

Hochschule für Technik und Wirtschaft (FH)

Fachbereich Informatik/Mathematik

Dokumentation zum Beleg

Farbquantisierung durch neuronale Netze

im Fach Neuroinformatik I

Bearbeitet von:

Juliane Korsinek

Matr.Nr: 16621

Bib.Nr: s51315

Betreuender Professor:

Prof. Dr. rer. nat. habil. H. Iwe

Dresden, 1. Februar 2006

Farbquantisierung durch neuronale Netze	0
1. Einleitung	2
2. Problemstellung	3
3. Aufbau des Versuches	4
3.1 Das neuronale Netz	4
3.2 “The Winner takes it all”	5
3.3 Ein Netz lernt	7
4. Die Applikation	9
4.1 Laden eines Bildes und initialisieren des Netzes	9
4.2 Training und Trainingsparameter	13
4.3 Das Ergebnis	14
5. Fazit	18
Literaturverzeichnis	19
Quellenverzeichnis	19

1. Einleitung

Dieser Beleg entstand im Rahmen des Faches Neuroinformatik. Er soll dazu dienen, die Leistungsfähigkeit neuronaler Netze zu verdeutlichen.

Die Idee des Beleges war es, eine Farbquantisierung mit Hilfe eines neuronalen Netzes durchzuführen. Diese Farbquantisierung soll dazu dienen, den Farbbereich eines 24-Bit tiefen Bildes, mit großer Anzahl verschiedener Farben, auf höchstens 256 zu reduzieren und das Ursprungsbild in ein 8-Bit-Paletten-Bild zu verwandeln.

Diese Aufgabe musste ich bereits während meines Betriebs-Praktikums lösen. Damals habe ich die sogenannte Octree-Methode verwendet. Sie ist, in Bezug auf andere Methoden der Farbquantisierung, relativ einfach zu implementieren, besitzt eine durchschnittliche Rechenzeit und verbraucht relativ wenig Ressourcen. Die Nachteile sind, dass die Ergebnisse teilweise nicht so optimal ausfallen.

Meine Hoffnung bei der Verwendung eines neuronalen Netzes zur Lösung des Problems waren daher: bessere Farbergebnisse mit besseren Durchschnittswerten bei ungefähr gleicher Rechenzeit und etwa gleichem Ressourcenverbrauch.

Als Entwicklungsumgebung habe ich Visual-C++, zur Visualisierung MFC verwendet.

2. Problemstellung

In der heutigen Zeit, in denen Computern eine Masse von Ressourcen und eine Menge von Speicher im Gigabyte-Bereich zur Verfügung stehen und in denen Internet-Verbindungen so rasant wie noch nie Daten übertragen können, können Bilder in Fotoqualität abgespeichert und verbreitet werden.

Trotzdem ist es aus diversen Gründen manchmal notwendig, die Anzahl der Farben eines Bildes zu reduzieren und die Speicher zu verringern.

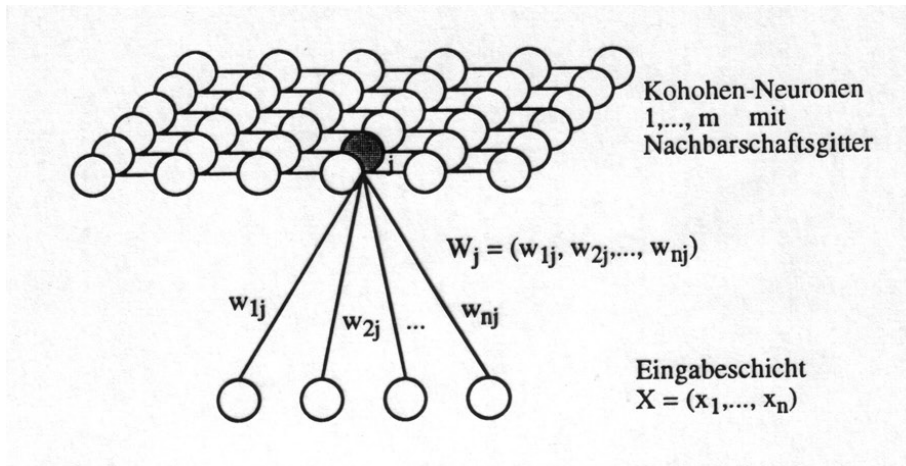
Zum Beispiel ist dies nötig, um 24-Bit tiefe RGB-Bilder im GIF-Format oder im 8-Bit-BMP-Format abzuspeichern. Diese Formate dienten ursprünglich dazu, Bilder, zum teil animiert, über die damals noch nicht so schnellen Datenverbindungen, auszutauschen und so schnell wie möglich zur Anzeigen bringen zu können.

Um Bilder in die angesprochenen Formate übertragen zu können, muss die Anzahl der Farben eines Bildes auf mindestens 256 herunterreduziert werden, die übrigbleibenden Farben in eine Farbtabelle eingetragen werden und die einzelnen Pixel durch Indizes in die Farbtabelle ersetzt werden.

3. Aufbau des Versuches

3.1 Das neuronale Netz

Als neuronales Netz wurde eine selbstorganisierende Karte (SOM - Self Organizing Map) ausgewählt. Diese besteht zumeist aus zwei Schichten, der Eingabeschicht und der Kohonen-Schicht:



Die Neuronen der Eingabeschicht sind über Gewichte mit den Neuronen der Kohonen-Schicht verbunden. Jedes Neuron der Eingabeschicht ist mit jedem Neuron der Kohonen-Schicht verbunden. Jedes Neuron der Kohonen-Schicht ist mit jedem Neuron der Eingabeschicht verbunden. Damit besitzt jedes Kohonen-Neuron einen Gewichtsvektor, dessen Dimension der Dimension der Eingabeschicht entspricht.

Die Neuronen der Kohonen-Schicht sind jeweils untereinander, mit den jeweiligen Nachbarn, verbunden. Dadurch ergibt sich eine Nachbarschaftsbeziehung, die in der Lernphase zum tragen kommt.

Die Dimension der Kohonen-Schicht wurde eindimensional mit 256 Neuronen gewählt. Dadurch lässt sich das Ergebnis leichter in die Form einer Tabelle übertragen. Die Eingangsschicht besteht aus jeweils 3 Neuronen, die jeweils einem Farbkanal des RGB-Raums zugewiesen sind.

3.2 “The Winner takes it all”

Das Prinzip der Kohonen-Karte lautet “The Winner takes it all“. Dabei wird zu einem Eingabe-Vektor das Erregungszentrum in der Kohonen-Schicht gesucht. Als Erregungszentrum gilt das Neuron, dessen Gewichtsvektor mit dem Eingabevektor am ehesten übereinstimmt.

Für das Auffinden des „Gewinner-Neurons“ während der Lernphase wird im Versuch folgende Formel verwendet:

$$d(X, W_j) = \sum_{i=0}^{n-1} (x_i - w_{ij}) - \mathbf{b}_j = (x_0 - w_{0j}) + \dots + (x_{n-1} - w_{n-1j}) - \mathbf{b}_j$$

mit $n=3$ (Dimension der Eingabeschicht), x_i – i -tes Neuron in der Eingabeschicht, w_{ij} – Gewichtsvektor vom i -ten Eingabeneuron zum j -ten Kohonen-Neuron, b_j – Bias des j -ten Kohonen-Neurons. Dabei gilt dasjenige Neuron als Gewinner, für das dieser minimal wird.

Der **Bias** soll dazu dienen, ein gleichverteilteres Lernen zu ermöglichen. Ein negativer Bias-Wert verringert die Wahrscheinlichkeit eines Neurons als Gewinner ausgewählt zu werden, ein positiver Wert erhöht die Wahrscheinlichkeit. Der Bias berechnet sich durch folgende Formel:

$$b_i = \gamma \left(\frac{1}{256} - f_i \right)$$

Dabei sind die Elemente der Formel wie folgt definiert:

γ ist eine Konstante, die mit 2^{10} initialisiert wird. f_i ist die Abschätzung der Frequenz, mit der der Gewichtsvektor i der Eingabe am nächsten kommt. Die Frequenz wird zu Beginn mit dem Wert $\frac{1}{256}$ initialisiert. Damit bekommt jedes Neuron zunächst den Bias-Wert 0 zugewiesen.

Nun werden in jedem Suchvorgang zunächst die Frequenzwerte der einzelnen Neuronen verändert. Die Formeln dafür lauten:

$$f_i := \begin{cases} f_i - \beta f_i + \beta & \text{für das Gewinner-Neuron} \\ f_i - \beta f_i & \text{für alle anderen Neuronen} \end{cases}$$

mit $\beta = \frac{1}{1024}$. Die einzelnen Bias-Werte verändern sich danach in folgender Weise:

$$b_i := \begin{cases} b_i + \gamma \beta f_i - \gamma \beta & \text{für das Gewinner-Neuron} \\ b_i + \gamma \beta f_i & \text{für alle anderen Neuronen} \end{cases}$$

Außerhalb des Lernprozesses wird der Abstand ohne den berechnet, um ein Verfälschen der Ergebnisse zu verhindern:

$$d(X, W_j) = \sum_{i=0}^{n-1} (x_i - w_{ij}) = (x_0 - w_{0j}) + \dots + (x_{n-1} - w_{n-1j})$$

Bemerkung: Die Werte für die Berechnung des Bias sind der Publikation [Q2] entnommen.

3.3 Ein Netz lernt

Wenn das Erregungszentrum gefunden ist, werden nun die Gewichte des Gewinner-Neurons und dessen Nachbarschaft, in geringerer Form, verändert. Sie werden so zu sagen in Richtung der Eingabe verschoben. Die Verschiebung der Gewichte erfolgt durch folgende Lernformel:

$$W_j(t+1) = W_j(t) + \eta \cdot h_{c_j}(t) \cdot (//X - W_j(t)//)$$

Die Lernrate η wird in jedem Durchlauf exponentiell verringert:

$$\eta(t) = e^{-0.03t}$$

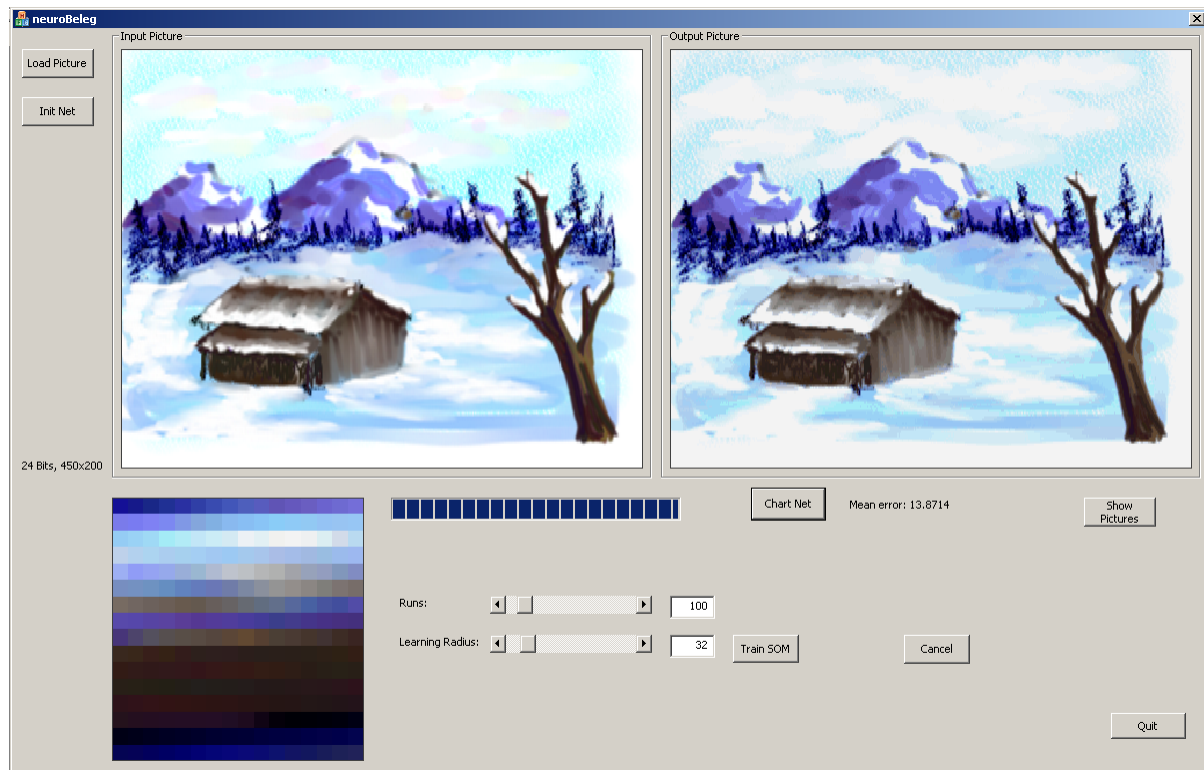
Die Nachbarschaftsfunktion h wurde konvex gewählt (siehe [L4], S. 19, 1.16):

$$h(t) = 1 - \left(\frac{x - y}{[\delta(t)]} \right)^2$$


Der Nachbarschaftsradius δ wird ebenfalls in jedem Durchlauf von einem Startwert R aus exponentiell verringert:

$$\delta(t) = R \cdot e^{-0.0325t}$$

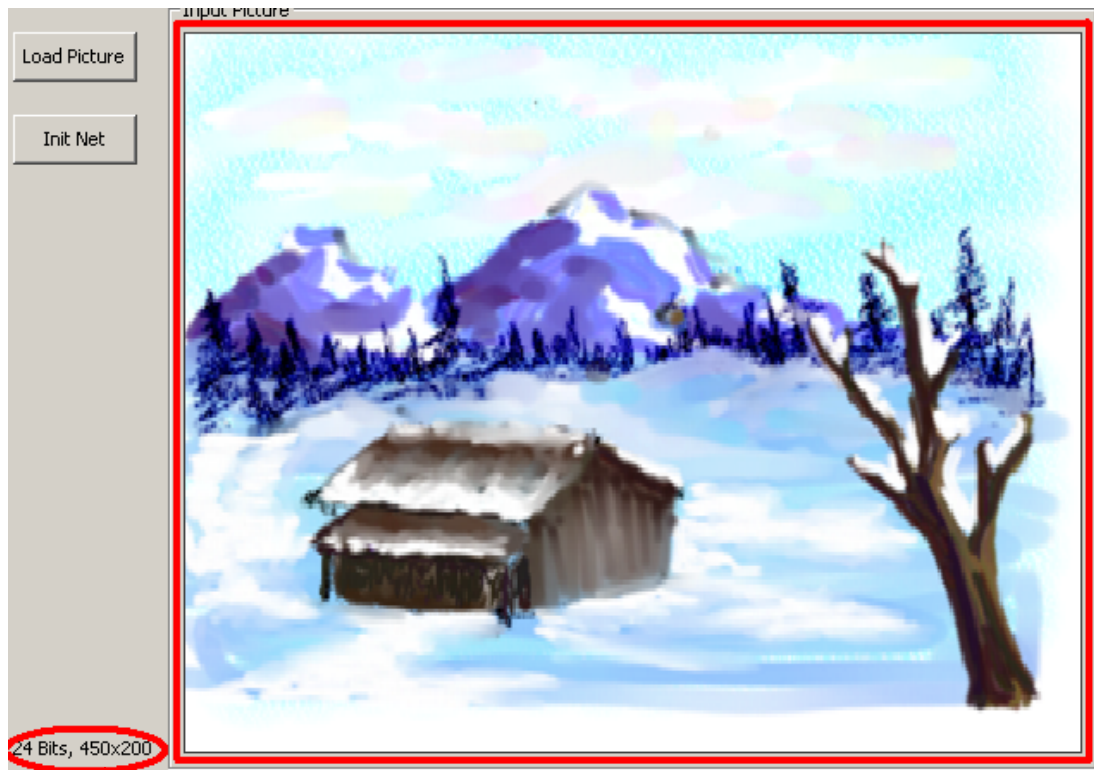
4. Die Applikation



4.1 Laden eines Bildes und initialisieren des Netzes

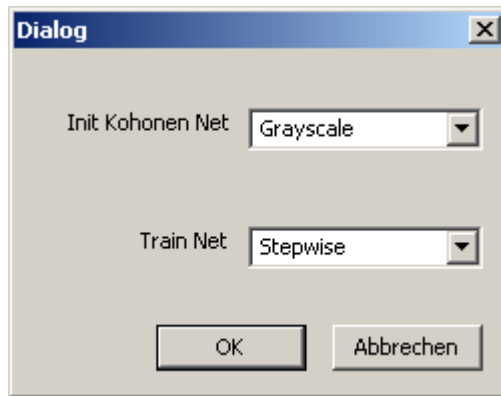
Zunächst muss über den Button  ein Bild in die Applikation eingeladen werden. Es erscheint der Windows-File-Dialog, in dem ein BMP-Bild ausgewählt werden kann.

Das ausgewählte Bild wird dann in einem Vorschaufenster angezeigt:



Die Ausmaße und die jeweilige Tiefe des Bildes werden unteren linken Rand angezeigt.

Über den Button wird ein Dialog aufgerufen, in dem ausgewählt werden kann, in welcher Art und Weise das Netz initialisiert wird und wie der Lerndatensatz ausgewählt werden soll:



Zur Auswahl stehen für die Initialisierung:

- Grayscale:

Die Gewichtsvektors der Kohonen-Neuronen werden in Absteigender Reihenfolge mit Grauwerten belegt: $(W_0, \dots, W_{255}) = ([255,255,255], \dots, [0,0,0])$.

- Random:

Die Gewichtsvektoren werden mit Zufallswerten initialisiert.

Für die Wahl der Eingabevektoren stehen folgende Möglichkeiten zur Auswahl:

- Stepwise:

Hierbei wird das Eingangsbild in einer bestimmte Anzahl von Schritten abgetastet und die erfassten Farbwerte jeweils in das Netz eingegeben.

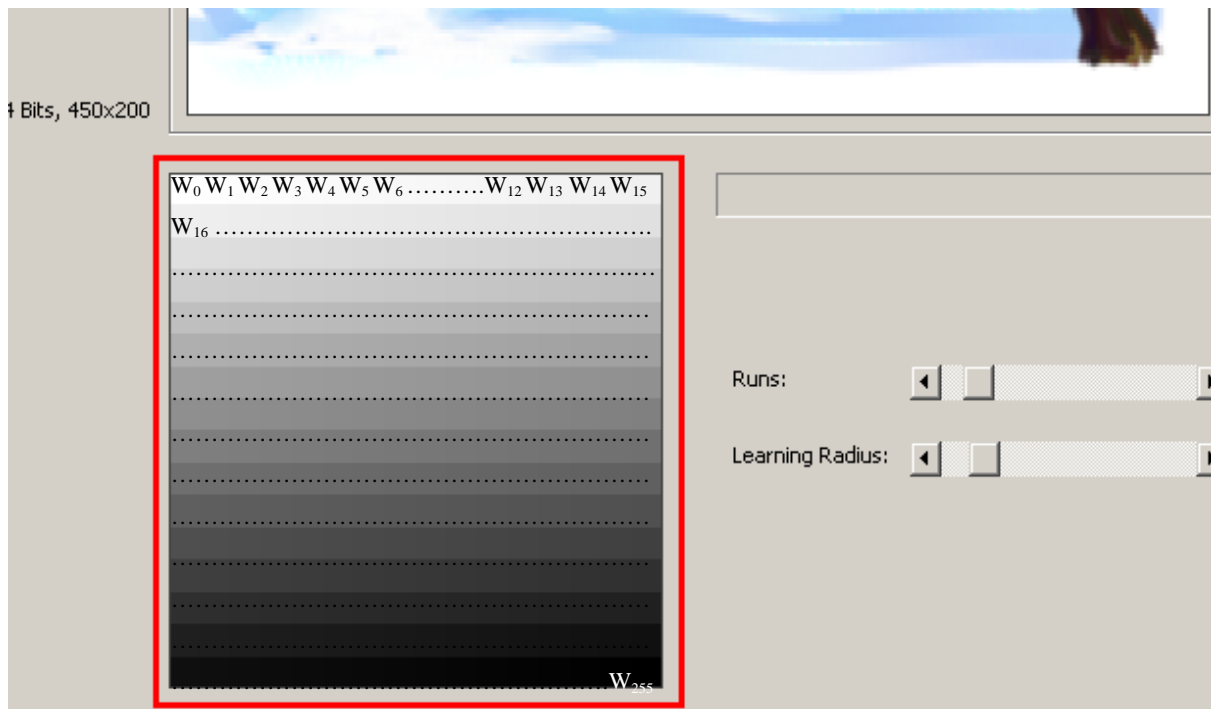
- Preselected Colors:

Hier wird aus der Menge der verschiedenen Farben des Eingangsbildes eine Art Vortabelle erstellt, deren Farben einen bestimmten Abstand zueinander haben. Der jeweilige Abstand kann vor dem Training eingestellt werden (s.u.)

- Random:

Hierbei wird durch Zufall eine bestimmte Anzahl an Punkten im Eingangsbildes ausgewählt und in das Netz eingegeben. Wie viele Zufallspunkte pro Durchlauf gewählt werden, kann vor dem Training eingestellt werden.

Nach einem Klick auf wird das Netz initialisiert. Die initialisierten Gewichtsvektoren werden grafisch in der unteren linken Ecke der Applikation dargestellt:

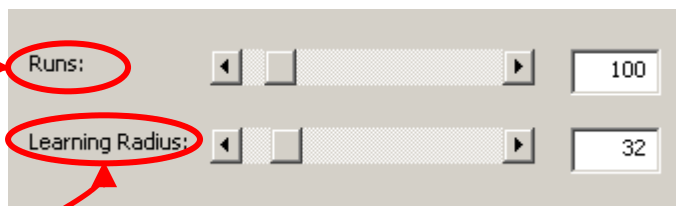


Die einzelnen Werte der Gewichtsvektoren werden dabei in Farbwerte umgesetzt. Aus Platzgründen und zur besseren Ansicht wird die eigentlich eindimensionale Karte als 16x16-Matrix dargestellt.

4.2 Training und Trainingsparameter

Vor dem Training des Netzes, können, je nach ausgewählter Trainingsart (bzw. der Art der Bestimmung des Trainingsatzes), noch einige Parameter eingestellt werden:

- Stepwise:



The image shows a software interface for setting training parameters. It features two horizontal sliders. The first slider is labeled 'Runs' and has a numerical value of 100. The second slider is labeled 'Learning Radius' and has a numerical value of 32. Both labels are circled in red. A red arrow points from the 'Learning Radius' label to the text below.

Hier können nur die Anzahl der Durchläufe (Runs) und der Nachbarschaftsradius (Learning Radius).

- Preselected Colors:



The image shows a software interface for setting training parameters. It features three horizontal sliders. The first slider is labeled 'Runs' with a value of 100. The second slider is labeled 'Learning Radius' with a value of 32. The third slider is labeled 'Pixel Distance' with a value of 80. The 'Pixel Distance' label is circled in red. A red arrow points from the 'Pixel Distance' label to the text below.

Bei dieser Variante kann zusätzlich noch der durchschnittliche Abstand (Pixel Distance), der mindestens zwischen zwei Farben liegen muss, eingestellt werden.

- Random:



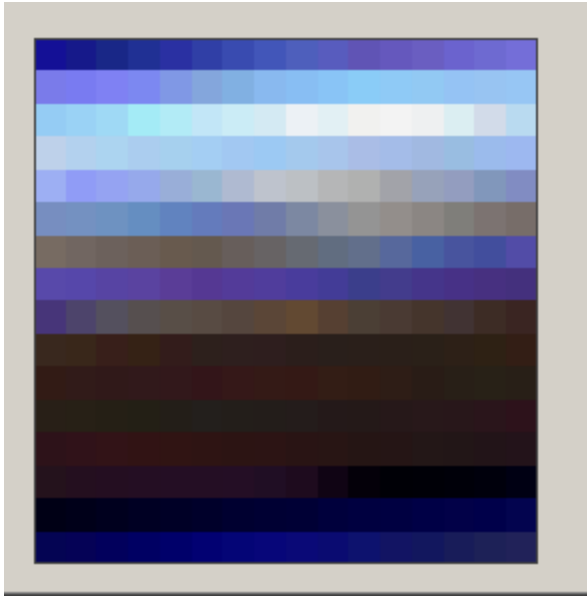
The image shows a control panel with three sliders. The first slider is labeled 'Runs' and has a value of 100. The second slider is labeled 'Learning Radius' and has a value of 32. The third slider is labeled 'Train Set Size' and has a value of 3428. The 'Train Set Size' label and its corresponding slider are circled in red. A red arrow originates from the bottom of this circle and points towards the text in the following paragraph.

Hier kann zusätzlich zu der Anzahl der Durchläufe und dem Nachbarschaftsradius die Größe des Trainingssets angegeben werden.

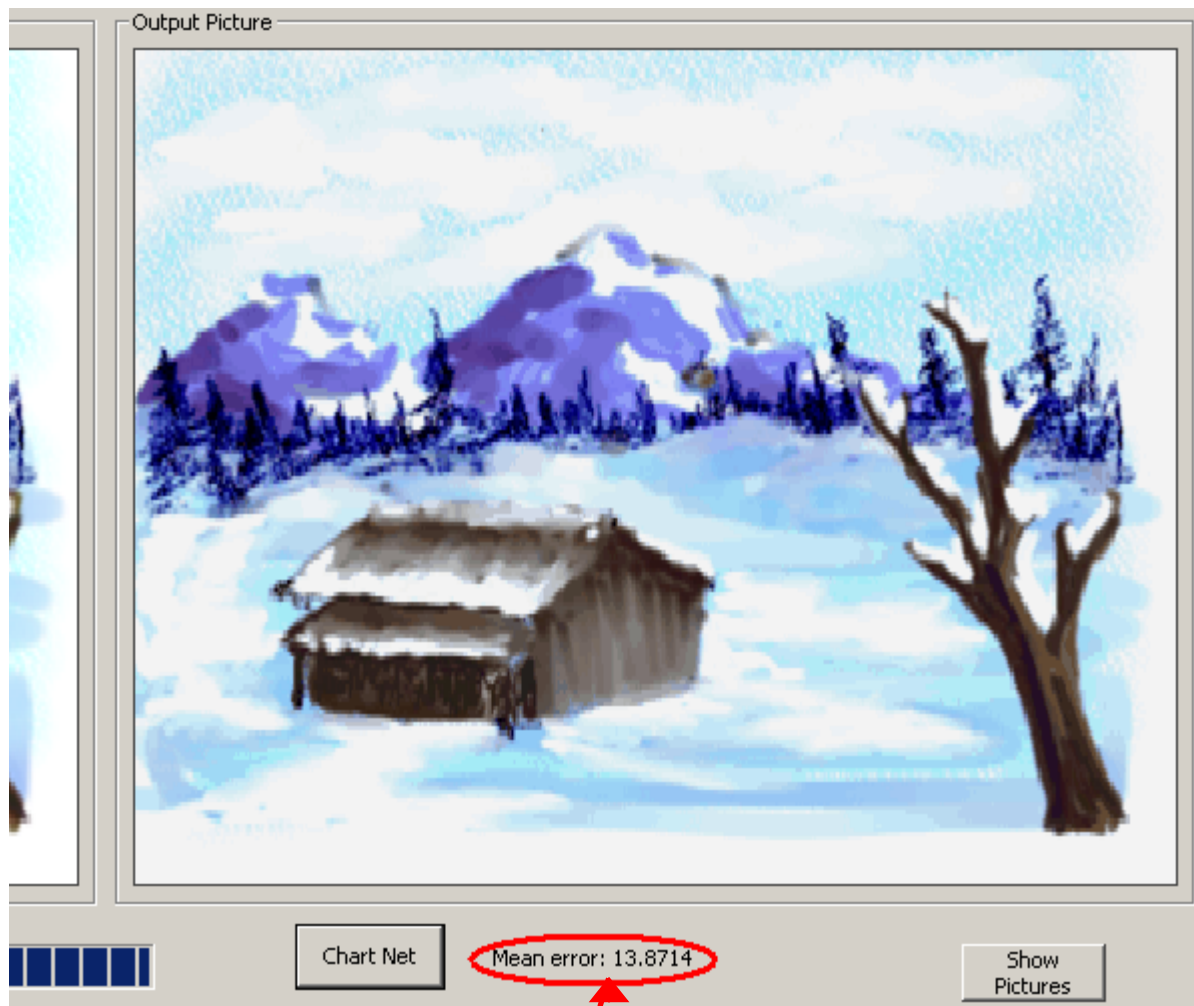
Nach einem Klick auf  Startet das Training.

4.3 Das Ergebnis

Das Ergebnis des trainierten Netzes wird nun wieder in der unteren linken Ecke der Applikation graphisch dargestellt. Dafür werden wiederum die Werte der Gewichtsvektoren in Farbwerte umgesetzt:

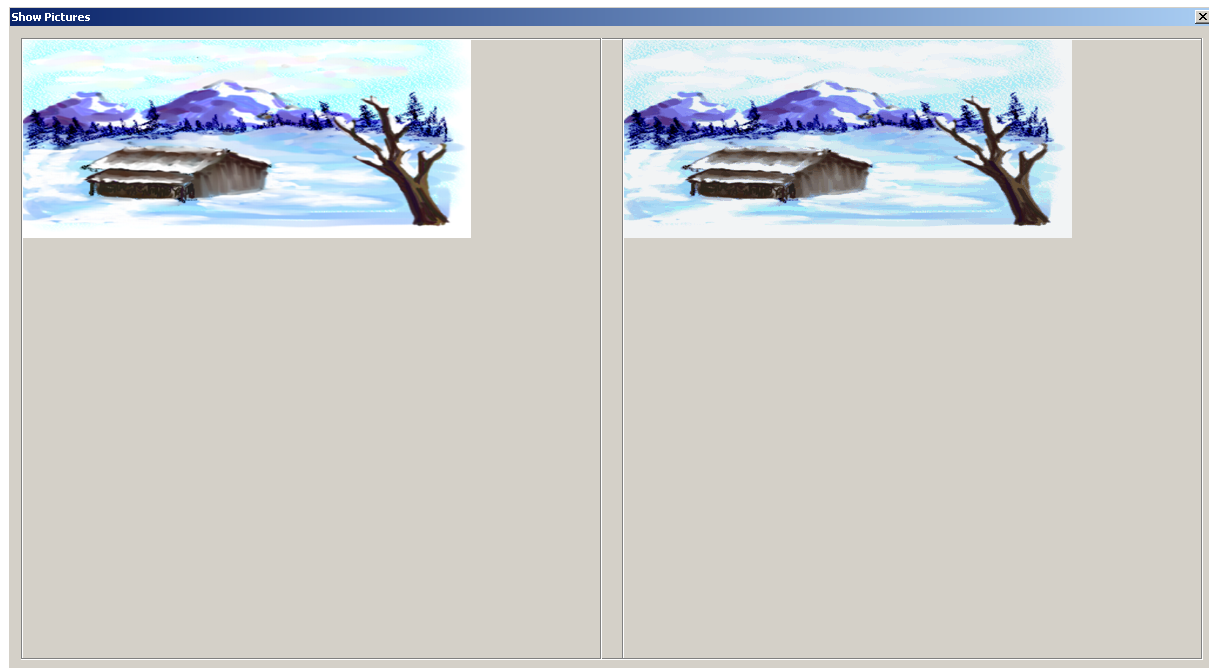


Durch einen Klick auf wird das trainierte Netz auf das Startbild angewendet. Das Ergebnisbild wird neben dem Ausgangsbild dargestellt:


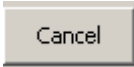


Neben der Anwendung der Kohonen-Karte auf das Ausgangsbild, wird auch der durchschnittliche Fehler des Ergebnisbildes zum Ausgangsbild berechnet.

Über können die Beiden Bilder in Originalgröße (bzw. vergrößert) in einem Extradialog angezeigt werden:



Der mit <Esc> oder über  Abgebrochen wird.

Die Applikation kann über den Button  verlassen. Über  können Vorgänge jederzeit abgebrochen werden. Es kann auch jederzeit ein neues Bild in die Applikation eingeladen werden, das Netz auf neue Art initialisiert werden, oder das Netz nachtrainiert werden.

5. Fazit

In Bezug auf Rechenzeit und den Verbrauch von Ressourcen ist das eingesetzte System der selbstorganisierenden Karte recht gut. Die Zeit, die für das Training gebraucht wird, hängt im wesentlichen von Größe des Trainingsatzes und der Anzahl der Trainingsdurchläufe ab. Da aber schon eine relativ kleiner Datensatz und eine relativ geringe Anzahl von Durchläufen zu recht guten Ergebnissen führen, hält sich auch die Rechenzeit in Grenzen.

Die Farbergebnisse entsprechen nicht ganz meinen Erwartungen. Sie liegen aber in einem durchaus annehmbaren Bereich.

Im großen und ganzen kann man sagen, dass neuronale Netze, speziell selbstorganisierende Karten, sehr leistungsfähige Systeme darstellen, die in sehr unterschiedlichen Bereichen Anwendung finden können. Je nach Problemstellung können sie gute bis sehr gute Ergebnisse liefern, was jedoch auch stark von den jeweils eingesetzten Parametern abhängen kann. „Optimale“ Werte in Bezug auf Parameter, Formeln und Einstellungen für die Lösung eines Problems zu finden, ist sicherlich nicht gerade einfach.

Literaturverzeichnis

- [1] Kohonen, Teuvo: Self-Organizing Maps, Third Edition; Springer Verlag, Berlin/Heidelberg, 2001
- [2] Scherer, Andreas: Neuronal Netze. Grundlagen und Anwendung.; Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1997
- [3] Braun, Heinrich: Neuronale Netze. Optimierung durch Lernen und Evolution.; Springer Verlag, Berlin/Heidelberg, 1997
- [4] Speckmann, Heike: Dem Denken abgeschaut. Neuronale Netze im praktischen Einsatz.; Friedr. Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig/Wiesbaden, 1996
- [5] Haun, Matthias: Simulation neuronaler Netze. Eine Praxisorientierte Einführung.; Expert Verlag, Renningen-Malmsheim, 1998
- [6] Müller, Johann-Adolf/Lemke, Frank: Self-Organising Data Mining. Extracting Knowledge From Data.;

Quellenverzeichnis

- [1] SOMFolien.pdf, Prof. Dr. rer. nat. habil. H. Iwe
- [2] <http://members.ozemail.com.au/~dekker/NeuQuant.pdf>
- [3] <http://de.wikipedia.org/wiki/Quantisierung>
- [4] www.uni-koblenz.de/~cg/veranst/ws0001/sem/Licker-neu.pdf