).

Листинг 5.3 – Класс GridManager

```
namespace LabWorkHopfield
{
    class GridManager
    //класс для отображения образа на компоненте и изменения образа
    {
        //рисует сетку, закрашиваем ячейки
        private Shape shape;
        Rectangle?[,] cells;
        int NN;
        int width;
        private int height;
        Point[] p1 = new Point[10];
        Point[] p2 = new Point[10];
        //конструктор
        public GridManager(int dimention, int width, int height)
        {
            this.NN = dimention;
            this.width = width;
            this.height = height;
            cells = new Rectangle?[NN, NN];
         }

        public void DrawGrid(Graphics g)
        //рисует сетку
        {

            double dx = width / NN; // ширина в пикселях клетки
            Pen pn = new Pen(Color.Red, 4);

            // вертикальные линии
            for (int k = 0; k <= NN; k++)
            {
                p1[k] = new Point(k * (int)dx, 0);
                p2[k] = new Point(k * (int)dx, height);
                g.DrawLine(pn, p1[k], p2[k]);
            }
            //горизонтальные линии
            for (int k = 0; k <= NN; k++)
            {
                p1[k] = new Point(0, k * (int)dx);
                p2[k] = new Point(width, k * (int)dx);
                g.DrawLine(pn, p1[k], p2[k]);
```

```
            }
        }

        public void Check(int x, int y)
        //выделяет ячейку по адресу x, y или снимает с нее выделение,
если она выделена.
        {
            // отметить клетку с заданными координатами
            double dx = width / NN; // ширина в пикселях клетки
            // проверить диапазон
            if ((x < 0) || (x > width)) { return; }
            if ((y < 0) || (y > height)) { return; }

            // рассчитать номер ячейки
            int i = Convert.ToInt16(Math.Floor(x / dx));
            int j = Convert.ToInt16(Math.Floor(y / dx));
            if (cells[i, j] == null)
            {
                shape[i, j] = 1;
                cells[i, j] = new Rectangle(i * (int)dx, j * (int)dx,
(int)dx, (int)dx);
            }
            else
            {
                cells[i, j] = null;

                shape[i, j] = -1;
            }
        }

        public void DrawRectangles(Graphics g)
        //закрашивает выделенные ячейки;
        {
            for (int i = 0; i < NN; i++)
                for (int j = 0; j < NN; j++)
                    if (cells[i, j] != null)
                        g.FillRectangle(Brushes.Blue, (Rectangle)cells[i,
j]);
        }

        public void Draw(Graphics g)
        //рисует сетку и выделенные ячейки(образ).
        {
            DrawRectangles(g);
            DrawGrid(g);
        }

        //доступ к shape
        public Shape Shape
        {
            get { return shape; }
```

```
            set
            {
                shape = value;

                double dx = width / NN; // ширина в пикселях ячейки
                for (int i = 0; i < NN; i++)
                {
                    for (int j = 0; j < NN; j++)
                    {
                        if (shape[i, j] > 0)
                        {
                            cells[i, j] = new Rectangle(i * (int)dx, j *
(int)dx, (int)dx, (int)dx);
                        }
                        else
                            cells[i, j] = null;
                    }
                }
            }
        }
    }
}
```

5. Создадим класс ShapeList (листинг 5.4). Класс ShapeList хранит набор образов (оригиналов). Метод public void Add(Shape B) добавляет образ и public bool Find(Shape B) находит образ в наборе.

Листинг 5.4 – Класс ShapeList

```
namespace LabWorkHopfield
{
    // хранит образы
    [Serializable]
    class ShapeList
    {
        public int NN;
        ArrayList arr;

        public ShapeList(int NN)
        {
            this.NN = NN;
            arr = new ArrayList();
        }

        public void Add(Shape B)
        //добавляет образ
        {
            arr.Add(B);
        }

        public bool Find(Shape B)
```

```
        //находит образ в наборе
        {
            for (int n = 0; n < Count; n++)
            {
                if (B.IsEq(this[n]))
                {
                    return true;
                }
            }
            return false;
        }

        public int N
        {
            get
            {
                return NN * NN;
            }
        }

        public Shape this[int k]
        {
            get
            {
                return (Shape)arr[k];
            }
        }

        public int Count
        {
            get
            {
                return arr.Count;
            }
        }
    }
}
```

6. Листинг для Form1.cs приведен ниже (листинг 5.5). В Windows Forms для взаимодействия с пользователем используется механизм событий. Чтобы создать событие формы private void Form1_Load(object sender, EventArgs e): мы должны выделить форму, затем открыть вкладку события и нажать на Load два раза.

В файле кода формы Form1.cs появиться автосгенерированный метод Form1_Load. Поэтому при загрузке формы будет срабатывать код в обработчике Form1_Load. Добавление обработчика, созданного таким образом, производится в файле Form1.Designer.cs. Для добавления обработчика используется стандартный синтаксис C#:

this.Load += new System.EventHandler(this.Form1_Load)

Чтобы удалить созданный обработчик, необходимо удалить добавление обработчика и метод из кода формы в Form1.cs.

Листинг 5.5 – Код Form1.cs

```
namespace LabWorkHopfield
{
    public partial class Form1 : Form
    {
        const int NN = 5;//число строк или столбцов
        GridManager gridManager;
        ShapeList shapeList;
        CalcHopfield hopfield;
        public Form1()
        {
            InitializeComponent();
            gridManager = new GridManager(NN, pictureBox1.Width,
pictureBox1.Height);
            gridManager.Shape = new Shape(NN);

            double dx = pictureBox1.Width / NN; // ширина в пикселях
ячейки
            shapeList = new ShapeList(NN);
            hopfield = new CalcHopfield(shapeList);

            ris();
        }

        private void Form1_Load(object sender, EventArgs e)
        {
             LoadImages();
        }

        public void ris()
        {
                        pictureBox1.Invalidate();
        }

        private void pictureBox1_MouseDown(object sender, MouseEventArgs
e)
        {
            gridManager.Check(e.Location.X, e.Location.Y);
            ris();
        }

        private void btnAdd_Click(object sender, EventArgs e)
        {
            LoadImages();
            shapeList.Add(gridManager.Shape);
```

```
        SaveImages();
    }
    int currentShape;

    private void btnShow_Click(object sender, EventArgs e)
    {
        if (shapeList.Count == 0)
        {
            return;
        }

        if (currentShape < shapeList.Count)
        {
        }
        else
        {
            currentShape = 0;
        }
        gridManager.Shape = new Shape(shapeList[currentShape]);
        currentShape++;

        ris();
     }

    private void btnRestore_Click(object sender, EventArgs e)
    {
        if (shapeList.Count < 1)
        {
            return;
        }

        CalcHopfield hopfield = new CalcHopfield(shapeList);
        Shape shape = hopfield.Find(gridManager.Shape, 100);
        if (shape == null)
        {
            MessageBox.Show("Образ не найден!");
        }
        else
        {
            gridManager.Shape = shape;
            ris();
         }
    }

    private void btnClearTable_Click(object sender, EventArgs e)
    {
        gridManager.Shape = new Shape(NN);
        ris();
     }
```

```
        private void Exit_Click(object sender, EventArgs e)
        {
            this.Close();
        }

        private void pictureBox1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            g.Clear(Color.Coral);
            gridManager.Draw(g);
        }

        void SaveImages()
        {
            BinaryFormatter formatter = new BinaryFormatter();

            using (FileStream fs = new FileStream("Data.bin",
FileMode.OpenOrCreate))
            {
                formatter.Serialize(fs, shapeList);
            }
        }

        void LoadImages()
        {
            BinaryFormatter formatter = new BinaryFormatter();

            if (File.Exists("Data.bin"))
            {
                using (FileStream fs = new FileStream("Data.bin",
FileMode.OpenOrCreate))
                {
                    shapeList = (ShapeList)formatter.Deserialize(fs);
                }

                if (shapeList.NN == NN)
                {
                    return;
                }
            }

            shapeList = new ShapeList(NN);

            using (FileStream fs = new FileStream("Data.bin",
FileMode.OpenOrCreate))
            {
                formatter.Serialize(fs, shapeList);
            }
        }
```

```
        private void btnClearImages_Click(object sender, EventArgs e)
        {
            shapeList = new ShapeList(NN);
            SaveImages();
        }
    }
}
```

5.3 Тестирование приложения

Любое приложение должно обеспечивать бесперебойную работу в любых ситуациях. Кроме того, приложение должно учитывать вероятные ошибки пользователя при работе с данными и по возможности предотвращать их.

Основное окно для распознавания образов, показано на рис. 5.4.

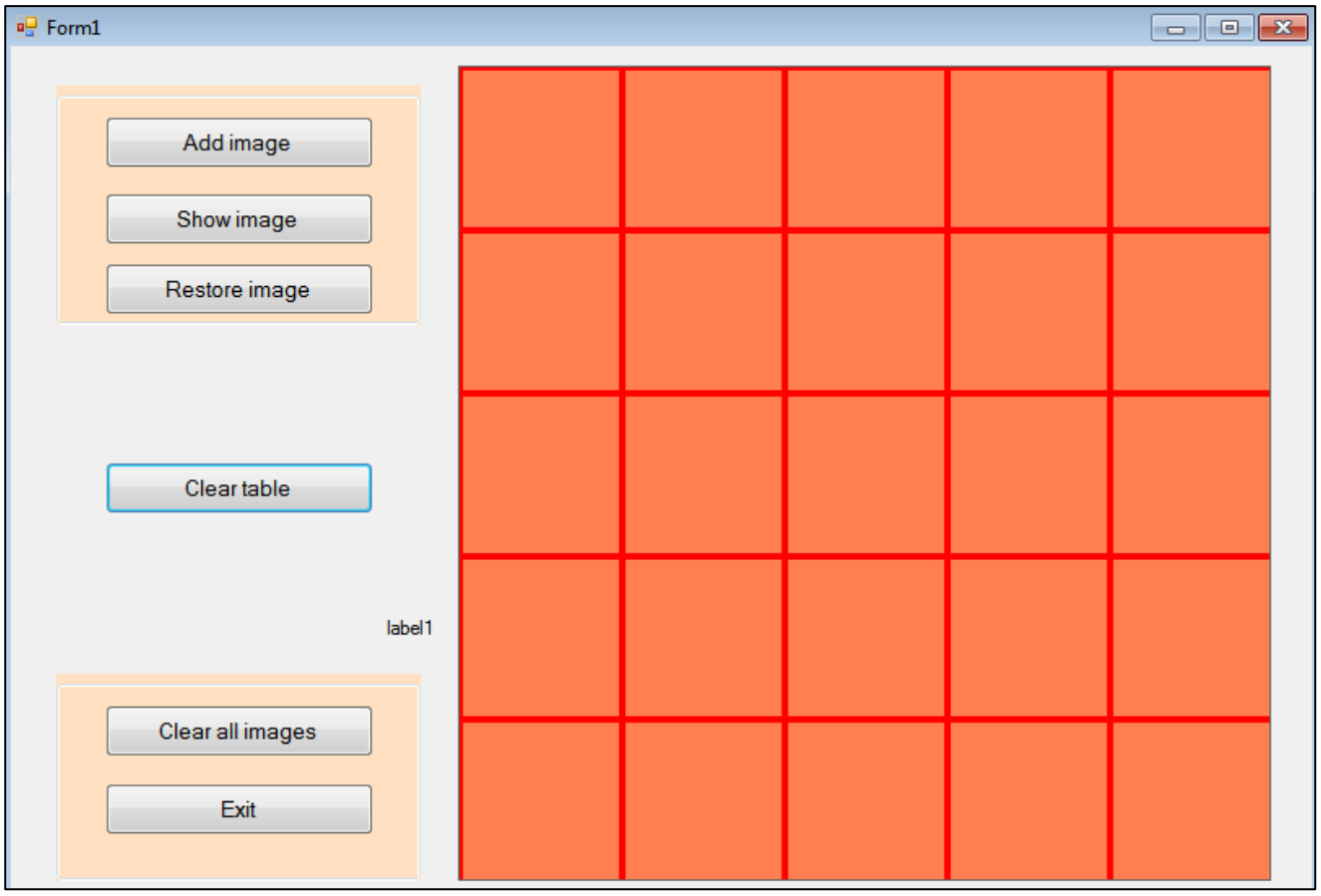


Рисунок 5.4 – Окно для распознавания образов

Изобразив в поле pictureBox1, желаемый объект, мы кликаем по кнопке Add image и добавляем объект в массив. Добавив несколько объектов, мы можем их просмотреть, последовательно кликая по кнопке Show Image (рис. 5.5).

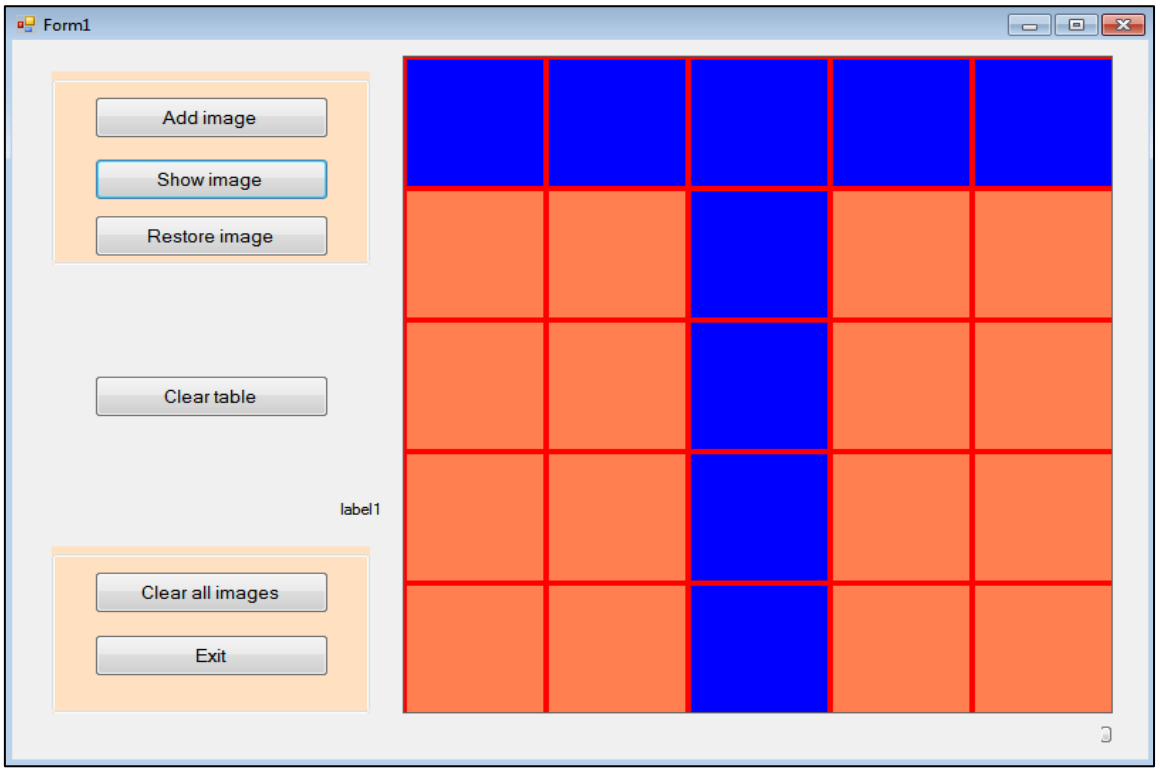


Рисунок 5.5 – Пример образа

Когда мы предъявляем зашумленный образ (рис. 5.6), то сеть Хопфилда может распознать исходный образ, который ей был представлен в процессе обучения. Для этого мы должны кликнуть по кнопке Restore image.

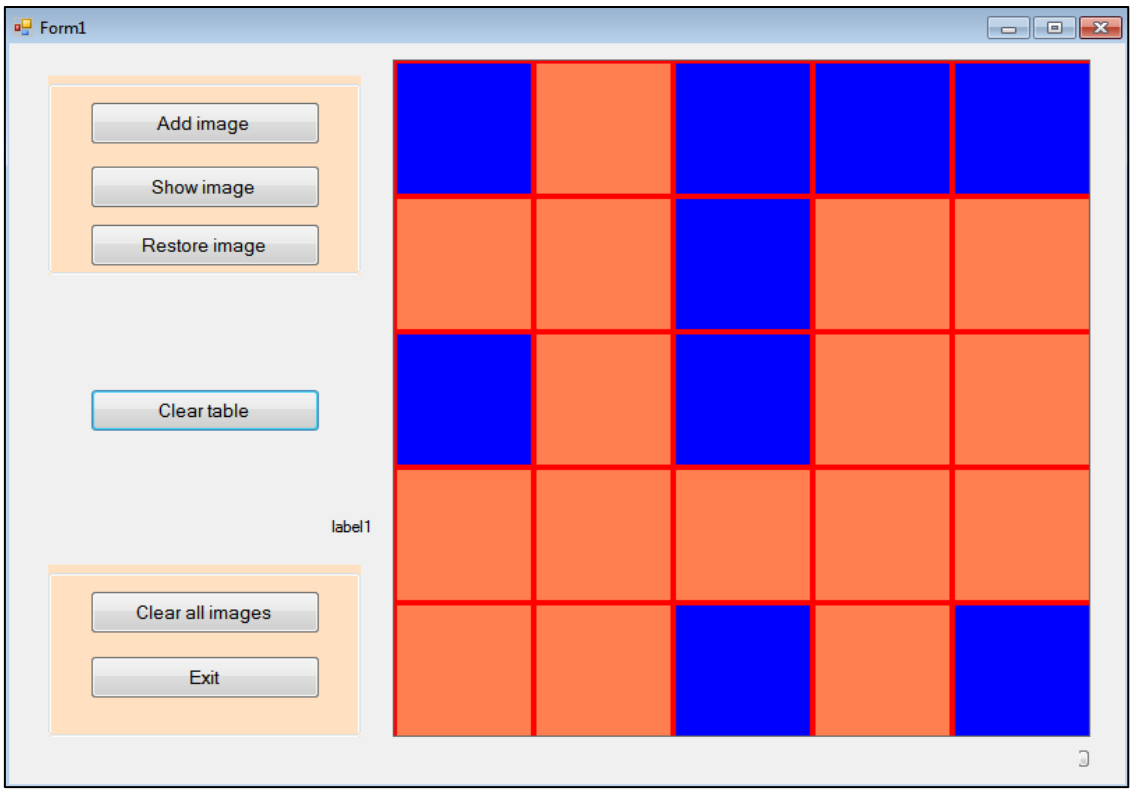


Рисунок 5.6 – Пример зашумленного образа

Задания для самостоятельной работы

1. Запустить приложение для моделирования работы сети Хопфилда по распознаванию образов.

2. Меняя число строк и столбцов в приложении проверить зависимость между количеством элементов сети N и количеством образцов, которые она может запомнить.

ЛАБОРАТОРНАЯ РАБОТА 6. ГЕТЕРОАССОЦИАТИВНАЯ ПАМЯТЬ

Цель работы: изучение алгоритма обучения и функционирования нейронной сети Коско.

Краткие теоретические сведения

Человеческая память обладает возможностью ассоциации каких-либо объектов с другими. То есть у нас, например, появляются образы, мысли, когда мы видим определенные слова. Нейронная сеть гетероассоциативной памяти, разработанная Бартом Коско, имитирует эту особенность человеческой памяти.

На рисунке 6.1 изображена структура этой сети, содержащей два слоя нейронов. В первом слое количество нейронов K равно количеству бинарных выходных переменных. Во втором слое число нейронов M равно числу бинарных переменных, кодирующие входной образ, соответствующий выходному.

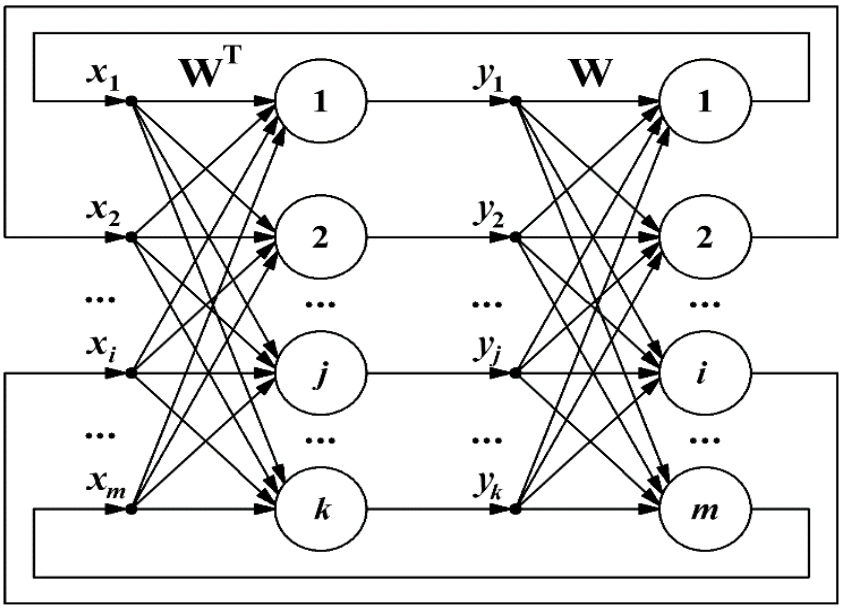
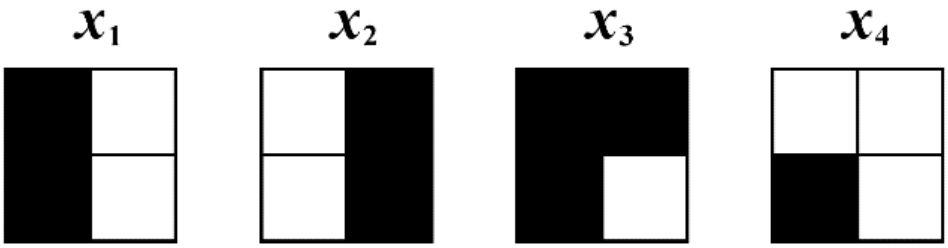


Рисунок 6.1 – Структура двухслойной гетероассоциативной нейронной сети

Алгоритм нейронной сети Коско включает две стадии: обучения и практического использования.

Рассмотрим пример. Пусть дан набор векторов для 4 образов:



В биполярной кодировке матрицы выходных образов мы можем записать в виде:

$$x_1 = (1;-1;1;-1)$$
$$x_2 = (-1;1;-1;1)$$
$$x_3 = (1;1;1;-1)$$
$$x_4 = (-1;-1;1;-1)$$

Индексы в полярной кодировке в двоичной системе координат:

$$1 \to 0\ 0\ 1$$
$$2 \to 0\ 1\ 0$$
$$3 \to 0\ 1\ 1 \ .$$
$$4 \to 1\ 0\ 0$$

А в биполярной кодировке выходные образы примут вид:

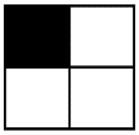
$$1 \to -1\ -1\ 1$$
$$2 \to -1\ 1\ -1$$
$$3 \to -1\ 1\ 1$$
$$4 \to 1\ -1\ -1$$

Поставим каждой картинке в соответствие индекс. Входной вектор содержит 4 элемента, выходной – 3 элемента.

Воспользуемся для тестирования сети зашумленным входным образом:

$$x_q = (1;-1;-1;-1)\,.$$

x_q



Мы должны определить номер q.

На стадии обучения рассчитываются элементы ω_{ij} матрицы весовых коэффициентов *W:*

$$W = X^T Y .$$

Для четырех образов:

$$\begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \begin{pmatrix} -1 & -1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix};$$

$$\begin{pmatrix} -1 \\ 1 \\ -1 \\ 1 \end{pmatrix} \begin{pmatrix} -1 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{pmatrix};$$

$$\begin{pmatrix} 1 \\ 1 \\ 1 \\ -1 \end{pmatrix} \begin{pmatrix} -1 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \end{pmatrix};$$

$$\begin{pmatrix} -1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \begin{pmatrix} 1 & -1 & -1 \end{pmatrix} = \begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix}.$$

Просуммировав полученные матрицы, вычислим матрицу весовых коэффициентов:

$$W = \begin{pmatrix} -1 & -1 & 1 \\ 1 & 1 & -1 \\ -1 & -1 & 1 \\ 1 & 1 & -1 \end{pmatrix} + \begin{pmatrix} 1 & -1 & 1 \\ -1 & 1 & -1 \\ 1 & -1 & 1 \\ -1 & 1 & -1 \end{pmatrix} + \begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \end{pmatrix} + \begin{pmatrix} -1 & 1 & 1 \\ -1 & 1 & 1 \\ 1 & -1 & -1 \\ -1 & 1 & 1 \end{pmatrix}.$$

Получим:

$$W = \begin{pmatrix} -2 & 0 & 4 \\ -2 & 4 & 0 \\ 0 & -2 & 2 \\ 0 & 2 & -2 \end{pmatrix}.$$

Процесс обучения на этом заканчивается. Далее матрица весовых коэффициентов изменяться не будет. В качестве функции активации возьмем:

$$sign(u) = f(u) = \begin{cases} 1, & \text{если } u > 0 \\ -1, & \text{если } u \le 0 \end{cases}.$$

Далее выполняем следующие действия:

1. Подаем зашумленный образ в нейронную сеть как значения элементов входного вектора-столбца. Для первого слоя рассчитываем состояния нейронов:

$$s_1^{(q+1)} = W^T x_q.$$

2. Применяем функцию активации, и получаем новый вектор выходных значений $y^{(q+1)}$.

3. Проверяем условие окончания (для всех итераций, кроме первой) – стабилизация элементов выходного вектора $\left| y^{(k)} - y^{(k-1)} \right| = 0$.

Если условие выполнено, то переходим к п. 6, иначе – цикл расчетов продолжается.

4. Вычисленные значения для элементов выходного образа подаются на входы нейронов второго слоя. Рассчитываются новые состояния нейронов:

$$s_2^{(q+1)} = W y^{(k+1)}.$$

5. Применяем функцию активации и вычисляем элементы нового вектора входного образа x_{q+1}.

6. Если установлено соответствие стабилизировавшегося вектора выходных значений одному из идеальных образов матрицы Y, то делается вывод, что нейросети удалось найти ассоциацию между зашумленным входом и одним из идеальных выходов. Если входной вектор был сильно зашумлен, то соответствие не будет установлено.

Например, вычислим $s_1^{(1)} = W^T x_q$ (итерация 1):

$$s_1^{(1)} = W^T x_q = \begin{pmatrix} -2 & -2 & 0 & 0 \\ 0 & 4 & -2 & 2 \\ 4 & 0 & 2 & -2 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ -1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ -4 \\ 4 \end{pmatrix}.$$

Применяем активационную функцию:

$$y_1^{(1)} = f \begin{pmatrix} 0 \\ -4 \\ 4 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}.$$

Вычисленные значения подаются на входы нейронов второго слоя:

$$s_2^{(1)} = W y_1^{(1)};$$

$$W y_1^{(1)} = \begin{pmatrix} -2 & 0 & 4 \\ -2 & 4 & 0 \\ 0 & -2 & 2 \\ 0 & 2 & -2 \end{pmatrix} \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix} = \begin{pmatrix} 6 \\ -2 \\ 4 \\ -4 \end{pmatrix}.$$

Применяем активационную функцию:

$$x_q^{(1)} = f \begin{pmatrix} 6 \\ -2 \\ 4 \\ -4 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix}.$$

Итерация 2:

$$s_1^{(2)} = W^T x_q^{(1)} = \begin{pmatrix} -2 & -2 & 0 & 0 \\ 0 & 4 & -2 & 2 \\ 4 & 0 & 2 & -2 \end{pmatrix} \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} = \begin{pmatrix} 0 \\ -8 \\ 8 \end{pmatrix};$$

$$y_1^{(2)} = f \begin{pmatrix} 0 \\ -8 \\ 8 \end{pmatrix} = \begin{pmatrix} -1 \\ -1 \\ 1 \end{pmatrix}.$$

На второй итерации наступила стабилизация выходного вектора. Она оказалась идентичной первому выходному эталонному образу $(q = 1)$.

Задания для самостоятельной работы

1. Даны входные и выходные образы.

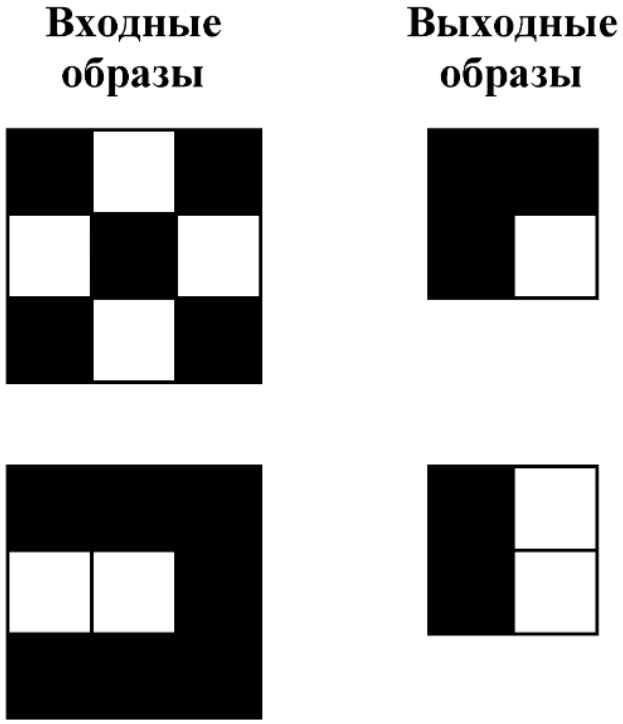


Рисунок 6.2 – Эталонные входные и выходные образы для обучения сети Коско

Согласно номеру в журнале выберите зашумленный графический образ и выполните тестирование сети Коско. Определите, какой выходной образ ему соответствует. Решение проводить до стабилизации выходного вектора.

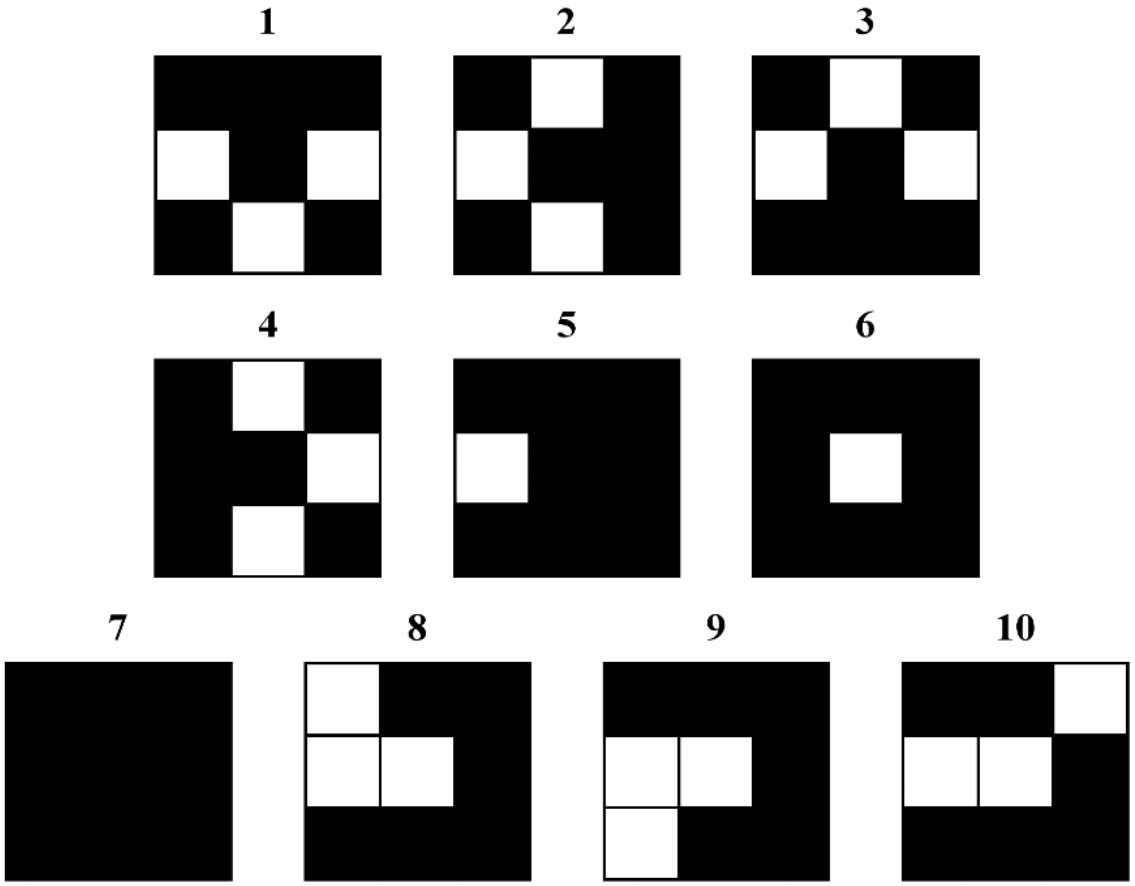


Рисунок 6.3 – Зашумленный входной графический образ

Замечание: при решении задачи воспользоваться функцией активации:

$$sign(u) = f(u) = \begin{cases} 1, \text{ если } u > 0 \\ -1, \text{ если } u \le 0 \end{cases}.$$

ЛАБОРАТОРНАЯ РАБОТА 7. АЛГОРИТМ ОБРАТНОГО РАСПРОСТРАНЕНИЯ ОШИБКИ

Цель работы: разработка и исследование алгоритма обратного распространения ошибки.

Краткие теоретические сведения

7.1 Алгоритм обратного распространения ошибки

Алгоритм обратного распространения ошибки – один из алгоритмов, предусматривающих обучение со скрытыми слоями. Суть метода состоит в

распространении сигналов ошибки от выходов сети к её входам, в направлении, обратном прямому распространению сигналов в обычном режиме работы.

Возьмем в качестве функции активации сигмоидальную функцию вида:

$$f_\sigma(S) = \frac{1}{1+e^{-S}}.$$

В процессе решения задачи мы должны изменить веса связей так, чтобы выход сети стал как можно ближе к требуемому значению. Таким образом, сеть должна давать определенный результат при предъявлении ей некоторых данных.

Для тестирования возьмем, например, медицинскую тематику. Пусть на входе будет 4 нейрона: T, A, S, F, скрытый слой и один выходной нейрон.
Входные нейроны:

– Temperature (принимает значение 0 если нормальная и 1 если повышенная);

– Age (возраст до 25 принимает значение 0);

– Smoking (если не курит, то 0, если курит – 1);

– Food (если правильно питается, то 0, если неправильно – 1).

Ошибка будет находиться как:

Error = actual-expected

Поправка к весам будет находиться по следующей формуле:

$$delta = \Delta\omega_{ij} = \alpha(d_i - y_i) f'_\sigma(S_i) x_j,$$

где $f'_\sigma(S_i)$ – производная сигмоидальной функции.

7.2 Разработка приложения

Разработка программного продукта для создания пользовательского интерфейса осуществляется на языке программирования C#. Средой разработки выбран «Visual Studio 2019».

1. Запустим MS Visual Studio. Выберем «Создать проект» (New Project). Тип приложения – Windows Forms. В поле ввода «Расположение» (Location) выберем рабочую папку, в которой будут храниться файлы проекта. Введем имя проекта и решения. Откроется окно конструктора формы Form1 [Design].

2. Создайте класс Neuron (листинг 7.1). Для этого выберем пункт меню Проект → Добавить класс.

Листинг 7.1 – Класс Neuron

```
namespace NNBackpropagation
{
    public class Neuron
    //для одного нейрона
    {
        public List<double> Weights { get; }
```

```
        public List<double> Inputs { get; }

        public NeuronType NeuronType { get; }

        public double Output { get; private set; }
        public double Delta { get; private set; }

        public Neuron(int inputCount, NeuronType type =
NeuronType.Normal)
        //inputCount количество входных нейронов
        {
            NeuronType = type;
            Weights = new List<double>();
            Inputs = new List<double>();

            InitWeightsRandomValue(inputCount);
        }

        private void InitWeightsRandomValue(int inputCount)
        {
            var rnd = new Random();

            for (int i = 0; i < inputCount; i++)
            {
                if (NeuronType == NeuronType.Input)
                {
                    Weights.Add(1);
                }
                else
                {
                    Weights.Add(rnd.NextDouble());
                }
                Inputs.Add(0);
            }
        }

        public double FeedForward(List<double> inputs)
        {
            for (int i = 0; i < inputs.Count; i++)
            //должны сохранить входные сигналы
            {
                Inputs[i] = inputs[i];
            }
            var sum = 0.0;
            for (int i = 0; i < inputs.Count; i++)
            {
                sum += inputs[i] * Weights[i];
            }

            if (NeuronType != NeuronType.Input)
            {
```

```
                Output = Sigmoid(sum);
            }
            else
            {
                Output = sum;
            }
            return Output;
        }

        private double Sigmoid(double x)
        {
            var result = 1.0 / (1.0 + Math.Pow(Math.E, -x));
            return result;
        }

        private double SigmoidDx(double x)
        {
            //производная сигмоидной функции
            var sigmoid = Sigmoid(x);
            var result = sigmoid * (1 - sigmoid);
            return result;
        }

        public void Learn(double error, double learningRate)
        {
            //метод для вычисления новых весов
            if (NeuronType == NeuronType.Input) //если входной не обучаем
            {
                return;
            }
            Delta = error * SigmoidDx(Output);
            for (int i = 0; i < Weights.Count; i++)
            {
                var weight = Weights[i];
                var input = Inputs[i];
                var newWeigth = weight - input * Delta * learningRate;
                //новый вес
                Weights[i] = newWeigth;
            }
        }

        public override string ToString()
        {
            return Output.ToString();
        }
    }
}
```

3. Создадим перечисления. Для этого создадим класс с именем NeuronType, а затем заменим ключевое слово class на enum (листинг 7.2).

Листинг 7.2 – Перечисления enum NeuronType

```
namespace NNBackpropagation
{
    public enum NeuronType
    {
        Input = 0,
        Normal = 1,
        Output = 2
    }
}
```

4. Далее нейроны объединяем в слои. Для этого создаем класс Layer (листинг 7.3).

Листинг 7.3 – Класс Layer

```
namespace NNBackpropagation
{
    public class Layer
    // объединяем нейроны в слои
    {
        public List<Neuron> Neurons { get; }
        public int NeuronCount => Neurons?.Count ?? 0;
                public NeuronType Type;

        public Layer(List<Neuron> neurons, NeuronType type =
NeuronType.Normal)
        //конструктор (список нейронов проверяем на корректность и тип)
        {
            // TODO: проверить все входные нейроны на соответствие типу
            Neurons = neurons;
            Type = type;
        }

        public List<double> GetSignals()
        {
            //собираем все сигналы со слоя
            var result = new List<double>();
            foreach (var neuron in Neurons)
            {
                result.Add(neuron.Output);
            }
            return result;
        }

        public override string ToString()
        {
            return Type.ToString();
        }
    }
}
```

5. Создадим класс Topology (листинг 7.4). Он будет содержать набор свойств, определяющих свойства нейронной сети.

Листинг 7.4 – Класс Topology

```
namespace NNBackpropagation
{
    public class Topology
    //задает набор свойств
        {
        public int InputCount { get; }

        //входной слой
        public int OutputCount { get; }

        //количество выходов
        public double LearningRate { get; }

        public List<int> HiddenLayers { get; }
        //количество скрытых слоев (для каждого слоя будет коллекция,
        //содержащая определенное число нейронов на каждом слое)

        public Topology(int inputCount, int outputCount, double
learningRate, params int[] layers)
        {
            InputCount = inputCount;
            OutputCount = outputCount;
            LearningRate = learningRate;
            HiddenLayers = new List<int>();
            HiddenLayers.AddRange(layers);
        }
    }
}
```

6. Реализуем нейронную сеть как коллекцию слоев. Для этого создадим класс NeuralNetwork (листинг 7.5).

Листинг 7.5 – Класс NeuralNetwork

```
namespace NNBackpropagation
{
    public class NeuralNetwork
    //реализация нейронной сети (как коллекция слоев)
    {
        public Topology Topology { get; }

        public List<Layer> Layers { get; }

        public NeuralNetwork(Topology topology)
        {
            Topology = topology;
            Layers = new List<Layer>();
```

```
            CreateInputLayer();
            CreateHiddenLayers();
            CreateOutputLayer();
        }

        public double Learn(List<Tuple<double, double[]>> dataset, int
epoch)
        {
            var error = 0.0;
            for (int i = 0; i < epoch; i++)
            {
                foreach (var data in dataset)
                {
                    error += Backpropagation(data.Item1, data.Item2);
                }
            }
            var result = error / epoch;
            return result;
        }

        private double Backpropagation(double exprected, params double[]
inputs)
        {
            var actual = FeedForward(inputs).Output;
            //double exprected-результат который мы ожидаем
            //params double[] inputs входные сигналы
            //метод обратного распространения ошибки, expected-ожидаемый
сигнал
            var difference = actual - exprected;
            foreach (var neuron in Layers.Last().Neurons)
            {
                neuron.Learn(difference, Topology.LearningRate);
            }
            //для других слоев, движение справа налево,
            //-2 так как один слой уже обучили
            for (int j = Layers.Count - 2; j >= 0; j--)
            {
                var layer = Layers[j];
                var previousLayer = Layers[j + 1];
                //обучаем послойно
                for (int i = 0; i < layer.NeuronCount; i++)
                // перебирает нейроны в одном слое
                {
                    var neuron = layer.Neurons[i];

                    for (int k = 0; k < previousLayer.NeuronCount; k++)
                    {
                        var previousNeuron = previousLayer.Neurons[k];
                        var error = previousNeuron.Weights[i] *
previousNeuron.Delta;
                        //обучаем текущий нейрон
```

```
                    neuron.Learn(error, Topology.LearningRate);
                }
            }
        }
        var result = difference * difference;
        return result;
    }

    public Neuron FeedForward(params double[] inputSignals)
    {
        SendSignalsToInputNeurons(inputSignals.ToArray());
        FeedForwardAllLayersAfterInput();
        if (Topology.OutputCount == 1)
        {
            return Layers.Last().Neurons[0];
        }
        else
            return Layers.Last().Neurons.OrderByDescending(n =>
n.Output).First();
    }

    private void FeedForwardAllLayersAfterInput()
    //прогонка для всех нейронов
    {
        for (int i = 1; i < Layers.Count; i++)
        {
            var layer = Layers[i];
            var previousLayerSingals = Layers[i - 1].GetSignals();
            // перебираем все нейроны этого слоя
            foreach (var neuron in layer.Neurons)
            {
                neuron.FeedForward(previousLayerSingals);
            }
        }
    }

    private void SendSignalsToInputNeurons(params double[]
inputSignals)
    {
        //передача сигнала с одного слоя на другой
        for (int i = 0; i < inputSignals.Length; i++)
        {
            var signal = new List<double>() { inputSignals[i] };
            var neuron = Layers[0].Neurons[i];

            neuron.FeedForward(signal);
        }
    }

    private void CreateOutputLayer()
    {
```

```
            //выходной слой
            var outputNeurons = new List<Neuron>();
            var lastLayer = Layers.Last();
            for (int i = 0; i < Topology.OutputCount; i++)
            {
                var neuron = new Neuron(lastLayer.NeuronCount,
NeuronType.Output);
                //lastLayer.NeuronCount- количество нейронов
                //на предыдущем слое
                outputNeurons.Add(neuron);
            }
            var outputLayer = new Layer(outputNeurons,
NeuronType.Output);
            Layers.Add(outputLayer);
        }

        private void CreateHiddenLayers()
        //генерация скрытых слоев
        {
            for (int j = 0; j < Topology.HiddenLayers.Count; j++)
            {
                var hiddenNeurons = new List<Neuron>();
                var lastLayer = Layers.Last();
                for (int i = 0; i < Topology.HiddenLayers[j]; i++)
                {
                    var neuron = new Neuron(lastLayer.NeuronCount);
                    hiddenNeurons.Add(neuron);
                }
                var hiddenLayer = new Layer(hiddenNeurons);
                Layers.Add(hiddenLayer);
            }
        }

        private void CreateInputLayer()
        {
            //создаем входной слой
            var inputNeurons = new List<Neuron>();
            // количество входных нейронов хранится в топологии
            for (int i = 0; i < Topology.InputCount; i++)
            {
                var neuron = new Neuron(1, NeuronType.Input);
                // у входных нейронов вход всегда 1
                inputNeurons.Add(neuron);
            }
            //создаем слой
            var inputLayer = new Layer(inputNeurons, NeuronType.Input);
            Layers.Add(inputLayer);
        }
    }
}
```

Диаграмма классов представлена на рисунке 7.1.

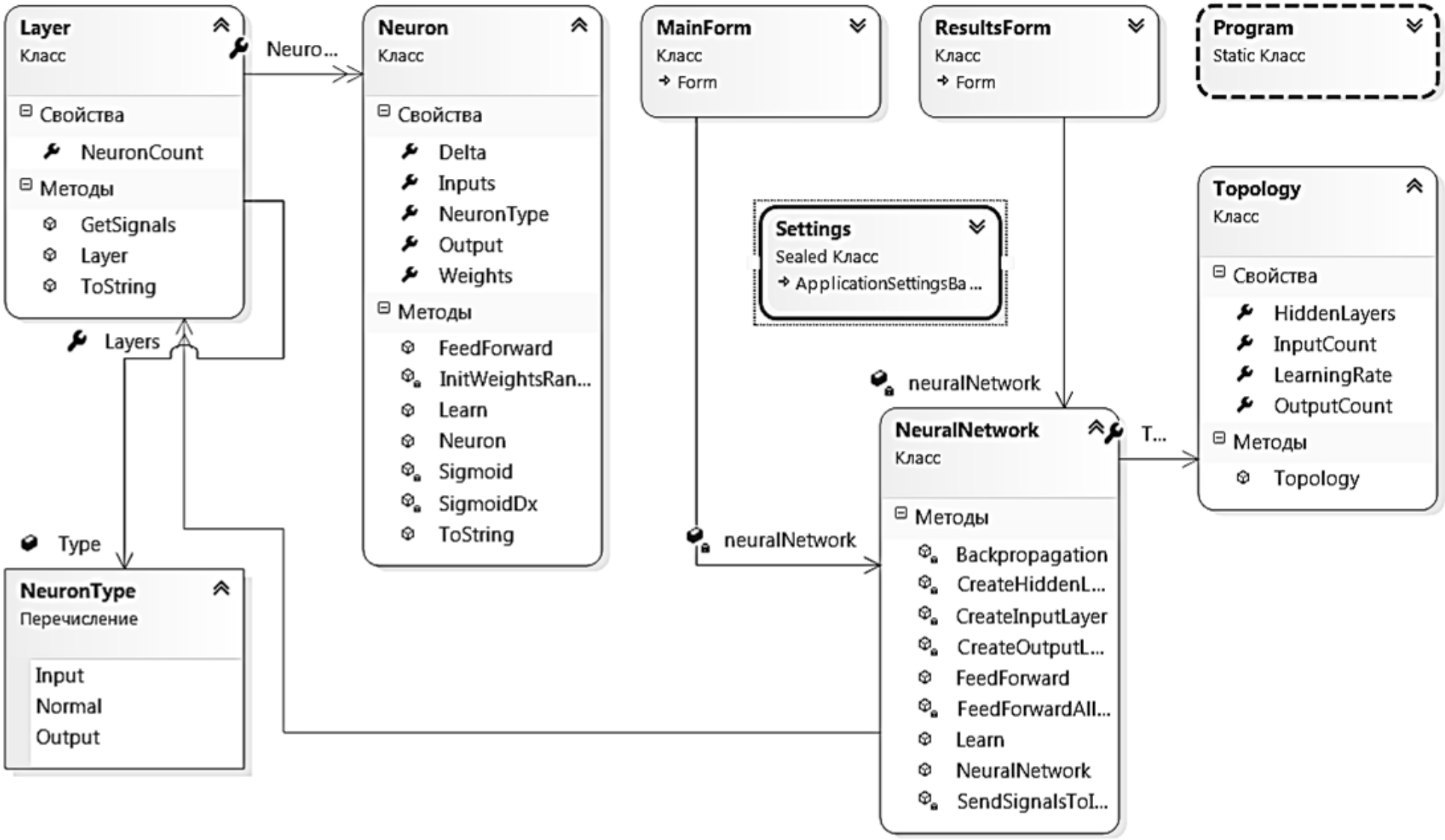


Рисунок 7.1 – Диаграмма классов

7. Создаем две формы:

– Backpropagation – окно с топологией нейронной сети и её обучением;

– ResultsForm – окно для проверки обучения нейронной сети.

Вначале создаем папку Forms. Для этого правой мышью становимся на проект, нажимаем Добавить → Создать папку. И вписываем имя Forms.

Удаляем Form1 (правой кнопкой мыши нажмем на Form1 и выбираем удалить).

Опять правой кнопкой мыши нажимаем на папку «Form», выбираем Добавить → Новый элемент → Форма Windows.

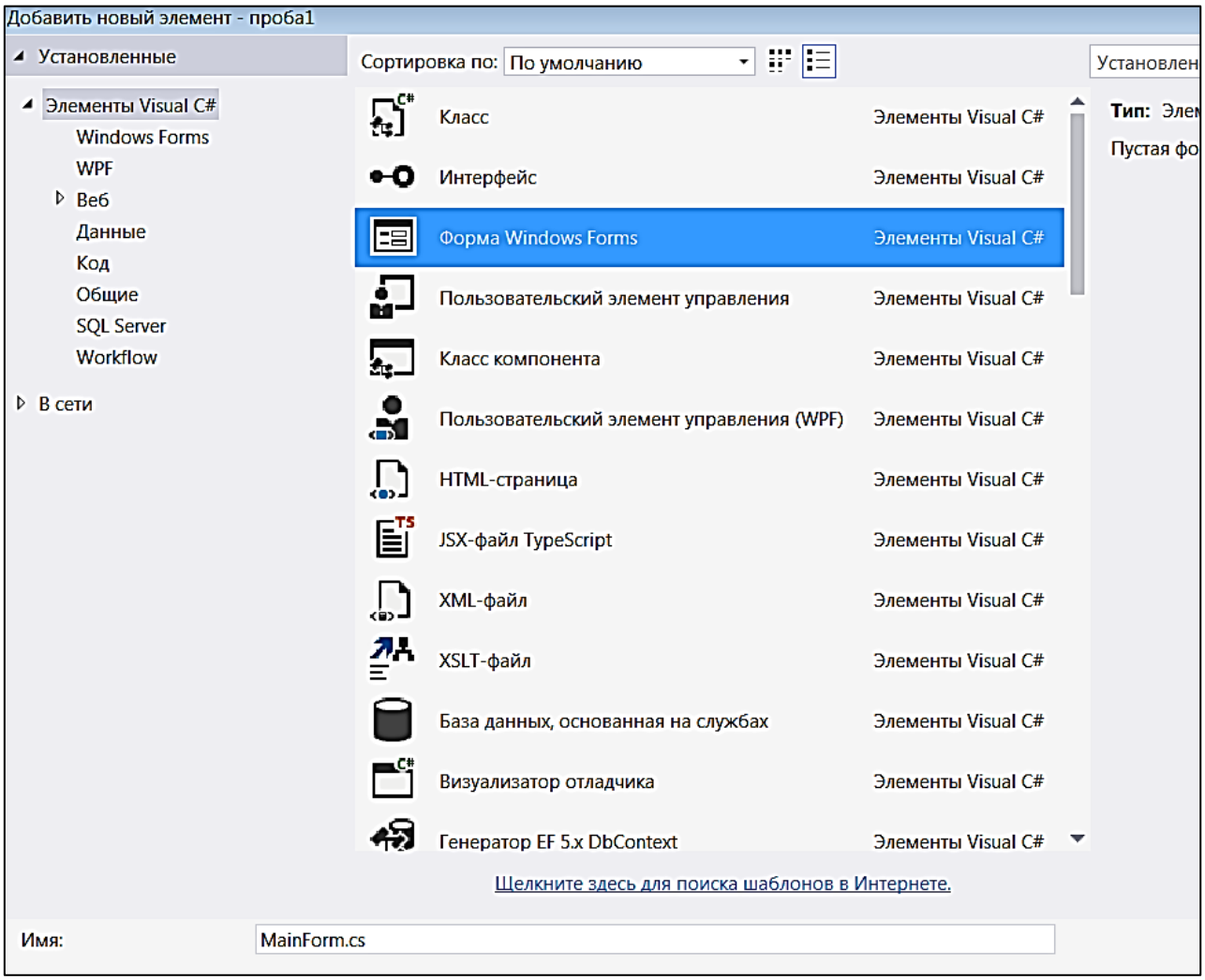


Рисунок 7.2 – Создание формы

Далее необходимо перейти в класс Program.cs и заменить Form1 на MainForm.

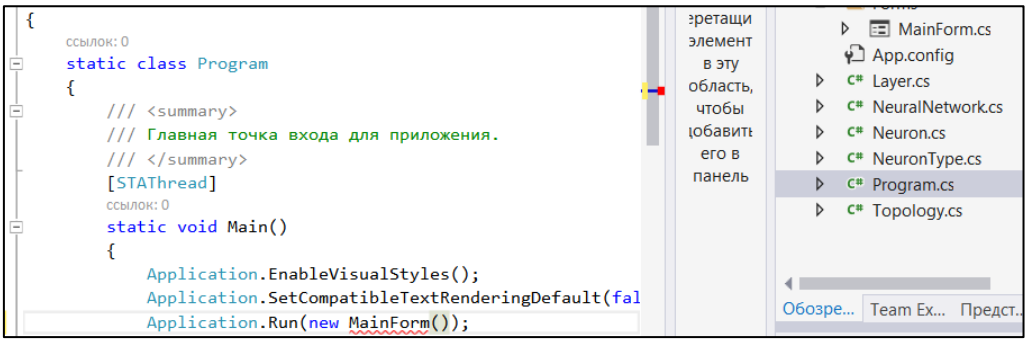


Рисунок 7.3 – Исправления в классе Program.cs

Затем наводим мышку на MainForm и выбраем Показать возможные решения. Добавляем ссылку на пространство имен.

8. Аналогично, становимся на папку Forms. Выбираем Добавить → Форма Windows и вводим имя ResultsForm.cs (рис. 7.4).

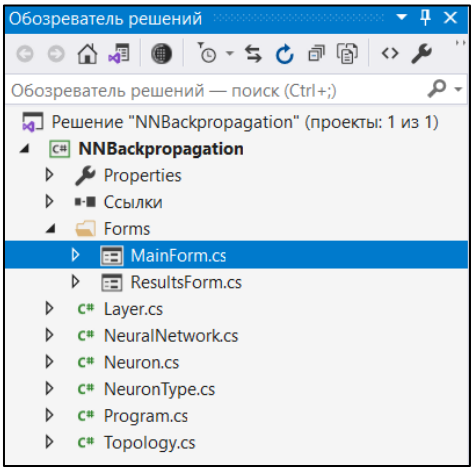


Рисунок 7.4 – Общий вид обозревателя решений

На главной форме выбираем в панели элементов MenuStript. Набираем в нем File и Open. Название свойства изменится само (рис. 7.5).

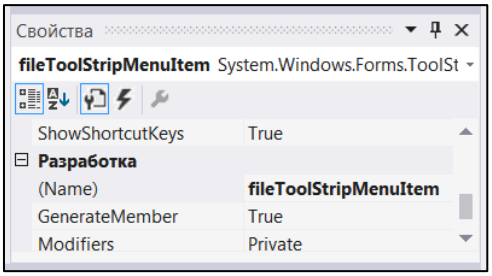


Рисунок 7.5 – Свойства file ToolStripMenuItem

На рисунке 7.6 приведены названия необходимых элементов, расположенных на MainForm.

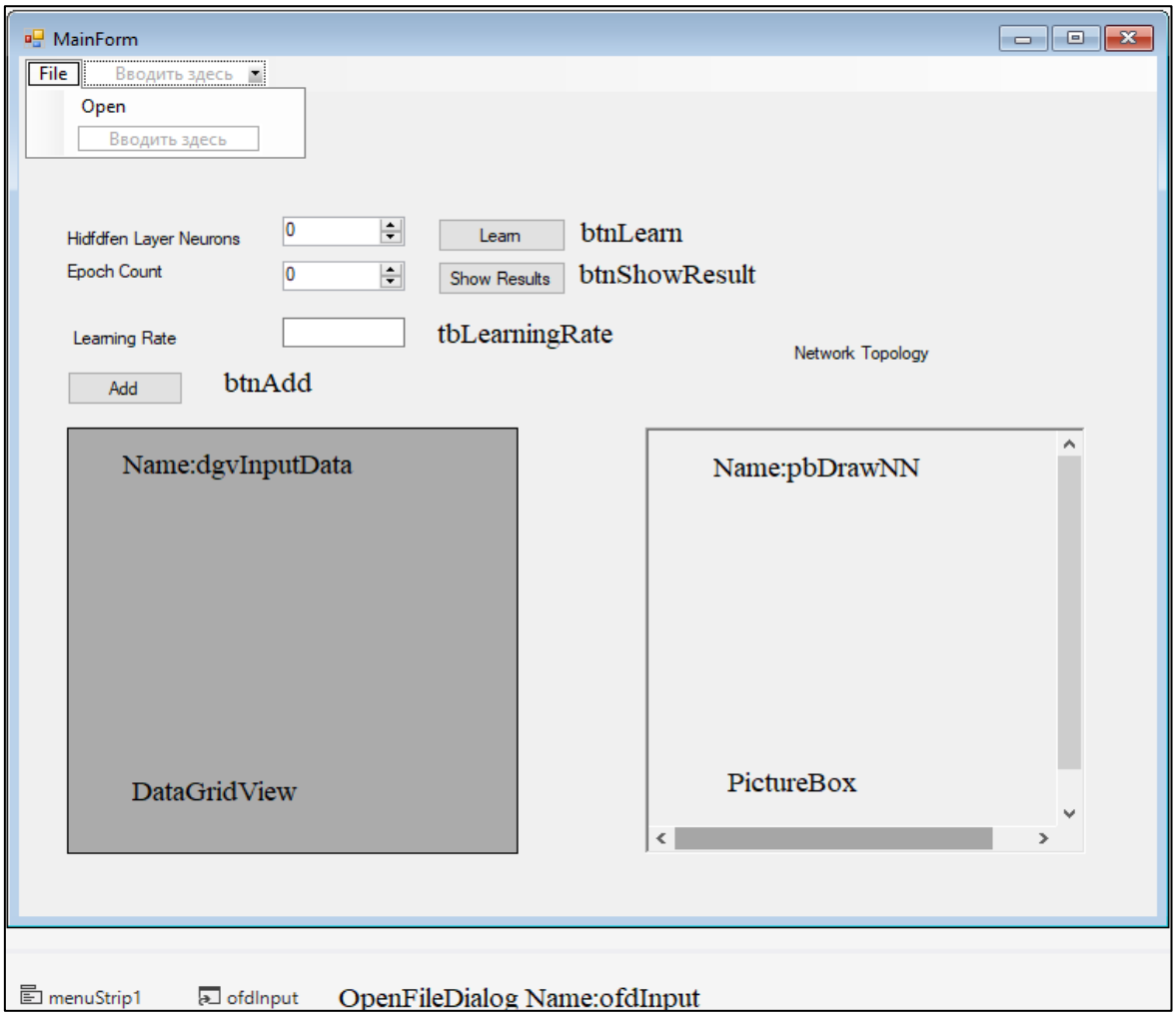


Рисунок 7.6 – Общий вид MainForm

На главной форме так же расположены два NumericUpDoun с именами nudHiddenLayerCount и nudEpochCount) (рис. 7.7).

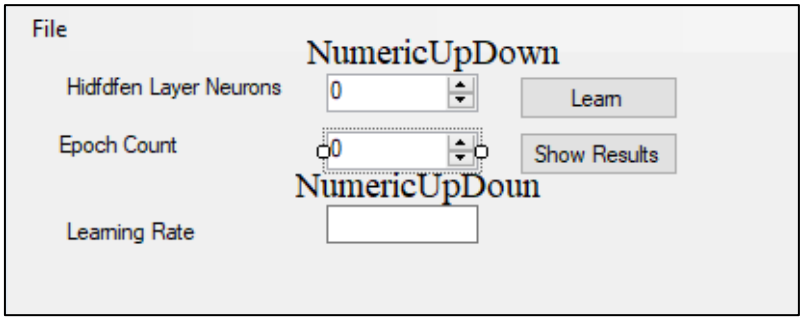


Рисунок 7.7 – Элементы NumericUpDoun

Прежде чем выбрать PictureBox поместим на форму Panel. И указываем для нее свойства Border Style = Fixed3d; AutoScroll = True. Для того, чтобы стал виден Scroll (прокрутка) немного увеличим PictureBox.

Для PictureBox зададим имя Name = pbDrawNN и создадим обработчик события (рис. 7.8).

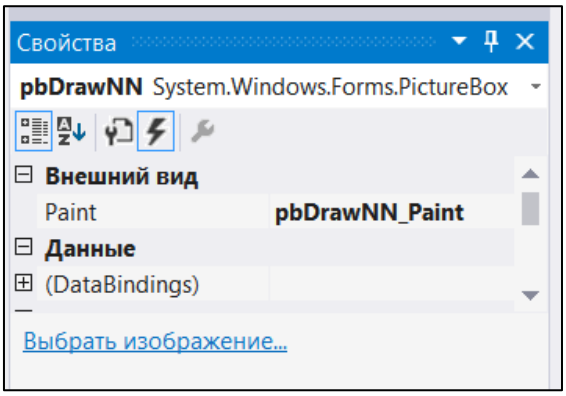


Рисунок 7.8 – Обработчик события PictureBox

Код MainForm представлен в листинге 7.6.

Листинг 7.6 – MainForm

```
namespace NNBackpropagation.Forms
{
    public partial class MainForm : Form
    {
        private NeuralNetwork neuralNetwork;
        private List<Tuple<double, double[]>> dataset;

        private int horizontalSpacing = 70;
        private int neuronSize = 40;
        private int verticalSpacing = 40;
        private Brush neuronFillBrush = Brushes.Blue;
        private Pen neuronBorderPen = new Pen(Color.Black, 2);
        private Pen neuronArcPen = new Pen(Color.Red, 3);

    public MainForm()
        {
            InitializeComponent();
            dgvInputData.ColumnCount = 5;
            dgvInputData.Columns[0].Name = "T";
            dgvInputData.Columns[1].Name = "A";
            dgvInputData.Columns[2].Name = "S";
            dgvInputData.Columns[3].Name = "F";
            dgvInputData.Columns[4].Name = "Expected";
            foreach (DataGridViewColumn column in dgvInputData.Columns)
            {
                column.AutoSizeMode =
DataGridViewAutoSizeColumnMode.ColumnHeader;
            }
        }

        private void btnAdd_Click(object sender, EventArgs e)
        {
            dgvInputData.Rows.Add();
        }
```

```
        private void FillGrid(string fileName)
        {
            DataGridViewRowCollection rows = this.dgvInputData.Rows; //
из методички
            rows.Clear(); // перед тем как добавлять строки, надо убрать
старые
            var lines = File.ReadAllLines(fileName);
            foreach (var line in lines)
            {
                rows.Add(line.Split());
            }
        }

        private void openToolStripMenuItem_Click_1(object sender,
EventArgs e)
        {
//станьте на File-Open
            if (ofdInput.ShowDialog() == DialogResult.OK)
            {
                FillGrid(ofdInput.FileName);
            }
        }

        private List<Tuple<double, double[]>> GetDataSet()
        {
            //список входных данных с ожидаемым результатом
            List<Tuple<double, double[]>> results = new
List<Tuple<double, double[]>>();
            foreach (DataGridViewRow row in dgvInputData.Rows)
            {
                bool anyEmptyCellPresent = false;
                var currentRow = new List<double>();
                foreach (DataGridViewCell cell in row.Cells)
                {
                    anyEmptyCellPresent = cell.Value == null
                        || string.IsNullOrEmpty(cell.Value.ToString());
                    if (!anyEmptyCellPresent)
                    {
                        var currentCellValue =
Convert.ToDouble(cell.Value);
                        currentRow.Add(currentCellValue);
                    }
                    else
                    {
                        break;
                    }
                }
                if (!anyEmptyCellPresent)
                {
                    var expected = currentRow.Last();
                    currentRow.RemoveAt(currentRow.Count - 1);
```

```
                    results.Add(new Tuple<double, double[]>(expected,
currentRow.ToArray()));
                }
                else
                {
                    continue;
                }
            }
            return results;
        }

        private void StartLoading()
        {
            // выключает кнопку для показа результатов и добавления
записи
            btnLearn.Enabled = false;
            btnShowResults.Enabled = false;
            btnAdd.Enabled = false;
        }

        private void StopLoading()
        {
            btnLearn.Enabled = true;
            btnShowResults.Enabled = true;
            btnAdd.Enabled = true;
        }

        private async void btnLearn_Click(object sender, EventArgs e)
        {
     //Руками доберите async (автоматически не вставляется!!!)
     //смотри задание
            var hiddenLayerCount = (int)nudHiddenLayerCount.Value;
            var epochCount = (int)nudEpochCount.Value;
            double learningRate = 0.1;
            try
            {
                learningRate = Convert.ToDouble(tbLearningRate.Text);
                if (learningRate > 0)
                {
                    var topology = new Topology(4, 1, learningRate,
hiddenLayerCount);
                    neuralNetwork = new NeuralNetwork(topology);
                    dataset = GetDataSet();
                    StartLoading();
                    await Learn(dataset, epochCount);
                    MessageBox.Show("Learning ended");
                    StopLoading();
                }
                else
                {
```

```
                    MessageBox.Show("Learning rate should be greater then
0");
                }
            }
            catch (FormatException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void btnShowResults_Click(object sender, EventArgs e)
        {
            var resultsForm = new ResultsForm(neuralNetwork, dataset);
            resultsForm.Show();
        }

        private void nudHiddenLayerCount_ValueChanged(object sender,
EventArgs e)
        {
            pbDrawNN.Invalidate();
        }

        private Task Learn(List<Tuple<double, double[]>> dataset, int
epochCount)
        {
            return Task.Run(() =>
            {
                neuralNetwork.Learn(dataset, epochCount);
            });
        }

        private void pbDrawNN_Paint(object sender, PaintEventArgs e)
        {
            DrawNetwork(e.Graphics, new int[] { 4,
(int)nudHiddenLayerCount.Value, 1 });
        }

        // вычисляет высоту слоя нейронной сети
        private int ComputeLayerHeight(int layerSize)
        {
            return layerSize * neuronSize + (layerSize - 1) *
verticalSpacing;
        }

        private void DrawNetwork(Graphics g, int[] layerSizes)
        {
            var maxLayerSize = layerSizes.Max();
            var maxLayerHeight = ComputeLayerHeight(maxLayerSize);
            if (pbDrawNN.Height < maxLayerHeight)
            {
                pbDrawNN.Height = maxLayerHeight + verticalSpacing;
```

```
            }
            var currentLayerX = horizontalSpacing;
            var layerNeronRectangles = new List<Rectangle[]>();
            foreach (var layerSize in layerSizes)
            {
                var currentLayer = GetLayerNeuronRectangles(layerSize,
currentLayerX);
                layerNeronRectangles.Add(currentLayer);
                currentLayerX += horizontalSpacing + neuronSize;
            }
            DrawLines(layerNeronRectangles, g);
            DrawNeurons(layerNeronRectangles, g);
        }

        private Rectangle[] GetLayerNeuronRectangles(int layerSize, int
leftLayerBorderX)
        {
            var layerHeight = ComputeLayerHeight(layerSize);
            var layerTop = (pbDrawNN.Height - layerHeight) / 2;
            var currentRectangles = new List<Rectangle>();
            var currrentTop = layerTop;
            var currentLeft = leftLayerBorderX;
            for (var i = 0; i < layerSize; i++)
            {
                currentRectangles.Add(new Rectangle(currentLeft,
currrentTop, neuronSize, neuronSize));
                currrentTop += neuronSize + verticalSpacing;
            }
            return currentRectangles.ToArray();
        }

        private void DrawNeurons(List<Rectangle[]> layerNeronRectangles,
Graphics g)
        {
            foreach (var rectangles in layerNeronRectangles)
            {
                foreach (var rectangle in rectangles)
                {
                    g.FillEllipse(neuronFillBrush, rectangle);
                    g.DrawEllipse(neuronBorderPen, rectangle);
                }
            }
        }

        private void DrawLines(List<Rectangle[]> layerNeronRectangles,
Graphics g)
        {
            List<PointF> previousPoints = null;

            foreach (var rectangles in layerNeronRectangles)
            {
```

```
                var currentPoints = new List<PointF>();
                foreach (var rectangle in rectangles)
                {
                    var currentPoint = new PointF(rectangle.X +
(float)rectangle.Width / 2, rectangle.Y + (float)rectangle.Height / 2);
                    currentPoints.Add(currentPoint);

                    if (previousPoints != null)
                    {
                        foreach (var previousPoint in previousPoints)
                        {
                            g.DrawLine(neuronArcPen, previousPoint.X,
previousPoint.Y, currentPoint.X, currentPoint.Y);
                        }
                    }
                }
                previousPoints = currentPoints;
            }
        }
    }
}
```

На рисунке 7.9 приведены названия необходимых элементов, расположенных на ResultsForm. Код ResultsForm представлен в листинге 7.7.

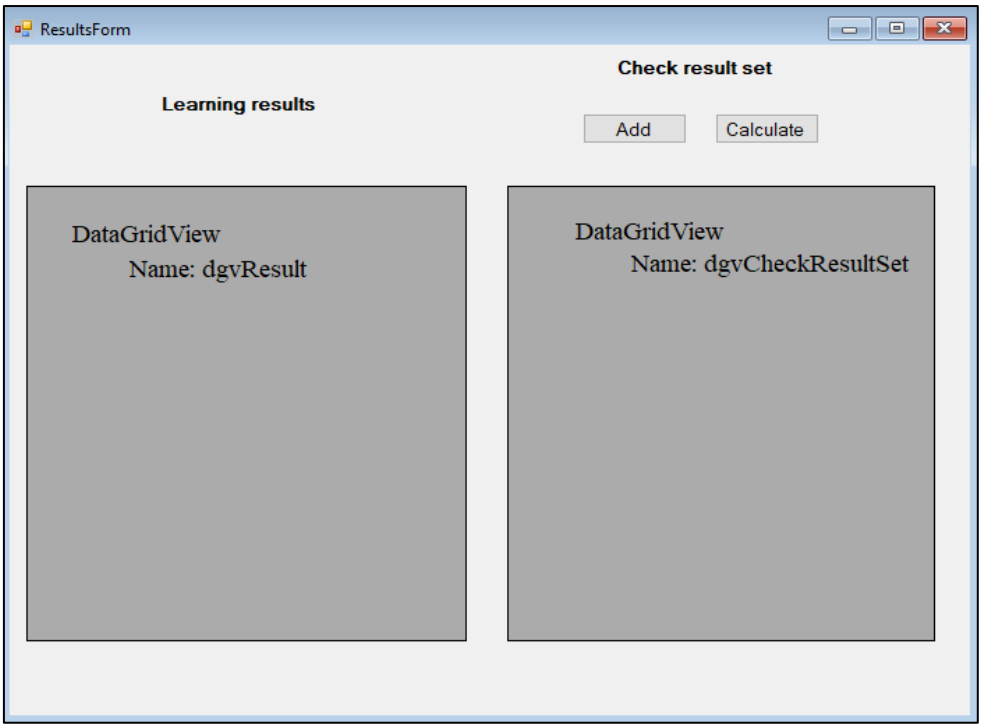


Рисунок 7.9 – Общий вид ResultsForm

Листинг 7.7 – ResultsForm

```
namespace NNBackpropagation.Forms
{
```

```
    public partial class ResultsForm : Form
    {
        private NeuralNetwork neuralNetwork;
        private List<Tuple<double, double[]>> dataset;
        public ResultsForm(NeuralNetwork neuralNetwork,
List<Tuple<double, double[]>> dataset)
        {
            InitializeComponent();
            this.neuralNetwork = neuralNetwork;
            this.dataset = dataset;
            InitializeResultsGrid();
            InitializeCheckResultSet();
        }
        private void InitializeResultsGrid()
        {
            dgvResults.ColumnCount = 6;
            dgvResults.Columns[0].Name = "T";
            dgvResults.Columns[1].Name = "A";
            dgvResults.Columns[2].Name = "S";
            dgvResults.Columns[3].Name = "F";
            dgvResults.Columns[4].Name = "Expected";
            dgvResults.Columns[5].Name = "Actual";
            foreach (DataGridViewColumn column in dgvResults.Columns)
            {
                column.AutoSizeMode =
DataGridViewAutoSizeColumnMode.ColumnHeader;
            }
            foreach (var set in dataset)
            {
                var result = neuralNetwork.FeedForward(set.Item2).Output;
                var expected = Math.Round(set.Item1, 3);
                var actual = Math.Round(result, 3);
                var row = new List<double>();
                row.AddRange(set.Item2);
                row.Add(expected);
                row.Add(actual);
                dgvResults.Rows.Add(row.Select(x =>
x.ToString()).ToArray());
            }
        }
        private void InitializeCheckResultSet()
        {
            dgvCheckResultSet.ColumnCount = 5;
            dgvCheckResultSet.Columns[0].Name = "T";
            dgvCheckResultSet.Columns[1].Name = "A";
            dgvCheckResultSet.Columns[2].Name = "S";
            dgvCheckResultSet.Columns[3].Name = "F";
            dgvCheckResultSet.Columns[4].Name = "Result";
            foreach (DataGridViewColumn column in
dgvCheckResultSet.Columns)
            {
```

```
                column.AutoSizeMode =
DataGridViewAutoSizeColumnMode.ColumnHeader;
            }
        }
        private void btnAdd_Click(object sender, EventArgs e)
        {
            dgvCheckResultSet.Rows.Add();
        }
        private void btnCalculate_Click(object sender, EventArgs e)
        {
            try
            {
                foreach (DataGridViewRow row in dgvCheckResultSet.Rows)
                {
                    var inputSignalsLength = row.Cells.Count - 1;
                    var inputSignals = new double[inputSignalsLength];
                    bool anyEmptyCellPresent = false;
                    for (var i = 0; i < inputSignalsLength; i++)
                    {
                        anyEmptyCellPresent = row.Cells[i].Value == null
                            ||
string.IsNullOrEmpty(row.Cells[i].Value.ToString());
                        if (!anyEmptyCellPresent)
                        {
                            var inputSignalStr = row.Cells[i].Value;
                            inputSignals[i] =
Convert.ToDouble(inputSignalStr);
                        }
                        else
                        {
                            break;
                        }
                    }
                    if (!anyEmptyCellPresent)
                    {
                        var result =
neuralNetwork.FeedForward(inputSignals).Output;
                        row.Cells[row.Cells.Count - 1].Value =
Math.Round(result, 3);
                    }
                }
            }
            catch (FormatException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }
    }
}
```

9. Создаем файл inputData1 (рис. 7.10). Поместим его в bin → Debug.

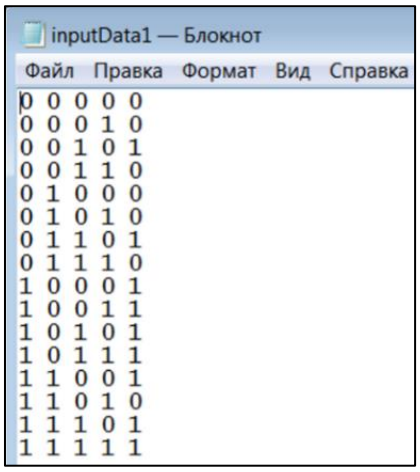


Рисунок 7.10 – Файл inputData1

7.3 Запуск приложения

При запуске открывается стартовое окно программы (рис. 7.11). В DataSet записываются входные данные (симптомы и параметры пациента) с ожидаемым результатом. Hidden Layer Neurons является изменяемым параметром и отвечает за количество нейронов скрытого слоя. Epoch Count – параметр, который определяет, сколько раз алгоритм обучения будет работать во всем наборе обучающих данных. Learning rate – параметр, отвечающий за скорость и точность обучения: чем он ниже, тем ниже скорость и выше точность.

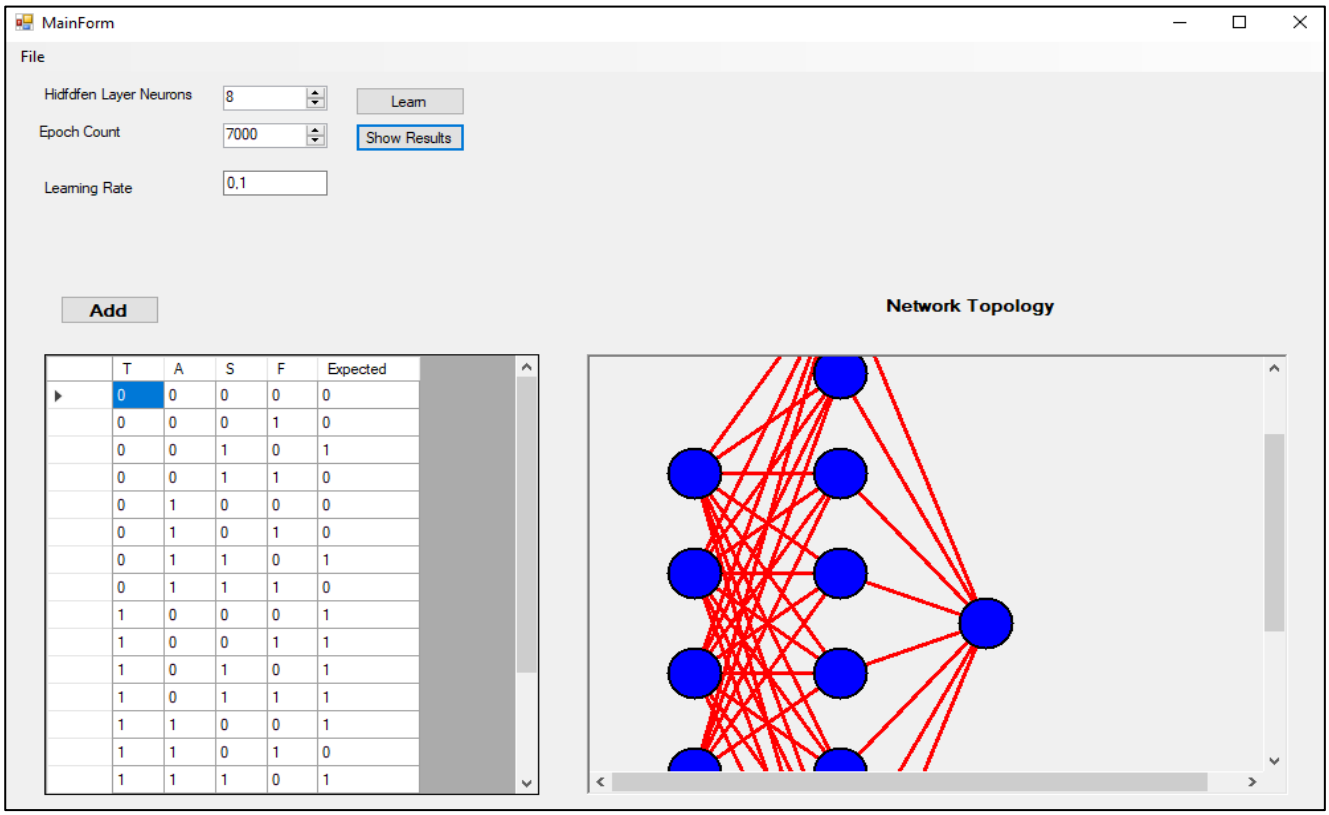


Рисунок 7.11 – Стартовое окно программы NNBackpropagation.exe

После того как нейронная сеть пройдет процесс обучения, выведется об этом сообщение (рис. 7.12).

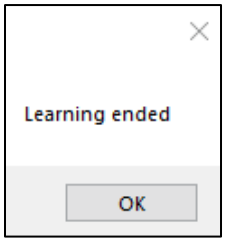
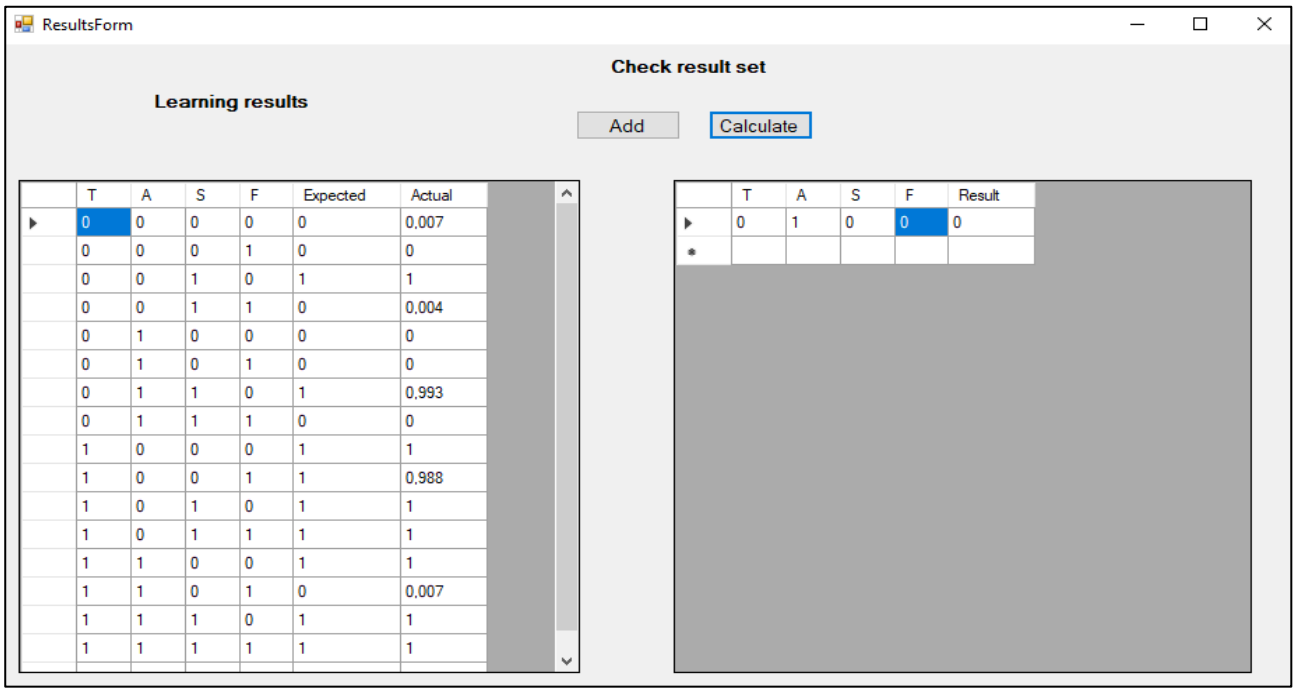


Рисунок 7.12 – Сообщение о завершении процесса обучения

После нажатия на кнопку Show results откроется новое окно, в котором реализован интерфейс проверки на корректность обучения нейронной сети.



T	A	S	F	Expected	Actual
0	0	0	0	0	0,007
0	0	0	1	0	0
0	0	1	0	1	1
0	0	1	1	0	0,004
0	1	0	0	0	0
0	1	0	1	0	0
0	1	1	0	1	0,993
0	1	1	1	0	0
1	0	0	0	1	1
1	0	0	1	1	0,988
1	0	1	0	1	1
1	0	1	1	1	1
1	1	0	0	1	1
1	1	0	1	0	0,007
1	1	1	0	1	1
1	1	1	1	1	1

T	A	S	F	Result
0	1	0	0	0

Рисунок 7.13 – Окно для проверки результата обучения

В столбце Actual отображается реальный результат обучения нейронной сети. Значения зависят от параметров обучения.

Задания для самостоятельной работы

1. Запустить приложение.
2. Исследовать, как зависит скорость обучения от коэффициента обучения.
3. Исследовать, как зависит скорость обучения от числа нейронов на скрытом слое.
4. Исследовать, как зависит скорость обучения от числа эпох.

5. Добиться 100% согласия между ожидаемыми и действительными значениями.

ЛАБОРАТОРНАЯ РАБОТА 8. РАЗРАБОТКА ПРИЛОЖЕНИЯ «РАСПОЗНАВАНИЕ ИЗОБРАЖЕНИЙ»

Цель работы: применение метода обратного распространения ошибки для распознавания простейших образов.

Краткие теоретические сведения

В настоящее время нейронные сети широко применяются в распознавании образов. Для нейронных сетей не имеет значения, с какими данными работать. Это могут быть и текстовые данные, и изображения, и обработка звука.

При работе с изображениями, необходимо информацию об изображении представить в удобном формате. Например, в виде нулей и единиц. Это соответствует черному и белому пикселу. Поэтому при работе с цветными изображениями их удобно перевести в черно-белый формат. Чтобы выполнить преобразование цветного изображения в черно-белое, можно найти среднее арифметическое значение R, G и B для каждого пиксела, а затем присвоить ему это значение. Для этих целей существует формула, например:

$$A = 0,299 \cdot R + 0,587 \cdot G + 0,114 \cdot B.$$

В качестве порогового значения часто берется 128. Если полученное значение меньше или равно 128, то присваиваем пикселу значение нуль, если больше 128, то единицу.

8.1 Разработка приложения

Диаграмма классов приложения приведена на рисунке 8.1.

На диаграмме классов:

Neuron – класс для создания весов, нахождения ошибки и вычисления новых весов.

Layer – класс для проверки всех весов на соответствие типу.

Topology – класс, описывающий топологию нейронной сети.

NeuralNetwork – класс, отвечающий за обучение нейронной сети методом обратного распространения ошибки.

PictureConverter – класс для преобразования изображения из цветного формата в черно-белый.

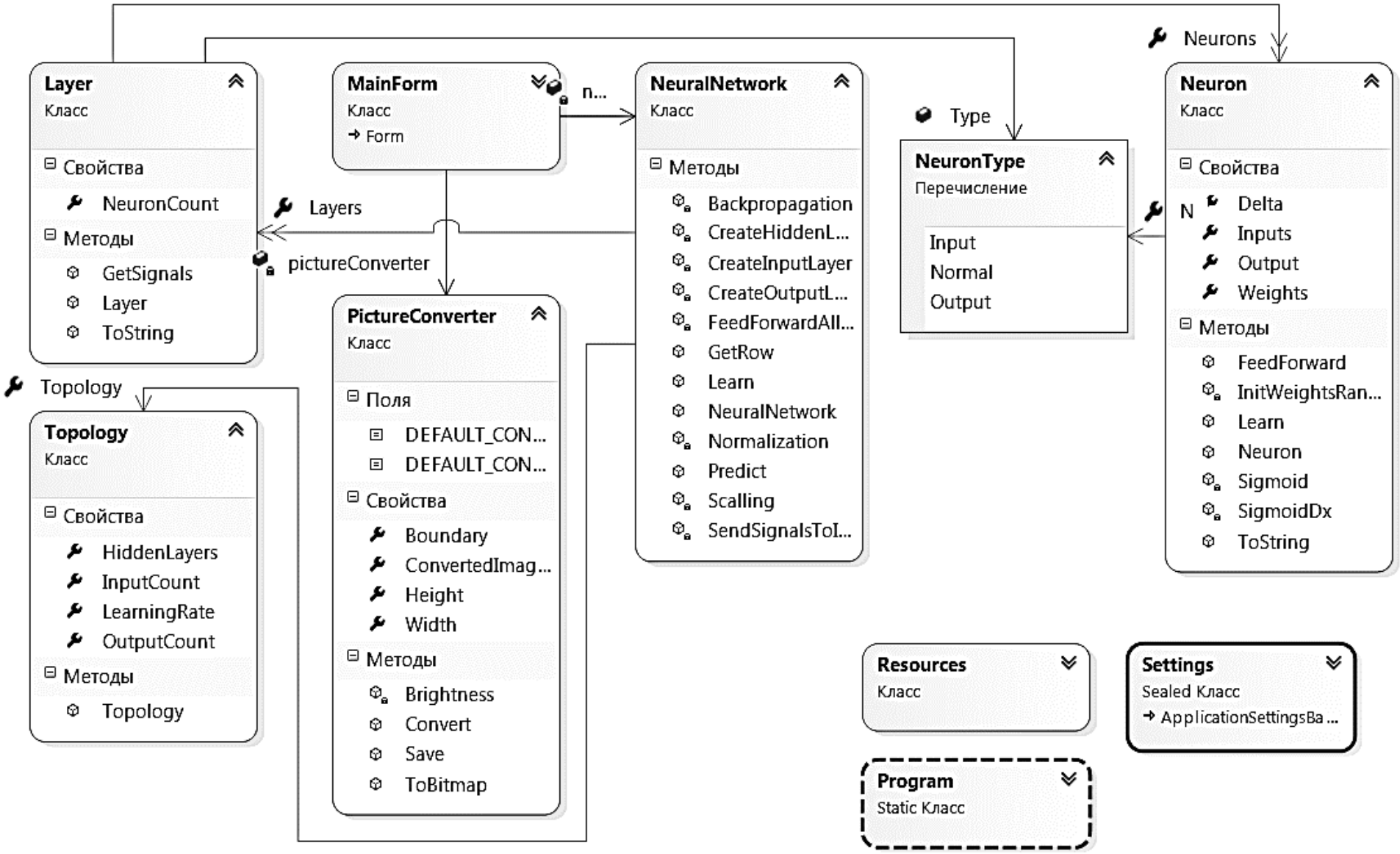


Рисунок 8.1 – Диаграмма классов

1. Листинги классов Neuron, Topology, NeuronType возьмем из лабораторной работы 5.

2. Создадим класс Layer (листинг 8.1). Для этого перейдем в Проект → Добавить класс.

Листинг 8.1 – Класс Layer

```
namespace RecogImage
{
    public class Layer
    {
        public List<Neuron> Neurons { get; }
        public int NeuronCount => Neurons?.Count ?? 0;
        public NeuronType Type;

        public Layer(List<Neuron> neurons, NeuronType type =
NeuronType.Normal)
        {
            // TODO: проверить все входные нейроны на соответствие типу
            Neurons = neurons;
            Type = type;
        }

        public List<double> GetSignals()
        {
            var result = new List<double>();
            foreach (var neuron in Neurons)
            {
                result.Add(neuron.Output);
            }
            return result;
        }

        public override string ToString()
        {
            return Type.ToString();
        }
    }
}
```

3. Создадим класс NeuralNetwork (листинг 8.2).

Листинг 8.2 – Класс NeuralNetwork

```
namespace RecogImage
{
    public class NeuralNetwork
    {
        public Topology Topology { get; }

        public List<Layer> Layers { get; }
```

```
        public NeuralNetwork(Topology topology)
        {
            Topology = topology;

            Layers = new List<Layer>();

            CreateInputLayer();
            CreateHiddenLayers();
            CreateOutputLayer();
        }

        public Neuron Predict(params double[] inputSignals)
        {
            SendSignalsToInputNeurons(inputSignals);
            FeedForwardAllLayersAfterInput();
            if (Topology.OutputCount == 1)
            {
                return Layers.Last().Neurons[0];
            }
            else
            {
                return Layers.Last().Neurons.OrderByDescending(n =>
n.Output).First();
            }
        }

        public double Learn(double[] expected, double[,] inputs, int
epoch)
        {
            var error = 0.0;
            for (int i = 0; i < epoch; i++)
            {
                for (int j = 0; j < expected.Length; j++)
                {
                    var output = expected[j];
                    var input = GetRow(inputs, j);

                    error += Backpropagation(output, input);
                }
            }
            var result = error / epoch;
            return result;
        }

        public static double[] GetRow(double[,] matrix, int row)
        {
            var columns = matrix.GetLength(1);
            var array = new double[columns];
            for (int i = 0; i < columns; ++i)
                array[i] = matrix[row, i];
            return array;
```

```
        }

        private double[,] Scalling(double[,] inputs)
        {
            var result = new double[inputs.GetLength(0),
inputs.GetLength(1)];
            for (int column = 0; column < inputs.GetLength(1); column++)
            {
                var min = inputs[0, column];
                var max = inputs[0, column];
                for (int row = 1; row < inputs.GetLength(0); row++)
                {
                    var item = inputs[row, column];
                    if (item < min)
                    {
                        min = item;
                    }
                    if (item > max)
                    {
                        max = item;
                    }
                }
                var divider = max - min;
                for (int row = 1; row < inputs.GetLength(0); row++)
                {
                    result[row, column] = (inputs[row, column] - min) /
divider;
                }
            }
            return result;
        }

        private double[,] Normalization(double[,] inputs)
        {
            var result = new double[inputs.GetLength(0),
inputs.GetLength(1)];
            for (int column = 0; column < inputs.GetLength(1); column++)
            {
                // среднее значение сигнала нейрона
                var sum = 0.0;
                for (int row = 0; row < inputs.GetLength(0); row++)
                {
                    sum += inputs[row, column];
                }
                var average = sum / inputs.GetLength(0);
                // стандартное квадратичное отклонение нейрона
                var error = 0.0;
                for (int row = 0; row < inputs.GetLength(0); row++)
                {
                    error += Math.Pow((inputs[row, column] - average),
2);
```

```
                }
                var standardError = Math.Sqrt(error /
inputs.GetLength(0));

                for (int row = 0; row < inputs.GetLength(0); row++)
                {
                    result[row, column] = (inputs[row, column] - average)
/ standardError;
                }
            }
            return result;
        }

        private double Backpropagation(double exprected, params double[]
inputs)
        {
            // метод обратного распространения ошибки, expected-ожидаемый
сигнал
            var actual = Predict(inputs).Output;
            //  var actual = FeedForward(inputs).Output;
            var difference = actual - exprected;

            foreach (var neuron in Layers.Last().Neurons)
            {
                neuron.Learn(difference, Topology.LearningRate);
            }
            // для других слоев, движение справа налево, -2 один слой уже
обучили
            for (int j = Layers.Count - 2; j >= 0; j--)
            {
                var layer = Layers[j];
                var previousLayer = Layers[j + 1];
                //обучает послойно
                for (int i = 0; i < layer.NeuronCount; i++)
                // перебирает нейроны в одном слое
                {
                    var neuron = layer.Neurons[i];

                    for (int k = 0; k < previousLayer.NeuronCount; k++)
                    //
                    {
                        var previousNeuron = previousLayer.Neurons[k];
                        var error = previousNeuron.Weights[i] *
previousNeuron.Delta;
                        //обучает текущий нейрон
                        neuron.Learn(error, Topology.LearningRate);
                    }
                }
            }
            var result = difference * difference;
            return result;
```

```
        }

        private void FeedForwardAllLayersAfterInput()
        {
            for (int i = 1; i < Layers.Count; i++)
            {
                var layer = Layers[i];
                var previousLayerSingals = Layers[i - 1].GetSignals();

                foreach (var neuron in layer.Neurons)
                {
                    neuron.FeedForward(previousLayerSingals);
                }
            }
        }

        private void SendSignalsToInputNeurons(params double[]
inputSignals)
        {
            for (int i = 0; i < inputSignals.Length; i++)
            {
                var signal = new List<double>() { inputSignals[i] };
                var neuron = Layers[0].Neurons[i];
                neuron.FeedForward(signal);
            }
        }

        private void CreateOutputLayer()
        {
            var outputNeurons = new List<Neuron>();
            var lastLayer = Layers.Last();
            for (int i = 0; i < Topology.OutputCount; i++)
            {
                var neuron = new Neuron(lastLayer.NeuronCount,
NeuronType.Output);
                outputNeurons.Add(neuron);
            }
            var outputLayer = new Layer(outputNeurons,
NeuronType.Output);
            Layers.Add(outputLayer);
        }

        private void CreateHiddenLayers()
        {
            for (int j = 0; j < Topology.HiddenLayers.Count; j++)
            {
                var hiddenNeurons = new List<Neuron>();
                var lastLayer = Layers.Last();
                for (int i = 0; i < Topology.HiddenLayers[j]; i++)
                {
                    var neuron = new Neuron(lastLayer.NeuronCount);
```

```
                hiddenNeurons.Add(neuron);
            }
            var hiddenLayer = new Layer(hiddenNeurons);
            Layers.Add(hiddenLayer);
        }
    }

    private void CreateInputLayer()
    {
        var inputNeurons = new List<Neuron>();
        for (int i = 0; i < Topology.InputCount; i++)
        {
            var neuron = new Neuron(1, NeuronType.Input);
            inputNeurons.Add(neuron);
        }
        var inputLayer = new Layer(inputNeurons, NeuronType.Input);
        Layers.Add(inputLayer);
    }
  }
}
```

4. Создадим класс для преобразования изображения из цветного формата в черно-белый (листинг 8.3).

Листинг 8.3 – Класс PictureConverter

```
namespace RecogImage
{
    public class PictureConverter
    {
        public const int DEFAULT_CONVERTED_IMAGE_SIDE = 15;
        public const int DEFAULT_CONVERTED_IMAGE_SIZE =
DEFAULT_CONVERTED_IMAGE_SIDE * DEFAULT_CONVERTED_IMAGE_SIDE;
        public int Boundary { get; set; } = 128;
        public int Height { get; set; }
        public int Width { get; set; }
        public int ConvertedImageSide { get; set; } =
DEFAULT_CONVERTED_IMAGE_SIDE;
        public List<int>Convert(string path)
        {
            //var result = new List<double>();
            var result = new List<int>();
            //считываем изображение
            var image = new Bitmap(path);
            var resizeImage = new Bitmap(image, new
Size(ConvertedImageSide, ConvertedImageSide));
            Height = resizeImage.Height;
            Width = resizeImage.Width;
            for(int y = 0; y < resizeImage.Height; y++)
            {
                for(int x = 0; x < resizeImage.Width; x++)
                {
```

```
                    var pixel = resizeImage.GetPixel(x, y);
                    var value = Brightness(pixel);
                    result.Add(value);
                }
            }
            return result;
            //return result.ToArray();
        }

        private int Brightness(Color pixel)
        {
            //превращение картинки в черно-белую из цветной
            var result = 0.299 * pixel.R + 0.587 * pixel.G + 0.114 *
pixel.B;
            return result < Boundary ? 0 : 1;
        }

        public void Save(string path, List<int> pixels)
        {
            var image = new Bitmap(Width, Height);
            for (int y = 0; y < image.Height; y++)
            {
                for (int x = 0; x < image.Width; x++)
                {
                    var color = pixels[y * Width + x] == 1 ? Color.White
: Color.Black;
                    image.SetPixel(x, y, color);
                }
            }
            image.Save(path);
        }

        // преобразует массив пикселей в изображение для формы
        public Bitmap ToBitmap(List<int> pixels)
        {
            var image = new Bitmap(Width, Height);
            for (int y = 0; y < image.Height; y++)
            {
                for (int x = 0; x < image.Width; x++)
                {
                    var color = pixels[y * Width + x] == 1 ? Color.White
: Color.Black;
                    image.SetPixel(x, y, color);
                }
            }
            return image;
        }
    }
}
```

5. На главной форме в приложении расположено окно (MenuStrip) (рис. 8.2). Оно позволяет открыть файлы или папки, в которых сохранены цветные изображения. Так же расположена секция с параметрами обучения сети. Цветные картинки переводятся в черно-белые.

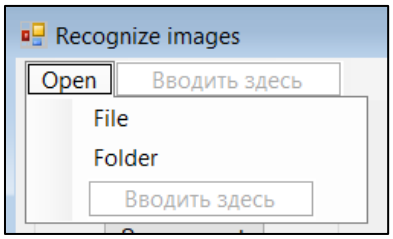


Рисунок 8.2 – Окно MenuStrip

На форме расположены два PictureBox (pbImage и pbBlackWhite), в которых хранится цветное изображение и его черно-белый аналог (рис. 8.3). Код класса MainForm приводится в листинге 8.4.

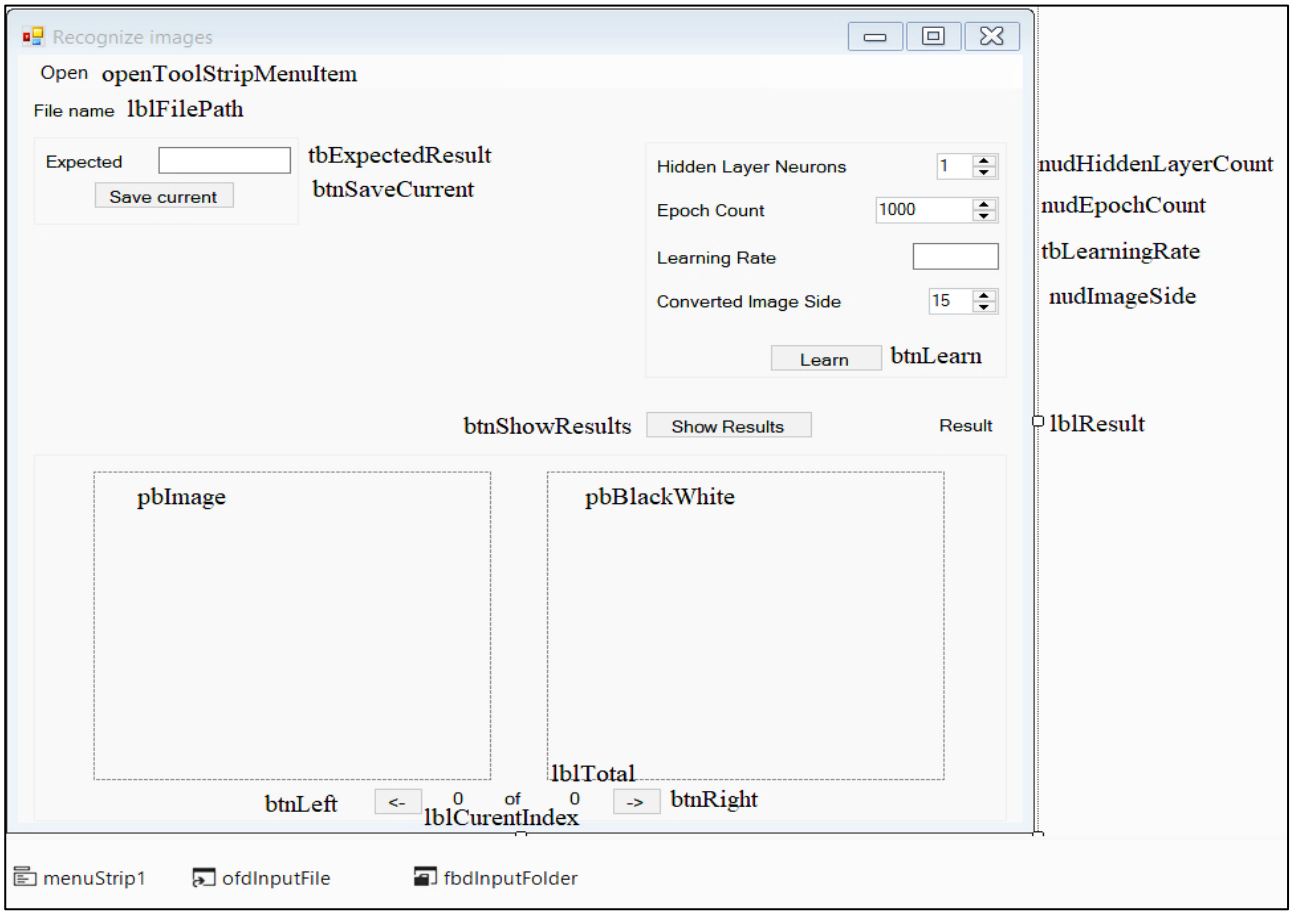


Рисунок 8.3 – Окно для распознавания изображений

Листинг 8.4 – Класс MainForm

```
namespace RecogImage.Forms
{
    public partial class MainForm : Form
```

```
    {
        private List<string> imagePathes = new List<string>();
        private List<double> expectedResults = new List<double>();
        private List<int> blackWhitePixels;
        private PictureConverter pictureConverter = new
PictureConverter();
        private int currentIndex = -1;
        private NeuralNetwork neuralNetwork;

        public MainForm()
        {
            InitializeComponent();
            nudHiddenLayerCount.Value =
PictureConverter.DEFAULT_CONVERTED_IMAGE_SIZE / 5;
            nudImageSide.Value =
PictureConverter.DEFAULT_CONVERTED_IMAGE_SIDE;
        }

        private void btnLeft_Click(object sender, EventArgs e)
        {
            if (currentIndex > 0)
            {
                ShowImage(currentIndex - 1);
            }
        }

        private void btnRight_Click(object sender, EventArgs e)
        {
            if (currentIndex < imagePathes.Count - 1)
            {
                ShowImage(currentIndex + 1);
            }
        }

        private void btnSaveCurrent_Click(object sender, EventArgs e)
        {
            SaveExpectedResult();
        }

        // покажет изображение в pictureBox и изменит значение в textBox
        private void ShowImage(int index)
        {
            currentIndex = index;
            pbImage.ImageLocation = imagePathes[index];
            tbExpectedResult.Text = expectedResults[index].ToString();
            var imageSide = (int)nudImageSide.Value;
            pictureConverter.ConvertedImageSide = imageSide;
            blackWhitePixels =
pictureConverter.Convert(imagePathes[currentIndex]);
            pbBlackWhite.Image =
pictureConverter.ToBitmap(blackWhitePixels);
```

```
            lblFilePath.Text = imagePathes[index];
            lblCurrentIndex.Text = (index + 1).ToString();
            lblTotal.Text = imagePathes.Count.ToString();
            lblResult.Visible = false;
        }

        private void fileToolStripMenuItem_Click(object sender, EventArgs
e)
        {
            if (ofdInputFile.ShowDialog() == DialogResult.OK)
            {
                imagePathes.Add(ofdInputFile.FileName);
                expectedResults.Add(1);
                ShowImage(imagePathes.Count - 1);
            }
        }

        private void folderToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if (fbdInputFolder.ShowDialog() == DialogResult.OK)
            {
                var fileNames =
Directory.GetFiles(fbdInputFolder.SelectedPath);
                var lastIndex = imagePathes.Count;
                foreach (var fileName in fileNames)
                {
                    imagePathes.Add(fileName);
                    expectedResults.Add(1);
                }
                ShowImage(lastIndex);
            }
        }

        private void SaveExpectedResult()
        {
            try
            {
                expectedResults[currentIndex] =
Convert.ToDouble(tbExpectedResult.Text);
            }
            catch (FormatException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private void pbBlackWhite_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.NearestNeighbor;
```

```
            e.Graphics.PixelOffsetMode =
System.Drawing.Drawing2D.PixelOffsetMode.Half;
            if (pbBlackWhite.Image != null)
            {
                var imageRectangle = new Rectangle(0, 0,
pbBlackWhite.Width, pbBlackWhite.Height);
                e.Graphics.DrawImage(pbBlackWhite.Image, imageRectangle);
            }
        }

        private void pbImage_Paint(object sender, PaintEventArgs e)
        {
            e.Graphics.InterpolationMode =
System.Drawing.Drawing2D.InterpolationMode.NearestNeighbor;
            e.Graphics.PixelOffsetMode =
System.Drawing.Drawing2D.PixelOffsetMode.Half;
            if (pbImage.Image != null)
            {
                var imageRectangle = new Rectangle(0, 0,
pbBlackWhite.Width, pbBlackWhite.Height);
                e.Graphics.DrawImage(pbImage.Image, imageRectangle);
            }
        }

        private async void btnLearn_Click(object sender, EventArgs e)
        {
            var hiddenLayerCount = (int)nudHiddenLayerCount.Value;
            var epochCount = (int)nudEpochCount.Value;
            double learningRate = 0.1;
            try
            {
                learningRate = Convert.ToDouble(tbLearningRate.Text);
                if (learningRate > 0)
                {
                    var imageSide = (int)nudImageSide.Value;
                    var imageSize = imageSide * imageSide;
                    pictureConverter.ConvertedImageSide = imageSide;
                    var inputs = GetData(imageSize);
                    var topology = new Topology(imageSize, 1,
learningRate, hiddenLayerCount);
                    neuralNetwork = new NeuralNetwork(topology);

                    StartLoading();
                    await Learn(expectedResults.ToArray(), inputs,
epochCount);
                    MessageBox.Show("Learning ended");
                    StopLoading();
                }
                else
                {
```

```
                    MessageBox.Show("Learning rate should be greater then
0");
                }
            }
            catch (FormatException ex)
            {
                MessageBox.Show(ex.Message);
            }
        }

        private Task Learn(double[] expected, double[,] inputs, int
epochCount)
        {
            return Task.Run(() =>
            {
                neuralNetwork.Learn(expected, inputs, epochCount);
            });
        }

        private void StartLoading()
        {
            // выключает кнопку для показа результатов и добавления
записи
            btnLearn.Enabled = false;
            btnShowResults.Enabled = false;
        }

        private void StopLoading()
        {
            btnLearn.Enabled = true;
            btnShowResults.Enabled = true;
        }

        private double[,] GetData(int imageSize)
        {
            var size = imagePathes.Count;
            var result = new double[size, imageSize];
            for (int i = 0; i < size; i++)
            {
                var image = pictureConverter.Convert(imagePathes[i]);
                for (int j = 0; j < image.Count; j++)
                {
                    result[i, j] = image[j];
                }
            }
            return result;
        }

        private void btnShowResults_Click(object sender, EventArgs e)
        {
```

```
            var result = neuralNetwork.Predict(blackWhitePixels.Select(t
=> (double)t).ToArray());
            lblResult.Text = Math.Round(result.Output, 2).ToString();
            lblResult.Visible = true;
        }
    }
}
```

8.2 Тестирование приложения

Загрузив на форму изображения, в поле Expected впишем ожидаемый результат (для кругов 0, для квадратов 1). Задав параметры, запустим обучение нейронной сети. Результат обучения можно будет посмотреть при нажатии на кнопку Show results для каждого изображения.

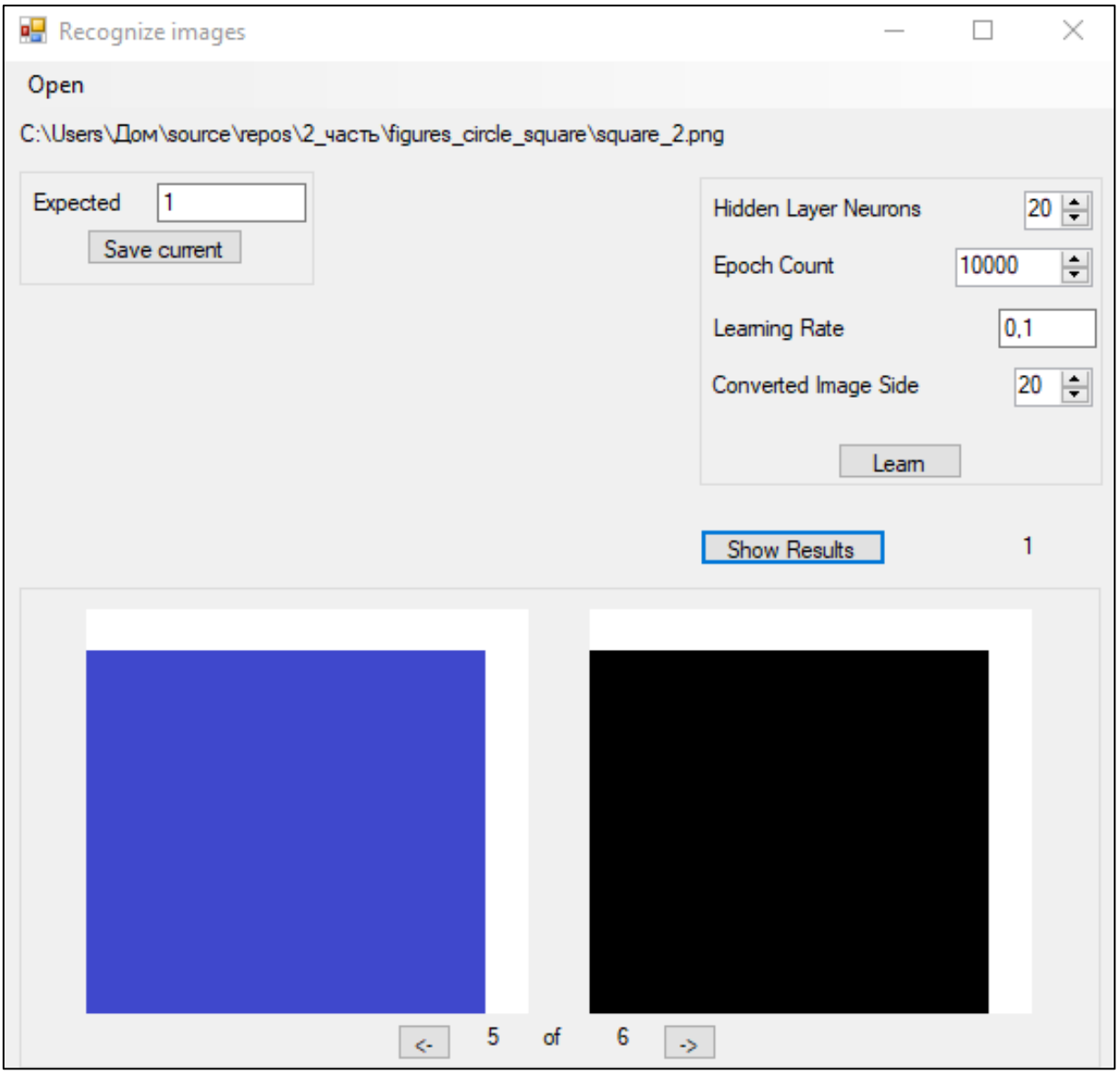


Рисунок 8.4 – Результат обучения

Задания для самостоятельной работы

1. Запустить приложение.
2. Выполнить обучение с предложенными рисунками.
3. Исследовать, как зависит скорость обучения от коэффициента обучения.
4. Исследовать, как зависит результат обучения от числа нейронов на скрытом слое.
5. Исследовать, как зависит скорость обучения от числа эпох.
6. Подготовить папку со своими изображениями. Выполнить для них пункты 2–5.

ЛИТЕРАТУРА

1. Искусственный интеллект : методические указания к практическим занятиям / УО «ВГТУ» ; сост.: Е. Б. Дунина, В. А. Жизневский, А. С. Соколова. – Витебск, 2020. – 47 с.

2. Сайт «CodeBlog» [Электронный ресурс] / Искусственный интеллект C#. Обучение нейронных сетей. Алгоритм обратного распространения ошибки – Режим доступа: https://www.youtube.com/playlist?list=PLIIXgDT0bKw7VnfMl5wWW_kjmmCrHjfvb. – Дата доступа: 21.04.2023.

3. Сайт «CodeBlog» [Электронный ресурс] / Искусственный интеллект и нейронные сети C#. Информационная система медицинской организации – Режим доступа: https://www.youtube.com/playlist?list=PLIIXgDT0bKw7VnfMl5wWW_kjmmCrHjfvb. – Дата доступа: 21.04.2023.

4. Сайт «Proglib» [Электронный ресурс] / Наглядное введение в нейросети на примере распознавания цифр – Режим доступа: https://proglib.io/p/neural-network-course. – Дата доступа: 21.04.2023.

5. Сайт «RomanShamin» [Электронный ресурс] / Научный канал Романа Шамина – Режим доступа: https://www.youtube.com/@romanshamin119/featured. – Дата доступа: 21.04.2023.

6. Сайт «Github» [Электронный ресурс] / NeuralNetwork.cs – Режим доступа: https://github.com/shwanoff/neuralnetworks/blob/master/NeuralNetworks/NeuralNetwork.cs. – Дата доступа: 21.04.2023.

7. Доррер, М. Г. Моделирование нейронных сетей в системе MatLab : лабораторный практикум / М. Г. Доррер. – Красноярск: Изд-во СибГУ им. М. Ф. Решетнева, 2021. – 98 с.

8. Дунина, Е. Б. Разработка приложения распознавания образов / Е. Б. Дунина, М. Л. Лапко, А. А. Корниенко // Тезисы докладов 54-й Международной научно-технической конференции преподавателей и студентов / УО «ВГТУ». – Витебск, 2021. – С. 221–222.

Учебное издание

Дунина Елена Брониславовна
Соколова Анна Сергеевна
Корниенко Алексей Александрович

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Учебно-методическое пособие

Редактор *Р.А. Никифорова*
Компьютерная верстка *А.С. Соколова*

Подписано к печати 02.02.2024. Формат 60x90 $^{1}/_{16}$. Усл. печ. листов 5,6.
Уч.-изд. листов 6,8.Тираж 30 экз. Заказ № 45.

Учреждение образования «Витебский государственный технологический университет»
210038, г. Витебск, Московский пр., 72.
Отпечатано на ризографе учреждения образования
«Витебский государственный технологический университет».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 1/172 от 12 февраля 2014 г.
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий № 3/1497 от 30 мая 2017 г.