```
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);
    }

}
```

```java
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]) {
```

```java
                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();
    }
}
```

```java
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.
```

```java
     * @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;
    }
}
```

```java
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 {
```

```java
                    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;
    }
}
```

```java
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) {
```

```java
                    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.
```

```java
     */
    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;
    }

}
```

```java
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();
                }
            }
        }


    }

}
```