Charakterisierung quasikristalliner Muster mittels künstlicher Intelligenz

Bachelorarbeit aus der Physik

Vorgelegt von
Jonas Buba
19.11.2021

Institut für Theoretische Physik I
Friedrich-Alexander-Universität Erlangen-Nürnberg

Betreuer: Prof. Dr. M. Schmiedeberg

Zusammenfassung

Quasikristalle wurden Ende des 20. Jahrhunderts entdeckt und besitzen trotz langreichweitiger Ordnung keine Translationssymmetrie wie klassische Kristalle. Daraus ergeben sich Muster hoher Komplexität. Ziel dieser Arbeit war deshalb herauszufinden inwiefern künstliche Intelligenz zu Charakterisierung dieser Muster genutzt werden kann.

Konkret wurde dafür ein neuronales Autoencoder Netzwerk programmiert, welches selbstständig eine geeignete Parametrisierung für zweidimensionale Quasikristallmuster erlernen soll. Dies geschieht indem das Netzwerk versucht das Muster nach einer Kompression auf wenige Werte wieder zu rekonstruieren. Dies erwies sich für alle getesteten Rotationssymmetrien als möglich. Hierbei benötigte das neuronale Netz teilweise deutlich weniger Parameter als zur Simulation der Muster ursprünglich verwendet wurden. Die Nutzung bestimmter Teile des Netzwerks zum Rückschluss auf die Simulationsparameter war nur bedingt erfolgreich. Mittels anderer Netzwerkteile war es wiederum möglich zufällige neue Muster zu generieren.

Auf welche Weise Quasikristalline Muster von neuronalen Netzen parametrisiert werden, sowie darauf aufbauend die gezielte Erzeugung bestimmter Muster, sind Gegenstand künftiger Forschung.

Inhaltsverzeichnis

1. Quasikristalle

Lange Zeit wurden als Kristalle nur solche Strukturen verstanden, die eine Periodizität aufweisen. Mit der Wiederholung bestimmter Bereiche geht auch eine Ordnung über große bzw. unbeschränkte Entfernungen einher. Dass auch Strukturen mit langreichweitiger Ordnung ohne die Einschränkung auf Periodizität existieren können, zeigt sich in Mosaiken des Islamischen Mittelalters (Grafik 1) [1]. Dass solche Muster auch in der Natur in Form von sogenannten Quasikristallen auftreten, wurde erst im 20. Jahrhundert von D. Shechtman [2] entdeckt. Dies führte zu einem Umdenken, wie Kristalle definiert werden können [3].

Abbildung 1: Photografie des Darb-i Imam Schrein in Isfahan, Iran. Grafik aus [1].

Eine neue mathematische Erkenntnis ist, dass raumfüllende Kachelungen mit langreichweitiger Ordnung aber ohne sich regelmäßig wiederholende Abschnitte existieren. Dies galt bis Mitte des vergangenen Jahrhunderts als unmöglich. Nur mittels Dreiecken, Rechtecken und Sechsecken war es möglich, eine Fläche mit gleichseitigen Vielecken zu füllen, ohne dass Lücken auftreten. Dies führt immer zu einer periodischen Anordnung [3]. Das wohl bekannteste Beispiel für ein aperiodisches Kachelmuster ist die nach Roger Penrose benannte Penrose-Parkettierung, welche in Abbildung 2 gezeigt wird. Sie besteht aus 2 verschiedenen Kacheln und weist eine 5-zählige Rotationssymmetrie auf. Dies sieht man gut an den fünfzackigen Sternen. Trotz der fehlenden Periodizität ist hier eine Art von langreichweitiger Ordnung erkennbar.

Da Quasikristalle beliebige Rotationssymmetrien aber keine sich regelmäßig wiederholenden Bereiche besitzen, sind diese Muster für das menschliche Auge mitunter schwer zu erfassen. Selbst eine grundlegende Eigenschaft wie die Symmetrie des Musters kann schwer zu erkennen sein. Deswegen soll in dieser Arbeit untersucht werden, inwiefern künstliche Intelligenz in Form neuronaler Netze genutzt werden kann, um derartige Muster zu charakterisieren.

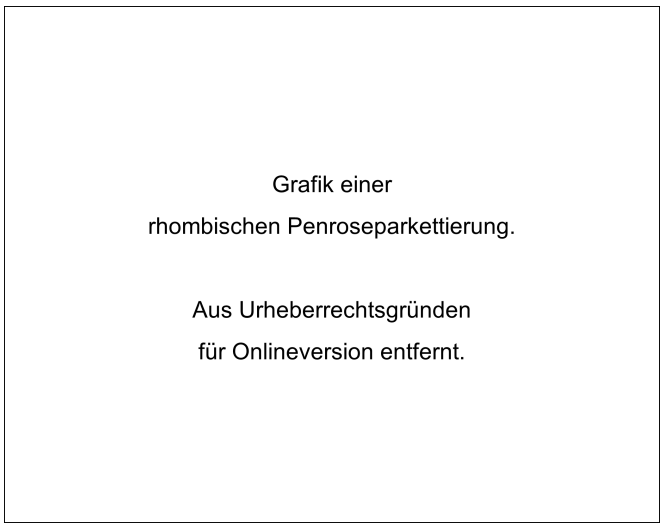


Abbildung 2: Grafik einer Penrose-Parkettierung aus [4].

2. Künstliche neuronale Netze

Nach Vorbild des menschlichen Gehirns, welches in der Lage ist, Reize zu verarbeiten, zu abstrahieren und selbstständig dazuzulernen, wird bereits seit etwa 1940 versucht, künstliche Strukturen mit ähnlichen Eigenschaften zu schaffen, sogenannte künstliche neuronale Netze. Dabei steht nicht unbedingt eine möglichst genaue Modellierung des Gehirns im Vordergrund, sondern das Nutzen einzelner für die entsprechende Problemstellung relevanter Eigenschaften. Die deutlichen Fortschritte auf diesem Gebiet in den letzten Jahren resultieren vor allem aus der deutlich gesteigerten Rechenleistung moderner Computer und der Verfügbarkeit großer, für das Training notwendiger Datenmengen [5, S.3].

Die kleinste Recheneinheit wird, wie beim Gehirn auch, als Neuron bezeichnet. Dieses besitzt Inputs $x_1, x_2, ..., x_n$, welche in der Regel jeweils mit einem Faktor $w_i, i \in 1, ..., n$ gewichtet und anschließend aufsummiert werden. Wird noch ein Offset b addiert, erhält man also

$$g(x) = \sum_{i=1}^{n} w_i x_i + b. \tag{1}$$

Den Output des Neurons erhält man, indem noch eine Aktivierungsfunktion $f(g(x))$ angewendet wird. Für diese können verschiedenste in der Regel nichtlineare Funktionen verwendet werden. Sofern nicht anders angegeben, wurde hierfür die ReLU (rectified linear unit) Funktion verwendet. Dieser allgemeine Aufbau ist veranschaulicht in Grafik 3. Erst durch die Verwendung dieser nichtlinearen Aktivierungsfunktion erhält das Netzwerk später viele seiner gewünschten Eigenschaften [6, S.31].

In vielen neuronalen Netzen werden die Neuronen in aufeinander folgenden Schichten, den Layern, angeordnet, wie in Grafik 4 gezeigt. Die Neuronen haben als Input dann jeweils den Output der Neuronen im vorangegangenen Layer.

Auf diese Weise lassen sich viele verschiedene Arten von neuronalen Netzen zusam-

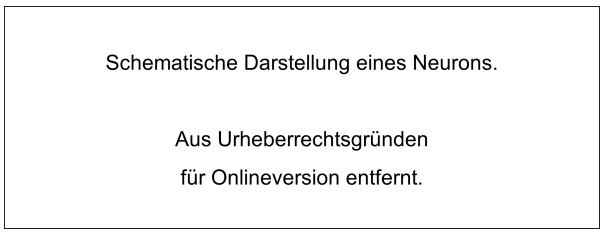


Abbildung 3: Schematische Darstellung eines Neurons, Bild aus [6].

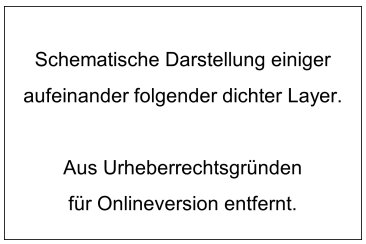


Abbildung 4: Darstellung einiger aufeinander folgender Layer mit unterschiedlicher Neuronenzahl, wobei die meisten Verbindungen zwischen den Neuronen der Übersichtlichkeit halber weggelassen wurden, Bild aus [5]

menstellen, etwa um beliebige Funktionen zu approximieren oder zur Bild Klassifikation und Manipulation. Im Folgenden soll jedoch ein sogenannter Autoencoder verwendet werden, welcher schematisch in Abbildung 5 dargestellt ist. Dieser komprimiert den Input zunächst im Encoder, sodass das Bottleneck in der Mitte deutlich weniger Neuronen enthält als der Input. Danach wird im Decoder versucht, den ursprünglichen Input wieder zu rekonstuieren.

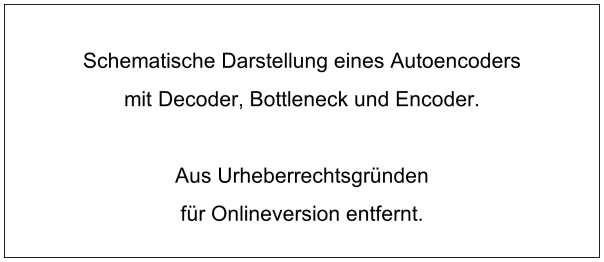


Abbildung 5: Schematische Darstellung eines Autoencoders, Bild aus [5]

Dieser Netzwerktyp eignet sich vor allem dann, wenn die Inputdaten wenige freie Parameter haben, zum Beispiel Position und Radius im Falle von 2D Grafiken mit jeweils einer Kreisscheibe zufälliger Position und Größe. Nur dann ist es möglich, dass das Netzwerk eine geeignete Parametrisierung findet. Im Falle völlig zufälligen Rauschens ist eine Komprimierung unmöglich.

Die hier betrachteten Quasikristalle haben jedoch eine langreichweitige Ordnung und Symmetrieeigenschaften, wodurch es dem Autoencodernetzwerk grundsätzlich möglich sein sollte, eine geeignete Parametrisierung zu finden.

3. Trainings- und Validierungsdaten

Den Ausgangspunkt zum Training des Autoencoders bilden zweidimensionale Quasikristallmuster der Symmetrien 2 bis 18 mit Kantenlänge 1000. Außerdem wurden die zur Erzeugung genutzten Parameter (2 pro Symmetrie und einen Skalierungsfaktor) mit zur Verfügung gestellt. Für Details zur Mustergenerierung siehe Appendix A. Da diese Muster allerdings sehr groß für die Verwendung in einem Neuronalen Netzwerk und die Erzeugung in geeigneter Menge sehr zeitaufwändig wäre, wurden $100 \cdot 100$ Pixel große Ausschnitte gewählt. Aus jedem großen Muster wurden auf diese Weise sowie durch Spiegelungen jeweils 10 kleinere Muster erzeugt, wie in Grafik 6 am Beispiel eines Musters fünfzähliger Symmetrie gezeigt wird. Diese Ausschnitte werden zusätzlich durch fünf weitere Parameter charakterisiert. Für Muster 18-zähliger Symmetrie ergibt sich dann also eine maximale Parameterzahl von 42.

Insgesamt standen damit für jede Symmetrie zwischen 1620 und 2060 kleine Muster zur Verfügung. Von diesen wurden jeweils die ersten 90% für das Training des Netzwerks verwendet und die Restlichen für die Validierung des Trainingsfortschritts genutzt. Dabei wurde darauf geachtet, dass die aus einem großen Muster abgeleiteten Daten alle einem der Datensätze zugeordnet wurden, um sicherzugehen, dass das Trainingsergebnis nicht dadurch verfälscht wird, dass das Netzwerk bereits einen anderen Teil des Musters gesehen hat.

Da in einem Autoencoder die Kantenlänge der Inputdaten typischerweise wiederholt um einen Faktor 2 reduziert wird, wurden die $100 \cdot 100$ Muster außerdem auf eine Größe von $96 \cdot 96$ Pixeln zugeschnitten. Dies ermöglicht es, die Kantenlänge bis zu 5 Mal einfach zu reduzieren. Des Weiteren wurden die gesamten Daten reskaliert, sodass sie in einem für die Anwendung in neuronalen Netzen typischen Wertebereich zwischen 0 und 1 liegen. Dadurch sollten auch die Parameter des Netzwerks in dieser Größenordnung liegen, was die Initialisierung erleichtert. [7]

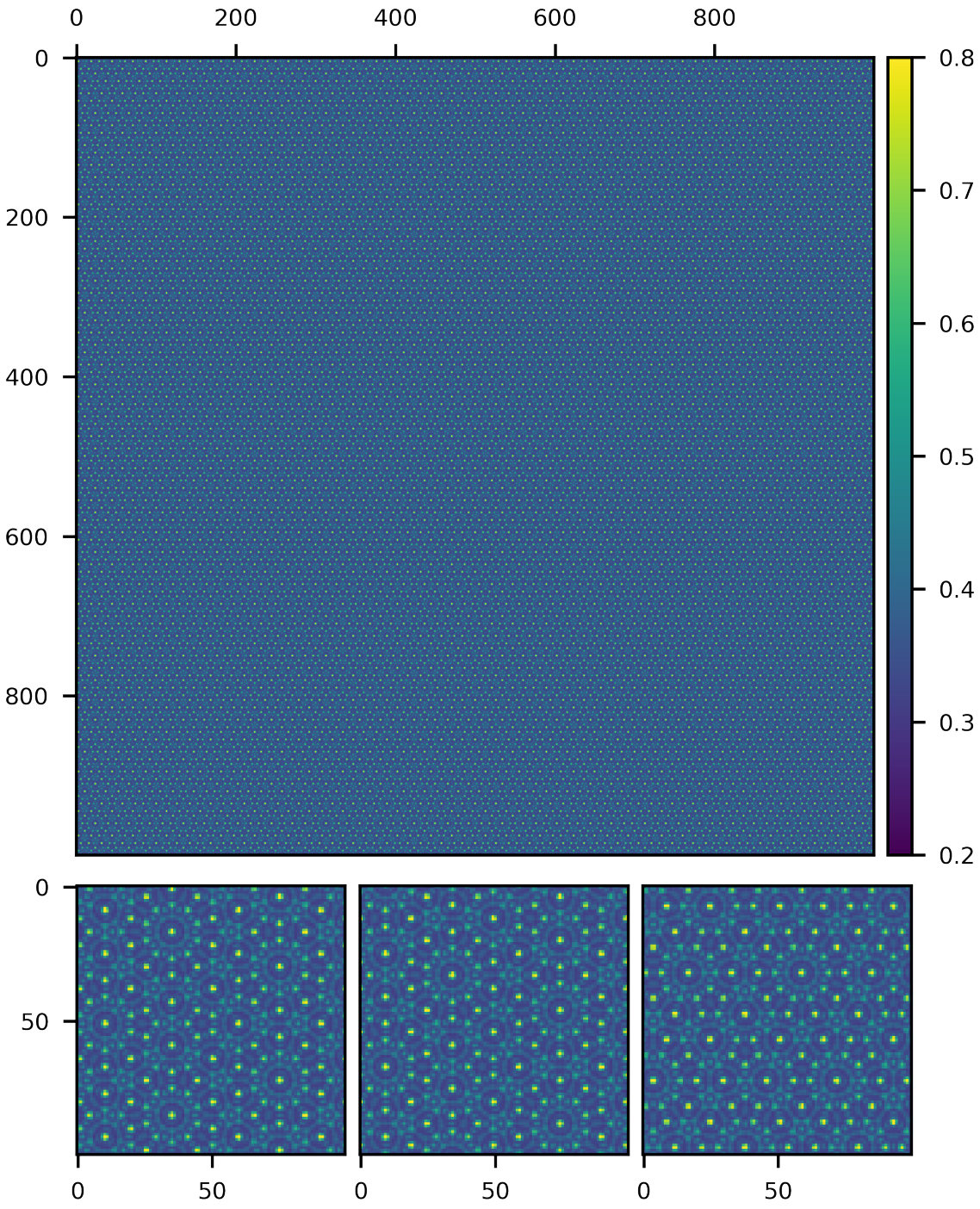


Abbildung 6: Großes Quasikristallmuster mit 5 zähliger Symmetrie und Kantenlänge von 1000 oben und 3 daraus abgeleitete Muster der Größe $100 \cdot 100$ unten

4. Aufbau des Autoencoders

Für den Autoencoder wurde ein bezüglich des Bottlenecks symmetrischer Aufbau gewählt, wie in Tabelle 1 gezeigt. Der Input wird zunächst im Encoder, welcher sich aus N Encode-Schritten zusammensetzt, komprimiert. Das auf einen Verbindungsschritt folgende Bottleneck enthält dadurch sehr viel weniger Werte (Neuronen) als der Input. Der Output wird entsprechend durch N Decode-Schritte erzeugt, welche mit dem Bottleneck verbunden sind.

<table>
<tr><td colspan="3">Input</td></tr>
<tr><td rowspan="6">Autoencoder</td><td rowspan="2">Encoder</td><td>Encode Schritt 1
...
Encode Schritt N</td></tr>
<tr><td>Encoder Verbindung</td></tr>
<tr><td colspan="2">Bottleneck</td></tr>
<tr><td rowspan="2">Decoder</td><td>Decoder Verbindung</td></tr>
<tr><td>Decode Schritt N
...
Decode Schritt 1</td></tr>
<tr><td colspan="3">Output</td></tr>
</table>

Tabelle 1: Grundlege Autoencoder Struktur

Da es sich bei den Input Daten um zweidimensionale Quasikristallmuster handelt, ist das Problem ein recht grafisches. Es liegt also nahe, sogenannte convolutional Layer zu verwenden, wie sie auch in neuronalen Netzen zur Bildklassifikation eingesetzt werden. Convolutional Layer funktionieren sehr ähnlich wie Filter in der Bildbearbeitung, wo das Inputbild diskret mit einer meist quadratischen Matrix (auch genannt Kernel) gefaltet wird. Man erhält also für Input $I(x, y)$ und Kernel $\boldsymbol{K}$

$$I^*(x, y) = \sum_{i=1}^{n} \sum_{j=1}^{n} I(x - i + m, y - j + m) k(i, j) \tag{2}$$

wobei m die Position des Mittelpunktes und $k(i, j)$ die Elemente von $\boldsymbol{K}$ sind. Es ist möglich als Input mehrere Channel mit jeweils zweidimensionalem Input zu haben (wie zum Beispiel RGB Kanäle im Fall eines Bildes). Der Input I_c jedes Channels wird dann mit einem eigenen Kernel $\boldsymbol{K}_c$ gefalten. Im Falle eines Convolutional-Layers werden dann ähnlich zueinem dichten Layer die gefaltenen Bilder I_c^* addiert und mit einem Offset b versehen, bevor die Aktivierungsfunktion angewendet wird.

$$I_f^*(x, y) = f\left(\sum_c I_c^*(x, y) + b_f\right) \tag{3}$$

Dies bildet einen Filter, wovon jeder Convolutional-Layer aus mehreren besteht, sodass der Output auch wieder mehrere Channel entsprechend der Anzahl der Filter besitzt.

Da die Kantenlänge bei der Anwendung eines Convolutional Layers beibehalten werden soll, werden am Rand Nullen ergänzt, um auch dort den Kernel anwenden zu können.

Da es im Encoder aber grundsätzlich das Ziel ist, die Anzahl an Werten bis zum Bottleneck zu reduzieren, besteht jeder Schritt zusätzlich aus einem Maximum-Pooling-Layer, der die Werte in einem meist quadratischen Bereich (hier $2 \cdot 2$) durch deren Maximalwert ersetzt, wodurch sich in jedem Encoder-Schritt die Kantenlänge um einen Faktor 2 reduziert.

Um die Kantenlänge im Decoder wieder zu erhöhen, wird Upsampling symmetrisch zum Pooling verwendet, sodass ein Wert danach ein $2 \cdot 2$ Quadrat füllt. Die Convolutiuonal Layer werden durch deren transponierte Operation ersetzt (Conv2DTranspose). Ein Pixel des Inputs wirkt sich also gemäß dem Kernel auf benachbarte Pixel des Outputs aus. Je Output Channel bzw. Filter werden die Beiträge der verschiedenen Inputpixel sowie -channel (dann jeweils mit eigenem Kernel) addiert, ein Bias hinzugefügt und die Aktivierungsfunktion angewendet.

Zunächst wurden zwei verschiedene Möglichkeiten getestet, um die einzelnen Schritte aufzubauen. In Typ A besteht jeder Encoder-Schritt aus einem Convolutional-Layer mit Kernel der Größe $5 \cdot 5$ gefolgt von einer Maximum-Pooling Operation. In Typ B wird statt eines Convolutional-Layers mit großem Kernel zwei einzelne aber mit kleinerem Kernel ($3 \cdot 3$) verwendet, sodass pro Schritt auch wieder Werte aus einer $5 \cdot 5$ Umgebung berücksichtigt werden, allerdings auf zwei Layer verteilt. Die Decoder-Schritte werden daraus, wie im vorherigen Absatz beschrieben, konstruiert. Das in der Mitte des Netzwerks liegende Bottleneck wird mittels dichter Layer mit linearer Aktivierungsfunktion $f(x) = x$ mit Decoder und Encoder verbunden. Für das Bottleneck wurde eine Größe von 42 Neuronen gewählt, was der Anzahl der Simulations- bzw. Verschiebungsparameter für Muster mit 18 zähliger Symmetrie entspricht. Dieser Aufbau ist in Tabelle 2 dargestellt.

Um nicht alle Layer und anderen, für ein neuronales Netz notwendigen, Funktionen per Hand implementieren zu müssen, wurde auf die TensorFlow Bibliothek [8] für Python zurückgegriffen, welche die meisten gängigen Funktionen bereits beinhaltet.

5. Training einzelner Symmetrien

Nachdem nun sowohl erste neuronale Netze definiert als auch die Trainingsdaten vorbereitet sind, kann mit dem Training des Netzwerkes begonnen werden. Hierfür wird für mehrere Muster (ein Batch) der Output des Autoencoders berechnet und anhand der Loss-Funktion die Abweichung vom erwarteten Output quantifiziert. Da es in diesem Falle das Ziel ist, den Input möglichst gut zu rekonstruieren, ist der angestrebte Output genau der Input. Als Loss-Funktion wurde die mittlere quadratische Abweichung verwendet. Nach jedem Batch werden mittels des Optimizers die freien Parameter des Netzwerk aktualisiert, sodass schrittweise die Loss-Funktion minimiert wird. Als Optimizer wurde der von TensorFlow bereitgestellte Adam Algorithmus (sofern nicht anders spezifiziert mit den Standardeinstellungen) genutzt. Das Trainingsintervall, in dem das

Netzwerk jedes Muster der Trainingsdaten genau einmal sieht, nennt man eine Epoche. Nach jeder Epoche werden die Daten zufällig neu sortiert, sodass die einzelnen Batches jedes Mal unterschiedliche Muster enthalten. Außerdem wird der Trainingsfortschritt evaluiert, indem die Loss-Funktion für alle Validierungsdaten berechnet wird.

Getestet wurden so 2 Netzwerktypen A und B (siehe Tabelle 2) jeweils mit Schrittanzahl 2, 3, 4 und verschiedener Anzahl an Filtern je Convolutional-Layer.

<table>
<tr><td></td><td>Typ A</td><td>Typ B</td></tr>
<tr><td rowspan="2">Encode Schritt</td><td>(5*5) Conv2D</td><td>(3*3) Conv2D
(3*3) Conv2D</td></tr>
<tr><td colspan="2">(2*2) MaxPool2D stride=2</td></tr>
<tr><td>Encoder Verbindung</td><td colspan="2">Dense</td></tr>
<tr><td>Bottleneck</td><td colspan="2">Größe: 42 Neuronen</td></tr>
<tr><td>Decoder Verbindung</td><td colspan="2">Dense</td></tr>
<tr><td rowspan="2">Decode Schritt</td><td colspan="2">UpSampling2D size=(2*2)</td></tr>
<tr><td>(5*5) Conv2DTranspose</td><td>(3*3) Conv2DTranspose
(3*3) Conv2DTranspose</td></tr>
</table>

Tabelle 2: Aufbau der einzelnen Schritte

Dieser Ansatz für den Netzwerkaufbau wurde zunächst separat für Muster mit 5- und 17-zähliger Symmetrie getestet. Diese beiden Symmetrien wurden gewählt um einen möglichst großen Bereich der Trainingsdaten abzudecken. Die 5-zähligen Muster stellen die kleinste Symmetrie mit wirklichen Quasikristallmustern dar, welche bereits auf relativ kurze Distanzen eine gut erkennbare Ordnung aufweisen. Dagegen besitzen die 17-zähligen die Symmetrie mit der größten Primzahl. Dadurch haben diese Muster wenig mit Mustern niedrigerer Symmetrie gemeinsam bzw. lassen sich schlecht aus diesen ableiten. Durch die hohe Symmetriezähligkeit wird erst auf größere Distanz eine Ordnung erkennbar. Damit sollten die 17-zähligen Muster eine der für das Netzwerk am schwersten zu lernenden Muster sein. Wenn diese beiden Symmetrien gut funktionieren, liegt es also nahe, dass das Netzwerk auch mit den anderen Mustern gut zurecht kommen sollte.

Für die Initialisierung des Netzwerks wurde für die Kernelwerte der Convolutional-Layer der HeNormal-Initializer genutzt. Dieser zieht die Kernelwerte aus einer Normalverteilung begrenzt auf das Intervall $[-2stddev, +2stddev]$, wobei die Standardabweichung $stddev = \sqrt{(2/N_{in})}$ abhängig von der Anzahl Kernelwerte N_{in} ist. Die Gewichtungsfaktoren der Dense-Layer werden mit den Standardeinstellungen (GlorotUniform) und der Offset für beide Layertypen mit 0 initialisiert.

Die Anzahl an Filtern pro Convolutional-Layer wurde variiert, um die beste Netzwerkkonfiguration zu finden, wobei die Filterzahl in einem Encoder- bzw. Decoderschritt jeweils konstant bleibt und sich nur zwischen den Schritten ändert. Dafür wurde jeweils die Filterzahl $filter_0$ im ersten Encoderschritt spezifiziert und dann auf verschiedene Weise zum Bottleneck hin erhöht:

konstant	$filter_{const,n} = filter_0$
linear	$filter_{lin,n} = n \cdot filter_0$
exponentiell	$filter_{exp,n} = 2^{n-1} \cdot filter_0$

Im Decoder wird die Filteranzahl dann entsprechend symmetrisch wieder reduziert.

Die verschiedenen Konfigurationen wurden jeweils 3 Mal für 200 Epochen oder bis sich über 20 Epochen keine Verbesserung des Validierungslosses mehr einstellt trainiert und das beste Trainingsergebnis verwendet.

5.1. Training der 5-zähligen Muster

Die Trainingsergebnisse für 5-zählige Muster sind in Tabelle 3 zusammen gefasst. Dabei bezeichnet loss das Minimum der loss Funktion bezüglich der Trainigsdaten über das Netzwerktraining von 200 Epochen hinweg. Das Minimum der Loss-Funktion bezüglich der Validierungsdaten wird analog mit val loss bezeichnet. Evaluiert wurde die Loss-Funktion jeweils am Ende einer Epoche.

verwendetes Netzwerk	Trainingsergebnis n05 $[10^{-3}]$		Filteranzahl	Parameterzahl $[10^3]$
	loss	val loss		
A-MaxPool-2Schritte	0.76	2.22	12,12	595
A-MaxPool-3Schritte	0.84	1.92	12,24,48	660
A-MaxPool-4Schritte	1.87	2.58	12,24,36,48	292
B-MaxPool-2Schritte	0.41	1.69	12,24	1194
B-MaxPool-3Schritte	0.68	1.49	12,24,48	668
B-MaxPool-4Schritte	0.79	1.23	12,24,36,48	277

Tabelle 3: Trainingsergebnisse verschiedene Netzwerke mit maximum Pooling für Muster mit 5-zähliger Symmetrie.

Man sieht sowohl am Validierungsloss in Tabelle 3 als auch an den Beispielmustern in Grafik 7, dass Netzwerk Typ B deutlich besser abschneidet als Typ A. Die erhöhte Komplexität durch mehr Convolutional-Layer scheint sich hier vorteilhaft auszuwirken. Betrachtet man den Trainingsfortschnitt anhand der Entwicklung der Loss-Funktion bezüglich der Trainingsdaten (loss) und Validierungsdaten (val_loss) in Grafik 8, sieht man, dass der Trainingsloss grundsätzlich einen niedrigeren Wert hat als der Validierungsloss. Dies ist zu erwarten, da das Netzwerk speziell auf die Trainingsdaten trainiert wurde. Es hat also gelernt, diese möglichst gut zu rekonstruieren. Auf die Validierungsdaten wird das Netzwerk nicht trainiert. Anhand dieser wird nur überprüft, dass das Gelernte auch auf unbekannte Muster übertragen werden kann. Ein etwas schlechteres Ergebnis ist hier normal. Zusätzlich sind die jeweiligen Minimalwerte angegeben. Auch eine Erhöhung der Schrittzahl führt zu einer Verbesserung des Trainingsergebnisses mit Ausnahme der A-4Schritte Konfiguration. Bei Vergleich des Trainingsfortschritts (Grafik 8) fällt auf, dass die besten Ergebnisse für 2 und 3 Schritte gemeinsam haben, dass der Validierungsloss ein Plateau bei knapp unter $4 \cdot 10^{-3}$ aufweist. Beide Konfigurationen

verbessern sich danach aber noch einmal deutlich. Dies wäre grundsätzlich auch für das Netz mit 4 Schritten denkbar. Da dieses oft nicht über ein Plateau mit homogenem Output bei einem Validierungsloss von ungefähr $0.95 \cdot 10^{-3}$ hinauskommt, trat dies bei lediglich 3 Trainingsanläufen aber nicht auf. Möglicherweise ist hier aber auch die geringere Parameterzahl durch die kleineren Dense-Layer limitierend. Allgemein lässt sich aber nicht sagen, dass über verschieden aufgebaute Netzwerke hinweg eine höhere Parameterzahl zu einem besseren Ergebnis führt. Das Typ A Netzwerk mit 12, 24 Filtern hat, trotz geringeren Wertes der Loss-Funktion und somit einer besseren Anpassung an die Trainingsdaten, schlechter abgeschnitten als das des Typs B mit 12, 12, da die Anpassung wohl wenig generalisierend und nicht auf die Validierungsdaten übertragbar war.

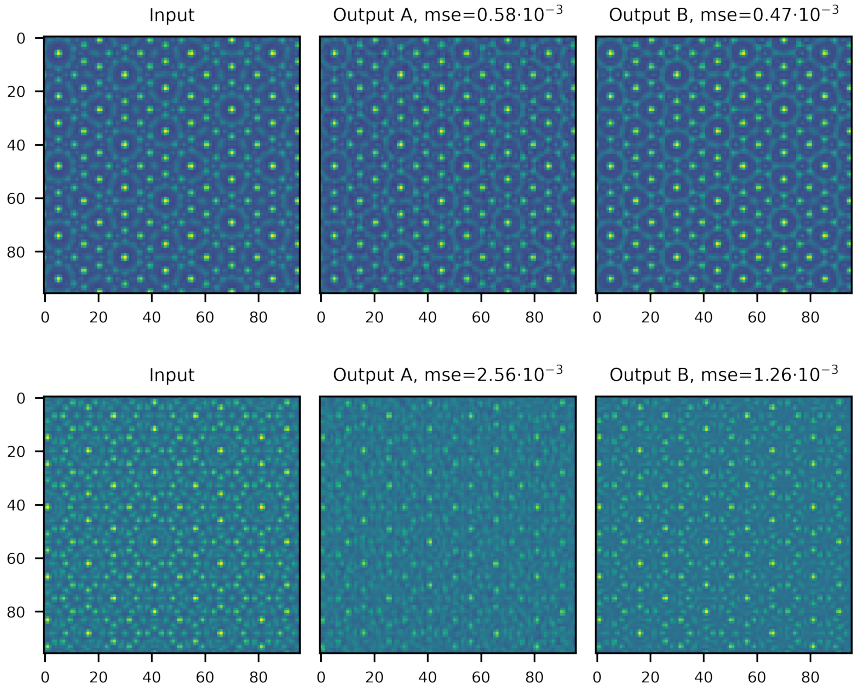


Abbildung 7: Input und zugehöriger Output des jeweils besten Netzwerkes von Typ A bzw. B für 5-zählige Muster mit angegebenem Loss

Ansonsten kamen die besten Ergebnisse grundsätzlich mit den höchsten getesteten Filterzahlen zustande (12 initiale Filter und exponentielle Erhöhung), abgesehen von den 4-Schritte Netzwerken, welche in dieser Filterkonfiguration wiederholt in einen Zustand mit homogenem Output lief. Hierbei handelt es sich wohl um ein lokales Minimum der Loss-Funktion. Außerdem ist auffällig, dass mit Zunahme der Schritte sich der Validierungsloss reduziert, sich die Loss-Funktion bezüglich der Trainingsdaten aber erhöht. Dies sieht man auch in Grafik 8, dass Tranings- und Validierungsloss für höhere Schrittzahlen näher beisammen liegen als für kleine. Das Netzwerk ist also nicht mehr so gut in der Lage sich an die Trainingsdaten anzupassen, ist aber besser darin, generelle Muster zu erkennen, was zu einem letztendlich besseren Ergebnis führt.

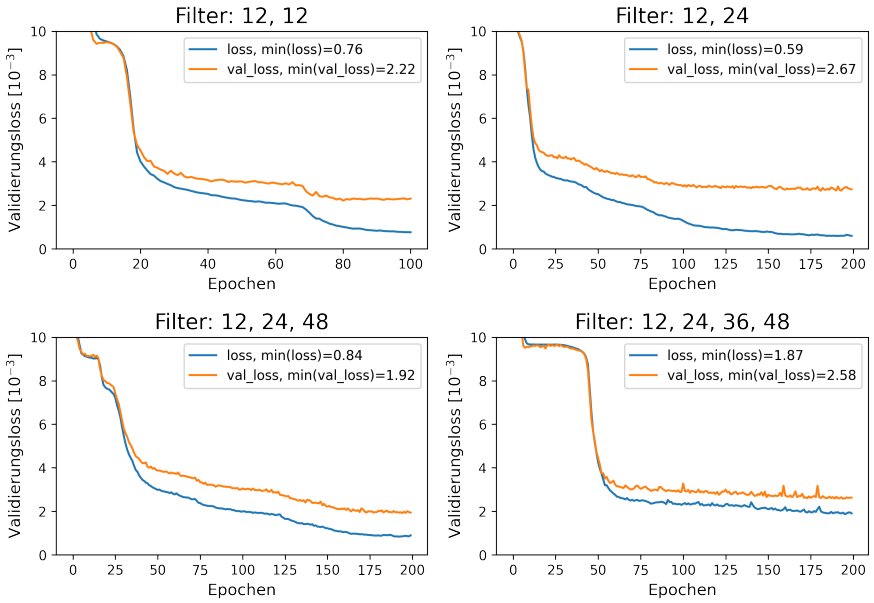


Abbildung 8: Trainingsfortschritt verschiedener TypA-MaxPool Netzwerke für 5-zählige Muster. Für jeden Encoderschritt sind die Filter angegeben.

5.2. Training der 17-zähligen Muster

Die Ergebnisse für die 17-zähligen Muster (Tabelle 4) scheinen auf den ersten Blick deutlich besser als für die 5-zähligen. Betrachtet man allerdings den Netzwerkoutput in Grafik 9 wird deutlich, dass keine der Konfigurationen in der Lage war, die Muster adäquat zu rekonstruieren.

verwendetes Netzwerk	Trainingsergebnis n17 $[10^{-3}]$		Filteranzahl	Parameterzahl $[10^3]$
	loss	val loss		
A-MaxPool-2Schritte	0.72	1.00	8-8	395
A-MaxPool-3Schritte	0.89	1.04	8-8-8	105
A-MaxPool-4Schritte	0.62	1.01	12-24-48-96	597
B-MaxPool-2Schritte	0.19	1.06	12-12	596
B-MaxPool-3Schritte	0.86	1.04	12-12-12	160
B-MaxPool-4Schritte	0.93	1.02	8-16-24-32	156

Tabelle 4: Trainingsergebnisse verschiedene Netzwerke mit Maximum-Pooling für Muster mit 17-zähliger Symmetrie.

Dies wird auch deutlich, wenn man den Validationloss ins Verhältnis zu einem optimierten homogenen Output setzt. Für die 5-zähligen Muster entspricht ein homogener Output einem Loss von ca. $9.5 \cdot 10^{-3}$. Das beste Trainingsergebnis ist mit $1.23 \cdot 10^{-3}$ also mehr als Faktor 7 besser, wohingegen bei 17-zähligen Mustern ein homogener Output bereits ein Loss von $1.95 \cdot 10^{-3}$ erreicht, sodass das trainierte Netzwerk nicht einmal einen

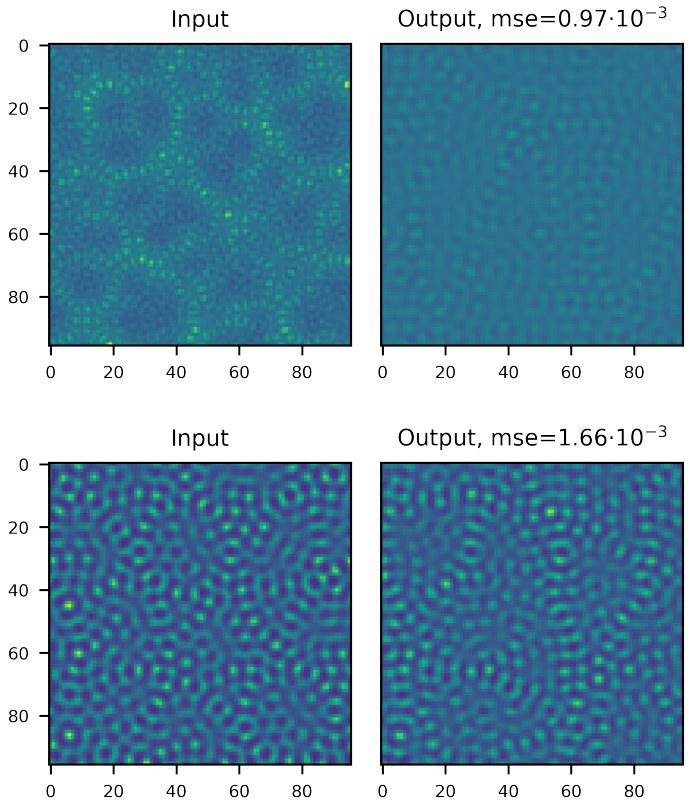


Abbildung 9: Input und zugehöriger Output des besten A-MaxPool-2Schritte Netzwerkes für Muster 17-zähliger Symmetrie mit angegebenem Loss

Faktor 2 erreicht. Das erklärt auch, warum das Netzwerk mit wenig komplexem Aufbau und geringen Filterzahlen maximale Performance erreicht, da alle Konfigurationen nur grundlegende Features rekonstruieren konnten.

5.3. Verbesserter Netzwerkaufbau

Aufgrund der Tatsache, dass das bisherige Netzwerk an der Reproduktion der 17-zähligen Muster gescheitert ist, soll nun ein Aufbau gefunden werden, der dazu in der Lage ist. Da weder eine Erhöhung der Anzahl an Convolutional-Layern pro Schritt (Typ A zu B) noch mehr Schritte oder Filter eine Verbesserung erzielten, wurde der Ansatz gewählt die Pooling bzw. Upsampling Operationen zu verändern, um den Informationsverlust dort zu reduzieren, indem zusätzliche freie Parameter eingeführt werden. Eine Möglichkeit, dies zu erreichen, wäre das Ersetzen durch dichte Layer, wobei die Anzahl an Input und Output Neuronen beibehalten wird. Die Parameterzahl dichter Layer ist durch das Produkt der Zahl Neuronen von In- und Output $(N_{in}+1)\cdot N_{out}$ gegeben, wobei "+1"die Offsets b berücksichtigt (vgl. Gleichung 1). Da die Neuronenzahl durch die Filteranzahl f sowie die Kantenlänge l definiert ist und im Encoder jeweils eine Reduktion der Kantenlänge um Faktor 2 bei konstanter Filterzahl angestrebt wird, ist die resultierende Parameterzahl gegeben durch

$$N_{parameter} = (l_{in}^2 f + 1)\cdot l_{out}^2 f = (l_{in}^2 f + 1)\cdot \frac{1}{4} l_{in}^2 f \approx \frac{1}{4} f^2 l_{in}^4.$$

Die Parameterzahl ist also von 4. Ordnung in der Kantenlänge des Inputs, was sehr Rechen- und Arbeitsspeicher aufwändig ist. Aus diesem Grund wurden dichte Layer als Ersatz für das Pooling wieder verworfen.

Stattdessen wurden Convolutional-Layer genutzt, bei denen der Kernel nicht auf jedes Pixel angewendet wird, sondern nur in bestimmten Abständen. Dieser Abstand wird Stride genannt. Im Falle von Stride = 2 wird der Kernel sowohl in x- als auch in y-Richtung also nur auf jedes zweite Pixel angewendet.Da sich durch jede Kernelanwendung ein Pixel im Output ergibt, halbiert sich dadurch die Kantenlänge wie zuvor mit Maximum-Pooling. Dies lieferte bereits in der Bildklassifikation mit einer Kernel Größe von 3 gute Ergebnisse [9] und wurde deswegen auch hier so verwendet.

verwendetes Netzwerk	Trainingsergebnis n05 $[10^{-3}]$		Filteranzahl	Parameterzahl $[10^3]$
	loss	val loss		
A-ConvPool-2Schritte	0.37	1.14	12,24	1203
A-ConvPool-3Schritte	0.46	0.88	12,24,36	535
A-ConvPool-4Schritte	0.91	1.21	8,16,32,64	429
B-ConvPool-2Schritte	0.46	1.38	12,24	1207
B-ConvPool-3Schritte	0.52	0.89	12,24,36	535
B-ConvPool-4Schritte	0.81	1.06	12,24,36,48	355

Tabelle 5: Trainingsergebnisse verschiedene Netzwerke mit Convolutional-Pooling für Muster mit 5-zähliger Symmetrie.

Sowohl der Trainings- als auch der Validierungsloss der 5-zähligen Muster ist für Typ A und B nun etwa gleichwertig, wie in Tabelle 5 gezeigt wird. Außerdem schneiden

beide Netzwerktypen mit 3 Schritten am besten ab, jeweils mit dem getesteten Maximum an initialen Filtern sowie einer linearen Erhöhung. Im Vergleich zu den Resultaten mit Maximum-Pooling in Tabelle 3 hat sich der Validierungsloss für Typ A etwa um 50% reduziert. Dadurch ist die Performance von Typ A nun etwa gleichwertig mit den Netzwerken von Typ B, da sich letztere nur mäßig verbessern.

verwendetes Netzwerk	Trainingsergebnis n17 $[10^{-3}]$		Filteranzahl	Parameterzahl $[10^3]$
	loss	val loss		
A-ConvPool-2Schritte	0.64	1.00	6,6	297
A-ConvPool-3Schritte	0.11	0.27	12,24,48	715
A-ConvPool-4Schritte	0.79	0.95	6,12,18,24	129
B-ConvPool-2Schritte	0.63	0.96	8,8	398
B-ConvPool-3Schritte	0.73	0.98	12,12,12	168
B-ConvPool-4Schritte	0.91	0.92	12,12,12,12	66

Tabelle 6: Trainingsergebnisse verschiedene Netzwerke mit Convolutional-Pooling für Muster mit 17-zähliger Symmetrie.

Die größte Verbesserung trat allerdings für 17-zählige Muster mit der Konfiguration A und 3 Schritten auf. Hier reduzierte sich das Validierungsloss im Vergleich zum vorher besten Ergebnis fast um einen Faktor 4 von 1.00 (vgl. Tabelle 4) auf 0.27 (vgl. Tabelle 6). Diese Verbesserung ist auch deutlich am Output des Netzwerkes erkennbar, wie in Grafik 10 gezeigt wird. Allerdings sieht man dort auch, dass nach wie vor Muster existieren, vor allem recht filigrane mit geringem Kontrast, welche nicht rekonstruiert werden.

Andere Netzwerkkonfigurationen führten für 17-zählige Muster zu keiner deutlichen Verbesserung unabhängig von der verwendeten Filterzahl. Die besten Ergebnisse kamen dort, wie zuvor, mit geringen Filterzahlen zustande. Im Gegensatz dazu erzielte das gute Ergebnisse liefernde A-ConvPool-3Schritte Netzwerk die besten Resultate mit möglichst hohen getesteten Filterzahlen ($filter_n = 2^{n-1} \cdot 12$). Die vorhandenen Kapazitäten konnten dort also tatsächlich in bessere Netzwerkperformance umgesetzt werden.

Auf Grund dessen, dass das Netzwerk des Typs A mit Convolutional-Pooling und 3 Schritten das einzige war, welches sowohl für 5- als auch für 17-zählige Muster gute Ergebnisse liefert, soll dieses im Folgenden auch für das Training auf Muster verschiedener Symmetrien gleichzeitig zum Einsatz kommen.

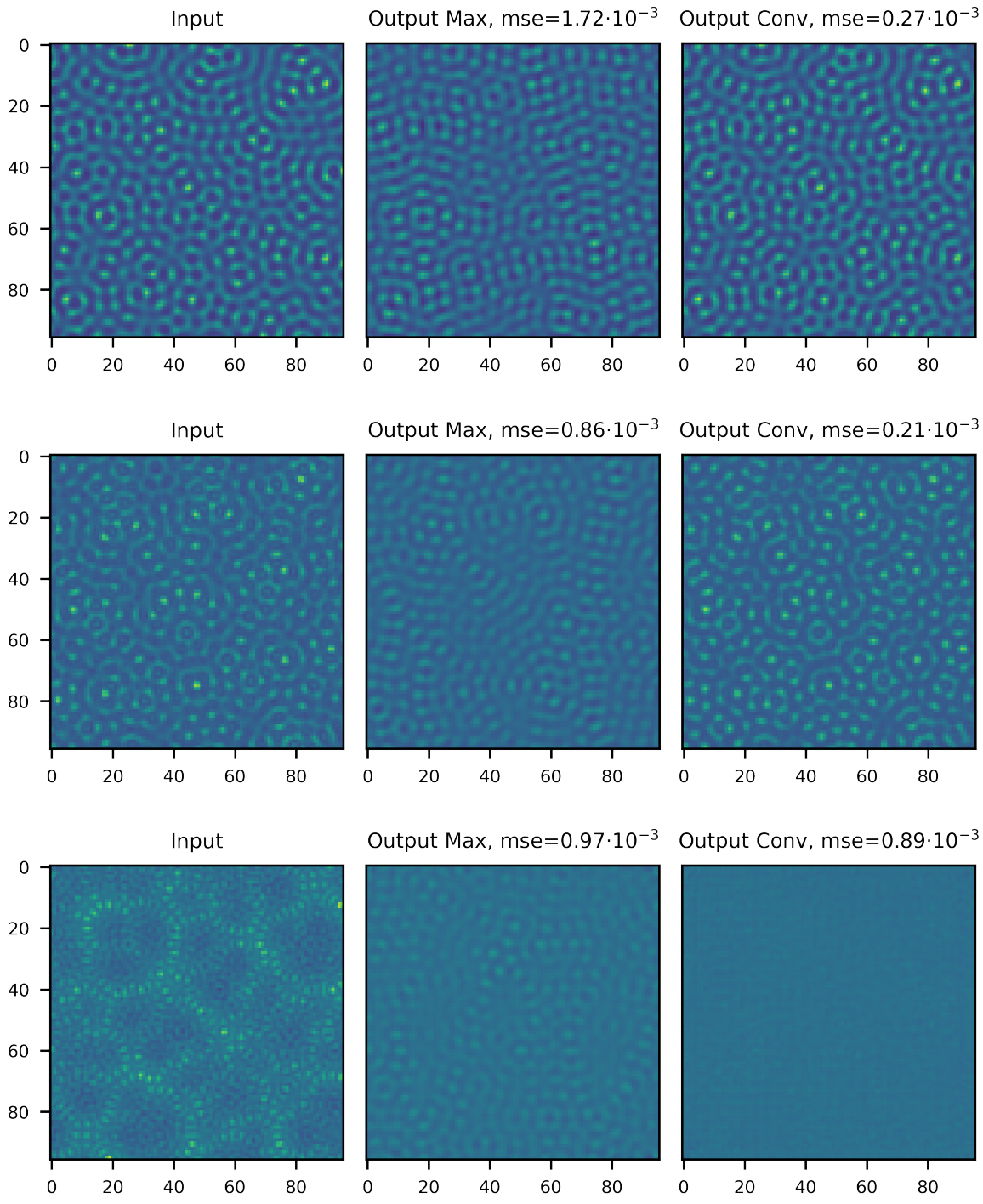


Abbildung 10: Input und zugehöriger Output des jeweils besten Netzwerkes mit Maximum- bzw Convolutional-Pooling für 17-zählige Muster mit angegebenem Loss

6. Training aller Symmetrien

Da das Ziel aber ist, nicht nur Quasikristallmuster einer, sondern verschiedener Symmetrien zu charakterisieren, wurde der Autoencoder nun auf den gesamten Datensatz aller 2- bis 18-zähliger Symmetrien trainiert. Als Netzwerk-Konfiguration wurde dafür die A-ConvPool-3Schritte Version mit linear ansteigender Filterzahl gewählt. Eine exponentielle Zunahme der Filter lieferte mitunter etwas bessere Ergebnisse, allerdings bestand eine deutlich höhere Chance, dass das Training nicht über einen Zustand mit homogenem Output hinausging. Das Training wurde über maximal 100 Epochen durchgeführt oder bis über 10 Epochen keine Verbesserung des Validierungslosses eintrat. In Epochen betrachtet ist die Trainingszeit also kürzer, allerdings sind es nun ca. 17 mal so viele Trainingsdaten wie für die einzelnen Symmetrien. Dies führt insgesamt, zusammen mit den erhöhten Filterzahlen, zu einem deutlich längeren Trainingsprozess. Die Resultate für verschiedene Filterzahlen sind in Tabelle 7 dargestellt.

verwendete Filterzahl	Trainingsergebnis $[10^{-3}]$		Parameterzahl $[10^3]$
	loss	val loss	
8,16,24	1.44	1.41	336
12,24,36	1.04	1.04	535
16,32,48	0.83	0.81	756
24,48,72	0.60	0.57	1259

Tabelle 7: Trainingsergebnisse des A-ConvPool-3Schritte Netzwerks mit unterschiedlichen Filterzahlen für Muster mit 2- bis 18-zähliger Symmetrie.

Man sieht, dass eine initiale Filterzahl von 8 deutlich zu knapp bemessen ist. Dies war zu erwarten, da selbst für einzelne Symmetrien das neuronale Netz mindestens bis zu $f_0 = 12$ skalierte. Ab $f_0 = 16$ liegt der Validierungsloss dann zwischen den Ergebnissen für 5- und 17-zählige Muster (vgl. Tabelle 5 und 6). Demzufolge sollte das Netzwerk die meisten Muster also relativ gut rekonstruieren können, wie in Grafik 11 auch zu sehen.

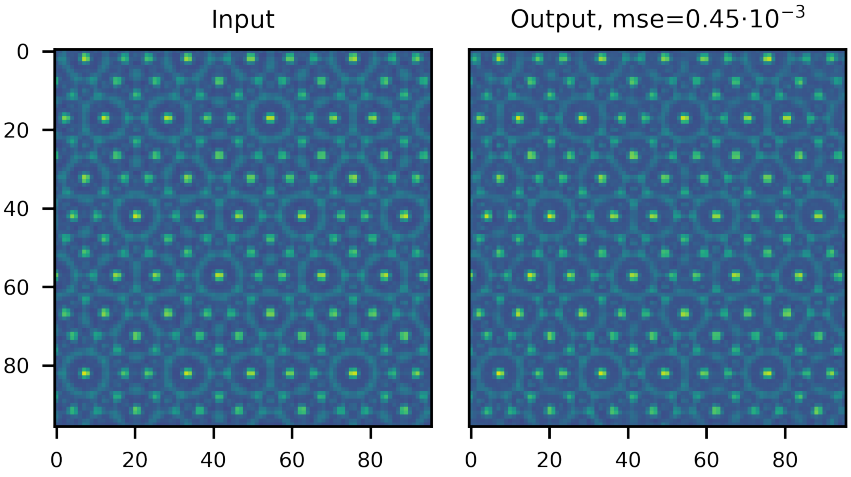
Um zu analysieren, welche Symmetrien besonders gut oder schlecht rekonstruiert werden, wurde nun der Validierungsloss für alle Symmetrien einzeln berechnet und in Tabelle 8 dargestellt. Zum Vergleich dient das Ergebnis für das selbe Setup mit reduzierter Filterzahl (12, 24, 36), jeweils ausschließlich auf die entsprechende Symmetrie trainiert. Das Training wurde äquivalent zu Kapitel 5 durchgeführt. Dabei schneidet das Netzwerk mit 24 initialen Filtern über alle Symmetrien hinweg besser ab, als das mit 16 Filtern. Auffallend ist die deutliche Verbesserung gerade bei hohen Symmetrien wie 15 und 16. Hier erweisen sich die zusätzlichen Filter als besonders hilfreich.

Überraschenderweise ist das Resultat für 2-zählige Muster mit dem separat trainierten Muster schlechter. Vergleicht man den Netzwerkoutput in Grafik 12 oben, so sieht man an den linken und rechten Bildrändern sowie in den Ecken des einzeln trainierten Netzes, dass dies vor allem an Faltungsartifakten liegt. Grundsätzlich schlecht schneiden die auf alle Symmetrien trainierten Netzwerke bei Mustern größerer Primzahlen wie 11 , 13 und 17 ab, wenn man das Ergebnis mit dem Einzeltraining vergleicht (Grafik 12

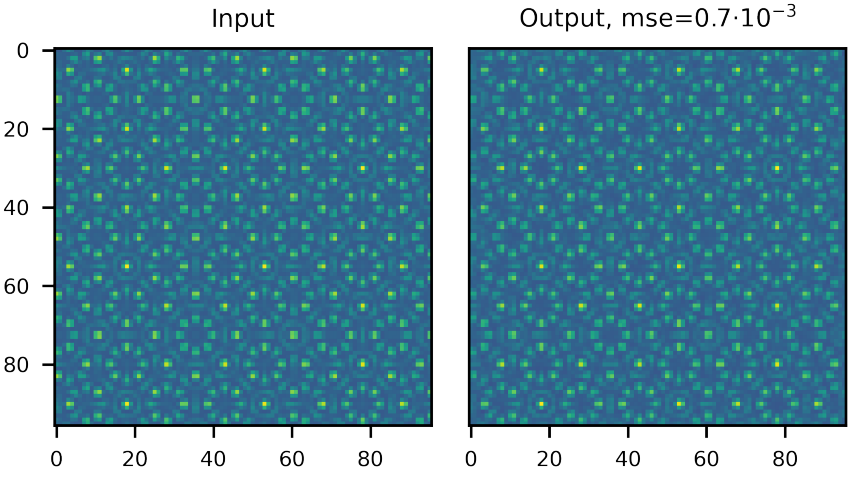
Symmetrie	Trainingsergebnis (val loss) $[10^{-3}]$		
	gesamt		einzeln
	$f_0 = 16$	$f_0 = 24$	$f_0 = 12$
2	0.39	0.21	0.67
3	0.29	0.22	0.16
4	0.29	0.17	0.13
5	1.16	1.11	0.86
6	0.19	0.15	0.15
7	0.78	0.65	0.73
8	0.51	0.48	0.13
9	0.48	0.40	0.44
10	0.70	0.62	0.77
11	1.85	1.39	0.66
12	0.29	0.23	0.20
13	1.61	1.24	0.41
14	0.70	0.57	0.47
15	1.08	0.55	0.20
16	1.72	0.44	0.27
17	1.28	0.95	0.31
18	0.33	0.25	0.18

Tabelle 8: Trainingsergebnisse des A-ConvPool-3Schritte Netzwerks trainiert auf alle Symmetrien nach Symmetrien aufgeschlüsselt und Vergleich mit dem Ergebnis bei Training ausschließlich auf die jeweilige Symmetrie.

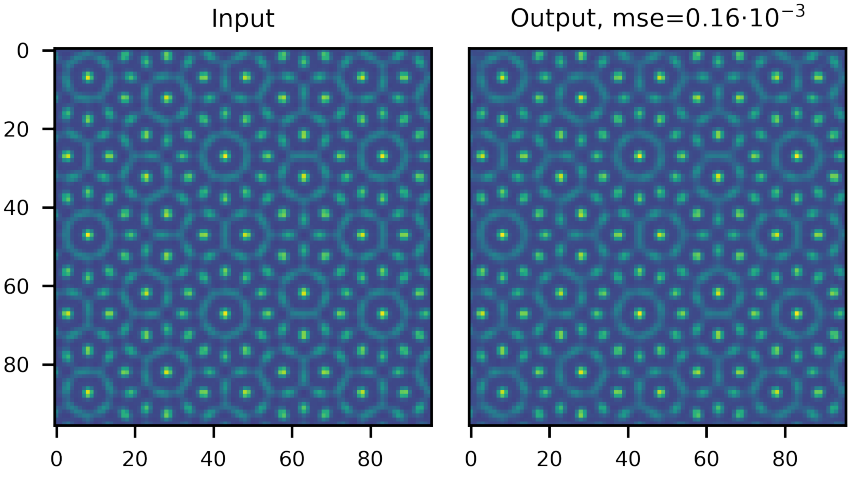
unten sowie Grafik 13). Dies ist wenig überraschend, da diese Symmetrien selbst im Einzeltraining schwer zu rekonstruieren sind. Man sieht dies sowohl beim oberen der 11-zähligen als auch beim unteren der 13-zähligen Muster, welche grundsätzlich nicht rekonstruiert werden konnten. In den Beispielen 17-zähliger Muster sieht man, dass aber immer noch mehr Strukturen zumindest ansatzweise rekonstruiert werden konnten, als es mit den MaxPool Netzwerken der Fall war (Grafik 10). Eventuell könnte eine weitere Erhöhung der Filterzahl noch zur Verbesserung beitragen, allerdings nimmt auf Grund der großen Datenmenge der Rechenaufwand stark zu. Außerdem ist wegen des schlechten Ergebnisses im Einzeltraining fraglich, ob dies tatsächlich zu einer deutlichen Verbesserung führen würde, oder ob ein anderer Netzwerkaufbau notwendig wäre.



5-zähliges Muster

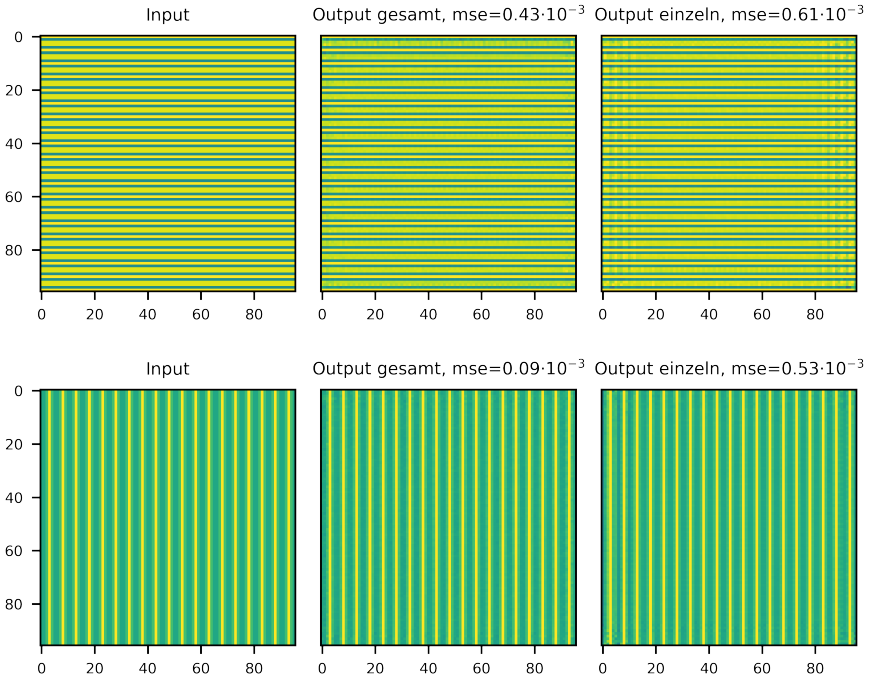


8-zähliges Muster

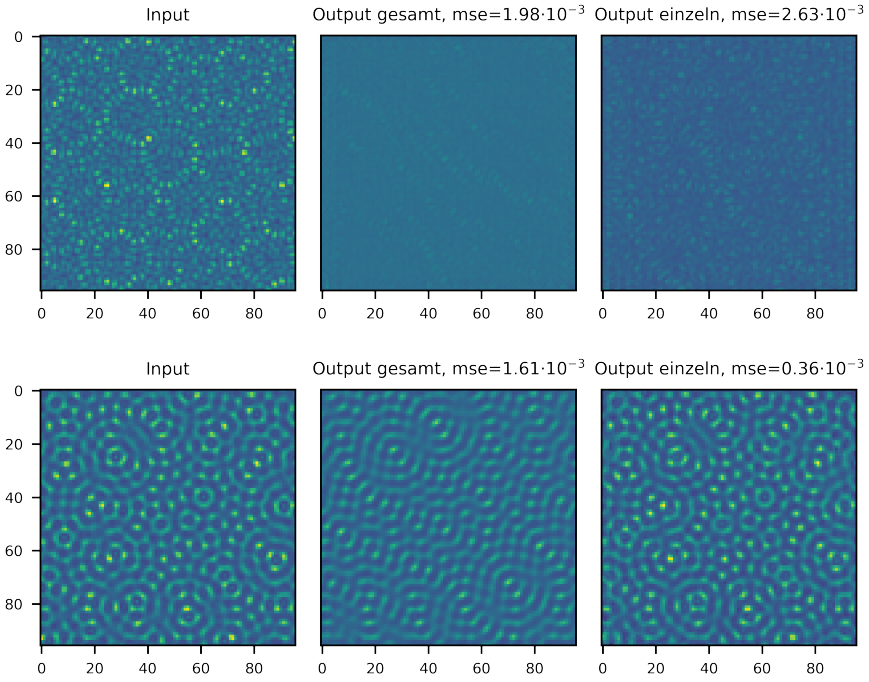


12-zähliges Muster

Abbildung 11: Output des auf alle Symmetrien trainierten Netzwerks ($f_0 = 24$) für verschiedene Muster

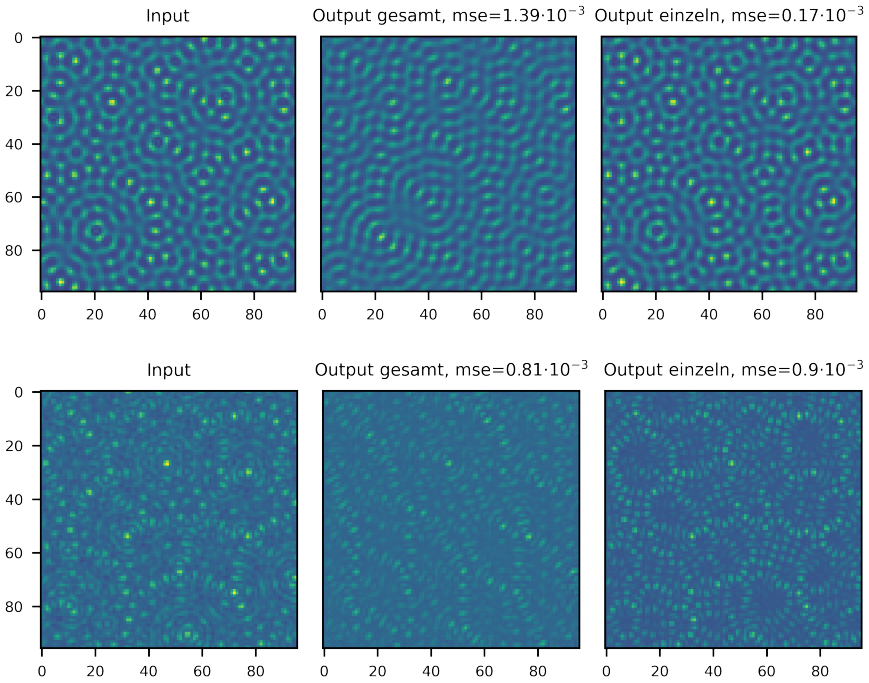


2-zählige Muster

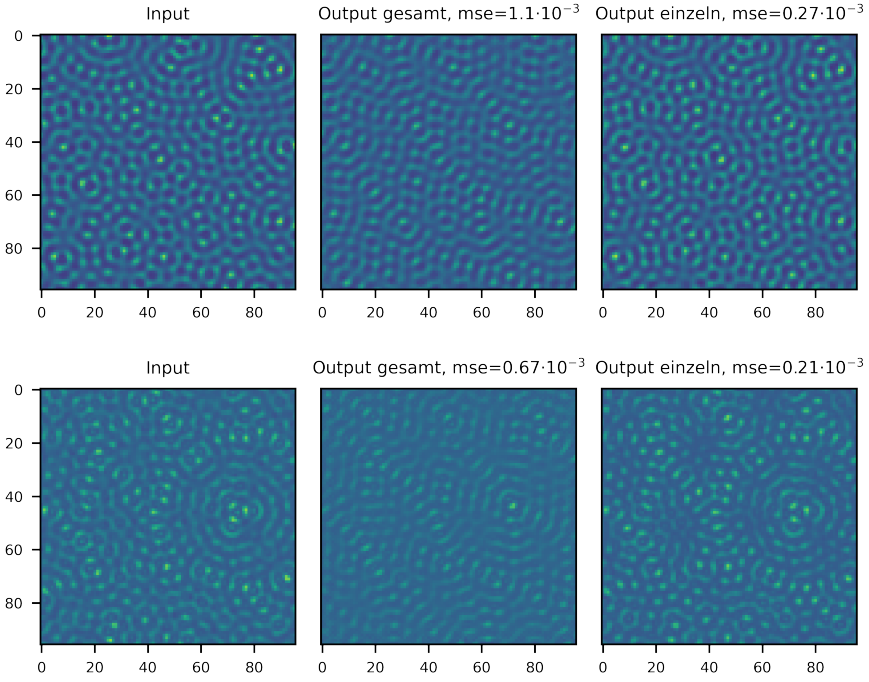


11-zählige Muster

Abbildung 12: Vergleich des Outputs des auf alle Symmetrien mit dem nur auf die jeweilige Symmetrie trainierten Netzwerks



13-zählige Muster



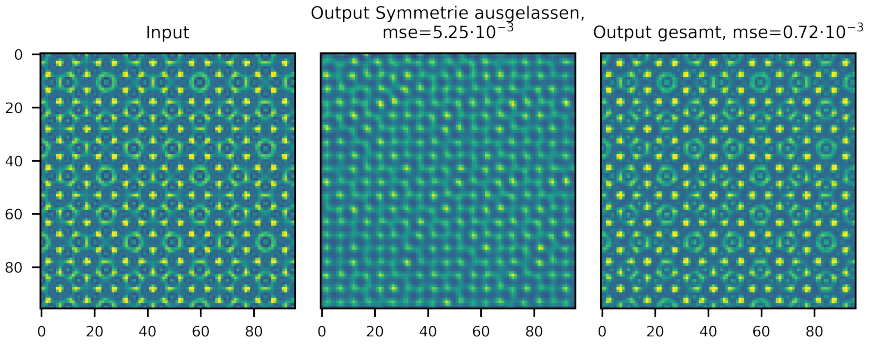
17-zählige Muster

Abbildung 13: Vergleich des Outputs des auf alle Symmetrien mit dem nur auf die jeweilige Symmetrie trainierten Netzwerks

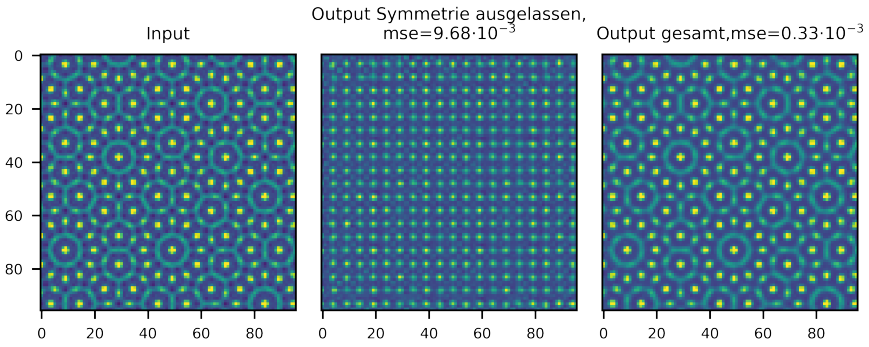
7. Training unter Auslassen einzelner Symmetrien

Im Weiteren soll untersucht werden, ob der Autoencoder nicht nur in der Lage ist von den Trainingsdaten auf die Validierungsdaten derselben Symmetrien zu schließen, sondern ob es möglich ist, Muster bestimmter Symmetrien zu rekonstruieren, auf welche es davor nicht trainiert wurde. Dafür wurde die beste Netzwerkkonfiguration aus dem vorangegangenen Kapitel weiterverwendet (A-ConvPool-3Schritte, Filter: 24, 48, 72). Beim Training aber jeweils eine der Symmetrien 8, 10 oder 12 ausgeschlossen. Ansonsten wurde das Training analog zum Training auf alle Symmetrien (Kapitel 6) durchgeführt.

Symmetrie	Trainingsergebnis (val loss) $[10^{-3}]$		
	ausgelassen	gesamt	einzeln
8	6.74	0.48	0.13
10	0.78	0.62	0.77
12	8.65	0.23	0.20

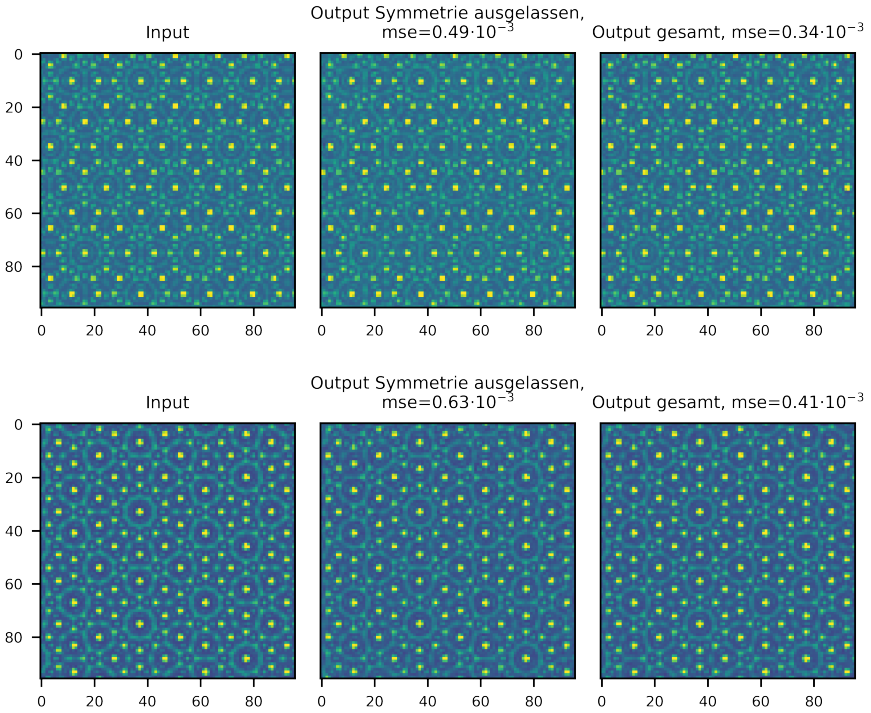


8-zähliges Muster



12-zähliges Muster

Abbildung 14: Vergleich des Outputs, des auf alle Symmetrien trainierten Netzwerks (jeweils rechts) mit Outputs eines Netzwerks, bei dem die spezifische Symmetrie beim Training ausgelassen wurde (jeweils Mitte).



10-zählige Muster

Abbildung 15: Vergleich des Outputs, des auf alle Symmetrien trainierten Netzwerks (jeweils rechts) mit Outputs eines Netzwerks, bei dem die spezifische Symmetrie beim Training ausgelassen wurde (jeweils Mitte).

Sowohl im Falle der 8- als auch der 12-zähligen Muster hat sich durch das Entfernen der entsprechenden Symmetrie aus den Trainingsdaten der Loss deutlich erhöht. Dies spiegelt sich auch grafisch im Netzwerkoutput wieder, wie in Grafik 14 zu sehen. Die Muster dieser Symmetrien können vom Netzwerk nicht mehr rekonstruiert werden, da nur die 4- aber nicht die 3-zählige Symmetriekomponente erkannt wird. Anders sieht es allerdings im Falle der 10-zähligen Muster aus. Hier lieferte das Netzwerk ein ähnliches Resultat wie mit dem auf alle Symmetrien beziehungsweise dem nur auf 10-zählige Muster trainierten Netzwerk. Entsprechend gut wurden die Beispielmuster in Grafik 15 rekonstruiert.

Die Ursache, weshalb die 10-zählige Symmetrie deutlich besser funktionierte als die 8- oder 12-zählige, liegt darin, dass sich die 10- den 5-zähligen Mustern kaum unterscheiden. Vergleicht man beide Symmetrien in Grafik 16 anhand einiger Beispiele, erkennt man, das sich die grundlegenden Features sehr ähnlich sind. Durch das Training auf die 5-zähligen Muster lernt das Netzwerk also bereits die meisten Strukturen zu rekonstruieren, die es auch für die 10-zähligen benötigt. Das Netzwerk ist also nicht in der Lage über gelernte Symmetrien hinaus zu generalisieren und Quasikristalle im Allgemeinen zu rekonstruieren.

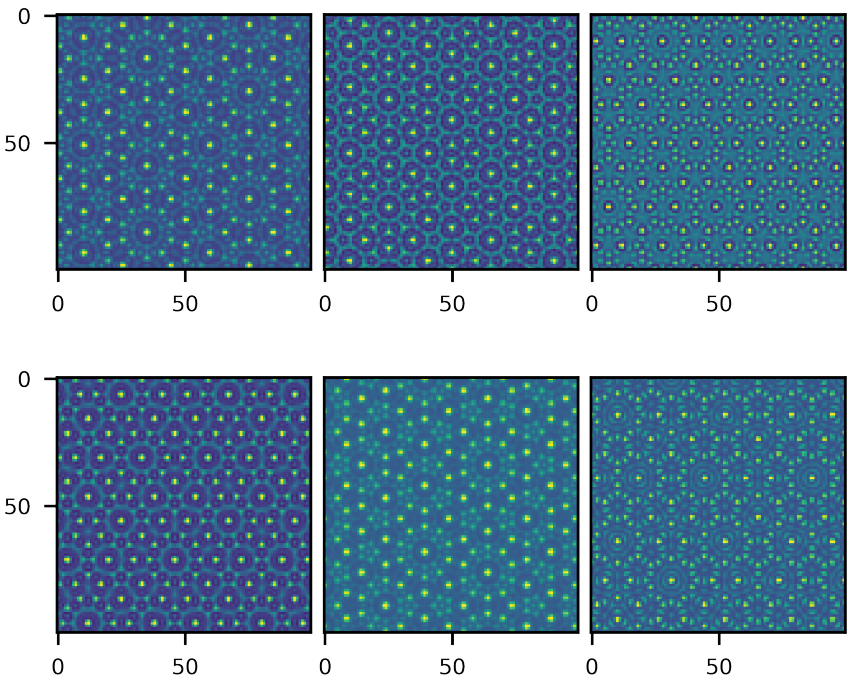


Abbildung 16: Vergleich einiger Beispielmuster 5-zähliger Symmetrie (obere Zeile) mit solchen 10-zähliger (untere Zeile).

8. Mustererzeugung

Darüber hinaus stellt sich die Frage, ob es möglich ist einen Autoencoder zur Mustererzeugung oder -Klassifikation zu nutzen. Als Ausgangsbasis wurde dafür das bereits trainierte A-ConvPool-3Schritte Netz aus Kapitel 6 genutzt (Filter: 24,48,72). Die Symmetrien 11, 13 und 17, welche dieser Autoencoder nicht zu rekonstruieren gelernt hatte, wurden im Folgenden ausgeschlossen.

8.1. Mustererzeugung und -klassifikation mittels weiterer neuronaler Netze

Zunächst stellte sich die Frage, ob es möglich ist, aus den Bottleneckwerten eines Musters auf die Simulationsparameter zurück zu schließen. Hierfür wurde ein weiteres neuronales Netz programmiert, das ausschließlich aus dichten Layern besteht (siehe Appendix B). Als Input bekommt dieses Netzwerk den Encoderoutput, also die Bottleneckwerte, eines Musters. Dann wird das Netz darauf trainiert die zugehörigen 42 Simulationsparameter zu auszugeben. Als Loss-Funktion wurde wieder die mittlere quadratische Abweichung gewählt. Außerdem wurden die beiden Parameter, welche die Position des Musterauschnitts definieren mit dem Faktor $\frac{1}{900}$ multipliziert, sodass sie wie alle anderen Parameter im Intervall $[0, 1]$ liegen.

Wie in Grafik 17 zu sehen, stimmen zwischen Simulationsparametern und Output des

Verbindungsnetzes grob die Bereiche ähnlicher Werte überein, aber nicht die Feinheiten. Vor allem auf die Parameter, welche die Bildausschnitte definieren (jeweils 5 Werte von rechts), konnte nicht geschlossen werden. Mit dem hier verwendeten einfachen dichten Netzwerks war es also nicht möglich, zur Klassifikation der Muster auf dem Encoder aufzubauen. Es wäre vermutlich ein fortgeschritteneres, speziell auf die Klassifikation zugeschnittenes, neuronales Netz notwendig.

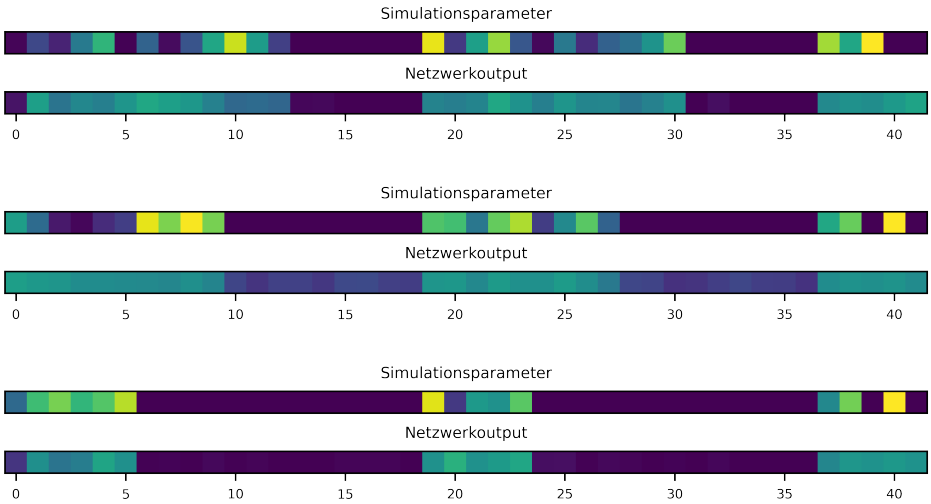


Abbildung 17: Vergleich zwischen den tatsächlichen Simulationsparametern und dem Ergebnis des Klassifikationsnetzes.

Ähnlich zum zusätzlichen, für die Klassifikation genutzten Netz, wurde ein weiteres Netzwerk trainiert, welches die Simulationsparameter so mit dem Decoder verbinden soll, dass das zugehörige Muster entsteht. Die Muster, die auf diese Weise erzeugt wurden, zeigen aber in jedem Fall einen näherungsweise homogenen Output, wie in Abbildung 18) für 2 Beispiele gezeigt wird. Für Details zum verwendeten Verbindungsnetzwerk siehe Appendix A.

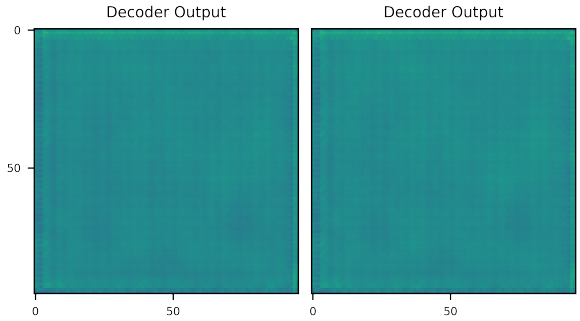


Abbildung 18: Output des Decoders beim Versuch aus 2 verschiedenen Sätzen Simulationsparameter die dazugehörigen Muster zu erzeugen.

8.2. Mustererzeugung mittels gleichverteilter Zufallszahlen

Da es nicht möglich war, gezielt aus bestimmten Parametern neue Quasikristallmuster zu erzeugen, soll untersucht werden, ob es funktioniert, beliebige Muster durch Einsetzen von Zufallszahlen direkt in das Bottleneck zu generieren.

Zunächst wurden dafür die Bottleneckwerte für alle Muster berechnet, wobei das gleiche Basisnetz und die gleichen Symmetrien wie im letzten Abschnitt verwendet wurden. Für jedes der 42 Bottleneckneuronen wurde daraus Mittelwert μ und Standardabweichung σ berechnet, um gleichverteilte Zufallszahlen aus einem $\mu \pm 2\sigma$ Intervall ziehen zu können. Diese wurden dann in das Bottleneck eingesetzt und der zugehörige Decoderoutput für einige Beispiele ist in Grafik 20. Man erkennt deutlich die 2-, 3- und die 4-zähligen Symmetrien. Außerdem sind Anteile der 8-zähligen Muster zu erkennen (mittlere Spalte Muster 1 und 2 von oben). Allerdings enthalten die Muster noch relativ viele Störfaktoren, sodass keine vollständigen Muster entstehen.

8.3. Mustererzeugung mittels normalverteilter Zufallszahlen

Ein Ansatz, um den Output zu verbessern, war es, die Verteilung der Bottleneckwerte genauer zu betrachten. Bereits an einigen Beispielen in Abbildung 19 fällt auf, dass die Werte der Neuronen im Bottleneck näherungsweise normalverteilt sind. Aus den Bottleneckwerten kann nun zusätzlich zu den Mittelwerten auch die Kovarianzmatrix berechnet werden, was es ermöglicht, die Zufallszahlen aus einer multivariaten Normalverteilung zu ziehen. Dadurch werden auch Korrelationen zwischen den Neuronen mit einbezogen, was theoretisch zu besseren Mustern führen sollte. Vergleicht man nun einige resultierende Muster aus Grafik 21 mit denen aus gleichverteilten Zufallszahlen (Grafik 20), so erkennt man, dass auch Muster fast ohne Störeinflüsse entstehen. Außerdem treten nun auch Muster höherer Symmetrien wie 12 (oben rechts) auf. Ungefähr die Hälfte der Ergebnisse zeigt nach wie vor kein definierbares Muster, die anderen können nun aber definitiv als Quasikristallmuster interpretiert werden.

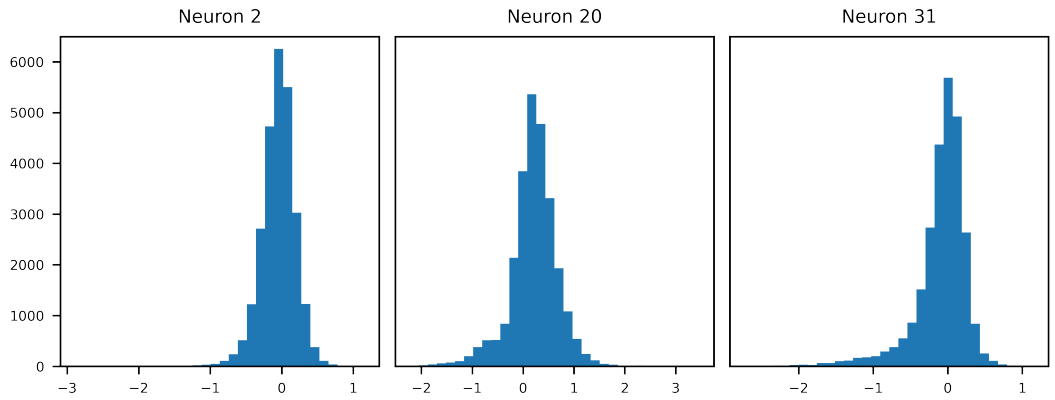


Abbildung 19: Histogramme der Bottleneckwerte für 3 Beispielneuronen

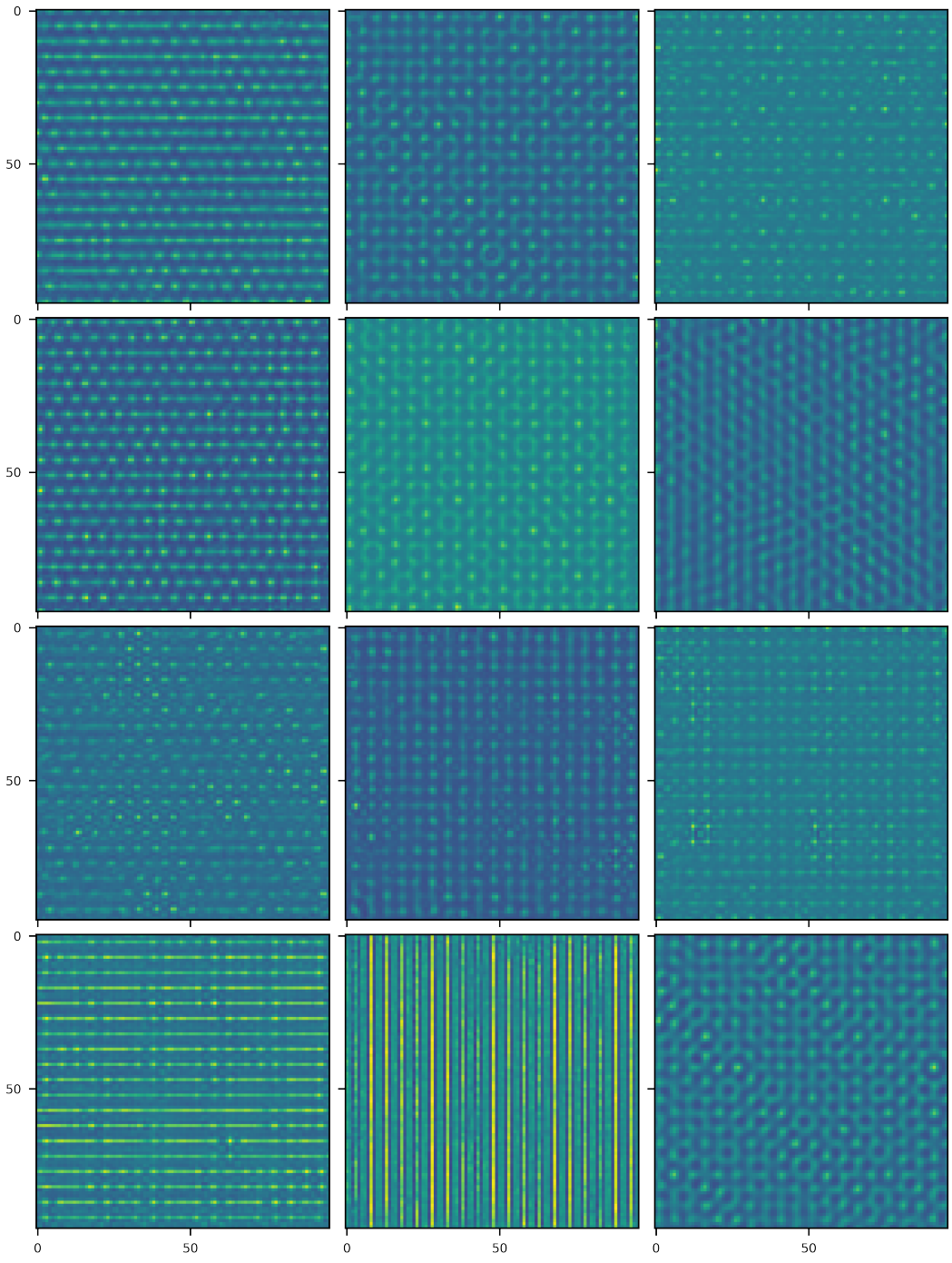


Abbildung 20: Beispielmuster, die als Output des Decoders für gleichverteilte Bottleneckwerte entstanden sind.

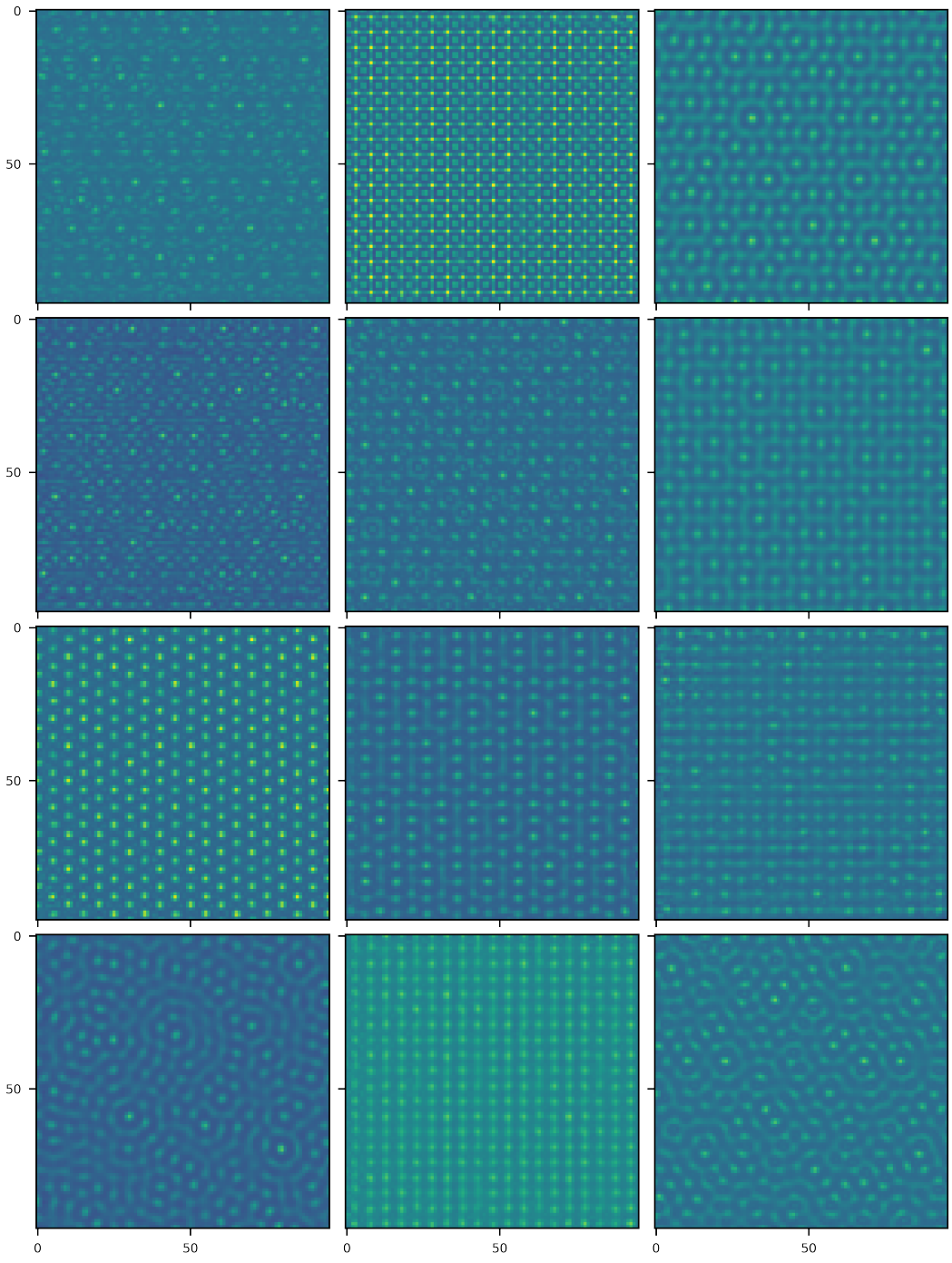


Abbildung 21: Muster erzeugt durch Einsetzen von Zufallszahlen in das Bottleneck, gezogen aus einer multivariaten Normalverteilung

9. Reduktion der Bottleneck Größe

Außerdem soll getestet werden, wie weit die Größe des Bottlenecks reduziert werden kann, ohne dass das Netzwerk die Fähigkeit zur Musterrekonstruktion verliert. Gleichzeitig um herauszufinden, welche Symmetrien am längsten erhalten bleiben. Dafür wurde grundsätzlich ein reduzierter Datensatz nur mit den 2- bis 10-zähligen sowie den 12-zähligen Mustern verwendet. Auf Grund weniger zu erlernender Symmetrien wurde außerdem die Filterzahl auf 16, 32, 48 reduziert mit ansonsten gleicher Autoencoderkonfiguration wie in den Kapiteln zuvor.

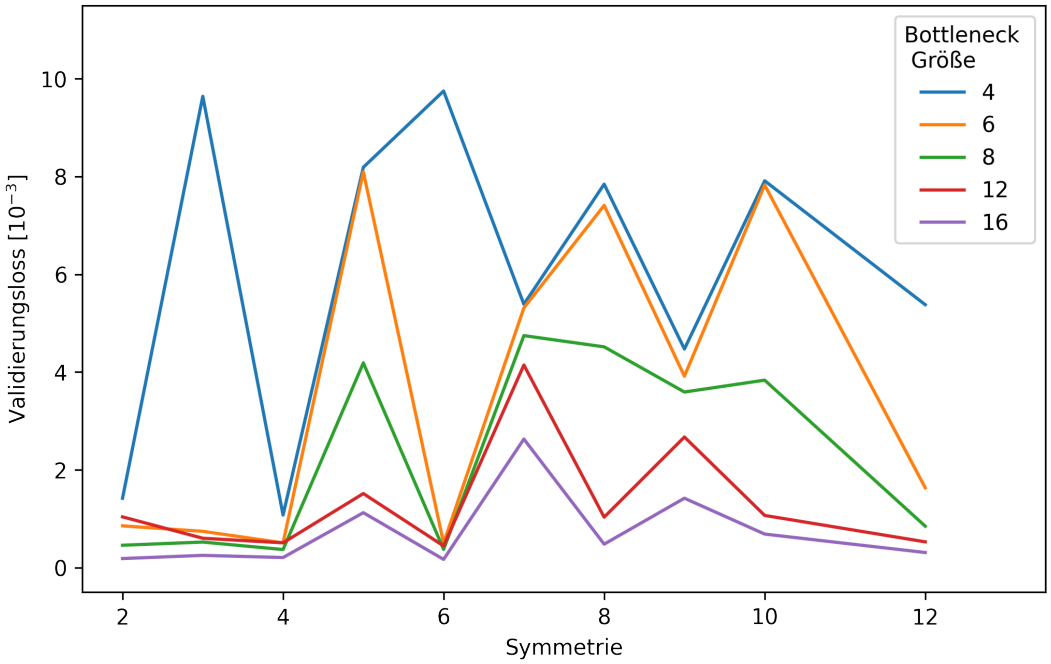
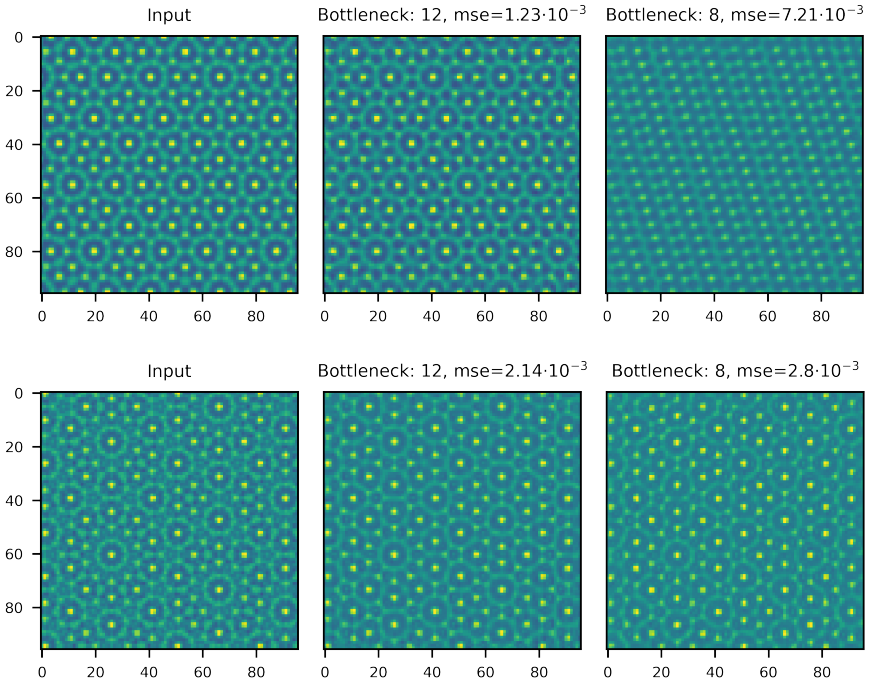
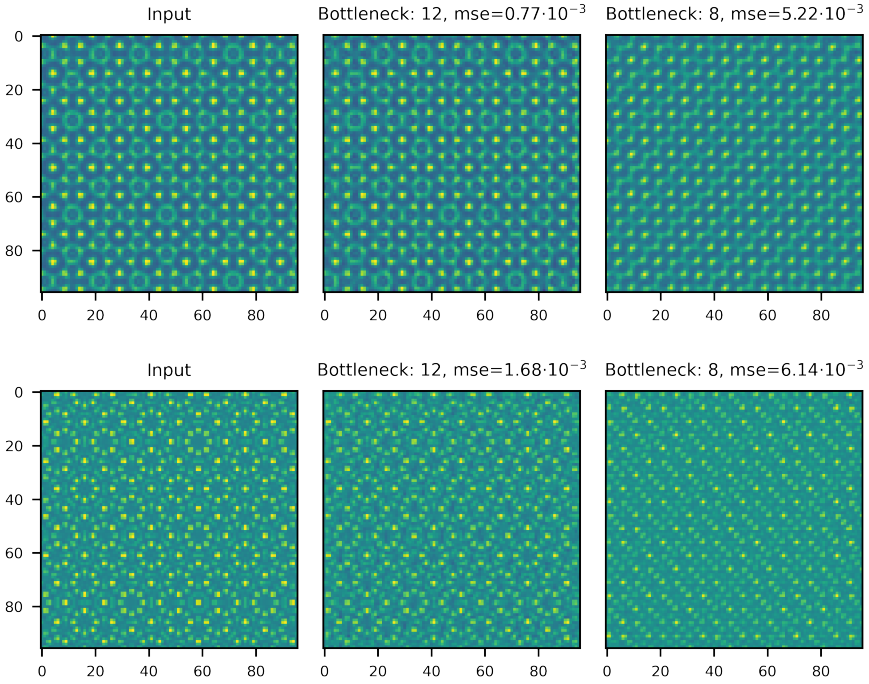


Abbildung 22: Validierungsloss für verschiedene Bottleneck Größen nach Symmetrien aufgeschlüsselt

Beginnend bei einer bereits deutlich reduzierten Bottleneck Größe von 16 wird das Bottleneck sukzessive verkleinert. Der Validierungsloss nach Symmetrie aufgeschlüsselt ist in Abbildung 22 dargestellt. Mit 16 Bottleneck Neuronen ist das Netzwerk noch in der Lage, alle getesteten Symmetrien bis auf die 7- und 9-zähligen zu rekonstruieren. Eine Reduktion auf 12 führt lediglich zu einer leichten Erhöhung des Validierungslosses, qualitativ ändert sich der Netzwerkoutput kaum. Deutlichere Unterschiede treten das erste Mal bei der Verkleinerung von 12 auf 8 auf, da sowohl 5-, 8- und 10-zählige Muster nur noch bedingt rekonstruiert werden. Diese Entwicklung verdeutlicht sich noch beim Schritt von 8 zu 6 Bottleneck Neuronen. Einige Beispiele hierfür sind in Grafik 23 dargestellt, wobei die 10-zähligen Muster auf Grund der großen Ähnlichkeit zu den 5-zähligen hier nicht gezeigt werden.

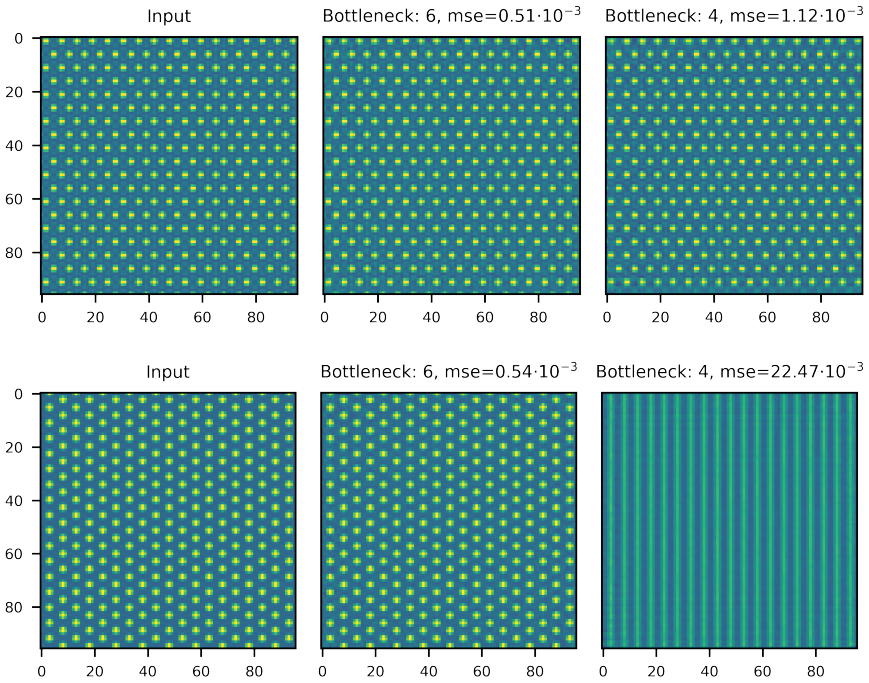


5-zählige Muster

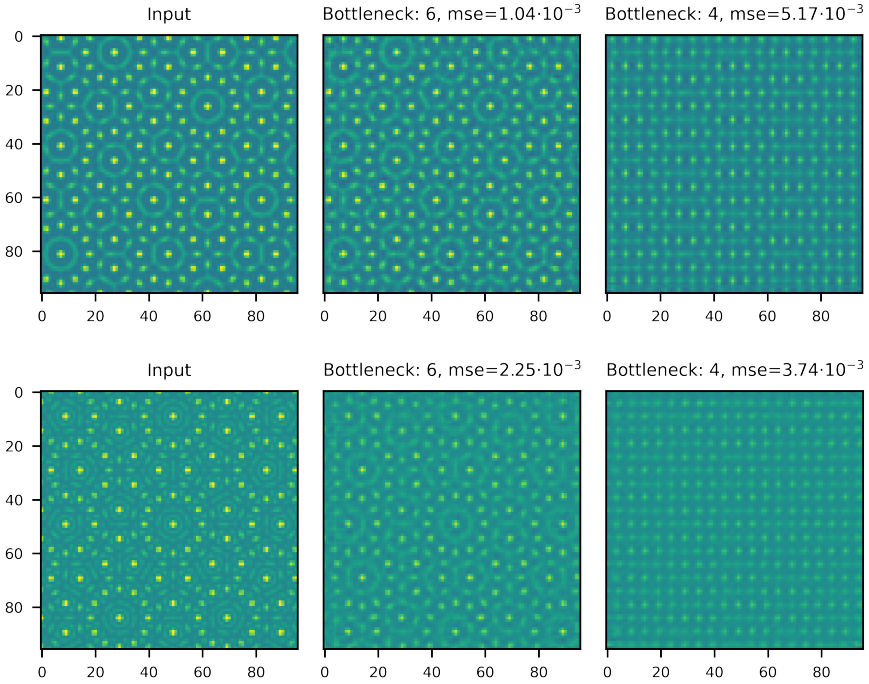


8-zählige Muster

Abbildung 23: Vergleich des Netzwerkoutputs für 12 beziehungsweise 8 Neuronen im Bottleneck



3-zählige Muster



12-zählige Muster

Abbildung 24: Vergleich des Netzwerkoutputs für 6 beziehungsweise 4 Neuronen im Bottleneck

Wird die Bottleneckgröße weiter reduziert auf 4, so werden nur noch die 3-zähligen Muster mit horizontalen Reihen erfolgreich rekonstruiert, wohingegen die mit vertikalen Reihen als 2-zählige Muster interpretiert werden. Dies ist auch bei den 6-zähligen zu beobachten. Die 12-zähligen Muster, welche sich trotz ihrer hohen Symmetrie bis hierhin als rekonstruierbar erwiesen hatten, können nun nicht mehr reproduziert werden. Da die 12-zähligen Muster sowohl eine 3- als auch eine 2-/4-zählige Komponente besitzen, liegt die Ursache dafür vermutlich in der nur noch bedingten Rekonstruierbarkeit der 3-zähligen Muster. Zugehörige Grafiken sind in Abbildung 24 gezeigt.

Netzwerkanalyse

Um darüber hinaus zu untersuchen, welche Parametrisierung das Netzwerk für die Muster findet, soll der Effekt einzelner Bottleneck Neuronen betrachtet werden. Dafür wurde das bereits trainierte Netzwerk mit einer Bottleneckgröße von 12 verwendet. Für die Bottleneckwerte der trainierten Symmetrien wurde Mittelwert μ_n und Standardabweichung σ_n für jedes Neuron n berechnet. Setzt man nun alle Neuronen im Bottleneck gleich 0, so erhält man als Decoderoutput ein homogenes Muster. Der Effekt einzelner Neuronen wurde dann betrachtet, indem jeweils 1σ (Abbildung 25) beziehungsweise 3σ (Abbildung 26) eingesetzt werden. Insgesamt fällt auf, dass ein größerer Bottleneckwert zu einem größeren Wertebereich der Outputmuster führt. Die 1σ-Muster sind insgesamt abstrakter und schwieriger Strukturen einer bestimmten Symmetrie auszumachen. Außerdem sind mehr höherzählige Symmetrien zu erkennen (zum Beispiel 8-zählig unten links) als dies bei den 3σ-Mustern der Fall ist. Diese zeigen vor allem 3- und 4-zählige Symmetrien.

Die Muster niedrigzahliger Rotationssymmetrie kommen also vermutlich in erster Linie durch die Aktivierung weniger Bottleneckneuronen aber mit großen Zahlenwerten zustande, wohingegen höherzählige Muster durch Überlagerung vieler, aber weniger stark aktivierter Neuronen zusammengesetzt werden.

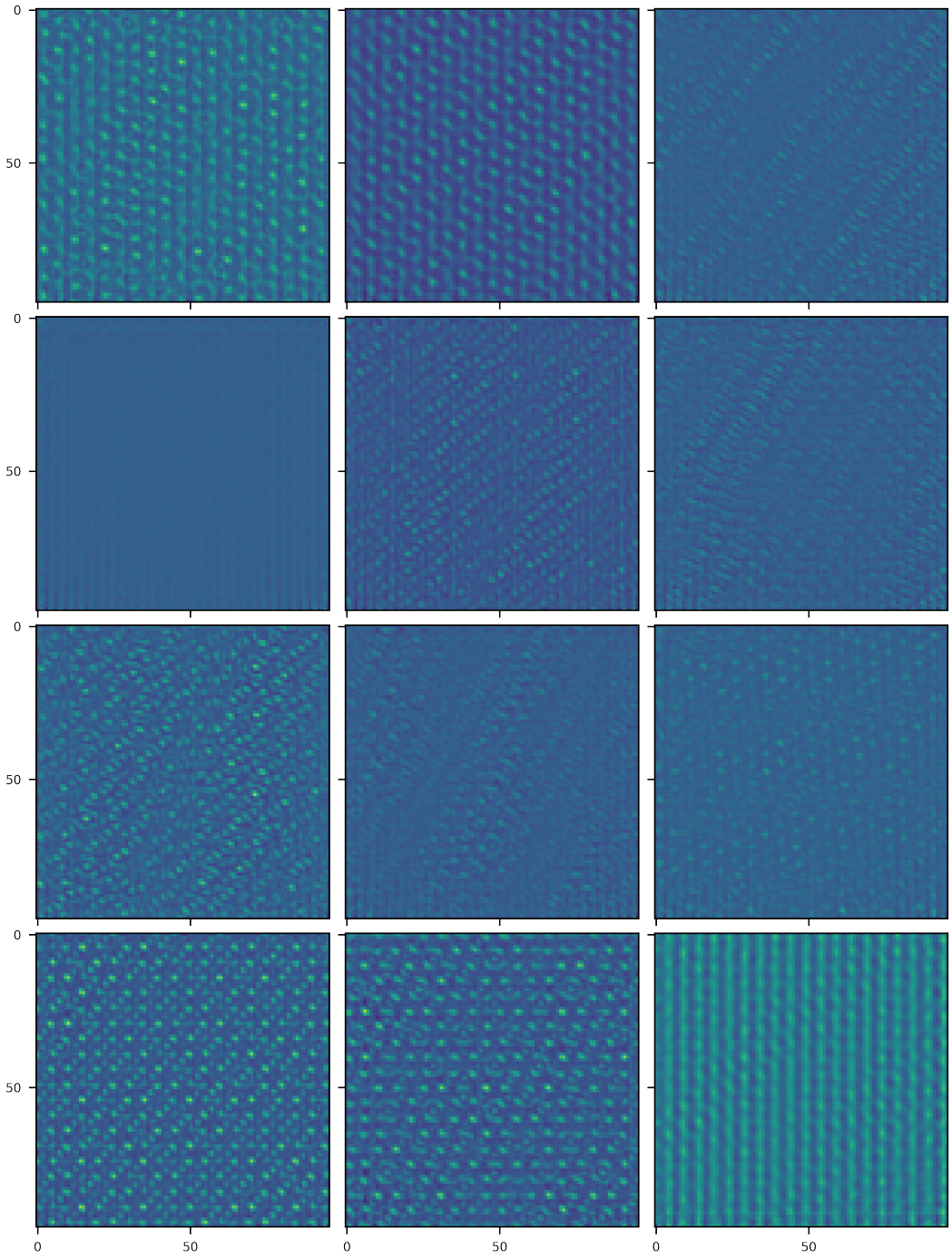


Abbildung 25: Muster entstanden aus der Aktivierung eines Einzelnen der 12 Bottleneck Neuronen. Stärke der Aktivierung war eine Standardabweichung bezüglich der für alle Muster berechneten Werte des jeweiligen Neurons.

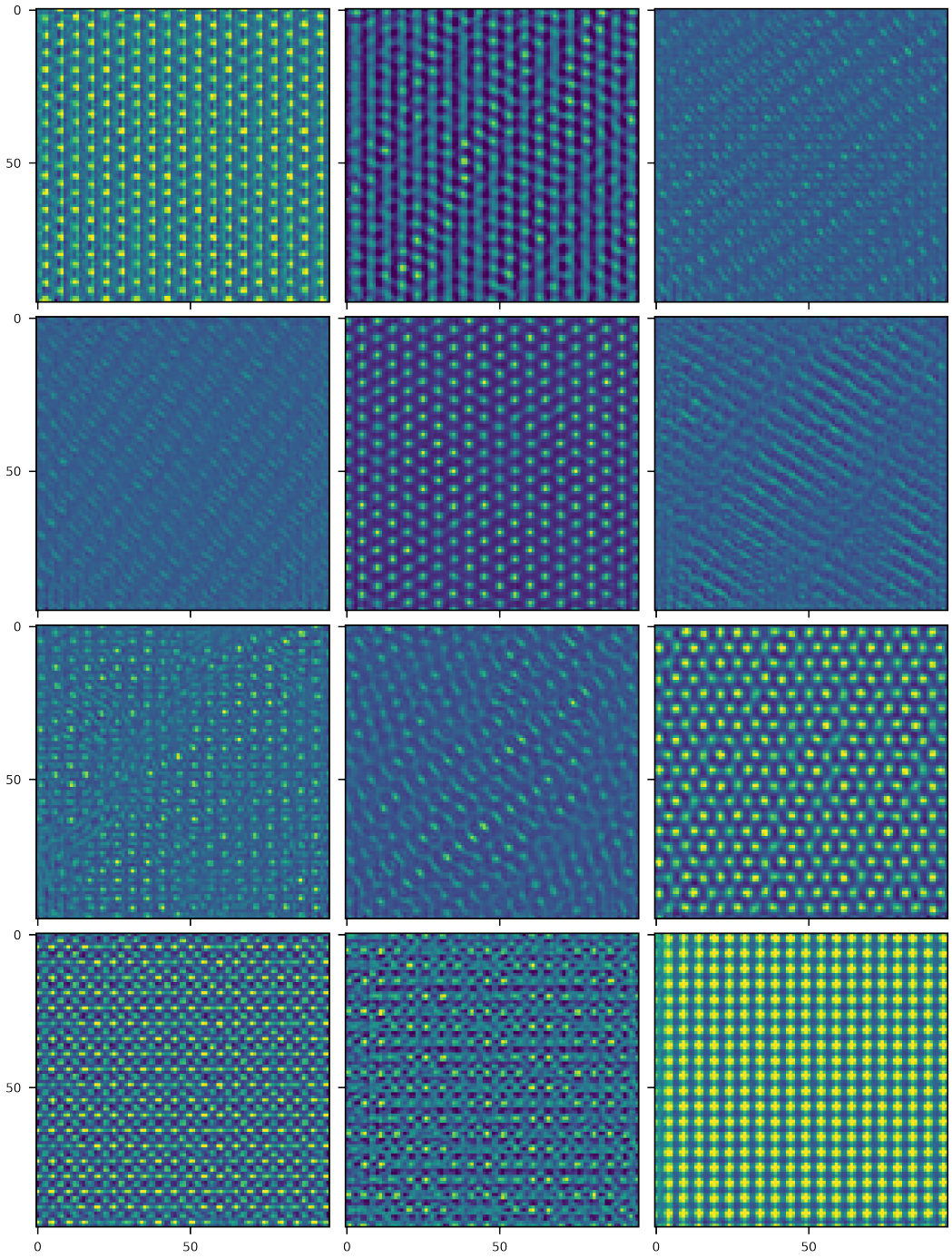


Abbildung 26: Muster entstanden aus der Aktivierung eines Einzelnen der 12 Bottleneck Neuronen. Stärke der Aktivierung war 3 Mal die Standardabweichung bezüglich der für alle Muster berechneten Werte des jeweiligen Neurons.

10. Fazit und Ausblick

Wie in Kapitel 6 gezeigt wurde, ist es möglich, ein neuronales Netz zu programmieren, welches die meisten Quasikristall-Muster anhand einer geringen Parameterzahl charakterisieren kann. Dies sollte mittels weiterer Verbesserungen an der Netzwerkstruktur auch auf alle getesteten Muster erweiterbar sein. Die Parametrisierung von unbekannten Mustern gelernter Symmetrien war für das neuronale Netzwerk gut möglich. Eine Generalisierung auf unbekannte Symmetrien war aber nicht gegeben.

Obwohl es nicht möglich war, vollständig auf die zur Mustererzeugung genutzen Parameter zurück zu schließen, stimmte die Anzahl der von 0 verschiedenen Parameter näherungsweise überein (vgl. Abb. 17). Da darüber die Symmetrie definiert ist, könnten die Muster nach dieser klassifiziert werden. Ein Rückschluss auf die ursprünglichen Parameter ist mit einem geänderten, speziell darauf zugeschnittenen neuronalen Netz nicht auszuschließen.

Außerdem könnte es mittels eines Variational Autoencoders [10] oder Generative Adversarial Networks [11] möglich werden, spezifische Quasikristallmuster zu erzeugen. Mit dem in dieser Arbeit verwendeten Autoencoder gelang dies nur in manchen Fällen ohne Kontrolle über Symmetrie oder Parameter.

Des Weiteren war das neuronale Netz in der Lage, Muster 8 verschiedener Rotationssymmetrien (2-6, 8, 10 und 12) mittels nur 12 Werten für eine Rekonstruktion ausreichend zu parametrisieren. Zur Erzeugung dieser Muster wurden aber bis zu 30 Parameter verwendet. Das Netzwerk scheint zumindest für die wichtigsten Features günstigere Parametrisierungen zu finden. Wie diese allerdings genau aussehen, bedarf weiterer Analyse.

Literatur

[1] Peter J. Lu and Paul J. Steinhardt. Decagonal and quasi-crystalline tilings in medieval islamic architecture. *Science*, 315, 2007.

[2] D. Shechtman, I. Blech, D. Gratias, and J. W. Cahn. Metallic phase with long-range orientational order and no translational symmetry. *Physical Review Letters*, 53(20), 1984.

[3] Ron Lifshitz. What is a crystal? *Zeitschrift für Kristallographie*, 222, 2007.

[4] Matthias Sandbrink. Tailored colloidal quasicrystals. Doktorarbeit, 2014.

[5] F. Marquardt. Machine learning and quantum devices. arXiv:2101.01759v2, 2021.

[6] Raúl Rojas. Neural networks - a systematic introduction. Springer-Verlag, 1996.

[7] Christopher M. Bishop. Neural networks for pattern recognition. Clarendon Press, 1995.

[8] Tensorflow. `https://www.tensorflow.org/api_docs/python/tf`.

[9] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. arXiv:1412.6806v3, 2015.

[10] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. arXiv:1906.02691v3, 2019.

[11] Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. arXiv:1710.07035v1, 2017.

Appendix

A. Methode zur Mustererzeugung

Die Muster, welche als Trainingsdaten für das neuronale Netzwerk dienten, wurden mittels folgender Gleichung berechnet:

$$M = \frac{1}{K}\left[a \cdot M^{(1)} + \sum_{l=1}^{N} b_l^{(+)} M_l^{(2+)} + \sum_{l=1}^{N} b_l^{(-)} M_l^{(2-)}\right].$$

Dabei sind a, $b_l^{(+)}$, $b_l^{(-)}$ freie Parameter, N die Symmetrie des Musters und K ein Normierungsfaktor gegeben durch

$$K = N^2 a + N \sum_{l=1}^{N} b_l^{(+)} + N \sum_{l=1}^{N} b_l^{(-)}.$$

Die Komponenten $M^{(1)}$, $M_l^{(2+)}$ und $M_l^{(2-)}$ ergeben sich aus

$$M^{(1)}(\vec{r}) = N \cdot \sum_{j=1}^{N} \cos\left(\vec{k_j} \cdot \vec{r}\right)$$

$$M_l^{(2+)}(\vec{r}) = \sum_{j=1}^{N} \cos\left((\vec{k_j} + \vec{k_{j+l}}) \cdot \vec{r}\right)$$

$$M_l^{(2-)}(\vec{r}) = \sum_{j=1}^{N} \cos\left((\vec{k_j} - \vec{k_{j+l}}) \cdot \vec{r}\right)$$

mit $\vec{k_j} = \left(\cos\frac{2\pi j}{N}, \sin\frac{2\pi j}{N}\right)$. Die Gesamtzahl der Parameter a, $b_l^{(+)}$, $b_l^{(-)}$ beläuft sich also auf $2N + 1$. Um die Ausschnitte zu spezifizieren kommen noch 5 weitere Parameter hinzu: Die Position Δx, Δy des Ausschnitts, sowie s_x, s_y, s_r für Spiegelungen an x- und y-Achse sowie an der ersten Winkelhalbierenden.

B. Neuronale Netze zur Verbindung das Bottlenecks mit den Simulationsparametern

Für die Verbindung zwischen Encoder und Simulationsparametern wurde folgendes Netzwerk genutzt:

Layertyp	Neuronen	Aktivierungsfunktion
Input = Encoder Output	42	-
Dense	420	ReLu
Dense	840	ReLu
Dense	840	ReLu
Dense	420	ReLu
Dense	42	39 linear + 3 HardSigmoid
Output = Parameter		

Da die Parameter für die Spiegelungen nur die Werte 0 oder 1 annehmen, wurde im letzten Layer für die 3 korrespondierenden Neuronen die HardSigmoid Aktivierungsfunktion verwendet. Dies sollte es dem Netzwerk erleichtern diese Parameter zu erlernen.

Für die Verbindung zwischen Simulationsparametern und Decoder wurde ein sehr ähnliches Netzwerk genutzt:

Layertyp	Neuronen	Aktivierungsfunktion
Input = Parameter	42	-
Dense	420	ReLu
Dense	840	ReLu
Dense	840	ReLu
Dense	420	ReLu
Dense	42	linear
Output = Decoder Input		

Eidesstattliche Erklärung

Ich versichere, dass ich die Bachelorarbeit ohne fremde Hilfe und ohne Benutzung anderer als der angegebenen Quellen angefertigt habe und dass die Arbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegen hat. Alle Ausführungen der Arbeit, die wörtlich oder sinngemäß übernommen wurden, sind als solche gekennzeichnet.

Erlangen, den Jonas Buba