Deep learning Neural Network in .Net core

Name: Firealem Alemayehu
email: falemaye@stud.fra-uas.de

Abstract— the objective of this project is to recognize hand written digits from an image by implementation of AForge.NET frame work. AForge.NET is an open source C# framework.

Introduction

It is known fact, that there are many different problems, for which it is difficult to find formal algorithms to solve them. Some problems cannot be solved easily with traditional methods; some problems even do not have a solution yet. For many such problems, neural networks can be applied, which demonstrate rather good results in a great range of them. Neural networks can be applied to such tasks, like classification, recognition, approximation, prediction, clusterization, memory simulation, and many other different tasks, and their amount is growing. [2]

In this project, AForge.NET frame work is used for recognition of hand written digits. For validation and testing MNIST data sets have been used. AForge.NET is an open source C# framework designed for developers and researchers in the fields of Computer Vision and Artificial Intelligence - image processing, neural networks, genetic algorithms, fuzzy logic, machine learning, robotics, etc.[1] The library is designed in a such a way that to be flexible, reusable, and easy to use and understand.

.

- **What Is a Neural Network**

"a computing system made up of a number of simple, highly interconnected processing elements, which process information by their dynamic state response to external inputs"[3]. In the most general form, a neural network is a machine designed to model the way in which our brain can perform a particular task of interest. The heart of a neural network is the learning algorithm, whose function is to modify the weights of the neural network in an optimized manner to achieve the desired objective. In a neural network we don't tell the computer how to solve our problem. Instead, it learns from observational data, figuring out its own solution to the problems presented to it.

- **The architecture of neural network**

The architecture of neural networks is shown below in figure 1. The leftmost yellow circles represent what we call the network's Input layer where each circle represents an input neuron. The rightmost red layer is known as the output layer and contains the output neurons, as in this case two output neurons. The middle layer is called a hidden layer, since the neurons in this layer are neither inputs nor outputs.[4] The example below has only one single hidden layer made of five neurons, but networks can have multiple hidden layers.

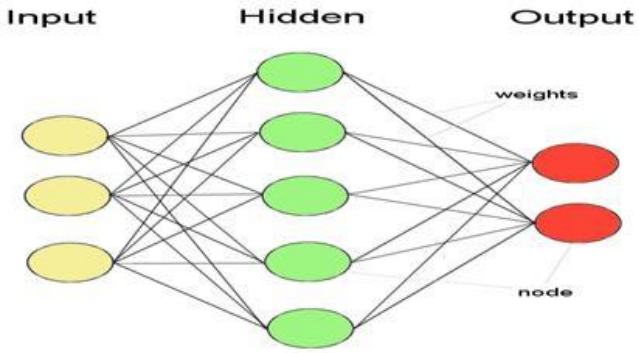


Figure 1.architecture of neural network

The hidden layer is a group of neurons which has activation function applied on it and it is an intermediate layer found between the input layer and the output layer. Its duty is to process the inputs obtained by its previous layer. So that it is the layer which is responsible for extracting the required features from the input data.

- **What is deep learning**

The field of artificial intelligence is essentially when machines can do tasks that typically require human intelligence. It encompasses machine learning, where machines can learn by experience and acquire skills without human involvement. Deep learning is a subset of machine learning where artificial neural networks, algorithms inspired by the human brain, learn from large amounts of data. Similarly to how we learn from experience, the deep learning algorithm would perform a task repeatedly, each time tweaking it a little to improve the outcome. We refer to 'deep learning' because the neural networks have various (deep) layers that enable learning. Just about any problem that requires "thought" to figure out is a problem deep learning can learn to solve. Deep learning allows machines to solve complex problems even when using a data set that is very diverse, unstructured and inter-connected. The more deep learning algorithms learn, the better they perform. [6]

- **AForge.NET frame work**

The framework has different libraries and here for neural network computations is described [1]. The library implements several popular neural network architectures and their training algorithms, like Back Propagation, Kohonen

Self-Organizing Map, Elastic Network, Delta Rule Learning, and Perceptron Learning.

The library contains six main entities:

- Neuron - a base abstract class for all neurons, which encapsulates such common entities like a neuron's weight, output value, and input value. Other neuron classes inherit from the base class to extend it with additional properties and specialize it.
- Layer - represents a collection of neurons. This is a base abstract class, which encapsulates common functionality for all neuron's layers.
- Network - represents a neural network, what is a collection of neuron's layers. This is a base abstract class, which provides common functionality of a generic neural network. To implement a specific neural network architecture, it is required to inherit the class, extending it with specific functionalities of any neural network architecture.
- IActivationFunction - activation function's interface. Activation functions are used in activation neurons - the type of neuron, where the weighted sum of its inputs is calculated and then the value is passed as input to the activation function, and the output value becomes the output value of the neuron.
- IUnsupervisedLearning - interface for unsupervised learning algorithms - the type of learning algorithms where a system is provided with sample inputs only during the learning phase, but not with the desired outputs. The aim of the system is to organize itself in such a way to find correlation and similarities between data samples.
- ISupervisedLearning - interface for supervised learning algorithms - the type of learning algorithms where a system is provided with sample inputs, with desired output values during the learning phase. The aim of the system is to generalize learning data, and learn to provide the correct output value when it is presented with the input value only.

The library provides the following neural network architectures:

- Activation Network - the neural network where each neuron computes its output as the activation function's output, and the argument is a weighted sum of its inputs combined with the threshold value. The network may consist of a single layer, or of multiple layers. Trained with supervised learning algorithms, the network allows to solve such tasks as approximation, prediction, classification, and recognition.
- Distance Network - the neural network where each neuron computes its output as a distance between its weight values and input values. The network consists of a single layer, and may be used as a base for such networks like Kohonen Self Organizing Map, Elastic Network, and Hamming Network.

Different learning algorithms are used to train different neural networks, and are used to solve different problems:

- Perceptron Learning - the algorithm may be considered as the first neural network learning algorithm, and its history starts from 1957. The algorithm may be used with a one-layer activation network, where each neuron has a threshold activation function. The range of its applications are rather small and limited the with classification of linearly separable data.
- Delta Rule Learning - the algorithm is a next step after the perceptron learning algorithm. It utilizes the activation function's derivative, and may be applicable to single-layer activation networks only, where each neuron has a continuous activation function instead of a threshold activation function. The most popular continuous activation function is the unipolar and bipolar sigmoid function. Because the algorithm may be applied to one-layer networks only, it is limited to some classification and recognition tasks mostly.
- Back Propagation Learning - this is one of the most popular and known algorithms for multi-layer neural network learning. Initially, it was described in 1974, and from that time, it was extensively studied and applied to a broad range of different tasks. Because the algorithm is able to train multi-layer neural networks, the range of its applications is very great, and includes such tasks as approximation, prediction, object recognition, etc.
- SOM Learning - this algorithm was developed by Kohonen, and may be considered as one of the most famous unsupervised learning algorithms for clusterization problems. It treats neural network as a 2D map of nodes, where each node may represent a separate class. The algorithm organizes a network in such a way, that it becomes possible to find the correlation and similarities between data samples.
- Elastic Network Learning - the algorithm is similar to the idea of the SOM learning algorithm, but it treats network neurons not as a 2D map of nodes, but as a ring. During the learning procedure, the ring gets some shape, which represents a solution. One of the most common demonstrations of this learning algorithm is the Traveling Salesman Problem (TSP).

For this project, Perceptron learning is used for neural network architectures and their training algorithm and bipolar Sigmoid is used as Activation function. The network performance is far better compared to back propagation learning algorithm. It will be discussed in a unit test part. Architecture of the project solution and flow charts are attached as appendix.

The code of project

The code of project structure is shown in Fig. 1a below and written in C# programming language. The name of the Solution is neuralnetwork and it contains two projects named DigitRecognizer and test.The project also uses System.drawing.common to convert image to double array of pixels, AForge to creat neural network and LearningApi as a

nuget package. This project is actually a module of LearningApi and from our test project we are calling the project through LearningApi object as per structure of LearningApi.

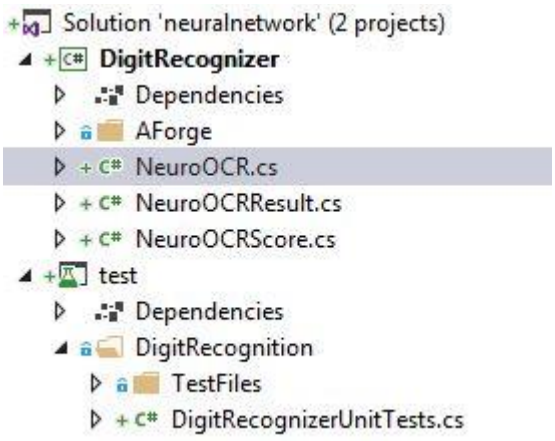


Fig.1a Project Structure

The functionalities of the main classes of the project are explained below:

- **Namespace DigitRecognizer**

This contains the neural learning algorithm. A brief description of the important class contained is given below:

1. **NeuroOCR (AForge.Neuro.IActivationFunction function, int inputsCount, int neuronCount, double learningRate)**

This is the constructor of class and it is used for creation of neural network and specifying the activation function, number of neurons in each three layers.

As a user of this project, within this class, you have to define two things two create your own neural network:

1. Learning Mechanism: you have the possiblilty to choose one of the learning mechanisms e.g. Pereceptron, backpropagation and so on which is you can find it at the beggninng of this class (see fig 1c).e.g.

AForge.Neuro.Learning.PerceptronLearning learning; // Learning mechanism.

2. You have to initialize as well the learning (see fig 1c).e.g.

learning = new AForge.Neuro.Learning.PerceptronLearning(network); // initialization of learning mechanism.

The screenshot below (in fig.1c) shows where exactly to change the learning mechanism and where to initialize it within this class.If you want to change the learning mechanism only you have to change this above mensioned part of the code.

e.g. for backpropagation learning

AForge.Neuro.Learning.BackPropagationLearning learning; // Learning mechanism

learning=newAForge.Neuro.Learning.BackPropagationLearning(network); // initialization of learning mechanism .

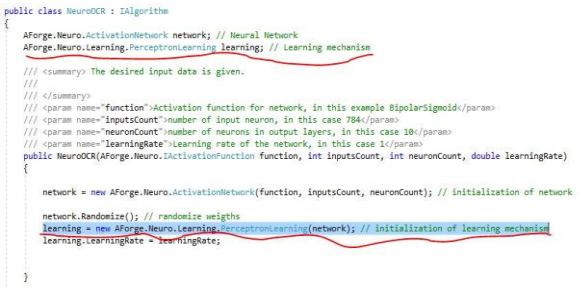


Fig.1c choosing and initialization of learning algorithm of a network

2. **IResult Predict (double [ ][ ]data , Icontext ctx)**

This method in **NeuroOCR** class recognizes (predict) the digit from an input pixel array. IResult is the return type method of interface called IAlgorithm of learning api. A class named **NeuroOCRResult** is created which implements **IResult** interface. In this way we can return our result through learning api and later we can cast IResult instance back to NeuroOCRResult to obtain actual result that is done in test method.

Predict method accepts an array of input pixels arrays and create an array of one-dimension arrays of type double. It loops through the pixel array and crates a new array by replacing with 1 for those input pixel values having greater than zero else put -1.The reason of converting input pixel array into 1 or -1 is that, the input data we have as an image or MNIST data file is color data pixels having value that ranges from 0 to 255.We have to determine what is background and what is foreground, because we only want those pixels that made the number not the blank background. So that if the given pixel contain data greater than zero means it has something different from background that means this pixel is used in our handwritten digit, thus we convert it into 1 and all others to -1.We have trained a network in such a way that which maps 1 in input to 1 in output, so we use same pattern for prediction.

The output array length is 10, which means from index 0 to 9.The trained network computes where output gets its maximum value (i.e. 1) and returns the index of it. Then that index will be the predicted digit. For example, if 1 is present at index seven our given number is seven.

3. **IScore Run (double [ ][ ]data , Icontext ctx)**

This method in **NeuroOCR** class trains the network by providing training input and output data. A class named **NeuroOCRScore** is created which implements **IScore** interface. **IScore** is the return type method of interface called IPipleline of learning api. The training is done in bulk, that means multiple inputs and multiple outputs is provided to network at same time in jagged arrays (i.e. array of arrays). The line double [][] input = new double[data.length][]; means there will be 60000 array of input data(MNIST train dataset has 60000). So we just created an array of length 60,000 each with a single element which is also an array (the next [] sign in the line). each array of array (All 60,000) are

empty for now, so we have 60,000 empty arrays as the elements of an array. The same way we will have 60,000 empty arrays of output. After creating arrays of array of input and output, it will loop through data one by one from 0 to 60,000 and initialize the new created input and output arrays.

The line input[i] = new double [network.InputsCount] creates empty array of length 784(previously provided to network as inputs count) at the ith element of input array. So when loop ends we have 60,000 empty arrays of length 784 each at each index of input array, in simple terms we have one array which contains 60,000 elements and each element is itself an array of 784 elements (i.e. array of array).The same way, creates empty array of length 10 elements for each 60,000 elements of output array and fill the output array by -1 as a starting point (default) latter on will be changed through the next steps.

Now, we have to fill empty array with actual data that is 1 and -1, for both input and output. For each element in input is array of 784 and we will put 1 where we have pixel data greater than zero otherwise we will put -1.the same thing will be done for the output as well. A new array will be created by combining the above input and output data converted to array of -1 and 1 to train the network.

Now, the data array (parameter of method) contains of 60,000 elements, each element is array itself contains 785 elements, the first elements of 785 is the output number i.e. any number from 0 to 9 that is represented by next 784 pixels. we have output and pixel data in one array, we need to put output in output and pixels data in input array so we created two loops, first go through from 0 to 60,000 and second go through from 1 to 785 in the data array. The second loop starts at 1 because 1st is output and we don't need output in the input array.

Then if given pixel has data greater than zero we put input array given index with 1 otherwise -1. The line input[i][j-1] means ith array of 60,000 elements in input array and then j-1 index of ith array should be filled with 1 or -1, why j-1 because data array has 785 elements while input count is 784 and we started at 1, so minus 1 index to put it in our array so after this out and input loop will have 1's and -1's.

So in the end of outer loop, we are taking the first value from the data array (i.e. output). For example let us say output is 3, and then in the next line we put 1 at index three since all others are already -1. That is the reason we filled all output array with -1 before as a default and now we just change only one value.

Next we create a while loop that runs 1000 times and train network or stops before if a network is trained with 0 error rate (almost impossible in real data). So we provide input and output to Perceptron learning and it returns the error rate, error rate means the difference between actual output and the output that network calculated (predicted). If error is zero simply stop learning otherwise keep training the network, after 1000 just stop it. If you need to train the network more change 1000 to greater number but it will take more time.

Errors are added in the error list and returned as IScore; so that the called method can see the performance of network.

- **Class DigitRecognizerUnittests:**

This class is used to make different unit tests and measure the algorithm performance by training and testing with image and MNIST data. Here below one of the main methods functionality is described:

Leaningapi TrainNetworkWithMNIST_TrainingData()

This method will train the DigitRecogziner network using LearningApi with MNIST data set and creates new instance of LearningApi, use it to train a network and return the instance as api, so api can be used by caller of this method to predict and test the results.

api.UseActionModule<object, double[][]>((notUsed, ctx) => :is used to provide data to our network through learning api and this method is defined in LearningApi Project . so if our data is single array we can change double[][] to double[], if our data is a string we can change it to string , so it provides dynamic data handling to the module .

api.AddModule(newNeuroOCR(new Forge.Neuro.BipolarSigmoidFunction(2), 784, 10, 1)):here we have added our module to api with all passed parameters. This line adds our module to the LearningApi so we can use it through learning api, you can think of it that we are registering our module with learning api to use it.

IScore score = api.Run() as IScore: this line has no parameters while our module Run method has a parameter which accepts data array. so when this line is called, it get the registered module, call its Run method internally (Learning Api use reflection for this) and then learning api call UseActionModule method defined above that returns the data, then that data will be used as a parameter to train network. Finally a trained network will be returned as api to a caller to this method.

- **Different Unit tests and results**

Different unit tests had been run by taking different inputs and also validation tests. The performance of each unit tests will be presented as follow:

Unit test 1:

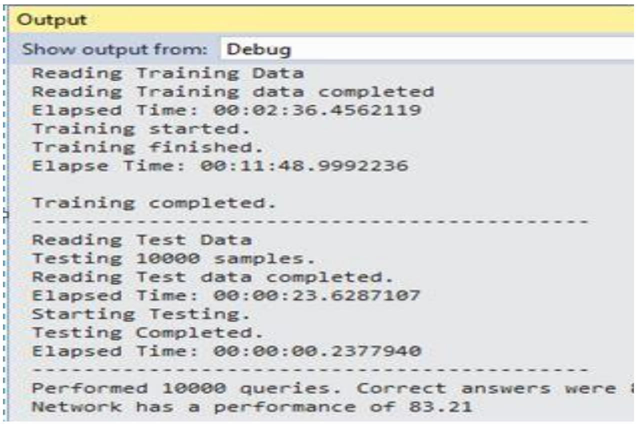
MNIST_Test_WithImages

This test consists of the MNIST dataset in original image form for training and testing. Training data are present in file called 'mnist_train_images.csv'

The learning algorithm was run on the above training data set, with Learning rate = 1, number of input nodes = 784, hidden layer neurons = 10 and bipolar sigmoid activation function.

The test data is present in file 'mnist_test_images.cs'

Test Result: The network is working with efficiency of **83.21%.**



```
Output
Show output from: Debug
Reading Training Data
Reading Training data completed
Elapsed Time: 00:02:36.4562119
Training started.
Training finished.
Elapse Time: 00:11:48.9992236

Training completed.
---------------------------------------------
Reading Test Data
Testing 10000 samples.
Reading Test data completed.
Elapsed Time: 00:00:23.6287107
Starting Testing.
Testing Completed.
Elapsed Time: 00:00:00.2377940
---------------------------------------------
Performed 10000 queries. Correct answers were
Network has a performance of 83.21
```

Figure 2a.Unit test 1 output capture

```
public void MNIST_Test_WithImages()
{

    Debug.WriteLine("----------MNIST_Test_WithImages----------");
    var api = Train_Network_With_MNISTIMagesTrainingData();
    //var api = Train_Network_With_MNISTIMagesTrainingData(); // To Train with images
    Stopwatch stopwatch = new Stopwatch();
    Debug.WriteLine("Reading Test Data");
    stopwatch.Start();

    var testDataFile = $"{Directory.GetCurrentDirectory()}\\DigitRecognition\\TestFiles\\mnist_test_images.csv";
    if (File.Exists(testDataFile) == false)
    {
        testDataFile = $"{Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName}\\DigitRecognition\\TestFiles\\mnist_test_
    }
    // Read the csv file which contains the test data
    using (var readerTestData = new StreamReader(testDataFile))
    {
        int lineCountTestData = 0;

        int indexTestData = 0;

        while (readerTestData.ReadLine() != null)
        {
            lineCountTestData++;
```

Figure 2b unit test 1 code capture

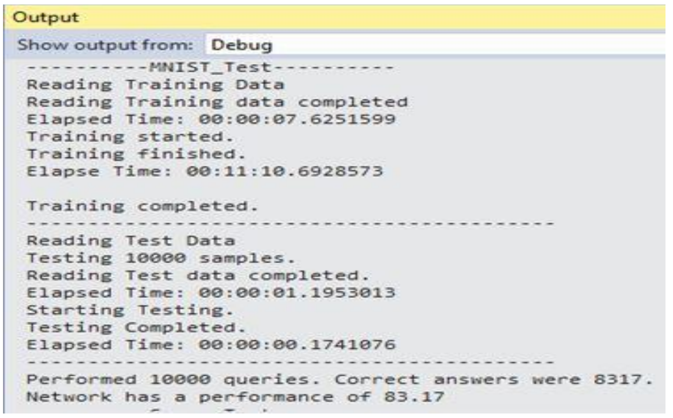
Unit test 2:

MNIST_Test

This test consists of the mnist dataset in numeric form for training and testing. Training data is present in file 'mnist_train.csv'.

The learning algorithm was run on the above training data set, with Learning rate = 1, number of input nodes = 784, hidden layer neurons = 10 and bipolar sigmoid activation function.The test data is present in file 'mnist_test.cs'

Test Result: The network is working with efficiency of **83.17%**



```
Output
Show output from: Debug
----------MNIST_Test----------
Reading Training Data
Reading Training data completed
Elapsed Time: 00:00:07.6251599
Training started.
Training finished.
Elapse Time: 00:11:10.6928573

Training completed.
---------------------------------------------
Reading Test Data
Testing 10000 samples.
Reading Test data completed.
Elapsed Time: 00:00:01.1953013
Starting Testing.
Testing Completed.
Elapsed Time: 00:00:00.1741076
---------------------------------------------
Performed 10000 queries. Correct answers were 8317.
Network has a performance of 83.17
```

Figure 3a.Unit test 2 output capture

```
public void MNIST_Test()
{

    Debug.WriteLine("----------MNIST_Test----------");
  var api = TrainNetworkWithMNIST_TrainingData();
    Stopwatch stopwatch = new Stopwatch();
    Debug.WriteLine("Reading Test Data");
    stopwatch.Start();

    var testDataFile = $"{Directory.GetCurrentDirectory()}\\DigitRecognition\\TestFiles\\mnist_test.csv";
    if (File.Exists(testDataFile) == false)
    {
        testDataFile = $"{Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName}\\D
    }
    // Read the csv file which contains the test data
    using (var readerTestData = new StreamReader(testDataFile))
    {
        int lineCountTestData = 0;

        int indexTestData = 0;

        while (readerTestData.ReadLine() != null)
        {
            lineCountTestData++;
        }
        Debug.WriteLine($"Testing {lineCountTestData} samples.");

        double[][] testData = new double[lineCountTestData][];

        List<double>[] listTestData = new List<double>[lineCountTestData];
```

Figure 3a.Unit test 2 code capture

Unit test 3:

Seven_Test

This test consists of the mnist dataset in numeric form for training and manual noisy data of digit 7 for testing.. Training data is present in file 'mnist_train.csv'.

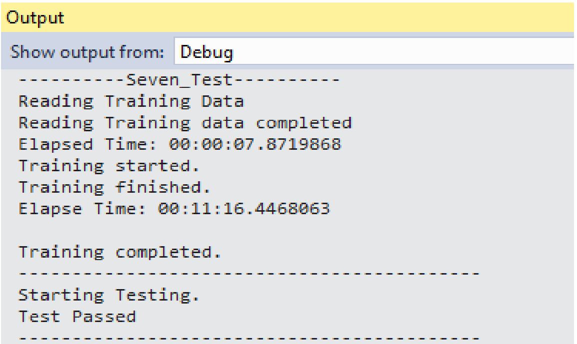
The learning algorithm was run on the above training data set, with Learning rate = 1, number of input nodes = 784, hidden layer neurons = 10 and bipolar sigmoid activation function.

The test data is given below.

testData[0] = new double[] { 7, 0, 84, 185, 159, 151, 60, 36, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 222, 254, 254, 254, 254, 241, 198, 198, 198, 198, 198, 198, 198, 198, 170, 52, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 67, 114, 72, 114, 163, 227, 254, 225, 254, 254, 254, 250, 229, 254, 254, 140, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 17, 66, 14, 67, 67, 67, 59, 21, 236, 254, 106, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 83, 253, 209, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 22, 233, 255, 83, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 129, 254, 238, 44, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 59, 249, 254, 62, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 133, 254, 187, 5, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 9, 205, 248, 58, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 126, 254, 182, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 75, 251, 240, 57, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 19, 221, 254, 166, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 203, 254, 219, 35, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 38, 254, 254, 77, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 31, 224, 254, 115, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 133, 254, 254, 52, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 61, 242, 254, 254, 52, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 121, 254, 254, 219, 40, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 121, 254, 207, 18, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

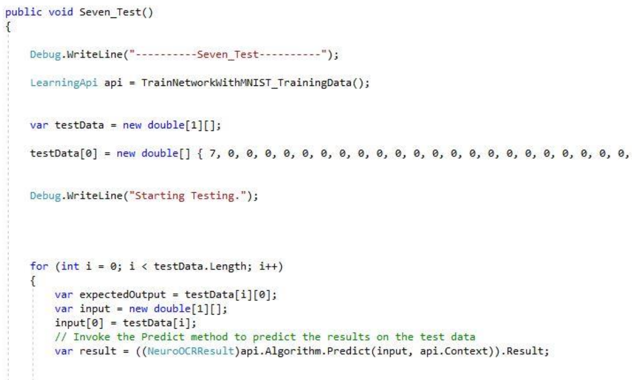
Test Result: The network correctly identifies the input digit.



```
Output
Show output from: Debug
----------Seven_Test----------
Reading Training Data
Reading Training data completed
Elapsed Time: 00:00:07.8719868
Training started.
Training finished.
Elapse Time: 00:11:16.4468063

Training completed.
-------------------------------------------
Starting Testing.
Test Passed
-------------------------------------------
```

Figure 4a.Unit test 3 output capture



```
public void Seven_Test()
{
    Debug.WriteLine("----------Seven_Test----------");

    LearningApi api = TrainNetworkWithMNIST_TrainingData();


    var testData = new double[1][];

    testData[0] = new double[] { 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,


    Debug.WriteLine("Starting Testing.");



    for (int i = 0; i < testData.Length; i++)
    {
        var expectedOutput = testData[i][0];
        var input = new double[1][];
        input[0] = testData[i];
        // Invoke the Predict method to predict the results on the test data
        var result = ((NeuroOCRResult)api.Algorithm.Predict(input, api.Context)).Result;
```

Figure 4a.Unit test 3 code capture

Unit test 4:

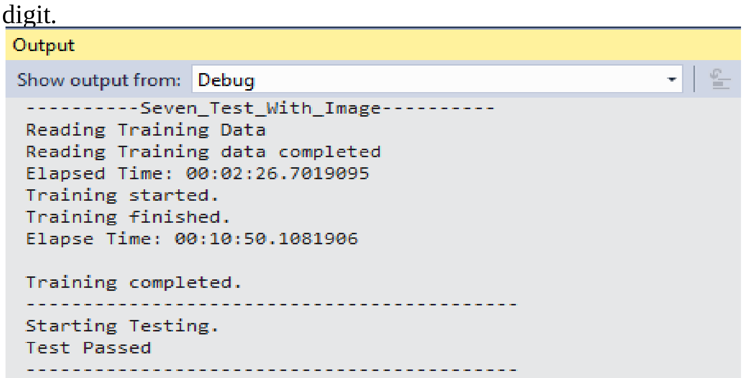
Seven_Test_With_Image

This test consists of the mnist dataset in numeric form for training and manual noisy data in image form of digit 7 for testing.. Training data is present in file 'mnist_train.csv'.

The learning algorithm was run on the above training data set, with Learning rate = 1, number of input nodes = 784, hidden layer neurons = 10 and bipolar sigmoid activation function.

The test data is given below.

Test Result: The network correctly identifies the input digit.



```
Output
Show output from: Debug
----------Seven_Test_With_Image----------
Reading Training Data
Reading Training data completed
Elapsed Time: 00:02:26.7019095
Training started.
Training finished.
Elapse Time: 00:10:50.1081906

Training completed.
-------------------------------------------
Starting Testing.
Test Passed
-------------------------------------------
```

Figure 5a.Unit test 4 capture

```
public void Seven_Test_With_Image()
{
    Debug.WriteLine("----------Seven_Test_With_Image----------");


    string trainingDataFile = $"{Directory.GetCurrentDirectory()}\\DigitRecognition\\TestFiles\\mnist_train_images.csv";
    if (File.Exists(trainingDataFile) == false)
    {
        trainingDataFile = $"{Directory.GetParent(Directory.GetCurrentDirectory()).Parent.Parent.FullName}\\DigitRecognit
    }

    var testData = new double[1][];

    var imgFilePath = Path.Combine(Directory.GetParent(trainingDataFile).FullName, "Test Images\\74.jpeg");


    testData[0] = new double[785];
    testData[0][0] = 7;
    GetImageData(new System.Drawing.Bitmap(System.Drawing.Bitmap.FromFile(imgFilePath))).CopyTo(testData[0], 1);

    LearningApi api = Train_Network_With_MNISTIMagesTrainingData();

    Debug.WriteLine("Starting Testing.");

    for (int i = 0; i < testData.Length; i++)
    {
        var expectedOutput = testData[i][0];
        var input = new double[1][];
        input[0] = testData[i];
        // Invoke the Predict method to predict the results on the test data
```

Figure 5b.Unit test 4 code capture

- **Conclusion**

A neural network having a maximum performance of 83.21% achieved for recognition of hand written digits using Perseptron learning algorithm, with Learning rate = 1, number of input nodes = 784, hidden layer neurons = 10, bipolar sigmoid activation function and 10,000 MNIST training data.

Aforge framework provides flexible options for a user to create neural network topologies according to the goal of a particular project.

REFERENCES

[1]. open source C# framework designed for developers and researchers in the fields of Computer Vision and Artificial Intelligence - image processing, neural networks, genetic algorithms, fuzzy logic, machine learning and robotics <http://www.aforgenet.com/framework/>

[2]NeuralnetworkinC# <https://www.codeproject.com/Articles/16447/Neural-Networks-on-C?msg=1952410#xx1952410xx >

[3]. In "Neural Network Primer: Part I" by Maureen Caudill, AI Expert, Feb. 1989

[4]. Neural Networks and Deep Learning – Michael Nielsen

[5]. Deep analysis of Perceptron in Data science <https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53>

[6]. Deep Learning Artificial Intelligence with examples <https://www.forbes.com/sites/bernardmarr/2018/10/01/what-is-deep-learning-ai-a-simple-guide-with-8-practical-examples/#722cfe688d4b>