

Hochschule für Technik und Wirtschaft (FH)

Fachbereich Informatik/Mathematik

Dokumentation zum Beleg

# **Zeichenerkennung unter Verwendung einer Self Organizing Map**

im Fach Neuroinformationsverarbeitung I

Bearbeitet von:  
Maik Zachacker  
Matr.Nr: 16633  
e-Mail: [s51317@informatik.htw-dresden.de](mailto:s51317@informatik.htw-dresden.de)

Betreuender Professor:  
Prof. Dr. rer. nat. habil. H. Iwe

Dresden, 24. Februar 2006

# Inhaltsverzeichnis

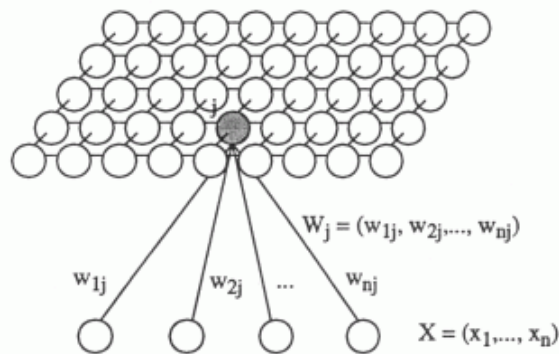
1. Einleitung.....	3
1.1 Das SOM (Kohonen Karte).....	3
1.2 Der Lernprozess.....	4
2. Systemvoraussetzungen.....	4
3. Installation.....	5
3.1 Installation von Java 2 unter Windows.....	5
3.2 Installation von Java 2 unter Linux.....	5
3.3 Installieren des Programmes SOM simulator.....	5
3.4 Installation von Java Advanced Imaging.....	6
4. Einschränkungen.....	6
5. Das Programm SOM simulator.....	6
6. Die Musterdatei.....	7
7. Das PlugIn NumberOCR.....	9
8. Die Bedienung.....	10
9. Die Bedienung des PlugIns NumberOCR.....	12
9.1 Bild öffnen.....	12
9.2 Bild invertieren.....	13
9.3 Ziffern trennen.....	13
9.4 Anzeigebereich bereinigen.....	14
9.5 Bild wechseln.....	15
9.6 Ziffern herunterrechnen.....	15
9.7 Erkennen der Nummer.....	16
10. Fazit.....	17
11. Änderungen am SOM simulator von Markus Schubert.....	17

## 1. Einleitung

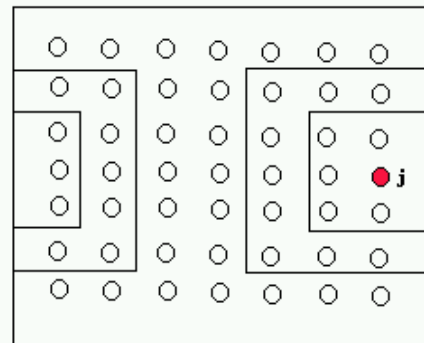
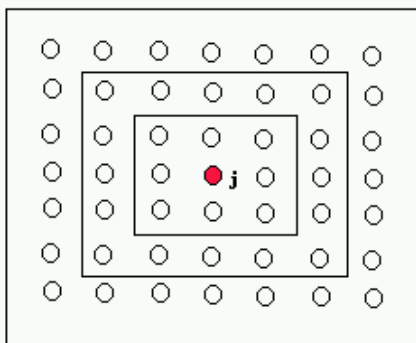
Aufgabe dieser Belegarbeit war es auf Basis des SOM simulator-Frameworks, von Markus Schubert, die Erkennung von Nummern mittels einer self organizing map zu realisieren. Das Programm NumberOCR wurde daher als PlugIn für diesen konzipiert und entsprechend umgesetzt. Das PlugIn fügt sich nahtlos in den SOM simulator ein und nutzt die damit erstellte Kohonen-Karte.

### 1.1 Das SOM (Kohonen Karte)

Kohonen-Karten (benannt nach dem Erfinder Teuvo Kohonen) oder auch bekannt unter dem Namen self organizing maps (SOM) sind eine Art von neuronalem Netz, welches Nachbarschaftsbeziehungen im Eingaberaum durch Nachbarschaften in den Neuronenbeziehungen ausdrückt. Das heißt, werden ähnliche Eingaben an das Netz angelegt, so reagieren auch benachbarte Neuronen auf diese Eingabe. Die self organizing map besitzt 2 Schichten. Eine Eingabe-Schicht und eine nach dem Entwickler benannte Kohonen-Schicht. Self organizing map's sind in der Lage mehrdimensionale Eingabemuster auf eine zweidimensionale Struktur abzubilden, welche sich durch Wettbewerbslernen (competitive learning) in einem unüberwachten Lernprozess von selbst ausbildet. Anhand folgender Skizze kann man sich eine Vorstellung eines SOM's machen.



Im Projekt wird wie im Bild zu erkennen eine zweidimensionale Kohonen-Karte verwendet. Jedes der Inputneuronen ist mit allen Neuronen der Karte verbunden, jedoch sind die Inputneuronen voneinander unabhängig. Die Neuronen in der Kohonen-Karte besitzen eine Nachbarschaftsbeziehung, die wichtig ist um ein auseinanderfallen der Karte zu ermöglichen. Die Karte kann dabei geschlossen oder geöffnet sein. Ist die Karte geschlossen, so haben Neuronen am Rand der Karte weniger Nachbarn als die in der Mitte und es erfolgt kein Lernen über die Ränder hinaus, siehe linkes Bild. Bei einer geöffneten Karte haben dagegen alle Neuronen die gleiche Anzahl an Nachbarn und es wird über den Rand der Karte hinaus gelernt, siehe rechtes Bild.



## 1.2 Der Lernprozess

Als erstes wird für jedes Zeichen, das später einmal erkannt werden soll, ein Muster benötigt. Im Beleg werden diese Muster als XML-Datei geliefert, welche für jedes Zeichen eine Matrix mit 5x7 Pixeln beinhaltet, aus welchen die Merkmale extrahiert werden. Prinzipiell überprüft das SOM ob ein bestimmtes Pixel zum Symbol oder zum "weißen" Hintergrund gehört. Dieses wird im Featurevektor entsprechend codiert eingetragen. Der Inputvektor hat folgenden Aufbau:

$$(\vec{x}^p) = (x_0^p, x_1^p, \dots, x_n^p)^T$$

Die Neuronen in der Kohonen-Karte werden nun mit den angelegten Mustern verglichen. In der Lernphase erfolgt dabei eine Veränderung der Gewichte der Neuronen der Kohonen Karte. Bei der späteren Erkennung erfolgt nur noch ein Vergleich. Damit überhaupt zulässige Vektoroperationen möglich sind ist es zwingend notwendig, dass die Neuronen der Karte mindestens genauso viele Elemente haben wie Inputneuronen:

$$(\vec{w}_j) = (w_{1j}, w_{2j}, \dots, w_{nj})^T$$

Liegen die Eingabevektoren vor, kann das Netz angelernet werden. Im Allgemeinen wird ein SOM mit zufälligen Werten initialisiert. Auch wenn das Netz bereits angelernet ist, können neue Eingabemuster angelernet werden.

Beim Lernen werden nun alle Inputneuronen der Reihe nach immer wieder mit allen Kartenneuronen verglichen. Es wird ein Gewinner in der Kohonen-Karte für das angelegte Muster ermittelt. Das Neuron mit dem kleinsten Abstand zwischen den beiden Vektoren ist das so genannte Gewinnerneuron.

## 2. Systemvoraussetzungen

- Zur Programmausführung wird ein Java kompatibles Betriebssystem, wie beispielsweise Linux oder Windows, vorausgesetzt.
- Des Weiteren muss eine Java JRE von Sun im System installiert sein, empfohlen wird die Version 1.5.0 oder höher.
- Es wird zwingend die Installation von Java Advanced Imaging 1.1.2 benötigt, da das PlugIn NumberOCR auf dessen Funktionen zugreift.

## 3. Installation

### 3.1 Installation von Java 2 unter Windows

Laden Sie die Datei „jre-1\_5\_0\_06-windows-i586-p.exe“ von „<http://www.java.com/de/download/manual.jsp>“ herunter und führen Sie die Datei aus.

### 3.2 Installation von Java 2 unter Linux

3.2.1 Laden Sie die Datei „jre-1\_5\_0\_06-linux-i586-rpm.exe“ von „<http://www.java.com/de/download/manual.jsp>“ herunter.

3.2.2 Machen Sie die Datei ausführbar mit:  
`chmod a+x jre-1_5_0_06-linux-i586-rpm.bin`

3.2.3 Starten Sie den Installationsvorgang und bestätigen Sie den Lizenzvertrag mit „Ja“:  
`./jre-1_5_0_06-linux-i586-rpm.bin`  
Die Installationsdatei erstellt im aktuellen Verzeichnis die Datei `jre-1_5_0-linux-i586.rpm`.

3.2.4 Führen Sie den RPM-Befehl am Terminal aus, um die Pakete zu installieren:  
`rpm -iv jre-1_5_0_06-linux-i586.rpm`  
JRE wird im aktuellen Verzeichnis im Unterverzeichnis `jre1.5.0_06` installiert.

3.2.5 Exportieren Sie das Verzeichnis `./jre1.5.0_06` als `JAVA_HOME` und `JAVA_ROOT`:  
`export JAVA_HOME="IHR VERZEICHNIS"/jre1.5.0_06/`  
`export JAVA_ROOT="IHR VERZEICHNIS"/jre1.5.0_06/`  
Exportieren Sie das Verzeichnis `./jre1.5.0_06/bin` als `JAVA_BINDIR`:  
`export JAVA_BINDIR="IHR VERZEICHNIS"/jre1.5.0_06/bin/`  
Fügen Sie die Exports der Pfadvariablen hinzu:  
`export PATH=$JAVA_ROOT:$JAVA_BINDIR:$PATH`  
Tragen Sie diese Zeilen am besten in die `.profile` oder `.bash_rc` ein, damit JAVA auch nach einem Neustart sofort verfügbar ist.

### 3.3 Installieren des Programmes SOM simulator

Das Programm SOM simulator liegt als ausführbare Java-Anwendung vor:

- `som.jar`

Zusätzlich werden folgende Dateien benötigt:

- `Ziffern.xml`
- `jai_codec.jar`
- `jai_core.jar`

Die Musterbilder liegen im Verzeichnis:

- `Pattern`

Eine Installation ist nicht erforderlich, es reicht die Dateien aus dem Archiv zu entpacken.

### **3.4 Installation von Java Advanced Imaging**

Kopieren Sie die Dateien „jai\_codec.jar“ und „jai\_core.jar“ in das Unterverzeichnis „jdk1.5.0\_04\lib\ext“ Ihres Java-Installationsverzeichnis (C:\Programme\Java\jdk1.5.0\_06\lib\ext\ oder JAVA\_ROOT/lib/ext/).

### **4. Einschränkungen**

Es werden Eingabebilder der Dimension 300x100 Pixel benötigt, die ausschließlich aus Hintergrund und Text bestehen.

- Das SOM erkennt kursive Ziffern nur unzureichend.
- Die Zeichentrennung muss manuell durchgeführt werden, da eine automatische Trennung durch Zeichenüberlappung nicht umsetzbar war.

### **5. Das Programm SOM simulator**

Der SOM simulator wird mittels des Kommandozeilenbefehles „java -jar som.jar“ im von Ihnen gewählten Verzeichnis oder per Doppelklick auf die Datei „som.jar“ im Explorer gestartet.

Nach dem Starten des Programms finden Sie die Oberfläche des SOM simulators vor, wo nur eine leere Arbeitsfläche und eine Menüleiste zu sehen ist.

Der SOM Simulator bietet die Möglichkeit ein ein- oder zweidimensionales SOM mit einer beliebigen Anzahl von Inputvektoren zu erzeugen. Außerdem kann das SOM mit verschiedenen Lernfunktionen trainiert werden und bietet mehrere Optionen das Lernverhalten zu beeinflussen. Die zum lernen notwendigen Muster müssen als PatternFile vorliegen, welche in XML verfasst sind und der beiliegenden XML Schema Definition genügen müssen .

Als Plugins werden die Darstellung einer Topologischen Karte des SOM, eine grafische Anzeige für das Handlungsreisenden Problem, eine zweidimensionale Karte und eine Komponentenkarte angeboten. Im Rahmen dieses Beleges wurde das Plugin NumberOCR hinzugefügt. Dieses dient der Erkennung von Ziffern, die in verschiedenen Schriftarten als 6-stellige Zahl im JPG-Format vorliegen. Die Zeichentrennung wird dabei manuell durch den Nutzer durchgeführt.

## 6. Die Musterdatei

Die Musterdatei, auch PatternFile genannt, beinhaltet die Informationen über ein PatternSet. Das PatternSet besteht aus einem Namen, der Anzahl der im Set enthaltenen Muster, der Featurevektorgroße, der Position der einzelnen Featurevektoren in einer zweidimensionalen Karte und natürlich den Mustern (Pattern). Die Musterdatei ist entsprechend einer XML-Notation aufgebaut, welche in der Datei PatternSet.xsd beschrieben ist.

### Im PatternFile enthaltene Informationen:

- NameOfPatternSet - Name des Mustersatzes
- NumberOfPattern - Anzahl der Muster
- DimensionOfFeatureVector - benötigte Featurevektorgroße
- NameOfData - Position der Featurevektorelemente im 2D-Raum
- Pattern - Muster
  - NameOfPattern - Name des jeweiligen Musters
  - Data - Wert des Featurevektors an der jeweiligen Position

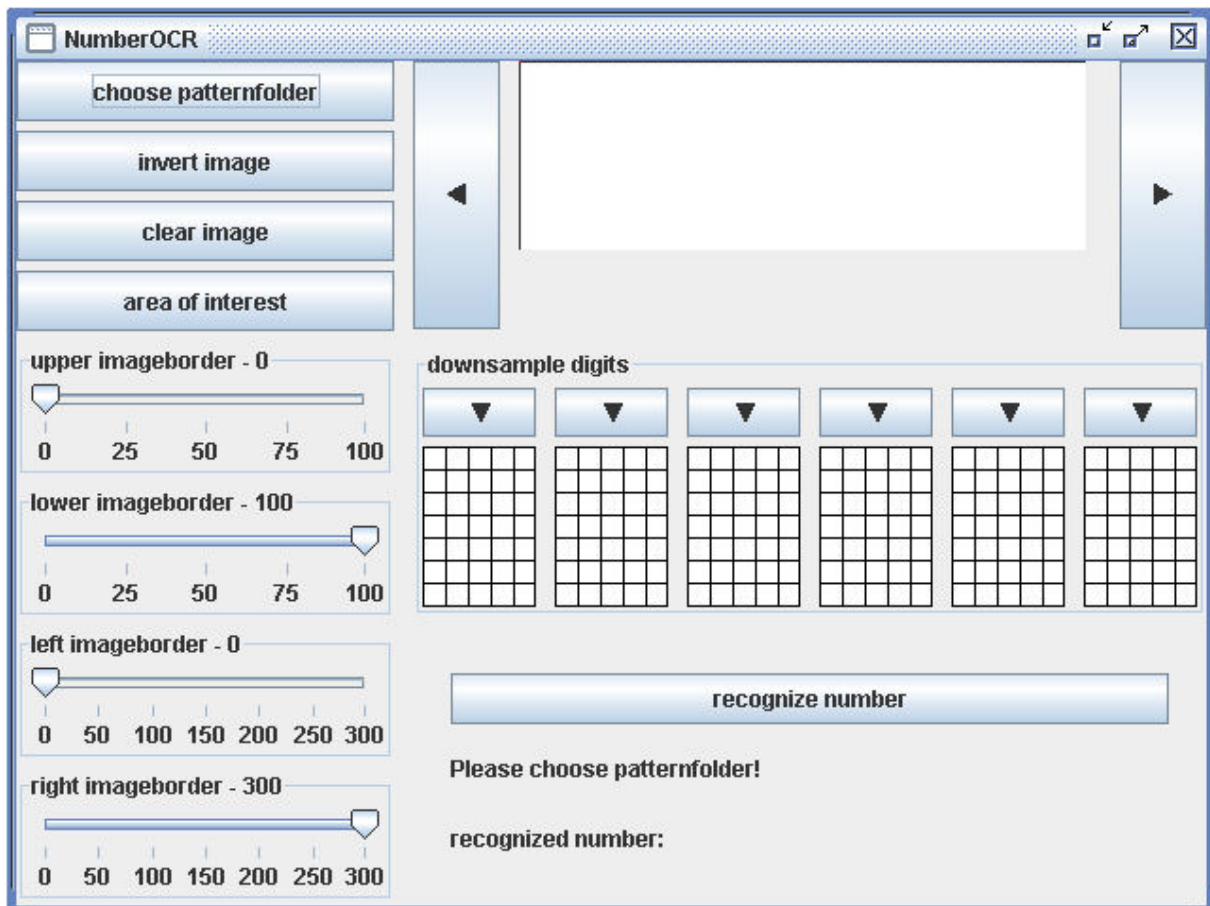
### Auszug aus dem PatternFile Ziffern.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<PatternSet>
  <NameOfPatternSet>Ziffern in 5x7 Matrix</NameOfPatternSet>
  <NumberOfPattern>10</NumberOfPattern>
  <DimensionOfFeatureVector>35</DimensionOfFeatureVector>
  <NameOfData>0-0</NameOfData>
  <NameOfData>1-0</NameOfData>
  <NameOfData>2-0</NameOfData>
  <NameOfData>3-0</NameOfData>
  <NameOfData>4-0</NameOfData>
  <NameOfData>0-1</NameOfData>
  <NameOfData>1-1</NameOfData>
  <NameOfData>2-1</NameOfData>
  <NameOfData>3-1</NameOfData>
  <NameOfData>4-1</NameOfData>
  <NameOfData>0-2</NameOfData>
  <NameOfData>1-2</NameOfData>
  <NameOfData>2-2</NameOfData>
  <NameOfData>3-2</NameOfData>
  <NameOfData>4-2</NameOfData>
  <NameOfData>0-3</NameOfData>
  <NameOfData>1-3</NameOfData>
  <NameOfData>2-3</NameOfData>
  <NameOfData>3-3</NameOfData>
  <NameOfData>4-3</NameOfData>
  <NameOfData>0-4</NameOfData>
  <NameOfData>1-4</NameOfData>
  <NameOfData>2-4</NameOfData>
  <NameOfData>3-4</NameOfData>
  <NameOfData>4-4</NameOfData>
  <NameOfData>0-5</NameOfData>
  <NameOfData>1-5</NameOfData>
  <NameOfData>2-5</NameOfData>
  <NameOfData>3-5</NameOfData>
  <NameOfData>4-5</NameOfData>
  <NameOfData>0-6</NameOfData>
  <NameOfData>1-6</NameOfData>
  <NameOfData>2-6</NameOfData>
```



## 7. Das PlugIn NumberOCR

Das Programm NumberOCR wurde entwickelt um Ziffern verschiedener Schriftarten zu erkennen. In den letzten Jahren wurden einige Websites mit einem Zugangsschutz für Bots ausgerüstet, dies wird über eine Grafik erreicht, welche Buchstaben oder Zahlen, teilweise auch beides enthält. Durch die Verwendung verschiedenster Schriftarten wird eine Erkennung durch Bots umgangen, selbst der Mensch hat oft Probleme die Zeichen zu erkennen. Dies war die Inspiration für dieses PlugIn, welches die Erkennung der Ziffern ermöglichen soll. Eine automatische Zeichentrennung ist leider noch nicht implementiert, daher gibt es die Möglichkeit die Zeichen selbst zu trennen.

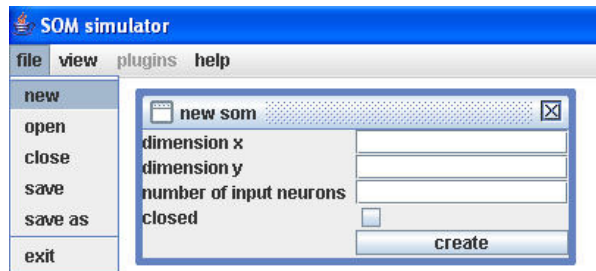


Das PlugIn NumberOCR besteht aus mehreren Bereichen. Im linken oberen Bereich befinden sich die Buttons zum öffnen eines Verzeichnisses, in welchem sich die zu Erkennenden Bilder befinden, sowie zum Invertieren des Bildes, Leeren des Anzeigebereiches und zum Auffinden der Randpixel der Zahl. Darunter befinden sich vier Schieberegler, mit denen Sie die vier Grenzen (oben, unten, links und rechts) der Ziffern einstellen können. Die rechte Seite unterteilt sich in den Anzeigebereich der Zahl, sowie 6 Anzeigeelementen für die getrennten Ziffern und den Ausgabebereich, wo das Ergebnis ausgegeben wird. Links und rechts des Anzeigebereiches sind Buttons zum Wechseln des Bildes positioniert. Die Buttons darunter sorgen für die Übertragung des markierten Bereiches in das darunter befindliche Anzeigeelement. Der Inhalt des markierten Bildausschnittes wird dabei auf eine Größe von 5x7 Pixel heruntergerechnet.

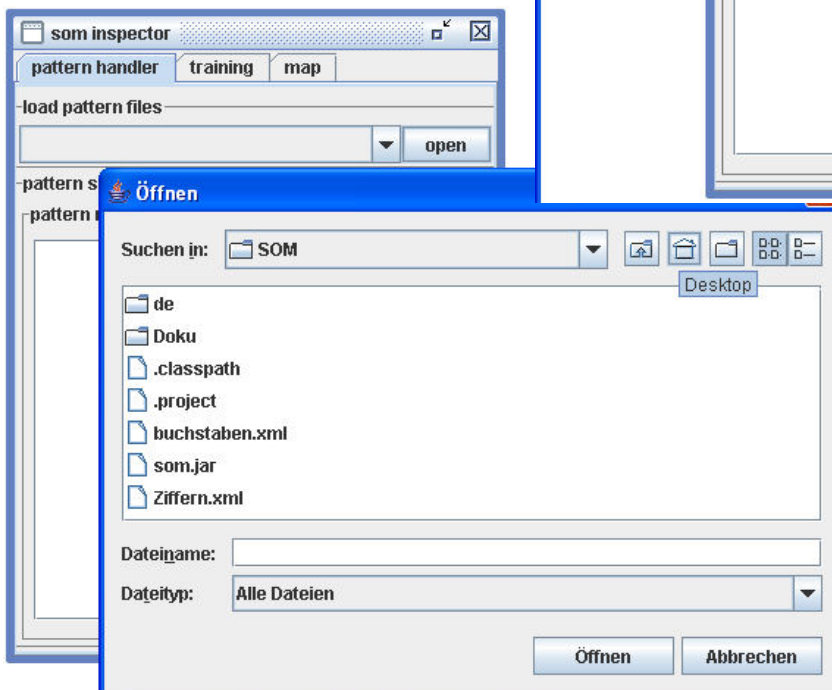
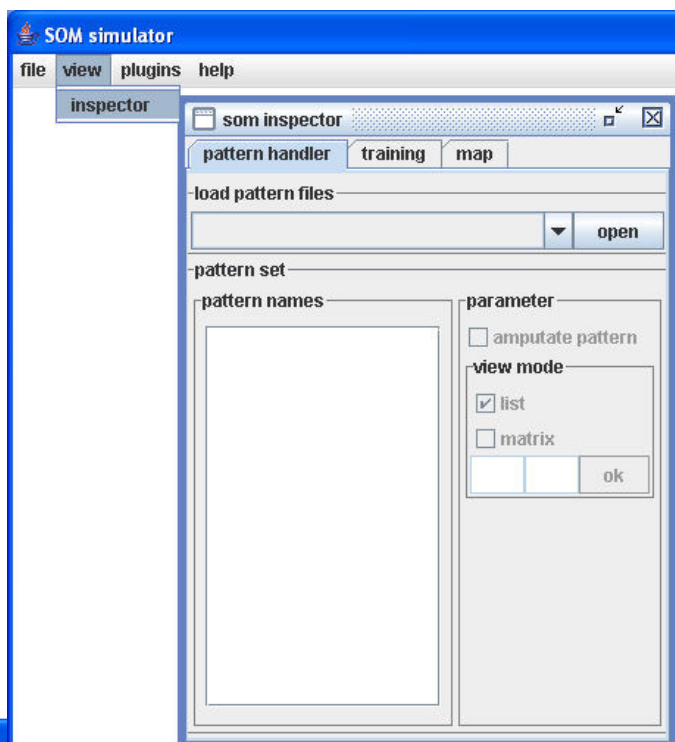
## 8. Die Bedienung

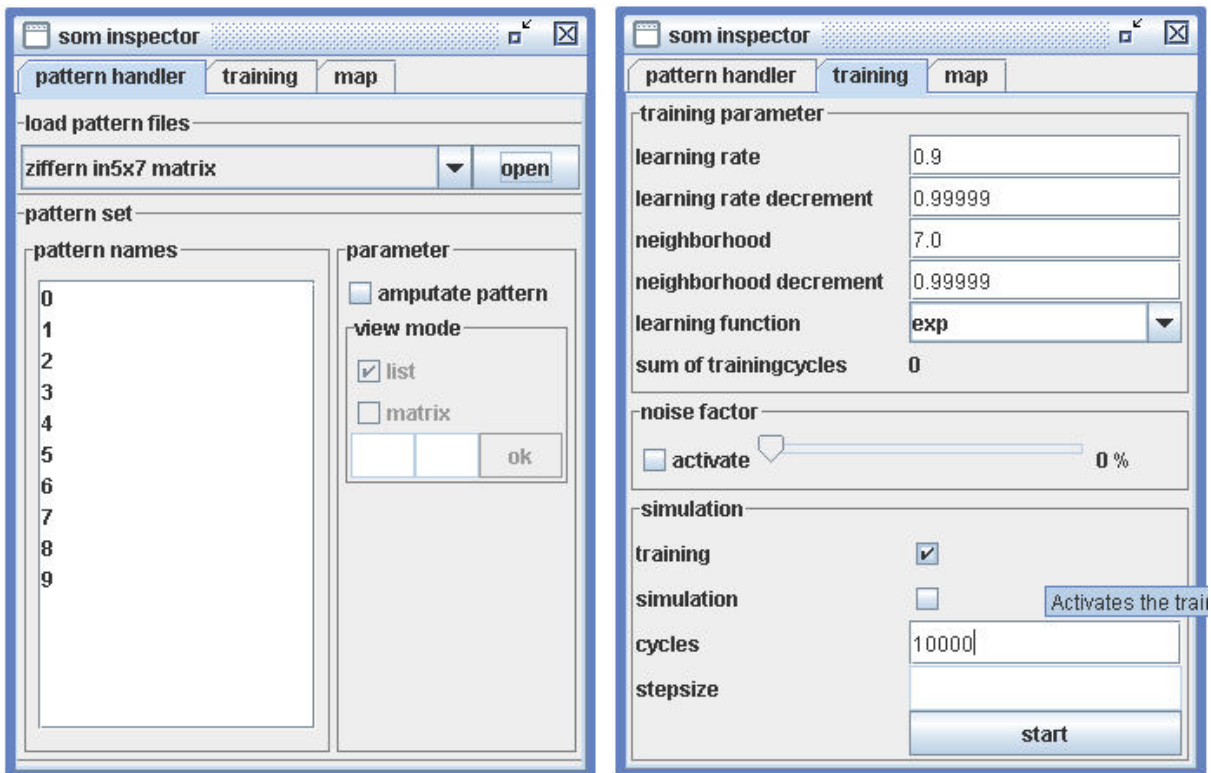
Der erste Schritt nach dem starten ist das Erstellen einer self organizing map. Hierzu klicken Sie bitte auf den Menüeintrag „file“ und dann auf „new“.

Nun wird ein Fenster mit dem Titel „new som“ angezeigt, in welchem Sie die Größe der self organizing map festlegen können. Mittels „dimension x“ und „dimension y“ können Sie eine beliebig große ein- oder zweidimensionale Kohonen-Karte erzeugen und „number of input neurons“ legt die Größe des Featurevektors fest. In alle Felder müssen ganze positive Zahlen eingetragen werden. Erst nach erzeugen der SOM mittels „create“ steht der Menüpunkt „plugins“ zur Verfügung, unter dem auch das Plugin NumberSOM zu finden ist.



Als nächstes können Sie eine Musterdatei einlesen. Dazu öffnen Sie unter „view“ den „inspector“. Im ersten Karteireiter des „inspector“, mit dem Namen „pattern handler“ finden Sie einen Button zum öffnen eines „pattern files“. Dieser öffnet eine Dialogbox zum öffnen einer XML-Datei. Entspricht diese der XML-Definition, so wird der Name des PatternSets im Pulldown Menü und im darunter befindlichen Textfeld die einzelnen Muster angezeigt.





Im Karteireiter „training“ können Sie das SOM trainieren. Ihnen stehen dafür mehrere Einstellungen zur Verfügung um das lernen des SOM zu beeinflussen. So können Sie beispielsweise die Lernrate (learning rate) und die Anfangsnachbarschaft (neighborhood) sowie die Lernfunktion selbst festlegen. Im unteren Bereich müssen Sie zum trainieren des SOM eine Anzahl der Lernschritte (cycles) angeben und mit „start“ wird der Lernprozess gestartet. Während des Lernprozesses wird sich die Nachbarschaft und die Lernrate bei jedem Lernschritt um den Kehrwert des angegebenen decrementes verringern.

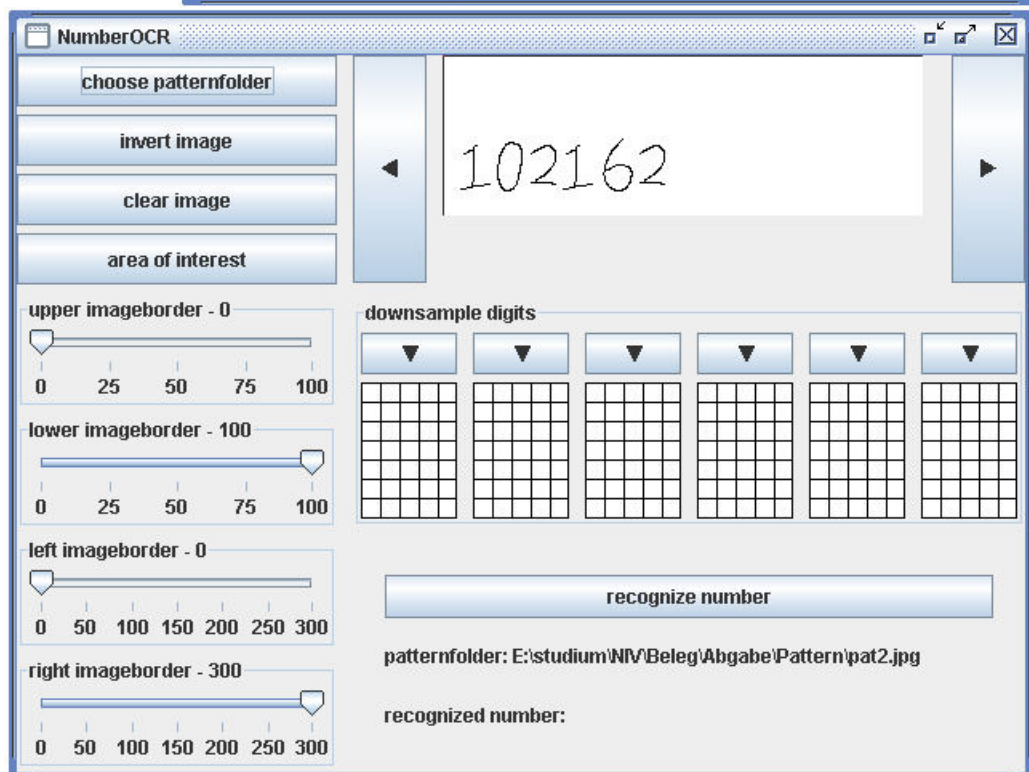
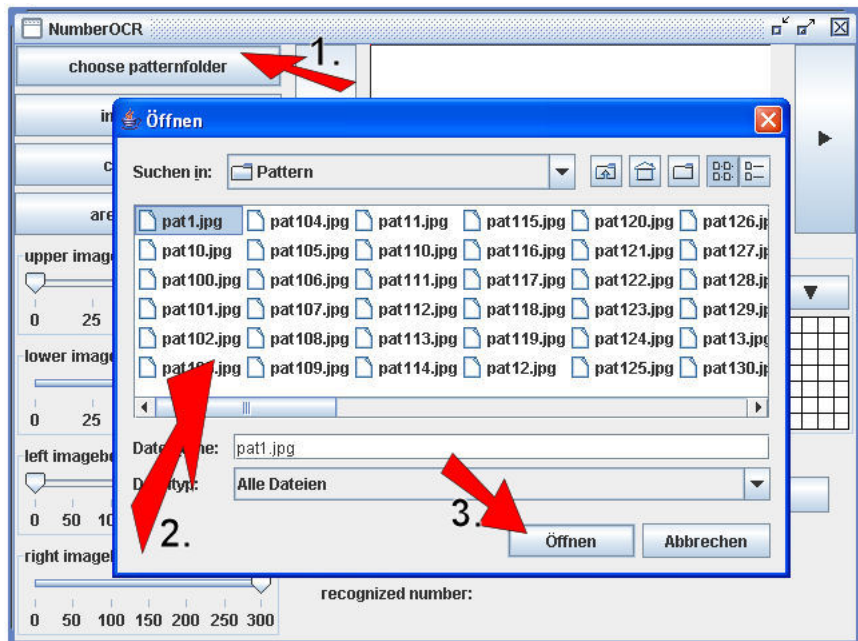
Nachdem das SOM trainiert ist, kann mittels der verfügbaren PlugIns das SOM visualisiert oder die Beziehung zwischen den Mustern angezeigt werden. Neu hinzugekommen ist nun das PlugIn NumberOCR, welches das SOM zur Erkennung von 6-stelligen Zahlenfolgen benutzt.

## 9. Die Bedienung des PlugIns NumberOCR

Das PlugIn NumberOCR ist im Unterschied zu den anderen PlugIns keine passive Anzeige, sondern führt selbst Aktionen aus. Die Bilder werden beim Öffnen binarisiert und skelettiert.

### 9.1 Bild öffnen

Als erstes muss ein Ordner ausgewählt werden, in dem sich die zu erkennenden Bilder befinden. Um dies zu tun, betätigen Sie bitte den Button „open patternfolder“. Die Bilder müssen im JPG-Format vorliegen und eine Auflösung von 300x 100 Pixel besitzen. Wenn Sie einen Ordner ausgewählt haben wird sofort ein Bild aus dem Ordner geladen. Erfüllt dieses die Kriterien, so wird es angezeigt, anderenfalls erscheint eine Fehlermeldung. Durch betätigen der Buttons rechts und links des Bildes können Sie das Bild wechseln. Im unteren Bereich werden Pfad und Dateiname des Bildes angezeigt.



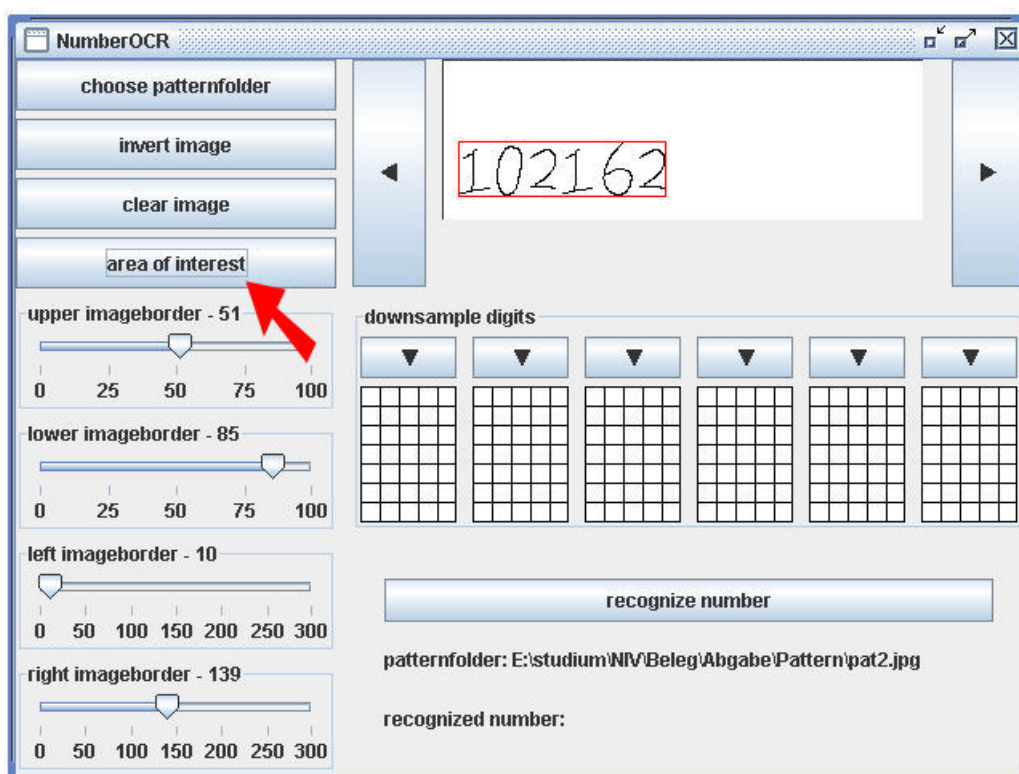
## 9.2 Bild invertieren

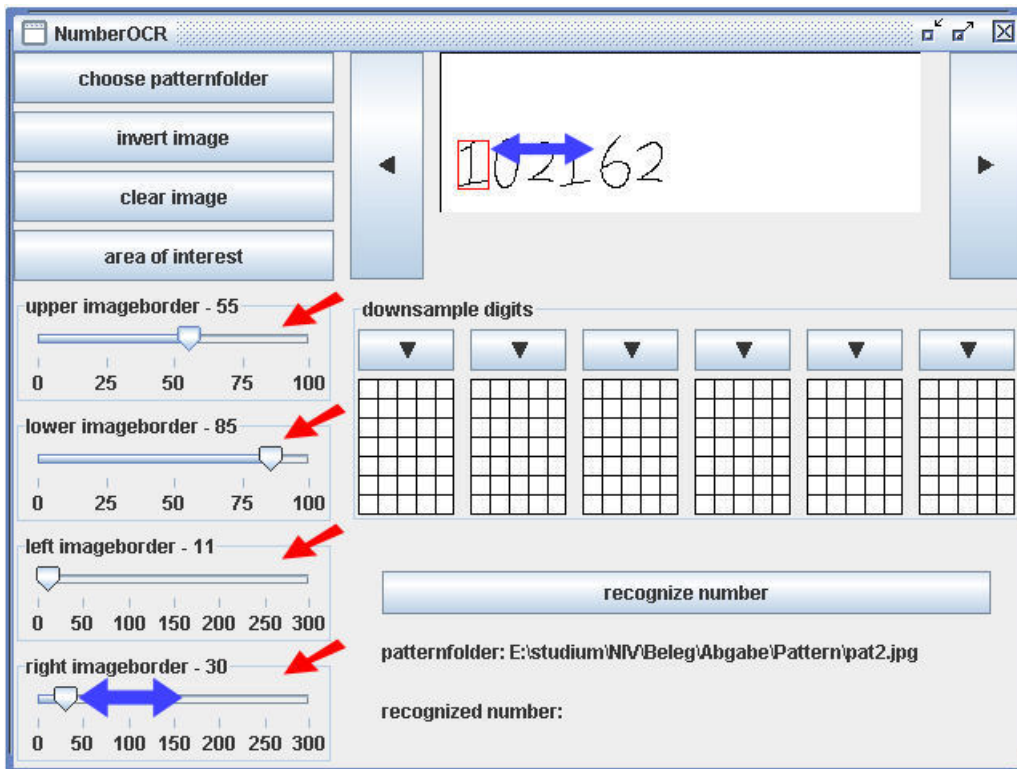
Da die vorliegenden Bilder weisse Schrift auf schwarzem Grund zeigen, invertiert das Programm die Bilder gleich beim öffnen. Sollte das Bild schwarzen Text auf weisem Grund haben, müssen Sie das Bild per „invert image“ invertieren, um eine Erkennung durchführen zu können.



## 9.3 Ziffern trennen

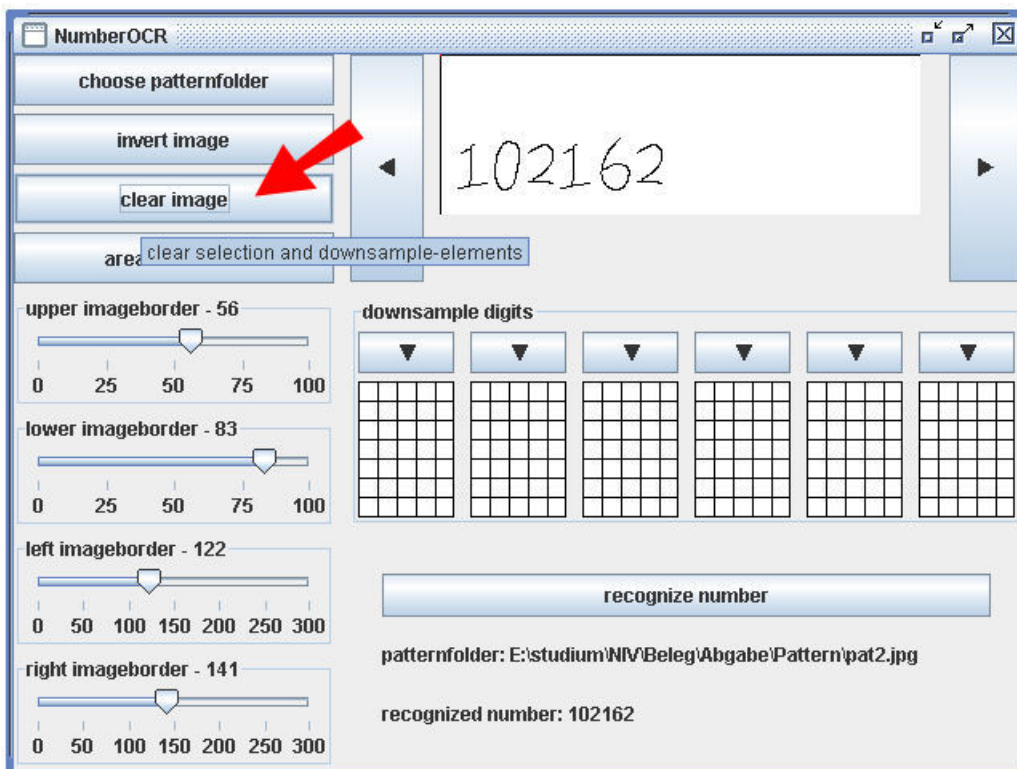
Da die Ziffern nur einzeln erkannt werden können, müssen diese zuerst separiert werden. Um dies zu ermöglichen, stehen vier Schieberegler zur Verfügung, mit deren Hilfe man jede Ziffer in ein Rechteck einschließen kann. Um das Finden der Ränder der Nummer zu vereinfachen, wurde eine Funktion zum Auffinden der Ränder implementiert, welche sich über den Button „area of interest“ aufrufen lässt. Hat man eine Ziffer eingerahmt, so kann man diese mit den unter dem Bild befindlichen Pfeiltasten in das jeweils zugehörige Anzeigeelement kopieren. Das eingerahmte Zeichen wird dabei in ein Bild mit der Größe 5x7 Pixel umgewandelt. Nur Zeichen in den sechs Anzeigeelementen werden dem SOM zur Erkennung vorgelegt.





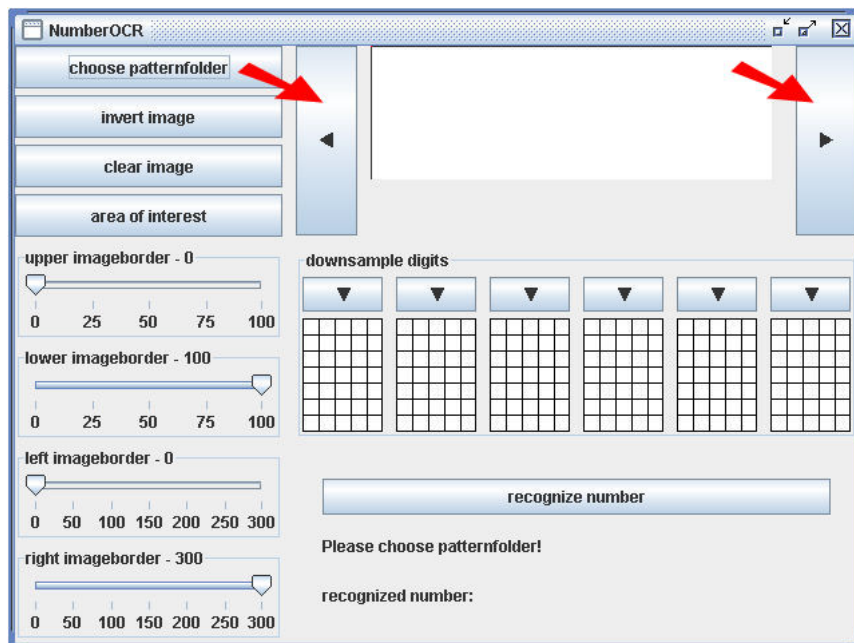
#### 9.4 Anzeigebereich bereinigen

Mit dem Button „clear image“ können Sie die Anzeigeelemente leeren und die Rahmen im Anzeigebereich entfernen.



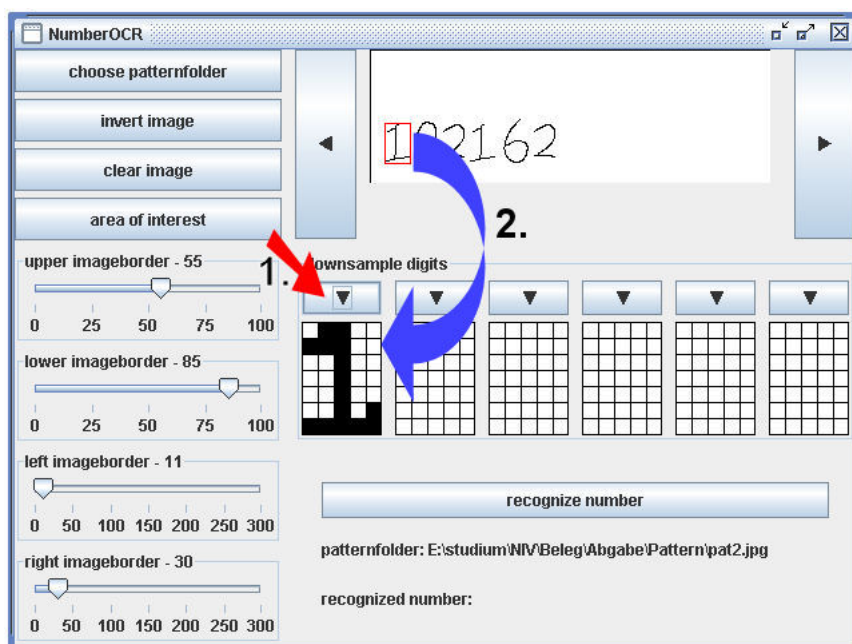
## 9.5 Bild wechseln

Sie können mittels der Pfeiltasten links und rechts des Anzeigebereiches das aktuelle Bild wechseln. Dies ist nur möglich, wenn sich weitere Bilder in dem ausgewählten Ordner befinden.



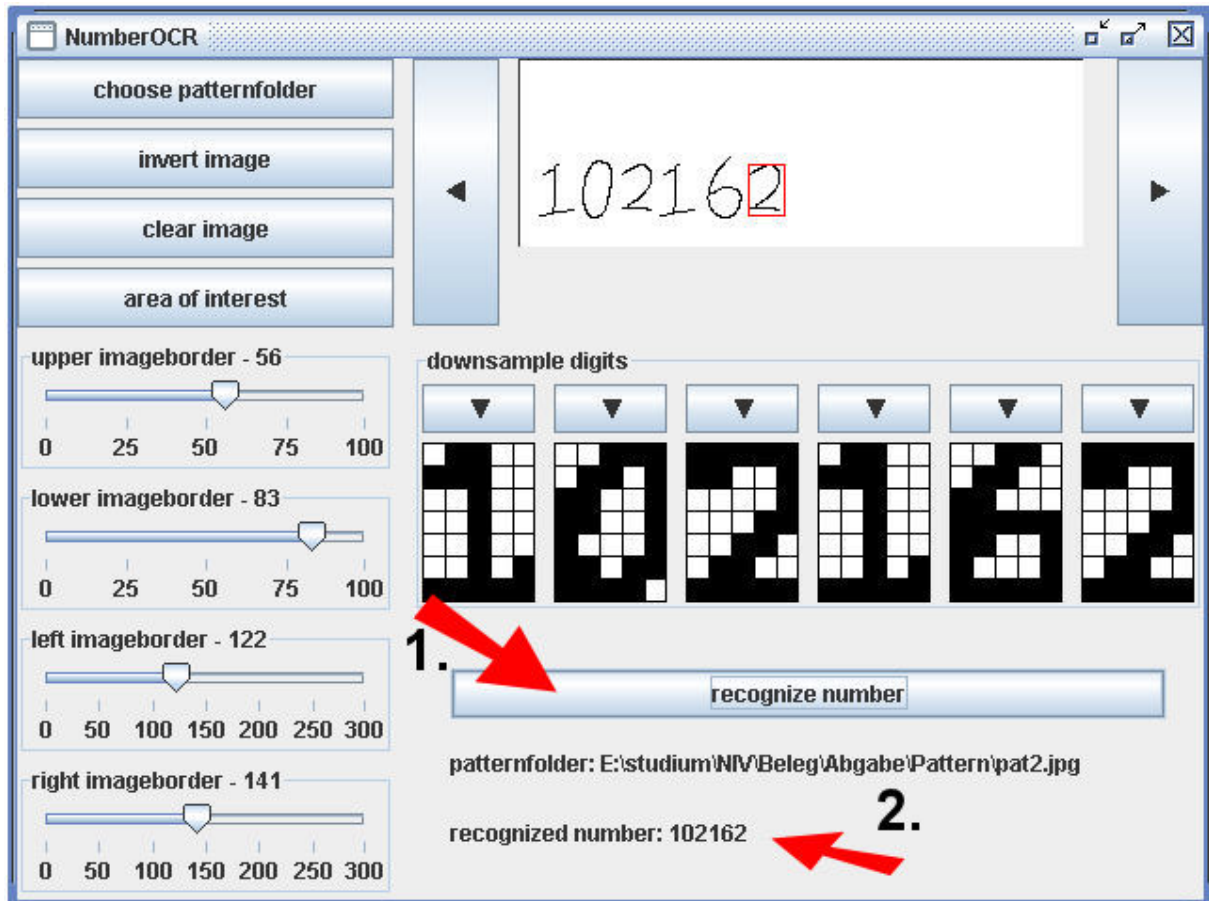
## 9.6 Ziffern herunterrechnen

Durch betätigen der abwärts zeigenden Buttons, kopieren Sie den markierten Bereich in ein 5x7 Pixel großes Anzeigeelement, welches später als Eingabemuster für das SOM dient. Da die Ziffern im Bild größer als 5x7 Pixel sind, werden diese heruntergerechnet.



## 9.7 Erkennen der Nummer

Wenn Sie den Button „recognize number“ betätigen, werden die in den sechs Anzeigeelementen enthaltenen Muster der Reihe nach dem SOM vorgelegt und ermittelt die zugehörige Ziffer. Leere Anzeigeelemente werden dabei übergangen. Das Ergebnis wird dann im unteren Bereich des Plugins angezeigt.



## 10. Fazit

Die im Rahmen des Beleges erarbeitete Software ist so noch nicht produktiv einsetzbar. Sie zeigt jedoch, dass die Nutzung selbstorganisierender Karten im Bereich der Zeichenerkennung ein hohes potential besitzt. So liegt die Erkennungsleistung schon bei nur einem erlernten Muster pro zu erkennendem Zeichen bei über 90%. Der Rechenaufwand zum trainieren des SOM liegt bei wenigen Sekunden. Bei den voreingestellten Lernparametern ist die selbstorganisierende Karte nach etwa siebzig- bis achtzig-tausend lernschritten voll ausgebildet, was etwa 10 Sekunden dauert. Da die Trainingsdauer aber von der Größe des SOM's, der Lernrate, der Nachbarschaft und der Anzahl der erlernbaren Muster abhängt, sind auch deutlich höhere Werte möglich.

## 11. Änderungen am SOM simulator von Markus Schubert

- AboutDialog.java:  
Zeile 19: `this.add(new JLabel("Modified in 2005 by Maik Zachacker"));`  
eingefügt
- Application.java:  
Zeile 3: `import java.awt.*;` ersetzt durch `import.awt.Point;`  
Zeile 4: `import.javafx.swing;` gelöscht
- Main.java:  
Zeile 3-7: Imports reorganisiert  
Zeile 21: `frame.setSize(800,600);` in `frame.setSize(800,600);` geändert
- NoPatternSetFoundException.java:  
Zeile 3: `import de.def.somsimulator.som.SOMEException;` entfernt
- PluginHandler.java:  
Zeile 3-7: Imports reorganisiert  
Zeile 55 & 127: „geladen“ in „loaded“ geändert -> einheitliche Oberfläche  
Zeile 101:  
`String jarName=System.getProperty("user.dir") + "\\som.jar";`  
ersetzt durch  
`String jarName = System.getProperty("user.dir") +  
System.getProperty("file.separator") + "som.jar";`  
zwecks kompatibilität mit Linux
- Simulator.java:  
Zeile 8: `throws Exception` entfernt
- SOMSimulator.java:  
Zeile 82: `this.addChild(this.frameNewSOM,0,0,180,120);`  
geändert in  
`this.addChild(this.frameNewSOM,0,0,300,120);`  
da Größenänderung durch Übersetzung des Textes notwendig  
Zeile 194: „fehler - view nicht verfügbar“ in  
„error - view not available“ geändert -> einheitliche Oberfläche  
Zeile 221: `throws Exception` entfernt  
Zeile 241: `else Zeig mit throw new Exception;` entfernt  
Zeile 248: `throw ex in`  
`System.out.println(„File not found exception - „+ex);`  
geändert,  
`return null;` hinzugefügt

- Zeile 272: „File“ in „file“ geändert, damit Übereinstimmung mit restlichen Menüeinträgen
- Zeile 337-354 und 405-418, 487, 489, 525, 527: enum durch Enum ersetzt, da enum in Java 1.5 reserviert
- Zeile 364: Funktion `private void addChild(JInternalFrame jif)` entfernt, da nicht benutzt
- Zeile 385: Größe des Fensters angepasst, da Text durch Übersetzung länger
- Zeile 448-454: Übersetzung der deutschen Label-Texte
- Zeile 514: Übersetzung der Fehlermeldung
- /views/PatternView.java:
  - Zeile 3-9: Imports reorganisiert
  - Zeile 19, 42, 43, 123, 162, 166, 176: `currentViewMode` entfernt, da nicht benutzt
  - Zeile 194, 199: Fehlermeldungen übersetzt
- /views/SOMMapView.java:
  - Zeile 64-92: Texte übersetzt
- SOMPatternHandlerView.java:
  - Zeile 47: `if(file!=null)` Bedingung eingefügt, da `NullPointerException` beim öffnen weiterer PatternFiles
  - Zeile 63: `catch` Block nicht mehr notwendig, dafür Auswahl des geladenen PatternFile in Combobox
  - Zeile 133: ItemListener für Combobox entfernt, da Fehler beim löschen der Elemente
  - Zeile 134: ItemListener für Combobox wieder hinzugefügt.
  - Zeile 160: Fehlerauschrift hinzugefügt
- SOMTrainingsView.java:
  - Zeile 71, 75: Fehlermeldung übersetzt
  - Zeile 77: `catch` Block ersetzt
  - Zeile 162-225: Label übersetzt
- /som/DefaultPatternSetModel.java:
  - Zeile: 65, 79, 93, 106, 120, 135, 143: Fehlermeldungen übersetzt
- /som/IncompatibleSOMException.java:
  - Zeile 7: `ViewPlugin` in `viewplugin` geändert
- /som/Pattern.java:
  - Zeile 8: `private int size;` entfernt, da nicht notwendig
- /som/SOM.java:
  - Zeile 3-5: Imports reorganisiert
  - Zeile 20: Nachbarschaftsfunktionen übersetzt
  - Zeile 56, 61, 66: `protected` in `public` geändert, da Zugriff durch Klasse `NumberOCR` notwendig
  - Zeile 110: `public Vector[][] patternMap;` hinzugefügt, da für Klasse `NumberOCR` notwendig
  - Zeile 111: `protected` in `public` geändert, da Zugriff durch Klasse `NumberOCR` notwendig
  - Zeile 113, 114: gelöscht, da nicht notwendig
  - Zeile 470: Grenznachbarschaft von 0,2 auf  $(\text{Math.sqrt}(\text{this.nKX} * \text{this.nKY}) / 25)$  geändert, somit Abhängigkeit von Matrixgröße
  - Zeile 567-571: Startwerte für Lernrate, Abnahme der Lernrate, Nachbarschaft und Abnahme der Nachbarschaft geändert. Anfangsnachbarschaft abhängig von Matrixgröße (50% der

Neuronen) ( $\text{Math.sqrt}(\text{this.nKX} * \text{this.nKY}) / 2$ )

- Zeile 684: Groß-/Kleinschreibung korrigiert
- /plugins/tsp/TSP.java:
  - Zeile 76: Groß-/Kleinschreibung korrigiert
- /plugins/tsp/TSPMapView.java:
  - Zeile 10-13, 27-29: entfernt, da nicht notwendig
  - Zeile 76: Groß-/Kleinschreibung korrigiert
- /plugins/topo/SOMTopology.java:
  - Zeile 3-10: Imports reorganisiert
  - Zeile 15: entfernt, da nicht notwendig
  - Zeile 66, 70, 129, 143, 172, 178, 202:
    - Änderung von `this.patternMap` in `this.som.patternMap`
  - Zeile 81: Rechtschreibfehler beseitigt
  - Zeile 89, 91: Rechtschreibfehler beseitigt
  - Zeile 125: `if(this.view!=null)` Bedingung eingefügt, da sonst Fehler beim Training wenn Plugin geöffnet und anschließend wieder geschlossen wurde
- /plugins/topo/SOMTopologyView.java:
  - Zeile 3-7: Imports reorganisiert
  - Zeile 11, 15, 17, 20-22, 26, 39, 42, 43, 46, 67-69, 98-102, 204, 210: entfernt, da nicht notwendig
- /plugins/net/Net.java:
  - Zeile 75: Rechtschreibfehler beseitigt
- /maps/SOMComponentMap.java:
  - Zeile 131: `if(this.view!=null)` Bedingung eingefügt, da sonst Fehler beim Training wenn Plugin geöffnet und anschließend wieder geschlossen wurde
- /maps/SOMComponentMapView.java:
  - Zeile 3-7: Imports reorganisiert
  - Zeile 21, 23, 28-30, 47, 71-73, 133: entfernt, da nicht notwendig

### ***Ergebnis der Änderungen:***

- Oberfläche englisch – keine Mischung aus deutschem und englischen Text mehr
- Menü-Einträge einheitlich, keine Mischung aus Groß- und Kleinschreibung mehr
- Fehler beim Öffnen von PatternFiles behoben
- Fehler beim Lernen mit geschlossenen Plugins behoben, wenn diese schon einmal geöffnet wurden
- Programm ist jetzt auch unter Linux lauffähig