Interpolation handgeschriebener Ziffern

Bernard Jaquet

13. Januar 2019

Zusammenfassung

Diese Arbeit versucht die Entscheidungsfindung eines einfachen Feed Forward Neuronal Network zu visualisieren. Dafür wurden zwei Ansätze verwendet. Im ersten wird versucht, das Neuronale Netzwerk zu invertieren und verschiedene Ziffern oder Interpolationen von Ziffern rückwärts durch das Netzwerk zu rechnen, um Bilder zu erhalten. Dies erwies sich aus verschiedenen Gründen als Unmöglich. Nicht-quadratische Matrizen lassen sich nicht invertieren und Rundungsfehler aus Gleitkommazahlen werden durch die Aktivierungsfunktion amplifiziert. Daher wurden in einem zweiten Ansatz Bilder durch die Reduzierung des Fehlers im Neuronalen Netz an ein gewünschtes Resultat angenähert. Bei dieser Methode entstand anscheinend zufälliges Rauschen in den angenäherten Bildern. Dies lässt darauf schliessen, dass das Netzwerk kein gutes Konzept für Ziffern erlernt hat.

1 Einleitung

Betrachtet man die Funktionsweise von Künstlichen Neuronalen Netzen, erkennt man eine einfache Serie von Matrixmultiplikationen und Vektoradditionen, welche einen Punkt im Problemraum in einen Punkt im Zielraum transformieren. Dies ist in der Abbildung 1 zu sehen. Um das Netzwerk lernen zu lassen, müssen die Werte in diesen Matrizen und Vektoren so angepasst werden, dass diese die gestellte Aufgabe besser lösen. Schwierig hingegen ist das Interpretieren des Gelernten. Weshalb stehet in der ersten Gewichtsmatrix in der zweiten Zeile und fünften Spalte genau dieser Wert. Ist dieser Wert korrekt?

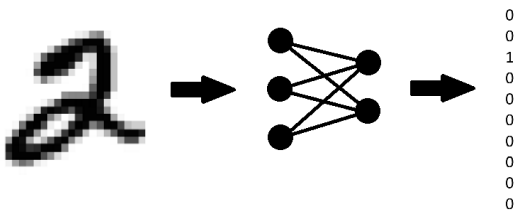


Abbildung 1: Standard Prozess eines Neuronalen Netzes

Diese Arbeit befasst sich mit dieser Problematik und versucht gewisse "Gedankengänge" eines Künstlichen Neuronalen Netzes zu visualisieren. Analysiert werden die Vorgänge in einem Deep Feed-Forward Neuronal Network, welches zur Erkennung handgeschriebenen Ziffern aus dem MNIST Datensatz trainiert wurde. Für die Visualisierung wird das Inverse des Neuronalen Netzes berechnet und gezielt mit Daten gespiesen um Bilder zu produzieren. Dieser Prozess wurde in der Abbildung 2 visualisiert.

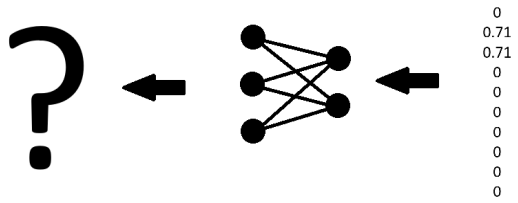


Abbildung 2: Intertierung des Neuronalen Netzes

Die Ausgabe des trainierten Neuronalen Netzes ist ein Vektor der Länge zehn. Jeder Wert in diesem Vektor gibt an, wie sicher sich das Netzwerkt ist, diese Ziffer im bewerteten Bild entdeckt zu haben. Der Index des Wertes gibt Aufschluss über die Ziffer. So wird der Index mit der höchsten Wahrscheinlichkeit als Resultat gewertet.

Als Input für das invertierte Netzwerk sollen zwei verschiedene Vektortypen verwendet werden. Dies sollen einerseits "perfekte" Ziffervektoren sein, welche eine Eins am Index der Ziffern haben und den Rest mit Nullen besetzt ist. Andererseits sollen Vektoren von interpolierten Ziffern verwendet werden, welche eine Mischung aus zwei verschiedenen Ziffern darstellen. Diese Interpolation soll durch eine Rotation zwischen den beiden idealen Ziffervektoren realisiert werden. Dabei kann der Winkel der Rotation als Parameter für das Mischverhältnis der beiden Ziffern betrachtet werden.

Somit zur Fragestellung dieser Arbeit: Lassen sich die ursprünglichen Ziffern der Interpolation nach der Rückrechnung in den Problemraum erkennen?

2 Material und Methoden

2.1 Aufbereitung der Daten

Als Datensatz wurde der MNIST Datensatz von der offiziellen Webseite[1] von Yann LeCun verwendet. Dieser umfasst ein Trainingsset von 60 000 und ein Testset von 10 000 klassifizierten Bildern mit handgeschriebenen Ziffern. Diese sind in zwei verschiedenen Dateien aufgeteilt, eine für die Bilder und eine für die Labels. Von der Bilder-Datei werden jeweils 28x28 Bytes eingelesen, welche ein Bild repräsentieren und in einen 784-dimensionalen Vektor geschrieben. Dabei werden die 255 Graustufenwerte auf Gleitkommazahlen zwischen Null und Eins verwandelt. Die Label-Datei wird Byteweise eingelesen und in eine One-Hot Vektor der Länge zehn verwandelt, bei welchem der Index der Eins dem Wert des Labels entspricht.

2.2 Das Modell erstellen und trainieren

Das Neuronale Netz wurde mit Tensorflow[2] modelliert und trainiert. Das verwendete Modell ist ein Feed Forward Neuronal Network mit Weights und Biases für jede Schicht. Um den Output des Netzes zu berechnen, muss $\vec{x_n}$ berechnet werden. Dabei is $\vec{x_0}$ der Bildvektor. Die restlichen Werte werden gemäss Gleichung 1 berechnet. n entspricht dabei der Anzahl Schichten im Neuronalen Netz.

$$\vec{x_i} = a(\vec{x_{i-1}} \times w_{i-1} + b_{i-1}) \tag{1}$$

Als Aktivierungsfunktion $a(x)$ wird die Logistische Sigmoidfunktion verwendet, welche der Gleichung 2 zu entnehmen ist. Sie wird auf alle Elemente im Vektor $\vec{x}$ angewendet.

$$a(x) = \frac{1}{1 + e^{-x}} \tag{2}$$

Sie hat die Aufgabe zu verhindern, dass Zahlen durch die Matrixmultiplikationen zu gross im positiven oder negativen Raum werden. Dies erreicht die Funktion in dem sie den Definitionsbereich von $[-\infty, \infty]$ auf den Wertebereich $[0, 1]$ einschränkt. Dieses Verhalten wurde in der Abbildung 3 visualisiert. So ist die Eingabe für jede Schicht im Netzwerk ein Vektor mit Werten zwischen Null und Eins, welcher der Stärke der Aktivierung jedes Neurons in dieser Schicht entspricht.

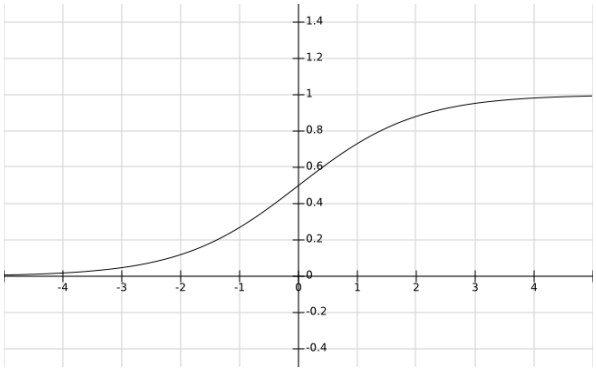


Abbildung 3: Verhalten der Sigmoid Funktion

Um das Neuronale Netzwerk zu trainieren muss der Fehler der Klassifikation berechnet werden. Dieser wird mit der Kreuzentropie zwischen dem normalisierten Output-Vektor $\vec{x}$ des Netzwerks und dem Label-Vektor $\vec{y}$ berechnet. Die Formel für die Kreuzentropie ist in Gleichung 3 zu sehen.

$$H(\vec{x}, \vec{y}) = -\sum_{n}^{k} x_k log(y_i) \tag{3}$$

Die Normalisierung des Vektors wurde mit der Softmax-Funktion aus der Gleichung 4 realisiert. Dies hat zur Folge, dass alle Werte des Vektors im Bereich $[0, 1]$ liegen und die Länge des Vektors eins beträgt. Durch diese Eigenschaften kann der Vektor als diskrete Wahrscheinlichkeitsverteilung für das Erkennen der Ziffer im Input-Vektor betrachtet werden.

$$\sigma(v)_i = \frac{e^{v_i}}{\sum_{n}^{k=1} e^{v_k}} \tag{4}$$

Für die Korrektur des Fehlers im Neuronalen Netz wird ein Gradientenverfahren eingesetzt, welches den über die Kreuzentropie berechneten Fehler minimiert. In diesem Verfahren wird die Ableitung des Modells bestimmt und jeder Parameter um einen bestimmten Betrag in die Richtung des steilsten Abfalls korrigiert. In diesem Modell sind die Weight-Matrizen und Bias-Vektoren die Parameter. Der Korrekturbetrag ergibt sich aus der Steigung am Punkt der Funktion und dem Hyperparameter der Learning Rate.

$$b = a - \gamma \Delta f(a) \tag{5}$$

Das grundlegende Konzept des Gradientenverfahrens kann der Gleichung 5 entnommen werden. Der neue Wert des Parameters b ergibt sich aus dem Lerning Rate γ multipliziert mit der Richtung des steilsten Abfalls $\Delta f(a)$ subtrahiert vom ursprünglichen Wert a. Dieses Verfahren wurde in der Abbildung 4 visualisiert.

[1] LeCun, Cortes und Burges, *THE MNIST DATABASE*
[2] Abadi u. a., *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*

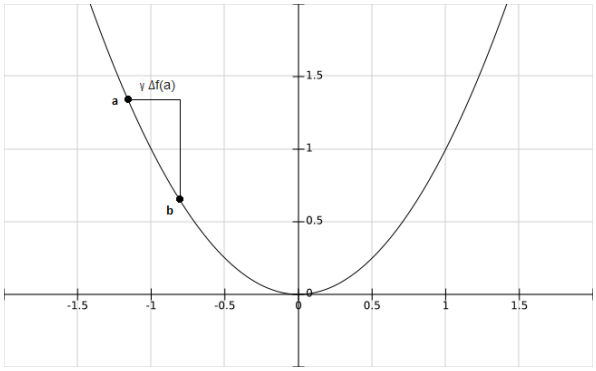


Abbildung 4: Visualisierug Gradientenverfahren

2.3 Invertierung des Neuronalen Netzes

Um die Funktion des Neuronalen Netzes zu invertieren wurde in einem ersten Schritt eine komprimierte Version des Netzes erstellt. Diese Komprimierung bestand darin die Weight-Matrizen und Bias-Vektoren in den homogenen Raum zu transformieren. Dadurch kann die Addition des Bias-Vektors und die Multiplikation der Weight-Matrix wie in Gleichung 6 beschrieben in eine Matrix pro Layer zusammengefasst werden.

$$M_i = \begin{bmatrix} I & \vec{b_i} \\ \vec{0}^T & 1 \end{bmatrix} \times \begin{bmatrix} w_i & \vec{0} \\ \vec{0}^T & 1 \end{bmatrix} \tag{6}$$

Als nächster Schritt wurde versucht die Matrizen aller Layers in eine Matrix zusammenzuführen. Diese könnte später mit dem Inputvektor multipliziert werden und so alle Schritte im Neuronalen Netz auf eine Matrixmultiplikation reduzieren. Dies wurde gemäss der Gleichung 7 gemacht. Dabei wurde angenommen, dass die Aktivierungsfunktion auf das Produkt der Matrixmultiplikation angewendet werden könnte, da deren Funktion das Einschränken des Wertebereichs ist. Diese Annahme erwies sich jedoch als falsch, was im Kapitel 3.2 aufgegriffen wird. Aus diesem Grund wurde eine zweite Methode verwendet, bei welcher die Aktivierungsfunktion in der Komprimierung weggelassen wurde und nur auf das Resultat der Matrixmultiplikation mit dem Inputvektor angewendet. Auch diese Methode ergab keine gültige Komprimierung.

$$N_0 = M_0, N_i = a(M_i \times N_{i-1}) \tag{7}$$

Da die Komprimierung des gesamten Netzes scheiterte, wurde nur die Komprimierung der einzelnen Layers beibehalten. Um das Netz zu invertieren wurde das Inverse aller Layer-Matrizen M_i berechnet. Da nicht alle Matrizen mit beliebigen Dimensionen eine echtes Inverse besitzen, wird die inverse Matrix mit Pseudoinverse approximiert. Gemäss der Dokumentation der verwendeten Library[3] wird dies über die KQ-Methode erreicht. Das Inverse des Neuronalen Netzes ist so durch die Gleichung 8 gegeben.

$$\vec{x_i} = a(M_{i-1}^{+} \times \vec{x_{i-1}})^{-1} \tag{8}$$

Dabei ist $a(x)^{-1}$ das Inverse der Aktiverungsfunktion und definiert durch die Gleichung 9.

$$a(x)^{-1} = -log_e(\frac{1}{x} - 1) \tag{9}$$

Da das Pseudoinverse nicht garantiert ein echtes Invers ist, könnte dies einen Einfluss auf die berechneten Bilder haben. Aus diesem Grund wurde ein weiteres Neuronales Netz trainiert, welches sich desselben Modells bedient, jedoch die nur quadratische Matrizen verwendet. So konnte sichergestellt werden, dass alle Matrizen M_i im komprimierten Netz ein echtes Invers haben.

2.4 Approximation von Input Bildern

Bei der Evaluation des invertieren Neuronalen Netzes konnte eine mögliche Fehlerquelle für verfälschte Resultate ausfindig gemacht werden: Gleitkommazahlen verursachen in Computer Rundungsfehler. Diese sind in der Regel insignifikant, werden aber durch die inverse Sigmoidfunktion verstärkt und in einen signifikanten Bereich gebracht. Wie stark sich diese Rundungsfehler auf die Resultate auswirkt wurde im Rahmen dieser Arbeit ebenfalls untersucht und die Resultate sind im Kapitel 3.5 einzusehen.

Um diese Fehlerquelle zu umgehen wurde ein weiteres Neuronales Netz trainiert. Dieses Neuronale Netz verwendet das gleiche Modell wie das ursprüngliche Netz. Der Unterscheid besteht darin, dass die Weigth-Matrizen und Bias-Vektoren als Konstanten definiert und als variabler Teil für die Optimierung der Input gewählt wurde. In diesem Modell versucht das Gradientenverfahren ein zufälliges Inputbild so zu verändern, dass dieses möglichst genau dem gesuchten Vektor im Zielraum entspricht.

3 Ergebnisse

3.1 Erkennen von Ziffern

Das Neuronale Netz zum Erkennen der Ziffern erreicht nach 100 Trainingsiterationen im Normalfall eine Genauigkeit von circa 88%, wie in der Abbildung 5 zu erkennen ist. Es gibt einige Ausreisser, welche schlechtere Werte haben, die sind aber die Ausnahme. Auch lassen sich in den Konfusionsmatrizen keine grossen Fehler erkennen. Erhöhte Fehlerwerte lassen sich lediglich bei ähnlichen Ziffern

[3] Jones, Oliphant und Peterson, *SciPy: Open source scientific tools for Python*

(wie 3 & 5 oder 4 & 9) erkennen. Alle Konfusionsmatrizen sind im Anhang im Kapitel A.2 zu finden.

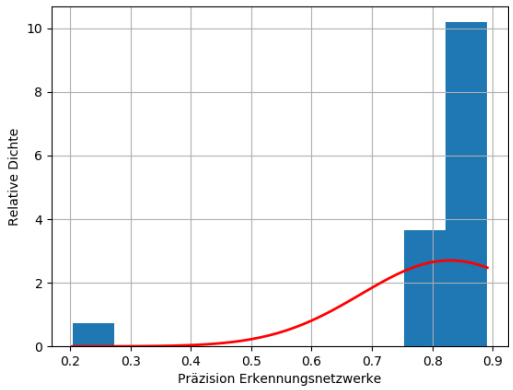


Abbildung 5: Präzision des Neuronalen Neztes

3.2 Netzwerk Komprimierung

Die Komprimierung des Netzwerkes gemäss Kapitel 2.3 in eine Matrix führt zu einem starken Qualitätsverlust der Vorhersagen. Der grosse Teil der komprimierten Netze erreicht eine Trefferquote von 30%, wie in Abbildung 6 zu erkennen ist. Dies sind die Resultate der Neuronalen Netze aus dem Kapitel 3.1. Es sind Ausreisser zu erkennen, deren Werte liegen aber mit einer Trefferquote von maximal 45% weit entfernt von den Werten der unkomprimierten Netze.

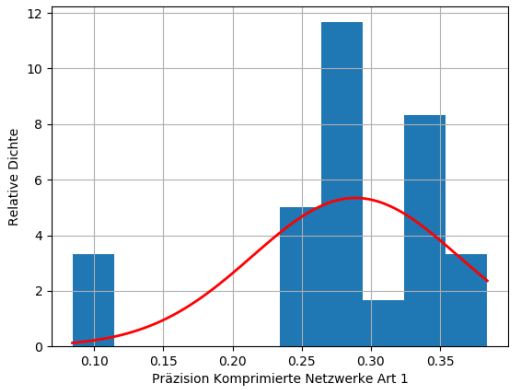


Abbildung 6: Präzision des komprimierten Neuronalen Neztes Methode 1

In der zweiten Methode der Komprimierung, bei welcher die Aktivierungsfunktion nur auf das Resultat der Vorhersage anwendet, ist ebenfalls eine Verschlechterung der Werte zu erkennen. Diese ist sogar noch stärker. So verteilen sich die Werte gemäss Abbildung 7 zwischen 10% und 20%. Es gilt zu beachten, dass die Verteilung der Resultate durch die Untergrenze von 10% begrenzt ist. Dieser Wert ist für eine zufällige Klassifizierung zu erwarten. Somit bietet der grösste Teil der Netze, welche auf diese Art komprimiert wurden, keinen Mehrwert.

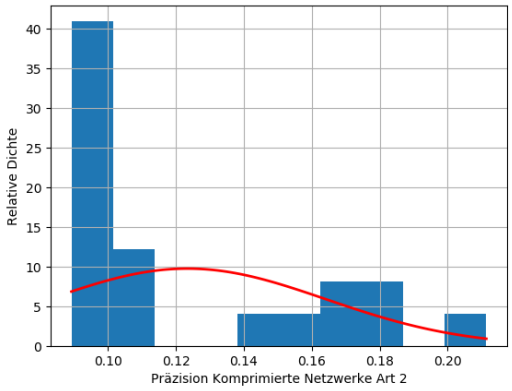


Abbildung 7: Präzision des komprimierten Neuronalen Neztes Methode 2

Die Invertierung mit mehreren Matrizen erreicht die selben Werte wie die unkomprimierten Versionen. Dies ist in der Abbildung 8 zu erkennen. Aus diesem Grund wurde diese Art der Komprimierung für das Generieren der Bilder im weiteren Verlauf der Arbeit verwendet.

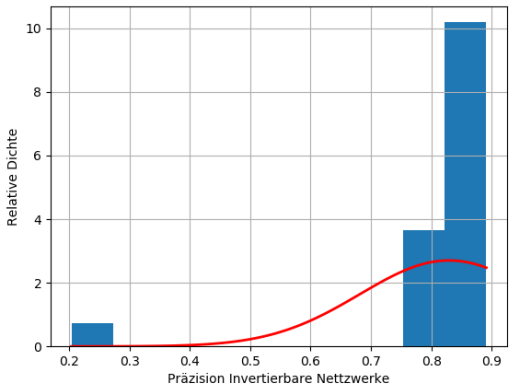


Abbildung 8: Präzision des komprimierten Neuronalen Neztes mehrere Matrizen

3.3 Invertierung mit Pseudoinvers

Auf den Bildern, welche durch die Invertierung des Neuronalen Netzes mit Hilfe des Pseudoinversen entstanden, ist nur Rauschen zu erkennen. In der Abbildung 9 ist das errechnete Bild für den Vektor $\begin{bmatrix}1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0\end{bmatrix}^T$ zu sehen. Dies würde einer idealen Null entsprechen. Auf den Resultaten von anderen Ziffern lassen sich ebenfalls keine Ziffern erkennen. Auch sind von Auge keine Muster zwischen den Bildern zu erkennen. Weitere Bilder zum Vergleich sind im Anhang im Kapitel A.4 zu finden.

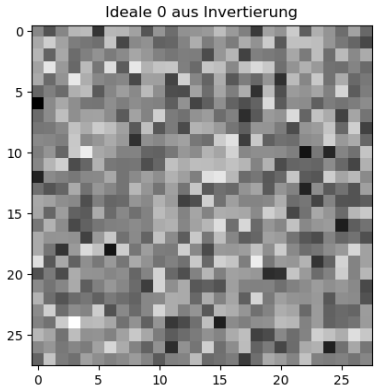


Abbildung 9: Ideale 0 aus Invertierung

Ähnliche Resultate sind auch bei den interpolierten Ziffern zu sehen. In der Abbildung 10 ist eine Interpolation zwischen einer idealen Vier und einer idealen Neun abgebildet. Der Winkel für die Interpolation wurde so gewählt, dass sich der Vektor genau zwischen den beiden Werten befindet. Wie bereits bei den idealen Ziffervektoren lassen sich im Bild keine Ähnlichkeiten mit den im Eingabevektor definieren Ziffern erkennen.

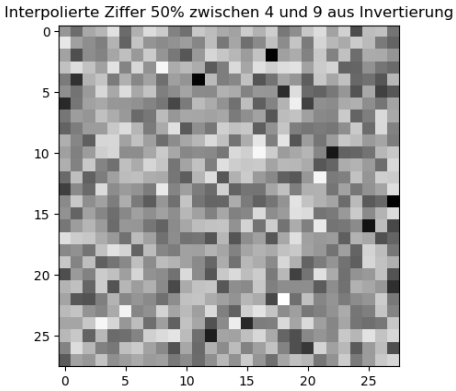


Abbildung 10: Interpolation von 4 und 9 durch Invertierung

3.4 Invertierung mit quadratischen Matrizen

Das Trainieren eines Neuronalen Netzes, welches nur quadratische Matrizen verwendet, erwies sich als äussert schwierig. Die Trainingszeiten wuchsen stark an und die Resultate waren deutlich schlechter als die der Netze ohne quadratischen Matrizen. Der Abbildung 11 kann entnommen werden, dass die meisten Netzwerke mit quadratischen Matrizen Trefferquoten von ungefähr 12% bis 16% erreichen. Es gilt auch festzuhalten, dass ein grosser Teil der trainierten Netze lediglich 10% der Bilder korrekt klassifiziert, was einer zufälligen Klassifikation entspricht.

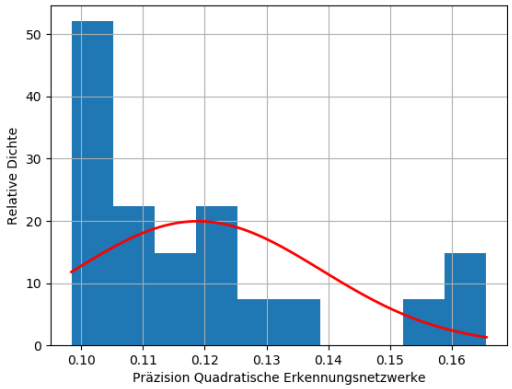


Abbildung 11: Präzision des Neuronalen Neztes mit quadratischen Matrizen

Wegen der Schwierigkeit Neuronale Netze mit quadratischen Matrizen zu trainieren, musste bestimmt werden, ob dieser Ansatz für diesen Bericht weiterverfolgt wird. Daher wurde Untersucht, ob eine Invertierung eines Neuronalen Netzes mit ausschliesslich quadratischen Matrizen realisierbar ist. Die Resultate dazu sind im Kapitel 3.5 zu finden.

3.5 Fehler bei Gleitkommazahloperationen

Um zu bestimmen, wie sich die Rundungsfehler der Gleitkommazahlen auf Invertierung des Neuronalen Netzwerks auswirken, wurden verschiedene zufällig generierte Matrizen vor- und rückwärts durch das Neuronalen Netz gerechnet und der durchschnittliche Fehler pro Wert in der Matrix bestimmt. Die Matrizen wurden so gewählt, dass $M \in \mathbb{R}^{n \times n}, n \in [3, 9]$ und im Neuronalen Netz aus einer bis vier Schichten besteht. Die Resultate sind der Tabelle 1 zu entnehmen.Daraus lässt sich erkennen, dass bei einer Matrix aus $\mathbb{R}^{9\times9}$ bereits ein Netzwerk mit einer Schicht die Werte nach der Invertierung im Schnitt im Bereich von 10^0 verändert. Bei vier Schichten wird dieser Wert bereits bei einer Matrix aus $\mathbb{R}^{6\times6}$ überschritten.

	1 Layer	2 Layers	3 Layers	4 Layers
$\mathbb{R}^{3\times3}$	8.2e-15	3.8e-12	1.5e-08	1.6e-09
$\mathbb{R}^{4\times4}$	1.6e-13	3.3e-10	3.35e-05	1.7e-04
$\mathbb{R}^{5\times5}$	1.9e-10	5.0e-08	2.3e-06	6.1e-03
$\mathbb{R}^{6\times6}$	1.0e-02	1.6e-02	3.4e-02	4.7e+00
$\mathbb{R}^{7\times7}$	9.7e-02	2.3e-01	2.5e+00	1.3e+01
$\mathbb{R}^{8\times8}$	4.5e-01	2.3e-01	6.2e+00	1.7e+01
$\mathbb{R}^{9\times9}$	2.1e+00	1.6e+00	7.3e+00	1.9e+01

Tabelle 1: Durchschnittlicher Fehler pro Wert nach Invertierung

3.6 Approximieren der Inputbilder

Wegen der signifikanten Fehler aus der Rundung der Gleitkommazahlen im Ansatz der Invertierung wurde der Ansatz der Approximation von Bildern weiterverfolgt. Bei diesem Ansatz wurde als Qualitätsmass der Fehler aus der Kreuzentropie zwischen dem erreichten Vektor und dem gewünschten Vektor für das zu generieren Bild verwendet. Hier lassen sich deutliche Unterschiede für die Approximation der idealen Ziffern und der interpolierten Ziffern erkennen. In der Abbildung 12 ist zu erkennen, dass für die idealen Ziffern der durchschnittliche Fehler bei 1.5 liegt, wobei es auch einige Ausreisser bei über 2.2 gibt.

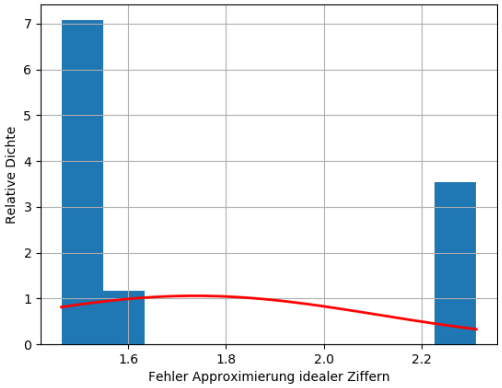


Abbildung 12: Fehler bei Approximation idealer Ziffern

In der Abbildung 13 ist zu erkennen, dass der Fehler der interpolierten Ziffern höher liegt als der der idealen Ziffern. Ebenfalls lässt sich erkennen, dass der Median in der Nähe von 2.2 liegt, was an den Fehlerwert der Ausreisser bei den idealen Ziffern erinnert.

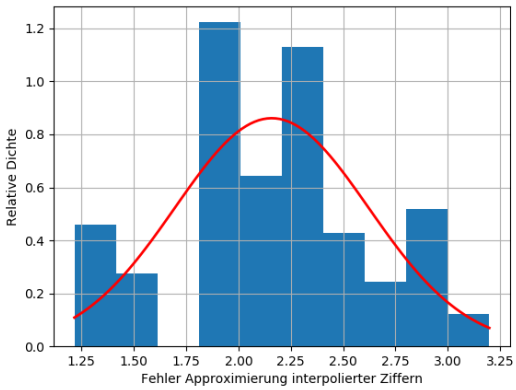


Abbildung 13: Fehler bei Approximation interpolierter Ziffern

Die generierten Bilder enthalten auch hier nur Rauschen. Auch bei den tiefen Fehlerwerten der idealen Ziffern ist keine Struktur zu erkennen. In der Abbildung 14 ist das approximierte Bild für eine ideale 7 zu sehen.

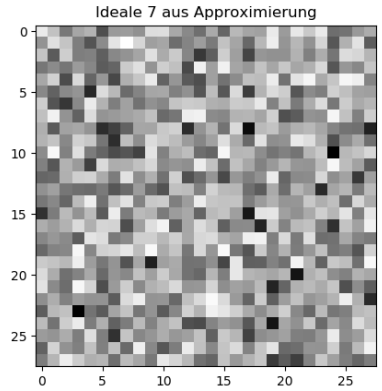


Abbildung 14: Ideale 7 durch Approximierung

In einem letzten Versuch wurden für die Approximation anstelle von zufällig generierten Matrizen als Startpunkt Bilder aus dem Trainingssatz verwendet. Dies führte dazu, dass die Approximation die Inputmatrix nicht veränderte, auch wenn eine andere Ziffer als Ziel der Approximation angegeben wurde.

3.7 Manuelle Untersuchung der Klassifizierung

Da die vorgängig verwendeten Methoden zur Untersuchung des Neuronalen Netzes grösstenteils zufälliges Rauschen produzierten, welche kaum eine sinnvolle Interpretation zulassen, wurde für eine letzte Analyse ein nicht-generativer Ansatz verwendet. Anstatt zu versuchen neue Bilder zu generieren, wurden Bilder aus dem Test-Datensatz gesucht, welche für eine Analyse interessant sein könnten. Gesucht wurde nach minimalen Fehlerwerten, maximale Fehlerwerten mit korrekter Klassifizierung und minimale Fehlerwerten mit falscher Klassifizierung. So sollte ein Set von gut erkennbaren Ziffern und ein Set mit für das Netzwerk grenzwertigen Ziffern für die Analyse gewonnen werden. Die Hoffnung war hierbei manuell Merkmale zu erkennen, welche dem Neuronalen Netz bei der Klassifizierung helfen oder stören. Dabei wurde erwartet, dass schön geschriebene Ziffern generell tiefe Fehlerwerte bei der Klassifizierung erhalten und Ziffern mit verschiedenen Schreibvarianten schlechter abschneiden. Alle Resultate sind im Kapitel A.5 zu finden.

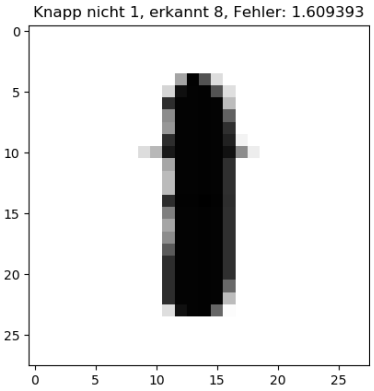


Abbildung 15: Eine 1 mit dicker Line erkannt als 8

Effektiv lassen sich gewisse Muster erkennen. Das Netz scheint Mühe mit dicken Linien zu haben. Ein Beispiel dafür ist in der Abbildung 15 zu sehen, in welcher eine Eins als Acht klassifiziert wird. Aber auch dünner als durchschnittliche Linien führen zu falschen Klassifizierungen. Eine weitere Schwierigkeit für das Neuronale Netz sind offenbar nicht durchgezogene Linien. Die Ziffer in der Abbildung 16 ist effektiv nicht leicht zu klassifizieren, aber eine Vier ist das bestimmt nicht.

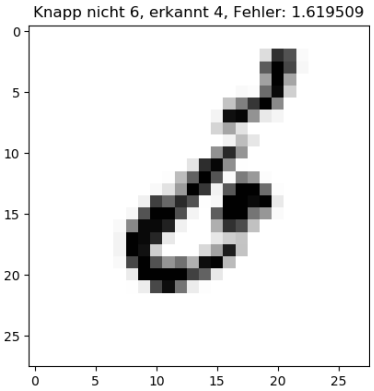


Abbildung 16: Eine 6 mit unterbrochener Line erkannt als 4

Bei den Klassifizierungen mit tiefen Fehlerwerten lassen sich ebenfalls Muster erkennen. Zum ersten könnten fast alle Ziffern aus einem Lehrbuch zum Schreiben lernen aus der ersten Klasse entnommen worden sein. Weiter ist ebenfalls zu erkennen, dass auch unterschiedliche Schreibweisen für Ziffern kaum Einfluss auf die Klassifizierung hat: Sowohl eine Sieben mit also auch eine ohne Querstrich erkennt das Netzwerk mit guter Zuverlässigkeit, wie in Abbildung 17 zu erkennen ist.

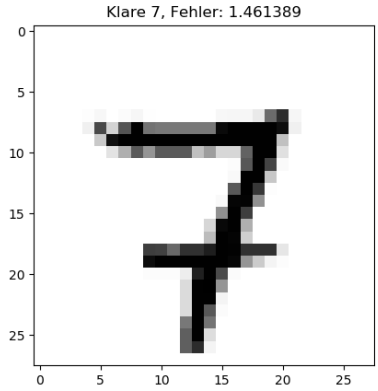


Abbildung 17: Eine 7 mit Variante Querstrich

4 Diskussion

4.1 Interpretation Resultate

Aus den Resultaten lassen sich verschiedene Schlüsse ziehen. Zum Ersten zeigen die Resultate im Kapitel 3.1, dass das gewählte Modell für das Neuronale Netzwerk einen Trainingsprozess hat, der nicht immer zu einem verwertbaren Resultat führt. Eine mögliche Erklärung dafür könnte der Verlust der Nachbarschaftsbeziehung von Pixeln durch die Umformung des Bildes in einen Vektor sein. Das Neuronale Netz muss diese Beziehung zuerst erlernen.

Weiter haben die Resultate im Kapitel 3.2 gezeigt, dass die Komprimierung von Neuronalen Netzen in eine Matrix schwer zu realisieren ist. Der Grund dafür ist wahrscheinlich die nichtlineare Aktivierungsfunktion, welche ein wichtiger Bestandteil der Funktion des Neurons ist. Diese Nicht-Linearität verhindert, dass die einzelnen Schichten in Netz zu einer Matrix zusammengefasst werden können[4]. Das Verwenden einer linearen Funktion als Aktivierung hätte einen negativen Einfluss auf die Leistungsfähigkeit des Neurons.

Eine weitere Schwierigkeit zeigen die Resultate im Kapitel 3.5 auf. Die verwendete Aktivierungsfunktion hat die Eigenschaft, alle Zahlen $\in \mathbb{R}$ auf den Bereich $[0, 1]$ einzuschränken. Dadurch amplifiziert das Inverse der Funktion aber auch kleine Änderungen, welche durch Rundungsfehler entstehen können, und lässt diese signifikant werden. Bei einer gewissen Anzahl Gleitkommazahloperationen sind solche Rundungsfehler unvermeidbar, und durch das Verwenden von Matrixmultiplikationen werden solche Fehler durch die ganze Matrix propagiert. Dies verunmöglicht das einfache zurückrechnen durch das Neuronalen Netzwerk.

Um trotz der Rundungsfehler der Gleitkommazahlen Ziffern im Zielraum zu interpolieren, wurde der Ansatz der Approximation des Inputbildes ge-

[4] Standford, *CS231n Convolutional Neural Networks for Visual Recognition*

wählt. Im Kapitel 3.6 ist zu sehen, dass die Fehler der Kreuzentropie bei der Approximation von idealen Ziffern mit bis zu 1.5 relativ vielversprechend ist. Jedoch lassen sich in den generierten Bildern keine Ziffern erkennen. Bei den interpolierten Ziffern ist der Fehler zu hoch, um gute Ergebnisse zu erwarten.

4.2 Weiterführende Arbeiten

Viele Fehler sind vermutlich auf den Verlust der Nachbarschaftsbeziehung der einzelnen Pixel zurückzuführen. Diese Problematik könnte sich mit der Verwendung eines Convolutional Neural Networks entschärfen lassen. Solange nur Faltungen verwendet würden, könnten diese durch eine Fourier Transformation invertiert werden[5]. Sollte am Schluss noch ein Fully Connected Layer verwendet werden, müsste sichergestellt werden, dass dieser klein genug ist um die Wahrscheinlichkeit für einen signifikanten Gleitkommafehler gering zu halten. Ein anderer Ansatz wäre die Pixel komplett loszuwerden und die Ziffer als Linie mathematisch zu beschreiben. Eine Möglichkeit dafür wären die "Snakes: Active contour models"[6]. So wäre das Resultat bei einer Invertierung auch garantiert eine geometrische Form.

4.3 Fazit

Die Frage, ob Vorgänge innerhalb eines Neuronalen Netzes mithilfe von Rückrechnung durch ein Neuronale Netz visualisieren lassen, kann diese Arbeit nicht vollständig klären. Gezeigt hat sich, dass eine Invertierung eines Feed Forward Neuronalen Netzes sich nicht für eine Visualisierung eignet, wenn das Netzwerk etwas grösser wird. Die Resultate der Approximation legen jedoch nahe, dass ein Feed Forward Neuronales Netzwerk kein gutes Verständnis von Ziffern aus reinen Pixeln erlernt hat. Dies wird weiter durch den letzten Versuch dieser Arbeit, für den Start der Approximation Bilder aus dem Trainingsset zu verwenden, unterstützt. Bei diesem Versuch konnte festgestellt werden, dass keine Veränderung am Inputbild vorgenommen wird, auch wenn eine andere Ziffer als Zielvektor angegeben wurde. So scheint es, als fände das trainierte Netz ein lokales Maximum, welches gut genug ist, um die meisten Ziffern zu erkennen. Von einem Verständnis für Ziffern kann daher nicht die Rede sein.

Literatur

Abadi, Martín u. a. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: `https://www.tensorflow.org/` (besucht am 15. 11. 2018).

Jones, Eric, Travis Oliphant, Pearu Peterson u. a. *SciPy: Open source scientific tools for Python*. 2001–. URL: `"https://docs.scipy.org/doc/numpy-1.15.1/reference/generated/numpy.linalg.pinv.html"` (besucht am 30. 11. 2018).

Kass, Michael, Andrew Witkin und Demetri Terzopoulos. "Snakes: Active contour models". In: *INTERNATIONAL JOURNAL OF COMPUTER VISION* 1.4 (1988), S. 321–331.

LeCun, Yann, Corinna Cortes und Christopher J.C. Burges. *THE MNIST DATABASE*. 1998. URL: `"http://yann.lecun.com/exdb/mnist/"` (besucht am 12. 11. 2018).

Nielson, Michael A. *Neuronal Networks and Deep Learning*. 2015.

Standford, CS231N Staff. *CS231n Convolutional Neural Networks for Visual Recognition*. 2018. URL: `"http://cs231n.github.io/neural-networks-1/"` (besucht am 08. 12. 2018).

Weisstein, Eric W. *Convolution Theorem*. 2018. URL: `"http://mathworld.wolfram.com/ConvolutionTheorem.html"` (besucht am 08. 12. 2018).

[5] Weisstein, *Convolution Theorem*

[6] Kass, Witkin und Terzopoulos, "Snakes: Active contour models"

A Anhang: Resultate

A.1 Histograme Leistung

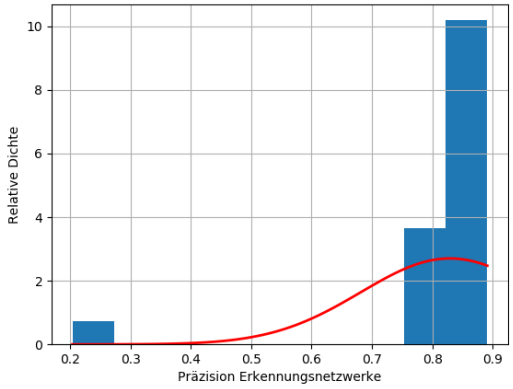


Abbildung 18: Präzision der Klassifizierungsnetze

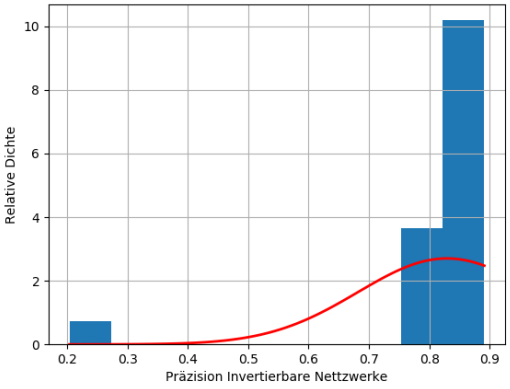


Abbildung 21: Präzision Invertierbares Netz

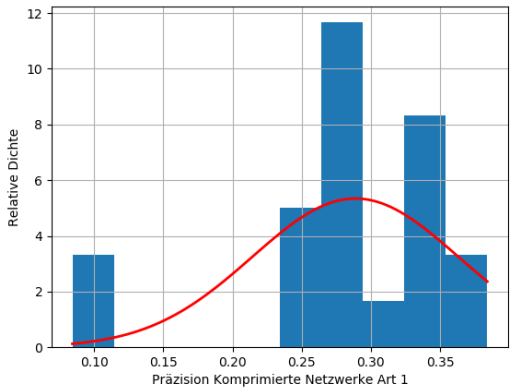


Abbildung 19: Präzision mit Komprimierung Art 1

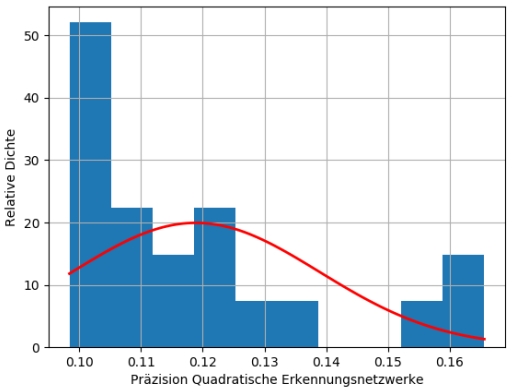


Abbildung 22: Präzision Kassifikation Quadratische Matrizen

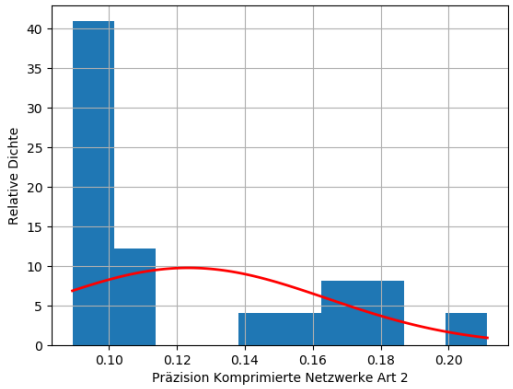


Abbildung 20: Präzision mit Komprimierung Art 2

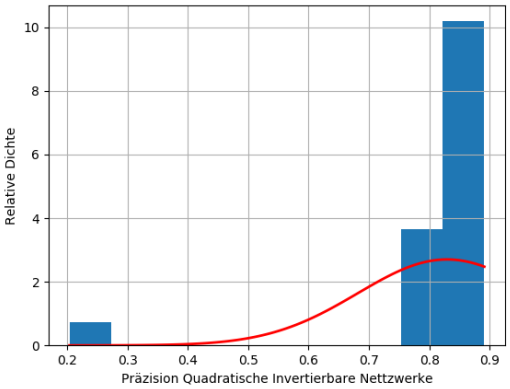


Abbildung 23: Präzision Invertierung Quadratische Matrizen

A.2 Konfusionsmartizen

A.2.1 Klassifizierungs Netz

	0	1	2	3	4	5	6	7	8	9
0	941	0	6	3	3	7	10	4	6	0
1	0	1104	5	2	2	4	6	1	10	1
2	13	3	892	20	17	1	17	20	46	3
3	4	1	27	844	3	61	6	15	37	12
4	1	1	13	0	883	0	19	4	12	49
5	21	4	6	53	9	718	19	5	45	12
6	19	5	20	4	25	22	858	2	3	0
7	4	14	34	9	14	1	3	890	9	50
8	6	6	14	39	15	35	17	9	815	18
9	10	5	12	11	59	15	2	41	9	845

Tabelle 2: Konfusionsmatrix Netzwerk 0

	0	1	2	3	4	5	6	7	8	9
0	932	1	9	4	2	6	12	5	8	1
1	0	1103	3	3	1	4	5	2	12	2
2	17	2	875	31	13	4	21	16	44	9
3	3	4	37	826	4	79	1	13	35	8
4	3	3	12	0	843	9	16	4	11	81
5	22	2	4	71	10	715	17	9	33	9
6	15	2	11	1	17	17	887	1	7	0
7	6	13	31	12	15	1	3	884	12	51
8	12	9	14	26	8	27	27	13	819	19
9	17	5	5	10	69	11	1	27	22	842

Tabelle 3: Konfusionsmatrix Netzwerk 1

	0	1	2	3	4	5	6	7	8	9
0	935	0	6	2	1	17	8	3	8	0
1	0	1105	7	6	0	3	3	0	11	0
2	14	4	890	29	15	9	12	16	41	2
3	7	1	23	888	1	34	4	16	30	6
4	2	5	8	0	869	1	21	6	9	61
5	21	3	8	75	17	703	17	5	38	5
6	12	3	18	2	21	19	876	1	5	1
7	5	18	26	8	15	0	1	903	5	47
8	6	9	19	40	10	38	12	12	813	15
9	9	6	9	13	51	7	2	33	20	859

Tabelle 4: Konfusionsmatrix Netzwerk 2

	0	1	2	3	4	5	6	7	8	9
0	936	1	6	1	1	19	8	4	3	1
1	0	1101	4	1	0	2	4	3	18	2
2	16	3	892	24	23	1	23	16	30	4
3	10	3	24	861	6	47	3	18	32	6
4	1	4	10	0	870	4	18	3	9	63
5	15	3	6	66	13	724	18	7	30	10
6	17	2	12	2	13	18	883	3	8	0
7	7	7	37	6	10	2	3	920	3	33
8	13	7	12	28	15	32	13	9	829	16
9	14	2	5	17	73	12	3	42	13	828

Tabelle 5: Konfusionsmatrix Netzwerk 3

	0	1	2	3	4	5	6	7	8	9
0	0	1	154	89	7	366	187	39	114	23
1	0	1096	7	4	1	3	2	2	19	1
2	0	2	898	22	21	4	23	18	32	12
3	0	2	33	870	0	53	3	16	28	5
4	0	3	7	0	888	0	15	3	11	55
5	0	2	5	56	14	742	19	8	39	7
6	0	3	9	3	17	23	895	2	5	1
7	0	13	30	6	12	2	1	918	7	39
8	0	6	12	30	10	40	23	19	814	20
9	0	5	8	14	81	12	3	38	20	828

Tabelle 6: Konfusionsmatrix Netzwerk 4

	0	1	2	3	4	5	6	7	8	9
0	930	0	7	4	1	12	17	2	5	2
1	0	1104	3	5	1	1	6	1	13	1
2	18	4	885	24	16	3	22	20	35	5
3	8	1	27	854	4	55	4	18	33	6
4	1	3	3	3	876	2	20	7	11	56
5	19	2	6	67	15	705	20	8	42	8
6	15	3	21	5	13	20	879	0	2	0
7	3	16	28	13	11	7	1	906	2	41
8	12	9	15	39	14	37	10	9	817	12
9	9	5	3	18	61	14	2	33	7	857

Tabelle 7: Konfusionsmatrix Netzwerk 5

	0	1	2	3	4	5	6	7	8	9
0	919	0	3	2	4	19	22	3	7	1
1	0	1108	7	3	0	0	3	3	11	0
2	12	3	893	22	21	3	21	23	30	4
3	4	0	29	864	1	52	5	16	31	8
4	3	1	7	0	860	4	11	2	13	81
5	16	3	6	74	15	704	19	8	39	8
6	14	4	9	2	21	25	879	1	3	0
7	5	11	24	3	10	4	0	925	7	39
8	14	4	19	31	13	42	8	18	807	18
9	14	4	2	9	64	12	3	43	15	843

Tabelle 8: Konfusionsmatrix Netzwerk 6

	0	1	2	3	4	5	6	7	8	9
0	950	0	3	1	0	8	14	1	3	0
1	0	1102	3	7	0	2	4	1	15	1
2	13	6	900	21	12	3	17	15	42	3
3	7	2	29	863	2	53	4	14	30	6
4	1	2	10	0	832	2	12	3	8	112
5	21	1	13	67	14	726	12	5	28	5
6	23	3	12	0	12	9	893	1	5	0
7	6	18	33	5	11	1	0	904	9	41
8	10	10	19	37	9	29	16	15	813	16
9	11	6	10	10	86	17	1	35	12	821

Tabelle 9: Konfusionsmatrix Netzwerk 7

	0	1	2	3	4	5	6	7	8	9
0	933	0	7	3	3	12	12	3	7	0
1	0	1101	3	3	2	6	6	0	14	0
2	20	5	882	19	20	4	15	22	42	3
3	8	2	29	885	1	40	5	18	16	6
4	3	1	5	1	904	1	12	0	10	45
5	24	3	8	67	27	711	10	13	24	5
6	22	2	12	3	23	13	871	4	7	1
7	2	13	31	2	20	1	2	916	8	33
8	9	8	15	32	13	28	9	16	828	16
9	10	6	5	15	66	6	1	33	12	855

Tabelle 10: Konfusionsmatrix Netzwerk 8

	0	1	2	3	4	5	6	7	8	9
0	937	0	7	1	3	12	18	2	0	0
1	0	1115	4	5	1	2	6	1	0	1
2	15	10	905	30	15	3	21	25	0	8
3	10	1	27	892	2	43	2	18	0	15
4	4	2	2	6	897	4	9	4	0	54
5	20	3	8	71	12	727	27	13	0	11
6	12	4	13	4	16	26	882	1	0	0
7	2	13	29	3	14	1	2	930	0	34
8	61	40	86	187	94	259	63	24	0	160
9	11	4	3	16	98	8	2	31	0	836

Tabelle 11: Konfusionsmatrix Netzwerk 9

	0	1	2	3	4	5	6	7	8	9
0	938	0	6	3	1	12	13	2	5	0
1	0	1106	4	3	1	1	4	2	14	0
2	15	6	899	24	14	1	19	21	30	3
3	4	3	32	863	2	48	4	15	34	5
4	3	1	6	1	822	4	17	2	11	115
5	12	3	4	55	11	731	26	10	33	7
6	18	3	11	3	12	24	879	1	7	0
7	3	12	34	6	11	1	0	913	4	44
8	8	6	11	35	13	35	16	16	823	11
9	15	2	3	10	87	12	1	41	16	822

Tabelle 12: Konfusionsmatrix Netzwerk 10

	0	1	2	3	4	5	6	7	8	9
0	937	0	3	6	2	11	14	3	3	1
1	0	1104	9	4	1	3	4	0	9	1
2	12	1	908	22	17	2	16	13	39	2
3	9	4	36	866	3	51	2	16	15	8
4	1	3	7	3	882	2	18	3	11	52
5	21	4	11	50	9	715	22	9	37	14
6	13	3	8	2	20	14	892	2	4	0
7	4	12	31	11	15	0	0	912	6	37
8	6	4	15	26	15	43	12	5	831	17
9	11	6	2	11	54	14	2	30	17	862

Tabelle 13: Konfusionsmatrix Netzwerk 11

	0	1	2	3	4	5	6	7	8	9
0	931	0	2	13	2	17	5	4	4	2
1	0	1097	5	4	1	2	6	0	19	1
2	20	5	857	42	16	4	18	21	44	5
3	7	2	24	881	1	31	3	19	38	4
4	3	1	4	3	878	5	18	2	11	57
5	18	2	10	52	16	730	19	8	31	6
6	19	2	8	3	15	16	886	1	8	0
7	4	14	25	9	15	3	1	914	12	31
8	6	8	16	40	8	32	13	8	831	12
9	11	3	2	15	72	14	2	35	17	838

Tabelle 14: Konfusionsmatrix Netzwerk 12

	0	1	2	3	4	5	6	7	8	9
0	940	0	0	4	1	16	9	3	6	1
1	0	1106	0	7	3	1	4	2	11	1
2	147	78	0	305	60	19	142	45	187	49
3	7	6	0	900	4	52	1	14	19	7
4	2	2	0	1	854	3	24	4	11	81
5	11	3	0	60	8	736	22	9	35	8
6	22	2	0	6	17	18	880	2	10	1
7	9	24	0	11	15	2	2	932	6	27
8	10	4	0	32	10	26	17	10	844	21
9	12	8	0	13	70	13	4	42	15	832

Tabelle 15: Konfusionsmatrix Netzwerk 13

	0	1	2	3	4	5	6	7	8	9
0	934	1	7	3	4	12	11	3	5	0
1	0	1100	6	6	1	2	5	1	13	1
2	26	4	858	38	16	6	29	14	38	3
3	5	1	27	873	1	42	3	22	33	3
4	1	1	8	0	876	8	15	4	13	56
5	14	3	10	58	9	739	23	4	28	4
6	11	3	15	1	15	17	885	4	7	0
7	4	12	28	4	10	5	0	920	10	35
8	10	7	13	29	10	24	9	16	835	21
9	11	4	5	14	59	9	4	35	18	850

Tabelle 16: Konfusionsmatrix Netzwerk 14

	0	1	2	3	4	5	6	7	8	9
0	947	0	4	7	0	16	0	2	2	2
1	0	1100	7	5	2	4	0	2	15	0
2	16	7	887	29	20	8	0	18	44	3
3	6	2	24	905	1	25	0	15	26	6
4	3	4	7	4	905	6	0	8	3	42
5	18	2	12	47	12	751	0	7	33	10
6	161	26	201	9	202	132	0	2	103	122
7	5	12	33	4	11	2	0	918	6	37
8	10	9	14	28	15	33	0	12	833	20
9	6	7	2	7	59	22	0	42	16	848

Tabelle 17: Konfusionsmatrix Netzwerk 15

	0	1	2	3	4	5	6	7	8	9
0	943	0	5	5	0	15	5	1	5	1
1	0	1107	3	5	1	4	6	2	7	0
2	14	5	896	24	15	3	19	21	33	2
3	7	1	32	869	1	48	3	15	29	5
4	1	3	6	1	881	2	12	5	6	65
5	17	1	8	62	22	703	25	10	36	8
6	14	2	14	1	19	13	886	1	7	1
7	1	12	23	12	11	1	1	935	7	25
8	8	9	16	34	14	39	10	9	818	17
9	7	5	3	15	74	9	5	39	21	831

Tabelle 18: Konfusionsmatrix Netzwerk 16

	0	1	2	3	4	5	6	7	8	9
0	4	0	0	0	0	0	0	0	976	0
1	0	100	0	0	0	0	0	0	1035	0
2	0	0	400	0	0	0	0	2	630	0
3	0	0	3	9	0	0	0	1	997	0
4	0	0	0	0	38	0	0	0	944	0
5	0	0	0	0	0	29	0	0	863	0
6	0	0	0	0	0	0	10	0	948	0
7	0	0	4	0	0	0	0	470	554	0
8	0	0	1	0	0	0	0	3	970	0
9	0	1	0	0	0	0	0	1	997	10

Tabelle 19: Konfusionsmatrix Netzwerk 17

	0	1	2	3	4	5	6	7	8	9
0	927	0	5	3	0	20	16	3	5	1
1	0	1097	3	6	1	2	7	1	18	0
2	21	2	895	24	11	2	23	16	35	3
3	10	0	25	875	2	48	3	13	24	10
4	2	6	9	0	880	2	11	7	11	54
5	18	2	7	63	21	727	10	7	27	10
6	15	3	18	1	14	21	875	6	5	0
7	5	6	32	6	8	3	2	918	10	38
8	15	5	16	20	13	45	13	9	814	24
9	10	5	5	12	62	13	3	34	15	850

Tabelle 20: Konfusionsmatrix Netzwerk 18

	0	1	2	3	4	5	6	7	8	9
0	0	5	102	85	12	369	187	69	76	75
1	0	1107	3	4	1	1	2	2	15	0
2	0	7	895	29	15	7	17	23	36	3
3	0	2	32	856	2	52	6	20	34	6
4	0	6	7	0	833	5	18	3	12	98
5	0	6	13	55	14	747	15	10	24	8
6	0	2	19	2	23	18	883	0	11	0
7	0	15	28	8	17	2	0	910	9	39
8	0	10	13	30	9	30	18	12	831	21
9	0	7	6	5	71	20	3	26	14	857

Tabelle 21: Konfusionsmatrix Netzwerk 19

A.2.2 Komprimierung Version 1

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	980	0	0
1	0	1102	0	0	0	0	0	33	0	0
2	0	225	0	0	0	0	0	807	0	0
3	0	119	0	0	0	0	0	891	0	0
4	0	30	0	0	0	0	0	952	0	0
5	0	145	0	0	0	0	0	747	0	0
6	0	90	0	0	0	0	0	868	0	0
7	0	16	0	0	0	0	0	1012	0	0
8	0	155	0	0	0	0	0	819	0	0
9	0	13	0	0	0	0	0	996	0	0

Tabelle 22: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 0

	0	1	2	3	4	5	6	7	8	9
0	700	0	0	0	20	0	260	0	0	0
1	1135	0	0	0	0	0	0	0	0	0
2	995	0	1	0	1	0	35	0	0	0
3	995	0	2	0	0	0	13	0	0	0
4	936	0	0	0	0	0	46	0	0	0
5	847	0	1	0	19	0	25	0	0	0
6	949	0	0	0	0	0	9	0	0	0
7	315	0	0	0	0	0	713	0	0	0
8	968	0	0	0	2	0	4	0	0	0
9	915	0	0	0	0	0	94	0	0	0

Tabelle 23: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 1

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1135	0	0	0	0	0	0	0	0	0
2	1032	0	0	0	0	0	0	0	0	0
3	1010	0	0	0	0	0	0	0	0	0
4	982	0	0	0	0	0	0	0	0	0
5	892	0	0	0	0	0	0	0	0	0
6	958	0	0	0	0	0	0	0	0	0
7	1028	0	0	0	0	0	0	0	0	0
8	974	0	0	0	0	0	0	0	0	0
9	1009	0	0	0	0	0	0	0	0	0

Tabelle 24: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 2

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 25: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 3

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	835	0	145	0	0
1	0	0	0	0	0	231	0	904	0	0
2	0	0	0	0	0	266	0	766	0	0
3	0	0	0	0	0	712	0	298	0	0
4	0	0	0	0	0	226	0	756	0	0
5	0	0	0	0	0	825	0	67	0	0
6	0	0	0	0	0	664	0	294	0	0
7	0	0	0	0	0	6	0	1022	0	0
8	0	0	0	0	0	652	0	322	0	0
9	0	0	0	0	0	124	0	885	0	0

Tabelle 26: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 4

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 27: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 5

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1112	0	23	0	0
2	0	0	0	0	0	1007	0	25	0	0
3	0	0	0	0	0	1002	0	8	0	0
4	0	0	0	0	0	970	0	12	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	956	0	2	0	0
7	0	0	0	0	0	423	0	605	0	0
8	0	0	0	0	0	973	0	1	0	0
9	0	0	0	0	0	947	0	62	0	0

Tabelle 28: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 6

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	2	956	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 29: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 7

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	59	0	0	0	1076	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 30: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 8

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 31: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 9

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 32: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 10

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	134	846	0	0	0	0
1	0	25	0	0	936	174	0	0	0	0
2	0	0	10	0	870	150	2	0	0	0
3	0	0	0	0	527	481	0	2	0	0
4	0	0	0	0	977	5	0	0	0	0
5	0	0	0	0	212	679	0	1	0	0
6	0	0	1	0	835	122	0	0	0	0
7	0	0	0	0	931	92	0	5	0	0
8	0	0	0	0	745	229	0	0	0	0
9	0	0	0	0	980	29	0	0	0	0

Tabelle 33: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 11

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 34: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 12

	0	1	2	3	4	5	6	7	8	9
0	0	980	0	0	0	0	0	0	0	0
1	0	1135	0	0	0	0	0	0	0	0
2	0	1032	0	0	0	0	0	0	0	0
3	0	1010	0	0	0	0	0	0	0	0
4	0	982	0	0	0	0	0	0	0	0
5	0	892	0	0	0	0	0	0	0	0
6	0	958	0	0	0	0	0	0	0	0
7	0	1028	0	0	0	0	0	0	0	0
8	0	974	0	0	0	0	0	0	0	0
9	0	1009	0	0	0	0	0	0	0	0

Tabelle 35: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 13

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 36: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 14

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	980	0	0	0	0
1	0	0	0	0	0	1135	0	0	0	0
2	0	0	0	0	0	1032	0	0	0	0
3	0	0	0	0	0	1010	0	0	0	0
4	0	0	0	0	0	982	0	0	0	0
5	0	0	0	0	0	892	0	0	0	0
6	0	0	0	0	0	958	0	0	0	0
7	0	0	0	0	0	1028	0	0	0	0
8	0	0	0	0	0	974	0	0	0	0
9	0	0	0	0	0	1009	0	0	0	0

Tabelle 37: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 15

	0	1	2	3	4	5	6	7	8	9
0	624	0	0	0	0	356	0	0	0	0
1	47	0	0	0	2	1086	0	0	0	0
2	764	0	0	0	21	247	0	0	0	0
3	135	0	0	0	0	875	0	0	0	0
4	56	0	0	0	73	853	0	0	0	0
5	23	0	0	0	0	869	0	0	0	0
6	541	0	0	0	13	404	0	0	0	0
7	52	0	0	0	13	963	0	0	0	0
8	27	0	0	0	0	947	0	0	0	0
9	40	0	0	0	11	958	0	0	0	0

Tabelle 38: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 16

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	980	0
1	0	0	0	0	0	0	0	0	1135	0
2	0	0	0	0	0	0	0	0	1032	0
3	0	0	0	0	0	0	0	0	1010	0
4	0	0	0	0	0	0	0	0	982	0
5	0	0	0	0	0	0	0	0	892	0
6	0	0	0	0	0	0	0	0	958	0
7	0	0	0	0	0	0	0	0	1028	0
8	0	0	0	0	0	0	0	0	974	0
9	0	0	0	0	0	0	0	0	1009	0

Tabelle 39: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 17

	0	1	2	3	4	5	6	7	8	9
0	0	0	979	0	1	0	0	0	0	0
1	0	0	1130	0	5	0	0	0	0	0
2	0	0	1025	0	7	0	0	0	0	0
3	0	0	995	0	15	0	0	0	0	0
4	0	0	269	0	713	0	0	0	0	0
5	0	0	687	0	205	0	0	0	0	0
6	0	0	954	0	4	0	0	0	0	0
7	0	0	890	0	138	0	0	0	0	0
8	0	0	938	0	36	0	0	0	0	0
9	0	0	334	0	675	0	0	0	0	0

Tabelle 40: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 18

	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	980	0	0
1	0	0	0	0	0	0	0	1135	0	0
2	0	0	0	0	0	0	0	1032	0	0
3	0	0	0	0	0	0	0	1010	0	0
4	0	0	0	0	0	0	0	982	0	0
5	0	0	0	0	0	1	0	891	0	0
6	0	0	0	0	0	0	0	958	0	0
7	0	0	0	0	0	0	0	1028	0	0
8	0	0	0	0	0	0	0	974	0	0
9	0	0	0	0	0	0	0	1009	0	0

Tabelle 41: Konfusionsmatrix Komprimiertes Netzwerk Art 1 Nr. 19

A.2.3 Komprimierung Version 2

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	21	1108	5	1	0	0	0	0	0	0
2	782	71	175	0	3	0	0	1	0	0
3	842	29	89	48	2	0	0	0	0	0
4	461	31	108	125	257	0	0	0	0	0
5	852	16	17	6	1	0	0	0	0	0
6	783	16	155	1	3	0	0	0	0	0
7	639	66	157	80	52	2	0	32	0	0
8	855	35	63	13	8	0	0	0	0	0
9	437	26	34	213	299	0	0	0	0	0

Tabelle 42: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 0

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	10	1116	5	2	2	0	0	0	0	0
2	762	106	154	2	7	0	0	1	0	0
3	832	35	55	83	4	0	0	1	0	0
4	396	18	60	27	481	0	0	0	0	0
5	798	28	8	36	17	5	0	0	0	0
6	617	28	299	0	10	4	0	0	0	0
7	534	83	114	102	123	0	0	71	1	0
8	577	76	95	58	90	15	8	6	49	0
9	378	30	26	115	457	0	0	3	0	0

Tabelle 43: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 1

	0	1	2	3	4	5	6	7	8	9
0	968	2	9	1	0	0	0	0	0	0
1	0	1127	6	1	1	0	0	0	0	0
2	117	233	679	1	2	0	0	0	0	0
3	108	103	620	178	1	0	0	0	0	0
4	25	46	615	82	214	0	0	0	0	0
5	281	91	190	284	36	9	0	1	0	0
6	181	72	685	15	3	2	0	0	0	0
7	71	71	675	123	88	0	0	0	0	0
8	46	123	707	81	12	1	0	0	4	0
9	32	37	517	275	148	0	0	0	0	0

Tabelle 44: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 2

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	8	1127	0	0	0	0	0	0	0	0
2	597	215	216	1	3	0	0	0	0	0
3	588	146	143	132	1	0	0	0	0	0
4	108	161	254	98	361	0	0	0	0	0
5	776	49	6	57	4	0	0	0	0	0
6	541	108	242	11	35	6	15	0	0	0
7	457	76	143	283	43	0	0	26	0	0
8	486	268	83	107	25	0	0	3	2	0
9	166	50	91	345	354	0	0	3	0	0

Tabelle 45: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 3

	0	1	2	3	4	5	6	7	8	9
0	433	0	255	238	19	25	10	0	0	0
1	831	303	0	0	1	0	0	0	0	0
2	1021	0	6	5	0	0	0	0	0	0
3	632	6	166	206	0	0	0	0	0	0
4	873	10	2	24	73	0	0	0	0	0
5	558	6	19	228	59	22	0	0	0	0
6	933	1	14	5	3	1	1	0	0	0
7	765	9	10	145	91	0	0	8	0	0
8	961	0	1	12	0	0	0	0	0	0
9	715	8	2	161	122	0	1	0	0	0

Tabelle 46: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 4

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1	1132	2	0	0	0	0	0	0	0
2	556	327	147	0	2	0	0	0	0	0
3	552	155	203	99	0	0	0	0	1	0
4	155	52	545	35	195	0	0	0	0	0
5	698	64	44	50	22	14	0	0	0	0
6	527	51	376	0	2	2	0	0	0	0
7	215	284	236	79	164	0	1	49	0	0
8	208	290	396	28	38	10	1	0	3	0
9	151	112	276	129	341	0	0	0	0	0

Tabelle 47: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 5

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	16	1119	0	0	0	0	0	0	0	0
2	732	112	182	0	6	0	0	0	0	0
3	743	80	89	94	4	0	0	0	0	0
4	75	42	69	66	729	0	0	0	1	0
5	799	19	18	23	27	3	0	2	1	0
6	505	34	260	3	111	3	41	0	1	0
7	402	187	126	67	74	0	0	172	0	0
8	597	154	56	66	38	5	7	15	36	0
9	155	84	17	128	557	2	0	61	5	0

Tabelle 48: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 6

	0	1	2	3	4	5	6	7	8	9
0	976	0	3	1	0	0	0	0	0	0
1	4	1127	3	1	0	0	0	0	0	0
2	349	380	296	5	2	0	0	0	0	0
3	336	81	134	459	0	0	0	0	0	0
4	68	30	224	155	503	1	1	0	0	0
5	548	33	30	242	21	15	1	1	1	0
6	420	53	463	9	4	4	5	0	0	0
7	215	164	76	401	48	7	0	117	0	0
8	130	124	283	369	18	15	1	4	30	0
9	67	41	121	453	319	0	0	6	2	0

Tabelle 49: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 7

	0	1	2	3	4	5	6	7	8	9
0	978	1	1	0	0	0	0	0	0	0
1	3	1122	8	2	0	0	0	0	0	0
2	413	170	438	5	6	0	0	0	0	0
3	276	34	335	361	2	0	0	2	0	0
4	128	19	108	144	582	0	1	0	0	0
5	586	22	34	230	14	2	1	2	1	0
6	482	13	358	27	60	0	18	0	0	0
7	254	60	144	234	47	0	6	283	0	0
8	158	61	238	349	38	0	59	13	58	0
9	98	20	67	349	397	1	13	52	12	0

Tabelle 50: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 8

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	22	1112	1	0	0	0	0	0	0	0
2	759	170	102	0	1	0	0	0	0	0
3	764	55	145	43	3	0	0	0	0	0
4	473	38	209	20	242	0	0	0	0	0
5	814	15	26	14	19	4	0	0	0	0
6	684	89	180	0	5	0	0	0	0	0
7	655	145	145	13	37	0	0	33	0	0
8	609	144	151	7	43	12	2	4	0	2
9	462	54	87	109	295	1	0	1	0	0

Tabelle 51: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 9

	0	1	2	3	4	5	6	7	8	9
0	977	1	2	0	0	0	0	0	0	0
1	2	1129	3	0	1	0	0	0	0	0
2	539	266	222	2	3	0	0	0	0	0
3	455	117	259	177	2	0	0	0	0	0
4	94	58	255	208	367	0	0	0	0	0
5	576	48	38	190	36	4	0	0	0	0
6	569	47	336	3	3	0	0	0	0	0
7	170	121	310	297	88	2	0	40	0	0
8	171	223	356	190	32	1	0	0	1	0
9	89	57	163	432	268	0	0	0	0	0

Tabelle 52: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 10

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	21	1110	1	1	0	2	0	0	0	0
2	611	134	280	1	5	0	0	1	0	0
3	474	125	216	193	1	0	0	1	0	0
4	66	84	81	49	701	0	1	0	0	0
5	693	47	34	62	22	33	0	1	0	0
6	536	41	295	0	45	16	25	0	0	0
7	412	98	262	109	64	8	0	75	0	0
8	307	141	329	72	21	51	12	9	32	0
9	117	92	47	104	576	15	0	52	6	0

Tabelle 53: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 11

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	10	1121	3	0	1	0	0	0	0	0
2	589	182	258	0	3	0	0	0	0	0
3	629	63	224	92	2	0	0	0	0	0
4	158	26	321	91	386	0	0	0	0	0
5	788	18	38	41	7	0	0	0	0	0
6	501	42	410	3	2	0	0	0	0	0
7	508	51	280	100	69	0	0	20	0	0
8	399	102	411	38	22	0	0	1	1	0
9	244	16	184	305	258	0	0	2	0	0

Tabelle 54: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 12

	0	1	2	3	4	5	6	7	8	9
0	977	0	1	0	1	1	0	0	0	0
1	2	1108	5	11	6	1	0	0	2	0
2	511	112	97	118	101	1	67	7	18	0
3	388	44	136	430	10	0	0	2	0	0
4	94	11	21	81	774	0	0	1	0	0
5	673	11	41	102	46	17	1	0	1	0
6	453	6	433	2	62	0	2	0	0	0
7	127	42	59	286	272	7	1	230	4	0
8	214	32	21	330	192	26	58	14	87	0
9	81	12	14	217	680	1	0	4	0	0

Tabelle 55: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 13

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	4	1130	1	0	0	0	0	0	0	0
2	555	255	218	0	3	0	0	1	0	0
3	515	166	226	100	2	0	0	1	0	0
4	136	36	304	77	425	0	2	2	0	0
5	810	22	20	31	5	3	0	0	1	0
6	589	61	273	2	11	0	22	0	0	0
7	486	201	172	112	17	0	0	40	0	0
8	436	209	274	30	15	0	1	3	6	0
9	153	70	117	269	290	1	2	98	9	0

Tabelle 56: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 14

	0	1	2	3	4	5	6	7	8	9
0	977	1	2	0	0	0	0	0	0	0
1	3	1129	3	0	0	0	0	0	0	0
2	198	439	394	1	0	0	0	0	0	0
3	412	398	157	43	0	0	0	0	0	0
4	52	224	457	129	120	0	0	0	0	0
5	658	143	42	38	6	5	0	0	0	0
6	332	372	254	0	0	0	0	0	0	0
7	339	181	330	89	35	1	2	51	0	0
8	187	547	181	42	3	6	0	1	7	0
9	102	303	230	230	136	0	1	3	4	0

Tabelle 57: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 15

	0	1	2	3	4	5	6	7	8	9
0	963	1	12	4	0	0	0	0	0	0
1	0	1131	3	1	0	0	0	0	0	0
2	139	270	611	5	6	0	0	1	0	0
3	96	253	430	229	2	0	0	0	0	0
4	23	82	194	255	428	0	0	0	0	0
5	215	183	76	380	29	9	0	0	0	0
6	267	119	515	30	23	1	3	0	0	0
7	30	137	236	491	48	5	1	80	0	0
8	31	364	321	212	15	20	8	2	1	0
9	22	155	134	470	227	0	0	1	0	0

Tabelle 58: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 16

	0	1	2	3	4	5	6	7	8	9
0	975	2	1	1	1	0	0	0	0	0
1	0	1127	5	1	2	0	0	0	0	0
2	266	291	469	1	4	0	0	1	0	0
3	181	106	528	185	5	2	0	1	1	1
4	33	59	351	7	532	0	0	0	0	0
5	330	75	109	179	133	66	0	0	0	0
6	264	49	605	0	25	10	5	0	0	0
7	77	127	447	13	263	0	3	98	0	0
8	128	214	426	14	130	19	19	5	10	9
9	40	51	214	40	659	2	0	2	0	1

Tabelle 59: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 17

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	8	1123	3	1	0	0	0	0	0	0
2	701	112	212	2	4	0	0	1	0	0
3	622	24	124	240	0	0	0	0	0	0
4	203	124	173	139	341	1	1	0	0	0
5	720	19	12	129	9	3	0	0	0	0
6	688	59	179	20	6	4	2	0	0	0
7	385	84	202	248	28	49	0	32	0	0
8	382	84	143	328	14	20	1	0	2	0
9	164	63	88	435	212	17	0	22	8	0

Tabelle 60: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 18

	0	1	2	3	4	5	6	7	8	9
0	34	4	248	349	32	201	88	1	23	0
1	1134	1	0	0	0	0	0	0	0	0
2	957	8	57	4	2	1	0	1	2	0
3	970	4	6	28	1	0	0	1	0	0
4	328	6	85	20	543	0	0	0	0	0
5	514	88	21	193	60	14	1	1	0	0
6	708	8	129	6	54	8	45	0	0	0
7	438	34	30	282	132	0	0	112	0	0
8	845	26	25	29	40	0	0	1	8	0
9	533	1	34	96	344	0	1	0	0	0

Tabelle 61: Konfusionsmatrix Komprimiertes Netzwerk Art 2 Nr. 19

A.2.4 Invertierbares Netz

	0	1	2	3	4	5	6	7	8	9
0	941	0	6	3	3	7	10	4	6	0
1	0	1104	5	2	2	4	6	1	10	1
2	13	3	892	20	17	1	17	20	46	3
3	4	1	27	844	3	61	6	15	37	12
4	1	1	13	0	883	0	19	4	12	49
5	21	4	6	53	9	718	19	5	45	12
6	19	5	20	4	25	22	858	2	3	0
7	4	14	34	9	14	1	3	890	9	50
8	6	6	14	39	15	35	17	9	815	18
9	10	5	12	11	59	15	2	41	9	845

Tabelle 62: Konfusionsmatrix Invertiertes Netzwerk 0

	0	1	2	3	4	5	6	7	8	9
0	932	1	9	4	2	6	12	5	8	1
1	0	1103	3	3	1	4	5	2	12	2
2	17	2	875	31	13	4	21	16	44	9
3	3	4	37	826	4	79	1	13	35	8
4	3	3	12	0	843	9	16	4	11	81
5	22	2	4	71	10	715	17	9	33	9
6	15	2	11	1	17	17	887	1	7	0
7	6	13	31	12	15	1	3	884	12	51
8	12	9	14	26	8	27	27	13	819	19
9	17	5	5	10	69	11	1	27	22	842

Tabelle 63: Konfusionsmatrix Invertiertes Netzwerk 1

	0	1	2	3	4	5	6	7	8	9
0	935	0	6	2	1	17	8	3	8	0
1	0	1105	7	6	0	3	3	0	11	0
2	14	4	890	29	15	9	12	16	41	2
3	7	1	23	888	1	34	4	16	30	6
4	2	5	8	0	869	1	21	6	9	61
5	21	3	8	75	17	703	17	5	38	5
6	12	3	18	2	21	19	876	1	5	1
7	5	18	26	8	15	0	1	903	5	47
8	6	9	19	40	10	38	12	12	813	15
9	9	6	9	13	51	7	2	33	20	859

Tabelle 64: Konfusionsmatrix Invertiertes Netzwerk 2

	0	1	2	3	4	5	6	7	8	9
0	936	1	6	1	1	19	8	4	3	1
1	0	1101	4	1	0	2	4	3	18	2
2	16	3	892	24	23	1	23	16	30	4
3	10	3	24	861	6	47	3	18	32	6
4	1	4	10	0	870	4	18	3	9	63
5	15	3	6	66	13	724	18	7	30	10
6	17	2	12	2	13	18	883	3	8	0
7	7	7	37	6	10	2	3	920	3	33
8	13	7	12	28	15	32	13	9	829	16
9	14	2	5	17	73	12	3	42	13	828

Tabelle 65: Konfusionsmatrix Invertiertes Netzwerk 3

	0	1	2	3	4	5	6	7	8	9
0	0	1	154	89	7	366	187	39	114	23
1	0	1096	7	4	1	3	2	2	19	1
2	0	2	898	22	21	4	23	18	32	12
3	0	2	33	870	0	53	3	16	28	5
4	0	3	7	0	888	0	15	3	11	55
5	0	2	5	56	14	742	19	8	39	7
6	0	3	9	3	17	23	895	2	5	1
7	0	13	30	6	12	2	1	918	7	39
8	0	6	12	30	10	40	23	19	814	20
9	0	5	8	14	81	12	3	38	20	828

Tabelle 66: Konfusionsmatrix Invertiertes Netzwerk 4

	0	1	2	3	4	5	6	7	8	9
0	930	0	7	4	1	12	17	2	5	2
1	0	1104	3	5	1	1	6	1	13	1
2	18	4	885	24	16	3	22	20	35	5
3	8	1	27	854	4	55	4	18	33	6
4	1	3	3	3	876	2	20	7	11	56
5	19	2	6	67	15	705	20	8	42	8
6	15	3	21	5	13	20	879	0	2	0
7	3	16	28	13	11	7	1	906	2	41
8	12	9	15	39	14	37	10	9	817	12
9	9	5	3	18	61	14	2	33	7	857

Tabelle 67: Konfusionsmatrix Invertiertes Netzwerk 5

	0	1	2	3	4	5	6	7	8	9
0	919	0	3	2	4	19	22	3	7	1
1	0	1108	7	3	0	0	3	3	11	0
2	12	3	893	22	21	3	21	23	30	4
3	4	0	29	864	1	52	5	16	31	8
4	3	1	7	0	860	4	11	2	13	81
5	16	3	6	74	15	704	19	8	39	8
6	14	4	9	2	21	25	879	1	3	0
7	5	11	24	3	10	4	0	925	7	39
8	14	4	19	31	13	42	8	18	807	18
9	14	4	2	9	64	12	3	43	15	843

Tabelle 68: Konfusionsmatrix Invertiertes Netzwerk 6

	0	1	2	3	4	5	6	7	8	9
0	950	0	3	1	0	8	14	1	3	0
1	0	1102	3	7	0	2	4	1	15	1
2	13	6	900	21	12	3	17	15	42	3
3	7	2	29	863	2	53	4	14	30	6
4	1	2	10	0	832	2	12	3	8	112
5	21	1	13	67	14	726	12	5	28	5
6	23	3	12	0	12	9	893	1	5	0
7	6	18	33	5	11	1	0	904	9	41
8	10	10	19	37	9	29	16	15	813	16
9	11	6	10	10	86	17	1	35	12	821

Tabelle 69: Konfusionsmatrix Invertiertes Netzwerk 7

	0	1	2	3	4	5	6	7	8	9
0	933	0	7	3	3	12	12	3	7	0
1	0	1101	3	3	2	6	6	0	14	0
2	20	5	882	19	20	4	15	22	42	3
3	8	2	29	885	1	40	5	18	16	6
4	3	1	5	1	904	1	12	0	10	45
5	24	3	8	67	27	711	10	13	24	5
6	22	2	12	3	23	13	871	4	7	1
7	2	13	31	2	20	1	2	916	8	33
8	9	8	15	32	13	28	9	16	828	16
9	10	6	5	15	66	6	1	33	12	855

Tabelle 70: Konfusionsmatrix Invertiertes Netzwerk 8

	0	1	2	3	4	5	6	7	8	9
0	937	0	7	1	3	12	18	2	0	0
1	0	1115	4	5	1	2	6	1	0	1
2	15	10	905	30	15	3	21	25	0	8
3	10	1	27	892	2	43	2	18	0	15
4	4	2	2	6	897	4	9	4	0	54
5	20	3	8	71	12	727	27	13	0	11
6	12	4	13	4	16	26	882	1	0	0
7	2	13	29	3	14	1	2	930	0	34
8	61	40	86	187	94	259	63	24	0	160
9	11	4	3	16	98	8	2	31	0	836

Tabelle 71: Konfusionsmatrix Invertiertes Netzwerk 9

	0	1	2	3	4	5	6	7	8	9
0	938	0	6	3	1	12	13	2	5	0
1	0	1106	4	3	1	1	4	2	14	0
2	15	6	899	24	14	1	19	21	30	3
3	4	3	32	863	2	48	4	15	34	5
4	3	1	6	1	822	4	17	2	11	115
5	12	3	4	55	11	731	26	10	33	7
6	18	3	11	3	12	24	879	1	7	0
7	3	12	34	6	11	1	0	913	4	44
8	8	6	11	35	13	35	16	16	823	11
9	15	2	3	10	87	12	1	41	16	822

Tabelle 72: Konfusionsmatrix Invertiertes Netzwerk 10

	0	1	2	3	4	5	6	7	8	9
0	937	0	3	6	2	11	14	3	3	1
1	0	1104	9	4	1	3	4	0	9	1
2	12	1	908	22	17	2	16	13	39	2
3	9	4	36	866	3	51	2	16	15	8
4	1	3	7	3	882	2	18	3	11	52
5	21	4	11	50	9	715	22	9	37	14
6	13	3	8	2	20	14	892	2	4	0
7	4	12	31	11	15	0	0	912	6	37
8	6	4	15	26	15	43	12	5	831	17
9	11	6	2	11	54	14	2	30	17	862

Tabelle 73: Konfusionsmatrix Invertiertes Netzwerk 11

	0	1	2	3	4	5	6	7	8	9
0	931	0	2	13	2	17	5	4	4	2
1	0	1097	5	4	1	2	6	0	19	1
2	20	5	857	42	16	4	18	21	44	5
3	7	2	24	881	1	31	3	19	38	4
4	3	1	4	3	878	5	18	2	11	57
5	18	2	10	52	16	730	19	8	31	6
6	19	2	8	3	15	16	886	1	8	0
7	4	14	25	9	15	3	1	914	12	31
8	6	8	16	40	8	32	13	8	831	12
9	11	3	2	15	72	14	2	35	17	838

Tabelle 74: Konfusionsmatrix Invertiertes Netzwerk 12

	0	1	2	3	4	5	6	7	8	9
0	940	0	0	4	1	16	9	3	6	1
1	0	1106	0	7	3	1	4	2	11	1
2	147	78	0	305	60	19	142	45	187	49
3	7	6	0	900	4	52	1	14	19	7
4	2	2	0	1	854	3	24	4	11	81
5	11	3	0	60	8	736	22	9	35	8
6	22	2	0	6	17	18	880	2	10	1
7	9	24	0	11	15	2	2	932	6	27
8	10	4	0	32	10	26	17	10	844	21
9	12	8	0	13	70	13	4	42	15	832

Tabelle 75: Konfusionsmatrix Invertiertes Netzwerk 13

	0	1	2	3	4	5	6	7	8	9
0	934	1	7	3	4	12	11	3	5	0
1	0	1100	6	6	1	2	5	1	13	1
2	26	4	858	38	16	6	29	14	38	3
3	5	1	27	873	1	42	3	22	33	3
4	1	1	8	0	876	8	15	4	13	56
5	14	3	10	58	9	739	23	4	28	4
6	11	3	15	1	15	17	885	4	7	0
7	4	12	28	4	10	5	0	920	10	35
8	10	7	13	29	10	24	9	16	835	21
9	11	4	5	14	59	9	4	35	18	850

Tabelle 76: Konfusionsmatrix Invertiertes Netzwerk 14

	0	1	2	3	4	5	6	7	8	9
0	947	0	4	7	0	16	0	2	2	2
1	0	1100	7	5	2	4	0	2	15	0
2	16	7	887	29	20	8	0	18	44	3
3	6	2	24	905	1	25	0	15	26	6
4	3	4	7	4	905	6	0	8	3	42
5	18	2	12	47	12	751	0	7	33	10
6	161	26	201	9	202	132	0	2	103	122
7	5	12	33	4	11	2	0	918	6	37
8	10	9	14	28	15	33	0	12	833	20
9	6	7	2	7	59	22	0	42	16	848

Tabelle 77: Konfusionsmatrix Invertiertes Netzwerk 15

	0	1	2	3	4	5	6	7	8	9
0	943	0	5	5	0	15	5	1	5	1
1	0	1107	3	5	1	4	6	2	7	0
2	14	5	896	24	15	3	19	21	33	2
3	7	1	32	869	1	48	3	15	29	5
4	1	3	6	1	881	2	12	5	6	65
5	17	1	8	62	22	703	25	10	36	8
6	14	2	14	1	19	13	886	1	7	1
7	1	12	23	12	11	1	1	935	7	25
8	8	9	16	34	14	39	10	9	818	17
9	7	5	3	15	74	9	5	39	21	831

Tabelle 78: Konfusionsmatrix Invertiertes Netzwerk 16

	0	1	2	3	4	5	6	7	8	9
0	4	0	0	0	0	0	0	0	976	0
1	0	100	0	0	0	0	0	0	1035	0
2	0	0	400	0	0	0	0	2	630	0
3	0	0	3	9	0	0	0	1	997	0
4	0	0	0	0	38	0	0	0	944	0
5	0	0	0	0	0	29	0	0	863	0
6	0	0	0	0	0	0	10	0	948	0
7	0	0	4	0	0	0	0	470	554	0
8	0	0	1	0	0	0	0	3	970	0
9	0	1	0	0	0	0	0	1	997	10

Tabelle 79: Konfusionsmatrix Invertiertes Netzwerk 17

	0	1	2	3	4	5	6	7	8	9
0	927	0	5	3	0	20	16	3	5	1
1	0	1097	3	6	1	2	7	1	18	0
2	21	2	895	24	11	2	23	16	35	3
3	10	0	25	875	2	48	3	13	24	10
4	2	6	9	0	880	2	11	7	11	54
5	18	2	7	63	21	727	10	7	27	10
6	15	3	18	1	14	21	875	6	5	0
7	5	6	32	6	8	3	2	918	10	38
8	15	5	16	20	13	45	13	9	814	24
9	10	5	5	12	62	13	3	34	15	850

Tabelle 80: Konfusionsmatrix Invertiertes Netzwerk 18

	0	1	2	3	4	5	6	7	8	9
0	0	5	102	85	12	369	187	69	76	75
1	0	1107	3	4	1	1	2	2	15	0
2	0	7	895	29	15	7	17	23	36	3
3	0	2	32	856	2	52	6	20	34	6
4	0	6	7	0	833	5	18	3	12	98
5	0	6	13	55	14	747	15	10	24	8
6	0	2	19	2	23	18	883	0	11	0
7	0	15	28	8	17	2	0	910	9	39
8	0	10	13	30	9	30	18	12	831	21
9	0	7	6	5	71	20	3	26	14	857

Tabelle 81: Konfusionsmatrix Invertiertes Netzwerk 19

A.2.5 Klassifizierung Quadratische Netze

	0	1	2	3	4	5	6	7	8	9
0	978	2	0	0	0	0	0	0	0	0
1	1067	67	1	0	0	0	0	0	0	0
2	1021	10	1	0	0	0	0	0	0	0
3	980	29	1	0	0	0	0	0	0	0
4	978	3	1	0	0	0	0	0	0	0
5	885	7	0	0	0	0	0	0	0	0
6	950	7	0	1	0	0	0	0	0	0
7	1012	13	3	0	0	0	0	0	0	0
8	972	2	0	0	0	0	0	0	0	0
9	1005	3	1	0	0	0	0	0	0	0

Tabelle 82: Konfusionsmatrix Quadratisches Netzwerk 0

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	1085	50	0	0	0	0	0	0	0	0
2	919	92	21	0	0	0	0	0	0	0
3	990	19	1	0	0	0	0	0	0	0
4	784	55	143	0	0	0	0	0	0	0
5	879	9	4	0	0	0	0	0	0	0
6	953	4	1	0	0	0	0	0	0	0
7	1007	17	4	0	0	0	0	0	0	0
8	930	43	1	0	0	0	0	0	0	0
9	977	18	14	0	0	0	0	0	0	0

Tabelle 83: Konfusionsmatrix Quadratisches Netzwerk 1

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	982	153	0	0	0	0	0	0	0	0
2	1029	3	0	0	0	0	0	0	0	0
3	1004	5	1	0	0	0	0	0	0	0
4	915	37	29	1	0	0	0	0	0	0
5	880	9	2	1	0	0	0	0	0	0
6	928	30	0	0	0	0	0	0	0	0
7	976	5	42	5	0	0	0	0	0	0
8	962	12	0	0	0	0	0	0	0	0
9	927	26	56	0	0	0	0	0	0	0

Tabelle 84: Konfusionsmatrix Quadratisches Netzwerk 2

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	885	249	1	0	0	0	0	0	0	0
2	932	100	0	0	0	0	0	0	0	0
3	965	45	0	0	0	0	0	0	0	0
4	814	163	5	0	0	0	0	0	0	0
5	860	29	3	0	0	0	0	0	0	0
6	856	102	0	0	0	0	0	0	0	0
7	972	56	0	0	0	0	0	0	0	0
8	888	86	0	0	0	0	0	0	0	0
9	880	128	1	0	0	0	0	0	0	0

Tabelle 85: Konfusionsmatrix Quadratisches Netzwerk 3

	0	1	2	3	4	5	6	7	8	9
0	977	0	2	1	0	0	0	0	0	0
1	871	264	0	0	0	0	0	0	0	0
2	960	59	13	0	0	0	0	0	0	0
3	895	109	6	0	0	0	0	0	0	0
4	866	72	44	0	0	0	0	0	0	0
5	862	21	9	0	0	0	0	0	0	0
6	876	38	44	0	0	0	0	0	0	0
7	995	25	8	0	0	0	0	0	0	0
8	952	19	3	0	0	0	0	0	0	0
9	978	16	15	0	0	0	0	0	0	0

Tabelle 86: Konfusionsmatrix Quadratisches Netzwerk 4

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	469	665	1	0	0	0	0	0	0	0
2	988	32	12	0	0	0	0	0	0	0
3	999	5	6	0	0	0	0	0	0	0
4	866	75	41	0	0	0	0	0	0	0
5	804	55	28	5	0	0	0	0	0	0
6	925	30	3	0	0	0	0	0	0	0
7	967	35	26	0	0	0	0	0	0	0
8	910	54	10	0	0	0	0	0	0	0
9	935	33	40	1	0	0	0	0	0	0

Tabelle 87: Konfusionsmatrix Quadratisches Netzwerk 5

	0	1	2	3	4	5	6	7	8	9
0	978	1	1	0	0	0	0	0	0	0
1	979	156	0	0	0	0	0	0	0	0
2	760	249	23	0	0	0	0	0	0	0
3	970	39	1	0	0	0	0	0	0	0
4	903	77	2	0	0	0	0	0	0	0
5	880	10	2	0	0	0	0	0	0	0
6	887	67	3	1	0	0	0	0	0	0
7	913	110	5	0	0	0	0	0	0	0
8	935	36	3	0	0	0	0	0	0	0
9	964	45	0	0	0	0	0	0	0	0

Tabelle 88: Konfusionsmatrix Quadratisches Netzwerk 6

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	757	378	0	0	0	0	0	0	0	0
2	949	82	1	0	0	0	0	0	0	0
3	975	34	1	0	0	0	0	0	0	0
4	915	59	8	0	0	0	0	0	0	0
5	856	34	2	0	0	0	0	0	0	0
6	953	3	2	0	0	0	0	0	0	0
7	681	344	3	0	0	0	0	0	0	0
8	917	55	2	0	0	0	0	0	0	0
9	957	51	1	0	0	0	0	0	0	0

Tabelle 89: Konfusionsmatrix Quadratisches Netzwerk 7

	0	1	2	3	4	5	6	7	8	9
0	978	2	0	0	0	0	0	0	0	0
1	1124	11	0	0	0	0	0	0	0	0
2	1032	0	0	0	0	0	0	0	0	0
3	1005	4	1	0	0	0	0	0	0	0
4	952	20	10	0	0	0	0	0	0	0
5	864	16	12	0	0	0	0	0	0	0
6	935	23	0	0	0	0	0	0	0	0
7	992	7	24	5	0	0	0	0	0	0
8	967	6	1	0	0	0	0	0	0	0
9	938	51	20	0	0	0	0	0	0	0

Tabelle 90: Konfusionsmatrix Quadratisches Netzwerk 8

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1026	109	0	0	0	0	0	0	0	0
2	1018	9	5	0	0	0	0	0	0	0
3	1000	8	2	0	0	0	0	0	0	0
4	808	106	68	0	0	0	0	0	0	0
5	877	9	6	0	0	0	0	0	0	0
6	949	3	6	0	0	0	0	0	0	0
7	984	33	10	1	0	0	0	0	0	0
8	959	13	2	0	0	0	0	0	0	0
9	900	22	87	0	0	0	0	0	0	0

Tabelle 91: Konfusionsmatrix Quadratisches Netzwerk 9

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1115	20	0	0	0	0	0	0	0	0
2	1025	7	0	0	0	0	0	0	0	0
3	1008	2	0	0	0	0	0	0	0	0
4	971	8	3	0	0	0	0	0	0	0
5	891	1	0	0	0	0	0	0	0	0
6	942	16	0	0	0	0	0	0	0	0
7	1013	15	0	0	0	0	0	0	0	0
8	974	0	0	0	0	0	0	0	0	0
9	987	21	1	0	0	0	0	0	0	0

Tabelle 92: Konfusionsmatrix Quadratisches Netzwerk 10

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	867	268	0	0	0	0	0	0	0	0
2	1010	20	2	0	0	0	0	0	0	0
3	997	13	0	0	0	0	0	0	0	0
4	916	63	3	0	0	0	0	0	0	0
5	867	25	0	0	0	0	0	0	0	0
6	953	5	0	0	0	0	0	0	0	0
7	968	60	0	0	0	0	0	0	0	0
8	971	3	0	0	0	0	0	0	0	0
9	965	41	3	0	0	0	0	0	0	0

Tabelle 93: Konfusionsmatrix Quadratisches Netzwerk 11

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1050	85	0	0	0	0	0	0	0	0
2	1003	27	2	0	0	0	0	0	0	0
3	973	34	3	0	0	0	0	0	0	0
4	949	32	1	0	0	0	0	0	0	0
5	891	1	0	0	0	0	0	0	0	0
6	948	9	1	0	0	0	0	0	0	0
7	998	28	2	0	0	0	0	0	0	0
8	971	3	0	0	0	0	0	0	0	0
9	966	42	1	0	0	0	0	0	0	0

Tabelle 94: Konfusionsmatrix Quadratisches Netzwerk 12

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	1132	3	0	0	0	0	0	0	0	0
2	1023	7	2	0	0	0	0	0	0	0
3	997	12	1	0	0	0	0	0	0	0
4	970	5	7	0	0	0	0	0	0	0
5	862	29	1	0	0	0	0	0	0	0
6	951	3	4	0	0	0	0	0	0	0
7	769	197	62	0	0	0	0	0	0	0
8	970	4	0	0	0	0	0	0	0	0
9	971	21	17	0	0	0	0	0	0	0

Tabelle 95: Konfusionsmatrix Quadratisches Netzwerk 13

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	591	544	0	0	0	0	0	0	0	0
2	662	362	8	0	0	0	0	0	0	0
3	918	91	1	0	0	0	0	0	0	0
4	412	532	36	2	0	0	0	0	0	0
5	799	79	11	2	1	0	0	0	0	0
6	795	159	3	1	0	0	0	0	0	0
7	530	474	24	0	0	0	0	0	0	0
8	707	266	1	0	0	0	0	0	0	0
9	424	450	135	0	0	0	0	0	0	0

Tabelle 96: Konfusionsmatrix Quadratisches Netzwerk 14

	0	1	2	3	4	5	6	7	8	9
0	978	0	1	1	0	0	0	0	0	0
1	1083	52	0	0	0	0	0	0	0	0
2	1017	13	2	0	0	0	0	0	0	0
3	1001	8	1	0	0	0	0	0	0	0
4	950	26	6	0	0	0	0	0	0	0
5	881	6	5	0	0	0	0	0	0	0
6	951	6	0	1	0	0	0	0	0	0
7	1008	19	1	0	0	0	0	0	0	0
8	970	3	1	0	0	0	0	0	0	0
9	995	13	1	0	0	0	0	0	0	0

Tabelle 97: Konfusionsmatrix Quadratisches Netzwerk 15

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	1104	31	0	0	0	0	0	0	0	0
2	1029	3	0	0	0	0	0	0	0	0
3	993	14	3	0	0	0	0	0	0	0
4	981	1	0	0	0	0	0	0	0	0
5	867	18	5	2	0	0	0	0	0	0
6	952	6	0	0	0	0	0	0	0	0
7	930	96	2	0	0	0	0	0	0	0
8	969	3	2	0	0	0	0	0	0	0
9	1001	8	0	0	0	0	0	0	0	0

Tabelle 98: Konfusionsmatrix Quadratisches Netzwerk 16

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	513	622	0	0	0	0	0	0	0	0
2	918	110	4	0	0	0	0	0	0	0
3	902	94	11	3	0	0	0	0	0	0
4	981	1	0	0	0	0	0	0	0	0
5	869	14	9	0	0	0	0	0	0	0
6	941	17	0	0	0	0	0	0	0	0
7	1018	10	0	0	0	0	0	0	0	0
8	948	26	0	0	0	0	0	0	0	0
9	1005	4	0	0	0	0	0	0	0	0

Tabelle 99: Konfusionsmatrix Quadratisches Netzwerk 17

	0	1	2	3	4	5	6	7	8	9
0	978	2	0	0	0	0	0	0	0	0
1	1029	106	0	0	0	0	0	0	0	0
2	917	92	23	0	0	0	0	0	0	0
3	980	25	5	0	0	0	0	0	0	0
4	786	74	114	8	0	0	0	0	0	0
5	886	4	1	1	0	0	0	0	0	0
6	915	21	22	0	0	0	0	0	0	0
7	1003	17	8	0	0	0	0	0	0	0
8	960	13	1	0	0	0	0	0	0	0
9	961	32	15	1	0	0	0	0	0	0

Tabelle 100: Konfusionsmatrix Quadratisches Netzwerk 18

	0	1	2	3	4	5	6	7	8	9
0	977	3	0	0	0	0	0	0	0	0
1	892	243	0	0	0	0	0	0	0	0
2	1021	10	1	0	0	0	0	0	0	0
3	976	34	0	0	0	0	0	0	0	0
4	942	33	7	0	0	0	0	0	0	0
5	875	17	0	0	0	0	0	0	0	0
6	927	17	14	0	0	0	0	0	0	0
7	985	33	9	1	0	0	0	0	0	0
8	971	3	0	0	0	0	0	0	0	0
9	988	20	1	0	0	0	0	0	0	0

Tabelle 101: Konfusionsmatrix Quadratisches Netzwerk 19

A.2.6 Invertierung Quadratische Netze

	0	1	2	3	4	5	6	7	8	9
0	978	2	0	0	0	0	0	0	0	0
1	1057	77	1	0	0	0	0	0	0	0
2	1021	10	1	0	0	0	0	0	0	0
3	980	29	1	0	0	0	0	0	0	0
4	978	3	1	0	0	0	0	0	0	0
5	885	7	0	0	0	0	0	0	0	0
6	949	8	0	1	0	0	0	0	0	0
7	1012	13	3	0	0	0	0	0	0	0
8	972	2	0	0	0	0	0	0	0	0
9	1005	3	1	0	0	0	0	0	0	0

Tabelle 102: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 0

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	1084	51	0	0	0	0	0	0	0	0
2	919	91	22	0	0	0	0	0	0	0
3	989	20	1	0	0	0	0	0	0	0
4	781	55	146	0	0	0	0	0	0	0
5	879	9	4	0	0	0	0	0	0	0
6	953	3	2	0	0	0	0	0	0	0
7	1005	19	4	0	0	0	0	0	0	0
8	928	45	1	0	0	0	0	0	0	0
9	974	21	14	0	0	0	0	0	0	0

Tabelle 103: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 1

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	975	160	0	0	0	0	0	0	0	0
2	1029	3	0	0	0	0	0	0	0	0
3	1004	5	1	0	0	0	0	0	0	0
4	913	37	31	1	0	0	0	0	0	0
5	880	9	2	1	0	0	0	0	0	0
6	927	31	0	0	0	0	0	0	0	0
7	975	5	43	5	0	0	0	0	0	0
8	961	13	0	0	0	0	0	0	0	0
9	925	27	57	0	0	0	0	0	0	0

Tabelle 104: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 2

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	885	249	1	0	0	0	0	0	0	0
2	925	107	0	0	0	0	0	0	0	0
3	963	47	0	0	0	0	0	0	0	0
4	813	164	5	0	0	0	0	0	0	0
5	857	32	3	0	0	0	0	0	0	0
6	855	103	0	0	0	0	0	0	0	0
7	972	56	0	0	0	0	0	0	0	0
8	883	91	0	0	0	0	0	0	0	0
9	875	133	1	0	0	0	0	0	0	0

Tabelle 105: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 3

	0	1	2	3	4	5	6	7	8	9
0	977	0	2	1	0	0	0	0	0	0
1	862	272	1	0	0	0	0	0	0	0
2	958	60	14	0	0	0	0	0	0	0
3	894	110	6	0	0	0	0	0	0	0
4	866	72	44	0	0	0	0	0	0	0
5	861	22	9	0	0	0	0	0	0	0
6	874	38	46	0	0	0	0	0	0	0
7	995	25	8	0	0	0	0	0	0	0
8	952	19	3	0	0	0	0	0	0	0
9	978	16	15	0	0	0	0	0	0	0

Tabelle 106: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 4

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	461	673	1	0	0	0	0	0	0	0
2	986	33	13	0	0	0	0	0	0	0
3	999	5	6	0	0	0	0	0	0	0
4	865	75	41	1	0	0	0	0	0	0
5	803	56	28	5	0	0	0	0	0	0
6	925	30	3	0	0	0	0	0	0	0
7	964	34	30	0	0	0	0	0	0	0
8	909	54	11	0	0	0	0	0	0	0
9	932	36	40	1	0	0	0	0	0	0

Tabelle 107: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 5

	0	1	2	3	4	5	6	7	8	9
0	978	1	1	0	0	0	0	0	0	0
1	972	163	0	0	0	0	0	0	0	0
2	756	253	23	0	0	0	0	0	0	0
3	969	40	1	0	0	0	0	0	0	0
4	903	77	2	0	0	0	0	0	0	0
5	880	10	2	0	0	0	0	0	0	0
6	884	69	4	1	0	0	0	0	0	0
7	911	112	5	0	0	0	0	0	0	0
8	935	36	3	0	0	0	0	0	0	0
9	961	48	0	0	0	0	0	0	0	0

Tabelle 108: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 6

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	752	383	0	0	0	0	0	0	0	0
2	948	82	2	0	0	0	0	0	0	0
3	974	35	1	0	0	0	0	0	0	0
4	914	60	8	0	0	0	0	0	0	0
5	854	36	2	0	0	0	0	0	0	0
6	953	3	2	0	0	0	0	0	0	0
7	676	349	3	0	0	0	0	0	0	0
8	917	55	2	0	0	0	0	0	0	0
9	956	52	1	0	0	0	0	0	0	0

Tabelle 109: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 7

	0	1	2	3	4	5	6	7	8	9
0	978	2	0	0	0	0	0	0	0	0
1	1124	11	0	0	0	0	0	0	0	0
2	1032	0	0	0	0	0	0	0	0	0
3	1005	4	1	0	0	0	0	0	0	0
4	950	20	12	0	0	0	0	0	0	0
5	864	16	12	0	0	0	0	0	0	0
6	935	23	0	0	0	0	0	0	0	0
7	991	6	26	5	0	0	0	0	0	0
8	967	6	1	0	0	0	0	0	0	0
9	936	52	21	0	0	0	0	0	0	0

Tabelle 110: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 8

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1026	109	0	0	0	0	0	0	0	0
2	1018	9	5	0	0	0	0	0	0	0
3	999	9	2	0	0	0	0	0	0	0
4	802	110	70	0	0	0	0	0	0	0
5	877	8	7	0	0	0	0	0	0	0
6	949	3	6	0	0	0	0	0	0	0
7	983	33	11	1	0	0	0	0	0	0
8	959	13	2	0	0	0	0	0	0	0
9	895	24	90	0	0	0	0	0	0	0

Tabelle 111: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 9

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1112	23	0	0	0	0	0	0	0	0
2	1025	7	0	0	0	0	0	0	0	0
3	1008	2	0	0	0	0	0	0	0	0
4	969	8	5	0	0	0	0	0	0	0
5	891	1	0	0	0	0	0	0	0	0
6	942	16	0	0	0	0	0	0	0	0
7	1011	17	0	0	0	0	0	0	0	0
8	974	0	0	0	0	0	0	0	0	0
9	984	23	2	0	0	0	0	0	0	0

Tabelle 112: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 10

	0	1	2	3	4	5	6	7	8	9
0	979	0	1	0	0	0	0	0	0	0
1	856	279	0	0	0	0	0	0	0	0
2	1010	20	2	0	0	0	0	0	0	0
3	997	13	0	0	0	0	0	0	0	0
4	915	64	3	0	0	0	0	0	0	0
5	866	26	0	0	0	0	0	0	0	0
6	952	6	0	0	0	0	0	0	0	0
7	968	60	0	0	0	0	0	0	0	0
8	971	3	0	0	0	0	0	0	0	0
9	964	42	3	0	0	0	0	0	0	0

Tabelle 113: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 11

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	1045	90	0	0	0	0	0	0	0	0
2	1003	27	2	0	0	0	0	0	0	0
3	972	35	3	0	0	0	0	0	0	0
4	949	32	1	0	0	0	0	0	0	0
5	891	1	0	0	0	0	0	0	0	0
6	947	10	1	0	0	0	0	0	0	0
7	997	28	3	0	0	0	0	0	0	0
8	971	3	0	0	0	0	0	0	0	0
9	966	42	1	0	0	0	0	0	0	0

Tabelle 114: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 12

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	1132	3	0	0	0	0	0	0	0	0
2	1022	7	3	0	0	0	0	0	0	0
3	997	12	1	0	0	0	0	0	0	0
4	969	6	7	0	0	0	0	0	0	0
5	862	29	1	0	0	0	0	0	0	0
6	951	2	5	0	0	0	0	0	0	0
7	766	197	65	0	0	0	0	0	0	0
8	969	5	0	0	0	0	0	0	0	0
9	968	23	18	0	0	0	0	0	0	0

Tabelle 115: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 13

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	575	560	0	0	0	0	0	0	0	0
2	659	365	8	0	0	0	0	0	0	0
3	914	94	2	0	0	0	0	0	0	0
4	401	539	40	2	0	0	0	0	0	0
5	795	82	12	2	1	0	0	0	0	0
6	794	160	3	1	0	0	0	0	0	0
7	523	478	27	0	0	0	0	0	0	0
8	700	273	1	0	0	0	0	0	0	0
9	416	455	138	0	0	0	0	0	0	0

Tabelle 116: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 14

	0	1	2	3	4	5	6	7	8	9
0	978	0	1	1	0	0	0	0	0	0
1	1080	55	0	0	0	0	0	0	0	0
2	1017	13	2	0	0	0	0	0	0	0
3	1001	8	1	0	0	0	0	0	0	0
4	950	26	6	0	0	0	0	0	0	0
5	881	6	5	0	0	0	0	0	0	0
6	951	6	0	1	0	0	0	0	0	0
7	1008	19	1	0	0	0	0	0	0	0
8	970	3	1	0	0	0	0	0	0	0
9	995	13	1	0	0	0	0	0	0	0

Tabelle 117: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 15

	0	1	2	3	4	5	6	7	8	9
0	979	1	0	0	0	0	0	0	0	0
1	1103	32	0	0	0	0	0	0	0	0
2	1029	3	0	0	0	0	0	0	0	0
3	992	15	3	0	0	0	0	0	0	0
4	981	1	0	0	0	0	0	0	0	0
5	867	18	5	2	0	0	0	0	0	0
6	952	6	0	0	0	0	0	0	0	0
7	928	98	2	0	0	0	0	0	0	0
8	969	3	2	0	0	0	0	0	0	0
9	1001	8	0	0	0	0	0	0	0	0

Tabelle 118: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 16

	0	1	2	3	4	5	6	7	8	9
0	980	0	0	0	0	0	0	0	0	0
1	507	628	0	0	0	0	0	0	0	0
2	917	110	5	0	0	0	0	0	0	0
3	901	95	10	4	0	0	0	0	0	0
4	981	1	0	0	0	0	0	0	0	0
5	869	14	9	0	0	0	0	0	0	0
6	940	18	0	0	0	0	0	0	0	0
7	1018	10	0	0	0	0	0	0	0	0
8	947	27	0	0	0	0	0	0	0	0
9	1005	4	0	0	0	0	0	0	0	0

Tabelle 119: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 17

	0	1	2	3	4	5	6	7	8	9
0	978	2	0	0	0	0	0	0	0	0
1	1025	110	0	0	0	0	0	0	0	0
2	915	93	24	0	0	0	0	0	0	0
3	978	26	5	1	0	0	0	0	0	0
4	781	72	121	8	0	0	0	0	0	0
5	886	4	1	1	0	0	0	0	0	0
6	914	21	23	0	0	0	0	0	0	0
7	1002	18	8	0	0	0	0	0	0	0
8	958	14	2	0	0	0	0	0	0	0
9	960	33	15	1	0	0	0	0	0	0

Tabelle 120: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 18

	0	1	2	3	4	5	6	7	8	9
0	977	3	0	0	0	0	0	0	0	0
1	886	249	0	0	0	0	0	0	0	0
2	1021	10	1	0	0	0	0	0	0	0
3	976	34	0	0	0	0	0	0	0	0
4	941	33	7	1	0	0	0	0	0	0
5	875	17	0	0	0	0	0	0	0	0
6	927	17	14	0	0	0	0	0	0	0
7	984	34	9	1	0	0	0	0	0	0
8	971	3	0	0	0	0	0	0	0	0
9	988	20	1	0	0	0	0	0	0	0

Tabelle 121: Konfusionsmatrix Quadratisches Invertiertes Netzwerk 19

A.3 Fehler Invertierung

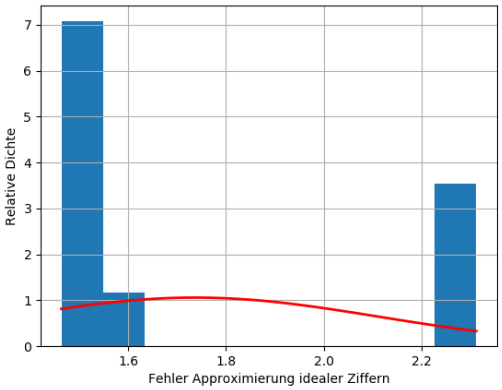


Abbildung 24: Fehler Approximierung idealer Ziffern

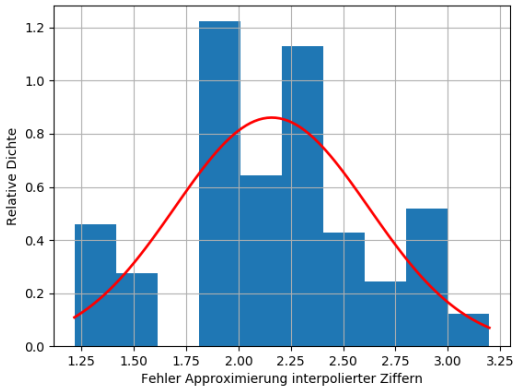
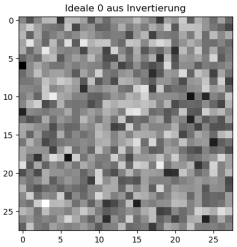
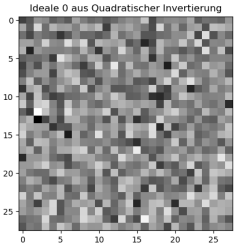
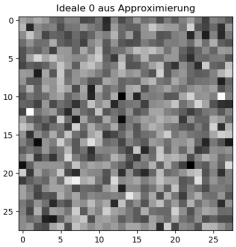
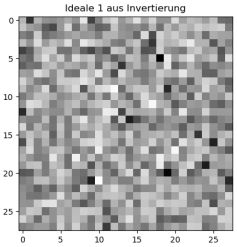
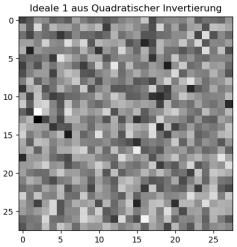
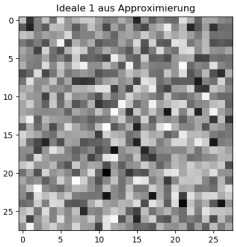
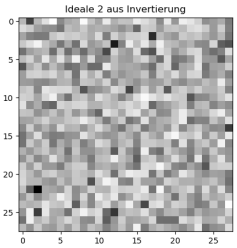
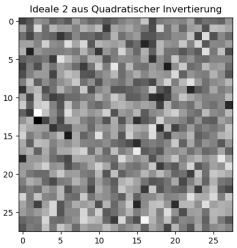
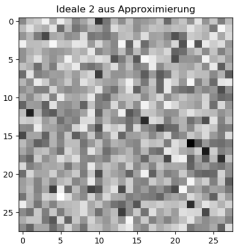
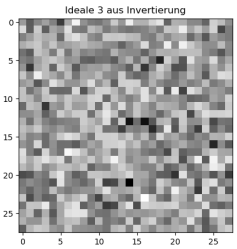
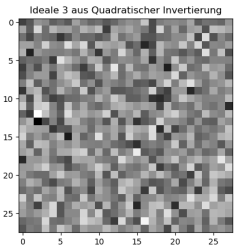
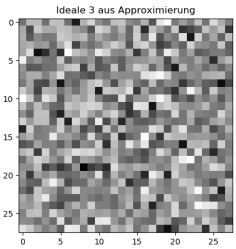
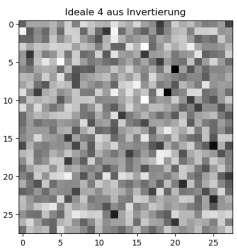
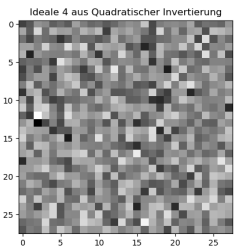
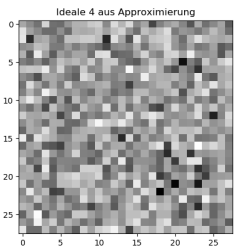
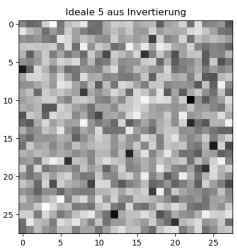
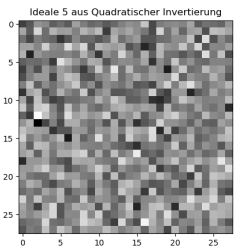
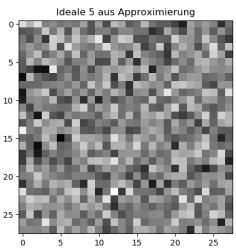
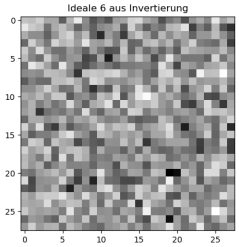
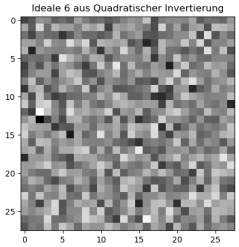
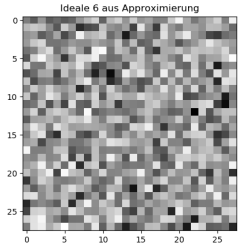
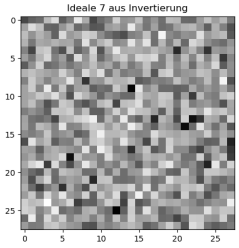
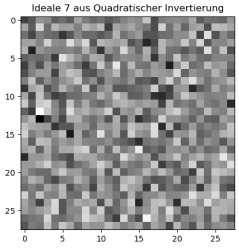
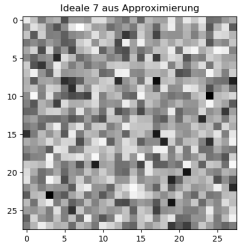
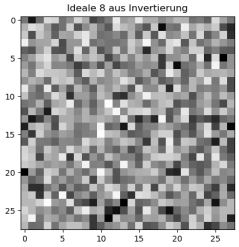
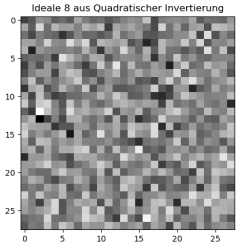
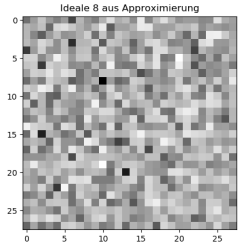
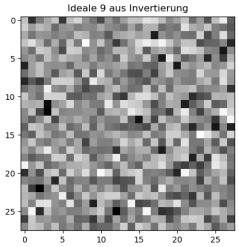
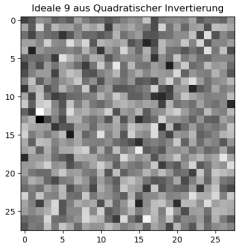
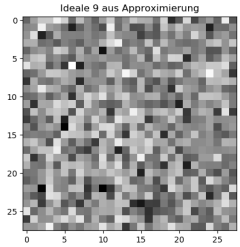


Abbildung 25: Fehler Approximierung interploierter Ziffern

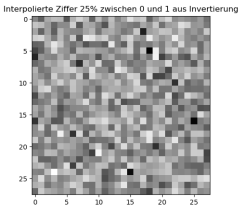
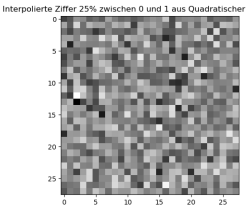
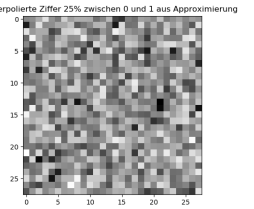
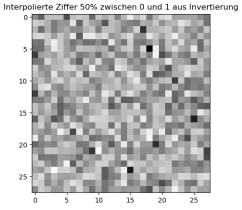
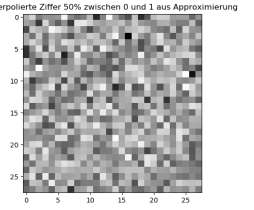
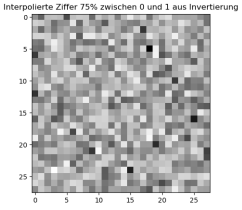
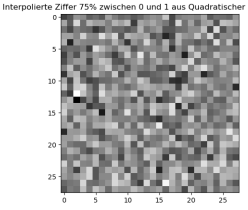
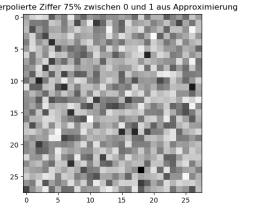
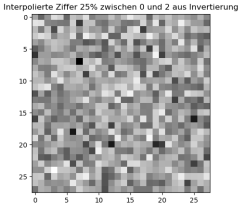
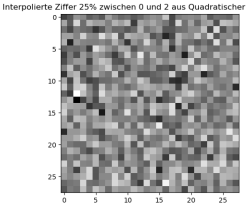
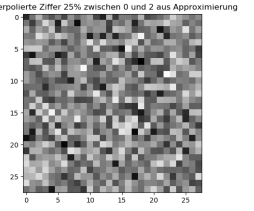
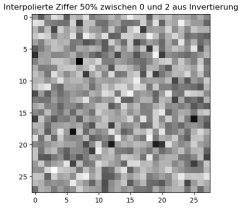
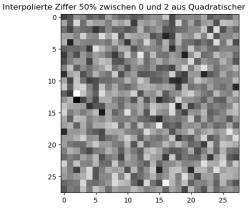
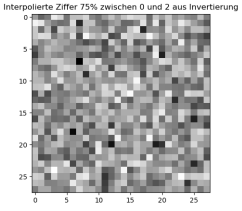
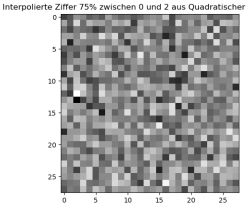
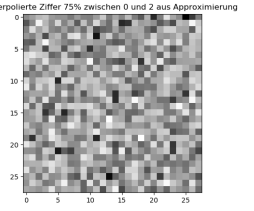
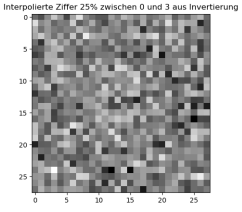
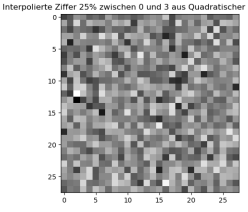
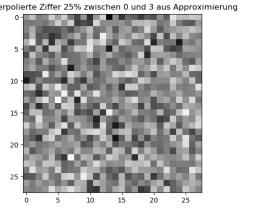
A.4 Bilder aus der Invertierung

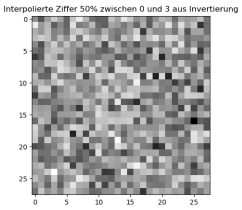
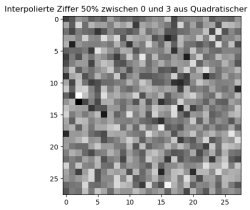
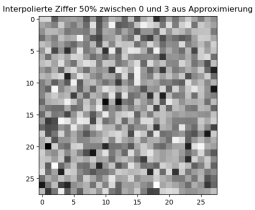
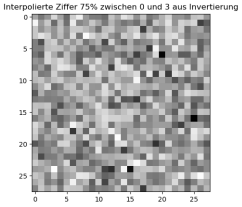
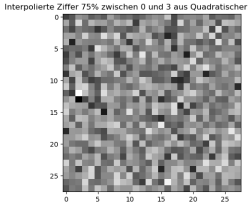
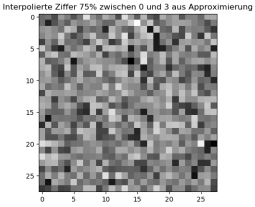
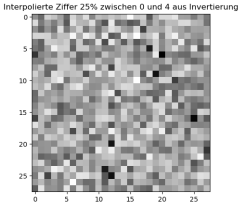
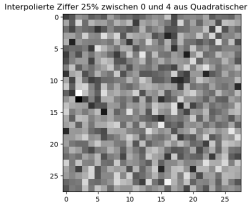
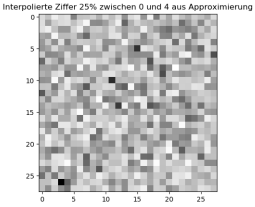
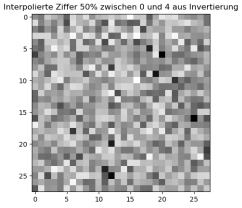
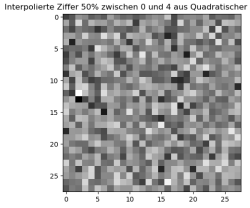
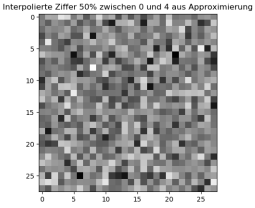
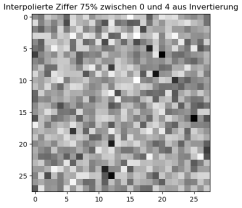
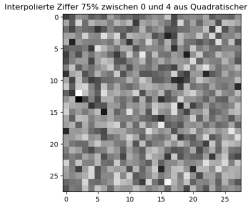
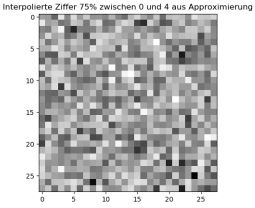
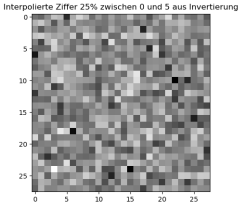
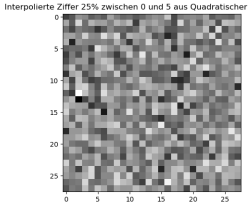
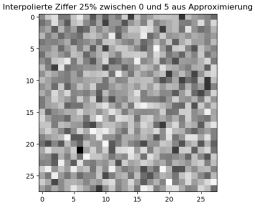
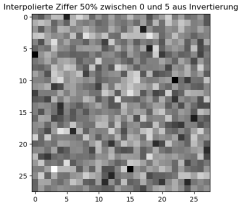
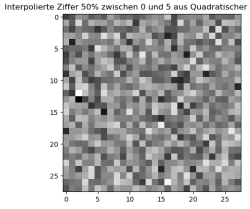
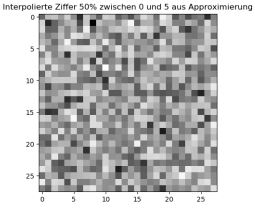
A.4.1 Ideale Ziffern

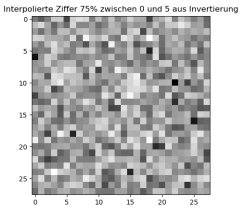
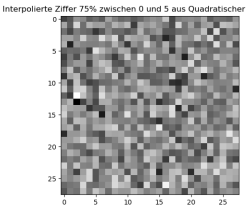
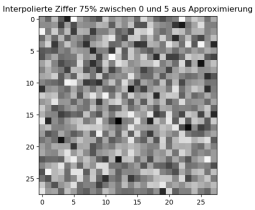
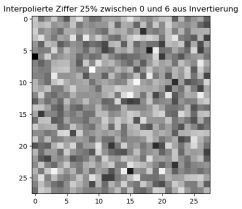
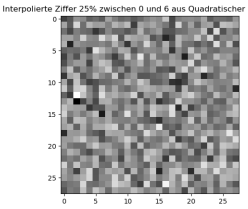
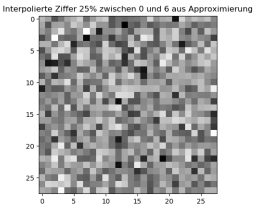
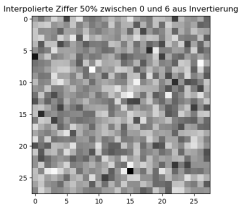
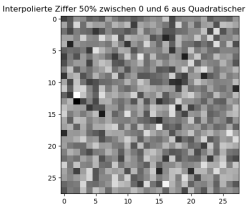
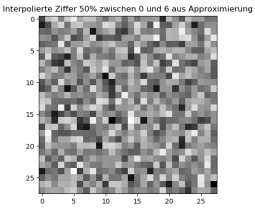
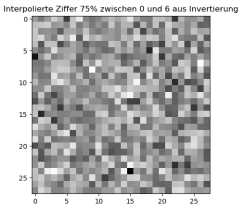
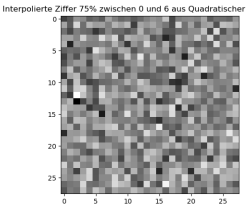
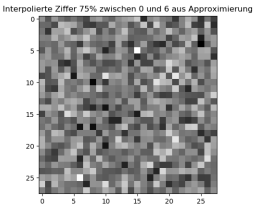
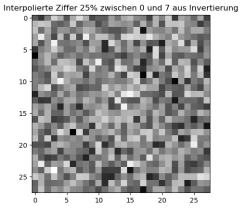
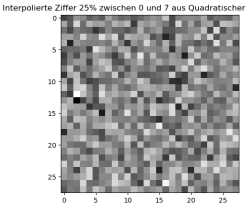
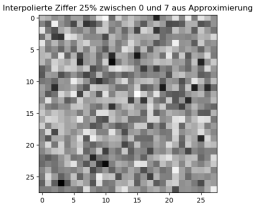
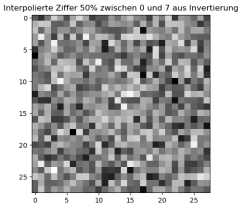
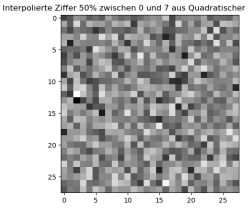
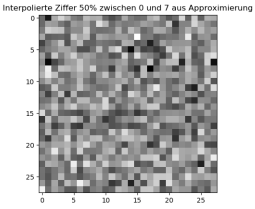
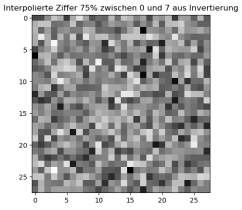
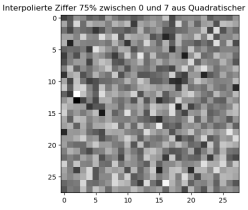
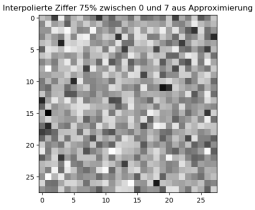
Ziffer	Invertierung	Quadratische Invertierung	Approximation
0			
1			
2			
3			
4			
5			

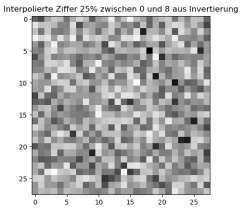
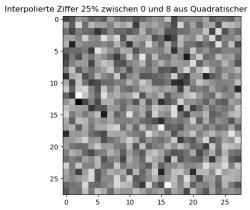
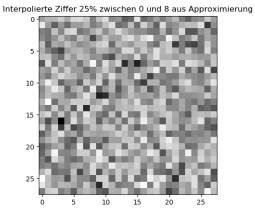
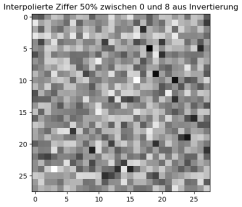
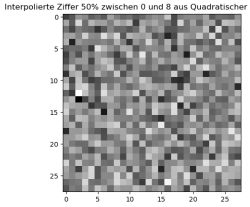
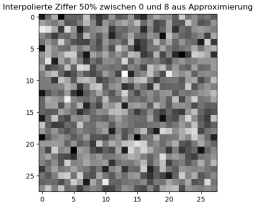
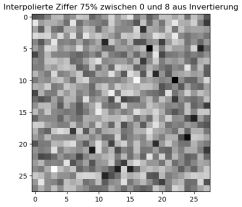
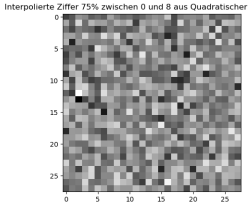
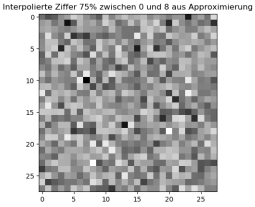
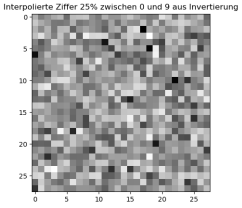
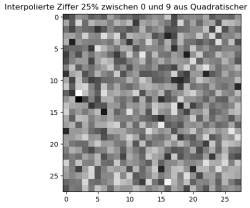
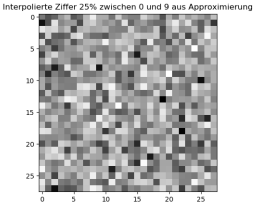
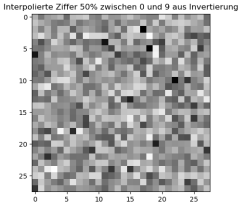
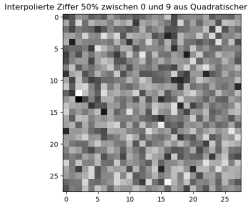
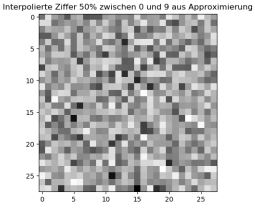
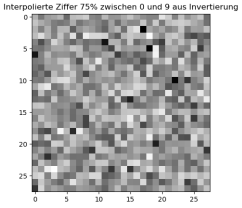
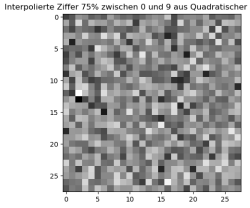
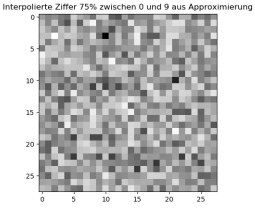
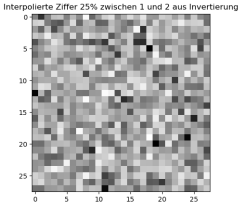
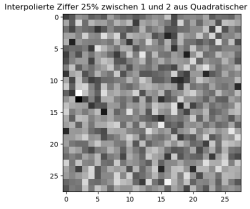
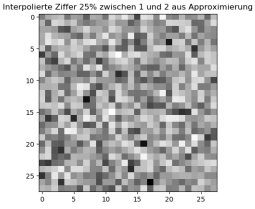
Ziffer	Invertierung	Quadratische Invertierung	Approximation
6			
7			
8			
9			

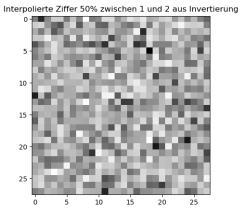
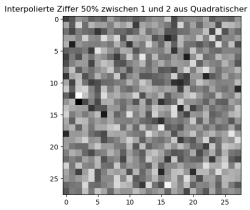
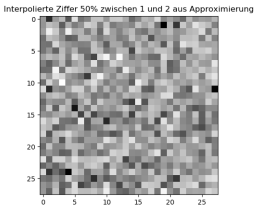
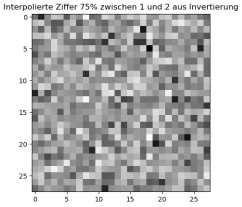
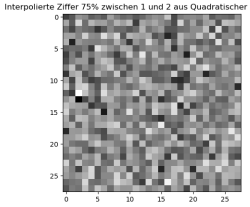
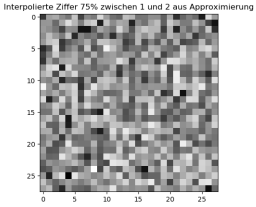
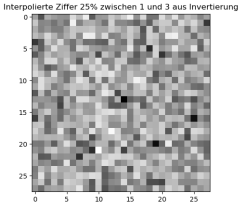
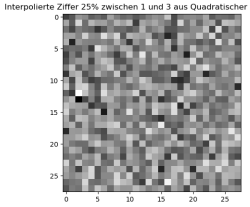
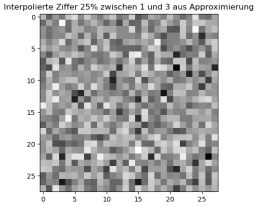
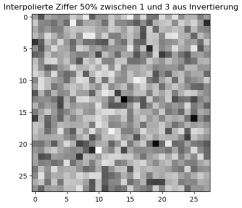
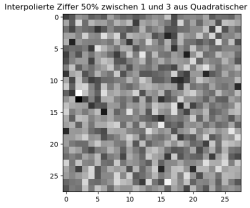
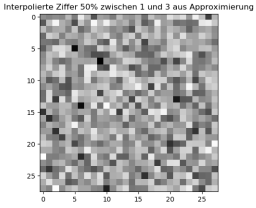
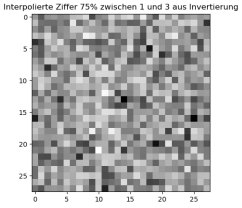
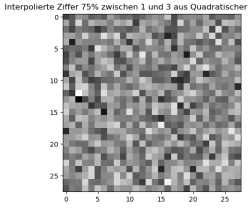
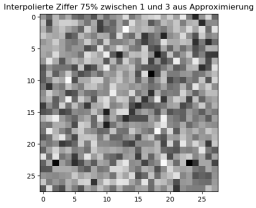
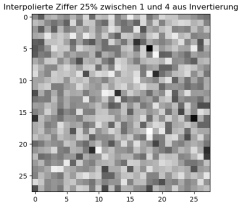
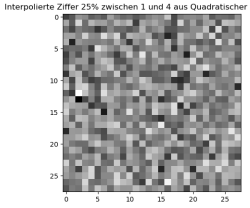
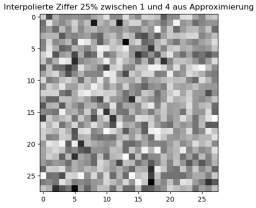
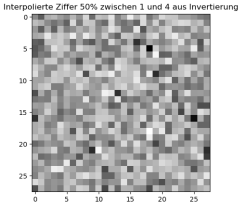
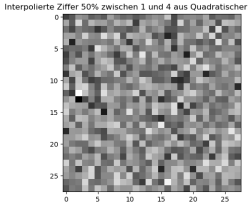
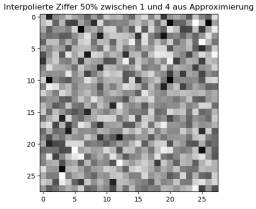
A.4.2 Interpolierte Ziffern

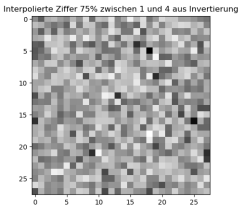
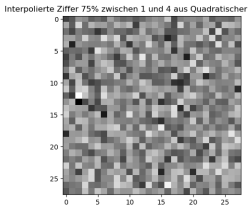
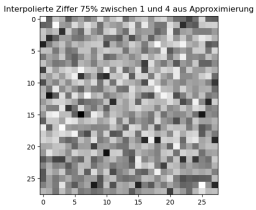
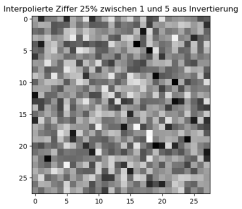
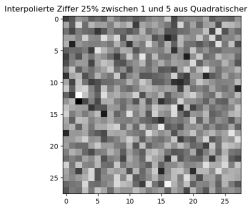
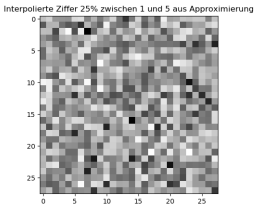
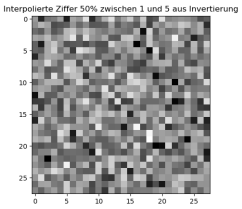
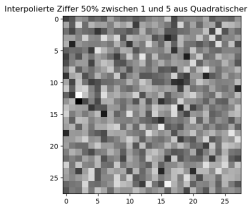
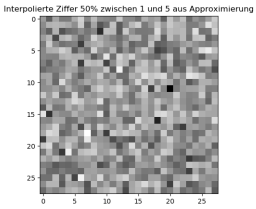
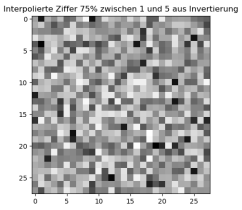
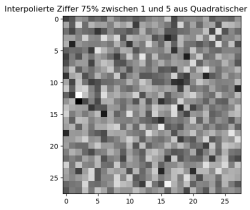
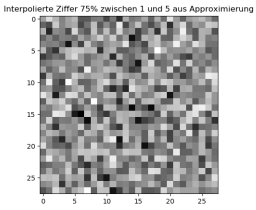
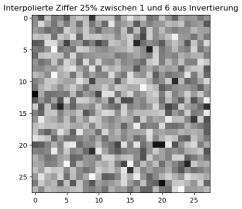
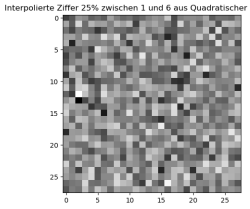
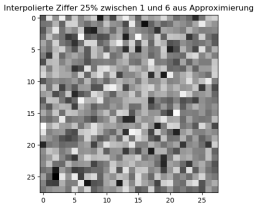
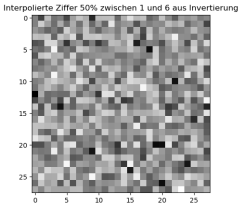
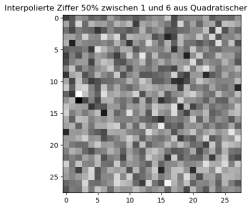
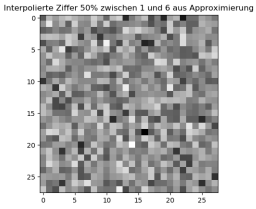
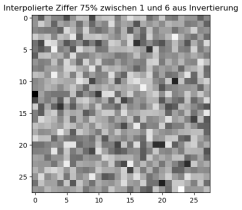
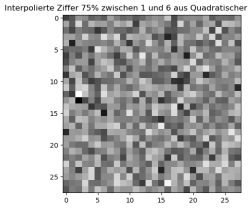
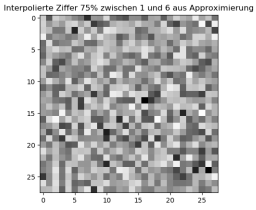
Interpolation	Invertierung	Quadratische Invertierung	Approximation
0 -> 1: 25 %			
0 -> 1: 50 %			
0 -> 1: 75 %			
0 -> 2: 25 %			
0 -> 2: 50 %			
0 -> 2: 75 %			
0 -> 3: 25 %			

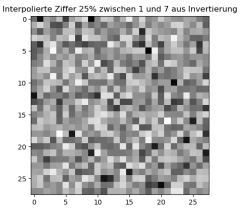
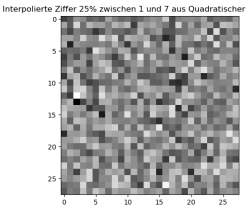
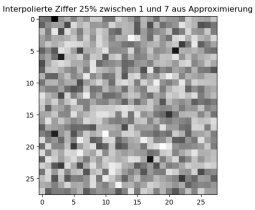
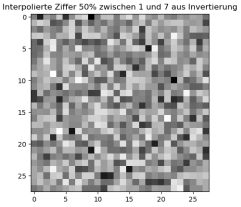
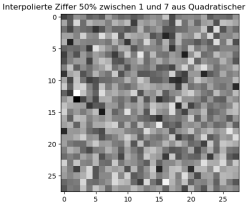
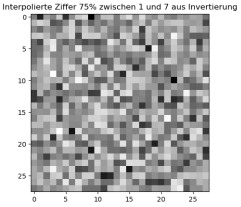
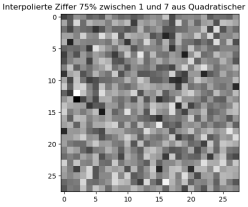
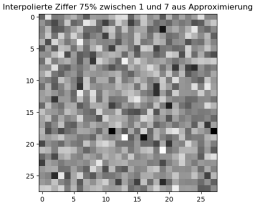
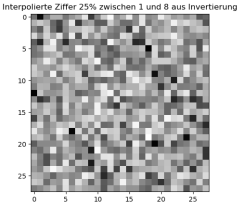
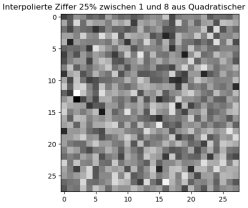
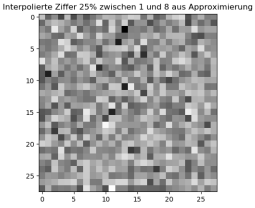
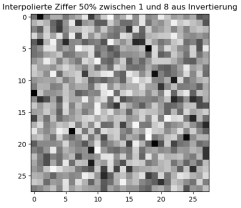
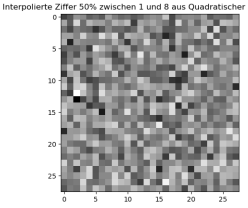
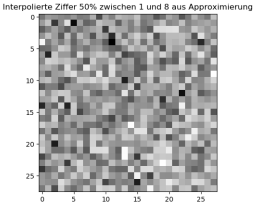
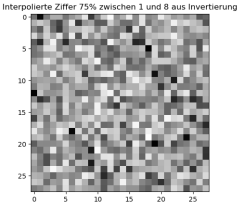
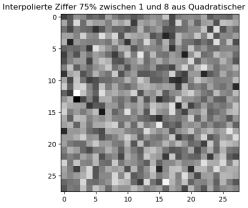
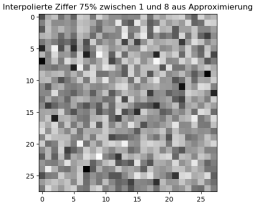
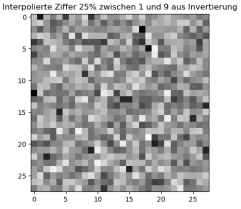
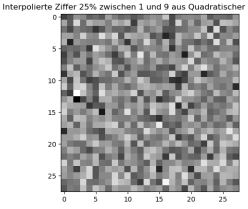
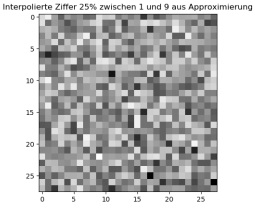
Interpolation	Invertierung	Quadratische Invertierung	Approximation
0 -> 3: 50 %			
0 -> 3: 75 %			
0 -> 4: 25 %			
0 -> 4: 50 %			
0 -> 4: 75 %			
0 -> 5: 25 %			
0 -> 5: 50 %			

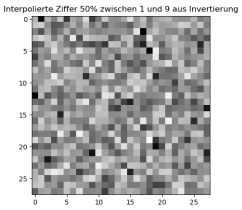
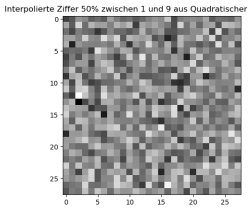
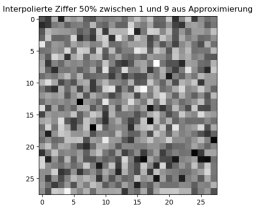
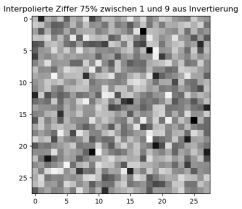
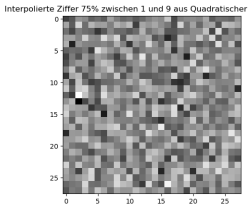
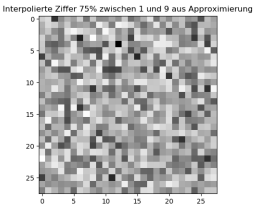
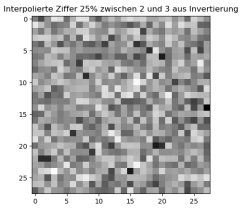
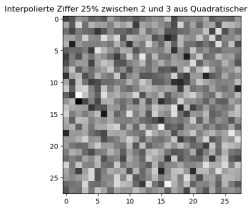
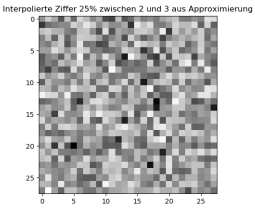
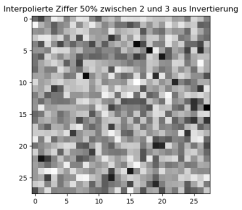
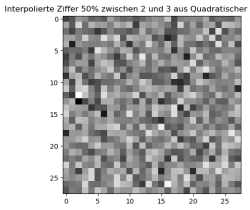
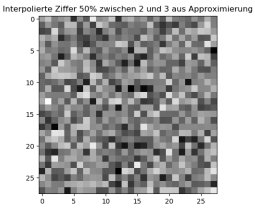
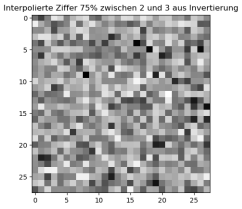
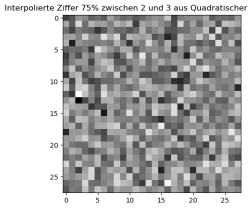
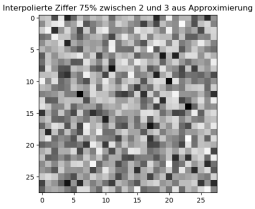
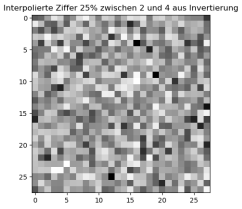
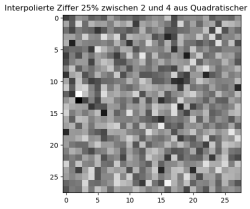
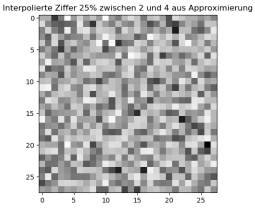
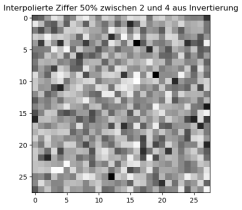
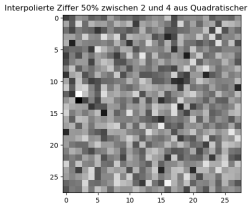
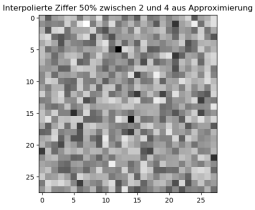
Interpolation	Invertierung	Quadratische Invertierung	Approximation
0 -> 5: 75 %			
0 -> 6: 25 %			
0 -> 6: 50 %			
0 -> 6: 75 %			
0 -> 7: 25 %			
0 -> 7: 50 %			
0 -> 7: 75 %			

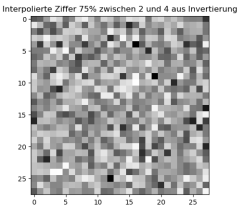
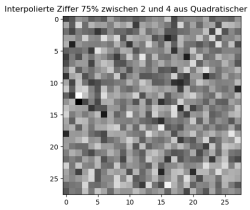
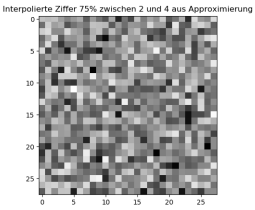
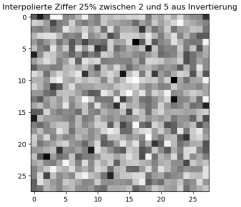
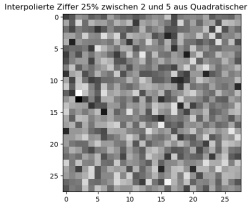
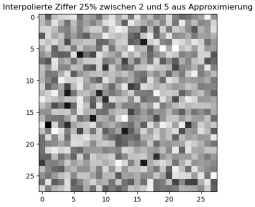
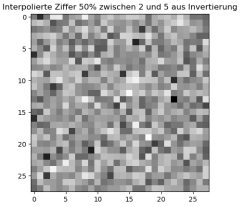
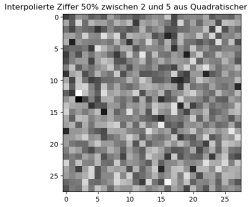
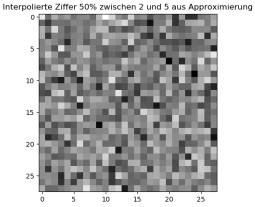
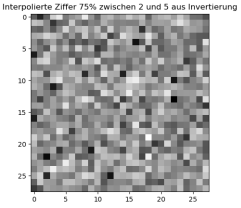
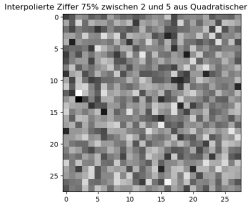
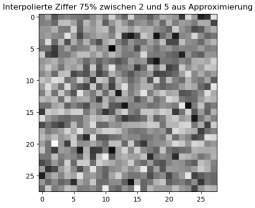
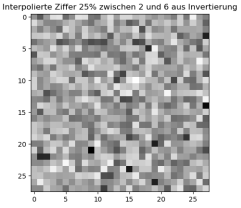
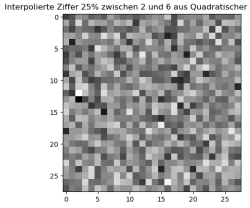
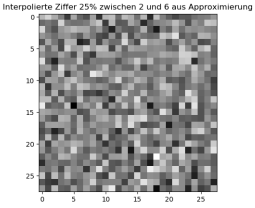
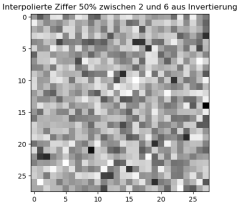
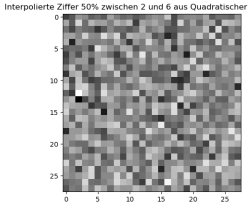
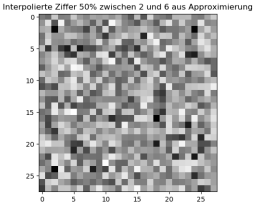
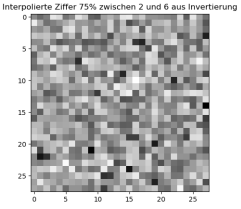
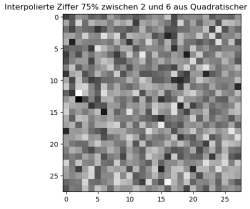
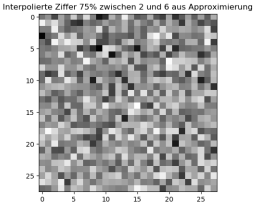
Interpolation	Invertierung	Quadratische Invertierung	Approximation
0 -> 8: 25 %			
0 -> 8: 50 %			
0 -> 8: 75 %			
0 -> 9: 25 %			
0 -> 9: 50 %			
0 -> 9: 75 %			
1 -> 2: 25 %			

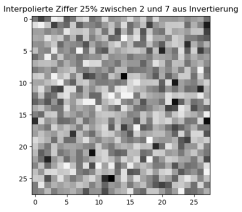
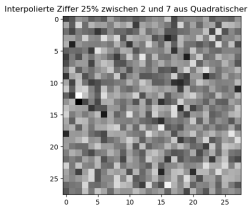
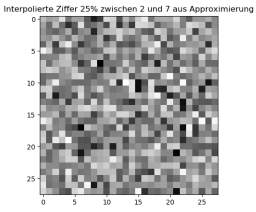
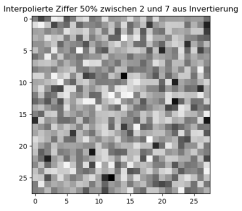
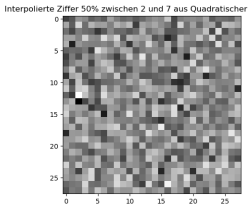
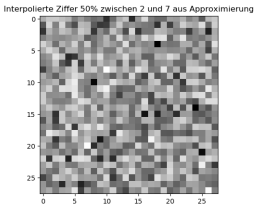
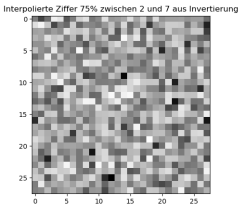
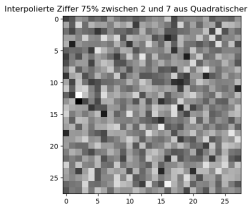
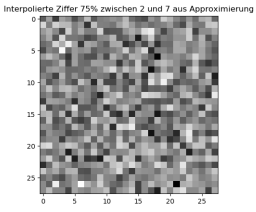
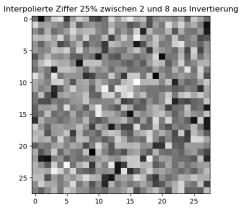
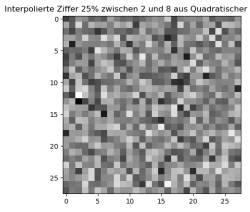
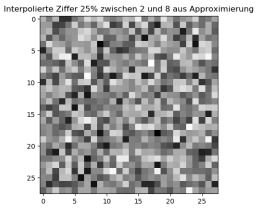
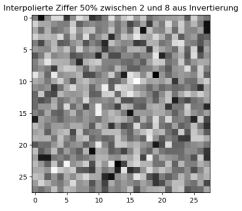
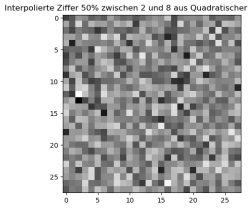
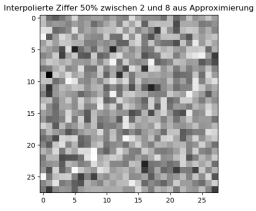
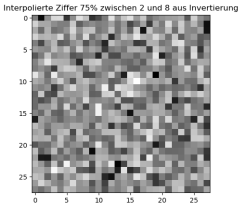
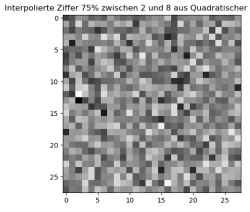
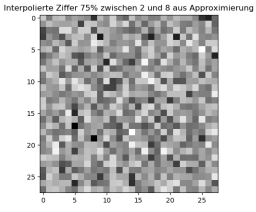
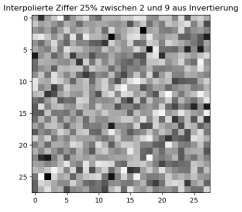
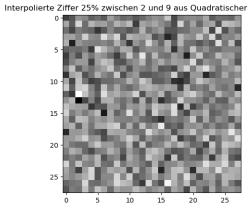
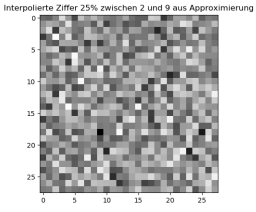
Interpolation	Invertierung	Quadratische Invertierung	Approximation
1 -> 2: 50 %			
1 -> 2: 75 %			
1 -> 3: 25 %			
1 -> 3: 50 %			
1 -> 3: 75 %			
1 -> 4: 25 %			
1 -> 4: 50 %			

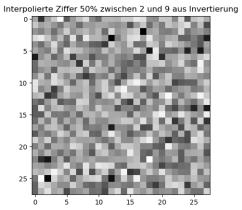
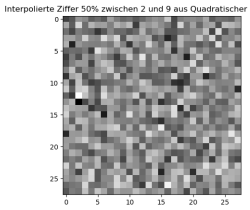
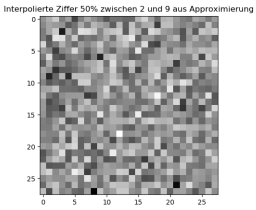
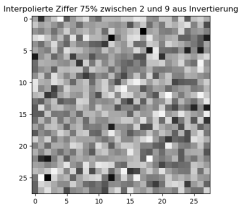
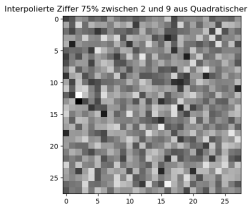
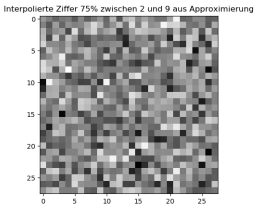
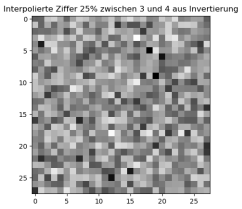
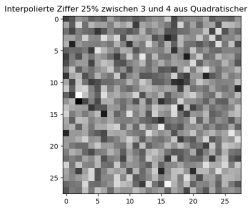
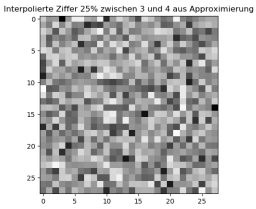
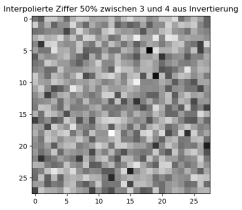
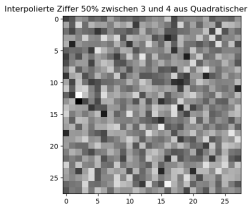
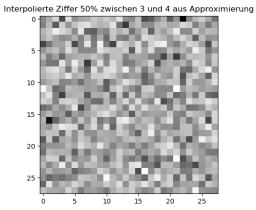
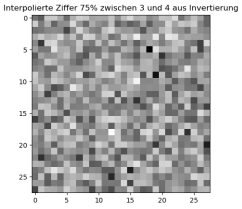
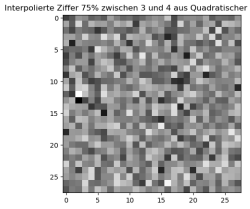
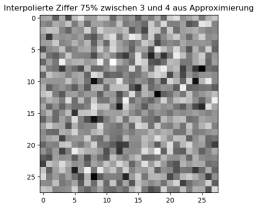
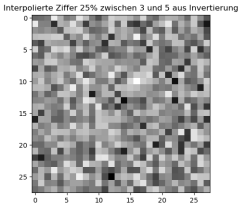
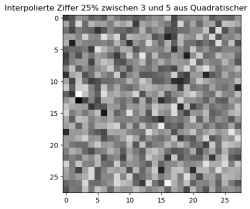
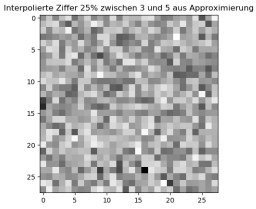
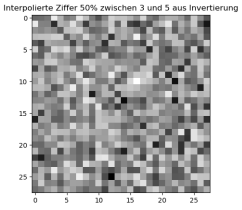
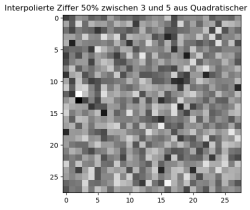
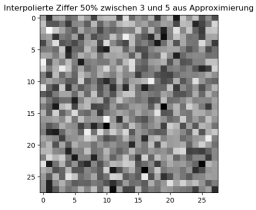
Interpolation	Invertierung	Quadratische Invertierung	Approximation
1 -> 4: 75 %			
1 -> 5: 25 %			
1 -> 5: 50 %			
1 -> 5: 75 %			
1 -> 6: 25 %			
1 -> 6: 50 %			
1 -> 6: 75 %			

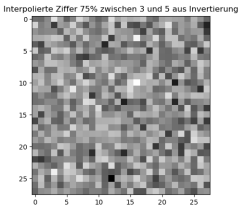
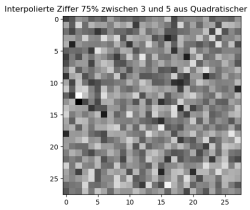
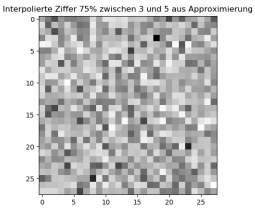
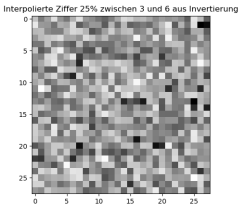
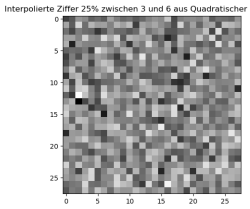
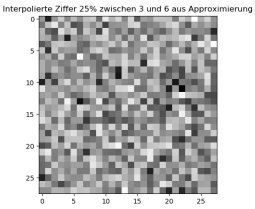
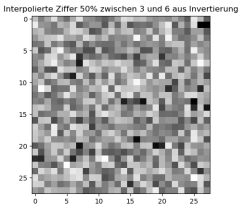
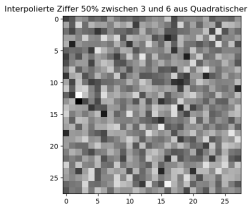
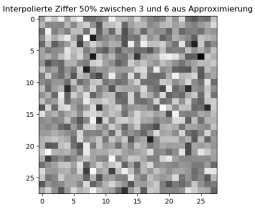
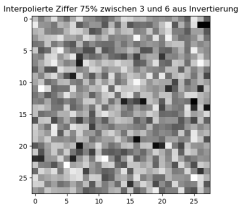
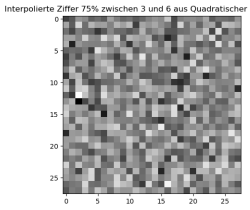
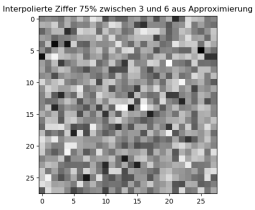
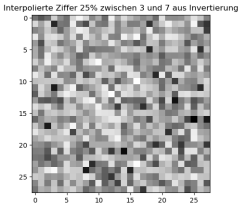
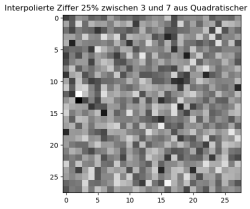
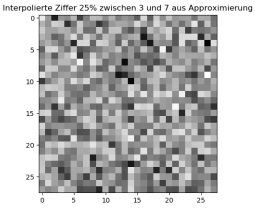
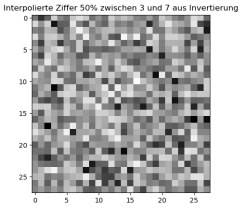
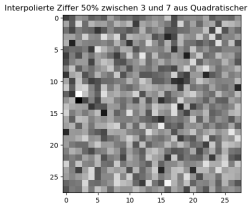
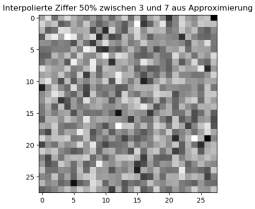
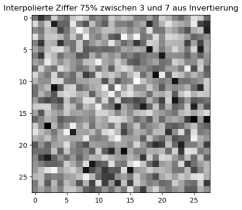
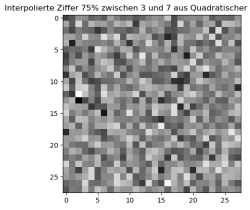
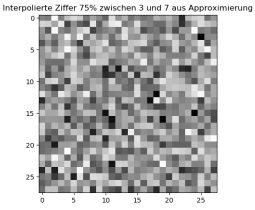
Interpolation	Invertierung	Quadratische Invertierung	Approximation
1 -> 7: 25 %			
1 -> 7: 50 %			
1 -> 7: 75 %			
1 -> 8: 25 %			
1 -> 8: 50 %			
1 -> 8: 75 %			
1 -> 9: 25 %			

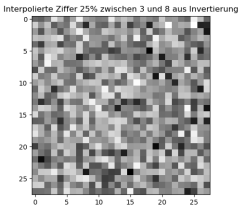
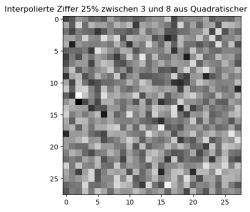
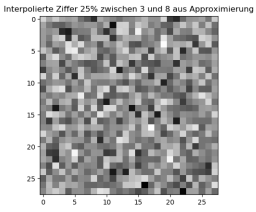
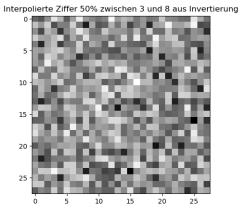
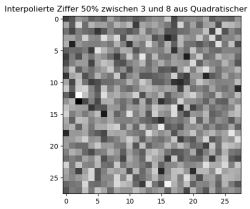
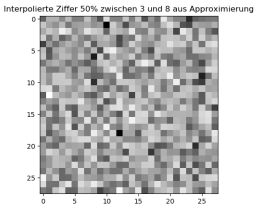
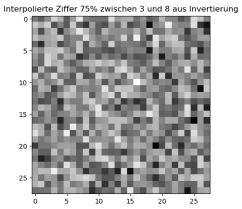
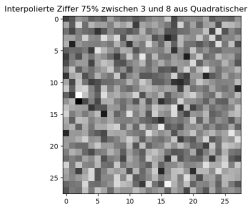
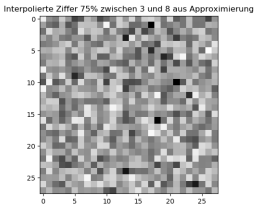
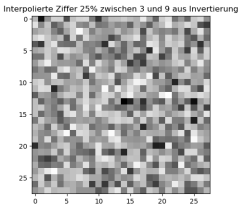
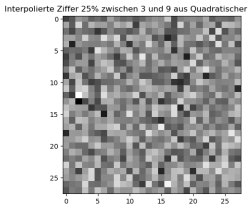
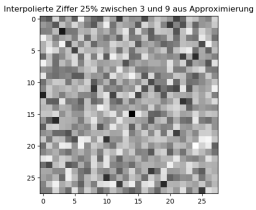
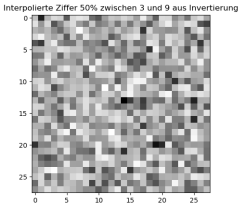
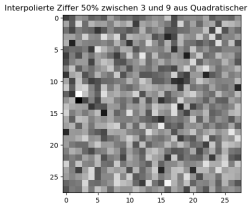
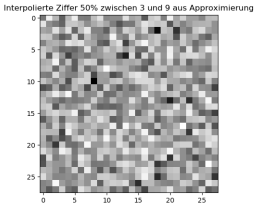
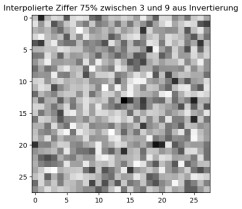
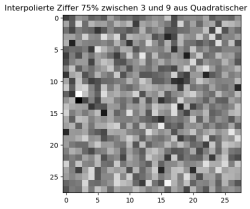
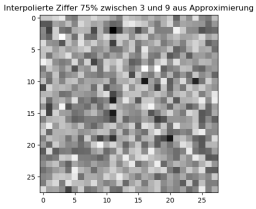
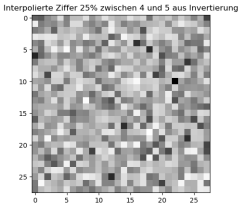
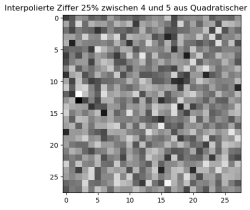
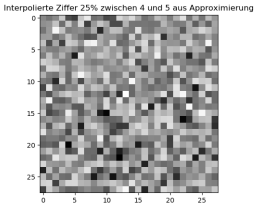
Interpolation	Invertierung	Quadratische Invertierung	Approximation
1 -> 9: 50 %			
1 -> 9: 75 %			
2 -> 3: 25 %			
2 -> 3: 50 %			
2 -> 3: 75 %			
2 -> 4: 25 %			
2 -> 4: 50 %			

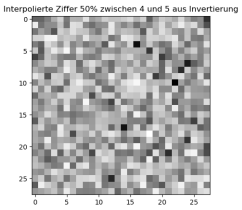
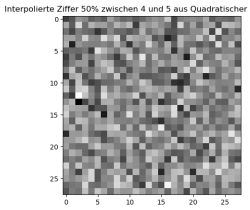
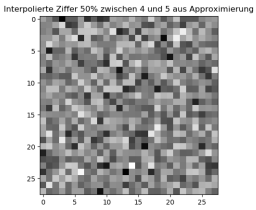
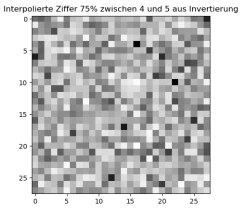
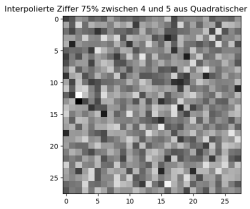
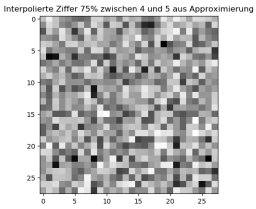
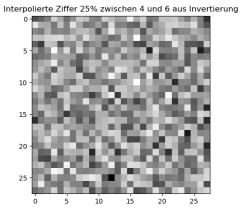
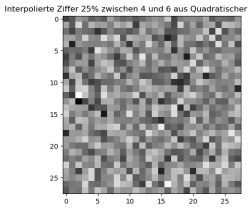
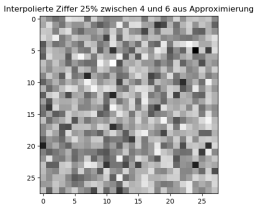
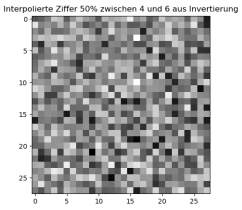
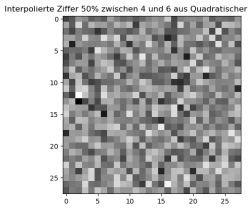
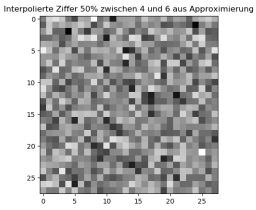
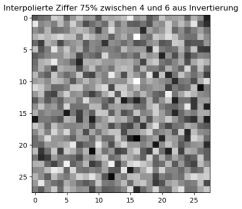
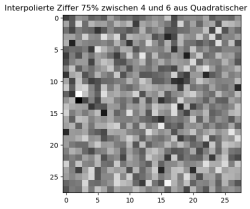
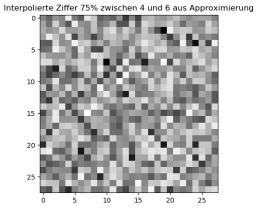
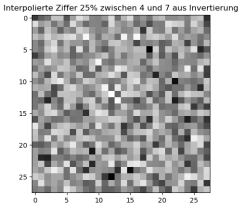
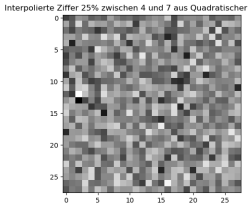
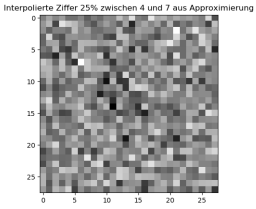
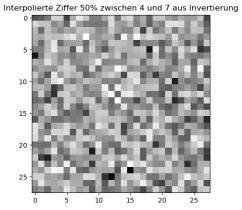
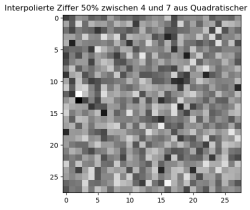
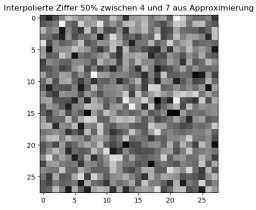
Interpolation	Invertierung	Quadratische Invertierung	Approximation
2 -> 4: 75 %			
2 -> 5: 25 %			
2 -> 5: 50 %			
2 -> 5: 75 %			
2 -> 6: 25 %			
2 -> 6: 50 %			
2 -> 6: 75 %			

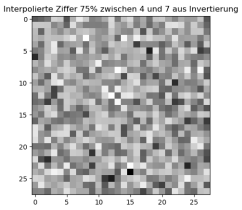
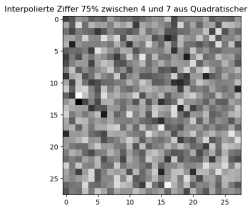
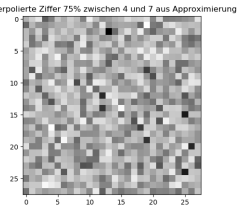
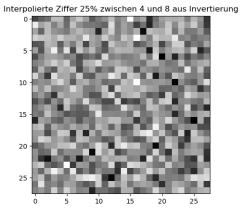
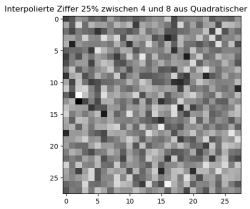
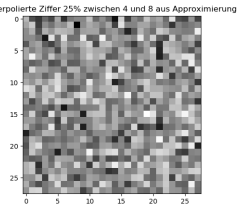
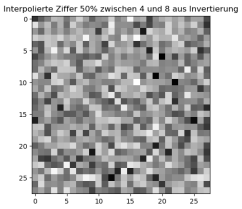
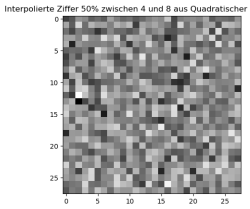
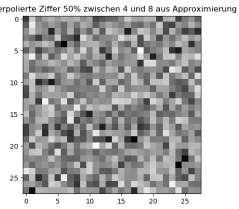
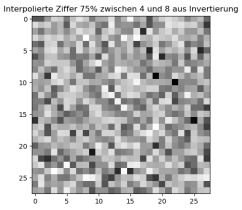
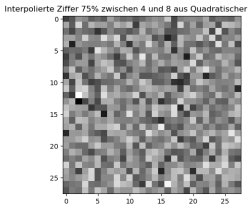
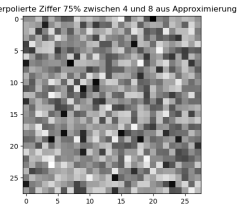
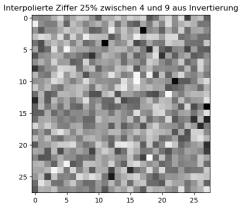
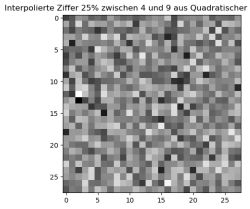
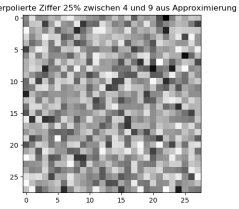
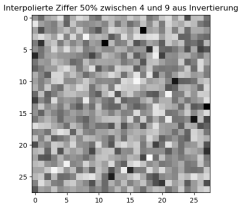
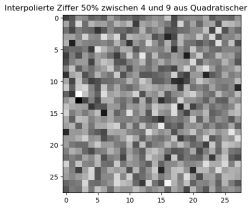
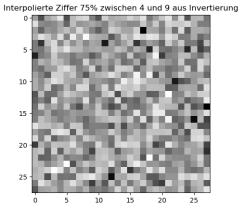
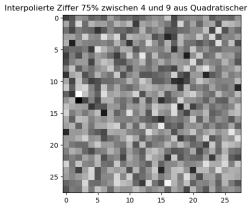
Interpolation	Invertierung	Quadratische Invertierung	Approximation
2 -> 7: 25 %			
2 -> 7: 50 %			
2 -> 7: 75 %			
2 -> 8: 25 %			
2 -> 8: 50 %			
2 -> 8: 75 %			
2 -> 9: 25 %			

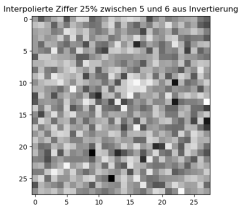
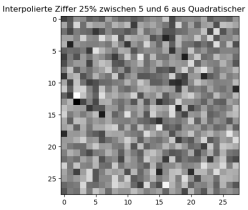
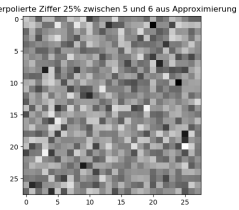
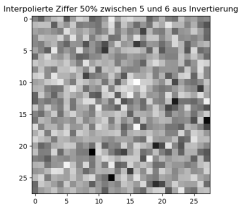
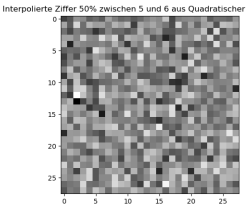
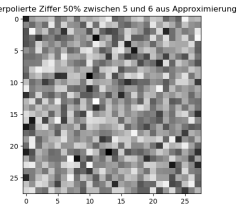
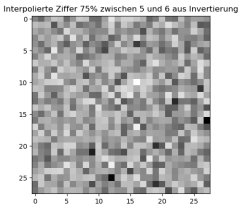
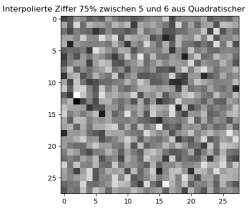
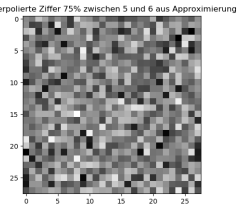
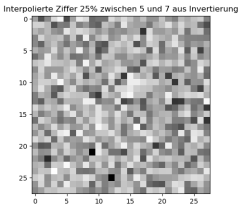
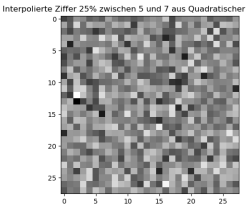
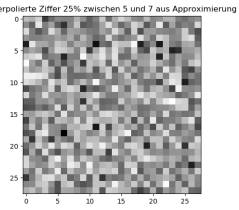
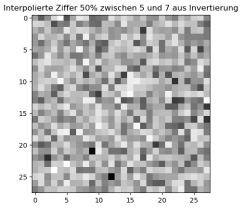
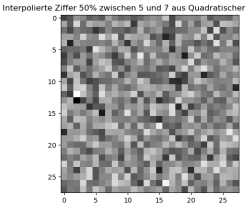
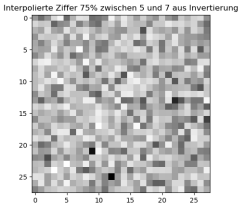
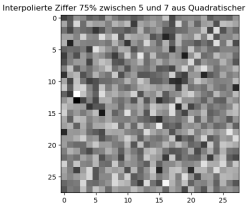
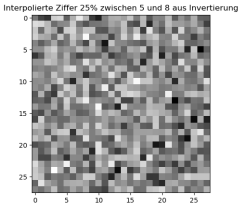
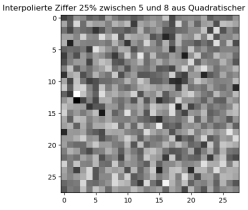
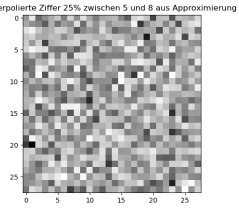
Interpolation	Invertierung	Quadratische Invertierung	Approximation
2 -> 9: 50 %			
2 -> 9: 75 %			
3 -> 4: 25 %			
3 -> 4: 50 %			
3 -> 4: 75 %			
3 -> 5: 25 %			
3 -> 5: 50 %			

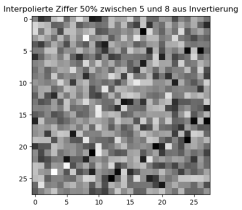
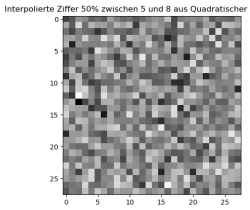
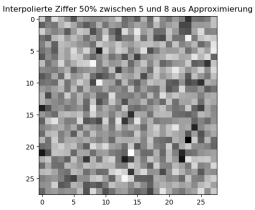
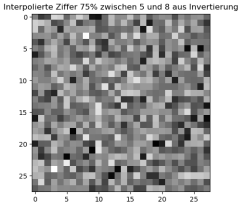
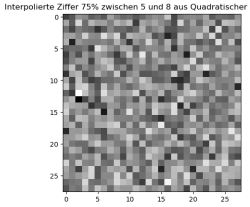
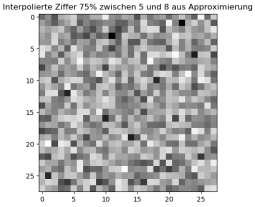
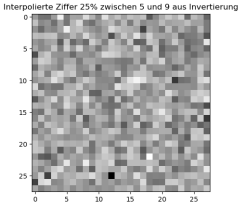
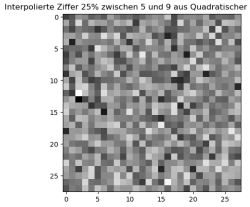
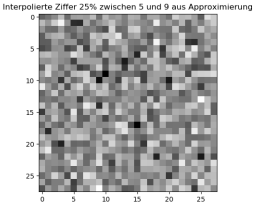
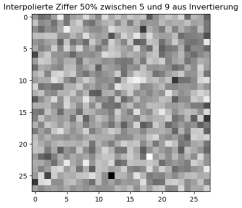
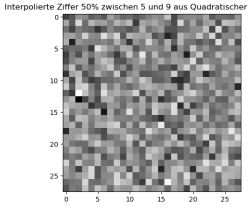
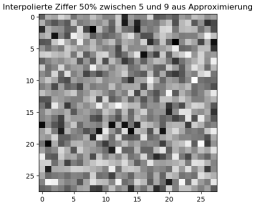
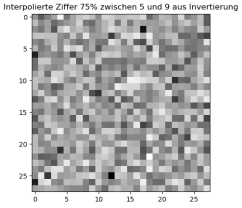
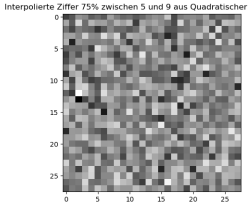
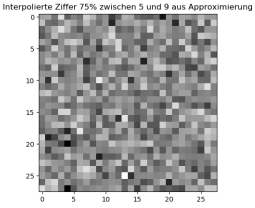
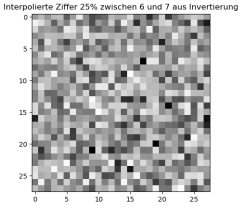
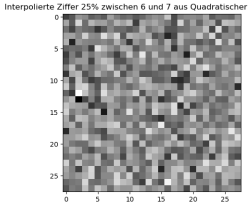
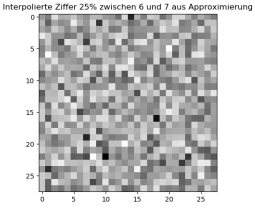
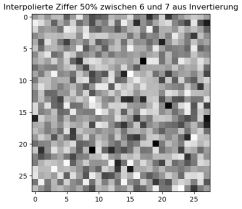
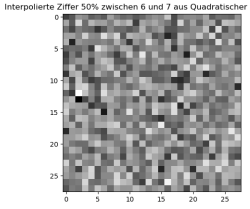
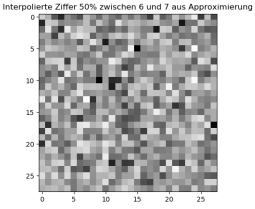
Interpolation	Invertierung	Quadratische Invertierung	Approximation
3 -> 5: 75 %			
3 -> 6: 25 %			
3 -> 6: 50 %			
3 -> 6: 75 %			
3 -> 7: 25 %			
3 -> 7: 50 %			
3 -> 7: 75 %			

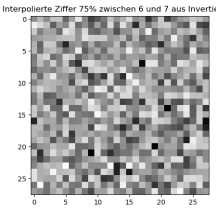
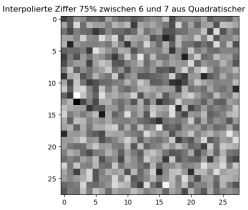
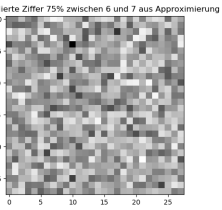
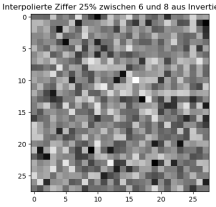
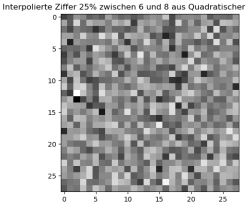
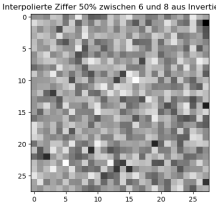
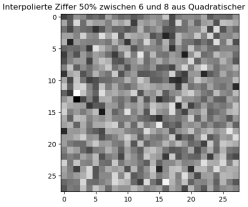
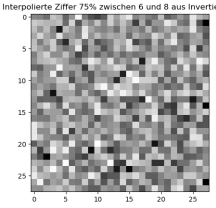
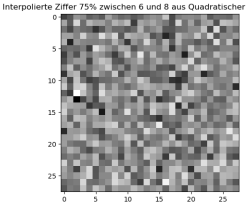
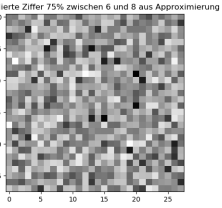
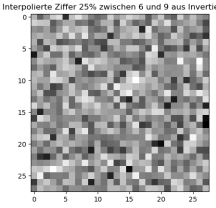
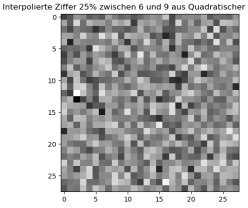
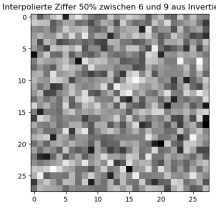
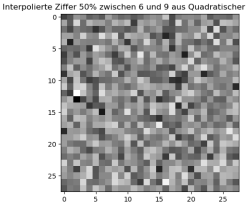
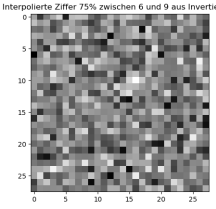
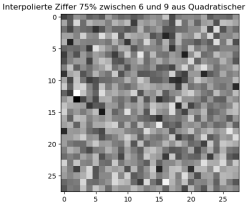
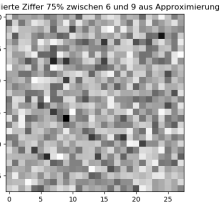
Interpolation	Invertierung	Quadratische Invertierung	Approximation
3 -> 8: 25 %			
3 -> 8: 50 %			
3 -> 8: 75 %			
3 -> 9: 25 %			
3 -> 9: 50 %			
3 -> 9: 75 %			
4 -> 5: 25 %			

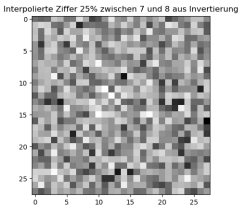
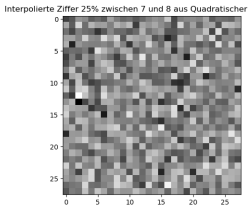
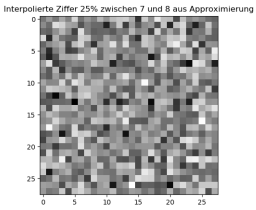
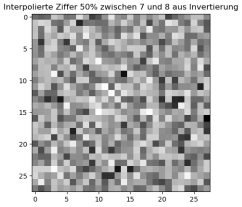
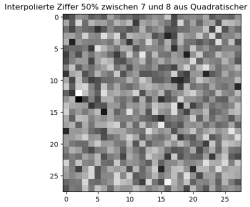
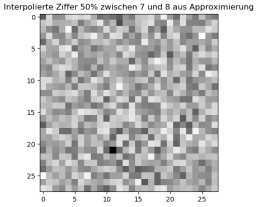
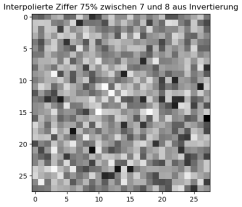
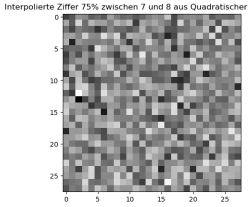
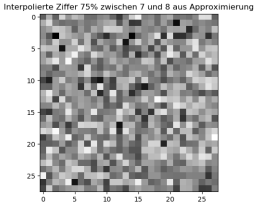
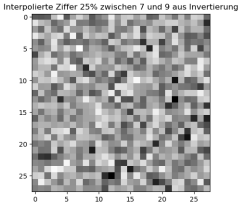
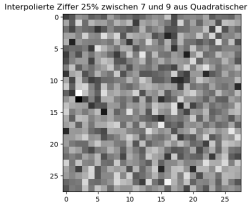
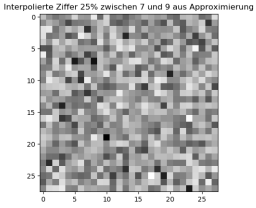
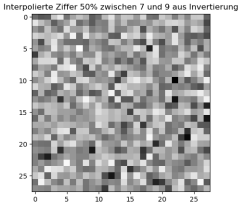
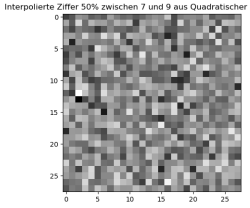
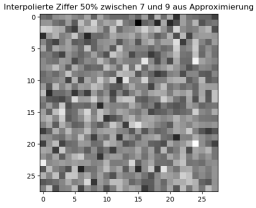
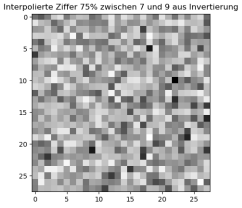
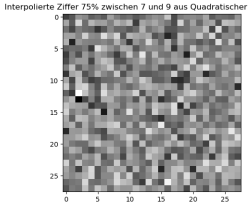
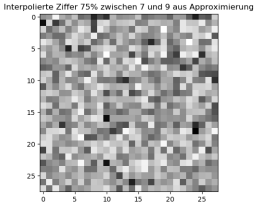
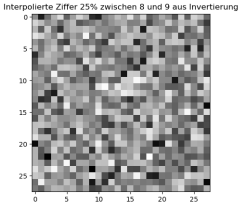
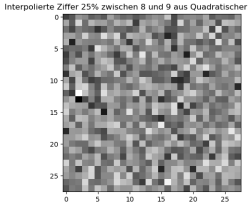
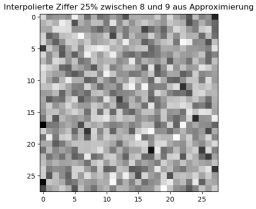
Interpolation	Invertierung	Quadratische Invertierung	Approximation
4 -> 5: 50 %			
4 -> 5: 75 %			
4 -> 6: 25 %			
4 -> 6: 50 %			
4 -> 6: 75 %			
4 -> 7: 25 %			
4 -> 7: 50 %			

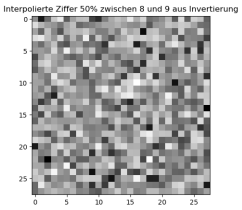
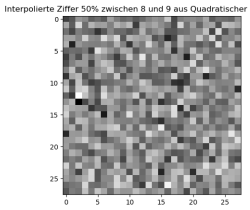
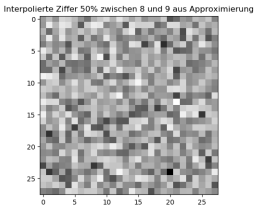
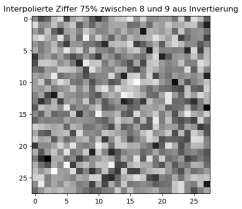
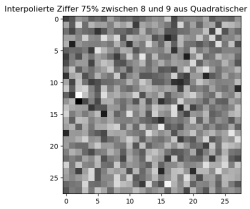
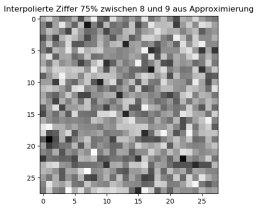
Interpolation	Invertierung	Quadratische Invertierung	Approximation
4 -> 7: 75 %			
4 -> 8: 25 %			
4 -> 8: 50 %			
4 -> 8: 75 %			
4 -> 9: 25 %			
4 -> 9: 50 %			
4 -> 9: 75 %			

Interpolation	Invertierung	Quadratische Invertierung	Approximation
5 -> 6: 25 %			
5 -> 6: 50 %			
5 -> 6: 75 %			
5 -> 7: 25 %			
5 -> 7: 50 %			
5 -> 7: 75 %			
5 -> 8: 25 %			

Interpolation	Invertierung	Quadratische Invertierung	Approximation
5 -> 8: 50 %			
5 -> 8: 75 %			
5 -> 9: 25 %			
5 -> 9: 50 %			
5 -> 9: 75 %			
6 -> 7: 25 %			
6 -> 7: 50 %			

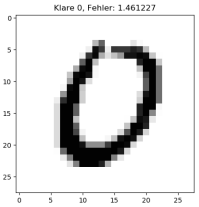
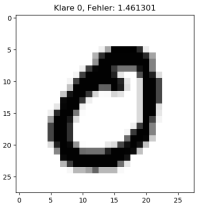
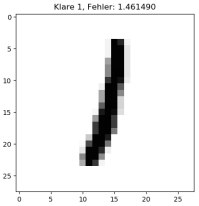
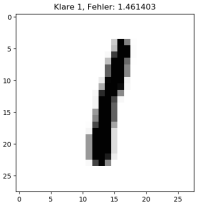
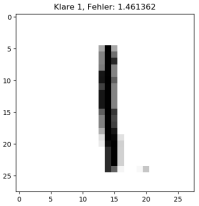
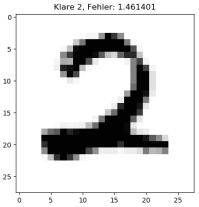
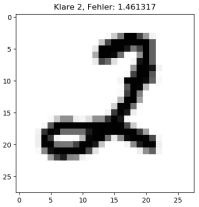
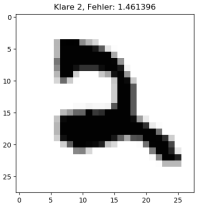
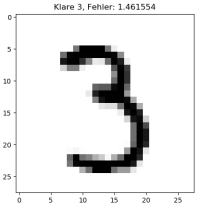
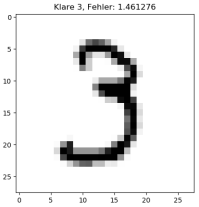
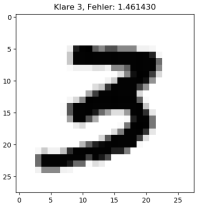
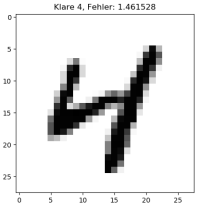
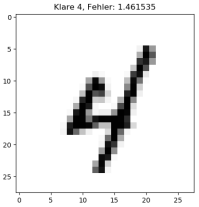
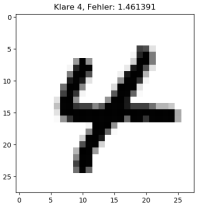
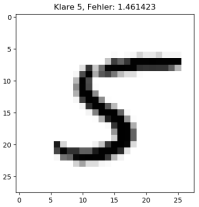
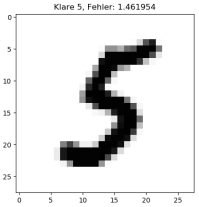
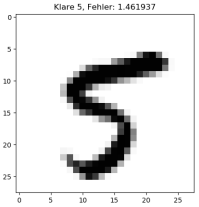
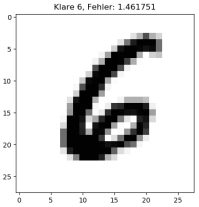
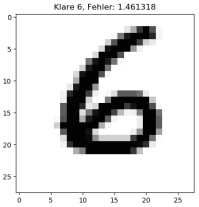
Interpolation	Invertierung	Quadratische Invertierung	Approximation
6 -> 7: 75 %			
6 -> 8: 25 %			
6 -> 8: 50 %			
6 -> 8: 75 %			
6 -> 9: 25 %			
6 -> 9: 50 %			
6 -> 9: 75 %			

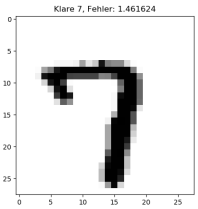
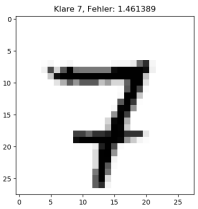
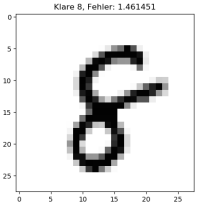
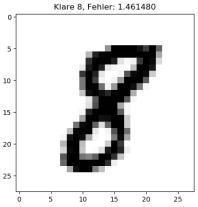
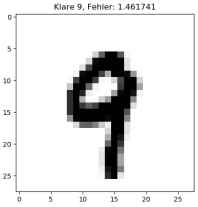
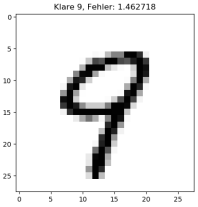
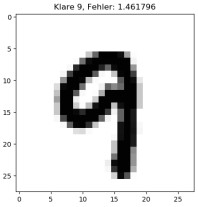
Interpolation	Invertierung	Quadratische Invertierung	Approximation
7 -> 8: 25 %			
7 -> 8: 50 %			
7 -> 8: 75 %			
7 -> 9: 25 %			
7 -> 9: 50 %			
7 -> 9: 75 %			
8 -> 9: 25 %			

Interpolation	Invertierung	Quadratische Invertierung	Approximation
8 -> 9: 50 %			
8 -> 9: 75 %			

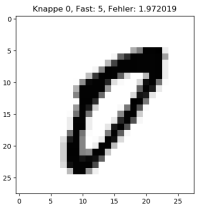
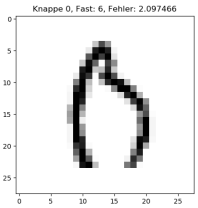
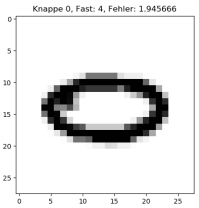
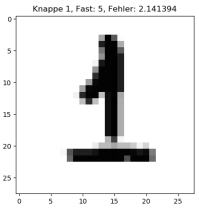
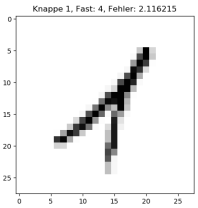
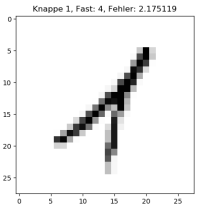
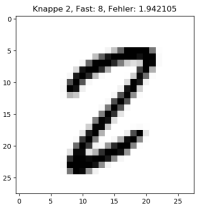
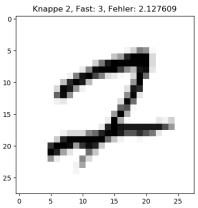
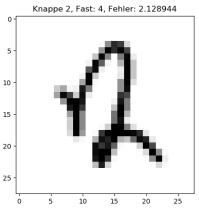
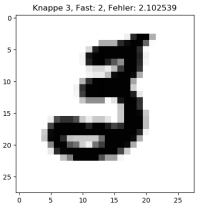
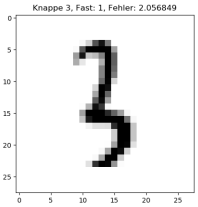
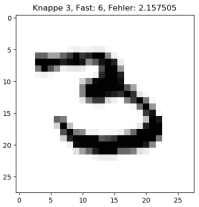
A.5 Manuelle Untersuchung

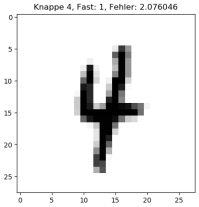
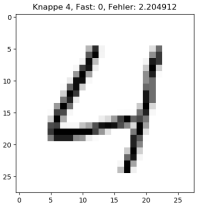
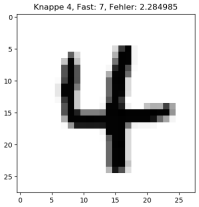
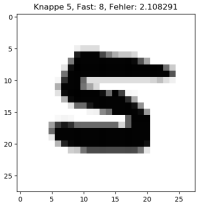
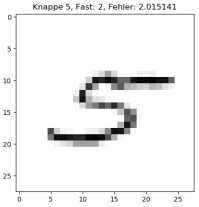
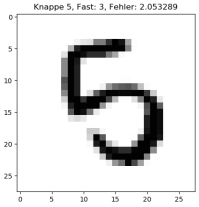
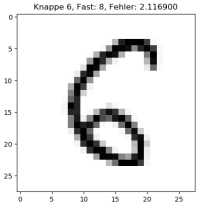
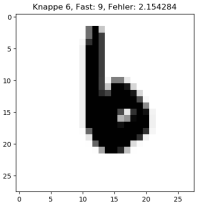
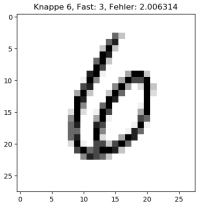
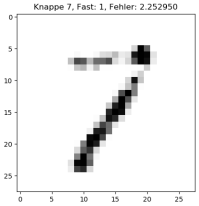
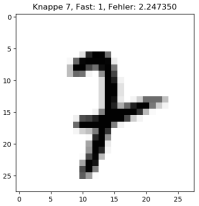
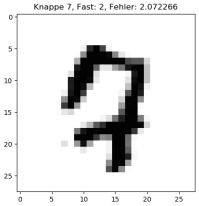
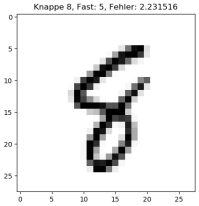
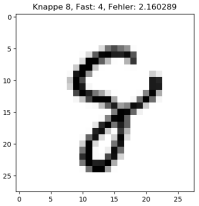
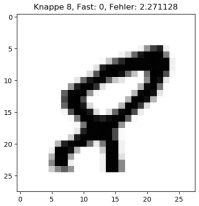
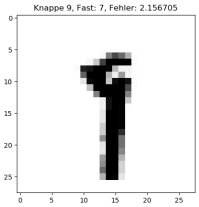
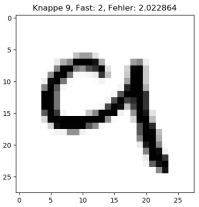
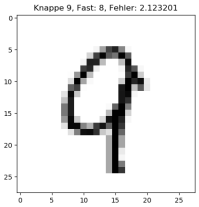
A.5.1 Gut erkannte Ziffern

Ziffer	Resultat 1	Resultat 2	Resultat 3
0			
1			
2			
3			
4			
5			
6			

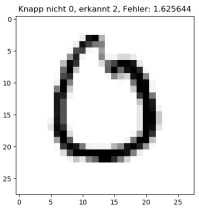
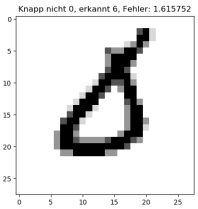
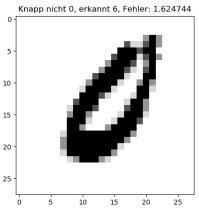
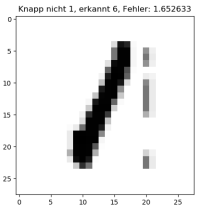
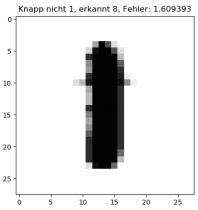
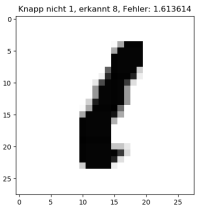
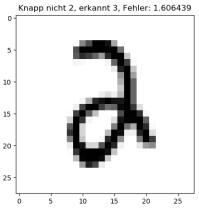
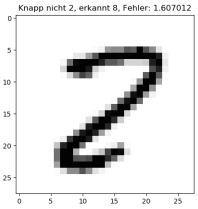
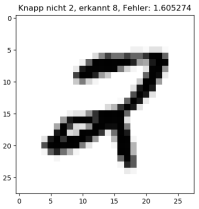
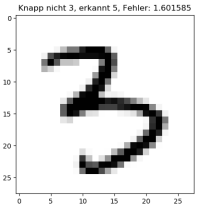
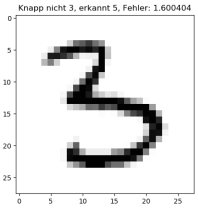
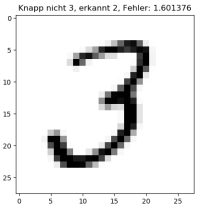
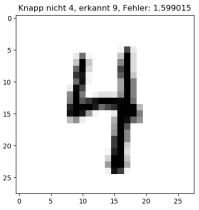
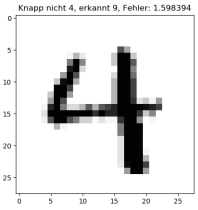
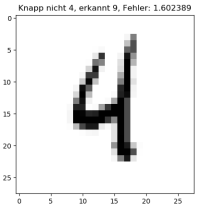
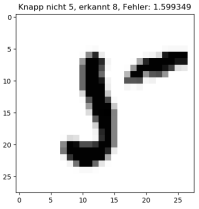
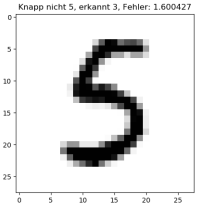
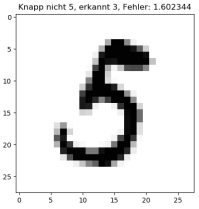
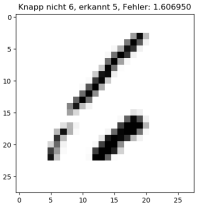
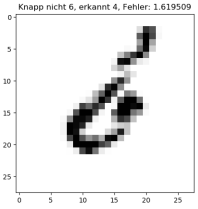
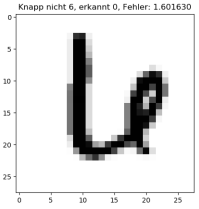
Ziffer	Resultat 1	Resultat 2	Resultat 3
7			
8			
9			

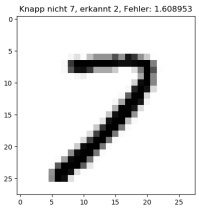
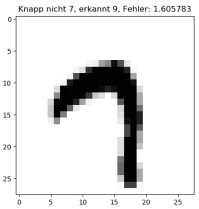
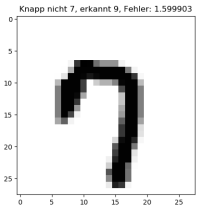
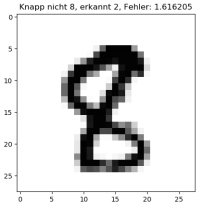
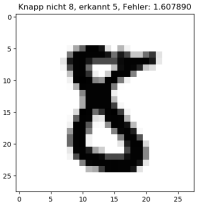
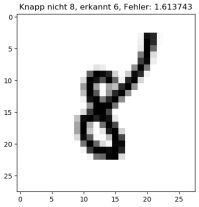
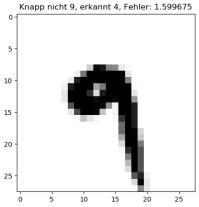
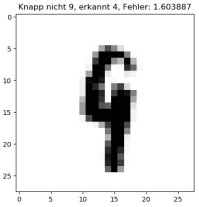
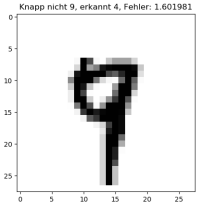
A.5.2 Knapp erkannte Ziffern

Ziffer	Resultat 1	Resultat 2	Resultat 3
0			
1			
2			
3			

Ziffer	Resultat 1	Resultat 2	Resultat 3
4			
5			
6			
7			
8			
9			

A.5.3 Knapp nicht erkannte Ziffern

Ziffer	Resultat 1	Resultat 2	Resultat 3
0			
1			
2			
3			
4			
5			
6			

Ziffer	Resultat 1	Resultat 2	Resultat 3
7			
8			
9			

B Anhang: Code

B.1 Neuronale Netze

B.1.1 Mnist Network

```
import gzip
import math
import os
import urllib.request
from typing import List, Tuple

import numpy as np

from . import FeedForwardNeuronalNetwork, NeuronalNetwork


class MnistNetwork(NeuronalNetwork):
    layers: List[int]
    learning_rate: float
    work_dir: str
    label_count: int

    network: FeedForwardNeuronalNetwork
    image_size: int
    train_data: np.ndarray
    test_data: np.ndarray

    MNIST_URL: str = 'http://yann.lecun.com/exdb/mnist/'

    def __init__(self, layers: List[int], learning_rate: float, work_dir: str, squared:
        self.layers = layers
        self.learning_rate = learning_rate
        self.work_dir = work_dir
        self.network = None
        self.image_size = 0
        self.train_data = None
        self.test_data = None
        self.label_count = 28 * 28 if squared else 10

    def train(self, epochs: int, batch_size: int) -> float:
        if self.train_data is None:
            self.train_data = self.load_train_data()
        if self.test_data is None:
            self.test_data = self.load_test_data()

        if self.network is None:
            self.network = FeedForwardNeuronalNetwork([self.image_size] + self.layers +

        train_data, eval_data = self.split_data_set(self.train_data[0], 0.9)
        train_label, eval_label = self.split_data_set(self.train_data[1], 0.9)

        self.network.train(train_data, train_label, epochs, batch_size, eval_data, eval_
        return self.network.evaluate(self.test_data[0], self.test_data[1])

    def feed_forward(self, x: np.ndarray) -> np.ndarray:
        return self.network.feed_forward(x)

    def get_label_count(self) -> int:
```

```
        return 10

    def split_data_set(self, data_set: np.ndarray, ratio: float) -> Tuple[np.ndarray, np
        size = len(data_set)
        indeces = range(size)
        delim = int(math.floor(size * ratio))
        p1 = np.array([data_set[i] for i in indeces[:delim]])
        p2 = np.array([data_set[i] for i in indeces[delim:]])
        return p1, p2

    def load_train_data(self) -> Tuple[np.ndarray, np.ndarray]:
        images = self.extract_images(self.download_file_if_needed('train-images-idx3-uby
        labels = self.extract_labels(self.download_file_if_needed('train-labels-idx1-uby
        return images, labels

    def load_test_data(self) -> Tuple[np.ndarray, np.ndarray]:
        images = self.extract_images(self.download_file_if_needed('t10k-images-idx3-ubyt
        labels = self.extract_labels(self.download_file_if_needed('t10k-labels-idx1-ubyt
        return images, labels

    def download_file_if_needed(self, file: str) -> str:
        if not os.path.exists(self.work_dir):
            os.mkdir(self.work_dir)
        file_path = os.path.join(self.work_dir, file)
        if not os.path.exists(file_path):
            print("Downloading " + file)
            urllib.request.urlretrieve(self.MNIST_URL + file, file_path)
        else:
            print(file + " already downloaded")
        return file_path

    def extract_images(self, file: str) -> np.ndarray:
        with gzip.open(file) as bytestream:
            bytestream.read(4)
            nr_of_images = int.from_bytes(bytestream.read(4), byteorder='big', signed=F
            image_width = int.from_bytes(bytestream.read(4), byteorder='big', signed=Fa
            image_height = int.from_bytes(bytestream.read(4), byteorder='big', signed=F
            self.image_size = image_width * image_height

            buffer = bytestream.read(self.image_size * nr_of_images)
            images = np.frombuffer(buffer, dtype=np.uint8).astype(np.float32)
            images = images.reshape(nr_of_images, self.image_size)
            return images

    def extract_labels(self, file: str) -> np.ndarray:
        with gzip.open(file) as bytestream:
            bytestream.read(4)
            nr_of_labels = int.from_bytes(bytestream.read(4), byteorder='big', signed=F

            buffer = bytestream.read(nr_of_labels)
            labels = np.frombuffer(buffer, dtype=np.uint8)
            return (np.arange(self.label_count) == labels[:, None]).astype(np.float32)
```

B.1.2 Feed Forward Neuronal Network

```
import random

import numpy as np
import tensorflow as tf
```

```
from typing import List
from tensorflow import Tensor

from . import NeuronalNetwork


class FeedForwardNeuronalNetwork(NeuronalNetwork):
    inputs: int
    outputs: int
    x: Tensor
    y: Tensor
    w: List[Tensor]
    b: List[Tensor]
    argmax: int
    updates: any
    session: tf.Session

    def __init__(self, layers: List[int], learning_rate: float) -> None:
        self.inputs = layers[0]
        self.outputs = layers[len(layers) - 1]

        self.x = tf.placeholder(tf.float32, shape=(None, self.inputs))
        self.y = tf.placeholder(tf.float32, shape=(None, self.outputs))

        self.w = [tf.Variable(tf.random_normal(shape=(layers[i - 1], layers[i]))) for i
        self.b = [tf.Variable(tf.random_normal(shape=(1, layers[i]))) for i in range(1,

        a = self.x
        for i in range(len(self.w)):
            a = tf.nn.sigmoid(tf.add(tf.matmul(a, self.w[i]), self.b[i]))

        self.yhat = a

        self.argmax = tf.argmax(self.yhat, axis=1)

        cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=self.yha
        self.updates = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

        self.sess = tf.Session()
        init = tf.global_variables_initializer()
        self.sess.run(init)

    def get_label_count(self) -> int:
        return self.outputs

    def feed_forward(self, x) -> np.ndarray:
        return self.sess.run(self.yhat, feed_dict={self.x: x})

    def predict(self, x) -> np.ndarray:
        return self.sess.run(self.argmax, feed_dict={self.x: x})

    def train(self, x, y, epochs, batch_size, test_x=None, test_y=None) -> None:
        indices = [i for i in range(len(x))]
        for i in range(epochs):
            print("Epoch %d/%d" % (i + 1, epochs))
            random.shuffle(indices)
            for mini_batch in [indices[i:i + batch_size] for i in range(0, len(x), batch
```

```
                self.train_batch([x[i] for i in mini_batch], [y[i] for i in mini_batch])

            if test_x is not None and test_y is not None:
                print("Accuracy: %f%%" % (self.evaluate(test_x, test_y) * 100))

    def train_batch(self, x, y) -> None:
        self.sess.run(self.updates, feed_dict={self.x: x, self.y: y})

    def __del__(self) -> None:
        self.sess.close()
```

B.1.3 Compressable Neuronal Network

```
import math
from typing import List

import numpy as np

from ai import NeuronalNetwork
from ai.HomogenousFunctions import homogenize_matrix, homogenize_translation_vector, hom


def sigmoid(x):
    return 1 / (1 + math.exp(-x))


class CompressableNeuronalNetwork(NeuronalNetwork):
    M: np.ndarray
    R: np.ndarray
    a: any

    def __init__(self, w: List[np.ndarray], b: List[np.ndarray]):
        w = [homogenize_matrix(m) for m in w]
        b = [homogenize_translation_vector(v) for v in b]

        self.a = np.vectorize(sigmoid)

        m = np.identity(w[0].shape[0])

        for l in zip(w, b):
            m = self.a(np.matmul(np.matmul(m, l[0]), l[1]))

        self.M = m
        self.R = np.linalg.pinv(self.M)

    def get_label_count(self) -> int:
        return self.R.shape[0] - 1

    def feed_forward(self, x: np.ndarray) -> np.ndarray:
        x = homogenize_vector(x)
        x = np.matmul(x, self.M)
        return x[:, :-1]

    def feed_backwards(self, x: np.ndarray) -> np.ndarray:
        x = homogenize_vector(x)
        x = np.matmul(x, self.R)
        return x[:, :-1]
```

B.1.4 Compressable Neuronal Network Version 2

```
import math
from typing import List

import numpy as np

from ai import NeuronalNetwork
from ai.HomogenousFunctions import homogenize_matrix, homogenize_translation_vector, hor


def sigmoid(x):
    return 1 / (1 + math.exp(-x))


def sigmoid_inverse(x):
    return -math.log(1/x - 1)


class CompressableNeuronalNetworkV2(NeuronalNetwork):
    M: np.ndarray
    R: np.ndarray
    a: any
    ai: any

    def __init__(self, w: List[np.ndarray], b: List[np.ndarray]):
        w = [homogenize_matrix(m) for m in w]
        b = [homogenize_translation_vector(v) for v in b]

        self.a = np.vectorize(sigmoid)
        self.ai = np.vectorize(sigmoid_inverse)

        m = np.identity(w[0].shape[0])

        for l in zip(w, b):
            m = np.matmul(np.matmul(m, l[0]), l[1])

        self.M = m
        self.R = np.linalg.pinv(self.M)

    def get_label_count(self) -> int:
        return self.R.shape[0] - 1

    def feed_forward(self, x: np.ndarray) -> np.ndarray:
        x = homogenize_vector(x)
        x = np.matmul(x, self.M)
        return self.a(np.clip(x[:, :-1], -500, 500))

    def feed_backwards(self, x: np.ndarray) -> np.ndarray:
        x = homogenize_vector(self.ai(np.clip(x, 0.000000001, 0.999999999)))
        x = np.matmul(x, self.R)
        return x[:, :-1]
```

B.1.5 Invertable Neuronal Network

```
import math
from typing import List

import numpy as np

from ai import NeuronalNetwork
```

```
from ai.HomogenousFunctions import homogenize_matrix, homogenize_translation_vector, hor


def sigmoid(x):
    return 1 / (1 + math.exp(-x))


def sigmoid_inverse(x):
    return -math.log(1/x - 1)


class InvertableNeuronalNetwork(NeuronalNetwork):
    M: List[np.ndarray]
    R: List[np.ndarray]
    a: any
    ai: any

    def __init__(self, w: List[np.ndarray], b: List[np.ndarray]):
        w = [homogenize_matrix(m) for m in w]
        b = [homogenize_translation_vector(v) for v in b]

        self.a = np.vectorize(sigmoid)
        self.ai = np.vectorize(sigmoid_inverse)

        self.M = [np.matmul(w[i], b[i]) for i in range(len(w))]
        self.R = list(reversed([np.linalg.pinv(m) for m in self.M]))

    def get_label_count(self) -> int:
        return self.R[0].shape[0] - 1

    def feed_forward(self, x: np.ndarray) -> np.ndarray:
        x = homogenize_vector(x)
        for m in self.M:
            x = np.matmul(x, m)
            x[:, :-1] = self.a(np.clip(x[:, :-1], -500, 500))
        return x[:, :-1]

    def feed_backwards(self, x: np.ndarray) -> np.ndarray:
        x = homogenize_vector(x)
        for m in self.R:
            x[:, :-1] = self.ai(np.clip(x[:, :-1], 0.000000001, 0.999999999))
            x = np.matmul(x, m)
        return x[:, :-1]
```

B.1.6 Squared Invertable Neuronal Network

```
import math
from typing import List

import numpy as np

from ai import NeuronalNetwork
from ai.HomogenousFunctions import homogenize_matrix, homogenize_translation_vector, hor


def sigmoid(x):
    return 1 / (1 + math.exp(-x))
```

```
def sigmoid_inverse(x):
    return -math.log(1/x - 1)


class SquaredInvertableNeuronalNetwork(NeuronalNetwork):
    M: List[np.ndarray]
    R: List[np.ndarray]
    a: any
    ai: any

    def __init__(self, w: List[np.ndarray], b: List[np.ndarray]):
        w = [homogenize_matrix(m) for m in w]
        b = [homogenize_translation_vector(v) for v in b]

        self.a = np.vectorize(sigmoid)
        self.ai = np.vectorize(sigmoid_inverse)

        self.M = [np.matmul(w[i], b[i]) for i in range(len(w))]
        self.R = list(reversed([np.linalg.inv(m) for m in self.M]))

    def get_label_count(self) -> int:
        return 10

    def feed_forward(self, x: np.ndarray) -> np.ndarray:
        x = homogenize_vector(x)
        for m in self.M:
            x = np.matmul(x, m)
            x[:, :-1] = self.a(np.clip(x[:, :-1], -500, 500))
        return x[:, :-1]

    def feed_backwards(self, x: np.ndarray) -> np.ndarray:
        v = np.zeros((1, 28 * 28))
        v[:, 0:10] = x
        x = homogenize_vector(v)
        for m in self.R:
            x[:, :-1] = self.ai(np.clip(x[:, :-1], 0.000000001, 0.999999999))
            x = np.matmul(x, m)
        return x[:, :-1]

    def predict(self, x: np.ndarray):
        return np.argmax(self.feed_forward(x)[:, 0:10], axis=1)
```

B.1.7 Homogenous Functions

```
import numpy as np


def homogenize_matrix(m: np.ndarray) -> np.ndarray:
    h = np.zeros((m.shape[0] + 1, m.shape[1] + 1))
    h[:-1, :-1] = m
    h[m.shape[0]][m.shape[1]] = 1
    return h


def homogenize_translation_vector(v: np.ndarray) -> np.ndarray:
    n = v.shape[1] + 1
    h = np.identity(n)
    h[n - 1, :-1] = v
    return h
```

```
def homogenize_vector(v: np.ndarray) -> np.ndarray:
    h = np.ones((v.shape[0], v.shape[1] + 1))
    h[:, :-1] = v
    return h
```

B.1.8 Approximation Neural Network

```
from typing import List, Tuple

import numpy as np
import tensorflow as tf


class ApproximationNeuralNetwork:

    input_size: int

    def __init__(self, w: List[np.ndarray], b: List[np.ndarray], learning_rate: float):
        self.input_size = w[0].shape[0]
        self.o = tf.Variable(tf.random_normal(shape=(1, self.input_size)))
        W = [tf.constant(m) for m in w]
        B = [tf.constant(m) for m in b]
        self.i = tf.placeholder(tf.float32, shape=(1, w[len(w) - 1].shape[1]))
        self.start = tf.placeholder(tf.float32, shape=(1, self.input_size))

        a = self.o
        for i in range(len(W)):
            a = tf.nn.sigmoid(tf.add(tf.matmul(a, W[i]), B[i]))

        self.cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(logits=a,
        self.updates = tf.train.GradientDescentOptimizer(learning_rate).minimize(self.co

        self.rest = tf.assign(self.o, self.start)

        self.sess = tf.Session()
        init = tf.global_variables_initializer()
        self.sess.run(init)

    def approximate(self, target: np.ndarray, start: np.ndarray = None) -> Tuple[np.ndar
        if start is None:
            start = np.random.normal(0, 1, self.input_size).reshape(1, self.input_size)
        self.sess.run(self.rest, feed_dict={self.start: start})
        for i in range(100):
            for _ in range(100):
                self.sess.run(self.updates, feed_dict={self.i: target})
        return self.get_status(target)

    def get_status(self, target: np.ndarray):
        return self.sess.run(self.o), self.sess.run(self.cost, feed_dict={self.i: target
```

B.2 Ausgabe

B.2.1 Bild Generierung

```
from typing import List

import matplotlib.pyplot as plt
import numpy as np
```

```
from scipy.stats import norm


def disp_image(image: np.ndarray, title: str = None) -> None:
    make_image(image, title)
    plt.show()


def save_image(file_name: str, image: np.ndarray, title: str = None) -> None:
    make_image(image, title)
    plt.savefig(file_name)
    print("Created %s" % file_name)


def make_image(image: np.ndarray, title: str = None) -> None:
    plt.clf()
    plt.imshow(image.reshape(28, 28), cmap=plt.cm.Greys)
    if title is not None:
        plt.title(title)


def disp_histogram(data: List, topic: str, title: str = None) -> None:
    make_histogram(data, topic, title)
    plt.show()


def save_histogram(file_name: str, data: List, topic: str, title: str = None) -> None:
    make_histogram(data, topic, title)
    plt.savefig(file_name)
    plt.close()
    print("Created %s" % file_name)


def make_histogram(data: List, topic: str, title: str = None) -> None:
    plt.clf()
    plt.hist(data, density=True)

    (mu, sigma) = norm.fit(data)
    x = np.linspace(min(data), max(data), 100)

    y = norm(mu, sigma).pdf(x)
    plt.plot(x, y, 'r', linewidth=2)

    if title is not None:
        plt.title(title)

    plt.xlabel(topic)
    plt.ylabel("Relative Dichte")

    plt.grid(True)
```

B.2.2 Konusions Matrix Generierung

```
import numpy as np
from typing import List


def create_confusion_table_matrix(filename: str, data: np.ndarray, labels: List[str], ca
                                  label: str = None):
```

```
    with open(filename, "w") as text_file:
        text_file.write("\\documentclass[Interploate_hadwritten_Digits.tex]{subfiles}\n"
        text_file.write("\\begin{document}\n")
        text_file.write("\\begin{table}[H]\n")
        text_file.write("\\centering\n")
        text_file.write("\\begin{tabular}{|%s|}\n" % "|".join(['r' for _ in range(len(la
        text_file.write("\\hline\n")
        text_file.write("Actual␣\\textbackslash␣Prediction␣&␣%s␣\\\\␣\\hline\n" % "␣&␣".
        for i in range(len(data)):
            text_file.write("%s␣&␣%s␣\\\\␣\\hline\n" % (labels[i], "␣&␣".join(map(str, c
        text_file.write("\\end{tabular}\n")
        if caption is not None:
            text_file.write("\\caption{%s}\n" % caption)
        if label is not None:
            text_file.write("\\label{%s}\n" % label)
        text_file.write("\\end{table}\n")
        text_file.write("\\end{document}")
    print("Created␣%s" % filename)
```

B.3 Auswertung

B.3.1 Leistung Netze und Invertierung

```
import math
from typing import Tuple

import numpy as np

from ai import *
from ui import save_histogram, save_image, create_confusion_table_matrix


def interpolate_vector(v: np.ndarray, dimen: Tuple[int, int], factor: float) -> np.ndarr
    m = np.identity(v.shape[1])
    a = math.pi / 4 * factor
    m[dimen[0], dimen[0]] = math.cos(a)
    m[dimen[0], dimen[1]] = math.sin(a)
    m[dimen[1], dimen[0]] = -math.sin(a)
    m[dimen[1], dimen[1]] = math.cos(a)
    return np.matmul(v, m)


work_dir = '.\\tmp'
out_dir = '.\\out'

accuracy = []
c_accuracy = []
c2_accuracy = []
i_accuracy = []

network = MnistNetwork([100, 50], 0.1, work_dir)

train_data = network.load_train_data()
test_data = network.load_test_data()

best_network = None
best_sq_network = None

best_acc = 0
```

```
sample_size = 20

labels = list(map(str, range(10)))

for i in range(sample_size):
    print("Iteration %d of %d" % (i + 1, sample_size))
    network = MnistNetwork([100, 50], 0.1, work_dir)
    acc = network.train(100, 100)
    accuracy.append(acc)
    confusion_matrix = network.confusion_matrix(test_data[0], test_data[1])
    create_confusion_table_matrix("%s\confusion_table_network_%d.tex" % (out_dir, i), co
                                  "Konfusionsmatrix Netzwerk %d" % i, "tbl:confusion_tab
    c_network = CompressableNeuronalNetwork([network.network.sess.run(t) for t in networ
                                             [network.network.sess.run(t) for t in networ
    c_accuracy.append(c_network.evaluate(test_data[0], test_data[1]))
    c_confusion_matrix = c_network.confusion_matrix(test_data[0], test_data[1])
    create_confusion_table_matrix("%s\confusion_table_compressed_network_%d.tex" % (out_
                                  labels, "Konfusionsmatrix Komprimiertes Netzwerk Art 1
                                  "tbl:confusion_table_compressed_network_%d" % i)
    c2_network = CompressableNeuronalNetworkV2([network.network.sess.run(t) for t in net
                                                [network.network.sess.run(t) for t in net
    c2_accuracy.append(c2_network.evaluate(test_data[0], test_data[1]))
    c2_confusion_matrix = c2_network.confusion_matrix(test_data[0], test_data[1])
    create_confusion_table_matrix("%s\confusion_table_compressed_network_v2_%d.tex" % (o
                                  labels, "Konfusionsmatrix Komprimiertes Netzwerk Art 2
                                  "tbl:confusion_table_compressed_network_v2_%d" % i)
    i_network = InvertableNeuronalNetwork([network.network.sess.run(t) for t in network.
                                           [network.network.sess.run(t) for t in network.
    i_accuracy.append(i_network.evaluate(test_data[0], test_data[1]))
    i_confusion_matrix = i_network.confusion_matrix(test_data[0], test_data[1])
    create_confusion_table_matrix("%s\confusion_table_inverted_network_%d.tex" % (out_di
                                  labels, "Konfusionsmatrix Invertiertes Netzwerk %d" %
                                  "tbl:confusion_table_inverted_network_%d" % i)
    if acc > best_acc:
        best_acc = acc
        best_network = network

save_histogram("%s\histogram_network_accuracy.png" % out_dir, accuracy, "Präzision Erker
save_histogram("%s\histogram_compressed_network_accuracy.png" % out_dir, c2_accuracy, "I
save_histogram("%s\histogram_compressed_network_v2_accuracy.png" % out_dir, c_accuracy,
save_histogram("%s\histogram_inverted_network_accuracy.png" % out_dir, accuracy, "Präzis

sq_accuracy = []
isq_accuracy = []
best_acc = 0
for i in range(sample_size):
    print("Iteration %d of %d" % (i, sample_size))
    network = MnistNetwork([], 0.1, work_dir, squared=True)
    acc = network.train(100, 100)
    sq_accuracy.append(acc)
    sq_confusion_matrix = network.confusion_matrix(test_data[0], test_data[1])
    create_confusion_table_matrix("%s\confusion_table_squared_network_%d.tex" % (out_dir
                                  labels, "Konfusionsmatrix Quadratisches Netzwerk %d" %
                                  "tbl:confusion_table_squared_network_%d" % i)
    i_network = SquaredInvertableNeuronalNetwork([network.network.sess.run(t) for t in r
                                                  [network.network.sess.run(t) for t in r
    isq_accuracy.append(i_network.evaluate(test_data[0], test_data[1]))
    isq_confusion_matrix = i_network.confusion_matrix(test_data[0], test_data[1])
```

```
    create_confusion_table_matrix("%s\confusion_table_squared_inverted_network_%d.tex" %
                                  isq_confusion_matrix, labels,
                                  "Konfusionsmatrix Quadratisches Invertiertes Netzwerk
                                  "tbl:confusion_table_squared_inverted_network_%d" % i)
    if acc > best_acc:
        best_acc = acc
        best_sq_network = network

save_histogram("%s\histogram_squared_network_accuracy.png" % out_dir, sq_accuracy,
               "Präzision Quadratische Erkennungsnetzwerke")
save_histogram("%s\histogram_squared_inverted_network_accuracy.png" % out_dir, accuracy,
               "Präzision Quadratische Invertierbare Nettzwerke")

i_reversed_network = InvertableNeuronalNetwork([best_network.network.sess.run(t) for t i
                                                [best_network.network.sess.run(t) for t i

a_reversed_network = ApproximationNeuralNetwork([best_network.network.sess.run(t) for t
                                                 [best_network.network.sess.run(t) for t

sq_i_reversed_network = SquaredInvertableNeuronalNetwork(
    [best_sq_network.network.sess.run(t) for t in best_sq_network.network.w],
    [best_sq_network.network.sess.run(t) for t in best_sq_network.network.b])

approximation_error_ideal = []

for i in range(10):
    v = np.zeros((1, 10))
    v[0, i] = 1
    image = i_reversed_network.feed_backwards(v)
    save_image("%s\ideal_%d_inverted.png" % (out_dir, i), image, "Ideale %d aus Invertie
    image = sq_i_reversed_network.feed_backwards(v)
    save_image("%s\ideal_%d_squared_inverted.png" % (out_dir, i), image, "Ideale %d aus
    image, error = a_reversed_network.approximate(v)
    approximation_error_ideal.append(error)
    save_image("%s\ideal_%d_approximated.png" % (out_dir, i), image, "Ideale %d aus App

save_histogram("%s\histogram_approximation_error_ideal.png" % out_dir, approximation_er
               "Fehler Approximierung idealer Ziffern")

approximation_error_interpolated = []
for i in range(10):
    for j in range(i, 10):
        for k in [0.25, 0.5, 0.75]:
            l = int(k * 100)
            v = np.zeros((1, 10))
            v[0, i] = 1
            v = interpolate_vector(v, (i, j), k)
            image = i_reversed_network.feed_backwards(v)
            save_image("%s\interpolated_%d_%d_%d_inverted.png" % (out_dir, i, j, l), ima
                       "Interpolierte Ziffer %d%% zwischen %d und %d aus Invertierung" %
            image = sq_i_reversed_network.feed_backwards(v)
            save_image("%s\interpolated_%d_%d_%d_squared_inverted.png" % (out_dir, i, j,
                       "Interpolierte Ziffer %d%% zwischen %d und %d aus Quadratischer
            image, error = a_reversed_network.approximate(v)
            approximation_error_interpolated.append(error)
            save_image("%s\interpolated_%d_%d_%d_approximated.png" % (out_dir, i, j, l),
                       "Interpolierte Ziffer %d%% zwischen %d und %d aus Approximierung"
```

```
save_histogram("%s\histogram_approximation_error_interpolation.png" % out_dir, approxim
               "Fehler Approximierung interpolierter Ziffern")
```

B.3.2 Berechnung Rundungsfehler Gleitkommazahlen

```
from functools import reduce

import matplotlib.pyplot as plt
import numpy as np

from ai import InvertableNeuronalNetwork


def generate_random_matrix(x: int) -> np.ndarray:
    return np.random.normal(0, 1, x * x).reshape(x, x)


def generate_random_vector(x: int) -> np.ndarray:
    return np.random.normal(0, 1, x).reshape(1, x)


def disp(img: np.ndarray) -> None:
    plt.clf()
    plt.imshow(img, cmap=plt.cm.Greys)
    # plt.show()


m = generate_random_vector(25)
disp(m.reshape(5, 5))
n = InvertableNeuronalNetwork([generate_random_matrix(25) for _ in range(3)],
                              [generate_random_vector(25) for _ in range(3)])

disp(n.feed_backwards(n.feed_forward(m)).reshape(5, 5))


res = np.zeros((7, 4))

for i in range(3, 10):
    for j in range(1, 5):
        l = []
        for _ in range(20):
            m = generate_random_vector(i * i)
            n = InvertableNeuronalNetwork([generate_random_matrix(i * i) for _ in range(
                                          [generate_random_vector(i * i) for _ in range(
            x = n.feed_backwards(n.feed_forward(m))
            l.append(np.sum(np.abs(m - x)))
        err = (reduce(lambda x, y: x + y, l) / len(l))
        res[(i - 3, j - 1)] = err / (i * i)

print(res)
```

B.3.3 Bestimmung von Grenzfällen der Klassifizierung

```
import numpy as np

from ai import MnistNetwork
from ui import disp_image, save_image
```

```
def softmax_cross_entropy(x, y):
    e = np.exp(x)
    p = e / np.sum(e)
    return -np.sum(y * np.log(p))

work_dir = '.\\tmp'
out_dir = '.\\out'

network = MnistNetwork([100, 50], 0.1, work_dir)

train_data = network.load_train_data()
test_data = network.load_test_data()

for i in range(3):
    disp_image(train_data[0][i], 'Check train data, expected value: %d' % np.argmax(trai

for i in range(3):
    disp_image(test_data[0][i], 'Check test data, expected value: %d' % np.argmax(test_d

acc = network.train(100, 100)
print("Network accuracy: %f%%" % (acc * 100))
print(network.network.confusion_matrix(test_data[0], test_data[1]))

predictions = network.feed_forward(test_data[0])

top = [(10, -1) for i in range(10)]
flop = [(0, -1) for i in range(10)]
top_false = [(10, -1) for i in range(10)]

for i in range(len(predictions)):
    dig = int(np.argmax(test_data[1][i]))
    err = softmax_cross_entropy(predictions[i], test_data[1][i])
    if np.argmax(predictions[i]) != np.argmax(test_data[1][i]):
        if err < top_false[dig][0]:
            top_false[dig] = (err, i)
    else:
        if err < top[dig][0]:
            top[dig] = (err, i)
        if err > flop[dig][0]:
            flop[dig] = (err, i)

run = 2

for dig in range(10):
    (err, i) = top[dig]
    save_image("%s\\top-flop\\clear_%d_%d.png" % (out_dir, dig, run), test_data[0][i],
               "Klare %d, Fehler: %f" % (dig, err))
    print("Top %d, err: %f: vec: %s" % (dig, err, ["%.2f" % f for f in predictions[i]]))

for dig in range(10):
    (err, i) = flop[dig]
    p = predictions[i]
    p[np.argmax(p)] = 0
    almost = int(np.argmax(p))
    save_image("%s\\top-flop\\barely_%d_%d.png" % (out_dir, dig, run), test_data[0][i],
               "Knappe %d, Fast: %d, Fehler: %f" % (dig, almost, err))

    print("Flop %d, almost: %d, err: %f: vec: %s" % (dig, almost, err, ["%.2f" % f for f
```

```
for dig in range(10):
    (err, i) = top_false[dig]
    if i == -1:
        continue
    save_image("%s\\top-flop\\flop_%d_%d.png" % (out_dir, dig, run), test_data[0][i],
               "Knapp nicht %d, erkannt %d, Fehler: %f" % (dig, int(np.argmax(prediction
    print("Barely false %d, detected: %d, err: %f: vec: %s" % (dig, int(np.argmax(predic
                                                               ["%.2f" % f for f in pred
```