UMEÅ UNIVERSITET
Institutionen för datavetenskap
Laborationsrapport

October 23, 2015

Laboration 5
AI, HT-15

Neural Network

Ludwig Andersson, dv13lan@cs.umu.se
Tim Hedberg, dv13thg@cs.umu.se

Contents

Introduction

In this report we will go over the implementation of a machine learning system based on a Perceptron with a neural network as learning mechanism.

This report will go over the solution, the implementation some important algorithms that is vital to the solution. A system description with details for image processing, image rotation, image parsing and the learning abilities of the neural network and how it performs during different parameter values for the learning rate as well the number of training loops.

The reader will also be given a brief introduction to the Maven framework and will learn how to compile and execute the project.

Prerequisites

Before we can start compiling and running the assignment solution a few tools and programs is needed. The examples in the listings throughout the report will assume that you are running a UNIX environment and/or could be different on other platforms.

JRE and Java compiler

Since this is a Java project you will need the Java compiler, `javac` and a JRE(Java Runtime Environment). To see if you have java and javac installed on your machine you can run the following commands:

Listing 1: Javac and JRE

```
$ javac -version
javac 1.8.0_60

$ java -vesion
openjdk version "1.8.0_60"
OpenJDK Runtime Environment (build 1.8.0_60-b24)
OpenJDK 64-Bit Server VM (build 25.60-b23, mixed mode)
```

If you receieve an error saying that the command is not found then Java or Javac may not be installed on your system.

Maven

To be able to smoothly build the project and have automatic dependency handling of the projects source code the Maven framework is used. Maven can compile, test and package the solution easily. Later in this report you can find out how to build and run this solution.

To check if you have maven installed you can run this command:

Listing 2: Maven

```
$ mvn -v
Apache Maven 3.3.3
Maven home: /opt/maven
Java version: 1.8.0_60, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-8-openjdk/jre
Default locale: en_US, platform encoding: UTF-8
```

1 Assignment

Our assignment is to implement a perceptron with a neural network. The perceptron will guess the emotional states of image inputs (represented as a 20x20 2D array). The perceptron must be able to learn a set of images and will then be tested on a set of new images and it must score above 65 percent correct guesses.

The assignment could be divided into three parts

- Parse the images
- The Neural Network
- Output format

Another demand was to read the image training files in their specific format and output them in a specific format. This added some demands on the parser as the input test data looks like this (Example shortened for report):

Listing 3: Sample images

```
# - Happy, Sad, Mischievous or Mad - #
# Training data (300 images)
# Correct classifications can be found in training-facit.txt
# http://www8.cs.umu.se/kurser/5DV121/HT12/

Image1
0 3 3 3 3 0 1 0
4 1 9 0 0 0 0 0
0 7 0 0 0 6 1 2
0 8 31 5 0 0 0

Image2
0 3 3 5 3 0 1 0
4 1 9 6 0 4 0 0
0 7 0 0 0 6 1 2
0 8 31 5 0 12 0
```

The corresponding answer file is formated like this:

Listing 4: Answers example

```
# - Happy, Sad, Mischievous or Mad - #
# Correct answers for training data (training.txt)
# See http://www8.cs.umu.se/kurser/5DV121/HT12
Image1 2
Image2 1
Image3 4
Image4 4
```

The output from the Perceptron program will have the same formatted output as the answers file when launched in automatic mode.

2 Compile and run

2.1 Running Perceptron

The solution contains two modes of the execution of the Perceptron program. We have a Command line interface(CLI) which is used mainly for debugging of images. It allows you to load images and answers into memory and a feature to show the pixels in a GUI.

Listing 5: Launch in CLI Mode

```
$ java -jar Perceptron-1.0.jar
```

The other running mode is the automatic mode where you specify three arguments to the program, the training data, the answers to the training data and the test data which you don't supply the answers for.

Listing 6: Launch in Automatic mode

```
$ java -jar Perceptron-1.0.jar "traindata" "answers" "testdata"
```

2.2 Compiling

The solution can be compiled in the terminal by using Maven by using

Listing 7: Maven Compile

```
$ mvn compile
```

If you would like to package the solution into a runnable JAR file you can use this command

Listing 8: Maven Package

```
$ mvn package
```

Then the runnable .JAR file can be found in the target folder and be executed as described in the section above.

Maven ensures that all the dependencies for the project are solved automatically so we dont need to worry about manually handling the dependencies and libraries for the project.

If needed it is possible to clean the project in order to remove all the compiled classes and the runnable .JAR file. the command to clean a Maven project is:

Listing 9: Maven Clean

```
$ mvn clean
```

In the directory specified by the assignment specification there is a script called `compile.sh`. It will retrieve the latest version of the project from github, compile it using Maven and create a jar called `Faces.jar`. This install script will automaticly clean up the folders after generating the jar.

Listing 10: Compiler Script

```
$ sh compile.sh
```

3 System Description

The system has two different starting modes to choose from, one is a CLI (Command Line Interface) which was used under development for mostly debugging purposes, the other mode is the Automatic mode where no user interaction is needed.

For more information and implementation details please go ahead to view our `Javadoc` at `http://www8.cs.umu.se/~dv13lan/ai/`. It will give a brief overview on how packages and classes are sorted as well of documentation of methods and classes.

3.1 Imageparser

The `ImageParser` is responsible for loading the image files and the answers file into the program. The images will be read into an Array with the type of `FileImage` through the method `parseImages()`. This method takes a file name as parameter.

The answers are loaded into an hashtable with a `String` as the key and an `Integer` as value. This makes it important to have unique names for each image.

The `ImageParser` will filter out any lines that begins with a `'#'` character and will treat them as a comment in the file and should therefore be ignored by the parser. This is true for both the answers file and the image file.

3.2 Image Handler

The `ImageHandler` responsibility is to analyze if an image needs to be rotated. This is done by slicing the large 2D array image into four sub arrays of the image. In each sub array the `ImageHandler` will summarize all the pixels that are over a certain value to find out where the "eyes" are of the face. After that is done the image will be rotated so that the "eyes" will always be at the top of the array and the mouth at the south. This makes it easier for the neural network to output more correct guesses of the emotional class of the image.

This was however one of the hardest part to implement of the system since it was confusing and we had little experience in working and manipulating 2D array in this advanced manners. We had to look up some tips and tricks of rotating and mirroring of 2D arrays on the internet to be able to get a satisfying solution.

3.3 CLI

The CLI or Command Line Interface in this solution has been used for debugging and development purposes in the project. It has the functionality to display images in a GUI as well load images and answers with the `ImageParser`. It looks like a command prompt and can take some commands with arguments such as loading custom image files and showing images regarding to their index.

3.4 Autorunner

The Autorunner takes three arguments, a training image file, the answers to the training images and some test images. The `Perceptron` will then train X number of times with a learning rate and then output it guesses formatted as the assignment demanded.

The number of times it will train is set by a constant in the `AutoRunner` class and the learning rate of the neural network is also set by a constant in the `ANN` class.

Depending on how you set the two variables and the number of correct guesses can vary. By default the settings are optimized for 250 training images and then tested on 50 unknown images.

3.5 Neural Network

The neural network is implemented in the `ANN.java` class. It is responsibility is handling and training of the neural network. To construct a neural network you need to send in the training data and the answers as parameters to the constructor. The constructor will then handle the initiation of weights and shuffling of values.

The initiation of the weights values is done by randomizing a float value between 0 and 1 using the `Random` object provided by Java. This randomization is random and good enough for this assignment.

After the initiation of the neural network the user is free to train the network with a specified learning rate and with a and for a specified number of times. By adjusting these values you will get different behavior and the ability to learn.

To start to train the network the method `train( learningRate,loops )` is used, The learning rate and the number of training loops have huge impact on how the neural network will perform later when doing the classification test.

To run the classification test you need to call the `runTests(images)` method to try test the perceptron on a new set of images which you don't provide the answers for. The output will be as demanded by the specification and any extra information will be preceded by `#`.

The neural network consists of 200 nodes, one node for each pixel, each node will then have a weight that is initiated with a random float value between 0 and 1. These 200 nodes can generate one out of four outputs depending on the total weight between the nodes. Depending on the weight/activity generated in the network when received an image as input we can then return either sad, happy, mischievous or mad as an emotional classification.

4 Algorithms

4.1 Learning algorithm

The learning algorithm for the neural network is a central and important part of the solution of this assignment. As mentioned earlier the assignment needed to score at least 65 percent correct on a set of 100 untrained images. The neural network uses a set of weights which will tell you how much activation the nerual network will output given a image. And by analyzing this output you can make a more accurate guess on what type of classification the image has.

The trick to make the perceptron as accurate as possible is to provide it with as similar data as possible, as we cannot control the content of the images we could however adjust the rotation of them, by having all the images rotated in the same orientation the neural network will be more accurate.

Another important aspect of the learning algorithm is which learning rate you have, if set to low the weights may not be able to adjust enough to get good output or set to high it will constantly over adjust the weights.

1. Shuffle the training data
2. while x > 0
   (a) For each training image
      i. Calculate the error
      ii. For each pixel and weight calculate a new weight
3. x = x - 1

The formula to calculate the weights:

Listing 11: Delta Calculation

```
delta = LEARNING_RATE * error * PV;
```

4.2 Image rotation

Image rotation algorithm.

1. Convert the image into four chunks, a 2x2 matrix.
2. Calculate the sum of each chunk.
3. Get the value of each side of the image by adding two chunks together. (North side, East side, South side, west side)
4. Get the chunk with the greatest value
5. check which side connected to it has the greatest value.
6. The image is rotated upright by rotating the highest value chunk to the beginning of the array.

4.3 Image reading

The image reading algorithm.

1. While not end of file
   (a) while the line starts with a integer read the line
      i. Split the text at white space. From the split you receive a row of the image.
      ii. add the row to a matrix
   (b) save the matrix (containing the image).

4.4 Image Preprocesing

The preprocessing image has two parts, the first part where we convert the image to black/white with no gray scale at all. The second part is where we find lonely pixels and replace them with a white one.

`Convert to black/white`:

1. For each pixel in the image
   (a) If pixel value < THRESHOLD; set it to 0
   (b) If pixel value > THRESHOLD; set it to 1

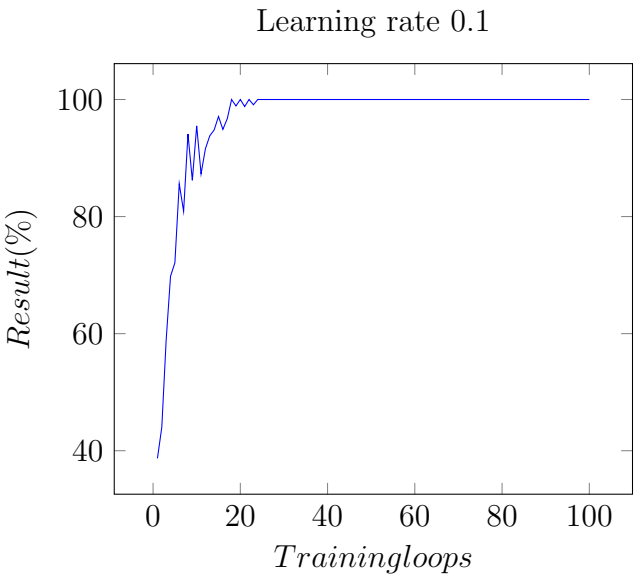
`Remove Jiitter`:

1. For each pixel in the image
   (a) If pixel has no neighbors that is black pixels
      i. turn current pixel value to 0

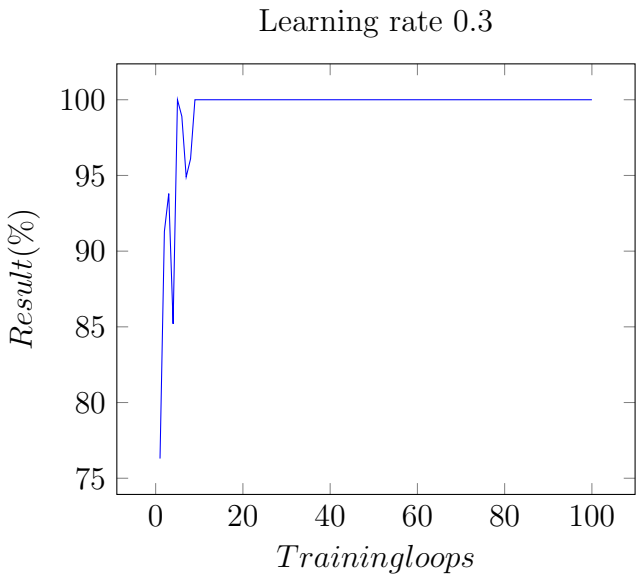
5 Performance

In the final stages of this project some performance testing of the neural network was done to make sure we had the right parameters for the learning rate and the number of training loops. A series of testing was done on the learning rates between 0.1 to 2.0 with the incremental rate of 0.2 between tests and with the loops between 1 to 100 with an incremental rate of 1. To get the optimal parameters for this we are looking for a graph where the fluctuation is minimal and we reach a stable 100 percent result as quickly as possible.

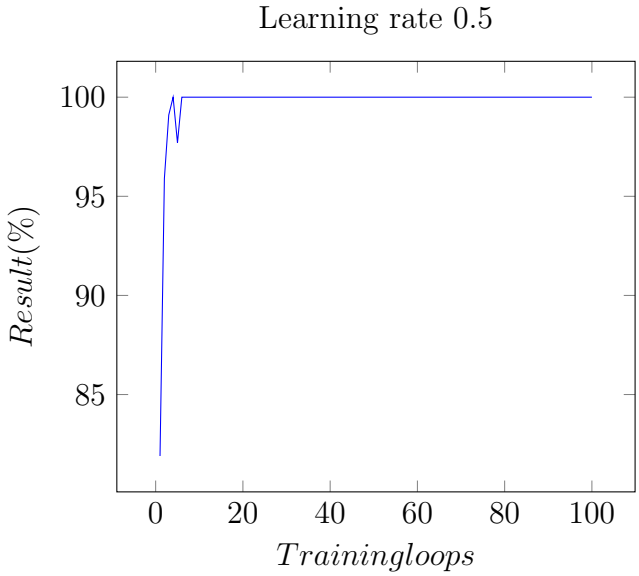
5.1 Performance Graphs



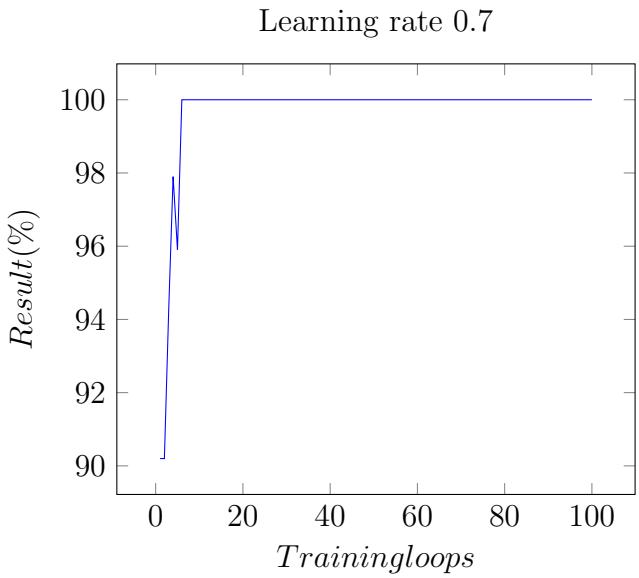
With a learning rate of 0.1 the performace reaches 100 already with around 20 loops of training, it runs stable at 100 percent after around 35 loops.



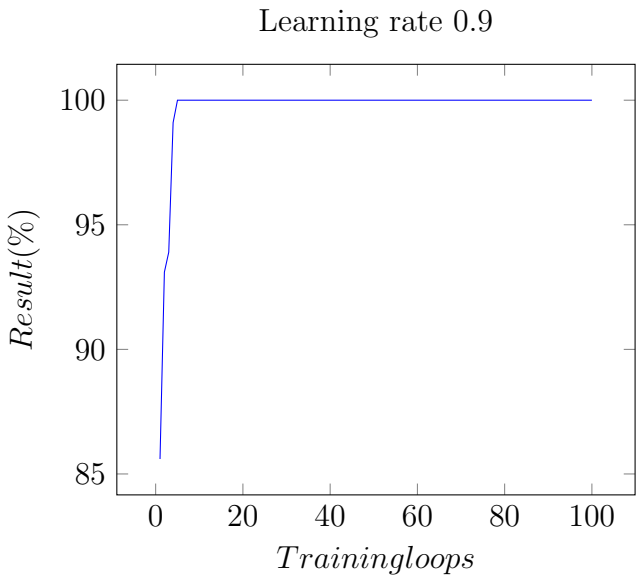
With 0.3 as learning rate it can be observed an similar behavior as the with the 0.1 learning rate but with a higher amount of fluctuation but fewer occurrences in the percentage of correct answers.



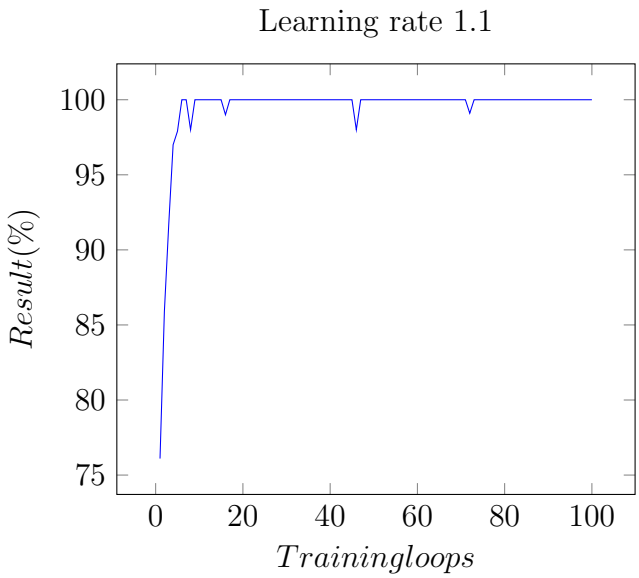
With the learning rate of 0.5 we get only a minimal amount of fluctuation and we reach a stable result of 100 percent quite quickly. This learning rate could work to get great results.



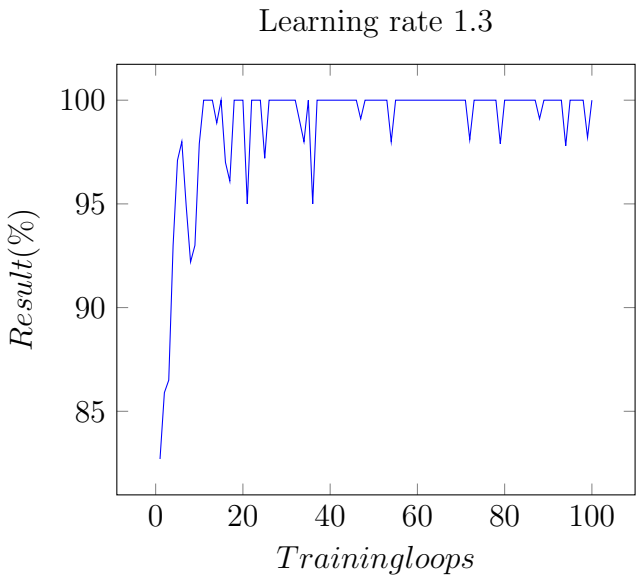
With a learning rate of 0.7 we get the same behavior as with the learning rate of 0.5, the difference here though is that we get is that the minor fluctuation is this case has more impact on the result percentage.



With a learning rate of 0.9 we get a very stable behavior of the perceptron and the neural network.

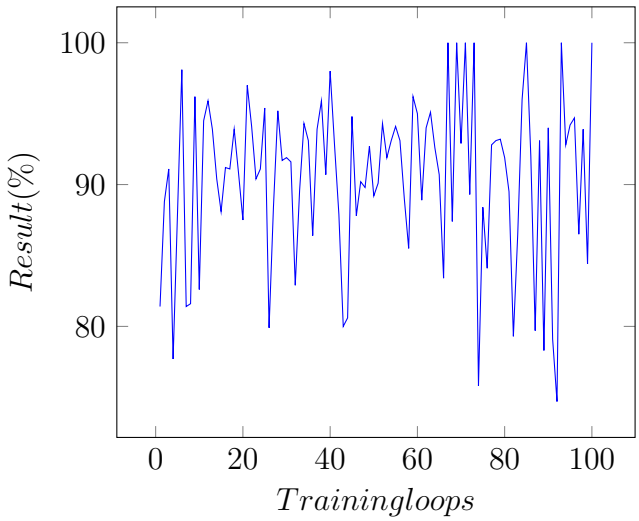


With a learning rate of 1.1 it starts we can observa some drop in the results along the number of loops. This can be because of a to high learning rate can lead to over compensation of the weights when learning.

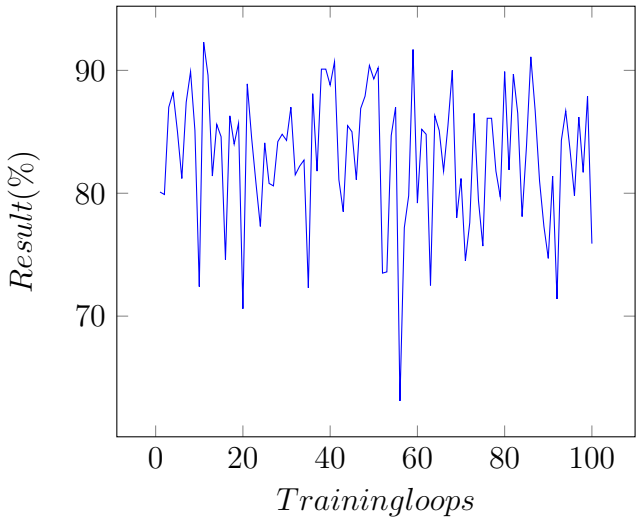


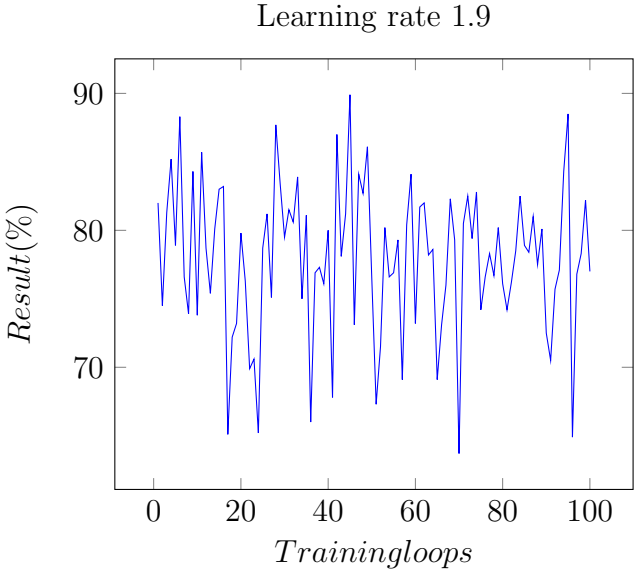
With the learning rate of 1.3 we can see that the fluctuation is much worse than before and this you can clearly notice that the training led to bad weights values in the learning process.

Learning rate 1.5



Learning rate 1.7





By analyzing these graphs we can see that a well balanced form of learning rate needs to be found to get a good learning ability within the neural network. However these tests are not 100 percent accurate due to the initiation of random weights in the neural network as this can affect the learning ability. The training images are shuffled and as well the test images which we get the result percentage from.

With a high learning rate we get irregular random behavior from the neural network which cannot be predicted as with a lower learning rate a larger amount of loops is needed to adjust the weights enough.

5.2 Image processing

We also found out that pre-processing the images did increase the performance results with as much as 10-20 percent. The pre-processing makes the image more alike and therefore its easier for the neural network to recognize the image and make correct guess. This together with making all images having the same orientation makes the neural network perform much better and generate a higher percentage in correct guesses.

6 Discussion

6.1 General feedback on the assignment

The assignment was interesting and at some points quite difficult. The implementation of the neural network was a bit hard to understand from the specification and it took some time to understand what you actually needed to do to make it function properly.

However it was a new interesting subject and has given some basic insight of how neural networks works and how they can be used to achieve machine learning abilities within programs. We believe that having knowledge about neural networks and machine learning and how they work and how to implement them gives you a great tool to be able to write interesting and powerful software in the future.

As it took some time to implement and other courses as well took time the error handling of this solution is not very good and could be extended further. We assume that all data that is given as input to the program will be correctly formatted and follows the guidelines and if not the behavior is undefined.

The parsing part/implementation where you read in the images and answers from a file could have been given, this would allow more time to be spent on the actual neural network and a more advanced report instead.

We think that we achieved good results and that the system works very well. The implementation was done in Java which we had some previous experience with.

The image pre-processing and rotation of the images makes a great performance boost to the performance of the neural network which makes it possible to score as well as it does.

```
package gui;

import file.FileImage;

import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;

/**
 * Will paint a image to a Jframe. Has been used to debug image processing
 * and rotation of images.
 *
 * @author dv13thg, dv13lan
 * @version 15 okt – 2015
 */
public class Gui {

    private final JFrame windowFrame;
    private final ArrayList<FileImage> imgMatris;
    private int imgIndex;

    /**
     * Constructs a new GUI.
     * @param imgMatris 2D image array to be used as database.
     * @param imgIndex index to fetch from the database.
     */
    public Gui(ArrayList<FileImage> imgMatris, int imgIndex) {
        this.imgMatris = imgMatris;
        this.imgIndex = imgIndex;
        windowFrame = new JFrame();
        windowFrame.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        windowFrame.setSize(640, 640);
        windowFrame.setFocusable(true);
        windowFrame.requestFocusInWindow();
        windowFrame.add(buildCanvas());
    }


    /**
     * builds a jtable with a OfferTableModel, rowsorter and
     * Tableselectionlistener and returns it as an scrollpane.
     *
     * @return JScrollPane the pane to be added to windowFrame.
     */
    public Canvas buildCanvas() {
        Canvas canvas = new Canvas() {
            //FIELDS
            public int WIDTH = 1024;
            public int HEIGHT = WIDTH / 16 * 9;

            public void paint(Graphics g) {
                g.setColor(Color.WHITE);
                g.fillRect(0, 0, WIDTH, HEIGHT);

                FileImage f = imgMatris.get(imgIndex);

                for (int x = 0; x < f.getImgMatrix().length; x++) {
                    for (int y = 0; y < f.getImgMatrix().length; y++) {
                        if (f.getImgMatrix()[x][y] >= 1) {
                            g.setColor(Color.black);
                        } else {
                            g.setColor(Color.WHITE);
```



```
                        }
                        g.fillRect(x * 10, y * 10, 10, 10);
                    }
                }
            }
        };
        canvas.setBackground(Color.WHITE);
        return canvas;
    }


    /**
     * sets the frame visibility to true
     */
    public void setVisible() {
        windowFrame.setVisible(true);
    }

}
```

```
package main;

import core.ANN;
import file.FileImage;
import file.ImageParser;
import gui.Gui;

import java.io.FileNotFoundException;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Scanner;

/**
 * @author dv13lan
 * @version 2015-10-13
 *          <p/>
 *          A basic commandline interface for the perception robot.
 */
public class CLI {

    public static final String RESOURCES_TRAINING_TXT = "resources/training.txt";
    public static final String RESOURCES_TRAINING_FACIT_TXT = "resources/training-facit.t
xt";

    private ImageParser parser;
    private HashMap<String, Integer> facitMap;
    private ArrayList<FileImage> fileImages;

    private Scanner scanner;

    /**
     * Constructs a new CLI. Setups the hashmap and the array lists.
     * It also gets an instance of the Imageparser as well as a new scanner from
 System.in
     */
    public CLI() {
        facitMap = new HashMap<>();
        fileImages = new ArrayList<>();

        scanner = new Scanner(System.in);
        parser = ImageParser.getInstance();
    }

    /**
     * Main shell loop, will read and execute commands. Split arguments at space
s.
     */
    public void run() {
        boolean quit = false;
        String userInput;

        System.out.println("Welcome to Percetron CLI (: ");
        System.out.println("Use help for available commands");

        while (!quit) {

            System.out.print("skynet -> ");
            userInput = scanner.nextLine();
            String[] argv = userInput.split(" ");

            switch (argv[0]) {
```



```
                case "help":

                    printHelp();

                    break;

                case "loadfacit":

                    if (argv.length == 2)
                        loadfacit(argv[1]);
                    else
                        loadfacit();

                    break;

                case "loadimages":

                    if (argv.length == 2)
                        loadimages(argv[1]);
                    else
                        loadimages();

                    break;

                case "status":
                    status();

                    break;

                case "showimg":
                    if (argv.length == 2) {
                        try {
                            showImage(Integer.parseInt(argv[1]));
                        } catch (NumberFormatException ex) {
                            System.err.println("Error: Second argument needs to be a number.
");
                        }
                    }
                    break;
                case "train":
                    startTraining(argv);
                    break;
                case "quit":
                    quit = true;
                    break;

                default:
                    System.err.println("Unknown command.");
                    break;
            }
        }
    }

    /**
     * Starts the trainer. Will reset the neural network
     *
     * @param argv An Array containing the arguments to the training.
     */
    private void startTraining(String[] argv) {
        if (argv.length == 3) {
            ANN trainer = new ANN(fileImages, facitMap);
            trainer.train(Double.parseDouble(argv[1]), Integer.parseInt(argv[2])
```

```
);
        }
    }

    /**
     * Shows an image from the training files.
     *
     * @param imgIndex Index of image to show.
     */
    private void showImage(int imgIndex) {
        Gui g = new Gui(fileImages, imgIndex);
        g.setVisible();

    }

    /**
     * Prints out the statistics of the program. Such as
     * how many noads that are loaded and how many answers that are loaded in.
     */
    private void status() {
        System.out.println("There is " + fileImages.size() + " nodes loaded.");
        System.out.println("There is " + facitMap.size() + " facit entries loaded");
    }

    /**
     * Loads the facit files into the facit map.
     */
    private void loadfacit() {

        try {
            facitMap = parser.parseFacit(RESOURCES_TRAINING_FACIT_TXT);
        } catch (FileNotFoundException ff) {
            System.err.println("Could not load file " + RESOURCES_TRAINING_FACIT_TXT);
        } catch (IOException e) {
            e.printStackTrace();
        }
        System.out.println("Loaded default faceit path, " + facitMap.size() + " entities loaded!
");
    }

    /**
     * Loads a facit file from a path.
     *
     * @param filePath A string representing the facit file path.
     */
    private void loadfacit(String filePath) {
        try {
            facitMap = parser.parseFacit(filePath);
        } catch (FileNotFoundException ff) {
            System.err.println("Could not load file " + filePath);
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("Loaded faceit path, " + facitMap.size() + " entities loaded!");
    }

    /**
     * Loads the imagefiles.
     */
    private void loadimages() {
        try {
```



```
            fileImages = parser.parseImage(RESOURCES_TRAINING_TXT);
            startImagePreProcessor();
        } catch (FileNotFoundException ff) {
            System.err.println("Could not load file " + RESOURCES_TRAINING_TXT);
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("Loaded default fileImages path, "
                + fileImages.size() + " entities loaded!");
    }

    /**
     * Overloaded method to use a custom filepath.
     *
     * @param filePath A path to the imagefile.
     */
    private void loadimages(String filePath) {
        try {
            fileImages = parser.parseImage(filePath);
            startImagePreProcessor();
        } catch (FileNotFoundException ff) {
            System.err.println("Could not load file " + filePath);
        } catch (IOException e) {
            e.printStackTrace();
        }

        System.out.println("Loaded fileImages path, " + fileImages.size()
                + " entities loaded!");
    }

    private void startImagePreProcessor() {
        for (FileImage image : fileImages) {
            image.preProcessImage();
        }
    }


    /**
     * Prints out the help menu.
     */
    private void printHelp() {
        System.out.println("Available Commands: ");
        System.out.println("\t help");
        System.out.println("\t loadimages");
        System.out.println("\t loadfacit");
        System.out.println("\t status");
    }
}
```

```
package main;

/**
 * @author dv13lan, dv13thg
 * @version 20 okt – 2015
 */
public class Perceptron {

    /**
     * Launches the program either in CLI mode or in automatic mode depending
     * on the number of arguments.
     *
     * @param args If 0 arguments is passed then the program will automatically
     *             launch in CLI mode. If 3 arguments are passed into the progra
m
     *             it will launch in automatic mode and will give
     */
    public static void main(String[] args) {

        // Launch in skynet mode.
        if (args.length == 0)
            new CLI().run();

            // Run in automatic mode
        else if (args.length == 3)
            new AutoRunner(args[0], args[1], args[2]).run();

            //invalid arguments.
        else {
            System.out.println("You need either 3 or 0 arguments" +
                    " to launch the program");

            System.exit(1);
        }

    }
}
```

```
package main;

import core.ANN;
import file.FileImage;
import file.ImageParser;

import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * Autoruns the program and will print out the results on standard output in
 * the format specified in the assignment.
 *
 * @author dv13lan
 * @version 20 okt - 2015
 */
public class AutoRunner {

    private final double LEARNING_RATE = 0.5;
    private final int TRAINING_LOOP = 14;

    private ArrayList<FileImage> testData;
    private HashMap<String, Integer> facitData;
    private ArrayList<FileImage> trainingData;

    /**
     * Constructs a new Autorunner object.
     *
     * @param trainingPath A string representing the file path to the training
     *                     file.
     * @param facitPath    A string representing the file path to the facit file
.
     * @param testFilePath A string representing the file path to the test
     *                     images that are not included in the training file.
     */
    public AutoRunner(String trainingPath, String facitPath, String testFilePath
) {
        try {
            trainingData = ImageParser.getInstance().parseImage(trainingPath);
            facitData = ImageParser.getInstance().parseFacit(facitPath);
            testData = ImageParser.getInstance().parseImage(testFilePath);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    /**
     * Starts running the automatic run of the NeuralNetwork.
     */
    public void run() {
        prepareData();
        ANN neuralNetwork = new ANN(trainingData, facitData);

        //Train train train
        neuralNetwork.train(LEARNING_RATE, TRAINING_LOOP);

        //Pray to god it works!
        neuralNetwork.runTest(testData);


    }
```



```
    /**
     * Will pre-process all images before use in the neural network.
     */
    private void prepareData() {
        //Pre process the training data
        for (FileImage img : trainingData)
            img.preProcessImage();

        //Pre process the test data.
        for (FileImage img : testData)
            img.preProcessImage();
    }
}
```

```
package core;

import file.FileImage;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Random;

/**
 * The ANN (A Neural Network) represents our neural network,
 * contains methods to train it, verify its performance and test it on new
 * images.
 *
 * An associated test for this class can be found and its called ANNTest.
 *
 * @author dv13lan, dv13thg
 * @version 20 okt - 2015
 */
public class ANN {

    private static final int IMG_SIZE = 20;

    private double[][] weights;
    private ArrayList<FileImage> imgList;
    private HashMap<String, Integer> facitFiles;

    /**
     * Constructs a new Trainer object set with a data set of image files and
     * the correct answers to them.
     *
     * @param imgList A list containing Facefile images.
     * @param facitFiles A list containing the correct answers.
     */
    public ANN(ArrayList<FileImage> imgList, HashMap<String, Integer> facitFiles
) {
        this.imgList = imgList;
        this.facitFiles = facitFiles;
        initANN();
    }

    /**
     * Creates and initiates a new ANN. Will allocate memory for the weights and
     * initiate them with random values. Will also shuffle the list of Faceimage
s.
     */
    private void initANN() {
        Collections.shuffle(imgList, new Random(System.nanoTime()));

        weights = new double[IMG_SIZE][IMG_SIZE];

        for (int x = 0; x < IMG_SIZE; x++) {
            for (int y = 0; y < IMG_SIZE; y++) {
                weights[x][y] = new Random().nextDouble();
            }
        }
    }

    /**
     * Trains the neural network with a set learning rate for a specific number
of
     * times.
```



```
     * @param noOfLoops The number of loops it will train.
     * @param learningRate The specified learning rate for the network.
     */
    public void train(double learningRate, int noOfLoops) {
        while (noOfLoops >= 0) {
            for (FileImage image : imgList) {
                double error;
                double[][] imageData = image.getImgMatrix();

                error = facitFiles.get(image.getName()) - activation(image);

                // iterate through every weight/pixel
                for (int j = 0; j < weights.length; j++) {
                    for (int k = 0; k < weights[0].length; k++) {
                        double delta = learningRate * error * imageData[j][k];
                        weights[j][k] += delta;
                    }
                }
            }
            noOfLoops--;
        }
    }

    /**
     * Given an image as parameter to this function, it will retrieve the 2d
     * image array from the FaceImage and sum the weights together times the
     * image data.
     *
     * If the image data at a pixel is 0 the sum of the weights will not increas
e.
     *
     * After the sums has been added together we will normalize the sum and
     * then run the Sigmoid function on it.
     *
     * Finally then we will return the correct integer representation of
     * SAD, HAPPY etc depending on the output from Sigmoid.
     * @param image An image to analyse.
     * @return the value
     */
    private int activation(FileImage image) {

        double weightSum = calculateWeights(image.getImgMatrix());

        if (weightSum < .25) {
            return 1;
        } else if (weightSum < .5) {
            return 2;
        } else if (weightSum < .75) {
            return 3;
        } else if (weightSum <= 1.0) {
            return 4;
        } else {
            return 0;
        }
    }

    /**
     * Calculates the weights
     * @param imageData 2D image array to calculate weights from.
     * @return An double representing the total sum of all the weights.
     */
    private double calculateWeights(double[][] imageData) {
```

```
        double weightSum = 0;

        for (int j = 0; j < imageData.length; j++) {
            for (int k = 0; k < imageData[0].length; k++) {
                weightSum += imageData[j][k] * weights[j][k];
            }
        }

        weightSum = weightSum / (imageData.length * imageData[0].length);
        weightSum = (weightSum * 6) - 3;

        // Sigmoid function
        weightSum = 1 / (1 + Math.exp(-weightSum));
        return weightSum;
    }

    /**
     * Tests the performance of the neural network.
     * @return The percentage of correct answers as a double.
     */
    public double testPerformance(int numberOfTests) {
        double correctAnswers = 0;
        Collections.shuffle(imgList,new Random(System.nanoTime()));
        for(int i = 0; i < numberOfTests;i++){
            int imgIndex = new Random().nextInt(imgList.size());

            if (activation(imgList.get(imgIndex)) ==
                    facitFiles.get(imgList.get(imgIndex).getName())) {
                correctAnswers++;
            }
        }
        return 100.0 * (correctAnswers / numberOfTests);
    }

    /**
     * Runs a classification test on a set of images.
     * @param images An array of images to perform the test on.
     */
    public void runTest(ArrayList<FileImage> images) {
        System.out.println("# Output: ");
        for (FileImage image : images) {
            System.out.format("%s %d\n", image.getName(), activation(image));
        }
    }

}
```

```
package file;

/**
 * A FileImage is a 2d array of integers associated with a name.
 * The FileImage contains a name with a 2D array representing the
 * image data.
 *
 * @author dv13thg, dv13lan
 * @version 20 okt – 2015
 */
public class FileImage {

    public static final int PIXEL_THRESHOLD = 8;
    private String name;

    private double[][] imgMatrix;

    /**
     * Creates a new FileImage and allocates a 20*20 2d array for
     * the pixels.
     */
    public FileImage() {
        imgMatrix = new double[20][20];
    }

    /**
     * Sets the name of the Image, this is so it can be compared to the
     * facit file later.
     * @param name A string representing the name for this FileImage.
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Sets a value into the 2d array of integers. It uses a threshold to
     * determine if the pixel is black enough to be counted as 1 or as 0.
     *
     * @param x X axis coordinate.
     * @param y Y axis coordinate.
     * @param value An integer representing a pixel value.
     */
    public void setImgMatrix(int x , int y, int value) {
       this.imgMatrix[x][y] = value;
    }

    /**
     * Returns the name associated with this FaceImage object.
     * @return A String representing the name.
     */
    public String getName() {
        return name;
    }

    /**
     * Returns the 2d array (Image) of this Faceimage.
     * @return The image represented in a 2D array of integers.
     */
    public double[][] getImgMatrix() {
        return imgMatrix;
    }
```



```
    public void setCurrentImage(double[][] imgMatrix){
        this.imgMatrix = imgMatrix;
    }

    /**
     * Removes alone dots in the image for better looks
     * and easier processing.
     */
    public void preProcessImage() {
        for (int x = 0; x < imgMatrix.length; x++) {
            for (int y = 0; y < imgMatrix[0].length; y++) {
                if(imgMatrix[x][y] > PIXEL_THRESHOLD)
                    imgMatrix[x][y] = 1;
                else
                    imgMatrix[x][y] = 0;
            }
        }

        for (int i = 0; i < imgMatrix.length; i++) {
            for (int j = 0; j < imgMatrix[0].length; j++) {
                if(imgMatrix[i][j] != 0) {
                    if (!adjNodes(i, j)) {
                        imgMatrix[i][j] = 0;
                    }
                }
            }
        }

    }

    /**
     * Find out adjNodes in 8 corners.
     *
     * @param x Coordinate in the X-axis
     * @param y Coordinate in the Y-axis
     *
     * @return True if a another activated pixel has been
     * found in the vicinity of the pixel.
     */
    private boolean adjNodes(int x, int y) {
        for (int i = x-1; i <= x+1; i++)
            for(int j = y-1; j <= y+1; j++ )
                if(i >= 0 && i < imgMatrix.length) // i is within boundaries
                    if( j >= 0 && j < imgMatrix[0].length)// j is within boundar
ies
                        if(i != x || j != y )
                            if(imgMatrix[i][j] == 1)
                                return true;



        return false;
    }
}
```

```
package file;


import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.HashMap;

/**
 * @author dv13lan, dv13thg
 * @version 2015-10-13
 *
 * This class handles the parsing of the FileImage and facit files.
 * It is implemented using the singleton design pattern since we only need
 * one instance of this class.
 */
public class ImageParser {

    // Access outside this object by this field.
    private static ImageParser instance = new ImageParser();

    /**
     * private empty constructor as standard for singleton classes in java.
     */
    private ImageParser() { }

    /**
     * Parses the imagefiles, will ignore # signs as they are seens as comments.
     * @param filePath Path to the file containing the faceit images.
     * @return An arraylist containing faceit files.
     * @throws IOException
     * @throws NumberFormatException
     */
    public ArrayList<FileImage> parseImage(String filePath) throws IOException,
            NumberFormatException {

        ArrayList<FileImage> imgArr = new ArrayList<>();
        FileImage FileImage = new FileImage();
        int lineNumber = 0;

        BufferedReader bufferedreader = new BufferedReader(new FileReader(filePa
th));
        String line;

        while((line = bufferedreader.readLine()) != null) {
            if (line.startsWith("#") || line.trim().length() == 0) {
                if(FileImage.getName() != null &&
                        FileImage.getImgMatrix().length == 20){

                    imgArr.add(FileImage);
                    FileImage = new FileImage();
                }
            } else {
                if (line.matches("^[0-9]+$")) {
                    String[] ArrNumbers = line.split(" ");
                    for (int i = 0; i < ArrNumbers.length; i++) {
                        FileImage.setImgMatrix(i,lineNumber,
                                Integer.parseInt(ArrNumbers[i]));
                    }
                    lineNumber++;
                } else {
```



```
                    FileImage.setName(line);
                    lineNumber = 0;

                }

            }
        }
        ImageHandler ih = new ImageHandler();
        for(FileImage image : imgArr){
            ih.RotateImageAnalyzer(image);
        }
        return imgArr;
    }


    /**
     * Parses a facit file and writes the imagename as key and integer value as
     * integer value to the hashmap.
     * @param filepath file path to the facit file.
     * @return An hashmap.
     */
    public HashMap<String, Integer> parseFacit(String filepath) throws IOExcepti
on {
        HashMap<String, Integer> facitMap = new HashMap<>();
        BufferedReader bufferedReader = new BufferedReader(new FileReader(filepa
th));

        String line;

        while ((line = bufferedReader.readLine()) != null) {
            if(!line.startsWith("#")) {
                String[] tokens = line.split(" ");

                if(tokens.length == 2) {
                    facitMap.put(tokens[0], Integer.parseInt(tokens[1]));
                }
            }
        }
        return facitMap;
    }

    /**
     * Used by other classes and objects to get an instance of this parser.
     * @return An imageparser.
     */
    public static ImageParser getInstance() {
        return instance;
    }
}
```

```
package file;

/**
 * @author dv13lan, dv13thg
 * @version 2015-10-22
 *
 * This class handles the rotation of images.
 */
public class ImageHandler {

    /**
     * analyzes the image by converting the image into
     * four smaller ones, sums each image, calls rotate.
     * @param image the image to be rotated
     */
    public void RotateImageAnalyzer(FileImage image) {
        double[][] newImg = image.getImgMatrix();

        int noOfRotations = analyzeRotation(newImg);

        if(noOfRotations == -1) {
            newImg = mirrorX(newImg);
        } else if(noOfRotations == -2) {
            newImg = mirrorY(newImg);
        } else {
            for(int i = 0; i < noOfRotations; i++) {
                newImg = rotateImage(newImg);
            }
        }

        image.setCurrentImage(newImg);
    }

    /**
     * Splices the 2d array into 4 sub arrays, sums each sub array up and
     * returns the number of rotations needed to get a correctly rotated image.
     *
     * @param newImg 2D array to be sliced
     *
     * @return An integer representing the number of 90 degree
     * rotations.
     */
    private int analyzeRotation(double[][] newImg) {

        double[][] northWest = split(newImg,0,0,10,10); // North west
        double[][] southWest = split(newImg,0,10,10,20); // Southwest
        double[][] northEast = split(newImg,10,0,20,10); // North East
        double[][] southEast = split(newImg,10,10,20,20); // South East

        int sumNW = matrixSum(northWest);
        int sumNE = matrixSum(southWest);
        int sumSE = matrixSum(northEast);
        int sumSW = matrixSum(southEast);

        switch (rotationOffset(sumNW,sumNE,sumSE,sumSW)) {
            case -1:
                /** if two sides are the same */
                if(sumNE == sumNW) {
                    return 0;
                } else if(sumNE == sumSE) {
                    return 1;
                } else if(sumSE == sumSW) {
```



```
                    return 2;
                } else if(sumSE == sumNW) {
                    return 3;
                }
            case 0:
                /** Should not rotate */
                return 0;
            case 1:
                /** if north half has the most value, it's already upright */
                if(sumNE + sumNW > sumNE + sumSE){
                    return -1;
                } else {
                    /** east side has greatest value */
                    /** rotate 240 degrees */
                    return 3;
                }
            case 2:
                if(sumNW + sumSW > sumSE + sumSW){
                    /** West is greatest */
                    return -2;

                } else {
                    /** South has greatest value */
                    return 1;
                }
            case 3:
                if(sumNE + sumSE > sumSE + sumSW){
                    /** east half is greatest */
                    return 2;
                } else {
                    /** south has greatest value */
                    return 3;
                }
        }

        return 0;
    }

    /**
     * Mirrors the image horizontally.
     * @param matrix 2D array to mirror.
     *
     * @return the new mirrored 2d image.
     */
    private double[][] mirrorX(double[][] matrix) {
        double [][] out = new double[matrix.length][matrix[0].length];
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                out[i][matrix.length - j - 1] = matrix[i][j];
            }
        }
        return out;
    }

    /**
     * Mirrors the image vertically (Y axis).
     * @param matrix 2D array to mirror.
     * @return the new mirrored 2d image.
     */
    private double[][] mirrorY(double[][] matrix) {
        double [][] out = new double[matrix.length][matrix[0].length];
        for (int i = 0; i < matrix.length; i++) {
```

```
            System.arraycopy(matrix[i], 0, out[matrix.length - i - 1], 0,
                    matrix[0].length);
        }
        return out;
    }

    /**
     * Calculates the rotation offset from the 4 subarrays
     * @param sumNW 2D sub array of northwest corner.
     * @param sumNE 2D sub array of northeast corner.
     * @param sumSE 2D sub array of southeast corner.
     * @param sumSW 2D sub array of southwest corner.
     *
     * @return An integer representing the sub array of the
     * corner with most active pixels.
     */
    private int rotationOffset(int sumNW, int sumNE, int sumSE, int sumSW) {
        int[] collection = {sumNW,sumNE,sumSW,sumSE};

        int max = 0;
        int index = 0;
        for (int i = 0; i < collection.length; i++) {
            if (collection[i] >= max) {
                if(collection[i] == max){
                    return -1;
                }
                max = collection[i];
                index = i;

            }
        }
        return index;
    }

    /**
     * Sums all the values in an array and returns an int representing the
     * number of active pixels in that array.
     *
     * @param array a 2D array image.
     * @return An integer representing the number of active pixels.
     */
    private int matrixSum(double[][] array){
        int sum = 0;
        for (int x = 0; x < array.length; x++) {
            for (int y = 0; y < array[0].length; y++) {
                if(array[x][y] > 26) {
                    sum += 1;
                }
            }
        }
        return sum;
    }

    /**
     * Slices an array given a set of train and end coordinates.
     * @param image Array to be sliced.
     * @param startX train x value to begin slicing from.
     * @param startY Starting y value to begin slicing from.
     * @param endX End x coordinate to slice to.
     * @param endY End y coordinate to slice to.
     *
     * @return A sub array containing a part of the original array.
```



```
     */
    private double[][] split(double[][] image, int startX, int startY, int endX,
int endY) {
        double[][] subArray = new double[startX+endX][startY+endY];

        int xCounter = 0;
        int yCounter = 0;

        for (int i = startX; i < endX; i++) {
            for (int j = startY; j < endY; j++) {
                subArray[xCounter][yCounter] = image[i][j];
                yCounter++;
            }
            yCounter = 0;
            xCounter++;
        }

        return subArray;
    }

    /**
     * Rotates the image 90 degrees.
     * @param image Image to be rotated.
     * @return A new rotated 2D array.
     */
    private double[][] rotateImage(double[][] image) {
        double[][] ret = new double[20][20];

        for (int i = 0; i < image.length; ++i) {
            for (int j = 0; j < image[0].length; ++j) {
                ret[i][j] = image[image.length - j - 1][i];
            }
        }
        return ret;
    }

}
```

```
package test.test;

import file.FileImage;
import file.ImageHandler;
import file.ImageParser;
import main.CLI;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

public class ImageHandlerTest {

    private ImageHandler ih;
    private ImageParser ip;
    private FileImage fileImage;
    private double[][] image;


    @Before
    public void setUp() throws Exception {
        ip = ImageParser.getInstance();
        ih = new ImageHandler();
        fileImage = ip.parseImage(CLI.RESOURCES_TRAINING_TXT).get(8);
        image = fileImage.getImgMatrix();
    }

    @After
    public void tearDown() throws Exception {
        image = null;
    }

    @Test
    public void testRotateImageAnalyzer() throws Exception {
        printImage();

        ih.RotateImageAnalyzer(fileImage);
        image = fileImage.getImgMatrix();
        printImage();
    }

    private void printImage(){
        for(int x = 0; x < image.length;x++) {
            for (int y = 0; y < image[0].length; y++)
                System.out.printf("%3d ", (int)image[x][y]);
            System.out.println();
        }
        System.out.printf("\n");
    }
}
```

```
package test.test;

import file.ImageParser;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import java.util.HashMap;

import static org.junit.Assert.assertEquals;

/**
 * Tests the FileImage parser
 * @author dv13lan
 */
public class ImageParserTest {


    private ImageParser parser;

    @Before
    public void setUp() throws Exception {
        parser = ImageParser.getInstance();
    }

    @After
    public void tearDown() throws Exception {
        parser = null;
    }

    @Test
    public void testParseFacit() throws Exception {
        HashMap<String, Integer> map =
                parser.parseFacit("resources/training-facit.txt");
        assertEquals(map.size(), 300);
    }
}
```

```
package test.test;

import core.ANN;
import file.FileImage;
import file.ImageParser;
import main.CLI;
import org.junit.Before;
import org.junit.Test;

import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;

import static org.junit.Assert.assertTrue;

/**
 * Basic test class for the neural network.
 *
 * @author dv13lan
 * @version 20 okt - 2015
 */
public class ANNTest {

    //Training loops
    private static final int NO_OF_LOOPS = 1;
    private final double LEARNING_RATE = 0.5;
    private static final double PASS_PERCENTAGE = 0.5;

    private ANN neuralNetwork;
    private ArrayList<FileImage> images;

    /**
     * Setups the tests. It will read the default training file and
     * the default facit file from the CLI constants.
     *
     * @throws Exception
     */
    @Before
    public void setUp() throws Exception {
        ImageParser parser = ImageParser.getInstance();
        HashMap<String, Integer> facit =
                parser.parseFacit(CLI.RESOURCES_TRAINING_FACIT_TXT);

        images = parser.parseImage(CLI.RESOURCES_TRAINING_TXT);
        ArrayList<FileImage> clone = new ArrayList<>();
        Collections.shuffle(images);

        //Pre process the training data
        for (FileImage img : images)
            img.preProcessImage();

        for (int i = 0; i < 100; i++)
            clone.add(images.get(i));

        neuralNetwork = new ANN(clone, facit);
    }

    /**
     * Runs the performance test, the test will accept all and inclusive
     * the PASS_PERCENTAGE constant.
     *
     * @throws Exception
```



```
     */
    @Test
    public void testTestPerformance() throws Exception {
        neuralNetwork.train(LEARNING_RATE, NO_OF_LOOPS);

        double result = neuralNetwork.testPerformance(100);

        assertTrue(result >= PASS_PERCENTAGE);
    }

    /**
     * Runs the test for classification for the images.
     *
     * @throws Exception
     */
    @Test
    public void testClassificationTest() throws Exception {
        neuralNetwork.train(LEARNING_RATE, NO_OF_LOOPS);
        neuralNetwork.runTest(images);
    }


    /**
     * Automatic training value evaluation.
     */
    @Test
    public void testTrainingValue() {
        for (double learningRate = 0.1; learningRate < 2; learningRate += 0.2) {
            System.out.println("learnin RAte : " + learningRate);
            for (int loops = 1; loops < 101; loops += 1) {
                neuralNetwork.train(learningRate, loops);

                System.out.printf("%d %.1f\n", loops,
                        neuralNetwork.testPerformance(100));
                try {
                    setUp();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        }


    }
}
```