Progressive Bildkompression mittels hierarchischer Vektorquantisierung von Subbändern

Diplomarbeit

am Lehrstuhl für Kognitive Systeme
des Instituts für Informatik und Praktische Mathematik
der Christian–Albrechts–Universität zu Kiel

betreut von Dipl.–Ing. Udo Mahlmeister

vorgelegt von

Britta Heitkämper

Kiel, im Januar 1998

Inhaltsverzeichnis

Kapitel 1

Einleitung

In den letzten Jahren hat die Benutzung des Internets zunehmend an Bedeutung gewonnen. Weltweit sind die Computer von Firmen und Privatleuten über dieses Netzwerk miteinander verbunden, und es werden täglich ganze Fluten von Daten übertragen. Über das Medium Internet ist es möglich, mit weit entfernten Personen zu kommunizieren beziehungsweise Daten auszutauschen. Wurden früher auf diesem Wege noch hauptsächlich Texte übertragen, so sind es mittlerweile immer mehr Bilder und auch Audio–Dateien. Dies kommt vor allem in einer Variante des Internets vor: im WWW (World Wide Web). Hier stehen die sogenannten Homepages zur Verfügung, auf denen man sich durch Anklicken der Links, hinter denen sich weitere WWW–Adressen verbergen, von Seite zu Seite „weiterhangeln" kann. Aber zum Beispiel sind auch Bilddatenbanken denkbar, in denen man vom anderen Ende der Welt aus nach bestimmten Bildern suchen und sich diese dann „herunterladen" kann. Vor allem dafür ist unsere progressive Art der Codierung angebracht. Man stelle sich nur einmal vor, man sucht nach einem Bild und sieht sich dazu erst mehrere (falsche) an, bis man das richtige findet. Wenn die Übertragung jedes einzelnen Bildes insgesamt auch nur 5 Sekunden dauern würde, so wäre das für dieses Durchsuchen der Datenbank schon zu lange. Die progressive Codierung ist dann nützlich, wenn die Übertragung der Daten so lange dauert, daß sich ein Abbruch nach Lösung der Aufgabe (wenn man zum Beispiel feststellt, daß es sich nicht um das gewünschte Bild handelt) noch lohnt. Für diese Anwendungen ist es ebenso notwendig, die Bilddateien kompakt zu halten wie zum Beispiel für HDTV (hochauflösendes Fernsehen) oder Bildtelefon, das immer weiter entwickelt wird. Es müssen Kompressionsarten gefunden werden, die die Bilddateien drastisch verkleinern, damit sie schnell übertragen werden können; und wenn das Bild beim Empfänger „entpackt" wird, soll es möglichst originalgetreu rekonstruiert werden.

Kompression ist die Umwandlung von Daten in ein effizienteres Format [GG92]. Sie wird vorgenommen, um die Leitungskapazität zu minimieren, die die Daten bei ihrer Übertragung brauchen. Äquivalent dazu ist die Forderung, die beste Qualität bei einer gegebenen Bandbreite zu erreichen. Die Speicherung von Daten ist eigentlich nur eine andere Form der Übertragung. Hierbei wird eine Kompression vorgenommen, um den Speicherbedarf zu minimieren, den die Daten auf einem Speichermedium haben. Wiederum äquivalent dazu ist die Forderung, die bestmögliche Qualität bei gegebenem Speicherplatz zu erreichen. Manchmal wird Kompression auch vorgenommen, um die Komplexität nachfolgender Verarbeitungsalgorithmen wie Klassifikation, Transformation und Verschlüsselung zu vereinfachen. Wir beschäftigen uns hier mit der Übertragung von Bilddateien. Die Kompression erfolgt mit dem Ziel höchster Qualität unter den Bedingungen von niedriger Bitrate und geringer Implementierungskomplexität.

Für die genannten Anwendungen ist eine progressive Übertragung angebracht. Diese liefert zunächst ein Bild, das nur grob dem Original entspricht. Das eigentliche Bild ist darin verschwommen zu erkennen. Nach und nach werden dann zusätzliche Daten nachgeliefert und zum ersten Bild addiert, so daß das Gesamtbild immer klarer wird. Läßt man diese Übertragung bis zum Schluß durchlaufen, so erhält man am Ende die mehr oder weniger originalgetreue Rekonstruktion des gespeicherten Bildes. Wie gut die Rekonstruktion ist, hängt zum einen von der Kompressionsart und zum anderen von der Stabilität der Leitung ab. Die progressive Übertragung kann man jederzeit abbrechen. Hat man also das erste Bild empfangen, kann man schon ungefähr erkennen, ob es sich wirklich um dasjenige handelt, das man gesucht hat. Dies ist von großem Vorteil, wenn man zum Beispiel in einer Datenbank nach einem bestimmten Bild sucht. Handelt es sich nicht um das gewünschte Bild, so bricht man die Übertragung ab und sucht weiter. Auf diese Art und Weise geht die Suche nach einem Bild sehr schnell, denn für die erste grobe Annäherung eines Bildes braucht man nur sehr wenig Information. Das heißt diese wird kostengünstig und schnell übertragen.

Die Subband–Codierung ist ein Verfahren, das erst seit einigen Jahren erforscht wird [Pal93]. Sie hat Eigenschaften, die sie für die Kompression prädestiniert erscheinen lassen. Bei diesem Verfahren werden die Bilder mit Hilfe einer Subband–Analyse in verschiedene Frequenzbänder aufgespalten. Die sogenannten Koeffizienten aus diesen Frequenzbändern werden dann quantisiert; dies kann entweder skalar, also für jeden Koeffizienten einzeln, oder vektoriell erfolgen, indem man mehrere Koeffizienten zu einem Vektor zusammenfaßt. Hierzu werden Codebücher entworfen, mit denen man die Quantisierung vornimmt. Für die Übertragung der Daten werden dann nur noch die Codebücher und die Indizes der verwendeten Codevektoren gebraucht, denn zu jedem Vektor wird nur der Index seines Codevektors gespeichert. Bei der Dekompression des Bildes wird zu jedem Index der entsprechende Vektor aus dem Codebuch gesucht, aus den Vektoren wird das Koeffizientenbild erstellt, und es folgt abschließend eine Subband–Synthese. Das Ergebnis ist das rekonstruierte Bild.

In dieser Arbeit beschäftigen wir uns mit der Klasse der Gesichtsbilder. Wir wollen zeigen, daß ein Kompressionsverfahren effektiver gestaltet werden kann, wenn es auf die Klasse der Bilder, die damit komprimiert werden sollen, ausgerichtet ist. Zu diesem Zweck stehen uns 20 Gesichtsbilder in jeweils 256 Graustufen zur Verfügung. Sie haben alle einen einheitlich hellen Hintergrund, vor diesem ist jeweils ein Gesicht zu sehen. Für diese Bilder stellen wir eine Kompression zusammen, die in den Grundzügen immer wie folgt aussieht: Zunächst wird eine Wavelet–Transformation durchgeführt, die das Bild in Frequenzbänder aufspaltet. Diese Transformation ist dreistufig und liefert eine Oktav–Pyramide. Aus dem Koeffizientenbild dieser Pyramide erstellen wir Vektoren, die wir vektorquantisieren. Hieran anschließend würde normalerweise eine Codierung der Indizes der zu den Vektoren nächsten Codevektoren folgen, die wir aber in dieser Arbeit nicht durchgeführt haben, da sie für die Untersuchungen nicht relevant ist. Wir haben die Coderate berechnet, die das Bild nach dieser Codierung hätte, und damit können Vergleiche zu anderen Kompressionsarten angestellt werden. Bei uns folgen dann die Dequantisierung und eine abschließende Subband–Synthese. Dieser Ablauf ist die Kompression eines Bildes in einem Schritt, also die übliche Subband–Codierung. Abweichend hiervon führen wir eine hierarchische Kompression durch, das heißt, daß das Bild erst grob und dann der entstehende Fehler jeweils feiner und genauer quantisiert wird. Die hierzu inverse Dequantisierung liefert dann die oben angesprochene progressive Übertragung. Bei der Kompression können noch verschiedene Faktoren eingestellt werden, die wir zu gegebener Zeit erläutern.

Diese Arbeit ist folgendermaßen aufgebaut: Im nachfolgenden Kapitel werden wir einige grundlegende Dinge zur Bildkompression erklären. Wir zeigen, welche verschiedenen Kompressionsarten es gibt und an welchen Daten man „sparen“ kann. Begriffe wie *Redundanz* und *Kompressionsrate* werden erklärt, und wir geben Maße an, mit denen die Güte einer Rekonstruktion gemessen werden kann. Im anschließenden dritten Kapitel wird die Subband–Codierung näher erläutert. Dazu erklären wir, was Subband–Analyse ist und wie die entsprechenden Filter und die daraus aufgebauten Filterbänke aussehen. Dann erklären wir das Prinzip der Quantisierung erst anhand der Skalar– und dann — als Verallgemeinerung dieser — der Vektorquantisierung. Wir erläutern kurz, wie Codebücher entworfen werden, und mit welchen Mitteln man sie verbessern kann. Für weitergehende Untersuchungen sei dann auf die Literatur verwiesen. Das Kapitel schließt ab mit einigen Bemerkungen zur Codierung der Quantisierungsindizes. Kapitel 4 beinhaltet die Experimente. In diesen variieren wir die verschiedenen Parameter der Codierung und beurteilen jeweils die Auswirkungen. In Kapitel 5 geben wir einen kurzen Überblick über zwei andere progressive Codierungsarten und vergleichen diese mit unserer Codierung. Diskussion und Ausblick schließen die Arbeit ab.

Kapitel 2

Bildkompression

Durch das Aufkommen und die Verbesserung neuer technischer Anwendungen wie beispielsweise Bildtelefon und hochauflösendes Fernsehen (HDTV) wird der Ruf nach wirkungsvoller Datenkompression lauter. Bei den genannten Anwendungen müssen Bilder sehr schnell und kostengünstig übertragen werden. Beim Bildtelefon zum Beispiel reicht es auch, wenn zunächst ein verschwommenes Bild des „Gegenüber“ ankommt, das dann nach und nach (schnell) besser wird. Hierzu benötigt man eine progressive Übertragung, das heißt komprimierte Teilinformationen des Bildes werden nacheinander übertragen und beim Empfänger so zusammengesetzt, daß sich das Bild schrittweise verbessert.

Eine Datenkompression verringert die Menge der Daten, die man braucht, um eine bestimmte Information darzustellen. Hierbei muß man zwischen Daten und Information unterscheiden; erstere tragen letztere. Das Grundprinzip der Bildkompression ist die Umwandlung in eine andere Darstellungsform, so daß eine kompaktere Speicherung oder Übertragung möglich ist. Dabei sollen die Redundanz verringert und die irrelevanten Informationen entfernt werden. Unter Redundanz versteht man Information, die aufgrund von örtlichen oder zeitlichen Ähnlichkeiten im Bild mehrmals vorkommt. Irrelevant sind Informationsanteile, die zur Lösung einer Aufgabe — in unserem Fall zur Erkennung des Gesichts auf dem Bild — nicht erforderlich sind [Ohm95].

Es gibt zwei grundsätzliche Methoden der Bildkompression [Jai89]. Zum einen ist das die vorausgehende Codierung (*Predictive coding*). Diese nutzt die Redundanz in den Daten aus, die sich auf Vorhersagbarkeit, Unabhängigkeit/Zufälligkeit und Glattheit derselben bezieht. Zum Beispiel ist ein konstant graues Bild völlig vorhersagbar, weißes Rauschen hingegen gar nicht. Bei letzterem muß also jedes Pixel (Bildpunkt) gespeichert werden, und bei ersterem reicht es, den einen Wert zu kennen. Die zweite Methode ist die *Transformations–Codierung*, die wir auch in dieser Arbeit verwenden. Diese wandelt das Bild in ein anderes Format um, dabei wird die Information in weniger Daten (sogenannten *Samples*) zusammengefaßt. Man kann auch Kombinationen der beiden Methoden für die Bildkompression verwenden.

2.1 Redundanz

Um die Redundanz zweier Datenmengen, die dieselbe Information enthalten, herauszufinden, betrachtet man ihre Größen. Sind sie gleich groß, so ist keine Redundanz vorhanden. Ist die zweite kleiner, so enthält die erste redundante Daten [GW92].

Mathematisch läßt sich Redundanz wie folgt beschreiben: Seien n_1 und n_2 die Anzahlen von Informationsträgern zweier Datenmengen, die dieselbe Information beinhalten. Die relative *Datenredundanz* R_D der ersten Datenmenge ist

$$R_D = 1 - \frac{1}{C_R} \quad \text{mit} \quad C_R = \frac{n_1}{n_2}. \tag{2.1}$$

Dabei ist C_R die *Kompressionsrate.*

$n_2 = n_1 \quad \Longrightarrow \quad C_R = 1, \qquad R_D = 0$
Die erste Menge enthält (relativ zur zweiten) keine redundanten Daten.

$n_2 << n_1 \quad \Longrightarrow \quad C_R \longrightarrow \infty, \quad R_D \longrightarrow 1$
Es kann erheblich komprimiert werden, da die Daten sehr redundant sind.

$n_2 >> n_1 \quad \Longrightarrow \quad C_R \longrightarrow 0, \quad R_D \longrightarrow -\infty$
Die zweite Menge enthält viel mehr Daten als die erste; hier findet also gerade das Gegenteil zur gewünschten Kompression statt.

Zum Beispiel bedeutet eine Kompression von 10 (oder auch 10:1), daß die erste Menge 10 Informationseinheiten (zum Beispiel Bit) für eine Einheit in der zweiten Menge hat. Die entsprechende Redundanz von 0,9 heißt, daß 90% der Daten in der ersten Menge redundant sind.

Redundanz der Grauwerte

Zur Codierung eines Grauwertbildes kann man am einfachsten allen Grauwerten die gleiche Codewort–Länge zuordnen. Es gelte für die Grauwertwahrscheinlichkeit p_r

$$p_r(r_k) = \frac{n_k}{n}, \quad k = 0, \dots, L-1. \tag{2.2}$$

Dabei ist L die Anzahl der Grauwerte, n_k die Häufigkeit von Grauwert k im Bild, n die Anzahl aller Pixel im Bild und p_r die Wahrscheinlichkeit. Die durchschnittlich gebrauchte Anzahl von Bits pro Pixel ist dann

$$L_{\text{avg}} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k). \tag{2.3}$$

Dabei ist $l(r_k)$ die Anzahl der Bits pro Grauwert. Das bedeutet also, daß ein $(M \times N)$–Bild $M \cdot N \cdot L_{\text{avg}}$ Bit braucht. Wenn L_{avg} nicht reduziert wird, enthält das Bild Codierungsredundanz. Die Zahl kann reduziert werden, indem man die Größe von $l(r_k)$ für die einzelnen Grauwerte variiert. Dabei werden häufiger vorkommenden Grauwerten kürzere Codeworte zugeordnet und selteneren Grauwerten längere Codeworte. Diese Zuordnung ist eine sogenannte Entropiecodierung. Beispiele für Algorithmen zur Erzeugung der Codeworte sind die Huffman–Codierung und die arithmetische Codierung. Die Codierungsredundanz basiert auf der Entropie nullter Ordnung.

Interpixel–Redundanz

Unter dem Begriff Interpixel–Redundanz sind räumliche, geometrische und zeitliche Redundanz zusammengefaßt. Bei dieser Redundanz geht es um Entropien höherer Ordnung. Das heißt es wird nicht jedes Pixel für sich betrachtet, sondern die Zusammenhänge zwischen den Pixeln werden untersucht. Um Interpixel–Redundanz zu verringern, muß das Bild normalerweise in ein effizienteres, gewöhnlich „nicht visuelles“, Format transformiert werden. Statt eines statistischen Modells höherer Ordnung kann man dann nach der Transformation ein statistisches Modell nullter Ordnung anwenden, das sich also wieder nur auf die einzelnen Pixel konzentriert. Dies kann man machen, da durch die Transformation die Korrelation innerhalb der Pixel weitgehend entfernt wird.

2.2 Psychovisuelle Irrelevanz

Das Auge nimmt verschiedene Dinge unterschiedlich auf, dabei ist bestimmte Information weniger wichtig als andere für die Erfassung der Szene, das heißt sie ist psychovisuell irrelevant. Diese Information kann ohne große Beeinträchtigung der Wahrnehmungsqualität weggelassen werden.

Anders als bei den beiden Redundanzarten hat die Psychovisuelle Irrelevanz mit realer oder bestimmbarer visueller Information zu tun. Das Weglassen dieser ist nur möglich jenseits des Auflösungsvermögens des Menschlichen Visuellen Systems oder wenn sie für die konkrete Aufgabenstellung des Betrachters nicht von

Bedeutung ist. Läßt man Information weg, so ist diese Codierung zwangsläufig verlustbehaftet und damit irreversibel. Will man eine wirklich hohe Kompressionsrate erreichen, so muß man mit verlustbehafteter Codierung arbeiten.

2.3 Der Vorgang der Kompression

Das Bild wird mittels eines Encoders in eine kompaktere Form umgewandelt, und mit einem Decoder kann es wieder rekonstruiert werden. Es gibt zwei Arten der Kompression: die verlustfreie (*lossless*), bei der das Bild originalgetreu rekonstruiert werden kann, und die verlustbehaftete (*lossy*), bei der das nicht der Fall ist. Die in dieser Diplomarbeit behandelte Art der Kompression ist verlustbehaftet, denn bei der Quantisierung gehen Informationen verloren. Die Wavelet–Transformation erfolgt annähernd verlustfrei. Es kann hier also nur darum gehen, bei größtmöglichen Kompressionsraten eine bestmögliche Rekonstruktion des Bildes zu erhalten.

Ein Maß für die Qualität einer Approximation an das Original ist der sogenannte MSE, der *Mean Square Error*, also der mittlere quadratische Fehler. Dieser ergibt sich zu

$$\mathrm{MSE} = \frac{1}{M \cdot N} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} (x(m,n) - \widehat{x}(m,n))^2 \tag{2.4}$$

und ist ein objektives Maß. Hiermit wird einfach die Fehlerenergie der Rekonstruktion, bezogen auf das Original, gemessen. Dabei sind x und $\widehat{x}$ die Grauwerte der Pixel im Originalbild beziehungsweise in der Rekonstruktion, und M und N sind die Anzahlen der Pixel in der Breite beziehungsweise Höhe des Bildes.

Ein weiteres Maß für die Qualität einer Rekonstruktion ist der SNR (*Signal–to–Noise–Ratio, Signal–Rausch–Abstand*), der folgendermaßen definiert ist [GG92]:

$$\mathrm{SNR} \; = \; 10 \cdot \log_{10} \frac{\sum\limits_{m=0}^{M-1} \sum\limits_{n=0}^{N-1} x(m,n)^2}{\sum\limits_{m=0}^{M-1} \sum\limits_{n=0}^{N-1} (x(m,n) - \widehat{x}(m,n))^2} \;\; \mathrm{dB}\,. \tag{2.5}$$

Wiederum sind x und $\widehat{x}$ die Grauwerte der Pixel im Originalbild beziehungsweise in der Rekonstruktion, und M und N geben die Pixel in Breite und Höhe des Bildes an. Der SNR liefert eine Aussage über die Fehlerenergie bezogen auf die Signalenergie, er gibt also das Verhältnis der beiden zueinander an.

Der SNR ist ein allgemein vergleichbares Maß, das auch in der Literatur verwendet wird. Er sagt aus, wie gut das rekonstruierte Bild das Original approximiert;

je höher der Wert, desto besser die Approximation. Dieser Wert ist allerdings nicht unbedingt und alleinig zur Beurteilung geeignet, zusätzlich muß noch eine subjektive Bewertung stattfinden, da die Berechnung rein auf den Grauwerten erfolgt und nicht das subjektive Empfinden des Menschlichen Visuellen Systems mit seinen örtlichen Wechselwirkungen berücksichtigt. So kann ein Bild, bei dem die obere Hälfte perfekt und die untere schwarz ist, einen besseren SNR haben als ein Bild, über das gleichmäßig Rauschen verteilt ist. Dabei wird letzteres subjektiv natürlich besser beurteilt, da der Bildinhalt dort vollständig zu erkennen ist. In Abb. 2.1 wird dies deutlich. Allgemein korrelliert der SNR nur im oberen Wertebereich mit der subjektiven Bildqualität.

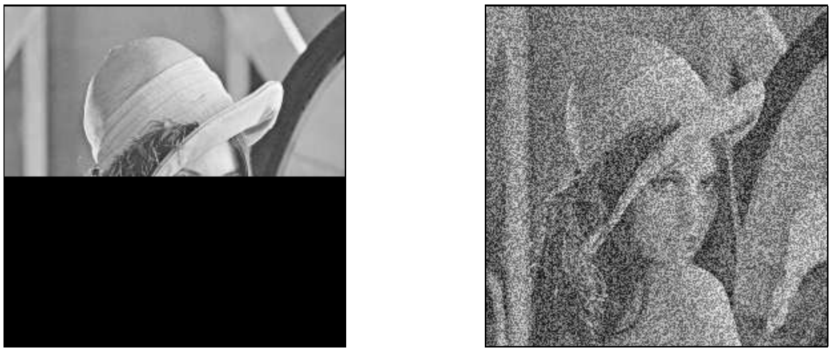


Abbildung 2.1: Das Bild „Lena" in zwei verschiedenen Qualitäten: links mit einem SNR von 3,27 dB, rechts mit 0,79 dB. (Der MSE–Wert ist beim linken Bild 9172,62 und beim rechten 13342,64.)

Um den Informationsgehalt einer Komponente eines Bildes (zum Beispiel eines Pixels) zu messen, betrachtet man das Bild als stochastischen Prozeß. Dann gilt für ein Ereignis η (zum Beispiel „das Pixel hat den Grauwert g"), das mit der Wahrscheinlichkeit $p(\eta)$ eintritt, daß es einen *Informationsgehalt* von

$$I(\eta) = \log \frac{1}{p(\eta)} = -\log p(\eta) \quad \text{Informationseinheiten} \tag{2.6}$$

hat [Ohm95]. Wird der Logarithmus zur Basis r genommen, so ist die Einheit r–adisch. Bei der Basis 2 heißt die Einheit Bit. Tritt ein Ereignis zum Beispiel mit der Wahrscheinlichkeit $p(\eta) = \frac{1}{2}$ auf, dann ist

$$I(\eta) = -\log_2 \frac{1}{2} = 1 \ \text{Bit}\,. \tag{2.7}$$

Der mittlere Informationsgehalt eines Bildes, die *Entropie* [Ohm95] wird berechnet durch

$$\begin{aligned} H &= -\sum_{i=0}^{N-1} p_i \, \log_2 p_i \ \text{ Bit} \\ &= E\{I(\eta)\}, \end{aligned} \tag{2.8}$$

wobei E der Erwartungswert ist. Er gibt den Mittelwert des Informationsgehaltes an, falls i eines von N möglichen Ereignissen ist. Eine größere Entropie steht für eine größere Unsicherheit, und das bedeutet mehr Information.

Dividiert man diese Entropie durch die Anzahl der Pixel im Bild, so erhält man die Coderate

$$r = \frac{H}{M \cdot N} \ \frac{\text{Bit}}{\text{Pixel}}. \tag{2.9}$$

Bei $256 = 2^8$ Graustufen in einem Bild kann jede Stufe mit 8 Bit dargestellt werden. Das heißt, daß ein solches Grauwertbild normalerweise eine Coderate von $r = 8 \ \frac{\text{Bit}}{\text{Pixel}}$ hat. Demnach gilt es, dies zu „unterbieten“.

Kapitel 3

Subband–Codierung

Bei der Subband–Codierung zerlegt man das Bild mittels der Diskreten Wavelet–Transformation in Subbänder. Die sich daraus ergebenden Wavelet–Koeffizienten werden skalar– oder vektorenquantisiert. Die Indizes der ausgewählten Vektoren werden anschließend encodiert und übertragen. Auf der Empfängerseite decodiert man die Indizes, ordnet die entsprechenden Codevektoren zu (Dequantisierung) und führt eine inverse Wavelet–Transformation durch (siehe Abb. 3.1).

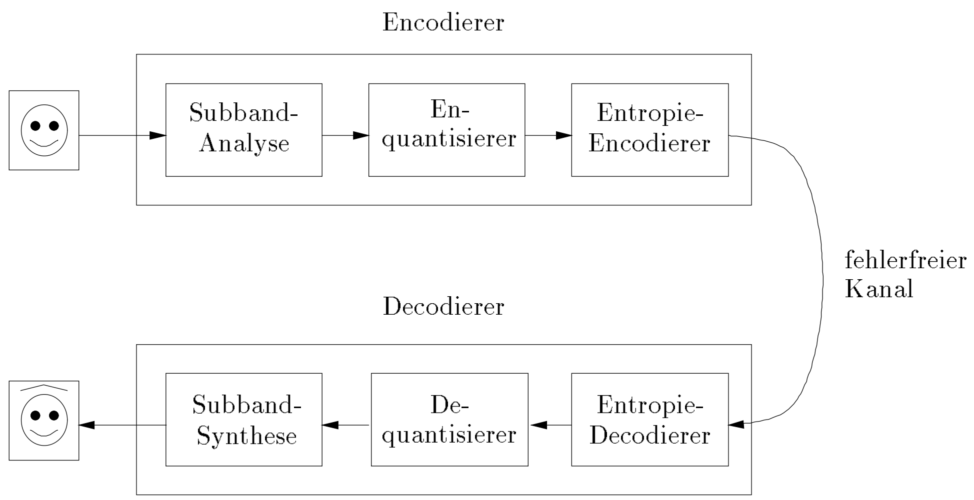


Abbildung 3.1: Schematischer Ablauf einer Subband–En– und Deodierung.

Wir sprechen hier ausdrücklich von En– und Decodierung beziehungsweise von En– und Dequantisierung, um deutlich zu machen, um welchen Vorgang es sich jeweils handelt und um diesen jeweils gegen die allgemeine „Codierung“ und „Quantisierung“ abzugrenzen. Mit Codierung bezeichnet man nämlich auch die gesamte Form der Kompression, und Quantisierung bezeichnet eigentlich die Hintereinanderausführung von En– und Dequantisierung.

3.1 Subband–Zerlegung

Mit einer Filterbank, die aus mehreren Filtern zusammengesetzt ist, kann man ein Signal in mehrere Frequenzbereiche auftrennen. Eine dazu inverse Filterbank setzt daraus das Ursprungssignal wieder zusammen. Führt man nur diese Analyse und Synthese nacheinander aus, also ohne dazwischenliegende Quantisierung oder andere Bearbeitungen, so erhält man ein originalgetreues Bild, sofern die Filterbank dies zuläßt, also perfekt rekonstruierend ist. In diesem Abschnitt erklären wir was Wavelets sind, wie sie als Filter arbeiten und wie aus diesen Filtern die Filterbänke zusammengesetzt sind.

3.1.1 Filter

Wavelets sind Funktionen, die bestimmten Bedingungen genügen. Ihren Namen haben sie von der Forderung, daß sie zu Null integrieren sollen und über und unter der x–Achse schwingen [VM91].

Wie Sinus und Cosinus in der Fouriertransformation werden Wavelets als Basisfunktionen für die Repräsentation von anderen Funktionen benutzt. Dazu wird ein Mother–Wavelet $\psi(x)$ festgelegt und eine Familie

$$\left\{ \psi \left(\frac{x-b}{a} \right) \middle| \; (a,b) \in \mathbb{R}^+ \times \mathbb{R} \right\} \tag{3.1}$$

von Translationen und Dilatationen des Motherwavelets erzeugt, die eine Basis darstellen. Für die diskrete Wavelet–Pyramide werden feste Werte für a und b gewählt: $a = 2^{-j}$ und $b = k \cdot 2^j$, mit $k, j \in \mathbb{N}_0$.

Was man mit „normalen“ Fourier–Methoden bei der Codierung erreichen kann, können Wavelets noch besser: Fourier–Basisfunktionen sind nur frequenz– und nicht zeit/ort–lokalisiert, das heißt kleine Änderungen in der Frequenz ziehen große Veränderungen überall im Zeit–Raum nach sich. Wavelets hingegen sind sowohl im Frequenz/Skalen–Raum (via Dilatationen) als auch im Zeit–Raum (via Translationen) lokalisiert.

Ein zweites Argument für die Wavelets ist, daß viele Klassen von Funktionen durch Wavelets kompakter repräsentiert werden können. Zum Beispiel brauchen Funktionen mit Diskontinuitäten und Funktionen mit scharfen Ecken weniger Wavelet–Basisfunktionen als Sinus/Cosinus–Basisfunktionen, um vergleichbare Approximationen zu erreichen.

Diese Eigenschaften machen die Wavelets zu einem exzellenten Werkzeug in der Datenkompression. Zum Beispiel benutzt das FBI Wavelets zur Kompression der digitalen Fingerabdruck–Bilder, und man kann verrrauschte Musikaufnahmen damit reinigen [VM91]. Große und verrauschte Datenmengen können mit Hilfe der Diskreten Wavelet–Transformation — dem Gegenstück zur Diskreten Fourier–Transformation — leicht und schnell transformiert werden. Die Daten werden durch die Wavelet–Koeffizienten codiert. Zusätzlich ist die Wavelet–Transformation auch schneller als die Fourier–Transformation. Die Fast Fourier–Transformation hat eine Zeitabschätzung von $\mathcal{O}(n \cdot log_2(n))$, die Fast Wavelet–Transformation hingegen läuft in $\mathcal{O}(n)$.

Unter den verschiedenen Arten von Wavelets ist das einfachste das *Haar*–Wavelet (siehe Abb. 3.2), das eine Treppenfunktion ist, die die Werte 1 auf dem Intervall $[0; 0,5)$ und -1 auf dem Intervall $[0,5; 1)$ annimmt.

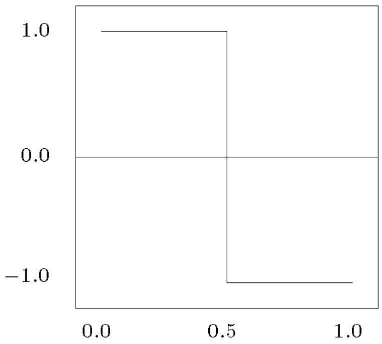


Abbildung 3.2: Haar–Wavelet.

Je nach Art oder Anzahl der Taps (so werden die Koeffizienten der Filter genannt) haben Wavelets unterschiedliche Eigenschaften. Also können bei verschiedenen Aufgabenstellungen ganz unterschiedliche Wavelets angebracht sein.

In [Pal93] werden verschiedene Arten von Filtern hinsichtlich ihrer Eigenschaften und Auswirkungen bei der Kompression untersucht, und aufgrund dieser ausführlichen Vergleiche verwenden wir in dieser Diplomarbeit Daubechies–Filter mit 8 Taps. Diese Filter haben wir mit Hilfe der entsprechenden Tools aus dem Khoros2–Paket erstellt.

3.1.2 Filterbänke

Mit einer Filterbank wird ein Signal in mehrere Frequenzbereiche aufgetrennt oder aus diesen zusammengesetzt [Pal93]. Eine *Analyse*–Filterbank besteht aus Filtern $H_0, \ldots, H_{M-1}$, die das Signal $x(n)$ in die Ausgangssignale $y_0, \ldots, y_{M-1}$

aufspalten. Die y_i stellen dabei die einzelnen Subbänder dar. An die Zerlegung schließt sich eine *Unterabtastung* (*Subsampling*) um den Faktor s_m an, das heißt der Ausgang des m–ten Filters wird nur an jedem s_m–ten Punkt ausgewertet. (Dies nennt man auch *Dezimation* [Ohm95].)

Entsprechend den Analyse–Filterbänken gibt es *Synthese*–Filterbänke, die aus Filtern $G_0, \ldots, G_{M-1}$ bestehen. Zuerst wird eine *Überabtastung* (*Upsampling*) der M Subbänder durchgeführt, das heißt zwischen den Eingangswerten werden jeweils $s_m - 1$ Nullen eingefügt. Dann werden diese Signale gefiltert und zu einem Ausgangssignal $\widehat{x}(n)$ aufaddiert. Ist die Synthese–Filterbank perfekt rekonstruierend, so gilt $x(n) = \widehat{x}(n)$.

Das Schema einer Filterbank ist in Abb. 3.3 dargestellt.

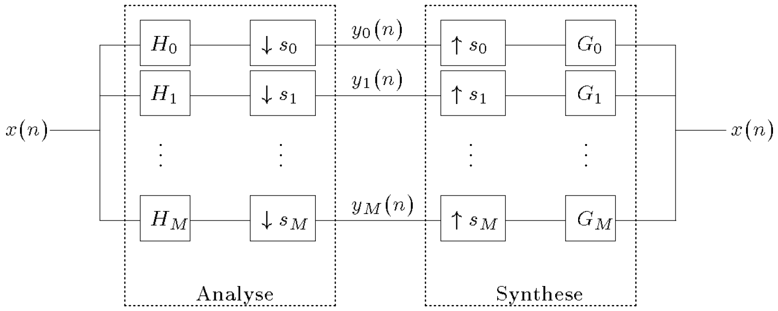


Abbildung 3.3: Schema einer Filterbank mit M Analyse– und M Synthese–Filtern und einem Abtastungsfaktor s_m für den m–ten Filter.

Gilt für eine Filterbank

$$\sum_{i=0}^{M-1} \frac{1}{s_i} = 1, \tag{3.2}$$

das heißt ist die Anzahl der Abtastwerte am Eingang gleich der Anzahl der Ausgabewerte an den Ausgängen (beziehungsweise die Anzahl der Werte an den Eingängen gleich der Anzahl der Werte am Ausgang), so heißt diese Filterbank *kritisch abgetastet*.

Für uns sind 2–Band–Filterbänke am interessantesten, da diese überschaubar in der Implementierung sind. In dieser Diplomarbeit werden nur Analyse/Synthese–Systeme (A/S–Systeme) behandelt, die aus solchen 2–Band–Filterbänken aufgebaut sind. Die Filter der Analyse–Filterbank werden mit H_0, H_1 bezeichnet, ihre Koeffizienten mit $h_0(k), h_1(k)$. Analog sind die Filter der Synthese–Filterbank G_0, G_1 und deren Koeffizienten $g_0(k), g_1(k)$ [Pal93].

In der Analyse werden die Koeffizienten folgendermaßen berechnet:

$$y_i(n) = \sum_k x(k)h_i(2n-k) \text{ für } i \in \{0,1\}. \tag{3.3}$$

Der Ausgang der Synthese–Filterbank ergibt sich zu

$$\widehat{x}(n) = \sum_k (y_0(k)g_0(2k-n) + y_1(k)g_1(2k-n)). \tag{3.4}$$

Dabei entspricht Gl. (3.3) einer Faltung mit h_i und anschließender zweifacher Unterabtastung, Gl. (3.4) entspricht einer zweifachen Überabtastung und anschließender Faltung mit g_i.

Bei den A/S–Systemen soll gelten, daß sie eine perfekte Rekonstruktion liefern, das heißt $x = \widehat{x}$. Setzen wir daher die eine Gleichung in die andere ein, so erhalten wir:

$$\sum_n (h_0(2n-k)g_0(2n-l) + h_1(2n-k)g_1(2n-l)) = \delta_{kl}. \tag{3.5}$$

Dabei bezeichnet δ_{kl} das Kronecker–Symbol, das heißt es ist

$$\delta_{kl} = \begin{cases} 1 & \text{für } k = l, \\ 0 & \text{sonst.} \end{cases} \tag{3.6}$$

Dieser Bedingung kann auf verschiedene Arten entsprochen werden, daher ergeben sich auch verschiedene Möglichkeiten zur Konstruktion von Filtern. Bei der Klasse der orthogonalen Filterpaare gilt $H_0 = G_0$, $H_1 = G_1$ mit der Orthogonalitätsbedingung

$$\sum_n h_0(n-2k)g_1(n-2l) = 0. \tag{3.7}$$

Aus den genannten Forderungen hat I. Daubechies die sogenannten Daubechies–Filter mit geradzahlig langen Impulsantworten entwickelt [Dau88]. In dieser Diplomarbeit verwenden wir Daubechies–Filter mit 8 Taps.

Kaskadierte Filterbänke

Man kann auch mehrere Filterbänke kaskadieren. So kann eines oder können mehrere der Ausgangssignale der einen Filterbank in der nächsten weiter gefiltert werden. Dabei kann man sowohl mehrere gleiche als auch verschiedene M–Band–Filterbänke hintereinanderschalten. Hier interessiert nur der Fall der *homogenen* Kaskadierung von 2–Band–Filterbänken. Eine einzelne 2–Band–Filterbank zerlegt ein Eingangssignal in einen tief– und einen hochfrequenten Anteil; mit der

Kaskadierung von Filterbänken können diese beiden weiter zerlegt werden. Dabei gibt es verschiedene Formen von Zerlegungen. Wird zum Beispiel immer nur der tieffrequente Anteil eines Signales weiter zerlegt, so erhält man die sogenannte *Pyramiden–Zerlegung* (siehe Abb. 3.4). Bei einer *vollen Zerlegung* (siehe Abb. 3.5) werden immer beide Ausgänge jeder Filterbank weiter zerlegt. Möglich ist auch die *gemischte Zerlegung* (Beispiel: siehe Abb. 3.6), bei der einige Subbänder noch weiter und andere nicht mehr zerlegt werden.

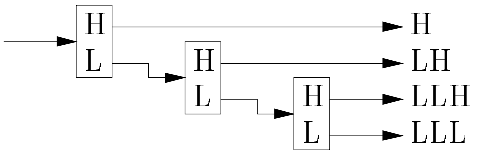


Abbildung 3.4: Pyramiden–Zerlegung.

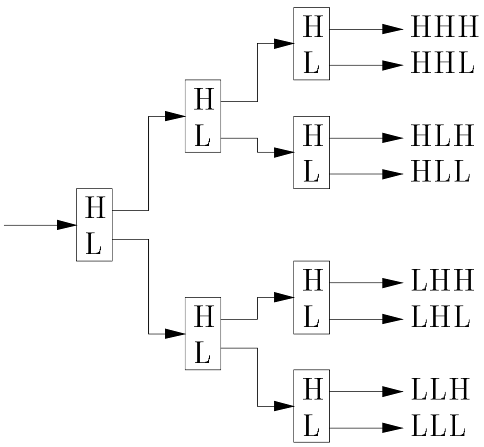


Abbildung 3.5: Vollständige Zerlegung.

Die Kaskadierung von 2–Band–Filterbänken hat einige Vorteile:

- Kompliziertere Subband–Aufteilungen lassen sich zurückführen auf eine mehrfache Anwendung der einfachen Auftrennung in zwei Bänder.
- Durch die mehrfache Anwendung der gleichen Filterbänke läßt sich das System einfacher implementieren.
- Bei zwei Bändern ist die Orthogonalitätsbedingung leichter erfüllbar, daher kann eine kritische Abtastung erfolgen.
- Die Optimierung der Filterbänke wird ebenfalls erleichtert.

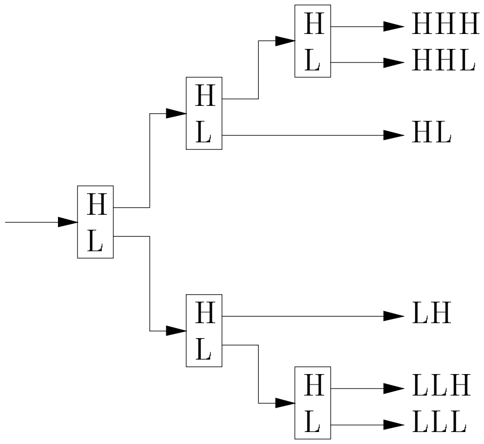


Abbildung 3.6: Gemischte Zerlegung.

- Die Wahl der geeigneten Filter kann unabhängig von der Wahl der geeigneten Zerlegung erfolgen, so daß jede für sich betrachtet werden kann.
- Die Erweiterung der Zerlegung von eindimensionalen auf mehrdimensionale Signale ist sehr einfach (siehe nächster Abschnitt).

Nachteilig bei der Kaskadierung von 2–Band–Filterbänken ist, daß die Zerlegung des Signals in beliebige Subbänder mit ihrer Hilfe nicht möglich ist, eine orientierungsselektive Zerlegung kann also nicht vorgenommen werden. Wegen der kritischen Abtastung verwenden wir in dieser Diplomarbeit die Kaskadierung von 2–Band–Filterbänken.

Zerlegung mehrdimensionaler Signale

Um ein Bild zu analysieren, kann man einfach eine separierbare Zerlegung auf einem rechteckigen Gitter nehmen, das heißt es werden zuerst die Zeilen und dann die Spalten (oder umgekehrt) getrennt zerlegt.

Mit einer eindimensionalen Filterbank, bestehend aus einem Hoch– und einem Tiefpaßfilter, wird das Bild zuerst horizontal transformiert und in dieser Richtung *unterabgetastet*, und anschließend werden die beiden so entstandenen Subbänder vertikal transformiert und in der Richtung unterabgetastet. Man erhält also insgesamt mit dieser 2D–Zerlegung vier Subbänder, die mit LL (zweimal tiefpaßgefiltert), HL (erst hoch–, dann tiefpaßgefiltert), LH (genau umgekehrt) und HH (zweimal hochpaßgefiltert) bezeichnet werden (siehe Abb. 3.7).

Bei der Oktav–Zerlegung, wie wir sie auch in dieser Arbeit durchführen, wird das LL–Band wiederholt auf die gleiche Weise weiter zerlegt (siehe Abb. 3.8).

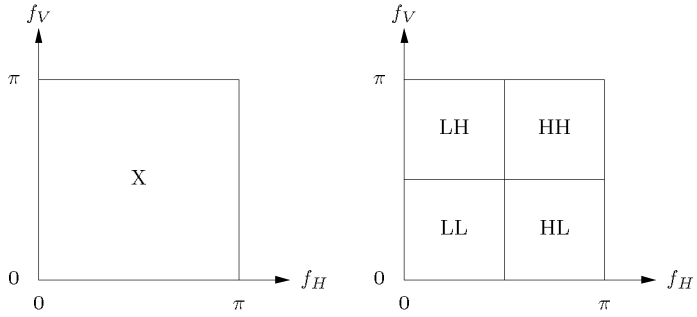


Abbildung 3.7: Frequenzaufteilung vor und nach einer 2D–Zerlegung.

Die Oktav–Zerlegung ist also eine dreifache Pyramiden–Zerlegung. Allgemein ist es üblich, das LL–Band oben links in der Ecke darzustellen, im Gegensatz zu der Darstellung der Frequenzaufteilung. Dies entspricht dem Koordinatensystem eines Bildschirmes, bei dem der Koordinatenursprung ebenfalls in die obere linke Ecke positioniert wird.

In [Pal93] werden verschiedene Zerlegungen verglichen und keine wesentlichen Unterschiede festgestellt. Da die Pyramiden–Zerlegung sehr effizient berechnet werden kann, verwenden wir diese.

Da wir mit mehrstufiger Zerlegung arbeiten, wollen wir die Bänder eindeutiger benennen, nämlich mit einem Tupel (Λ, Θ), wobei Λ der Anzahl der 2D–Zerlegungen entspricht, die das Band durchlaufen hat, und Θ gibt die Orientierung des Bandes an. Dabei soll gelten: $\Theta \in \{1, 2, 3, 4\} = \{LL, HL, LH, HH\}$.

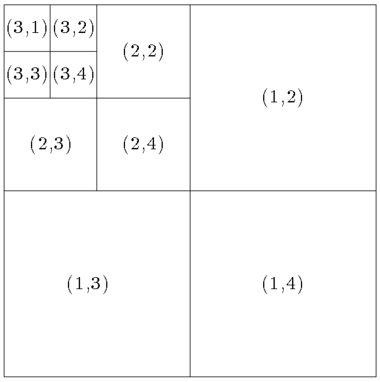


Abbildung 3.8: Oktav–Zerlegung. Das Paar (Λ, Θ) gibt die Stufe der Zerlegung und die Orientierung des Bandes an.

Am Beispiel des in der Bildbearbeitung oft verwendeten Bildes „Lena“ zeigt Abb. 3.9 die 2D–Zerlegung. Hierbei sieht man auf dem rechten Bild oben links

Abbildung 3.9: Das Bild „Lena“, links im Original und rechts nach einer einfachen 2D–Zerlegung.

das LL–Band, welches eine Verkleinerung des Originals darstellt. Rechts oben ist das HL–, links unten das LH– und rechts unten das HH–Band zu sehen.

Wie in Abb. 3.8 angedeutet, zeigt Abb. 3.10 das Bild nach einer dreistufigen Pyramiden–Zerlegung.

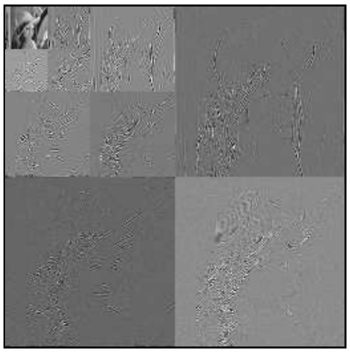


Abbildung 3.10: Das Bild „Lena“ nach einer dreistufigen Oktav–Zerlegung.

3.2 Quantisierung

Quantisierung ist eine Abbildung von einer Signalzustandsmenge $\mathcal{D}$ in eine Wertemenge $\mathcal{W}$ endlicher Kardinalität. Durch Quantisierung soll die Datenmenge, die zur Darstellung des Bildes benötigt wird, reduziert werden. Dies erreicht man dadurch, daß bei der Quantisierung die irrelevanten Datenanteile unterdrückt werden. Bei der Subband–Codierung kann man so die hochfrequenten Anteile des Bildes weniger bis gar nicht quantisieren, und die tieffrequenten Anteile werden bevorzugt, das heißt sie werden genauer quantisiert. Dies macht man, weil Quantisierungsfehler in den letzteren stärker wahrnehmbar sind.

In diesem Abschnitt gehen wir zunächst auf die Skalarquantisierung ein, und anschließend erläutern wir den allgemeineren Fall der Vektorquantisierung.

3.2.1 Skalarquantisierung

Definition: Ein N–stufiger *Skalarquantisierer* ist eine Abbildung $\mathcal{Q} : \mathbb{R} \longrightarrow \mathcal{C}$. Dabei ist $\mathcal{C} = \{y_1, \ldots, y_N\} \subset \mathbb{R}$ mit $|\mathcal{C}| = N$. Die Menge $\mathcal{C}$ heißt *Codebuch*, die y_i heißen *Repräsentativwerte* oder *Codeworte*. Das Codebuch ist von endlicher Größe, also $N < \infty$. Es wird $y_1 < \ldots < y_N$ angenommen [GG92].

Zu jedem N–stufigen Quantisierer gehört eine *Partitionierung* von $\mathbb{R}$ in N Quantisierungsintervalle oder *Zellen* Z_i mit $i = 1, \ldots, N$. Für die i–te Zelle gilt:

$$Z_i = \{x \in \mathbb{R} | \mathcal{Q}(x) = y_i\}.$$

Damit ist $Z_i = \mathcal{Q}^{-1}(y_i)$ also das inverse Bild von y_i unter $\mathcal{Q}$, das heißt $\mathcal{Q}$ ist eine surjektive Abbildung. Es ist

$$\bigcup_{i=1}^{N} Z_i = \mathbb{R} \ \text{ und } \ Z_i \cap Z_j = \emptyset \ \text{ für } \ i \neq j.$$

Definition: Ein Quantisierer ist *regulär*, falls

(a) jede Zelle Z_i ein Intervall der Form (x_{i-1}, x_i) zusammen mit einer oder beiden Intervallgrenzen und

(b) $y_i \in (x_{i-1}, x_i)$ ist.

Für eine vollständige Beschreibung wird bei regulären Quantisierern folgende Annahme getroffen: $Z_i = (x_{i-1}, x_i] = \{x | x_{i-1} < x \leq x_i\}$ für $i = 1, \ldots, N-1$. Dieselbe Definition gilt für $i = N$, falls x_N endlich ist; ist dies nicht der Fall, so gilt $Z_N = (x_{N-1}, x_N)$. Normalerweise ist $x_0 = -\infty$ und $x_N = +\infty$, die Zellen Z_0 und Z_N sind ein halboffenes und ein offenes Intervall. Die Menge der Intervallgrenzen spezifiziert vollständig und eindeutig den Quantisierer. Die Repräsentativwerte y_i der Zellen Z_i liegen jeweils in der Mitte der Zelle.

Ein Quantisierer ist gewöhnlich *monoton*, das heißt $\mathcal{Q}(x)$ ist eine monotone, nichtfallende Funktion von x. Offensichtlich ist ein regulärer Quantisierer monoton.

Ein besonderer Quantisierer ist der *symmetrische* Quantisierer: $\mathcal{Q}(x) = -\mathcal{Q}(-x)$. Der typische Graph der Funktion $\mathcal{Q}(x)$ für einen symmetrischen regulären Quantisierer zeigt einen Treppencharakter (siehe Abb. 3.11).

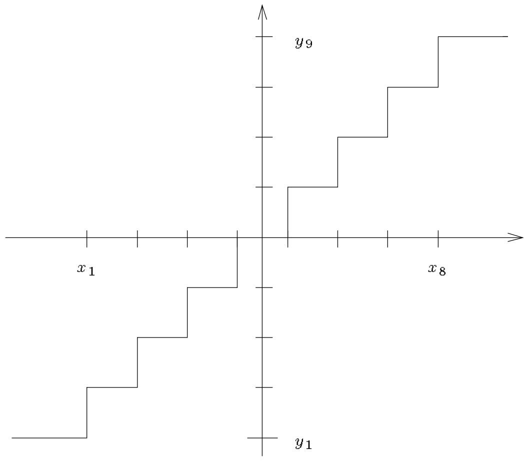


Abbildung 3.11: Treppencharakter eines symmetrischen regulären Quantisierers.

3.2.2 Vektorquantisierung

Will man mehrdimensionale Signale quantisieren, so kann man entweder eine separable Anwendung von Skalarquantisierern durchführen, oder man geht zur Vektorquantisierung über. Im ersten Fall gehen eventuelle Abhängigkeiten zwischen den Dimensionen verloren, bei der Vektorquantisierung hingegen können diese berücksichtigt werden. Daher soll nun analog zur Skalarquantisierung und als Verallgemeinerung dieser die Vektorquantisierung erläutert werden.

Definition: Ein N–stufiger *Vektorquantisierer* ist eine Abbildung $\mathcal{Q} : \mathbb{R}^k \longrightarrow \mathcal{C}$. Dabei ist $\mathcal{C} = \{\boldsymbol{y_1}, \ldots, \boldsymbol{y_N}\} \subset \mathbb{R}^k$ mit $|\mathcal{C}| = N$. Die Menge $\mathcal{C}$ heißt *Codebuch* oder *Code*, die $\boldsymbol{y}_i$ heißen *Repräsentativ–* oder *Codevektoren*. Das Codebuch ist von endlicher Größe, also $N < \infty$ [GG92].

Es ist $\boldsymbol{y}_i \neq \boldsymbol{y}_j$ für $i \neq j$. Die maximal mögliche Genauigkeit eines Vektorquantisierers wird durch

$$r_{\text{exakt}} = \frac{\log_2 N}{k} \tag{3.8}$$

beschrieben. Sie gibt die Anzahl der Bits pro Vektorkomponente an, die zur Repräsentation eines Eingabevektors zur Verfügung stehen.

Zu jedem N–stufigen Vektorquantisierer wird eine Partitionierung von $\mathbb{R}^k$ in N Regionen oder Zellen Z_i mit $i \in \mathcal{J}$ assoziiert. Dabei gilt für die i–te Zelle:

$$Z_i = \{\boldsymbol{x} \in \mathbb{R}^k | \mathcal{Q}(\boldsymbol{x}) = \boldsymbol{y}_i\}, \tag{3.9}$$

sie wird das *Urbild* von $\boldsymbol{y}_i$ genannt, das heißt $Z_i = \mathcal{Q}^{-1}(\boldsymbol{y}_i)$. Es ist

$$\bigcup_{i=1}^{N} Z_i = \mathbb{R}^k \text{ und } Z_i \cap Z_j = \emptyset \text{ für } i \neq j. \tag{3.10}$$

Eine wichtige Eigenschaft einer Menge in $\mathbb{R}^k$ ist die Konvexität. Damit wird analog zur Skalarquantisierung ein regulärer Quantisierer definiert.

Definition: Ein Vektorquantisierer ist *regulär*, falls

(a) jede Zelle Z_i eine konvexe Menge und

(b) $\boldsymbol{y}_i \in Z_i$ für jedes i ist.

Einen Vektorquantisierer kann man auch als zwei aufeinanderfolgende Operationen auffassen, nämlich als einen *Encoder* $\mathcal{E} : \mathbb{R}^k \longrightarrow \mathcal{J}$ und einen *Decoder* $\mathcal{D} : \mathcal{J} \longrightarrow \mathcal{C}(\subset \mathbb{R}^k)$. Der Encoder entscheidet, welcher Codevektor dem zu codierenden am nächsten liegt und liefert den entsprechenden Index. Diese Entscheidung wird mit Hilfe eines Abstandsmaßes getroffen, in diesem Fall mit Hilfe des Euklidischen Abstands, der folgendermaßen definiert ist:

$$d(\boldsymbol{x}, \widehat{\boldsymbol{x}}) = \sum_{i=1}^{k} (x_i - \widehat{x}_i)^2. \tag{3.11}$$

Mit diesem Abstandsmaß werden auch die Zellen der Partitionierung bestimmt:

$$Z_i := \{\boldsymbol{x} | d(\boldsymbol{x}, \boldsymbol{y}_i) \leq d(\boldsymbol{x}, \boldsymbol{y}_j) \text{ für alle } i \neq j\}. \tag{3.12}$$

Diese Zellen heißen *Voronoi*–Zellen. Die Bedingung, nach der sie gefunden werden, heißt auch Bedingung des nächsten Nachbarn oder *Nearest Neighbor*–Bedingung.

Die direkteste Encodierung für einen Voronoi–Encoder ist die folgende:

$$\mathcal{E} : \mathbb{R}^k \longrightarrow \mathcal{J} \text{ mit } \mathcal{E}(\boldsymbol{x}) = i = \operatorname*{argmin}_{j \in \mathcal{J}} d(\boldsymbol{x}, \boldsymbol{y}_j). \tag{3.13}$$

Das heißt also, daß der Index desjenigen Codevektors ausgewählt wird, für den der Abstand zum Vektor am kleinsten ist.

Der Hauptvorteil dieser Encodierung liegt darin, daß keine geometrische Beschreibung der Partitionierung gebraucht wird, um die Encodierungsregel durchzuführen. Das heißt die Encodierungsoperation kann beschrieben werden durch

$$\mathcal{E}(\boldsymbol{x}) = c(\boldsymbol{x}, \mathcal{C}), \tag{3.14}$$

wobei die funktionale Operation $c(\cdot,\cdot)$ unabhängig ist von dem spezifischen Quantisierer und nur vom Fehlermaß abhängt. Die Funktion $c(\cdot,\cdot)$ liefert zu einem Vektor den Index des ihm nächsten Codevektors aus einem gegebenen Codebuch.

Der Decoder liefert mit Hilfe einer LUT (*Look up table*) den dazugehörigen Codevektor. Für die Vektorquantisierung gilt also $\mathcal{Q}(\boldsymbol{x}) = (\mathcal{D} \circ \mathcal{E})(\boldsymbol{x}) = \mathcal{D}(\mathcal{E}(\boldsymbol{x}))$. Der Quantisierer kann auch den Index und den Ausgabewert $\mathcal{Q}(\boldsymbol{x})$ generieren (siehe Abb. 3.12).

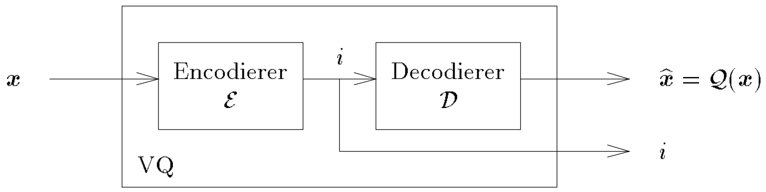


Abbildung 3.12: Schema der Quantisierung [GG92].

Für die Übertragung des komprimierten Bildes braucht man nur den Index. Dieser wird codiert und übertragen. Auf der Empfängerseite wird er entsprechend decodiert, und dann wird aus dem dort ebenfalls vorliegenden Codebuch der dazugehörige Codevektor herausgesucht. Für diese Diplomarbeit allerdings interessieren uns die Indizes nicht, sie sind nur für die Berechnung der Coderate von Bedeutung. Vielmehr lassen wir gleich die Codevektoren ausgeben, um uns das so quasi schon rekonstruierte Bild anzusehen.

3.2.3 Codebuch–Entwurf

Für den Codebuch–Entwurf verwenden wir den *Lloyd–Algorithmus* [GG92], auch als *LBG*–Algorithmus bekannt (*L*inde, *B*uzo, *G*ray). Für eine Trainingsmenge $\mathcal{T}$ von Vektoren und ein Codebuch $\mathcal{C}_m$ berechnet dieser die Gesamtabweichung, das heißt für alle Trainingsvektoren wird die Abweichung von ihrem nächsten Nachbarn addiert, und die Summe wird durch die Anzahl der Trainingsvektoren dividiert:

$$D_m = \frac{\sum\limits_{x \in \mathcal{T}} d(\boldsymbol{x}_i, \mathcal{Q}(\boldsymbol{x}_i))}{|\mathcal{T}|}. \tag{3.15}$$

Die Summe geht über alle Vektoren $\boldsymbol{x} \in \mathcal{T}$. Ist diese Gesamtabweichung klein genug gegenüber der vorherigen, so bricht der Algorithmus ab, und das Codebuch $\mathcal{C}_m$ ist das für diesen Grenzwert optimale. Für einen gegebenen Grenzwert thrs lautet das Abbruchkriterium somit: Ist

$$\frac{D_m - D_{m+1}}{D_m} \leq \text{thrs}, \tag{3.16}$$

so ist der gewünschte Optimierungsgrad erreicht. (Der Lloyd–Algorithmus konvergiert, somit bräuchte man diesen Grenzwert nicht. Hat man es allerdings mit einem sehr großen Codebuch zu tun, so kann es nützlich sein, die Optimierung nur bis zu einem bestimmten Grad durchlaufen zu lassen.) Das Optimierungsziel des Lloyd–Algorithmus' ist also, für die gewünschte Anzahl von Codevektoren ein Codebuch zu entwerfen, in dem die Codevektoren im Sinne des Fehlermaßes die Trainingsmenge möglichst genau appoximieren.

Das Flußdiagramm in Abb. 3.13 soll den Ablauf des Lloyd–Algorithmus' schematisch veranschaulichen.

Innerhalb des Lloyd–Algorithmus' findet die Lloyd–Iteration (siehe Abb. 3.14) statt. In dieser wird für ein Codebuch $\mathcal{C}_m$ die Trainingsmenge $\mathcal{T}$ in Teilmengen partitioniert. Dies geschieht dadurch, daß jeder Trainingsvektor entsprechend der Bedingung des nächsten Nachbarn seinem nächsten Codevektor zugeordnet wird. So entstehen zu jedem Codevektor $\boldsymbol{y}_i$ Partitionierungsmengen oder Zellen Z_i, in denen sich alle Vektoren aus der Trainingsmenge befinden, die zu diesem Codevektor gehören:

$$Z_i = \{\boldsymbol{x} \in \mathcal{T} | d(\boldsymbol{x}, \boldsymbol{y}_i) \leq d(\boldsymbol{x}, \boldsymbol{y}_j) \text{ für alle } i \neq j\}. \tag{3.17}$$

Für jede dieser Zellen wird ihr Mittel– oder Schwerpunkt berechnet, und diese Schwerpunkte bilden das neue Codebuch $\mathcal{C}_{m+1}$:

$$\mathcal{C}_{m+1} := \{\text{cent}(Z_i)\}. \tag{3.18}$$

Die Schwerpunkte werden dabei folgendermaßen berechnet:

$$\text{cent}(Z_i) = \frac{\sum\limits_{\boldsymbol{x} \in Z_i} \boldsymbol{x}}{|Z_i|}. \tag{3.19}$$

Wenn bei der Partitionierung eine leere Zelle entsteht, so wird anstelle des entsprechenden Codevektors ein neuer eingefügt. Welcher Vektor eingefügt werden soll, kann von Fall zu Fall festgelegt werden. In dieser Arbeit fügen wir denjenigen ein, der zwischen dem zu löschenden Vektor und dem Vektor liegt, der die meisten Trainingsvektoren in seiner Zelle hat. In [GG92] sind auch andere Möglichkeiten zur Bestimmung eines neues Vektors aufgezeigt.

Da der Lloyd–Algorithmus ein gegebenes Codebuch nur iterativ verbessert, muß also ein erstes Codebuch vorliegen. Für die Erstellung dieses ersten Codebuchs gibt es verschiedene Möglichkeiten. Es wurden auch mehrere Algorithmen entworfen, so daß das erste Codebuch schon möglichst optimal gestaltet werden kann.

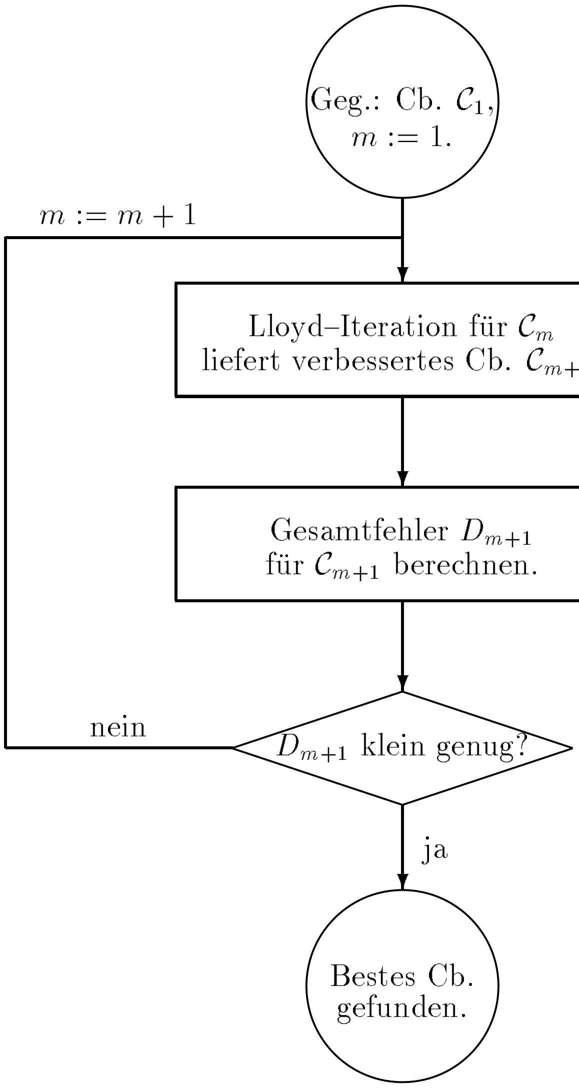


Abbildung 3.13: Das Flußdiagramm des Lloyd–Algorithmus'.

Für diese Theorien sei auf die Literatur verwiesen, zum Beispiel [GG92]. In dieser Arbeit erhalten wir das erste Codebuch einfach, indem wir die ersten Vektoren aus der Trainingsmenge einfügen, bis die gewünschte Anzahl von Codevektoren vorhanden ist:

$$\mathcal{C}_1 := \{\boldsymbol{x}_i \,|\, 1 \leq i \leq N,\ \boldsymbol{x}_i \in \mathcal{T}\}, \tag{3.20}$$

wobei N die Anzahl der Codevektoren ist. Sollte $N > |\mathcal{T}|$ angegeben worden sein, so wird $N := |\mathcal{T}|$ gesetzt.

3.2.4 Messung der Quantisierungsleistung

Hat man ein Codebuch generiert, so kann man dessen Leistung messen. Die dazu gebräuchlichen Fehlermaße stellen wir hier vor.

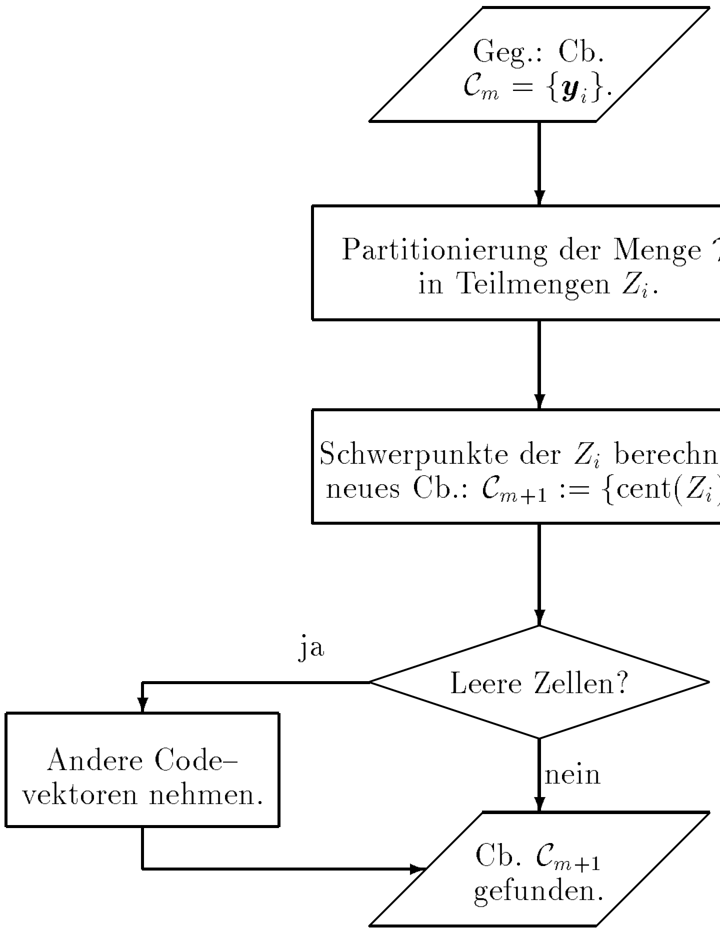


Abbildung 3.14: Das Flußdiagramm der Lloyd–Iteration.

Ein Fehlermaß d ist eine Zuweisung von nichtnegativen Kosten $d(\boldsymbol{x}, \widehat{\boldsymbol{x}})$, die durch das Quantisieren eines Eingabevektors $\boldsymbol{x}$ mit dem Repräsentativvektor $\widehat{\boldsymbol{x}}$ entstehen. Wenn ein solches Maß gegeben ist, kann man die Leistung eines Systems mit einem mittleren Fehler $D = E\{d(\boldsymbol{x}, \widehat{\boldsymbol{x}})\}$ zwischen Ein– und Ausgabe quantifizieren. Wenn die Reproduktion im Vordergrund steht, ist die Leistung eines Kompressionssystems gut, wenn der mittlere Fehler klein ist.

Sei $\boldsymbol{X}$ ein Zufallsvektor in $\mathbb{R}^k$ mit einer gegebenen Wahrscheinlichkeitsdichtefunktion $f_{\boldsymbol{X}_i}(\boldsymbol{x}_i)$ der Vektorkomponenten $\boldsymbol{X}_i$ für $i = 1, \ldots, k$. Dann kann der mittlere Fehler ausgedrückt werden durch

$$D = E\{d(\boldsymbol{x}, \widehat{\boldsymbol{x}})\} = \int\limits_{\mathbb{R}^k} d(\boldsymbol{x}, \mathcal{Q}(\boldsymbol{x})) f_{\boldsymbol{X}}(\boldsymbol{x}) d\boldsymbol{x}. \tag{3.21}$$

Durch Ausnutzen der Partitionierung und des Codebuchs $\mathcal{C} = \{\boldsymbol{y}_1, \ldots, \boldsymbol{y}_N\}$ für den gegebenen Quantisierer $\mathcal{Q}$ erhält man

$$D = \sum_{j=1}^{N} \int_{Z_j} d(\boldsymbol{x}, \boldsymbol{y}_j) f_{\boldsymbol{X}}(\boldsymbol{x}) d\boldsymbol{x}, \tag{3.22}$$

oder

$$\begin{aligned} D &= \int \sum_{j=1}^{N} P_j d(\boldsymbol{x}, \boldsymbol{y}_j) f_{\boldsymbol{X}|j}(\boldsymbol{x}|j) d\boldsymbol{x} \\ &= \sum_{j=1}^{N} P_j E\{d(\boldsymbol{x}, \boldsymbol{y}_j) | \boldsymbol{x} \in Z_j\}, \end{aligned} \tag{3.23}$$

wobei $P_j = P(\boldsymbol{x} \in R_j)$ ist und $f_{\boldsymbol{X}|j}(\boldsymbol{x}|j)$ die bedingte Wahrscheinlichkeitsdichte von $\boldsymbol{x} \in Z_j$.

Gelegentlich ist es besser, den *Worst–case*–Fehler als Maß für die Leistung zu benutzen statt eines Mittelmaßes. Dieser ist einfach definiert als maximal erreichbarer Fehler für einen gegebenen Quantisierer:

$$D_{max} = \max_{\boldsymbol{x} \in B} d(\boldsymbol{x}, \mathcal{Q}(\boldsymbol{x})), \tag{3.24}$$

wobei B die abgeschlossene Teilmenge von $\mathbb{R}^k$ ist, in die die Eingabe $\boldsymbol{x}$ gehört, das heißt B ist der Bereich, in dem die Wahrscheinlichkeitsdichtefunktion nicht Null ist. In einigen Fällen kann es sein, daß dieses Maximum nicht existiert, weil der Fehler beliebig groß werden kann und dieses Maß dann bedeutungslos ist. Der Gebrauch des maximalen Fehlers als Leistungsmaß ist beschränkt auf den Fall von begrenzten Zufallsvektoren oder Quantisierern mit abzählbar unendlicher Codebuchgröße ohne unbegrenzte Regionen.

Am häufigsten wird als Fehlermaß der *quadratische Fehler* (oder die *quadratische Euklidische Distanz*) zwischen zwei Vektoren genommen. Dieser ist folgendermaßen definiert:

$$\begin{aligned} d(\boldsymbol{x}, \widehat{\boldsymbol{x}}) &= ||\boldsymbol{x} - \widehat{\boldsymbol{x}}||_2^2 \\ &= (\boldsymbol{x} - \widehat{\boldsymbol{x}})^T (\boldsymbol{x} - \widehat{\boldsymbol{x}}) \\ &= \sum_{i=1}^{k} (x_i - \widehat{x}_i)^2. \end{aligned} \tag{3.25}$$

Damit ergibt sich der mittlere Fehler als

$$D = E\{d(\boldsymbol{x}, \widehat{\boldsymbol{x}})\} = E\{||\boldsymbol{x} - \widehat{\boldsymbol{x}}||^2\}. \tag{3.26}$$

Viele der für die Vektorquantisierung interessanten Maße haben die Form

$$d(\boldsymbol{x}, \widehat{\boldsymbol{x}}) = \sum_{i=1}^{k} d_m(x_i, \widehat{x}_i), \tag{3.27}$$

wobei $d_m(x, \widehat{x})$ ein skalares Fehlermaß ist. Insbesondere ist d_m oft gegeben durch $d_m(x, \widehat{x}) = |x - \widehat{x}|^m$ für $m \in \mathbb{N}$. Ist $m = 1$, so hat man die l_1–Norm, bei $m = 2$ das oben genannte Fehlermaß. Die m–te Wurzel aus d_m liefert die l_m–Norm.

Ein weiteres Fehlermaß ist der *gewichtete quadratische Fehler*

$$d(\boldsymbol{x}, \widehat{\boldsymbol{x}}) = (\boldsymbol{x} - \widehat{\boldsymbol{x}})^T \boldsymbol{W} (\boldsymbol{x} - \widehat{\boldsymbol{x}}), \tag{3.28}$$

wobei $\boldsymbol{W}$ eine symmetrische und positiv definite Wichtungsmatrix ist und die Vektoren $\boldsymbol{x}$ und $\widehat{\boldsymbol{x}}$ als Spaltenvektoren betrachtet werden. Im Fall $\boldsymbol{W} = \boldsymbol{I}$, wenn $\boldsymbol{W}$ also die Identitätsmatrix ist, entspricht die Darstellung dem normalen quadratischen Fehler. Ist $\boldsymbol{W}$ eine Diagonalmatrix mit Diagonalwerten $w_{ii} > 0$, hat man

$$d(\boldsymbol{x}, \widehat{\boldsymbol{x}}) = \sum_{i=1}^{k} w_{ii}(x_i - \widehat{x}_i)^2. \tag{3.29}$$

Mit dem gewichteten Fehlermaß können bestimmte Komponenten stärker hervorgehoben werden, das heißt diese Dimensionen fallen bei der Erstellung des Codebuchs für die Quantisierung stärker ins Gewicht.

3.2.5 Wahrnehmung von Quantisierungsfehlern

Gesucht wird ein Maß, mit dem man die Wahrnehmbarkeit von Quantisierungsfehlern beziffern kann. In [WYSV97] werden dazu Untersuchungen angestellt.

Um die Stimuli auf den Betrachter beziehen zu können, wird dort zunächst eine Formel zur Berechnung der DVR, der *Display visual resolution* in $\frac{\text{Pixel}}{\text{deg}}$ angegeben:

$$\text{DVR} \; = \; d \cdot v \cdot \tan(\frac{\pi}{180}) \; \approx \; d \cdot \frac{v\pi}{180} \; \approx \; d \cdot \frac{v}{57,3} \quad \frac{\text{Pixel}}{\text{deg}} \tag{3.30}$$

Dabei sind v der Abstand vom Bildschirm (in cm) und d die Display resolution, also die Bildschirmauflösung (in $\frac{\text{Pixel}}{\text{cm}}$).

Unser Bildschirm hat eine Auflösung von ca. $81,28 \times 84,67 \; \frac{\text{Pixel}}{\text{inch}}$, also ungefähr $32 \; \times \; 33,3 \; \frac{\text{Pixel}}{\text{cm}}$. Die Entfernung zum Monitor beträgt in etwa 70 cm. Daher ergibt sich mit $d \approx 32 \; \frac{\text{Pixel}}{\text{cm}}$ und $v \approx 70$ cm

$$\text{DVR} \; = \; d\frac{v}{57,3} \; \approx \; 39,09 \; \frac{\text{Pixel}}{\text{deg}}. \tag{3.31}$$

Desweiteren wird die Ortsfrequenz f von Level Λ wie folgt berechnet:

$$f = r \cdot 2^{-\Lambda} \; \frac{\text{cyc}}{\text{deg}}. \tag{3.32}$$

Für den Artikel wurden Versuchsreihen mit Testpersonen durchgeführt. Diese sollten bei bestimmten Bildern jeweils in einer *Two–alternative forced–choice (2AFC)*–Prozedur angeben, ob sie ein Artefakt im Bild sahen oder nicht. Die Artefakte wurden dabei von Basisfunktions–Stimuli beziehungsweise von DWT–Rausch–Bildern hervorgerufen, die auf ein Bild addiert wurden.

Anhand dieser Testreihen wurden mehrere Modelle ausgearbeitet, mit denen die Grenzwerte für die Graustufen als Funktion von Ortsfrequenz und Orientierung berechnet werden können. Ein angemessen erscheinendes ist das folgende:

$$\log Y = \log a + k(\log f - \log g_\Theta f_0)^2. \tag{3.33}$$

Dies ist eine Parabel in $\log Y$– gegen $\log f$–Koordinaten mit einem Minimum bei $g_\Theta f_0$ und der Breite k^{-2}. Der Term g_Θ verschiebt das Minimum um einen von der Orientierung abhängigen Betrag. Der Wert a definiert den Minimal–Threshold.

Die in Gl. (3.33) enthaltenen Parameter haben die folgenden Werte:

$$\begin{aligned} a &= 0,495 \\ k &= 0,466 \\ f_0 &= 0,401 \\ g_1 &= 1,501, \quad g_2 = g_3 = 1,0, \quad g_4 = 0,534 \end{aligned}$$

Sie wurden hinsichtlich des summierten quadratischen Fehlers in $\log Y$ optimiert.

Die mit Gl. (3.33) berechneten Grauwerte geben die obere Grenze an, bis zu der laut den Testreihen keine Artefakte in den Bildern festgestellt werden konnten. Sie geben also die *Empfindlichkeit* an: kleine Werte stehen für eine hohe Empfindlichkeit (was bei niedrigen Frequenzen der Fall ist), und hohe Werte stehen für eine geringe Empfindlichkeit (bei hohen Frequenzen) gegenüber Quantisierungsfehlern.

3.2.6 Vektorform

Die Koeffizienten des wavelet–transformierten Bildes kann man auf verschiedene Arten zu Vektoren zusammenfassen. Diese sollen nun verglichen werden.

Man kann zum Beispiel jeden Koeffizienten einzeln quantisieren. Damit hätte man eindimensionale Vektoren und könnte diese mit Hilfe der Skalarquantisierung codieren. Man kann auch mehrere Koeffizienten aus verschiedenen Subbändern in einem Vektor zusammenfassen, zum Beispiel aus allen Bändern einer Stufe oder nach Orientierungen geordnet über mehrere Stufen hinweg. Oder man faßt alle Koeffizienten aus allen Bändern in einem Vektor zusammen, so wie wir das

$4 \cdot 100 = 400$ CV
0,367 Bit/Pixel
23,87 dB

$4 \cdot 50 = 200$ CV
0,293 Bit/Pixel
23,32 dB

$3 \cdot 20 + 50 = 110$ CV
0,227 Bit/Pixel
22,36 dB

Abbildung 3.15: Vektoren über die Level gebildet (also jeweils die Orientierungen HL, HH und LH zusammen, LL einzeln). Dabei bedeutet zum Beispiel $4 \cdot 100$, daß für jede Orientierung ein Codebuch mit 100 Codevektoren entworfen wurde. Entsprechend gilt bei $3 \cdot 20 + 50$, daß das Codebuch für jede Orientierung 20 und für das LL–Band 50 Codevektoren enthält.

in dieser Arbeit getan haben. Damit steigt zwar die Komplexität des Codebuch–Entwurfs, aber statistischen Abhängigkeiten zwischen Orientierungen und Skalen kann so besser Rechnung getragen werden.

Wie man sieht, braucht man bei den Bildern, die mit 64D-Vektoren codiert wurden, weniger Codevektoren und erhält damit bei geringerer Coderate einen höheren SNR und auch subjektiv ein besseres Bild. Auch können so restliche Abhängigkeiten zwischen den Grauwerten miterfaßt werden. Daher verwenden wir im folgenden immer 64D-Vektoren, die also aus allen Bändern gebildet werden. Somit müssen wir immer nur für eine Vektormenge das Codebuch trainieren und nicht für mehrere gleichzeitig.

Die Vektoren werden dabei folgendermaßen gebildet (siehe Abb. 3.18): Aus den vier Bändern der höchsten Ebene wird je ein Pixel genommen, aus den drei Bändern der mittleren Ebene werden die dem nächsthöheren Band entsprechenden vier Pixel genommen, und aus der ersten Ebene werden die wiederum entsprechenden sechzehn Pixel verwendet.

3.2.7 Bedeutung von Totzone und Gewichtung

Wir haben es hier mit Vektoren zu tun, die in einem 64D-Raum liegen. Um den Koordinatenursprung herum werden wir eine sogenannte *Totzone* (*Deadzone*) definieren. Alle Vektoren, die in dieser liegen, werden auf den Nullvektor abgebildet, beziehungsweise mit diesem quantisiert. Den Begriff „in der Totzone

		
$4 \cdot 100 = 400$ CV	$4 \cdot 50 = 200$ CV	$3 \cdot 20 + 50 = 110$ CV
0,362 Bit/Pixel	0,288 Bit/Pixel	0,217 Bit/Pixel
24,45 dB	23,58 dB	22,30 dB

Abbildung 3.16: Vektoren innerhalb der Level gebildet (jedes Level, LL einzeln). Dabei bedeutet wiederum $4 \cdot 100$, daß für jedes Level ein Codebuch mit 100 Codevektoren entworfen wurde. Entsprechend gilt bei $3 \cdot 20 + 50$, daß das Codebuch für jedes Level 20 und für das LL–Band 50 Codevektoren enthält.

liegen“ definieren wir hierbei über die Länge der Vektoren. Sie wird für einen Vektor $\boldsymbol{x} \in \mathbb{R}^k$ bestimmt durch

$$|\boldsymbol{x}| = \sqrt{\sum_{i=1}^{k} x_i^2}. \tag{3.34}$$

Die Totzone bricht mit der Enquantisierung nach dem Prinzip des Nearest Neighbor. Sie wird in Prozent angegeben und bezieht sich auf den längsten Vektor. Das heißt für eine Totzone von $z\%$ ergibt sich eine Mindestlänge

$$\text{minlength} = \frac{z}{100}|\boldsymbol{x}_{max}| \;\text{ für }\; \max = \operatorname*{argmax}_{1 \leq i \leq |\mathcal{T}|} |\boldsymbol{x}_i| \tag{3.35}$$

für die Vektoren. Alle Vektoren, die kürzer als die daraus resultierende Länge sind, werden zu Null quantisiert, das heißt auf den Nullvektor abgebildet:

$$\mathcal{Q}(\boldsymbol{x}) = \vec{0}, \;\text{ falls } |\boldsymbol{x}| < \text{minlength}. \tag{3.36}$$

Dadurch erreicht man, daß nicht so viele Codevektoren für die nahe bei Null liegenden Trainingsvektoren „verschwendet“ werden, sondern für „wichtigere“ Vektoren erhalten bleiben. Dies können wir ohne größeren Qualitätsverlust machen, da die Vektoren, die dicht am Koordinatenursprung liegen, meist unter die Wahrnehmungsschwelle fallen.

Mit den Gewichtungen der einzelnen Bänder können wir erreichen, daß bestimmte Bänder beim Codebuch–Trainieren mehr zur Codevektor–Bildung beitragen als andere. Da Quantisierungsfehler in verschiedenen Komponenten unterschiedlich

100 CV
0,091 Bit/Pixel
20,52 dB

500 CV
0,133 Bit/Pixel
24,62 dB

700 CV
0,144 Bit/Pixel
27,44 dB

1000 CV
0,156 Bit/Pixel
40,43 dB

Abbildung 3.17: Vektoren über alle Level, also 64D, keine Gewichtung.

wahrgenommen werden, möchte man mit einer Gewichtung der Komponenten bei der Erstellung des Codebuchs quasi einen gewichteten Euklidischen Abstand bei der Enquantisierung erhalten. Das heißt es ergibt sich

$$d(\boldsymbol{x}, \widehat{\boldsymbol{x}}) = \sum_{i=1}^{k} G_i(\boldsymbol{x}_i - \widehat{\boldsymbol{x}}_i)^2 \tag{3.37}$$

mit Gewichten G_i für jede Komponente. Dadurch, daß eine Komponente eine höhere Gewichtung bekommt als die anderen, trägt sie mehr zum berechneten Fehler bei; mit anderen Worten: sie „zieht“ den berechneten Codevektor mehr zu sich hin. Zum Beispiel ist das LL–Band am wichtigsten. Es sollte also höher gewichtet werden als die anderen. Die anderen Bänder beinhalten kleinere Skalen, die für die kleineren Details wichtiger sind. Da wir mit progressiver Quantisierung arbeiten, ist es für uns also wichtiger, zuerst das LL–Band hervorzuheben und in nachfolgenden Schritten die anderen „nachzuliefern“. Mit den entsprechenden Gewichtungen können wir genau das erreichen.

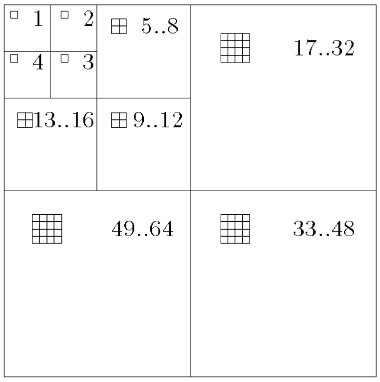


Abbildung 3.18: Aufbau der Vektoren. Die Zahlen in der Darstellung sollen die Komponente im Vektor angeben. Die erste Komponente ist also dem LL–Band entnommen, es folgt jeweils ein Pixel aus HL–, HH– und LH–Band der dritten Stufe. Die 5. bis 8. Komponente sind die entsprechenden vier Pixel aus dem HL–Band der zweiten Stufe, und zwar von links nach rechts und oben nach unten. Es folgen die vier Pixel des HH– und LH–Bandes der zweiten Stufe. Die Komponenten 17 bis 32 sind wiederum die entsprechenden Pixel aus dem HL–Band der ersten Stufe, gefolgt von den jeweils 16 Pixeln des HH– und des LH–Bandes der ersten Stufe.

Um die Wirkung von Totzone und Gewichtung zu verdeutlichen, zeigt Abb. 3.19 Beispiele im zweidimensionalen Fall für Codebücher, die ohne beziehungsweise mit den beiden Parametern für dieselbe Trainingsmenge entworfen wurden.

Bei der Frage nach der Größe der Gewichtungen in den Quantisierern der einzelnen Progressionsstufen hilft uns [WYSV97] weiter. In diesem Artikel wird untersucht, wie stark sich Quantisierungsrauschen in verschiedenen Subbändern visuell bemerkbar macht.

Um die Wellenlänge des von uns verwendeten Wavelets zu berechnen, transformieren wir einen Impuls, und aus der Pyramiden–Darstellung dieser Zerlegung lesen wir dann aus den HL–Bändern Grauwerte in den Zeilen ab und stellen sie als Funktion der Pixel in Kurven dar (siehe Abb. 3.20). Aus diesen Kurven können wir dann die Wellenlängen ablesen, und zwar sind das im HL1–Band $\lambda = 3,9$ Pixel, im HL2–Band $\lambda = 4,3$ Pixel und im HL3–Band $\lambda = 4,2$ Pixel. Da diese Werte aus interpolierten Kurven abgelesen werden, kleinere Abweichungen also zu erwarten sind, nehmen wir als Mittelwert die Wellenlänge $\lambda = 4,13$ Pixel.

Für die verschiedenen Bänder ergeben sich daraus nach Herausrechnen der Un-

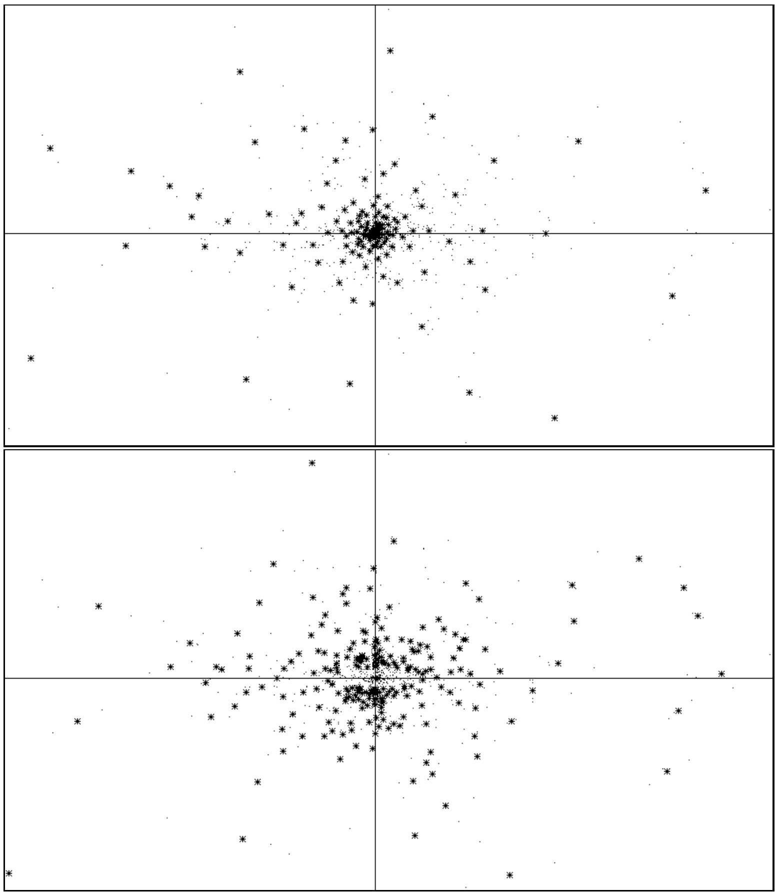


Abbildung 3.19: Codebücher für die gleiche Trainingsmenge ($2D$). Im oberen Bild wurde das Codebuch ohne Gewichtung und ohne Totzone entworfen. Das untere Bild zeigt ein Codebuch, für das die Trainingsvektoren in der x–Komponente mit 0,25 gewichtet waren, und das eine Totzone von 5% realisiert. (Die Codevektoren sind jeweils durch Sterne dargestellt, die Trainingsvektoren durch Punkte.)

terabtastung die folgenden Wellenlängen und Mittenfrequenzen:

$$\begin{array}{lllllllll}
\lambda_1 & = & 2^1 \cdot 4,13 \, \frac{\text{Pixel}}{cyc} & = & 8,26 \, \frac{\text{Pixel}}{cyc} & \Longrightarrow & f_1' & = \frac{1}{\lambda_1} & = 0,12 \, \frac{cyc}{\text{Pixel}} \\
\lambda_2 & = & 2^2 \cdot 4,13 \, \frac{\text{Pixel}}{cyc} & = & 16,52 \, \frac{\text{Pixel}}{cyc} & \Longrightarrow & f_2' & = \frac{1}{\lambda_2} & = 0,06 \, \frac{cyc}{\text{Pixel}} \\
\lambda_3 & = & 2^3 \cdot 4,13 \, \frac{\text{Pixel}}{cyc} & = & 33,04 \, \frac{\text{Pixel}}{cyc} & \Longrightarrow & f_3' & = \frac{1}{\lambda_3} & = 0,03 \, \frac{cyc}{\text{Pixel}}
\end{array}$$

Unter Berücksichtigung der DVR unseres Monitors ($\approx 39,09 \, \frac{\text{Pixel}}{deg}$) ergeben sich daraus:

$$\begin{array}{lllll}
\Longrightarrow f_1 & = & 0,12 \, \frac{cyc}{\text{Pixel}} \cdot 39,09 \, \frac{\text{Pixel}}{deg} & = & 4,69 \, \frac{cyc}{deg} \\
\Longrightarrow f_2 & = & 0,06 \, \frac{cyc}{\text{Pixel}} \cdot 39,09 \, \frac{\text{Pixel}}{deg} & = & 2,34 \, \frac{cyc}{deg} \\
\Longrightarrow f_3 & = & 0,03 \, \frac{cyc}{\text{Pixel}} \cdot 39,09 \, \frac{\text{Pixel}}{deg} & = & 1,17 \, \frac{cyc}{deg}
\end{array}$$

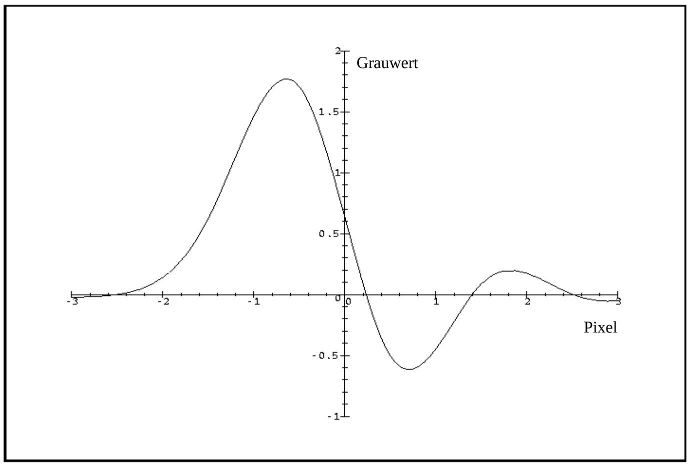


Abbildung 3.20: Kurve der Impulsantwort eines Filters. Aufgetragen sind die Grauwerte des transformierten Impulses in Abhängigkeit von den Pixeln.

Das beschriebene Modell (3.33) formen wir um zu

$$Y_{\Lambda,\Theta} = a \cdot 10^{k(\log \frac{f_\Lambda}{g_\Theta f_0})^2} \tag{3.38}$$

und mit den im Artikel angegebenen Parametern

$$\begin{aligned} a &= 0,495 \\ k &= 0,466 \\ f_0 &= 0,401 \\ g_1 &= 1,501, \quad g_2 = g_3 = 1,0, \quad g_4 = 0,534 \end{aligned}$$

und den entsprechenden Mittenfrequenzen berechnen wir die Thresholds:

Level Λ	Orientierung Θ	Mittenfrequenz $\frac{\text{cyc}}{\text{deg}}$	Threshold $Y_{\Lambda,\Theta}$
1	2	$f_1 = 4.69$	325,745
	3		325,745
	4		13620,497
2	2	$f_2 = 2.34$	13,950
	3		13,950
	4		228,744
3	2	$f_3 = 1.17$	1,694
	3		1,694
	4		10,925
	1		0,795

Die so berechneten Mittengrenzwerte helfen uns beim Festlegen der Gewichtungen der einzelnen Bänder. Die Größe des jeweiligen Wertes ist reziprokproportional zur Empfindlichkeit des Bandes. Ist also ein Wert sehr hoch, so bedeutet das, daß dieses Band wenig empfindlich gegenüber Gewichtsveränderungen ist und umgekehrt. Ein Band mit hohem $Y_{\Lambda,\Theta}$ sollte also eine kleine Gewichtung bekommen und umgekehrt. Daher kann man als Gewichtungen zum Beispiel einfach die Reziprokwerte der Thresholds nehmen, also

$$\mathrm{Gew}_{\Lambda,\Theta} = \frac{1}{\mathrm{Y}_{\Lambda,\Theta}}. \tag{3.39}$$

3.2.8 Quantisierungsarten

Die einfachste Quantisierungsart ist die, daß die Vektoren in einem Schritt quantisiert werden. Das Ergebnis wird übertragen oder gespeichert. Hierbei wird eine Codebuch–Größe gewählt, und Totzone und Gewichtung werden festgelegt. Mit diesem Codebuch wird dann die Vektormenge, also die Pyramiden–Darstellung nach der Zerlegung, quantisiert.

Näher betrachtet werden soll die hierarchische Quantisierung, da wir diese für die Arbeit verwenden. Mit der hierarchischen Quantisierung realisieren wir eine mehrschrittige Requantisierung des Fehlers. In jedem Schritt wird der Quantisierungsfehler des vorherigen verringert und somit die Gesamtapproximation verbessert.

Hierarchische Quantisierung

Bei hierarchischer Quantisierung, beziehungsweise bei deren Dekomprimierung, erhält man ein Bild, das mit jedem Schritt besser wird. Zuerst ist es eher schemenhaft, und nach und nach kommen mehr Details zum Vorschein. Das hat den Vorteil, daß man recht schnell entscheiden kann, ob dieses Bild auch wirklich das ist, was man (zum Beispiel aus einer Datenbank) haben wollte oder nicht. Die Übertragungen der einzelnen Verbesserungen gehen sehr schnell, weil die Datenmengen jeweils sehr klein sind. Stellt man also schon nach kurzer Zeit fest, daß es sich nicht um das gewünschte Bild handelt, kann man die Übertragung abbrechen und ein anderes Bild laden.

Die Enquantisierung des Bildes nimmt man in mehreren Schritten vor. Für jeden Schritt steht ein kleines Codebuch zur Verfügung, mit dem die Vektoren recht grob approximiert werden. Im nächsten Schritt wird das Differenzbild zwischen dem Original und der Approximation aus dem vorhergegangenen Schritt ebenfalls mit einem kleinen Codebuch grob quantisiert. (Wenn wir hier von Bildern sprechen, so meinen wir jeweils das Bild der Oktav–Zerlegung und die daraus gewonnenen Vektoren.) Der Vorteil dieser kleinschrittigen Enquantisierung ist, daß jeder Schritt sehr schnell abläuft, da die Komplexität der Suche des nächstgelegenen Codevektors annähernd linear abnimmt mit abnehmender Größe des Codebuchs (siehe Abb. 3.21).

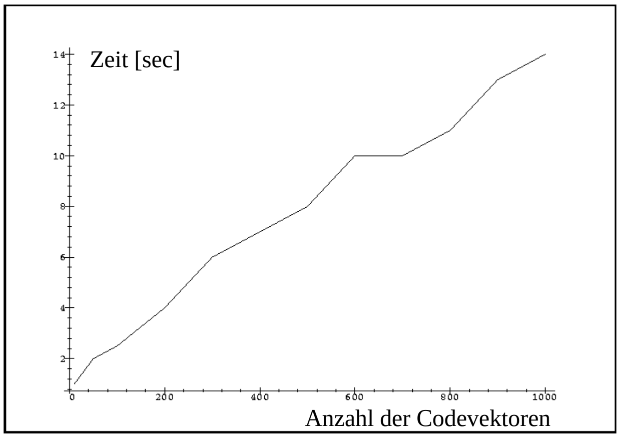


Abbildung 3.21: Kurve zur Komplexität der Nearest neighbor–Suche. Aufgetragen ist die Zeit, die für die Codevektor–Zuweisung gebraucht wird, in Abhängigkeit von der Codebuch–Größe.

Die hierarchische Quantisierung stellt eine progressive Kompression dar. Damit kann man für jeden Schritt das beste Codebuch entwerfen, so daß man bei der Dequantisierung auf der Empfängerseite auch in jedem Schritt die optimale Rekonstruktion erhält.

Ein Vektor (wir sprechen hier von Vektoren; die Aussagen gelten ebenso für Skalare, also eindimensionale Vektoren) wird hierarchisch quantisiert, indem er auf den ihm nächstgelegenen Codevektor abgebildet wird. Dann wird der Differenzvektor zwischen dem ursprünglichen und dem Codevektor wieder quantisiert und zum ersten Codevektor addiert und so weiter. Dies kann man in beliebig vielen Stufen durchführen und nähert so den Reproduktionsvektor weiter dem Original an. In Abb. 3.22 ist als Beispiel eine hierarchische Quantisierung in zwei Stufen dargestellt. Bei dieser Art der Quantisierung ist das Codebuch in jedem Schritt sehr klein, da jeweils nur relativ grob quantisiert wird [BF92].

Formal läßt sich die hierarchische Quantisierung folgendermaßen beschreiben: Sei $\boldsymbol{X}^1$ ein k–dimensionaler Zufallsvektor mit Wahrscheinlichkeitsverteilungsfunktion $F_{\boldsymbol{X}^1}(\cdot)$. Ein P–stufiger hierarchischer Quantisierer von $\boldsymbol{X}^1$ besteht aus einer Folge von P Quantisierern

$$\{(\mathcal{C}^p, \mathcal{Q}^p, \mathcal{P}^p) | 1 \leq p \leq P\}, \tag{3.40}$$

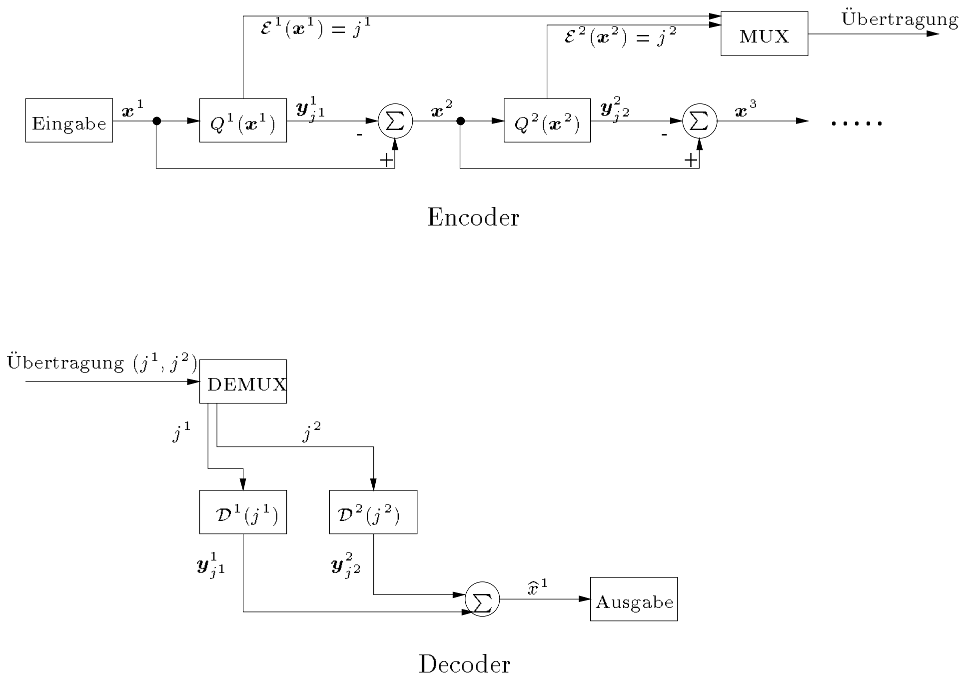


Abbildung 3.22: Encoder und Decoder bei hierarchischer Quantisierung [BF92].

wobei $\mathcal{C}^p$ das Codebuch für die p–te Stufe bezeichnet, $\mathcal{Q}^p$ ist die Quantisierungsabbildung, und $\mathcal{P}^p$ ist die Partitionierung. Dann codiert $(\mathcal{C}^1, \mathcal{Q}^1, \mathcal{P}^1)$ den Anfangsvektor $\boldsymbol{x}^1$ (eine bestimmte Realisierung von $\boldsymbol{X}^1$), und $(\mathcal{C}^{p+1}, \mathcal{Q}^{p+1}, \mathcal{P}^{p+1})$ quantisiert den Restvektor $\boldsymbol{x}^{p+1} = \boldsymbol{x}^p - \mathcal{Q}^p(\boldsymbol{x}^p)$ der vorangegangenen Stufe $(\mathcal{C}^p, \mathcal{Q}^p, \mathcal{P}^p)$ für $1 \leq p < P$. Jeder Restvektor $\boldsymbol{x}^p \in \mathbb{R}^k$ ist eine Realisierung eines Zufallsvektors $\boldsymbol{X}^p$ mit Verteilungsfunktion $F_{\boldsymbol{X}^p}(\cdot)$ auf $\mathbb{R}^k$ für $1 \leq p \leq P+1$. Die Menge der stufenweisen Restvektoren $\boldsymbol{X}^1, \ldots, \boldsymbol{X}^P$ hat die Verbundverteilung $F_{\boldsymbol{X}^1 \ldots \boldsymbol{X}^P}(\cdot)$.

Die Anzahl der Codevektoren in $\mathcal{C}^p$ und der Partitionszellen in $\mathcal{P}^p$ soll mit N^p bezeichnet werden. Die Codevektoren in $\mathcal{C}^p$ und die Zellen in $\mathcal{P}^p$ bekommen jeweils den Index j^p aus der p–ten Indexmenge $\mathcal{J}^p = \{0, 1, \ldots, N^p - 1\}$. Es sind also $\mathcal{C}^p = \{\boldsymbol{y}^p_0, \ldots, \boldsymbol{y}^p_{N^p-1}\}$ und $\mathcal{P}^p = \{Z^p_0, \ldots, Z^p_{N^p-1}\}$.

Die Abbildung $\mathcal{Q}^p : \mathbb{R}^k \longrightarrow \mathcal{C}^p$ in der p–ten Stufe bildet genau dann $\boldsymbol{x}^p$ auf $\mathcal{Q}^p(\boldsymbol{x}^p) = \boldsymbol{y}^p_{j^p}$ ab, wenn $\boldsymbol{x}^p \in Z^p_{j^p}$ ist. Der Zusammenhang zwischen den Vektoren $\boldsymbol{x}^p$ und den Abbildungen $\mathcal{Q}^p(\cdot)$ ist folgender:

$$\boldsymbol{x}^1 = \sum_{p=1}^{P} \mathcal{Q}^p(\boldsymbol{x}^p) + \boldsymbol{x}^{P+1}. \tag{3.41}$$

(Der Vektor $\boldsymbol{x}^{P+1}$ ist dabei sowohl Restfehler der jeweils letzten Stufe als auch Gesamtrestfehler aller Stufen.)

Der Quantisierer $\mathcal{Q}^p$ ist eine Komposition aus einem En– und einem Decoder: $\mathcal{Q}^p(\boldsymbol{x}^p) = \mathcal{D}^p(\mathcal{E}^p(\boldsymbol{x}^p))$ mit $\mathcal{E}^p : \mathbb{R}^k \longrightarrow \mathcal{J}^p$ und $\mathcal{E}^p(\boldsymbol{x}^p) = j^p \Longleftrightarrow \boldsymbol{x}^p \in Z^p_{j^p}$. Für jeden Eingabevektor werden die Indizes zu einem P–Tupel $\boldsymbol{j}^P = (j^1, \ldots, j^P)$ zusammengefaßt. Das P–Tupel $\boldsymbol{j}^P$ nennt man *Produkt–Codewort*, denn es ist $\boldsymbol{j}^P \in \{\mathcal{J}^1 \times \ldots \times \mathcal{J}^P\}$. Der Decoder ist die Abbildung $\mathcal{D}^p : \mathcal{J}^p \longrightarrow \mathcal{C}^p$ und $\mathcal{D}^p(j_p) = \boldsymbol{y}^p_{j_p}$. Das quantisierte Ergebnis $\widehat{\boldsymbol{x}}^1$ der Eingabe $\boldsymbol{x}^1$ ist:

$$\widehat{\boldsymbol{x}}^1 = \sum_{p=1}^{P} \boldsymbol{y}^p_{j^p}. \tag{3.42}$$

Optimierungsproblem

Das Maß $d(\boldsymbol{x}^1, \widehat{\boldsymbol{x}}^1)$ gibt die Verzerrung zwischen dem Originalvektor und seiner Rekonstruktion an. $D(\boldsymbol{x}^1, \widehat{\boldsymbol{x}}^1) = E\{d(\boldsymbol{x}^1, \widehat{\boldsymbol{x}}^1)\}$ ist die erwartete Verzerrung. Ein P–stufiger hierarchischer Quantisierer ist optimal für eine Eingabeverteilung $F_{\boldsymbol{X}^1}(\cdot)$, wenn es mindestens einen lokalen Minimalwert der mittleren Verzerrung gibt. Ein optimaler hierarchischer Quantisierer hat eine Menge $\{\mathcal{C}^p\}$ von Codebüchern und eine Menge $\{\mathcal{P}^p\}$ von Partitionen, die zusammen

$$D(\boldsymbol{x}^1, \widehat{\boldsymbol{x}}^1) = \int d(\boldsymbol{x}^1, \sum_{p=1}^{P} \mathcal{Q}^p(\boldsymbol{x}^p))\, dF_{\boldsymbol{X}^1 \ldots \boldsymbol{X}^P} \tag{3.43}$$

minimieren. Dies ist allerdings schwer zu berechnen, weil $F_{\boldsymbol{X}^1 \ldots \boldsymbol{X}^P}(\cdot)$ normalerweise unbekannt ist.

Um Gl. (3.43) zu vereinfachen, führte Barnes 1989 den Begriff des *Equivalent singlestage quantizers* (EQ) ein [BF92], also des äquivalenten einstufigen Quantisierers. Der HQ (hierarchischer Quantisierer) und der EQ sind insofern identisch, als sie für eine gleiche Eingabe auch eine gleiche Ausgabe liefern und somit die gleiche mittlere Verzerrung haben. Der EQ hat den Vorteil, daß der Erwartungswert der HQ–Abweichung abhängig von der bekannten Eingabeverteilung $F_{\boldsymbol{X}^1}(\cdot)$ ausgedrückt werden kann anstelle der unbekannten Verbundverteilung $F_{\boldsymbol{X}^1 \ldots \boldsymbol{X}^P}(\cdot)$.

Der EQ ist spezifiziert durch $(\mathcal{C}^e, \mathcal{Q}^e, \mathcal{P}^e)$, also das Tripel aus äquivalentem Codebuch, äquivalenter Abbildungsvorschrift und äquivalenter Partition. Die Elemente von $\mathcal{C}^e$ sind die Elemente der Menge aller möglichen Summen von stufenweisen Codevektoren $\mathcal{C}^e = \mathcal{C}^1 + \cdots + \mathcal{C}^P$ mit je einem Vektor aus jeder Stufe. Jeder Codevektor $\boldsymbol{y}^e \in \mathcal{C}^e$ kann durch das P–Tupel $\boldsymbol{j}^P = (j^1, \ldots, j^P)$ indiziert werden und geschrieben als

$$\boldsymbol{y}^e(\boldsymbol{j}^P) = \sum_{p=1}^{P} \boldsymbol{y}^p_{j^p}. \tag{3.44}$$

Die $\boldsymbol{j}^P$–te äquivalente Zelle des hierarchischen Quantisierers ist die Teilmenge $Z^e(\boldsymbol{j}^P) \subset \mathbb{R}^k$, so daß alle $\boldsymbol{x}^1 \in Z^e(\boldsymbol{j}^P)$ vom Quantisierer auf $\boldsymbol{y}^e(\boldsymbol{j}^P)$ abgebildet werden, das heißt

$$Z^e(\boldsymbol{j}^P) = \{\boldsymbol{x}^1 | \sum_{p=1}^{P} \mathcal{Q}^p(\boldsymbol{x}^p) = \boldsymbol{y}^e(\boldsymbol{j}^P)\}. \tag{3.45}$$

(Anmerkung: alle $Z^e(\boldsymbol{j}^P)$ sind nichtleer.)

Die äquivalente Partition ist einfach die Menge aller äquivalenten Zellen. Die äquivalente Abbildung $\mathcal{Q}^e : \mathbb{R}^k \longrightarrow \mathcal{C}^e$ ist definiert durch

$$\mathcal{Q}^e(\boldsymbol{x}^1) = \boldsymbol{y}^e(\boldsymbol{j}^P) \Longleftrightarrow \boldsymbol{x}^1 \in Z^e(\boldsymbol{j}^P). \tag{3.46}$$

Die mittlere Verzerrung des EQs ist

$$D(\boldsymbol{x}^1, \widehat{\boldsymbol{x}}^1) = \int d(\boldsymbol{x}^1, \mathcal{Q}^e(\boldsymbol{x}^1))\, dF_{\boldsymbol{X}^1} \tag{3.47}$$

als Ausdruck der bekannten Eingabeverteilungsfunktion $F_{\boldsymbol{X}^1}(\cdot)$. Aufgrund der Konstruktion sind die Ergebnisse des hierarchischen Quantisierers und des EQs gleich, das heißt es ist

$$\mathcal{Q}^e(\boldsymbol{x}^1) = \sum_{p=1}^{P} \mathcal{Q}^p(\boldsymbol{x}^p). \tag{3.48}$$

Daher muß auch die mittlere Abweichung der beiden gleich sein, aber Gl. (3.47) ist einfacher zu berechnen als Gl. (3.43).

3.3 Codierung der Indizes

Mit Hilfe des generierten Codebuchs können die Vektoren quantisiert werden. Daraus erhält man die Indizes der entsprechenden Codevektoren. Diese werden dann entropiecodiert, damit sie anschließend übertragen oder gespeichert werden können. Entropiecodierung ist eine verlustfreie Codierung von Symbolen aus einer Menge von Symbolen, in diesem Fall der Menge der Indizes $\mathcal{J} = \{1, \ldots, N\}$. Jeder dieser Indizes j hat eine Auftrittswahrscheinlichkeit $p(j)$. Die mittlere Rate, mit der diese Indizes codierbar sind, ist ihr mittlerer Informationsgehalt, also die Entropie $H(\mathcal{J})$ (vergleiche auch Gl. (2.8)). Das Ziel ist, annähernd mit der Rate der Entropie zu übertragen, was man mit einem Code variabler Länge erreichen kann. Das bedeutet, daß jedem Index die Anzahl von Bits als Code zugeordnet wird, die seinem Informationsgehalt (Gl. (2.6)) entspricht.

Es gibt mehrere Arten von Entropiecodierungen. Eine geläufige, die *Huffman–Codierung*, wollen wir hier beispielhaft näher erläutern. Für weitere soll auf die Literatur (zum Beispiel [GG92], [Ohm95]) verwiesen werden.

3.3.1 Huffman–Codierung

Beim Entwurf eines Huffman–Codes entsteht ein Binärbaum, der von den Blättern zur Wurzel hin aufgebaut wird. An den Blättern stehen beim fertigen Baum die zu codierenden Symbole, und deren Codeworte können von der Wurzel zum Blatt als Beschriftung der Äste abgelesen werden.

Gegeben sei eine Liste $\mathcal{L}$ mit Symbolen $a_0, \ldots, a_{M-1}$ und ihren zugehörigen Wahrscheinlichkeiten $p(a_0), \ldots, p(a_{M-1})$. Diese Liste sei nach absteigender Wahrscheinlichkeit sortiert, also $p(a_0) \geq \ldots \geq p(a_{M-1})$.

Nun nimmt man die beiden Symbole mit den kleinsten Wahrscheinlichkeiten. Diese werden zu „Brüdern" im Baum, und man ordnet ihnen einen Vater–Knoten zu. An den Ästen vom Vater zu den Söhnen steht einmal eine 0 und einmal eine 1. In der Liste werden die beiden Symbole entfernt und dafür ein neues eingefügt, das eine Verschmelzung der beiden darstellt und als Wahrscheinlichkeit die Summe der beiden Einzelwahrscheinlichkeiten zugeordnet bekommt. Sind in der Liste noch zwei Symbole vorhanden, so wird von vorne begonnen, andernfalls ist der Baum fertig. (Abb. 3.23 zeigt ein Beispiel.)

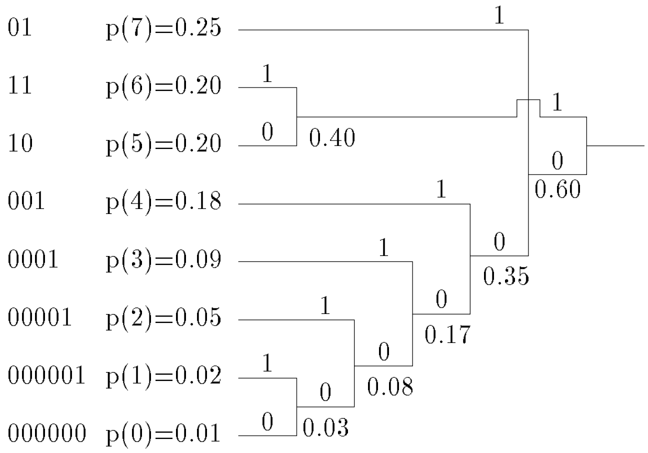


Abbildung 3.23: Huffman–Code [GG92].

Eine Zusammenfassung des Algorithmus' sieht folgendermaßen aus:

1. Gegeben sei eine Liste $\mathcal{L}$ von Symbolen mit ihren zugehörigen Wahrscheinlichkeiten.
2. Nimm die beiden Symbole mit den kleinsten Wahrscheinlichkeiten und mache sie zu „Brüdern" im Baum. Füge einen neuen Knoten als „Vater" der beiden ein und bezeichne den Pfad zum einen Sohn mit 0 und zum anderen mit 1.

3. Ersetze in der Liste die beiden Symbole und ihre Wahrscheinlichkeiten durch einen neuen Knoten mit der Summe der Wahrscheinlichkeiten. Wenn die Liste jetzt nur noch ein Element enthält, ist der Baum fertig. Ansonsten gehe zu Schritt 2.

Kapitel 4

Experimente

Für diese Arbeit stehen uns 20 Bilder zur Verfügung, die alle 256×256 Pixel groß sind und ein Gesicht vor einem weißen Hintergrund zeigen. Die Gesichter auf den Bildern sind von Männern und Frauen, Brillen– und Bartträgern, Kindern und Erwachsenen, Lang– und Kurzhaarigen. In Abb. 4.1 sind einige Beispiele zu sehen.

Abbildung 4.1: Einige Bilder aus der Datenbank.

Alle Bilder, die wir für diese Arbeit komprimieren, werden mit Hilfe einer dreistufigen Wavelet–Zerlegung in Subbänder aufgetrennt. Diese werden dann in einer Oktav–Pyramide dargestellt, und aus dem Bild dieser Oktav–Pyramide stellen wir die Vektoren zusammen, die enquantisiert werden.

Um nun verschiedene Experimente durchzuführen, überlegen wir uns zunächst die Parameter, die man dabei variieren kann. Diese beziehen sich zum einen auf die Informationsquelle und beeinflussen die Trainingsmenge. Mit Festlegung dieser bestimmt man die Statistik der zu codierenden Bilder. Zum anderen kann man mit der Gewichtung einzelner Subbänder und der Einstellung einer Totzone den Eigenschaften des Menschlichen Visuellen Systems Rechnung tragen. Ein letzter Punkt ist gegeben durch die Bandbreite, auf der man überträgt. Hierfür muß man entscheiden, wieviele Progressionsschritte man durchführen will und wie groß die Codebücher in jedem Schritt sein sollen. Damit sich die progressive Codierung überhaupt lohnt, sollte die Übertragungsdauer lang genug sein, so daß ein Abbruch nach Lösung der Aufgabe auch etwas bringt. Dauert die gesamte Übertragung eines progressiv codierten Bildes zum Beispiel nicht mehr als eine halbe Sekunde, so ist Progression überflüssig, wohingegen es sich schon lohnt, eine Übertragung abzubrechen, wenn sie insgesamt 5 Sekunden dauern würde. Wenn man sich vorstellt, daß man eine ganze Bilddatenbank durchsucht (englisch: browse), das heißt man will mehrere Bilder (teilweise) ansehen, so kann eine Sekunde Wartezeit schon zu lang sein. Dies muß man in die Überlegungen mit einbeziehen.

Fassen wir die Parameter also zusammen:

$\boxed{L}$ Progressionsschritte, $l = 1, \ldots, L$

Trainingsmenge $\boxed{\mathcal{T}_l}$ für Codebuch $\mathcal{C}_l$

$\boxed{N(l)}$ Codevektoren in $\mathcal{C}_l$

Gewichtsvektor $\boxed{\vec{g}(l) = (g_0, \ldots, g_M)}$ für $\mathcal{C}_l$

M: Anzahl der Pyramidenstufen
g_0: LL–Band
g_1: 3. Stufe
g_2: 2. Stufe
g_3: 1. Stufe

Totzone $\boxed{\mathrm{DZ}(l)}$ für $\mathcal{C}_l$

Beim Gewichtungsvektor haben wir $M = 3$ gesetzt, was einer Gewichtung für jede Ebene der Pyramide entspricht. Man kann auch zum Beispiel $M = 9$ wählen und jedem Subband eine einzelne Gewichtung zuordnen.

Die Experimente haben wir mit Hilfe der Software–Entwicklungsumgebung Khoros 2.0.2 auf einer SUN Workstation durchgeführt.

4.1 Variation der Trainingsmenge

Das erste Experiment bezieht sich auf die Auswahl der Trainingsmenge. Stellen wir uns eine Bilddatenbank vor, in der verschiedene Bilder gespeichert werden sollen, so ist die Frage, wie man die Codebücher für diese Bilder anlegt. Die Kosten für die Übertragung des Codebuchs sollen dabei möglichst gering, aber natürlich soll die Anpassung des Codebuchs an Bild und Aufgabe dennoch möglichst optimal sein. Das eine Extrem ist, daß man für jedes Bild ein eigenes Codebuch entwirft. Will man anschließend die Datenbank durchsuchen, so muß man für jedes Bild, das man ansehen will, erst das Codebuch übertragen, was sehr aufwendig ist. Das andere Extrem ist, daß man für alle Bilder dasselbe Codebuch nimmt. Damit dieses aber auch dennoch jedes Bild möglichst gut approximiert, müßte es sehr groß sein. Das wiederum würde hohen Übertragungsaufwand bedeuten. Ein Codevektor hat im „Roh"–Zustand, wenn er also nicht entropiecodiert ist, einen Speicherverbrauch oder eine Übertragungsgröße von 4 Byte pro Komponente, bei unseren $64D$–Vektoren sind das also $64 \cdot 4 = 256$ Byte pro Vektor.

Den „goldenen" Mittelweg untersuchen wir im ersten Experiment: wir entwerfen ein Codebuch für eine Bildklasse, in unserem Fall die Klasse der Gesichtsbilder. Ein solches Codebuch ist zwar größer als ein Codebuch, das für ein einzelnes Bild entworfen wird, aber es sollte kleiner sein können als ein Codebuch für alle Arten von Bildern, da es die Statistiken der entsprechenden Bilder ausnutzen kann. Dieses Codebuch muß man dann auch nur einmal übertragen und kann damit mehrere Bilder decodieren. Das hieße im Fall einer Datenbank, die nur eine Klasse von Bildern enthält, daß man nur einmal die Übertragungszeit und die Übertragungskosten für das Codebuch hat und damit die gesamte Datenbank durchsuchen kann.

In diesem Versuch codieren wir nicht progressiv. Wir haben verschiedene Trainingsmengen und erstellen damit Codebücher, die alle gleich groß sind. Bei der Codebuch–Erstellung sind die Vektoren nicht gewichtet, und es wird keine Totzone angegeben.

$L = 1$		
Trainingsmenge:		
	$\mathcal{T}_1 = \{\text{Einzelbild}\}$	(a)
	$\mathcal{T}_1 = \{\text{20 Gesichtsbilder}\}$	(b)
	$\mathcal{T}_1 = \{\text{19 Gesichtsbilder}\}$	(c)
	$\mathcal{T}_1 = \{\text{12 gemischte Bilder}\}$	(d)
$N(1) = 700$		
$\vec{g}(1) = (1,0;\ 1,0;\ 1,0;\ 1,0)$		
$\mathrm{DZ}(1) = 0,00\%$		

Die Menge der 19 Gesichtsbilder enthält Bilder aus der Bilddatenbank. Die Menge der 20 Gesichtsbilder ist dieselbe Menge, und zusätzlich ist das Bild „Aenne" enthalten. Die 12 gemischten Bilder enthalten keine Gesichter. Beispielhaft zeigt Abb. 4.2 vier der Bilder aus der Menge.

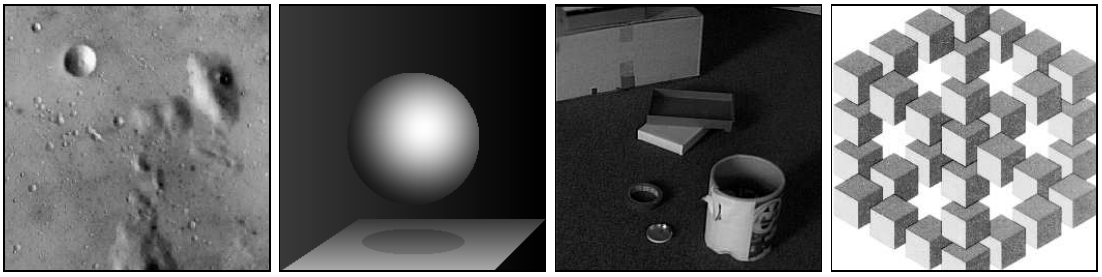


Abbildung 4.2: Vier Bilder aus der Menge der 12 gemischten Bilder.

Zunächst codieren wir mit diesen Parametern das Bild „Aenne" (siehe Abb. 4.3). Zum Vergleich zeigt Abb. 4.4 das Bild „Airport", auf dieselbe Art codiert (Version (b) haben wir allerdings ausgelassen, da es für die Lösung der Aufgabe keinen Unterschied macht, ob „Aenne" in der Trainingsmenge enthalten ist oder nicht).

Man sieht bei beiden Testbildern, daß das Bild am besten ist, dessen Codebuch nur mit sich selbst trainiert wird (Version (a)). Das verwundert auch nicht weiter. Am Beispiel „Aenne" sieht man außerdem, daß die Approximation besser ist, wenn das Bild selber mit in der Trainingsmenge enthalten ist (Version (b) gegenüber (c)). (Zumindest ist dies der Fall bei unserer doch relativ kleinen Trainingsmenge von 20 Bildern. Es bliebe zu untersuchen, wie das Ergebnis bei größeren Trainingsmengen ausfällt.) Version (c) beim Testbild „Airport" und Version (d) bei beiden Testbildern fallen wie erwartet schlecht aus, da die Trainingsmenge der Codebücher nicht auf den Bildinhalt abgestimmt ist.

Wir sehen also, daß es durchaus Sinn macht, ein Codebuch für eine Klasse von Bildern zu entwerfen. Damit erhält man ein Codebuch, mit dem man mehrere Bilder codieren beziehungsweise decodieren kann. Man muß also auch nur ein Codebuch übertragen, wenn man mehrere Bilder aus einer Klasse übertragen will.

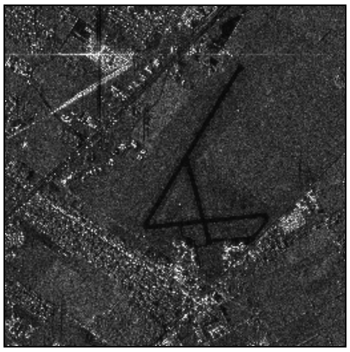
(a)
0,136 Bit/Pixel
31,32 dB

(b)
0,117 Bit/Pixel
25,26 dB

(c)
0,114 Bit/Pixel
22,26 dB

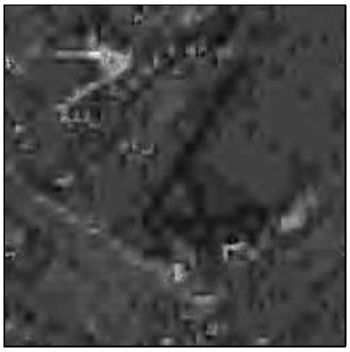
(d)
0,059 Bit/Pixel
21,26 dB

Abbildung 4.3: Das Bild „Aenne“, erst im Original und dann vier nicht progressiv codierte Versionen mit Codebüchern, die jeweils 700 Vektoren enthalten und mit Hilfe von verschiedenen Trainingsmengen entworfen wurden.



(a)
0,143 Bit/Pixel
14,04 dB

(c)
0,083 Bit/Pixel
6,96 dB



(d)
0,042 Bit/Pixel
6,68 dB

Abbildung 4.4: Das Bild „Airport", erst im Original, dann nicht progressiv codiert mit verschiedenen Codebüchern.

4.2 Variation der Anzahl der Codevektoren

Im zweiten Versuch untersuchen wir die Wirkung von unterschiedlich großen Codebüchern auf die Codierleistung. Die Frage hierbei ist, ob man mit kleinen Codebüchern in jedem Progressionsschritt auskommt. Diese zu übertragen kostet wenig. Andererseits können große Codebücher bessere Approximationen liefern. Wichtig ist auch die Überlegung, wie gut die Approximation und die Kompressionsrate des ersten Bildes sind. Das erste Bild soll schon einen möglichst guten Eindruck vom Bildinhalt vermitteln, aber es soll auch schnell übertragen werden, das heißt es soll möglichst klein sein. Mit der Anzahl der Codevektoren in den Codebüchern beeinflussen wir also die Kompressionsrate in den Übertragungsschritten, wobei die des ersten Schrittes schon sehr gut sein sollte. Desweiteren wird die Verbesserung der Approximationen in jedem Schritt durch die Codebuch–Größe bestimmt. Dabei ist noch die Anpassung an die jeweilige Bandbreite erforderlich.

In diesem Versuch codieren wir progressiv ohne Gewichtung und Totzone in den einzelnen Codebüchern. Die Trainingsmenge für die Codebücher ist jeweils nur das Bild „Aenne“ selbst beziehungsweise das jeweilige Restbild.

$L = 12$	
$\mathcal{T}_1 = \{\text{Einzelbild}\}$ $\mathcal{T}_l = \{\text{Restbild}\}$ für $l = 2, \ldots, 12$	
Anzahl der Vektoren:	
$N(l) = 5$ für $l = 1, \ldots, 12$	(a)
$N(l) = 50$ für $l = 1, \ldots, 12$	(b)
$\vec{g}(l) = (1,0;\ 1,0;\ 1,0;\ 1,0)$ für $l = 1, \ldots, 12$	
$\mathrm{DZ}(l) = 0,00\%$ für $l = 1, \ldots, 12$	

Die Abbildungen 4.5 und 4.6 zeigen die Approximationen für die einzelnen Progressionsschritte. Die Werte Coderate, Gesamt–SNR und SNR der einzelnen Bänder der Bilder sind in den Tabellen 4.1 und 4.2 aufgeführt. Abb. 4.7 zeigt jeweils den SNR in Abhängigkeit von der Progressionsstufe.

Betrachtet man die Approximationen in den beiden Testreihen, so stellt man fest, daß bei Version (a) die erste Approximation eine sehr gute Kompressionsrate hat. Dafür ist bei (b) die erste Approximation viel besser. Aus den Rate–Distortion–Funktionen (Abb. 4.7) der beiden Testreihen kann man allerdings ablesen, daß der SNR bei (a) recht schnell „gesättigt“ ist, also schon nach wenigen Schritten nicht mehr erkennbar zunimmt. Bei (b) hingegen verbessert er sich in jedem Schritt deutlich und zeigt, daß es in den einzelnen Bändern noch Approximationsreserven gibt. Das führt uns zu der Überlegung, daß eine selektive Quantisierung der Bänder Sinn machen kann. Damit soll sich die Auflösung des Vektorquantisierers auf die Bänder konzentrieren, die die größte Approximationsverbesserung

	SNR [dB] der einzelnen Bänder				Coderate	Gesamt–
	LL–Band	3. Band	2. Band	1. Band	[Bit/Pixel]	SNR [dB]
1	25,37	0,08	0,04	0,03	0,026	18,92
2	30,79	2,07	0,12	0,07	0,052	20,46
3	32,68	4,72	0,25	0,11	0,077	21,40
4	34,71	5,73	0,78	0,16	0,101	21,97
5	35,35	7,08	1,17	0,21	0,130	22,42
6	36,36	8,17	1,63	0,30	0,154	22,86
7	37,79	8,99	2,04	0,38	0,179	23,23
8	37,89	9,50	2,75	0,50	0,200	23,63
9	38,32	10,36	3,17	0,56	0,231	23,93
10	38,88	11,19	3,26	0,59	0,264	24,08
11	39,25	11,34	3,48	0,63	0,295	24,20
12	39,65	11,53	3,62	0,67	0,328	24,30

Tabelle 4.1: Werte zu Abb. 4.5.

pro Bit versprechen. Im nächsten Versuch setzen wir diese Überlegung mit Hilfe einer Nachquantisierung der einzelnen Bänder in die Tat um.

	SNR [dB] der einzelnen Bänder				Coderate	Gesamt–
	LL–Band	3. Band	2. Band	1. Band	[Bit/Pixel]	SNR [dB]
1	29,86	3,63	0,89	0,67	0,063	21,24
2	36,40	8,38	2,77	1,49	0,125	23,82
3	40,39	11,45	4,61	2,32	0,189	25,50
4	42,71	14,45	6,39	3,27	0,252	27,01
5	44,45	16,64	8,16	4,13	0,316	28,30
6	46,16	18,41	9,74	5,10	0,377	29,52
7	48,04	20,04	11,25	6,03	0,442	30,67
8	49,74	21,47	12,54	6,93	0,504	31,72
9	51,26	22,87	13,72	7,86	0,566	32,74
10	52,33	24,13	14,87	8,74	0,630	33,71
11	53,63	25,45	15,98	9,57	0,701	34,63
12	54,71	26,66	17,11	10,48	0,768	35,60

Tabelle 4.2: Werte zu Abb. 4.6.

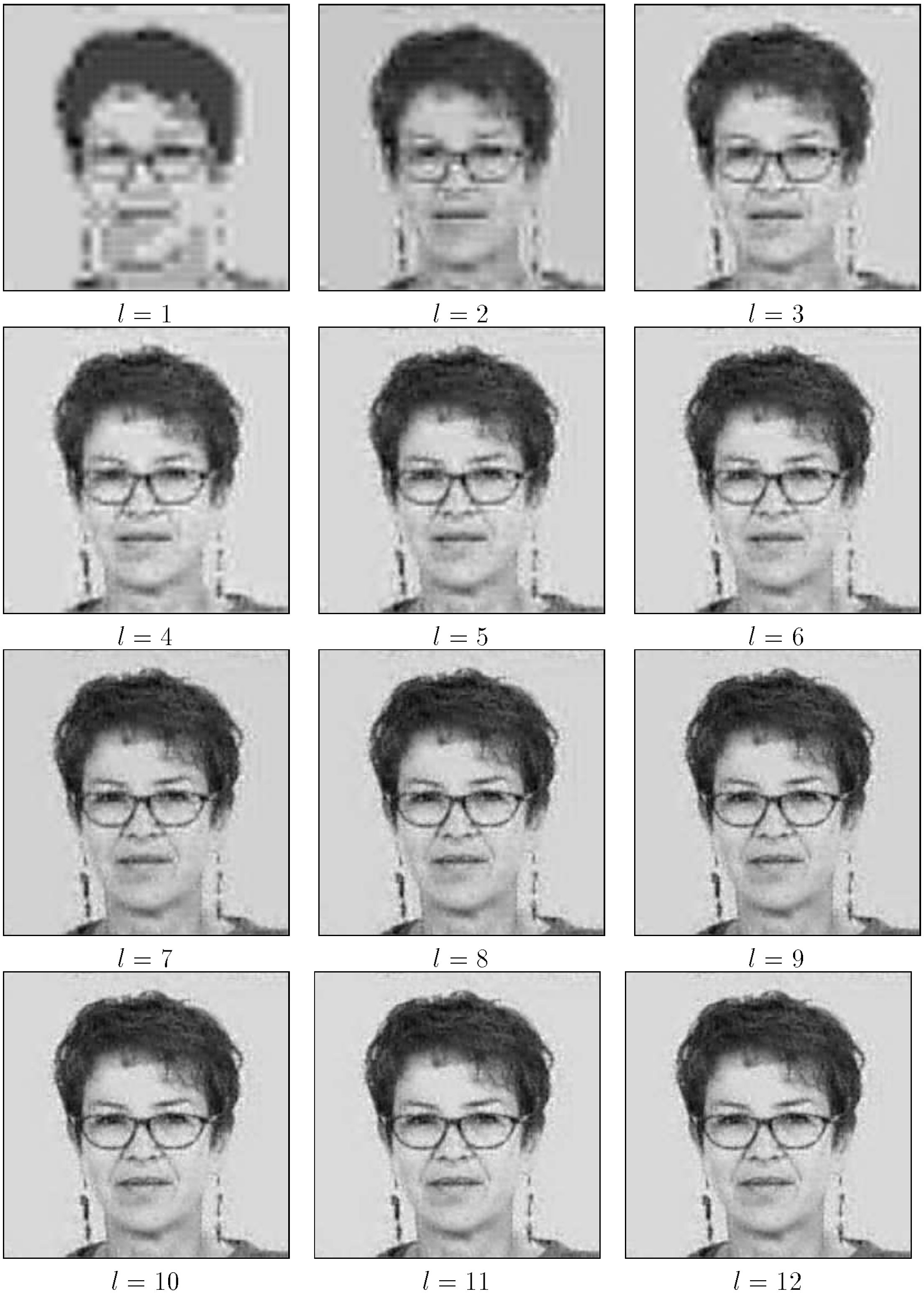


Abbildung 4.5: „Aenne“ progressiv quantisiert in 12 Schritten. Das Codebuch in jeder Stufe hat 5 Vektoren (Versuch (a)).

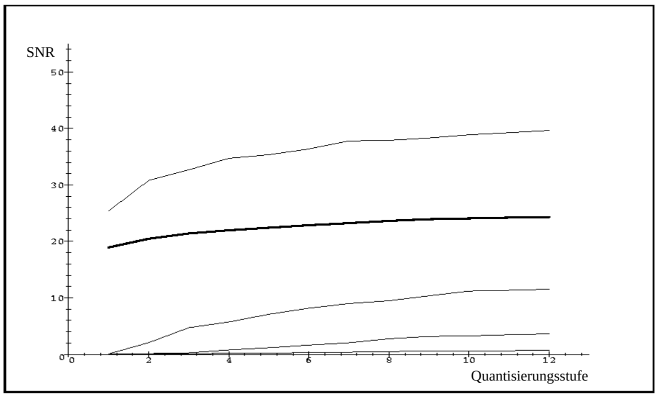
$l = 1$ $l = 2$ $l = 3$

$l = 4$ $l = 5$ $l = 6$

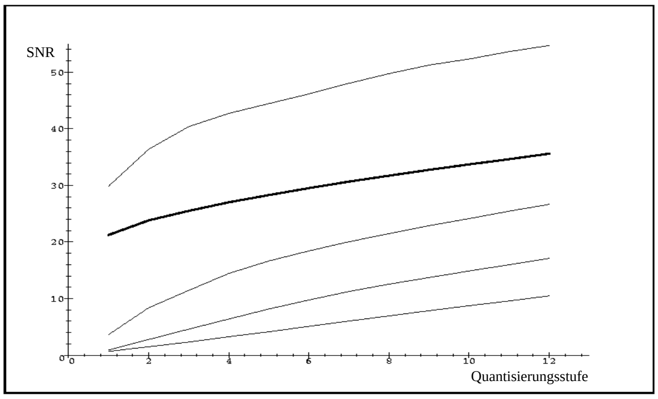
$l = 7$ $l = 8$ $l = 9$

$l = 10$ $l = 11$ $l = 12$

Abbildung 4.6: „Aenne“ progressiv quantisiert in 12 Schritten. Das Codebuch in jeder Stufe hat 50 Vektoren (Versuch (b)).



$N = 5$



$N = 50$

Abbildung 4.7: SNR in Abhängigkeit von der Quantisierungsstufe. Die breitere Linie zeigt den Verlauf des Gesamt–SNRs, die anderen sind die Kurven für die einzelnen Bänder: ganz oben die des LL–Bandes, dann nach unten 3., 2. und 1. Ebene. Dargestellt sind die Kurven für $N = 5$ Codevektoren (Versuch (a)) und $N = 50$ Codevektoren (Versuch (b)).

4.3 Variation der Gewichtung

In diesem Versuch variieren wir die Gewichtung der einzelnen Bänder und wollen daran sehen, ob eine Nachquantisierung bessere Approximationen liefert. Davon versprechen wir uns eine gezielte Verbesserung in den einzelnen Progressionsschritten. Es sollen diejenigen Bänder hervorgehoben werden, die für die Lösung der Aufgabe wichtiger sind. Mit den Gewichtungen können die „unwichtigen“ Bänder erst vernachlässigt werden, und bei ihrer Quantisierung kann man durch die Gewichtung der schon quantisierten Bänder jene nachquantisieren.

Wir führen eine progressive Codierung durch, die für 3 Schritte jeweils dieselbe Gewichtung hat. Die Trainingsmenge für die Codebücher in den einzelnen Schritten ist jeweils das Bild selbst beziehungsweise das Restbild. In jedem Schritt ist das Codebuch gleich groß, und es ist keine Totzone angegeben.

$L = 12$	
$\mathcal{T}_1 = \{\text{Einzelbild}\}$ $\mathcal{T}_l = \{\text{Restbild}\}$ für $l = 2, \ldots, 12$	
$N(l) = 5$ für $l = 1, \ldots, 12$	
Gewichtungen:	
$g(l) = (1,0;\ 0,0;\ 0,0;\ 0,0)$ für $l = 1, 2, 3$ $g(l) = (0,0;\ 1,0;\ 0,0;\ 0,0)$ für $l = 4, 5, 6$ $g(l) = (0,0;\ 0,0;\ 1,0;\ 0,0)$ für $l = 7, 8, 9$ $g(l) = (0,0;\ 0,0;\ 0,0;\ 1,0)$ für $l = 10, 11, 12$	(a)
$g(l) = (1,0;\ 0,0;\ 0,0;\ 0,0)$ für $l = 1, 2, 3$ $g(l) = (1,0;\ 1,0;\ 0,0;\ 0,0)$ für $l = 4, 5, 6$ $g(l) = (1,0;\ 1,0;\ 1,0;\ 0,0)$ für $l = 7, 8, 9$ $g(l) = (1,0;\ 1,0;\ 1,0;\ 1,0)$ für $l = 10, 11, 12$	(b)
$\mathrm{DZ}(l) = 0,00\%$ für $l = 1, \ldots, 12$	

Die Approximationen in den einzelnen Schritten zeigen die Abbildungen 4.8 und 4.9. Die Werte Coderate, Gesamt–SNR und SNR der einzelnen Bänder der Bilder sind in den Tabellen 4.3 und 4.4 aufgeführt. Abb. 4.10 zeigt jeweils den SNR in Abhängigkeit von der Progressionsstufe.

An diesem Versuch sehen wir, daß sich unsere Überlegungen nicht bestätigen. Bei den SNR–Kurven sieht man so gut wie keinen Unterschied zwischen den beiden Gewichtungen, und die Betrachtung der Bilder zeigt das gleiche Ergebnis.

	SNR [dB] der einzelnen Bänder				Coderate	Gesamt–
	LL–Band	3. Band	2. Band	1. Band	[Bit/Pixel]	SNR [dB]
1	22,84	0,00	0,00	0,00	0,024	18,18
2	25,18	0,00	0,00	0,00	0,052	18,84
3	39,19	0,00	0,00	0,00	0,078	19,92
4	39,19	4,01	0,00	0,00	0,097	21,34
5	39,19	6,65	0,00	0,00	0,124	21,88
6	39,19	9,36	0,00	0,00	0,148	22,21
7	39,19	9,36	1,12	0,00	0,172	22,75
8	39,19	9,36	1,86	0,00	0,199	23,07
9	39,19	9,36	2,76	0,00	0,219	23,41
10	39,19	9,36	2,76	0,02	0,242	23,42
11	39,19	9,36	2,76	0,02	0,262	23,42
12	39,19	9,36	2,76	0,06	0,270	23,44

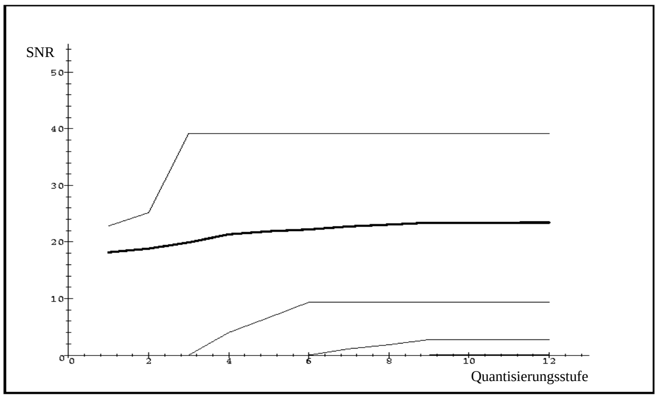
Tabelle 4.3: Werte zu Abb. 4.8.

	SNR [dB] der einzelnen Bänder				Coderate	Gesamt–
	LL–Band	3. Band	2. Band	1. Band	[Bit/Pixel]	SNR [dB]
1	22,84	0,00	0,00	0,00	0,024	18,177
2	25,18	0,00	0,00	0,00	0,052	18,837
3	39,19	0,00	0,00	0,00	0,078	19,920
4	39,29	4,01	0,00	0,00	0,097	21,345
5	39,35	6,62	0,00	0,00	0,123	21,879
6	39,36	9,26	0,00	0,00	0,146	22,204
7	39,42	9,38	1,35	0,00	0,162	22,859
8	39,46	9,72	2,19	0,00	0,180	23,246
9	39,68	10,10	2,93	0,00	0,201	23,571
10	39,71	10,88	3,37	0,16	0,223	23,886
11	39,74	11,26	3,82	0,38	0,240	24,196
12	40,00	12,39	4,13	0,50	0,265	24,467

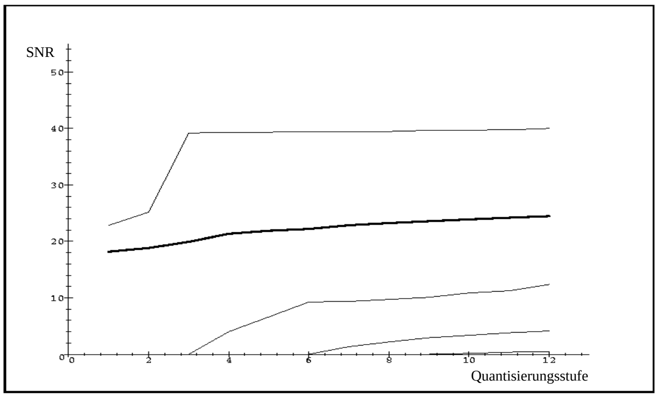
Tabelle 4.4: Werte zu Abb. 4.9.

Abbildung 4.8: „Aenne“ progressiv quantisiert in 12 Schritten, immer für 3 Schritte dieselbe Gewichtung (Versuch (a)).

Abbildung 4.9: „Aenne“ progressiv quantisiert in 12 Schritten, immer für 3 Schritte dieselbe Gewichtung (Versuch (b)).



Gewichtung (a)



Gewichtung (b)

Abbildung 4.10: SNR in Abhängigkeit von der Quantisierungsstufe. Die breitere Linie zeigt den Verlauf des Gesamt–SNRs, die anderen sind die Kurven für die einzelnen Bänder: ganz oben die des LL–Bandes, dann nach unten 3., 2. und 1. Ebene. Dargestellt sind die Kurven für die beiden verschiedenen Gewichtungen.

4.4 Variation der Anzahl der Codevektoren mit Nachquantisierung

In diesem Versuch untersuchen wir eine Nachquantisierung der Bänder und variieren dabei die Anzahl der Codevektoren. Damit schlagen wir eine Brücke zwischen dem zweiten und dem dritten Experiment. Diesmal haben wir mehr Codevektoren als beim vorigen Experiment, und die Gewichte für die Nachquantisierungen sind kleiner.

Wir codieren das Bild progressiv, wobei die Trainingsmengen der Codebücher jeweils nur aus dem Bild beziehungsweise aus dem Restbild bestehen. Es ist keine Totzone angegeben.

$L = 12$		
$\mathcal{T}_1 = \{\text{Einzelbild}\}$ $\mathcal{T}_l = \{\text{Restbild}\}$ für $l = 2, \dots, 12$		
Anzahl der Vektoren:		
	$N(l) = 50$ für $l = 1, \dots, 12$	(a)
	$N(l) = 100$ für $l = 1, \dots, 12$	(b)
$g(1) = (1,0;\ 0,0;\ 0,0;\ 0,0)$ $g(2) = (0,5;\ 1,0;\ 0,0;\ 0,0)$ $g(3) = (0,5;\ 0,5;\ 1,0;\ 0,0)$ $g(4) = (0,5;\ 0,5;\ 0,5;\ 1,0)$ $g(l) = (0,25;\ 0,5;\ 1,0;\ 1,0)$ für $l = 5, \dots, 12$		
$\mathrm{DZ}(l) = 0,00\%$ für $l = 1, \dots, 12$		

Die Abbildungen 4.11 und 4.12 zeigen die Approximationen in den einzelnen Progressionsschritten. Die Werte Coderate, Gesamt–SNR und SNR der einzelnen Bänder zu den Bildern sind in den Tabellen 4.5 und 4.6 aufgeführt. Abb. 4.13 zeigt jeweils den SNR in Abhängigkeit von der Progressionsstufe.

Es zeigt sich, daß der SNR bei der größeren Anzahl von Codevektoren (Versuch (b)) in jedem Schritt stärker ansteigt als bei der kleineren Anzahl (Versuch (a)). Dies gilt allerdings auch für die Coderate. Subjektiv stellt man beim Betrachten der Bilder diese Unterschiede so nicht fest. Das heißt die kleinere Anzahl von Codevektoren in jedem Schritt ist ausreichend.

	SNR [dB] der einzelnen Bänder				Coderate	Gesamt–
	LL–Band	3. Band	2. Band	1. Band	[Bit/Pixel]	SNR [dB]
1	36,78	0,00	0,00	0,00	0,071	19,88
2	38,37	9,88	0,00	0,00	0,141	22,24
3	39,28	10,57	4,30	0,00	0,199	24,05
4	40,44	11,76	5,37	1,98	0,246	25,64
5	40,89	12,65	7,55	3,07	0,301	27,01
6	41,42	13,85	9,29	4,15	0,359	28,25
7	41,95	15,25	10,93	5,19	0,416	29,45
8	42,37	16,53	12,34	6,11	0,476	30,49
9	42,97	17,81	13,79	7,08	0,537	31,56
10	43,51	19,17	15,07	8,06	0,595	32,60
11	44,28	20,22	16,30	9,06	0,656	33,62
12	44,93	21,37	17,41	10,05	0,719	34,62

Tabelle 4.5: Werte zu Abb. 4.11.

	SNR [dB] der einzelnen Bänder				Coderate	Gesamt–
	LL–Band	3. Band	2. Band	1. Band	[Bit/Pixel]	SNR [dB]
1	36,86	0,00	0,00	0,00	0,082	19,88
2	38,75	10,99	0,00	0,00	0,171	22,33
3	39,93	11,98	4,79	0,00	0,249	24,34
4	41,96	13,43	6,49	2,84	0,317	26,69
5	42,63	14,81	9,37	4,45	0,389	28,62
6	43,84	16,52	11,92	6,22	0,459	30,54
7	44,59	18,43	14,20	7,90	0,532	32,33
8	45,55	20,31	16,12	9,54	0,607	34,00
9	46,82	21,92	18,09	11,16	0,678	35,67
10	48,08	23,86	20,08	12,72	0,754	37,32
11	49,13	25,58	21,72	14,23	0,830	38,83
12	50,17	27,19	23,42	15,77	0,904	40,35

Tabelle 4.6: Werte zu Abb. 4.12.

Abbildung 4.11: „Aenne“ progressiv quantisiert in 12 Schritten, 50 Codevektoren in jedem Schritt (Versuch (a)).

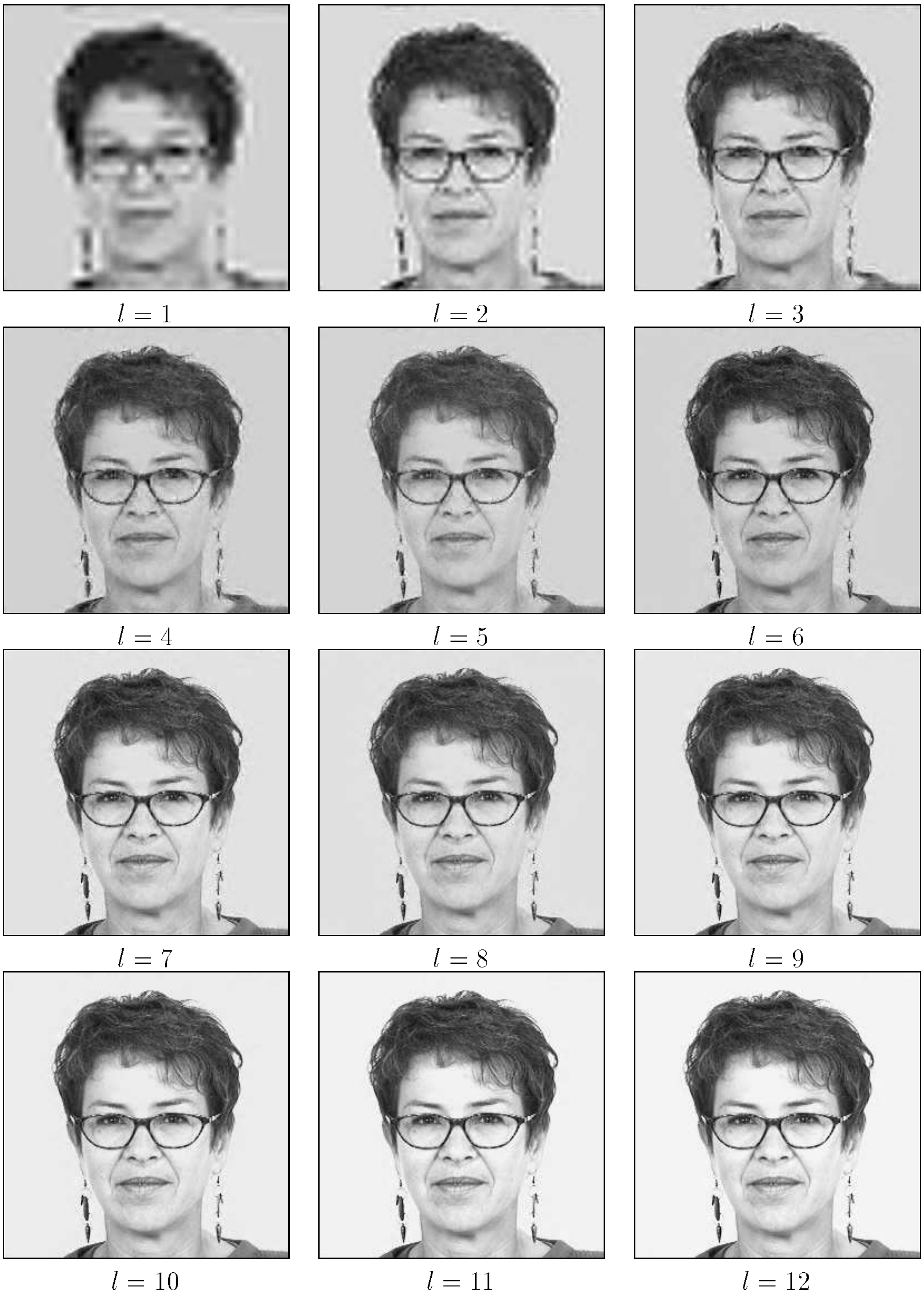
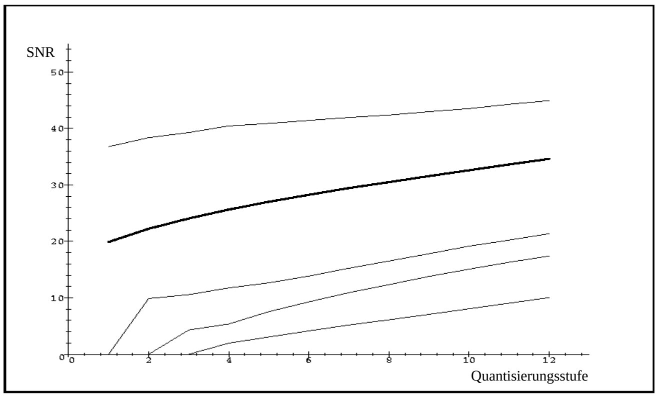
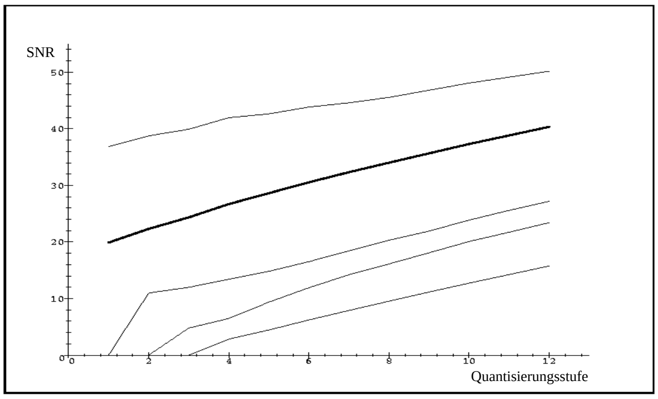


Abbildung 4.12: „Aenne“ progressiv quantisiert in 12 Schritten, 100 Codevektoren in jedem Schritt (Versuch (b)).



$N = 50$



$N = 100$

Abbildung 4.13: SNR in Abhängigkeit von der Quantisierungsstufe. Die breitere Linie zeigt den Verlauf des Gesamt–SNRs, die anderen sind die Kurven für die einzelnen Bänder: ganz oben die des LL–Bandes, dann nach unten 3., 2. und 1. Ebene. Dargestellt sind die Kurven für $N = 50$ Codevektoren (Versuch (a)) und $N = 100$ Codevektoren (Versuch (b)).

4.5 Variation der Testbilder

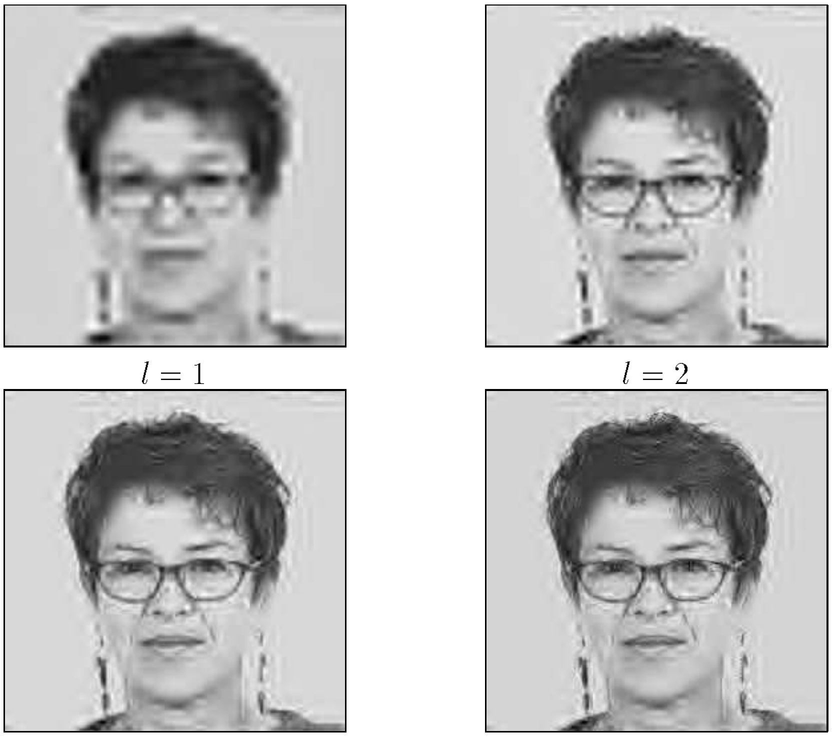
Dieser Versuch soll mit einer progressiven Codierung die Wirkung der Trainingsmenge auf die Approximationsgüte zeigen, das heißt wir gehen, wie im ersten Experiment, auf die Bildklasse ein.

Dazu codieren wir vier Testbilder progressiv mit denselben Codebüchern, die mit einer Menge von Gesichtsbildern trainiert werden. Die vier Bilder sind „Aenne", „Lena", „Mann" und „Airport". Davon ist „Aenne" als einziges in der Trainingsmenge enthalten. „Mann" ist unter denselben Bedingungen aufgenommen wie „Aenne", „Lena" ist ein Gesichtsbild mit einem anderen Hintergrund.

$L = 4$
$\mathcal{T}_l = \{20 \text{ Gesichtsbilder}\}$ für $l = 1, \ldots, 4$
$N(l) = 100$ für $l = 1, \ldots, 4$
$g(1) = (1,0;\ 0,0;\ 0,0;\ 0,0)$ $g(2) = (0,0;\ 1,0;\ 0,0;\ 0,0)$ $g(3) = (0,0;\ 0,0;\ 1,0;\ 0,0)$ $g(4) = (0,0;\ 0,0;\ 0,0;\ 1,0)$
$\mathrm{DZ}(l) = 0,00\%$ für $l = 1, \ldots, 4$

Die Abb. 4.14, 4.15, 4.16 und 4.17 zeigen die Approximationen in den einzelnen Schritten für die Bilder „Aenne", „Lena", „Mann" und „Airport".

Die Ergebnisse bestätigen, daß Codebücher für eine Klasse von Bildern sinnvoll sind, denn die Rekonstruktionen der drei Gesichtsbilder sind sehr gut. „Airport" hingegen ist praktisch nicht zu erkennen. Obwohl „Mann" nicht in der Trainingsmenge enthalten war, ist die Qualität der Rekonstruktion so gut wie die bei „Aenne". Das Bild „Lena" ist etwas verzerrt, denn die Bilder in der Trainingsmenge haben alle einen weißen Hintergrund, was hier nicht der Fall ist. Dennoch ist auch dieses Bild sehr gut zu erkennen.

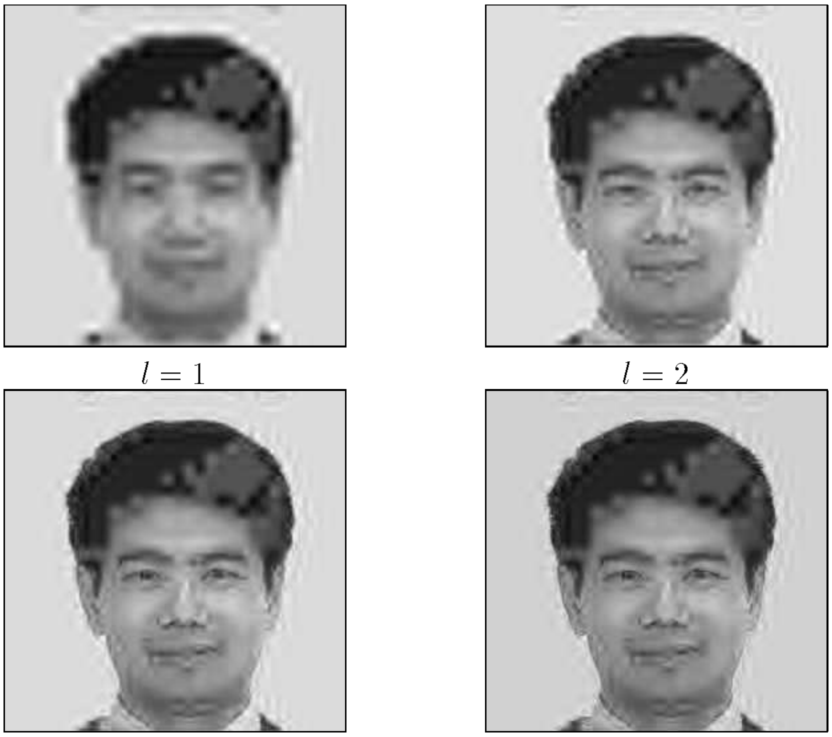


$l = 1$ $l = 2$

$l = 3$ $l = 4$

Abbildung 4.14: „Aenne“

Abbildung 4.15: „Lena“



$l = 1$ $l = 2$

$l = 3$ $l = 4$

Abbildung 4.16: „Mann“

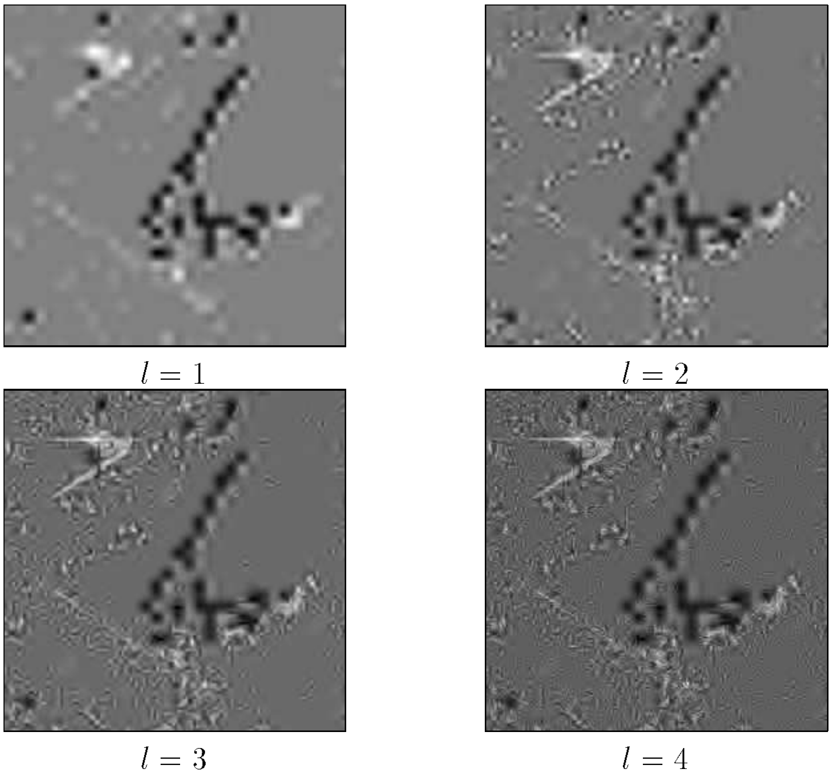


Abbildung 4.17: „Airport“

Kapitel 5

Vergleiche mit anderen progressiven Codierverfahren

Da unsere Arbeit die letzte von drei Diplomarbeiten an unserem Institut ist, die sich mit der progressiven Codierung mit Hilfe von Wavelets beschäftigt haben, wollen wir einen kurzen Vergleich der verschiedenen Methoden anschließen und damit auch aufzeigen, wie unterschiedlich man die Wavelet–Transformation und die progressive Codierung je nach Aufgabenstellung nutzen kann. In der einen Arbeit wird die Codierung mit lokalen Orientierungspyramiden (LOPC) untersucht [San97], und in der anderen geht es um die Codierung mit Waveletnetzen (WNC) [Pel97].

5.1 Progression mit lokalen Orientierungspyramiden

Bei der LOPC wird das Bild, das codiert werden soll, mit Hilfe einer Wavelet–Transformation in Subbänder zerlegt. Von den Transformations–Koeffizienten werden jeweils die lokale Amplitude und die Orientierung gespeichert, und die lokale Phase wird vernachlässigt. Damit ist das Verfahren verlustbehaftet. Lokale Amplitude und Orientierung werden mit einem Vektorquantisierer enquantisiert. Diese Quantisierung ist rotationsinvariant und wahrnehmungsgerecht.

Mit diesem Verfahren kann man in einer Bilddatenbank nach einem Bild suchen, indem man einen Bildausschnitt vorgibt, und das Verfahren liefert mit Hilfe einer Auffälligkeitskarte die Bildstellen, die dem Ausschnitt am ähnlichsten sind. Gibt man zum Beispiel ein Auge vor, so wird von dem zu decodierenden Bild zuerst der Teil übertragen, der dem Auge am ähnlichsten sieht und nach und nach in absteigender Reihenfolge der Rest des Bildes.

5.2 Progression durch Waveletnetze

Bei der WNC gibt man zur Codierung eines Bildes Punkte im Bild an, an denen Wavelets angesetzt werden sollen. Diese Wavelets haben als Parameter zwei Skalen, Orientierung, Position und Gewicht, und diese Parameter werden dann optimiert und an das Bild angepaßt. Für jeden Progressionsschritt werden Wavelets hinzugenommen, die auch wieder optimiert werden.

Decodiert man ein Bild mit diesem Verfahren, so werden zuerst einige Stellen im Bild klar, und nach und nach kommt der Rest des Bildes hinzu.

5.3 Gemeinsamkeiten und Unterschiede der Verfahren

Einen kurzen Überblick soll Tabelle 5.1 bieten.

	Progressive VQ	**Lokale Orien–tierungspyramide**	**Waveletnetze**
Repräs.	feste Transformation, Daubechies–Wavelets	feste Transformation, lokale Orientierung, verlustbehaftet	frei positionierte, skalierte, rotierte & gewichtete Wavelets
Approx.–güte	bestimmt durch Codevektoranzahl und Gewichtung im Vektorquantisierer (Lloyd–Algorithmus)	bestimmt durch Quantisierungsstufenanzahl im VQ (starre Quantisierungsvorschrift)	bestimmt durch Zahl der Wavelets & Quantisierungsfeinheit der Waveletparameter
Progr.–strategie	allgemeine Progression, globale Verbesserung der Approximation durch Requantisierung des Fehlers	Progression nach Ort, lokale Verbesserung der Approximation durch Rekonstruktion von Sub–Pyramiden	Progr. nach Ort, lokale Verbesserung der Approx. durch Hinzufügen und Optimieren von Waveletgruppen

Tabelle 5.1: Vergleich der drei Verfahren.

Zunächst einmal grenzen wir die Low–level–Repräsentationen der einzelnen Methoden gegeneinander ab. In unserer Arbeit und bei der LOPC wird eine feste Wavelet–Transformation mit Daubechies–Wavelets durchgeführt. Wir erstellen dann aus der entstehenden Oktav–Pyramide die Vektoren des Bildes. Bei der LOPC werden nur lokale Amplitude und Orientierung gespeichert, das heißt diese

Art der Repräsentation ist auf jeden Fall verlustbehaftet. Bei der WNC hingegen werden für jedes zu codierende Bild Stellen im Bild festgelegt, an denen frei skalierte, rotierte und gewichtete Wavelets angesetzt werden.

Die Approximationsgüte wird bei unserer Arbeit durch die Anzahl der Codevektoren in jedem Codebuch in jedem Schritt bestimmt. Desweiteren fließen Gewichtung und Totzone bei der Codebuch–Erstellung in die Güte mit ein. Bei der LOPC kann durch eine gesteigerte Quantisierungsstufenanzahl im Vektorquantisierer die Approximationsgüte verbessert werden. Die Quantisierungsvorschrift in diesem Verfahren ist festgelegt, während sie bei unserem durch Training gewonnen wird. Bei der WNC kann die Anzahl der Wavelets variiert werden, und zusammen mit der Quantisierungsfeinheit der Waveletparameter wird die Approximationsgüte bestimmt.

Die Progressionsstrategie ist bei LOPC und WNC lokal angelegt, das heißt bei der Decodierung der Bilder erhält man an einzelnen Stellen im Bild die Verbesserungen. Die LOPC macht dies, indem in jedem Schritt weitere Sub–Pyramiden rekonstruiert werden, bei der WNC werden einzelne Waveletgruppen hinzugefügt. Unser Verfahren hingegen liefert eine globale Verbesserung der Approximation, das heißt in jedem Schritt wird das gesamte Bild besser.

Kapitel 6

Diskussion und Ausblick

In dieser Diplomarbeit haben wir eine Methode zur progressiven Bildkompression vorgestellt, wie man sie zum Beispiel nutzen kann, um eine Bilddatenbank zu durchsuchen. Die Methode ist dann von Vorteil, wenn die Übertragung der einzelnen Schritte insgesamt lange genug dauert, so daß sich ein Abbruch der Übertragung lohnt, sobald man eine gestellte Aufgabe gelöst hat.

6.1 Diskussion

Bei unserer Methode wird das Bild mit Hilfe von Wavelet–Transformationen in Subbänder zerlegt. Daraus entsteht eine Oktav–Pyramide, die in Vektoren aufgeteilt wird. Diese Vektoren werden dann wahrnehmungsgerecht quantisiert. Hierzu wird ein gewichtetes Fehlermaß erzeugt, das die Empfindlichkeitsunterschiede des Menschlichen Visuellen Systems innerhalb der einzelnen Subbänder berücksichtigt. Bei der Erstellung des Quantisierers wird eine Totzone eingebracht, so daß weniger relevante Grauwerte nicht berücksichtigt werden.

Nach einer ersten Quantisierung des Bildes wird eine progressive Codierung durchgeführt, indem jeweils der Fehler, der in der vorhergegangenen Progressionsstufe entsteht, nachquantisiert wird. Auf diese Art wird der Approximationsfehler mit jeder Progressionsstufe minimiert, und die Rekonstruktion kann damit beliebig weit dem Original angenähert werden. Durch die Progression wird die Komplexität großer Codebücher vermieden. Da mehrere Schritte durchgeführt werden, muß das Codebuch in jedem Schritt nur relativ klein sein. Damit liefert der Lloyd–Algorithmus recht schnell ein gutes Codebuch, und die Enquantisierung insgesamt kann verkürzt werden.

Als erstes haben wir untersucht, wie sich die Menge der Bilder, mit der die Codebücher trainiert werden, auf die Güte eines komprimierten Bildes auswirkt.

Die Überlegung war, ob es sinnvoll ist, ein Codebuch für eine Klasse von Bildern zu entwerfen als Kompromiß zwischen den beiden Extremlösungen eines speziellen Codebuchs für jedes einzelne Bild auf der einen und eines Codebuchs für alle Arten von Bildinhalten auf der anderen Seite. Zwar ist ein Codebuch für eine Bildklasse größer als eines für ein Bild (bei gleicher Qualität der Approximation), aber es ist im allgemeinen kleiner als ein Codebuch, das alle möglichen Bildinhalte codieren soll, da es die Statistik der Bildklasse ausnutzen kann. Mit einem solchen Bildklassen–Codebuch können alle Bilder einer Datenbank encodiert werden, und wenn man diese durchsuchen will, so muß man nur einmal das Codebuch übertragen und kann damit jedes Bild aus der Datenbank decodieren. Würde man jedes Bild mit einem genau dafür entwickelten Codebuch encodieren, so müßte auch für jedes Bild, das man sich ansehen will, das entsprechende Codebuch erst übertragen werden, was viel zu lange dauern würde. Der Versuch hat gezeigt, daß unsere Überlegungen richtig waren. Mit Codebüchern, die auf die Bildklasse abgestimmt sind, können Bilder dieser Klasse in angemessener Qualität encodiert werden.

Die Größe eines Codebuchs haben wir im zweiten Experiment untersucht. Je größer ein Codebuch gemacht wird, desto genauer kann es das zu codierende Bild approximieren. Andererseits verbrauchen größere Codebücher natürlich auch mehr Speicherplatz und haben höhere Übertragungskosten als kleinere. Wenn man ein Bild progressiv aus einer Datenbank übertragen will, so soll die erste Approximation möglichst schnell beim Empfänger und möglichst genau erkennbar sein. Das heißt sie soll sehr klein encodiert und dennoch sehr genau sein. Der Versuch zeigte uns, daß man zwischen diesen beiden Forderungen genau abwägen muß. Wählt man die Codebücher für die einzelnen Progressionsschritte zu klein, so braucht man viele Übertragungsschritte, bis die endgültige Approximation eine akzeptable Güte hat. Sind die einzelnen Codebücher zu groß, so haben die jeweiligen Übertragungsschritte zu hohe Kosten.

Das dritte Experiment zielte auf den Einsatz der Gewichtung. Wir haben überlegt, daß man mit der Gewichtung erst die wichtigeren Bänder hervorheben und diese dann noch nachquantisieren könnte, während die unwichtigeren quantisiert werden. Die Ergebnisse dieses Versuchs waren allerdings nicht entsprechend; die Bilder zeigten nicht die gewünschte Verbesserung in den einzelnen Schritten.

Im vierten Experiment haben wir das zweite und das dritte miteinander verbunden, indem wir eine Nachquantisierung der Bänder durchgeführt haben für verschiedene Anzahlen von Codevektoren in jedem Schritt. Hierbei hat sich gezeigt, daß mit mehr Codevektoren der SNR zwar in jedem Schritt höher wird, mit ihm aber auch die Coderate ansteigt. Subjektiv sieht man diese SNR–Verbesserung allerdings auf den Bildern nicht, was bedeutet, daß man mit der kleineren Anzahl von Codevektoren in jedem Schritt auskommen kann.

Das letzte Experiment zeigt noch einmal von einer anderen Seite, daß sich Codebücher für eine Klasse von Bildern gut für die Codierung eignen. Wir haben vier verschiedene Bilder progressiv mit jeweils den gleichen Codebüchern encodiert, die mit einer Menge von Gesichtsbildern trainiert wurden. Drei dieser Bilder waren Gesichtsbilder, von denen eines in der Trainingsmenge enthalten war. Das vierte, das kein Gesichtsbild war, war nach der Rekonstruktion praktisch nicht zu erkennen. Die anderen drei zeigten gute Ergebnisse.

Zusammenfassend kann man sagen, daß die progressive Codierung vor allem dann sinnvoll ist, wenn eine Bilddatenbank durchsucht werden soll, deren Bilder in bestimmte Klassen einteilbar sind und in den einzelnen Progressionsschritten mit Hilfe von Codebüchern encodiert wurden, die an die entsprechende Bildklasse angepaßt sind. Dann muß man zum Durchsuchen der Datenbank nur einmal die Codebücher übertragen, die für die einzelnen Schritte notwendig sind, und damit kann man dann alle Bilder der Datenbank decodieren. Mit dieser Art der Bildkompression erreicht man sehr kleine Coderaten, wenn man die Übertragung nach Lösung der Aufgabe abbricht.

6.2 Ausblick

Weitere Untersuchungen der progressiven Kompression sollten sich auf die bessere Abstimmung der einzelnen Parameter konzentrieren. Hierzu gäbe es noch viele Experimente, die man durchführen könnte. Zum Beispiel haben wir die Auswirkung der Totzone auf die Approximationsgüte für diese Arbeit vernachlässigt, und die Wahrnehmung von Quantisierungsfehlern [WYSV97] könnte man weiter berücksichtigen.

Interessant wäre auch die Untersuchung, die im ersten Experiment angesprochen wurde, nämlich wie sehr sich das Vorhandensein beziehungsweise Nichtvorhandensein eines Bildes in der Trainingsmenge auf die Codierung dieses Bildes auswirkt, wenn die Trainingsmenge insgesamt sehr groß ist.

Außerdem wäre eine Automatisierung der Einstellungen wünschenswert, das heißt die Randgrößen wie endgültige Approximationsgüte und Anzahl der Progressionsschritte sollen vorgegeben werden, und daraus sollen sich die anderen Parameter ergeben.

Literaturverzeichnis

[BF92] C. F. Barnes and R. L. Frost. Residual vector quantizers with jointly optimized code books. *Advances in Electronics and Electron Physics*, 84, 1992.

[Dau88] I. Daubechies. Orthonormal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, 41:909–996, 1988.

[GG92] A. Gersho and R. M. Gray. *Vector Quantization and Signal Compression.* Kluwer Academic Publishers, 1992.

[GW92] R. C. Gonzalez and R. E. Woods. *Digital Image Processing.* Addison Wesley, 1992.

[Jai89] A. K. Jain. *Fundamentals of Digital Image Processing.* Prentice Hall, 1989.

[Ohm95] J.-R. Ohm. *Digitale Bildcodierung.* Springer–Verlag, 1995.

[Pal93] M. Pall. Einsatz von Subbandkodierung und Wavelets bei der Kompression von Bilddaten. Diplomarbeit, Uni Karlsruhe, 1993.

[Pel97] A. Pelc. Aufgabenbezogen progressive Repräsentation von Bildern durch Waveletnetze. Diplomarbeit, Lehrstuhl für Kognitive Systeme, Christian–Albrechts–Universität zu Kiel, 1997.

[San97] M. Sandgaard. Attentionsgesteuerte progressive Übertragung von Bildern. Diplomarbeit, Lehrstuhl für Kognitive Systeme, Christian–Albrechts–Universität zu Kiel, 1997.

[VM91] Brani Vidaković and Peter Müller. Wavelets for kids, a tutorial introduction. Technical report, Institute of Statistics and Decision Sciences, Duke University Durham, NC 27708-0251, 1991.

[WYSV97] A. B. Watson, G. Y. Yang, J. A. Solomon, and J. Villasenor. Visibility of wavelet quantization noise. *IEEE Transactions on Image Processing*, 6(8), 1997.

Danksagung

Zunächst möchte ich Herrn Prof. Dr. Sommer danken für das Ermöglichen dieser Diplomarbeit an seinem Lehrstuhl. Desweiteren bedanke ich mich bei meinem Betreuer Dipl.–Ing. Udo Mahlmeister, der mir geduldig meine Fragen beantwortet und mir hilfreich zur Seite gestanden hat. Außerdem danke ich meinen Eltern und Niels für das Korrekturlesen meiner Arbeit und Anja, Achim und Raimund für die Hilfe beim Lösen meiner LaTeX–Probleme. Kay danke ich für seine Geduld, das Ertragen meiner Launen und für seine Aufmunterungen.

Nicht zuletzt möchte ich mich bei meinen Eltern dafür bedanken, daß sie mir dieses Studium ermöglicht und so geduldig auf seinen Abschluß gewartet haben.

Erklärung:

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Kiel, den 25. Mai 1998