IMPLEMENT CENTER MODULE FOR LEARNING API

Scherr Valerian
v.scherr@gmx.de

Abstract— **The purpose of this paper is to present a simple but efficient algorithm that calculates the center of a data set and, if necessary, moves the data to that center.**
The calculation of the center of an image is done by calculating the center of the data of the data set that belongs to the image.
In order to calculate the center of the actual data set, the center of the matrix of this data set must be calculated. The distance of both calculated values to each other results in the shift, by which the data must be shifted to the center of the data set.

Keywords—Center Module, Implementation, Learning Api[1]

I. INTRODUCTION

The project “Implementation Center Module for Learning Api” is part of the module “Software Engineering” of the Frankfurt University of Applied Sciences. The goal of this project is to calculate the center of a dataset whose values belong to an image. If the data within this dataset is not in the center of the dataset, it should be moved to the center.
Data, especially images or digits, are usually not centered in the overall image. For this purpose, an algorithm is to be developed that automatically moves the data to the center of the dataset.
As an example, the centering of digits from the MNIST dataset of handwritten digits was given.
For this, the digits are transferred into a double [][] array, which is binarized. After binarization, the image consists only of ones and zeros. Now the ones can be centered between the zeros. Finally, the algorithm will be integrated into the Learning Api of the Frankfurt University of Applied Sciences. The Learning Api consists of different modules for machine learning and image processing. This project uses the interface "IPipeline module".

.

II. METHODS

The following section describes the methods, which are required to develop the algorithm. This includes what a MNIST digit is, how I can calculate the average or mean,how it is binarized, and how the center of the data set is calculated.

A. MNIST Database of handwritten Digits

MNIST (Modified National Institute of Standards and Technology) Database is a big Database of Handwritten Digits. The database is mainly used for training and testing in the field of machine learning. The MNIST database consists of 60,000 training images and 10,000 testing images. Each dataset consists of 2 files. One for the pixels and one for the labels. When loading the single images both files have to be read at the same time.
When loading the single Digits, they are reduced to a size of 28x28 pixels. The following picture shows a MNIST digit. [2]

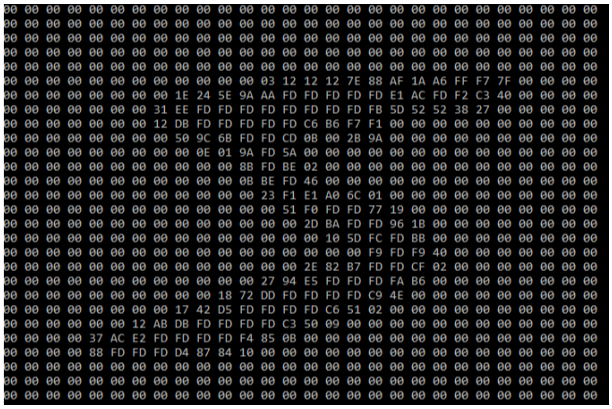


Figure 1: MNIST-Digit of number 5

B. Average

The average is basic statistical measure. It provides quick and easy information about general level of values in a data set The calculation of arithmetic average is straightforward. You sum up all the values and then divide the sum by the number of values.

$$\bar{x}_{\mathbf{arithm}} = \frac{1}{n}\sum_{i=1}^{n} x_i = \frac{x_1 + x_2 + \cdots + x_n}{n}$$

[3]

C. Binarization of the MNIST Digits

After they are loaded, the individual MNIST digits are transferred to a double [][] array with the size 28x28

(corresponding to their pixel size). However, the data is still in hexadecimal format. To replace them in with ones and zeros, a mean value is formed over the entire content. Everything that is smaller than the mean value is replaced with a zero, everything that is larger is replaced with a one. This makes it easier for the algorithm to determine the center point.

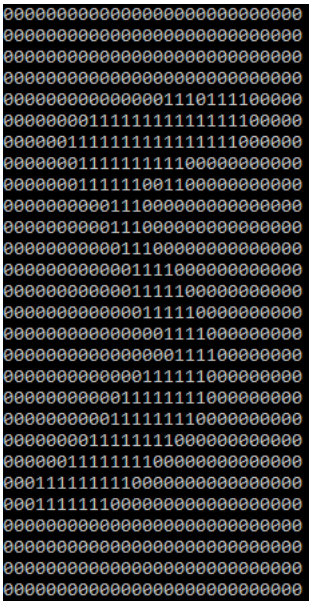


Figure 2: Binarized MNIST-Digit

D. Calculating Center of Dataset

The Calculation of the center of the Dataset is by Calculating the mean of the dataset.
The mean is the sum of the numbers in a dataset divided by the total number of values in the dataset. The mean can be used to find the center of data when the numbers in the dataset are fairly close together. In a vector, the mean value of all entries that are not 0 is calculated. [4]

III. Results

A. Architecture

The following block diagram shows the architecture and the structure of the third section. First, the two classes are explained with which the test data (MNIST data set) can be read in. Then the CenterAlgorithm is explained in more detail, which is used to center the test data. Finally, the Unit-test of the CenterAlgorithm is explained in more detail.

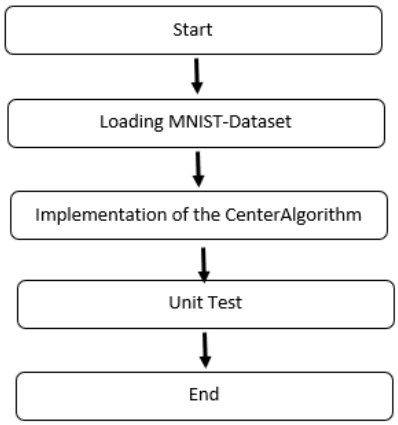


Figure 3: The Architecture of the Results

B. Loading MNIST-Dataset [5]

To load the MNIST dataset you need 2 classes. The classes DigitImage and LoadDigitImage. The DigitImage class declares the respective width, height, label as well as the individual pixel values of a single digit. In addition, the associated variables are initialized in the constructor so that they can be used in the LoadDigitImage class. It also ensures that the image width and height are constant (28x28).

```
class DigitImage
{
    public int width; // 28
    public int height; // 28
    public byte[][] pixels; // 0(white) - 255(black)
    public byte label; // '0' - '9'


    public DigitImage()
    {
    }

    public DigitImage(int width, int height,
      byte[][] pixels, byte label)
    {
        this.width = width;
        this.height = height;
        this.pixels = new byte[height][];
        for (int i = 0; i < this.pixels.Length; ++i)
            this.pixels[i] = new byte[width];
        for (int i = 0; i < height; ++i)
            for (int j = 0; j < width; ++j)
                this.pixels[i][j] = pixels[i][j];
        this.label = label;
    }
}
```

Figure 4: Snippet from the class Digit Image

The LoadDigitImage class consists of several methods.
The method LoadData has as input the pixel file and the label file of the MNIST dataset.
It opens and reads both files simultaneously. At the beginning, the method creates a local 28x28 matrix into which the values from the pixel file are read using the .NET BinaryReader class. It is important that the values are converted from the Big-Endian format to the Little-Endian format, otherwise erroneous digits will occur. For this the For this the method ReverseBytes is used, which has an integer as input.

```
public static int ReverseBytes(int v)
{
    byte[] intAsBytes = BitConverter.GetBytes(v);
    Array.Reverse(intAsBytes);
    return BitConverter.ToInt32(intAsBytes, 0);
}
```

Figure 5: Snippet of the method ReverseBytes

After reading in and converting the values, the data is reduced to the required format using the DigitImage class and written to an array of the required size.

C. Implementation of the CenterAlgorithm

The class CenterAlgorithm has the IPipelineModule of the LearningApi as interface. For this reason, its Run method has a double [][] as both input and output. The algorithm consists of several methods, the most important of which are explained below. The variable data of type double [][] is used as input. First, the data is binarized. For this purpose, the GetAverage method is used to calculate an average of all values of the double [][].

```
public double GetAverage(double[][] data)
{
    //building the average about all values
    for (int i = 0; i < data.GetLength(0); i++)
    {
        for (int j = 0; j < data.Length; j++)
        {
            average += data[i][j];

        }
    }
    average /= data.GetLength(0) * data.Length;
    return average;
}
```

Figure 6: Snippet of the method GetAverage

In the Binarization method, all values greater than the mean value are replaced with a 1 and all values less than the mean value are replaced with a 0.

```
public double[][] Binarization(double[][] data)
{
    double average = GetAverage(data);

    //divide each value by teh average
    for (int i = 0; i < data.GetLength(0); i++)
    {
        for (int j = 0; j < data.Length; j++)
        {
            if (data[i][j] > average)
            {
                data[i][j] = 1;
            }
            else { data[i][j] = 0; }
        }
    }
    return data;
```

Figure 7: Snippet of the method Binarization

After binarization, the ones are counted, and their positions stored in two arrays representing your X and Y coordinates. Then the arrays of the X and Y values are summed up and stored in an integer variable. This is done using the GetSumXValues and GetSumYValues methods. This is used to determine the center of the original double [][] in which the Mean is calculated. Therefore the summed X and Y values are divided by the previously counted number of ones (figure 9).

```
float xEnd = GetSumXValues(x, countOnes);
float yEnd = GetSumYValues(y, countOnes);

// Here I calculate the center of the origi

int xReallyEnd = (int)(xEnd / countOnes);
int yReallyEnd = (int)(yEnd / countOnes);
```

Figure 8: Snippet for Calculating the sum and mean

In the following snippet the value is calculated by which each single 1 must be shifted in order to be in the center. Therefore the center of the width and the height of the array is subtracted from the Mean calculated before.

```
// Calcute the shift of the ones in x and y direction
float diffMiddleNumberToMiddleMatrixX = xReallyEnd - widthHalf;
float diffMiddleNumberToMiddleMatrixY = yReallyEnd - heightHalf;
```

Figure 9: Snippet of the Calculation of the shift

At the end of the algorithm, the previously stored ones are increased or decreased by the previously calculated value in X and Y direction.

```
for (int i = 0; i < x.Length; i++)
{
    x[i] += diffMiddleNumberToMiddleMatrixX;
    y[i] += diffMiddleNumberToMiddleMatrixY;
    centeredData[(int)y[i]][(int)x[i]] = 1;
}
```

Figure 10: Shifting of the X- and Y-Values

The ones are stored with their new X and Y values in a newly initiated double [][] of the same size. The rest of the double [][] is filled with zeros.

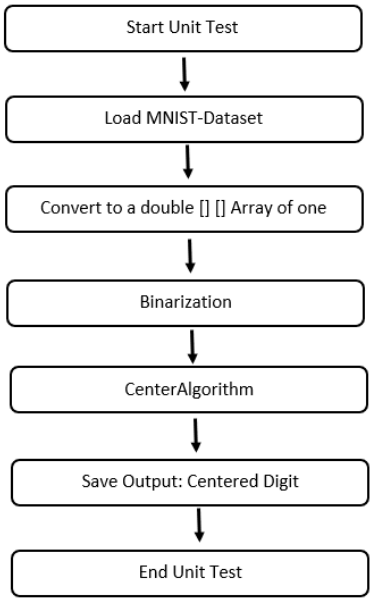
```
//Fill the Rest of the Array with Zeros
for (int i = 0; i < height; i++)
{
    for (int j = 0; j < width; j++)
    {
        if (centeredData[i][j] != 1
        )
            centeredData[i][j] = 0;
    }
}
```

Figure 11: Fill the Rest of the Array with 0

As return value the class CenterAlgorithm has again a double [][], so it is compatible with other projects from the LearningApi.

D. Unit-Test

This project contains a Unit-test in which the whole project is tested: loading the MNIST Digits, binarizing, centering and saving the centred image. In addition, there is another Unit-tests class with four additional Unit-tests that tests the individual methods of the CenterAlgorithm.



The Unit-test is structured as in Figure 19. After starting the Unit-test, a MNIST digit is loaded from the data set using the LoadData method of the LoadDigitImage class. The image is still in hexadecimal and looks like Figure 13.

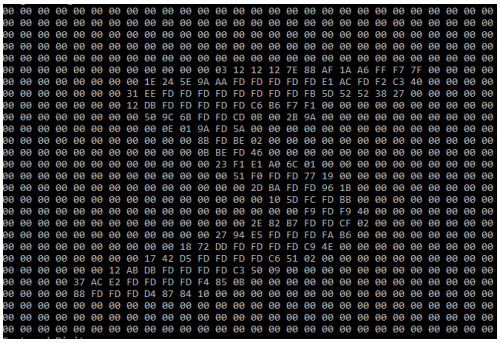


Figure 13: MNIST-Digit

It is then stored in a double [][] and passed to the CenterAlgorithm. The Storing to a double [][] is shown in Figure 15.

```
//Make Mnist pictures in Double Array
double[][] data = new double[images[digitNumber].pixels.Length][];
for (int i = 0; i < data.Length; i++)
{

    data[i] = new double[images[digitNumber].pixels.Length];
}
for (int i = 0; i < images[digitNumber].pixels.Length; i++)
{
    for (int j = 0; j < images[digitNumber].pixels.Length; j++)
    {
        data[i][j] = ((double)images[digitNumber].pixels[i][j]);
    }
}
```

Figure 14: Make MNIST-Digit in double[][]

In the CenterAlgorithm the double [][] is binarized with the method Binarization as described before.
The binarized image is now centered with the algorithm as described before. The result is stored in a text file using the Save File method of the Save class.

```
public void SaveFile(double[][] output)
{
    string path = Path.Combine(AppDomain.CurrentDomain.BaseDirectory, "CenteredImages\\result.txt");
    using (StreamWriter writer = new StreamWriter(path))
    {
        foreach (var value in output)
        {
            foreach (var saveValue in value)
            {
                writer.Write(saveValue);
            }
            writer.WriteLine();
        }
    }
```

Figure 15: Method SaveFile of the class Save

The saved Output in the text file looks like Figure 16.

```
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000001110111100000
00000000111111111111111100000
00000011111111111111111000000
00000001111111111100000000000
00000001111111001100000000000
00000000001110000000000000000
00000000001110000000000000000
00000000000111000000000000000
00000000000011110000000000000
00000000000001111000000000000
00000000000000111100000000000
00000000000000011110000000000
00000000000000001111100000000
00000000000000000111110000000
00000000000000000011111000000
00000000000000000001111100000
00000000000000000000111110000
00000000000000000000011111000
00000000000000000000001111100
00000000000000000000000111110
00000000000000000000000011111
00000000000000000000000001111
00000000000000000000000000111
00000000000000000000000000011
00000000000000000000000000001
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
00000000000000000000000000000
```

Figure 16: Output centered Digit

The other 4 Unit-tests of the CenterAlgorithm are described below.

```
[TestMethod]
public void TestCalculateAverage()
{

    double[][] data = new double[8][];
    double average = 0;
    for (int x = 0; x < 8; x++)
    {
        data[x] = new double[8];
        for (int y = 0; y < 8; y++)
        {
            data[x][y] = 1;
        }
    }

    CenterAlgorithm center = new CenterAlgorithm();
    average = center.GetAverage(data);
    Assert.AreEqual(1, average);
```

Figure 17: Unit-test CalculateAverage

The TestCalculateAverage Unit-test tests the GetAverage method.
For this purpose, an array is filled, and the result of the mean value calculation is compared with the previously correctly calculated value. Since all fields were filled with a 1, the mean value is also 1.

```
[TestMethod]

public void TestSumXYValues()
{
    float[] x = { 1, 2, 3, 4, 5, 6, 7 };
    float[] y = { 1, 2, 3, 4, 5, 6, 7 };
    int countOnes = 7;
    CenterAlgorithm center = new CenterAlgorithm();
    float xEnd = center.GetSumXValues(x, countOnes);
    float yEnd = center.GetSumYValues(y, countOnes);



    Assert.AreEqual(28, xEnd);
    Assert.AreEqual(28, yEnd);
}
```

Figure 18: Unit-test Calculate Sum of X- and Y-Values

The TestSumXYValues Unit-test tests the two methods GetSumXValues and GetSumYValues.
Here for an array for the X-values and an array for the Y-values is initiated and filled with numbers in the range 1-7.
The result of both summations is compared with the previously calculated value (28).

```
[TestMethod]
public void TestBinarization()
{
    double[][] data = new double[8][];
    for (int x = 0; x < 8; x++)
    {
        data[x] = new double[8];
        for (int y = 0; y < 8; y++)
        {
            data[x][y] = 1;
        }
    }

    CenterAlgorithm center = new CenterAlgorithm();
    double[][] binarizedData = center.Binarization(data);
    Assert.AreEqual(data, binarizedData);

}
```

Figure 19: Unit-test Binarization

This Unit-test tests the Binarization method. For this a test double [][] is created, which binarizes according to the method Binarization. The binarized double [][] is then compared again with the output double [][]. Since the output double [][] is only filled with ones, they must be identical.

```
[TestMethod]
public void TestInitiateOutputArray()
{
    int widthOriginArray = 8;
    int heightOriginArray = 8;
    CenterAlgorithm center = new CenterAlgorithm();
    double[][] centeredData = center.InitiateCenteredtArray(widthOriginArray, heightOriginArray
    int widthCenteredArray = centeredData[0].Length;
    int heightCenteredArray = centeredData.Length;
    Assert.AreEqual(widthOriginArray, widthCenteredArray);
    Assert.AreEqual(heightOriginArray, heightCenteredArray);
}
```

Figure 20: Unit-test Initiate outputarray

The Unit-test TestInitiateOutputArray tests the InitiateCenteredArray method. For this a certain array size is specified and passed to the method. Then the width and height of the created array is stored in variables and compared with the given array size.

IV. Discussion

The goal of the project is to implement an algorithm that centers a data set. In the course of the project an algorithm was developed which centers a data set in the form of a binarized double [][]. In addition, the MNIST dataset of handwritten digits was used to test the algorithm. The algorithm works reliably and can center all datasets in a binarized double array. A total of 5 Unit-tests were created to test the algorithm in all its methods. The testing of the main Unit-test was successful. A digit was loaded from the MNIST dataset, stored in a double [][], binarized and centered with the algorithm. The output is a digit that is exactly in the center of the dataset. In addition, four further Unit-tests were implemented to test the various calculations and initializations of the center algorithm. All Unit-tests were positive.
In summary, it can be said that the task was successfully fulfilled according to the specifications. A reliable algorithm was developed that can center datasets, especially digits.

V. References

[1]Dobric, D. *https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi.* Retrieved on 27. 12 2018 at https://github.com/UniversityOfAppliedSciencesFrankfurt/LearningApi

[2] Yann LeCun, Corinna Cortes, Christopher J.C. Burges (1998): http://yann.lecun.com/exdb/mnist/

[3] statisticshowto.datasciencecentral.com (2019): https://www.statisticshowto.datasciencecentral.com/probability-and-statistics/statistics-definitions/average/

[4] McCaffrey, James (2014): Arbeiten mit dem MNIST-Bilderkennungs-Datensatz, https://msdn.microsoft.com/de-de/magazine/dn745868.aspx

[5] Khan Academy (2013): https://courses.lumenlearning.com/introstats1/chapter/measures-of-the-center-of-the-data/