Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme
Rheinisch-Westfälische Technische Hochschule Aachen
Prof. Dr.-Ing. T. G. Noll

Diplomarbeit

Automatisierte Analyse von Musikarchiven mittels digitaler Signalprozessoren

Michael Haller

Matr.-Nr. 242280

Betreuung:
Univ.-Prof. Dr.-Ing. T. G. Noll
Dr.-Ing. H. Blume

Ich versichere, dass die vorliegende Arbeit – bis auf die offizielle Betreuung durch den Lehrstuhl – ohne fremde Hilfe von mir erstellt wurde. Die verwendete Literatur ist im Literaturverzeichnis vollständig angegeben.

Aachen, den 9. Januar 2008

(Michael Haller)

Inhaltsverzeichnis

1. EINLEITUNG

Die Verfügbarkeit und Anzahl an Musiktiteln steigt in der heutigen Zeit immer weiter. Insbesondere durch schnelle Internetzugänge und die steigende Speicherkapazität hat man leicht Zugriff auf mehrere tausende Musikstücke. Doch mit steigendem Umfang der Musikarchive wird es zunehmend schwerer, diese zu verwalten oder nach neuen Kriterien zu sortieren.

Für die Handhabung solch großer Musikarchive ist die automatisierte Analyse und Klassifikation der digitalen Musikdaten ein wertvolles Hilfsmittel. So lassen sich die Musikarchive beispielsweise nach verschieden feinen Unterteilungen von Genres sortieren, nach Liedern einer bestimmten Stilrichtung durchsuchen oder nach sonstigen subjektiven Kategorien einteilen.

Künftig werden solche Techniken nicht nur im professionellen Umfeld sondern auch in zunehmendem Umfang für den normalen Endverbraucher eingesetzt werden. Hier sind insbesondere mobile Geräte ein großes und wichtiges Einsatzfeld.

Aus den Musikstücken müssen verschiedene akustische Merkmale (Features[1]) extrahiert werden, auf deren Grundlage ein Klassifikationsalgorithmus arbeiten kann.

Das Extrahieren der Merkmale ist ein rechenintensiver Prozess, der attraktive kurze Laufzeiten aufweisen muss, damit die Gesamtfunktionalität vom Anwender akzeptiert wird.

Bisher existieren solche Verfahren nur ansatzweise für Desktop/Server-Architekturen. Mobilgeräte hingegen unterliegen starken Einschränkungen was den Energieverbrauch, die Taktfrequenz und damit die maximale Rechenleistung angeht. Hier ist daher ein besonderes Augenmerk auf den erforderlichen Rechenaufwand respektive die Laufzeiten zu legen.

Es existiert eine sehr umfangreiche Anzahl an vorgeschlagenen Features, die sich aus Musikstücken extrahieren lassen. Diese unterscheiden sich sehr stark hinsichtlich des Rechenaufwandes und ihrer Aussagekraft bezüglich der Differenzierbarkeit von Musikstücken. Abgesehen von einigen Experimenten, die auf vorgegebenen Feature-Sätzen arbeiten, existiert wenig grundlegende Kenntnis über die Eignung der diversen Features oder der erzielbaren Klassifikationsgüten. Insbesondere zur Auswahl geeigneter Feature-Sätze unter Berücksichtigung von Laufzeit und Klassifikationsgüte gibt es bislang keine Arbeiten.

[1] Im Folgenden wird der Begriff Feature verwendet, da er die in der Literatur gebräuchliche Bezeichnung ist.

Im Rahmen dieser Arbeit wurden die Extraktionsalgorithmen zahlreicher Features sowie Klassifikationsalgorithmen auf einem PC implementiert, verifiziert und ihre Laufzeiten und die mit ihnen erzielbaren Klassifikationsgüten ermittelt. Anschließend wurden diese Algorithmen auf die Architektur eines digitalen Signalprozessors (STn8810) portiert und dort wieder auf ihre Eigenschaften untersucht.

Zur Bestimmung attraktiver Feature-Kombinationen wurde ein statistischer Test durchgeführt, der signifikante Parameter identifiziert.

Abschließend wurden häufige Befehlskombinationen identifiziert, die sich für eine mögliche Optimierung anbieten.

Als Mobilprozessor wird im Rahmen dieser Arbeit die Nomadik-Plattform STn8810 der Firma STMicroelectronics eingesetzt. Neben einem 32 bit ARM926EJ S-Prozessorkern und mehreren On-Chip-Speichern sind zwei DSP-basierte, verlustleistungsoptimierte Akzeleratoren für die Audio- und Videoverarbeitung integriert, die als Smart Audio Accelerator (SAA) bzw. Smart Video Accelerator (SVA) bezeichnet werden. Der SAA ist ein 24 bit VLIW Prozessor, der speziell für die autonome Abarbeitung von Audioanwendungen auf der Nomadik-Plattform integriert ist und eine parallele Ausführung von maximal acht Operationen unterstützt.

Im zweiten Kapitel wird der Ablauf der Feature-Extraktion dargestellt und die verwendeten Features vorgestellt. Anschließend werden die im Rahmen dieser Arbeit behandelten Klassifikatoren vorgestellt.

Im dritten Kapitel wird die verwendete Hardwareplattform kurz beschrieben.

Das vierte Kapitel befasst sich mit der Durchführung von Klassifikationsexperimenten. Zunächst werden die Ergebnisse aus der Literatur vorgestellt und mit eigenen Experimenten verglichen. Anschließend wird ein Statistiktest vorgestellt und durchgeführt, der aussagekräftige Features ermittelt.

Im fünften Kapitel wird die Laufzeit der einzelnen Funktionen untersucht und auf Möglichkeiten zur programmtechnischen Optimierung eingegangen.

Das sechste Kapitel befasst sich schließlich mit Optimierungsansätzen auf Hardware-Ebene.

2. WAHRNEHMUNGSBASIERTE MUSIK-KLASSIFIKATION

2.1. Feature-Extraktion

Die im Rahmen dieser Arbeit verwendeten Musikstücke liegen als Folge von digitalen Abtastwerten mit einer Wortbreite von 16 Bit vor. Die Abtastrate der betrachteten Musikstücke liegt bei 22050 Hz.

Die verwendeten Features werden jeweils blockweise berechnet. Dazu wird das Musikstück in einzelne, nicht überlappende Fenster zu je einer festen Anzahl Abtastwerte unterteilt. Für diese Arbeit betrug die Fensterbreite 512 Samples, was bei der gegebenen Abtastrate 43 Fenstern pro Sekunde entspricht.

Pro Fenster werden jeweils alle Features extrahiert, die je nach Definition der einzelnen Features aus einem Wert oder einem Vektor von Werten bestehen.

Zur Reduktion der Datenmenge wird jede Folge der Feature-Werte jeweils zusammengefasst zu Mittelwert und Varianz.

In Abbildung 2-1 ist der Vorgang der Extraktion eines Features aus einem kurzen Musikausschnitt dargestellt. Im oberen Abschnitt ist das Musikstück dargestellt, es wird im ersten Schritt zunächst in einzelne Fenster unterteilt. Aus jedem der Fenster wird das Feature extrahiert, es ergibt sich eine Folge von Feature-Werten. Diese Folge wird zusammengefasst zu Mittelwert und Varianz.

Im Rahmen dieser Arbeit wurden 19 Features implementiert, aus denen sich insgesamt ein Feature-Vektor mit 74 Elementen ergab (z.B. besteht das Feature „MFCC“ aus 13 Elementen von denen jeweils Mittelwert und Varianz ermittelt wurden, womit sich 26 Elemente im Feature-Vektor ergeben).

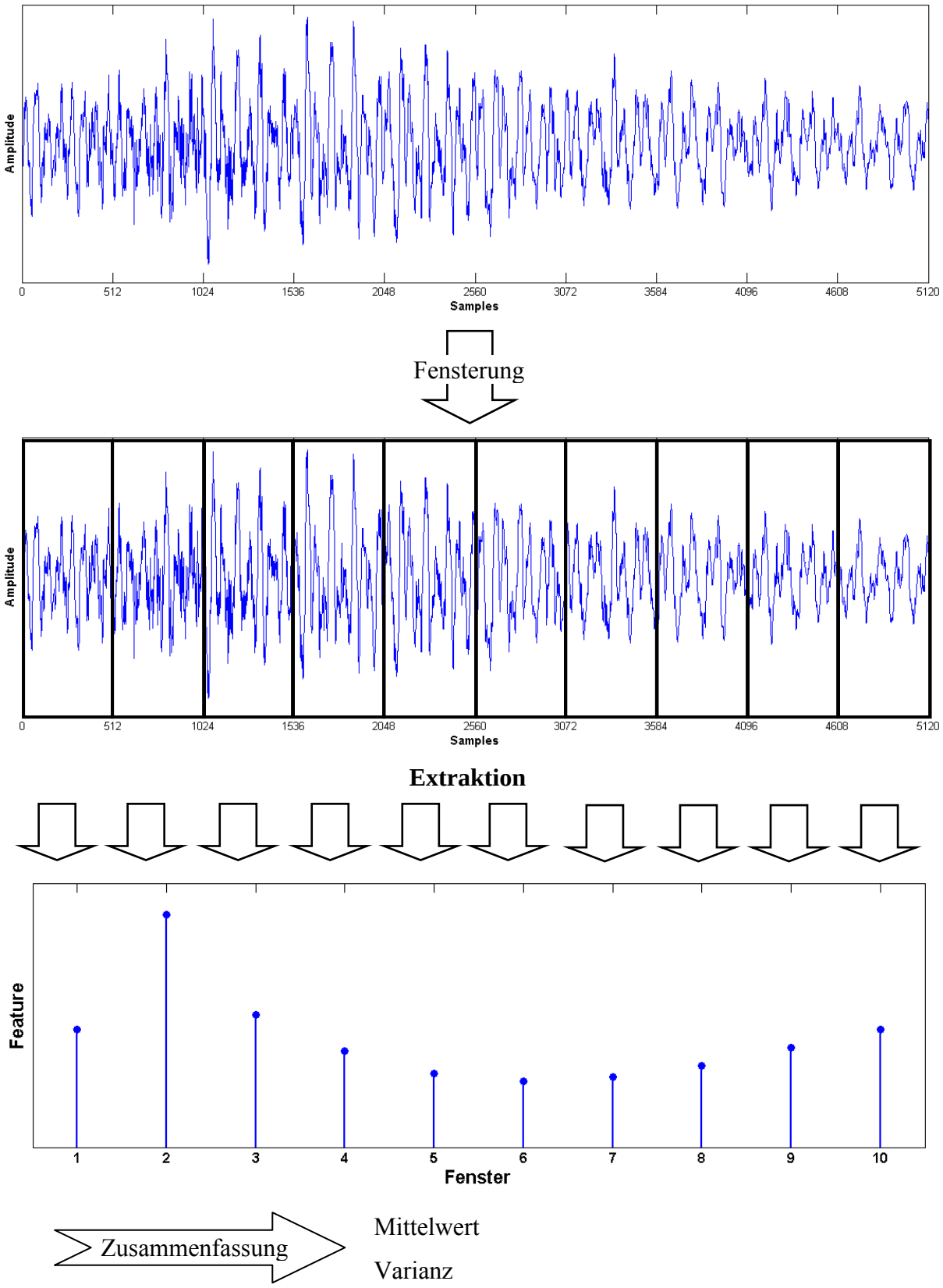


Abbildung 2-1
Exemplarische Extraktion eines Features aus 10 Abschnitten eines Musikstücks.

2.2. Verwendete Features

Im Folgenden sollen die im Rahmen dieser Arbeit verwendeten Features vorgestellt und mathematisch definiert werden. Die Beschreibung der Features orientiert sich an der Darstellung in [46].

2.2.1. Notation

Für die mathematische Beschreibung der Features wird folgende Notation verwendet:

Bezeichnung	Notation
Gesamtzahl Samplewerte innerhalb eines Musikstücks	N_{total}
Länge eines Zeitfensters	N
Gesamtzahl an Fenstern im Musikstück	$L_{Total} = \left\lceil \frac{N_{total}}{N} \right\rceil$
diskretes Zeitsignal	x(n), n ∈ [0, N_{total} -1]
Anzahl der Frequenzwerte	K
diskrete Spektralkomponente	X(k), k ∈ [0, K]
Amplitude der Spektralkomponente	A(k) = |X(k)|
Amplitude der Spektralkomponente in einem bestimmten Zeitfenster	$A^{(l)}(k)$, $l \in [1, L_{total}]$
Diskrete Fouriertransformation	$X(k) = \sum_{n=0}^{N-1} x(n) \cdot W_N^{kn}$, $W_N = e^{-j\frac{2\pi}{N}}$, $K = N$
Autokorrelationsfunktion	$r_{xx}(i) = \sum_{n=0}^{N-i-1} x(n) \cdot x(n+i)$

2.2.2. Features im Zeitbereich

Diese Features lassen sich direkt aus den zeitlichen Abtastwerten des Musikstückes ermitteln.

2.2.2.1. Zero Crossing Rate

Die Zero Crossing Rate gibt die Anzahl der Vorzeichenwechsel in einem Signal an. Sie ist ein Maß für den hochfrequenten Anteil des Signals.

$$r_{zero-crossing} = \frac{1}{2(N-1)} \cdot \sum_{n=0}^{N-2} \left| \operatorname{sgn} x(n+1) - \operatorname{sgn} x(n) \right|$$

2.2.2.2. Root Mean Square

Die Root Mean Square (RMS) ist ein normiertes Maß für die Signalenergie in einem Zeitfenster.

$$x_{RMS} = \sqrt{\frac{1}{N} \sum_{n=0}^{N-1} x(n)^2}$$

2.2.2.3. Low Energy Windows

Das Low Energy Windows – Feature wird über einen Zeitraum von mehreren Zeitfenstern ermittelt. Dieser Zeitraum wird als Analysezeitraum bezeichnet. Das Feature gibt den Anteil der Zeitfester innerhalb eines Analysezeitraums an, in denen der RMS niedriger ist als sein Durchschnitt in diesem Zeitraum. N_a sei die Anzahl der Zeitfenster innerhalb eines Analysezeitraums, $x_{RMS}(n)$ sei der RMS Wert des Zeitfensters n.

$$r_{le} = \frac{1}{N_a} \sum_{n=0}^{N_a-1} u\left(\mu_{RMS} - x_{RMS}(n)\right)$$

Wobei $\mu_{RMS} = \frac{1}{N_a} \sum_{n=0}^{N_a-1} x_{RMS}(n)$ der Mittelwert des RMS im Analysezeitraum ist

und $u(x) = \begin{cases} 1 & \text{für } x > 0 \\ 0 & \text{für } x \le 0 \end{cases}$

2.2.2.4. Sum Of Correlated Components

Die Sum of Correlated Components gibt ein Maß für die Korrelation innerhalb eines Zeitfensters an.

Weißes Rauschen hätte z.B. einen Wert von $S = 0$, da keinerlei Korrelation für Verschiebungen $i > 0$ vorliegt.

$$S = \sum_{i=1}^{N-1} \frac{\left| r_{xx}(i) \right|}{\left| r_{xx}(0) \right|}$$

2.2.2.5. Relative Periodicity Amplitude Peaks

Die Relative Periodicity Amplitude Peaks sind definiert als die normalisierten Amplituden der ersten beiden Maxima der Autokorrelationsfunktion.

$$r_1 = \frac{r_{xx}(n_1)}{\sum_{i=0}^{N-1} r_{xx}(i)} \quad \text{und} \quad r_2 = \frac{r_{xx}(n_2)}{\sum_{i=0}^{N-1} r_{xx}(i)}$$

Wobei n_1 und n_2 die Indizes des ersten und zweiten Maximums nach dem absoluten Maximum bei $n = 0$ sind.

2.2.2.6. Ratio Of Second And First Periodicity Peak

Das Verhältnis zwischen dem zweiten und ersten Maximum der Autokorrelationsfunktion beschreibt die Dominanz von schnelleren oder langsameren Takten in einem Zeitfenster.

$$r = \frac{r_2}{r_1}$$

wobei r_1 der kleinste Index > 0 ist, für den $r_{xx}(n)$ ein Maximum aufweist. $r_2 > r_1$ ist das nächste Maximum der Autokorrelationsfunktion nach r_1.

2.2.3. Features im Frequenzbereich

Diese Features werden aus dem Frequenzbereich des Musikstückes ermittelt.

2.2.3.1. Fast Fourier Transformation (FFT)

Mittels der Fouriertransformation lässt sich ein Zeitsignal in den Frequenzbereich transformieren.

$$X(f) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} x(t) \cdot e^{-j2\pi f t}$$

Die diskrete Fouriertransformation (siehe 2.2.1) ist die diskrete Entsprechung für ein abgetastetes Zeitsignal. Hierbei ist die maximale Frequenzauflösung nach dem Abtasttheorem auf die halbe Abtastrate beschränkt (Nyquist-Frequenz).

$$f_{\max} = \frac{f_{abtast}}{2}$$

Um den immensen Rechenaufwand zu reduzieren, bietet sich die Fast Fourier Transformation (FFT) mittels des Radix-2 Algorithmus an [48]. Dabei wird die Fouriertransformation der Länge $M = 2^x$ nach geraden und ungeraden Indizes in zwei Transformationen der Länge $M_2 = 2^{x-1}$ zerlegt. Als Korrekturoperationen werden M_2 Butterfly-Operationen eingefügt. Dieser Vorgang wird als „Decimation in Time“ bezeichnet. Er wird so oft wiederholt, bis nur noch Fouriertransformationen der Länge zwei übrig bleiben, die mittels der Butterfly-Operationen direkt berechnet werden können.

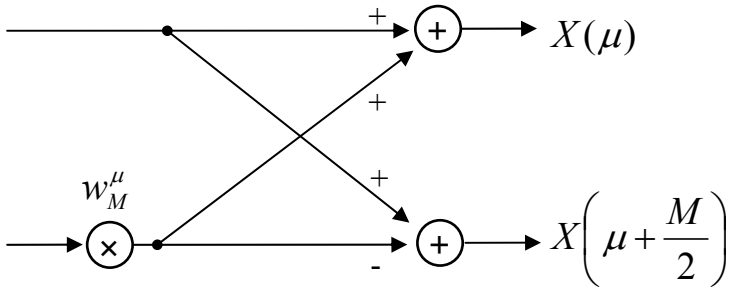


Abbildung 2-2
Butterfly-Operation
Die Butterfly-Operation setzt sich aus einer komplexwertigen Multiplikation und zwei Additionen zusammen. Dabei ist der Faktor der Multiplikation $w_M^\mu = e^{-j\frac{2\pi}{M}\mu}$, wobei M die Länge der FFT und μ abhängig von der Position des Operators in der Berechnung ist.

Für eine FFT der Länge M=8 ergibt sich der in Abbildung 2-3 abgebildete Signalflussgraph.

Die Reihenfolge der Eingangswerte verändert sich bei diesem Vorgang, sie müssen in „bit reversal“-Reihenfolge vorliegen. In dieser Reihenfolge wird die binäre Darstellung der Zahlen der einzelnen Positionen umgekehrt, um so das Sample an dieser Position zu ermitteln. Tabelle 1 zeigt wie die Samples für die Positionen der ersten Spalte in Abbildung 2-3 ermittelt werden.

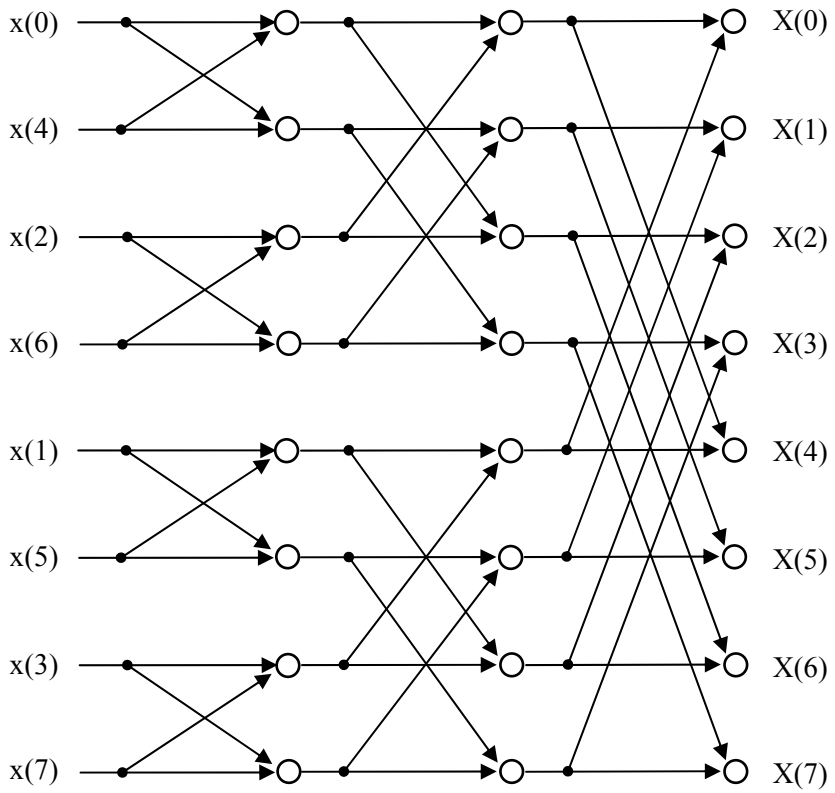


Abbildung 2-3
Signalflussgraph einer „Decimation in Time“ FFT der Länge M=8. Korrekturfaktoren sowie Vorzeichen sind nicht abgebildet.

Position		daraus in „bit reversal“ Reihenfolge berechnetes Sample	
binär	dezimal	binär	dezimal
000	0	000	0
001	1	100	4
010	2	010	2
011	3	110	6
100	4	001	1
101	5	101	5
110	6	011	3
111	7	111	7

Tabelle 1
Berechnung der „bit reversal“ Reihenfolge der Eingangswerte.

Für eine FFT der Länge N ergeben sich bei Verwendung des Radix-2 Algorithmus $\log_2 N$ Schritte, in denen jeweils N Multiplikationen ausgeführt werden. Die Komplexität der FFT beträgt damit $O(N \cdot \log_2 N)$ [36]. Die Komplexität bezüglich der Multiplikation einer Diskreten Fouriertransformation beträgt dagegen $O(N^2)$.

Der Rechenaufwand der in dieser Arbeit berechneten FFT der Länge 512 reduziert sich gegenüber der direkt berechneten Fouriertransformation somit auf knapp 2%.

2.2.3.2. Spectral Flux

Der Spectral Flux ist definiert als die quadrierte Differenz zwischen den Betragsspektren zweier aufeinander folgender Fenster. Er ist ein Maß für die Änderungsgeschwindigkeit der Spektralkomponente.

$$S_{flux} = \frac{2}{K} \sum_{k=0}^{\frac{K}{2}-1} \left(A^{(l)}(k) - A^{(l-1)}(k) \right)^2$$

2.2.3.3. Spectral Kurtosis

Die Kurtosis bildet ein Maß für die Flachheit einer Verteilung um ihren Mittelwert. Sie wird aus dem 4. Moment berechnet.

$$S_{kurtosis} = \frac{m_4}{\sigma^4}$$

wobei

$$m_4 = \frac{\sum_{k=0}^{\frac{K}{2}-1} \left(k - S_{centroid}\right)^4 \cdot A(k)}{\sum_{k=0}^{\frac{K}{2}-1} A(k)} \;,\; \sigma^2 = \frac{\sum_{k=0}^{\frac{K}{2}-1} \left(k - \mu\right)^2 \cdot A(k)}{\sum_{k=0}^{\frac{K}{2}-1} A(k)} \text{ und } \mu = \frac{1}{K} \sum_{k=0}^{K-1} k$$

2.2.3.4. Spectral Extend

Der Spectral Extend gibt die Frequenz an, bei der sich ein definierter Anteil der Signalenergie unterhalb dieser Frequenz befindet.

In der Literatur ist ein Anteil von $p_{extend} = 85\%$ gebräuchlich [22].

$$\sum_{k=0}^{k_{extend}} A^2(k) = p_{extend} \cdot \sum_{k=0}^{\frac{K}{2}-1} A^2(k)$$

2.2.3.5. Spectral Rolloff

Der Spectral Rolloff ist ähnlich definiert wie der Spectral Extend, nur dass hier nicht die quadrierte Amplitude des Frequenzsignals einfließt, sondern nur die Amplitude selber:

$$\sum_{k=0}^{k_{rolloff}} A(k) = p_{extend} \cdot \sum_{k=0}^{\frac{K}{2}-1} A(k)$$

2.2.3.6. Spectral Centroid

Der Spectral Centroid ist der Schwerpunkt des Amplitudenspektrums des Signals.

$$S_{centroid} = \frac{\sum_{k=0}^{\frac{K}{2}-1} k \cdot A(k)}{\sum_{k=0}^{\frac{K}{2}-1} A(k)}$$

2.2.3.7. Spectral Bandwidth

Die Spectral Bandwidth ist definiert als die Breite der power transfer function um die Mittenfrequenz.

$$B_{RMS}^2 = \frac{\sum_{k=0}^{\frac{K}{2}-1} \left(k - S_{centroid}\right)^2 A^2(k)}{\sum_{k=0}^{\frac{K}{2}-1} A^2(k)}$$

2.2.3.8. Position of the Main Peaks

Die Positionen der fünf höchsten Maxima des Spektrums beschreiben die Frequenzen der dominantesten Töne in der Musik.

2.2.3.9. Power Spectrum

Das Power Spectrum ist das logarithmierte Quadrat des Amplitudenspektrums.

$$S(k) = 10 \cdot \log_{10}\left(\frac{1}{N} A^2(k)\right)$$

2.2.3.10. Fundamentalfrequenz

Die Fundamentalfrequenz, auch Grundfrequenz genannt, ist die Frequenz, die zusammen mit ihren Vielfachen am meisten zum gesamten Signal beiträgt. Sie entspricht dem Grundton bei Musikinstrumenten, der zusammen mit seinen Obertönen den Klang erzeugt.

Zur Bestimmung der Fundamentalfrequenz wurden viele Methoden entworfen. In dieser Arbeit wird das Verfahren nach [46] und [24] verwendet. Die Grundidee basiert auf der Schätzung der Fundamentalfrequenz durch die Frequenzdifferenzen benachbarter harmonischer Töne.

Zunächst wird das Signal mit einem Hamming-Fenster $w_N(n)$ gefiltert und dann in den Frequenzbereich transformiert

$$Y(k) = FFT\left(x(n) \cdot w_N(n)\right), \quad w_N(n) = 0{,}54 - 0{,}46\cos\left(\frac{2\pi n}{N-1}\right)$$

Von diesem Spektrum werden die j Maxima A_j und ihre Positionen k_j ermittelt. Zur Verbesserung der Genauigkeit werden Korrekturfaktoren für Frequenz und Amplitude berechnet. Es kann gezeigt werden, dass dies die Genauigkeit der Schätzung der Frequenz erhöht [24].

$$cor_j = \frac{0.5\left(\ln(A(k_j - 1) - \ln(A(k_j + 1))\right)}{\ln(A(k_j - 1)) - 2\ln(A(k_j)) + \ln(A(k_j + 1))}$$

Frequenz: $f_j^* = \dfrac{f_s \cdot (k_j + cor_j)}{N}$

Amplitude: $A_j^* = \exp\left(\ln(A(k_j)) - 0{,}25\, cor_j \left(\ln(A(k_j - 1)) - \ln(A(k_j + 1))\right)\right)$

Der nächste Schritt imitiert eine akustische Hörschwelle, um schwache Frequenzanteile zu beseitigen. Dazu wird ein Fenster angelegt, innerhalb dessen alle Amplitudenwerte auf Null

gesetzt werden, die kleiner als ein Schwellwert sind. Der Schwellwert wird zu 0,9 mal dem maximalen Amplitudenwert innerhalb des Fensters gewählt.

Als nächstes wird eine Folge der Differenzen im Frequenzbereich berechnet.

$$\mathbf{f}_d = f_1^* - 0, f_2^* - f_1^*, f_3^* - f_2^*, \ldots$$

Die erste Näherung der Fundamentalfrequenz ist der Mittelwert der Folge: $fund = mean(\mathbf{f}_d)$

2.2.3.11. Mel Frequency Cepstral Coefficients (MFCC)

Generell ist das Cepstrum die Fouriertransformation (oder die DCT) des logarithmierten Amplitudenspektrums. Die hier betrachteten Mel Frequency Cepstral Coefficients sind das Cepstrum, das aus dem Mel-Band anstelle des Frequenzspektrums berechnet wird. Ihre Verwendung in der Sprachverarbeitung ist weit verbreitet.

Die Mel-Skala ist eine psychoakustische Maßeinheit, die die wahrgenommene Tonhöhe beschreibt. Eine als doppelt so hoch wahrgenommene Frequenz erhält den doppelten Wert auf der Mel-Skala. Als Referenzpunkt wurde die Frequenz f = 1000 Hz gewählt, der der Wert 1000 Mel zugewiesen wird. Die Umwandlung von linearen Frequenzwerten f in die Melwerte erfolgt nach einer logarithmischen Funktion: $m = 2595 \log_{10}\left(1 + \frac{f}{700}\right)$

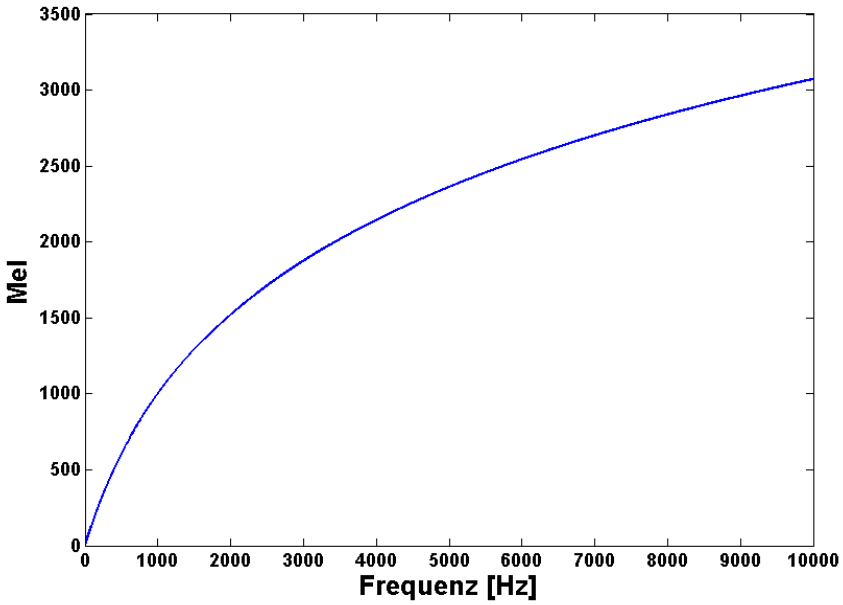


Abbildung 2-4 Mel-Skala

Die Werte des linearen Frequenzbandes werden über Dreiecksfenster zusammengefasst, die auf der Mel Skala gleichverteilt sind (und damit im linearen Frequenzbereich logarithmisch skaliert). Effektiv erfolgt also eine Bandpassfilterung der Spektralkomponente.

$$M(k') = \sum_{k=0}^{\frac{K}{2}-1} A(k) \cdot w_{k'}(k)$$

wobei $w_{k'}(k)$ Dreiecksfenster mit zunehmender Breite für wachsendes k darstellt. Für diese Transformation in das Mel-Band wird oft eine Auflösung von 12 oder 24 Dreiecksfenstern verwendet.

In dieser Arbeit wurde eine Auflösung von 12 Dreiecksfenstern verwendet, wie sie in der Literatur häufig genutzt wird [18][19][20]. Die DCT erzeugt daraus einen Vektor mit 13 Komponenten.

Die MFCC werden somit in folgenden Schritten berechnet:

$$x(n) \xrightarrow{FFT} A(k) \xrightarrow{Mel-Band} M(k') \xrightarrow{\log} \log M(k') \xrightarrow{DCT} MFCC(n)$$

2.2.3.12. Chroma Vector

Der Chroma Vector fasst alle Spektralkomponenten zusammen, die die gleiche Tonhöhe haben. Alle Frequenzen werden zu einer Oktave zusammengefasst, wobei alle spektralen Amplituden des gleichen Vierteltons aufsummiert werden.

$$A_{chroma}\left(\tilde{k}\right) = \sum_{k:p(k)=\tilde{k}} A(k)$$

mit $p(k) = \left\lfloor 24 \log_2\left(\frac{2k}{N}\frac{f_s}{f_1}\right)\right\rfloor \bmod 24$ und $f_1 = 440$ Hz

2.2.3.13. Amplitude of Maximum in Chromagram

Die maximale Amplitude des Chromagrams beschreibt die Stärke eines Tons in verschiedenen Oktavstufen.

$$A_{\max chroma} = \max_{\tilde{k}} A_{chroma}(\tilde{k})$$

2.2.3.14. Pitch Interval Between Two Maximum Peaks Of Chromagram

Das Pitch Intervall beschreibt das Verhältnis zwischen den lautesten Tönen des Musikstücks.

Es ist die Differenz zwischen den Indizes der beiden höchsten Maxima des Chromagrams.

2.3. Merkmalsbasierte Klassifikation

Bei der Klassifikation soll ein unbekanntes Musikstück einer bestimmten Klasse zugeordnet werden. Die Systematik, wie die Klassen eingeteilt werden, wird festgelegt durch die Vorgabe einer Reihe von anderen vorklassifizierten Musikstücken, dem so genannten Trainings-Datensatz.

Die Daten werden den Klassifikatoren in Form von eindimensionalen Vektoren übergeben. In diesen sind die Zahlenwerte, die für die einzelnen Merkmale bestimmt wurden, linear hintereinander angeordnet.

Es wird unterschieden zwischen der Trainingsphase und der Klassifikationsphase. In der Trainingsphase werden dem Klassifikator die Trainingsvektoren präsentiert, aus denen er sich ein Modell der Klassenverteilung erstellt. In der Klassifikationsphase wird dieses Modell auf den zu testenden Vektor angewendet und damit die wahrscheinlichste Klassenzugehörigkeit ermittelt.

Bei der im Rahmen dieser Arbeit betrachteten Aufgabenstellung entsprechen die Klassen den Genres, in die die Musikstücke eingeordnet werden. Im Folgenden werden die Klassen daher als Genres bezeichnet.

2.4. Verwendete Klassifikatoren

Es existiert eine Vielzahl an Klassifikatoren, die sich zur Klassifikation anhand von Merkmalsvektoren eignen. Einige der bekannteren oder in der Literatur verwendeten sind in Tabelle 2 aufgelistet. Im Folgenden werden die Klassifikatoren näher erläutert, die im Rahmen dieser Arbeit betrachtet wurden.

Klassifikator	Beschreibung
Support Vector Maschine	Trennt den Merkmalsraum mittels nichtlinearer Funktionen in Bereiche, die jeweils einer Klasse zugeordnet werden
k-Nearest-Neighbor	Klassifikation durch Analyse von Merkmalsdistanzen im Merkmalsraum
Radial Basis Function Network	Bildet die Wahrscheinlichkeitsdichtefunktion der Merkmale mittels Radialer Basisfunktionen nach. Klassifikation durch Bestimmung des Modells mit der höchsten Wahrscheinlichkeit [30]
Gaussian Mixture Model	Bildet die Wahrscheinlichkeitsdichtefunktion der Merkmale mittels Gaußfunktionen nach. Klassifikation durch Bestimmung des Modells mit der höchsten Wahrscheinlichkeit. [27]
C 4.5	Generiert einen Entscheidungsbaum anhand der Entropie der Merkmale [33] [34]
Multilayer Perceptron	Ein Neuronales Netzwerk, das auf die Trennung der Klassen trainiert wird

Tabelle 2
Im Rahmen der Arbeit betrachtete Klassifikatoren

2.4.1. k-Nearest-Neighbor

Der k-Nearest-Neighbor (KNN) Klassifikator beruht auf der Ähnlichkeit von Feature-Vektoren im Merkmalsraum. Er zählt zu den einfachsten Lernalgorithmen, der keinerlei a-priori Annahmen über die Verteilung der Daten benötigt.

2.4.1.1. Trainingsphase

Der k-Nearest-Neighbor Klassifikationsalgorithmus zählt zu der Klasse der sogenannten *lazy learner* Algorithmen, da die Lernphase nur aus einem Abspeichern der klassifizierten Vektoren des Trainingsdatensatzes besteht.

2.4.1.2. Klassifikationsphase

Ein neuer Datensatz wird klassifiziert, indem die Distanz zu den Trainingsdatensätzen ermittelt wird. Als Resultat gilt das Genre des Trainingsdatensatzes mit der geringsten Distanz. Der k-nearest-Neighbor Algorithmus erweitert dieses Prinzip, indem die Genres der *k* nächstgele-

genen Trainingsdatensätze ermittelt werden und das endgültige Ergebnis aus einer Mehrheitsentscheidung bestimmt wird.

Als Distanzmaß wird meist die euklidische Distanz genutzt, ebenso bieten sich aber auch andere Distanzmaße an (z.B. Manhattan-Distanz). Als problematisch für die Berechnung der Distanzen erweisen sich verschiedene Wertebereiche der einzelnen Komponenten der Vektoren. Um dies zu umgehen, müssen alle Komponenten auf den gleichen Wertebereich normiert werden.

Von entscheidender Bedeutung für Klassifikationsgüte und Laufzeit ist die Wahl von *k*. Bei sehr niedrigen Werten besteht die Gefahr von Missklassifikation durch Rauschen innerhalb der Vektoren. Bei großen Werten dagegen verschwimmen die Grenzen zwischen den einzelnen Genres zunehmend.

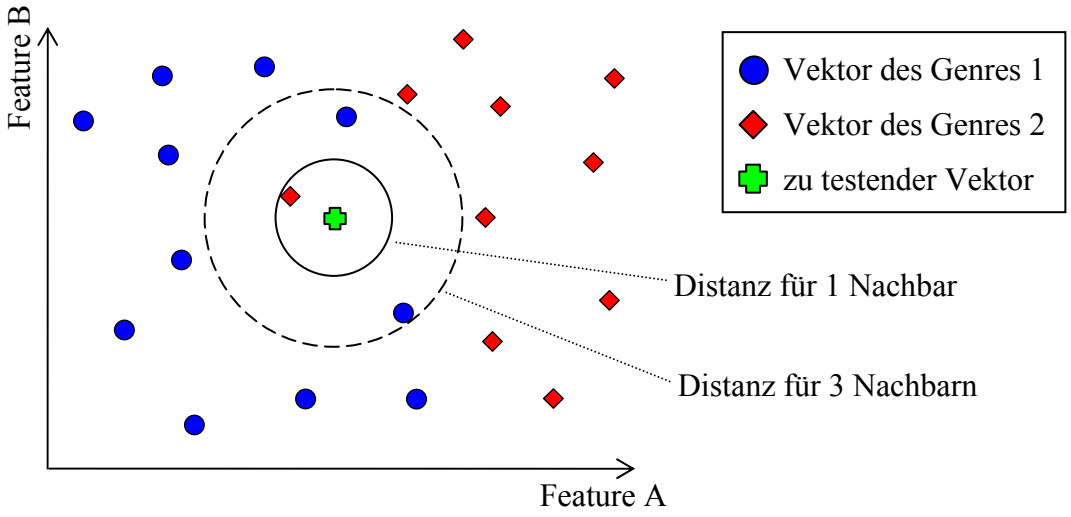


Abbildung 2-5
Beispiel für einen zweidimensionalen
k-NN Klassifikator für die Fälle k = 1 und k = 3

2.4.1.3. Anwendungsbeispiel

Nachfolgend wird ein Beispiel für einen zweidimensionalen KNN Klassifikator für die Fälle $k = 1$ und $k = 3$ betrachtet.

Es sei eine Reihe von n Trainingsvektoren $X_i = (x_{i1}, x_{i2})$ gegeben, die zwei verschiedenen Genres zugeordnet sind. Sie setzen sich aus den Features $x_1 = A$ und $x_2 = B$ zusammen.

Trägt man diese in einen zweidimensionalen Graphen ein, und wählt z.B. die x-Achse für A und die y-Achse für B, so könnte sich die in Abbildung 2-5 dargestellte Verteilung ergeben, wenn die durch blaue Kreise gekennzeichneten Vektoren Elemente des ersten Genres darstel-

len, und die durch rote Rauten gekennzeichneten Punkte entsprechend Vektoren des zweiten Genres darstellen.

Nun soll ein neuer, noch unbekannter Vektor T = (t_1, t_2) klassifiziert werden (hier durch ein grünes Kreuz gekennzeichnet). Es werden für den neuen Vektor die euklidischen Distanzen d_i zu allen Trainingsvektoren ermittelt und nach der Größe geordnet.

$$d_i = \sqrt{(x_{i1} - t_1)^2 + (x_{i2} - t_2)^2}$$

Für k = 1 wird der Trainingsvektor mit der geringsten Distanz ermittelt (durchgezogener Kreis). Das Genre dieses Vektors wird dem unbekannten Vektor zugewiesen. In diesem Beispiel wird der Testvektor als zum zweiten Genre gehörend klassifiziert.

Für k = 3 werden die drei nächstgelegenen Trainingsvektoren ermittelt, welche im Beispiel innerhalb des gestrichelten Kreises liegen. Das Genre wird durch Mehrheitsentscheidung aus den Genres dieser Trainingsvektoren ermittelt. Im Beispiel wird der unbekannte Vektor nun dem ersten Genre zugeordnet.

2.4.2. Radial Basis Function (RBF) Network

Ein Radial Basis Function Network ist ein lineares neuronales Netzwerk, das radiale Basisfunktionen[2] als Aktivierungsfunktionen verwendet.

Das RBF-Network versucht, eine Ausgangsfunktion $y = f(x)$ aus den Trainingsdaten anzunähern, ohne den Aufbau der Funktion zu kennen.

Typischerweise besteht ein solches Netzwerk aus drei Schichten: der Eingangsschicht, der verdeckten Schicht, sowie der linearen Ausgangsschicht.

[2] Radiale Basisfunktionen sind eine spezielle Klasse von Funktionen. Charakteristisch ist ihre monotone Abnahme (oder Zunahme) von einem Mittelpunkt aus. Dabei können der Mittelpunkt und die Skalierung mit der Entfernung variabel sein, ebenso wie die genaue Form. [30]

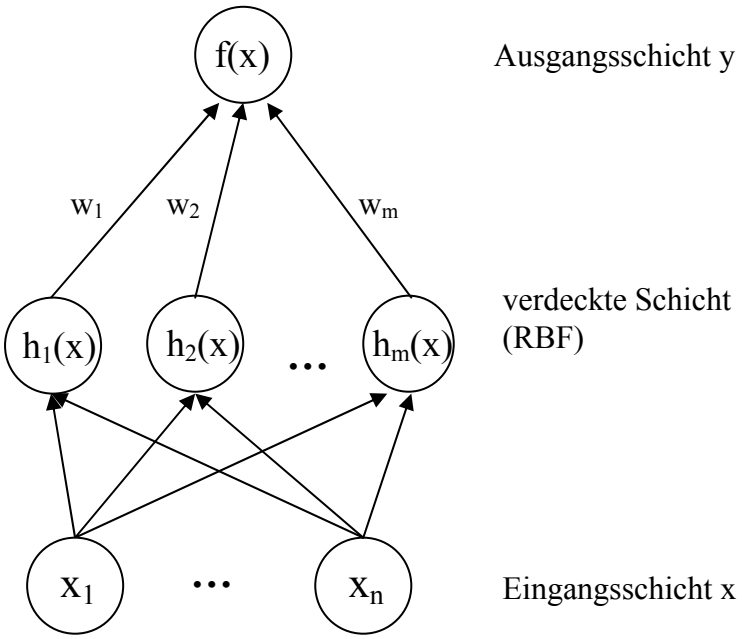


Abbildung 2-6
Ein RBF Network.
Der Eingangsvektor X wird auf den Ausgang y abgebildet.

Die verdeckte Schicht besteht aus *m* Knoten, so genannten Neuronen, die eine nichtlineare RBF-Funktion als Aktivierungsfunktion besitzen. Als Radiale Basisfunktion wird typischerweise die Gaußfunktion verwendet:

$$h(x) = \exp\left(-\frac{(x-c)^2}{r^2}\right) \qquad \text{(skalar)}$$

bzw.

$$h(X) = \exp\left(-\frac{\|X-c\|^2}{r^2}\right) = \exp\left(-\frac{(X-c)^T(X-c)}{r^2}\right) \qquad \text{(vektoriell)}$$

wobei $X = (x_1, x_2, \ldots x_n)$ den n-dimensionalen Eingangsvektor darstellt (bzw. im skalaren Fall, x den Eingangwert), *c* den Mittelpunkt und *r* den Radius der jeweiligen Basisfunktion.

Übertragen auf das Problem der Musikklassifikation entspricht der Eingangsvektor *X* dem Feature-Vektor, der zur Klassifikation verwendet wird. Die x_i stellen die einzelnen Komponenten des Feature-Vektors dar, also z.B. den Mittelwert eines bestimmten Features. Die Anzahl *m* der Neuronen der verdeckten Schicht ist eine vorgegebene Eigenschaft des Klassifikators. Die Variablen *c* und *r* sind Teil der verdeckten Schicht und somit nur indirekt von den Eingangsdaten abhängig.

Die Antwort des Netzwerkes am Ausgang lautet:

$$f(X) = \sum_{j=1}^{m} w_j \cdot h_j(X)$$

Die Parameter w_j stellen die individuellen Gewichtungen jedes Knotens der verdeckten Schicht für den jeweiligen Ausgang dar. Der Parameter m gibt die Anzahl der verwendeten Radialen Basisfunktion an.

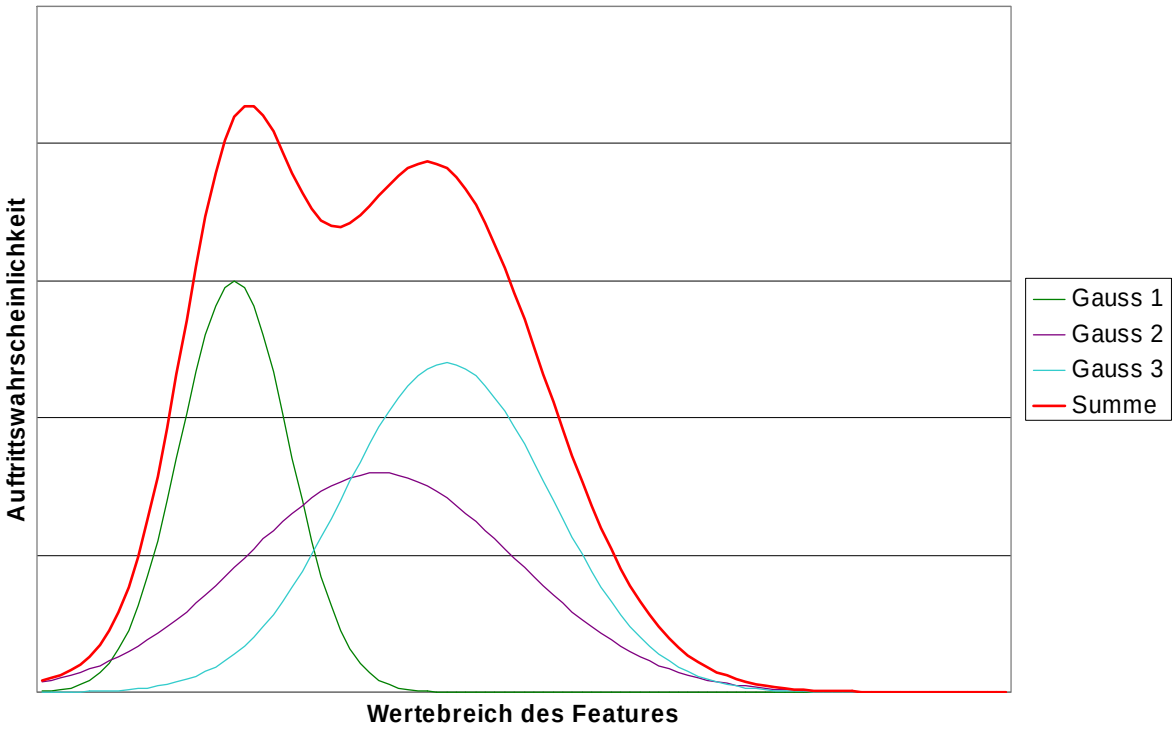


Abbildung 2-7
Eine aus drei Gaußfunktionen zusammengesetzte Funktion.

In Abbildung 2-7 ist eine eindimensionale Funktion abgebildet, die sich aus drei verschiedenen Gaußfunktionen zusammensetzt. Übertragen auf das Problem der Musikklassifikation könnte diese Funktion die Auftrittswahrscheinlichkeit der Werte eines einzelnen Features innerhalb einer bestimmten Klasse darstellen.

Die Aufgabe der Trainingsphase ist es, die zur Verfügung stehenden m Radialen Basisfunktionen möglichst optimal an den Verlauf dieser Funktion anzupassen.

Stehen mindestens drei Radiale Basisfunktionen zur Verfügung, so wäre es in diesem Beispiel möglich, den Verlauf der Funktion genau nachzubilden.

2.4.2.1. Trainingsphase

In der Trainingsphase wird für jedes Genre ein eigenes RBF-Network erstellt. Für jedes der Genres soll die Werteverteilung der Features aus den Trainingsdaten nachgebildet werden. Dazu wird aus den Trainingsvektoren mittels eines Histogramms ermittelt, wie häufig bzw. wahrscheinlich die Eingangswerte auftreten. $y_i = p(x_i)$

Als Unbekannte des Netzwerkes müssen jeweils c, r und die w_j bestimmt werden.

In einem ersten Schritt der Lernphase werden c und r ermittelt, oder sind manchmal auch fest vorgegeben. Als Lernalgorithmus für die Mittelpunkte werden meist Algorithmen der unüberwachten Clusteranalyse eingesetzt.

Im zweiten Schritt werden die Gewichtungen w_j optimiert, wofür sich ein least squares Algorithmus anbietet [30].

Als Term des quadrierten Fehlers ergibt sich (mit den p Trainingsdatensätzen (y_i , x_i)):

$$S = \sum_{i=1}^{p} \left(y_i - f(x_i) \right)^2$$

Die Minimierung dieser Funktion führt zu einer Reihe von m linearen Gleichungen für die m unbekannten Gewichte, die sich schreiben lässt als:

$$Aw = H^T y$$

Wobei H die *Design Matrix* ist

$$H = \begin{bmatrix} h_1(x_1) & h_2(x_1) & \cdots & h_m(x_1) \\ h_1(x_2) & h_2(x_2) & \cdots & h_m(x_2) \\ \vdots & \vdots & \ddots & \vdots \\ h_1(x_p) & h_2(x_p) & \cdots & h_m(x_p) \end{bmatrix}$$

und A^{-1} die *Varianz Matrix*

$$A^{-1} = \left(H^T H \right)^{-1}$$

Die gesuchten Gewichte ergeben sich schließlich als

$$w = A^{-1} H^T y$$

2.4.2.2. Klassifikationsphase

Für den zu klassifizierenden Feature-Vektor werden die Ausgangwerte aller RBF-Netzwerke berechnet. Als wahrscheinlichste Klasse des Testvektors gilt anschließend dasjenige Genre, das dem RBF-Netzwerk zugeordnet ist, dessen Ausgabewert am größten ist, d.h.

Klasse(X) = i | $f_i(X) = \max$, mit i = 1...k, *k* = Anzahl der Genres.

Der zu klassifizierende Feature-Vektor wird den RBF Netzwerken präsentiert und ihre Ausgangswerte berechnet. Als wahrscheinlichste Klasse des Testvektors gilt anschließend diejenige Klasse, die dem RBF Netzwerk zugeordnet ist, dessen Ausgabewert am größten ist.

2.4.3. Gaussian Mixture Model

Das Gaussian Mixture Model ist ein Modell um Funktionen nachzubilden, das sich aus der Summe mehrerer Gaußfunktionen zusammensetzt. Es nähert, eine Ausgangsfunktion $y = f(x)$ an die Eingangsdaten an, ohne den Aufbau der Funktion zu kennen.

Die Ausgangsfunktion $f(x)$ setzt sich aus mehreren gewichteten Gaußfunktionen zusammen.

$$f(X) = \sum_{i=1}^{k} a_i f_{Yi}(x_i)$$

Gebildet wird sie aus k Gaußfunktionen f_{Yi} , die mit den Gewichtungsfaktoren a_i gewichtet werden und dem n-dimensionalen Eingangsvektor $X = (x_1,...x_k)$. Diese k-dimensionalen Gaußfunktionen haben die Dichtefunktion $f_{Yi}(x)$, wobei μ den Mittelpunkt und Σ die Kovarianzmatrix der Funktion darstellt.

$$f_{Yi}(x) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu_i)^T \Sigma^{-1}(x-\mu_i)\right)$$

Die Ausgangsfunktion setzt sich damit genau so zusammen, wie bei dem RBF-Network, jedoch unterscheidet sich die Trainingsphase.

Übertragen auf das Problem der Musikklassifikation entspricht der Eingangsvektor X wieder dem Feature-Vektor, der zur Klassifikation verwendet wird. Die x_i stellen die einzelnen Komponenten des Feature-Vektors dar. Die Anzahl k der Gaußfunktionen ist eine gegebene Eigenschaft des Klassifikators.

2.4.3.1. Trainingsphase

In der Trainingsphase wird wieder für jedes Genre ein eigenes Gaussian Mixture Model erstellt. Die Aufgabe besteht auch hier wieder darin, die *k* Gaußfunktionen möglichst gut an die Werteverteilung der Features der Trainingsdaten anzupassen.

Für Gaussian Mixture Models wird meist ein sogenannter expectation-maximization Algorithmus angewendet. Dieser Algorithmus besteht aus zwei getrennten Schritten, dem *expectation* und dem *maximization* Schritt, die iterativ wiederholt werden, bis eine gewünschte Genauigkeit erreicht ist. Für die erste Iteration werden die zu bestimmenden Daten einmal geschätzt, sie konvergieren im Laufe der Iterationen zu einem lokalen Minimum [28].

Seien Θ die zu bestimmenden Parameter und *x* die Eingangsdaten. Der Ausdruck $p(x|\theta)$ gibt dann den Erwartungswert an, dass die Eingangsdaten beobachtet werden, falls die Parameter die angegebenen Werte haben. Der Index *n* steht für die Nummer der Iteration.

Im Expectation Schritt wird der Logarithmus dieses Erwartungswertes aufsummiert.

$$Q(\theta_n) = \log p(x|\theta_n) = \sum_k \log\left[p(x_k|\theta_n)\right]$$

Im Maximization Schritt werden daraus die neuen Parameter für den nächsten Durchlauf der Iteration bestimmt, indem das Maximum der Funktion $Q(\theta_n)$ bestimmt wird.

$$\theta_{n+1} = \arg\max_{\theta} Q(\theta_n)$$

2.4.3.2. Klassifikationsphase

Für den zu klassifizierenden Feature-Vektor werden die Ausgangwerte aller Gaussian Mixture Modelle berechnet. Als wahrscheinlichste Klasse des Testvektors gilt anschließend dasjenige Genre, das dem Gaussian Mixture Model zugeordnet ist, dessen Ausgabewert am größten ist, d.h.

Klasse(X) = i | $f_i(X)$ = max , mit i = 1...k, k = Anzahl der Genres.

3. VERWENDETE HW-PLATTFORMEN

Im Rahmen dieser Arbeit wurden Implementierungen auf einem General Purpose Prozessor und einem digitalen Signalprozessor betrachtet.

Weiterhin wurden Referenz-Werte, die auf einer weiteren HW-Plattform ermittelt wurden, einem OMAP2410 DSP, herangezogen.

Als General Purpose Prozessor diente ein Pentium 4 der Firma Intel mit 3,2 Ghz Taktfrequenz, dem 1 GB Arbeitsspeicher zur Verfügung stand.

Bei dem im Rahmen dieser Arbeit betrachteten Digitalen Signalprozessor handelt es sich um den Smart Audio Accelerator, der auf der Nomadik-Plattform *STn8810* der Firma *STMicroelectronics* (ST) integriert ist. Im Folgenden werden die wesentlichen Architekturmerkmale und -besonderheiten der Nomadik-Plattform beschrieben.

3.1. Nomadik Open Platform

Die Nomadik Prozessorfamilie der Firma STMicroelectronics (ST) besteht aus einer Reihe von Prozessoren, die speziell für Multimedia-Anwendungen auf Mobilgeräten entwickelt wurden.

Die heterogene Prozessorarchitektur besteht jeweils aus einem ARM® Prozessor, die industrieweit häufig in Mobilgeräten eingesetzt werden, sowie dedizierten Spezialprozessoren, on-chip Speicher und einer Vielzahl an Interfaces zur Ansteuerung von Peripheriegeräten. Untereinander sind alle Komponenten mit einer multi-layer advanced microcontroller bus architecture (AMBA™) verbunden, die für einen effizienten Datentransport zwischen den Komponenten sorgt.

Eine solche heterogene Architektur wird als System-on-Chip (SoC) bezeichnet. Der Vorteil einer SoC-Architektur gegenüber der Verwendung mehrerer separater Chips liegt in einer Verringerung der Herstellungskosten, der Chipgröße und der Verlustleistungsaufnahme, während sich gleichzeitig die Zuverlässigkeit erhöht.

Das Ziel der open platform ist es, aufbauend auf den MIPI (Mobile Industry Processor Interface) Spezifikationen, den Herstellern größere Freiheiten und Unabhängigkeit von einer proprietären CPU Architektur zu bieten. Entwurf und Design von Hardware und Software soll vereinfacht werden. Dazu baut die Nomadik-Architektur auf dem als Industriestandard ge-

ltenden ARM Prozessor auf, bietet eine offene API und lässt sich in der Hardware-unabhängigen Hochsprache C programmieren.

Die Nomadik-Plattform wurde auf folgende Eigenschaften hin optimiert:

- höchste Audio- und Video-Qualität
- sehr geringe Verlustleistungsaufnahme
- einfache Entwicklung von Anwendungen für kürzere Produktionszyklen
- Anpassung an verschiedene Marktsegmente
- Skalierbarkeit für zukünftige Mulimedia-Anwendungen

3.2. Nomadik-Plattform STn8810

Die Nomadik-Plattform *STn8810* wird in einer 130nm Technologie gefertigt und besteht aus drei Prozessorkernen, die mit 1,2 V betrieben werden. Der Hauptprozessor ist ein ARM926EJ-S Kern, der alle Systemkomponenten verwaltet und konfiguriert. Auf ihm läuft auch das Betriebssystem und Benutzeranwendungen. Als Hardware-Beschleuniger stehen dem Hauptprozessor folgende DSPs zur Seite, die auf der MMDSP+ Architektur von ST aufbauen:

- Der Smart Audio Accelerator (SAA) übernimmt die Audioverarbeitung, wie Kodieren, Dekodieren, Mischen von Audiostreams und Anwenden von Audioeffekten.
- Der Smart Video Accelerator (SVA) übernimmt die Bildverarbeitung. Er ist im Wesentlichen eine um zusätzliche video-spezifische Hard- und Softwarefunktionen erweiterte Version des MMDSP+.

Die verwendete Plattform verfügt über 40 kByte on-Chip SRAM, 32 MB SDRAM, 16 MB NOR-Flash-Speicher und 32 kByte on-chip Boot-ROM. Darüber hinaus bietet sie verschiedene Schnittstellen, wie z.B. USB On-The-Go, Fast IrDA, SD-/MMC-Card Schnittstelle und LCD Display Controller. Der NOR-Flash-Speicher und der SDRAM befinden sich mit dem SoC im selben Gehäuse.

Dem Energieverbrauch der Plattform wurde besondere Aufmerksamkeit gewidmet, um die Akkulaufzeit zu verlängern. Er wird durch eine advanced power management unit (PMU) reguliert, die die Energiesparmodi der Komponenten überwacht und deren Spannungen und Taktfrequenzen je nach Bedarf anpassen kann.

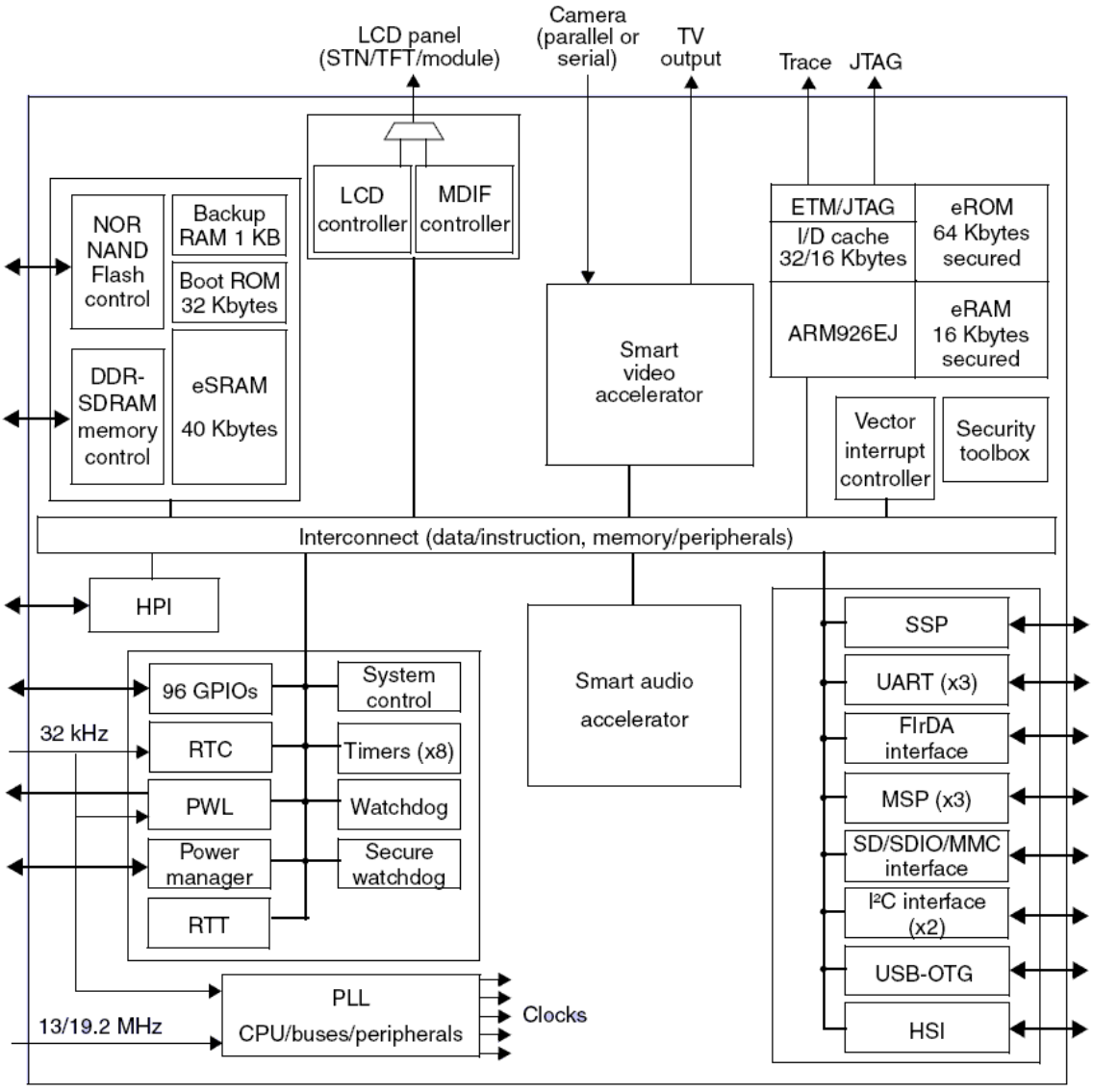


Abbildung 3-1
Blockschaltbild der Nomadik-Plattform

3.3. ARM926EJ-S

Der ARM926EJ-S Prozessor ist eine 32-Bit reduced instruction set computer (RISC) CPU und gehört zu der Familie der lizenzierbaren ARM® Prozessoren, die industrieweit in Mobilgeräten eingesetzt werden.

Er greift auf 32 kByte Instruktions- und 16 kByte Datencache zurück und verfügt über eine Memory Management Unit (MMU). Unterstützt werden drei verschiedene Befehlssätze, 32-bit für hohe Performance, 16-bit für geringe Codegröße und byte Java mode (Jazelle™) um

Javacode direkt auszuführen. Der Befehlssatz wurde um einige DSP-artige Operationen erweitert, und bietet eine MAC-Operation, die nur einen Taktzyklus zum Ausführen benötigt.

Die maximale Taktfrequenz beträgt bis zu 264 MHz, kann aber auch von der Power Management Unit reduziert werden.

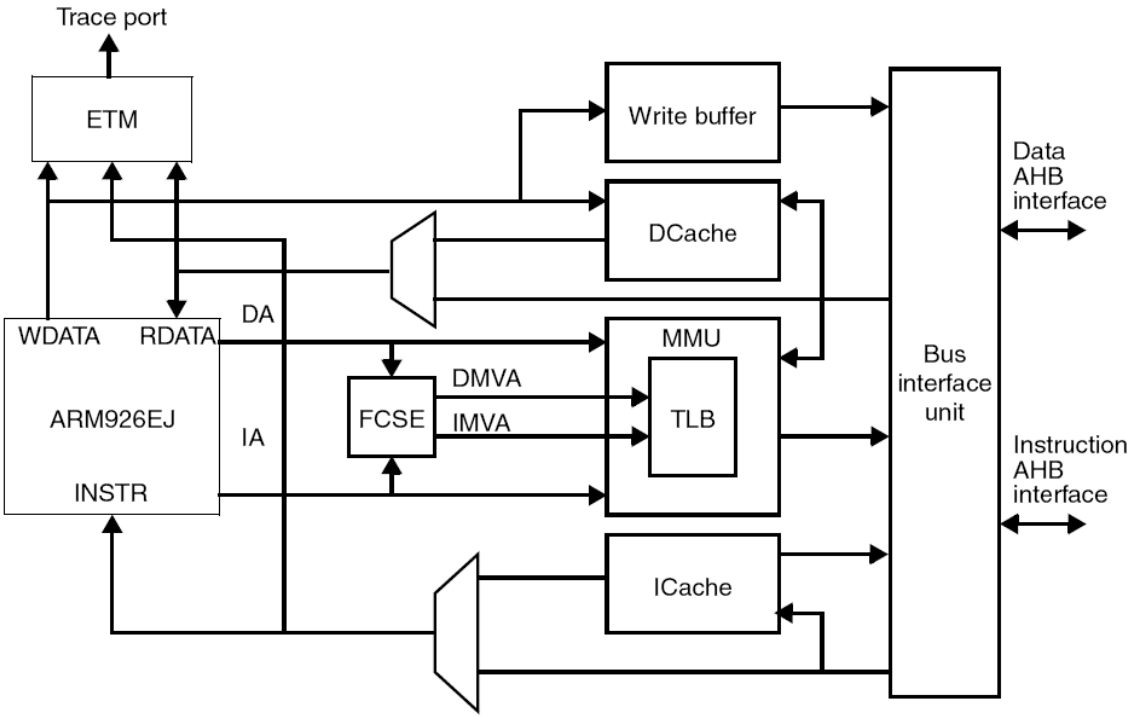


Abbildung 3-2
Blockschaltbild des ARM926EJ-S

3.4. Smart Audio Accelerator (SAA)

Der Smart Audio Accelerator basiert auf dem programmierbaren MMDSP+ Audio-DSP. Dieser ist ein 16 oder 24-bit Harvard VLIW Prozessor, der mit 133 MHz läuft und über einen 24bit breiten Datenbus verfügt. Alle Befehlsworte haben eine Länge von 64bit und benötigen lediglich einen Taktzyklus zur Abarbeitung, die Ausnahme bilden Sprungbefehle.

Der SAA kann wahlweise in einem 16 oder 24-bit Modus betrieben werden, abhängig von der benötigten Datenwortbreite.

Der Datenaustausch zwischen Registern und Datenspeichern erfolgt über den X-Bus oder den Y-Bus. Über den X-Bus können Daten sowohl gelesen als auch geschrieben werden. Auch

der Y-Bus erlaubt theoretisch Schreib- und Leseoperationen, ist in allen MMDSP+ Produkten aber auf Lesezugriffe beschränkt. Die beiden unabhängigen Busse erlauben es, spezielle Operationen stärker zu parallelisieren. Die beiden lokalen Speicher X RAM und Y RAM umfassen jeweils 32k Datenwörter à 24 bit.

Der SAA ist als VLIW Prozessor in der Lage bis zu acht Befehle parallel auszuführen. Als DSP-spezifische Eigenschaften verfügt er über Befehle, die Adresszeiger bei einem Speicherzugriff automatisch erhöhen, ebenso wie über eine Hardwareunterstützung für Programmschleifen.

Im Kontext der Nomadik-Plattform wird der SAA von dem ARM-Prozessor über Konfigurationsregister konfiguriert und gesteuert. Datenquellen und –senken für den SAA sind entweder der Arbeitsspeicher des Systems oder externe Ports und Interfaces.

Die Implementierung ist auf besonders geringen Energieverbrauch ausgelegt.

Intern stehen mehrere Ausführungseinheiten zur Verfügung:

- Data computation unit (DCU)
  Diese Einheit berechnet arithmetische Operationen. Sie enthält einen Multiplizierer/Akkumulator, eine ALU, einen Barrel Shifter und greift auf 2 Bänke mit je 7 x 16 bit oder 7 x 24 bit breiten Registern zurück. Diese Register können für Arithmetik mit doppelter Präzision (32 oder 48 bit) kombiniert werden.
- Floating point calculation unit (FPU)
  Diese Einheit übernimmt 32 bit Fließkomma-Berechnungen. Sie beinhaltet 7 x 32 bit Register und ist konform zum IEEE Standard. Diese Einheit ist optional und im STn8810 nicht vorhanden, ihr Fehlen schränkt den Funktionsumfang allerdings nicht ein.
- Address computation unit (ACU)
  Diese Einheit kümmert sich um die Adressberechnung für den Datenspeicher. Sie unterstützt verschiedene, teils DSP-spezifische, Adressierungsmodi: post increment/decrement, modulo, bit reverse, index und linear
- Controller unit (CU)
  Diese Einheit sorgt für den Programmablauf des SAA. Sie ist ein Standarddecoder mit 2-stufiger Pipeline. Neben gewöhnlichen Verzweigungsmöglichkeiten unterstützt sie auch über bis zu drei Ebenen geschachtelte Hardware-Schleifen.

Abbildung 3-3 zeigt ein Blockschaltbild des SAA.

Zu den Fähigkeiten des SAA zählen:

- Wiedergabe, Aufnahme und Transcodierung von Audiosignalen
- Gleichzeitiges Bearbeiten mehrerer Audiostreams mit Unterstützung folgender Transformationen:
  - Audio-Encodierung und Decodierung
  - Sprach-Encodierung und Decodierung
  - Pre- und Post-Transformationen
- Audio- und Sprachcodecs: u.a. MP3, AAC, AMR
- Konvertierung von Audio-Sampleraten auf 44,1 kHz und 48 kHz
- Audioeffekte und –verbesserungen: u.a. Mixer, Equalizer, dynamic range control, 3D Audio, Rauschunterdrückung, Echokompensation

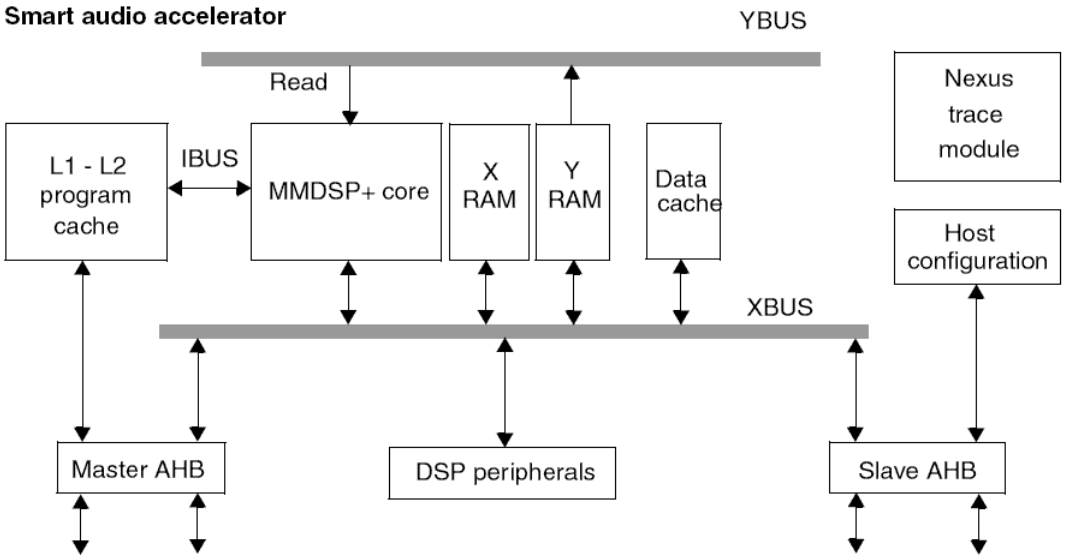


Abbildung 3-3
Blockschaltbild des Smart Audio Accelerators

3.5. Smart Video Accelerator (SVA)

Ebenso wie der SAA basiert der Smart Video Accelerator auf einem MMDSP+ Kern, läuft aber im Gegensatz zu diesem mit 66 MHz und bietet nur einen 16 bit Modus. Ihm steht ein 40 kByte großer eSRAM zur Verfügung, der als Buffer für Bilddaten dient. Konfiguriert wird er von dem ARM Hauptprozessor.

Er unterstützt unter anderem:

- MPEG-4 bis zu VGA Auflösung in Echtzeit bei 30 fps. Encodierung oder Decodierung.
- H.263 video codec, subQCIF oder QCIF bei 15 fps für Videokonferenz-Anwendungen
- H.263 video codec, CIF bei 30 fps, encodieren oder decodieren
- Beschleunigung für JPEG-Encodierung und Decodierung, bis 4080 x 4080 Pixel
- Bild Pre- und Post-Processing.

Der SVA wurde im Rahmen dieser Arbeit nicht verwendet.

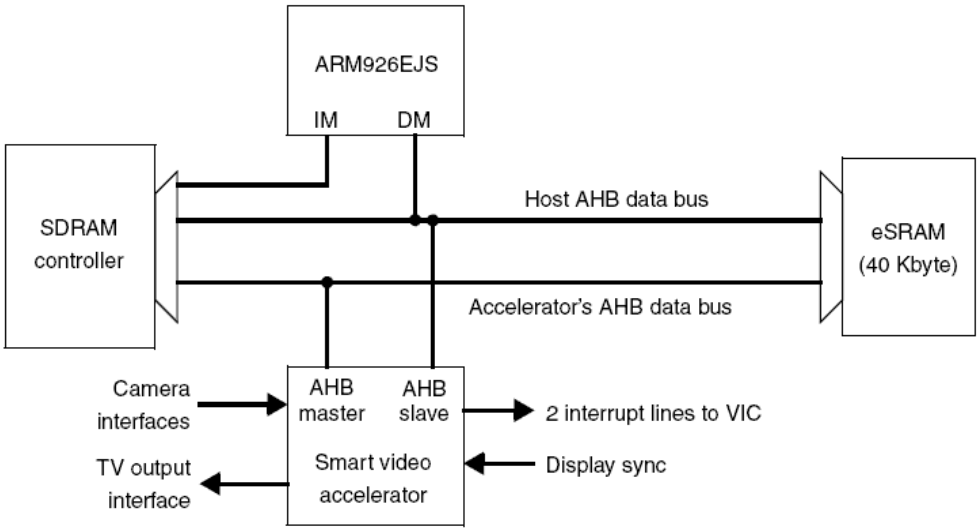


Abbildung 3-4
Blockschaltbild der Anbindung des Smart Video Accelerators.

4. DURCHFÜHRUNG VON KLASSIFIKATIONS-EXPERIMENTEN

4.1. Experimentalaufbau

Für die Nachvollziehbarkeit und Reproduzierbarkeit der Experimente sind die Experimentierumgebung und die Art der Durchführung der Experimente von großer Bedeutung. In diesem Kapitel wird darauf eingegangen, wie die Testdatensätze und die Versuche aufgebaut sind.

4.1.1. Musikdatensätze

Für die Versuche stand ein großer Satz von 729 Musikstücken zur Verfügung, bestehend aus den Alben mehrerer Künstler. Diese stammen vom dem Magnatune-Label und sind anhand der Künstler in mehrere Genres vorsortiert. Beim Magnatune-Label handelt es sich um eine Online-Plattenfirma, deren Ziel es ist, durch verringerte Overhead-Kosten den Künstlern faire Verträge anbieten zu können. Die hier verwendeten Musikstücke besitzen keinen Kopierschutz und dürfen für nicht-kommerzielle Zwecke lizenzfrei verwendet werden.

Die Einordnung der Musikstücke in Genres erfolgte nach folgendem Schema:

- Classical
- Electronic
- Jazz / Blues
- Metal / Punk
- Rock / Pop
- World

Aus diesen Liedern wurden mehrere Test-Datensätze zusammengestellt.

1. **Satz 1**
   Der erste Datensatz besteht aus handsortierten Liedern. Jedes Stück wurde kurz angehört und dann entschieden, ob es (dem subjektiven Empfinden des Hörers nach) repräsentativ für das jeweilige Genre ist.
   Auf diese Weise wurde ein Satz von 12 Liedern pro Genre zusammengestellt, also eine Anzahl von insgesamt 72 Liedern.
2. **Satz 2**
   Um den Einfluss von Art und Anzahl der Genres zu testen, besteht der zweite Datensatz nur aus Musikstücken der Genres „Classical“ und „Rock / Pop“. Für jedes Genre wurden 30 Lieder ausgewählt. Damit besteht dieser Datensatz aus 60 Liedern.

Da die Features für eine Samplerate von 22050 Hz definiert sind, die Musikstücke aber in 44100 Hz vorliegen, wurden alle verwendeten Lieder einmal auf die geforderte Samplerate konvertiert. Die einzelnen Abtastwerte liegen in einer Auflösung von 16 Bit pro Sample vor.

Aus jedem Test-Datensatz wurden alle implementierten Features extrahiert und in einen Feature-Vektor abgelegt. Dieser wird zusammen mit dem jeweiligen Genre zur weiteren Verwendung im arff-Dateiformat[3] abgespeichert.

Für die einzelnen Experimente wurden Dateien mit Gewichtungen erstellt, die bestimmen, welche der Features verwendet werden.

Die Durchführung der einzelnen Experimente erfolgte automatisiert per Batchdatei, die die Versuche mit den jeweils korrespondierenden Parametern über die Kommandozeile startet.

4.1.2. Kreuzvalidierung

Die Experimente wurden nach dem Kreuzvalidierungsverfahren durchgeführt. Dieses statistische Verfahren ist eine Standardmethode, um Fehlerraten von statistischen Modellen zu ermitteln.

[3] Eine Attribute-Relation File Format Datei (arff Datei) ist eine Textdatei, in der Datensätze gespeichert werden, die aus einem gemeinsamen Satz von Attributen bestehen. Auf einen Header, in dem die Attribute definiert werden, folgt der Datenbereich. In diesem besteht jede Zeile aus den durch Kommata getrennten Attributwerten eines Datensatzes. [51]

Der Datensatz wird dazu in *k* Teilmengen aufgeteilt. Anschließend werden *k*-1 dieser Teilmengen genutzt, um den Klassifikator zu trainieren, während der verbleibende Teil des Datensatzes zum Testen des trainierten Klassifikators verwendet wird. Diese Testdurchläufe werden *k* mal wiederholt, so dass alle Teilmengen einmal als Testdatensatz genutzt werden.

Bei einer stratifizierten Kreuzvalidierung werden die Teilmengen so gebildet, dass die Verteilung der Klassen in ihnen der Verteilung im gesamten Datensatz entspricht.

Das Ergebnis des Tests ist der Anteil der richtig klassifizierten Datensätze an der Gesamtzahl der Datensätze.

Die im Rahmen dieser Arbeit durchgeführten Versuche wurden als stratifizierte Kreuzvalidierungsexperimente durchgeführt. Dabei wurden $k = 10$ Teilmengen verwendet, was allgemein als guter Wert für diese Experimente angesehen wird.

Übertragen auf die Musikstücke des Test-Datensatzes Satz 1, der aus 72 Liedern besteht, ergibt sich folgende Aufteilung des Datensatzes:

Nummer der Teilmenge	1	2	3	4	5	6	7	8	9	10
Anzahl Lieder der Teilmenge	6	6	6	12	6	6	6	6	12	6

Um den Anteil der sechs Genres in allen Teilmengen konstant zu halten, werden die Musikstücke so in zehn Teilmengen unterteilt, dass acht Teilmengen entstehen, die sich aus je sechs Liedern zusammensetzen, und zwei Teilmengen, die sich aus je zwölf Liedern zusammensetzen. In den zehn Durchläufen der Kreuzvalidierung wird nun der Reihe nach immer eine der Teilmengen als Testdatensatz verwendet, während die restlichen neun Teilmengen den Trainingsdatensatz bilden. Während aller Durchläufe wird die Anzahl der richtig klassifizierten Musikstücke gezählt. Die Klassifikationsgüte ergibt sich anschließend als der Anteil der richtig klassifizierten Musikstücke an der Gesamtzahl der Musikstücke. Wenn *x* die Anzahl der richtig klassifizierten Musikstücke angibt, so berechnet sich die Güte in diesem Beispiel zu

$$p = \frac{x}{72} \cdot 100\%$$

4.2. Existierende Softwaretools

Es existieren zurzeit einige Software-Werkzeuge, die für die Aufgabe der Feature-Extraktion oder für die Klassifikation eingesetzt werden können. Alle aufgeführten Programme entstammen dem akademischen Umfeld. Die folgende, aus [47] übernommene Liste gibt einen Überblick über einige der gängigsten Programme.

jAudio	
Zweck des Programms	Wurde ursprünglich als Teil der ACE (Autonomous Classification Engine) Umgebung entwickelt und ist mittlerweile in die OMEN (On-demand Metadata Extraction Network) Umgebung eingebunden. jAudio ist ebenfalls als separates Programm erhältlich.
Entwickler	Schulich School of Music, McGill University
Beginn der Entwicklung	2005
Aktueller Status	beta
Programmiersprache	Java
Lizenz	LGPL
Unterstützte Audioformate	Wave, mp3 und andere Formate, die von der Java Sound API unterstützt werden
Ausgabeformate	Weka ARFF, ACE XML Format
Batch- und Kommandozeilenunterstützung	Ja, Batchdateien können konfiguriert und für die spätere Anwendung mit oder ohne GUI gespeichert werden
Details der Feature-Extraktion	Aktuell mehr als 20 vordefinierte Feature-Gruppen einschließlich Eigenschaften des Signal- und Spektralbereichs, MFCC, beat histogramme
WWW	http://jaudio.sourceforge.net/

M2K (Music to Knowledge)	
Zweck des Programms	Satz von Modulen für die D2K (Data to Knowledge) Umgebung, die für verschiedene MIR (Music Information Retrieval) Aufgaben und ihre Auswertung genutzt werden sollen. Verschiedene Erweiterungen von M2K können während der jährlichen MIREX (Music Information Retrieval Evaluation eXchange) bewertet werden.
Entwickler	Graduate School of Library & Information Science, University of Illinois at Urbana-Champaign, The Automated Learning Group, The National Center for Supercomputing Applications, School Computing Sciences, University of East Anglia, Sun Microsystems Laboratories
Beginn der Entwicklung	2004
Aktueller Status	1.2 release steht kurz bevor (einschließlich des evaluation code der MIREX 2006)
Programmiersprache	Java
Lizenz	Akademische Nutzung, Nutzung in der Forschung und kommerzielle Testlizenzen für D2K; M2K wird unter einer freien Lizenz vertrieben
Unterstützte Audioformate	Wave, mp3

Ausgabeformate	D2K Tabelle, ASCII Datei, ARFF (über D2K Tabelle), Java serialization
Batch- und Kommandozeilenunterstützung	Ja
Details der Feature-Extraktion	Beispiele verschiedener Features werden mitgeliefert, dazu zählen MFCC, spectral contrast Features, allgemeine Beschreibungen der spektralen Form (centroid, flux, etc.) und Funktionen zur onset Detektion
Details der Klassifikationsalgorithmen	Module zur Klassifikation sind in D2K enthalten und enthalten viele Methoden: Entscheidungsbäume, bayesian classifier, Weka learner, Neuronale Netze, SVM, tec.
WWW	http://www.music-ir.org/evaluation/m2k/release

MusicMiner	
Zweck des Programms	Musik-Browser, der die Unterschiede zwischen Liedern und Künstlern darstellt
Entwickler	Databionics Research Group, Universität Marburg
Beginn der Entwicklung	2005
Aktueller Status	stable
Programmiersprache	Java
Lizenz	GPL
Unterstützte Audioformate	Wave, mp3, ogg, wma, mp2, m4a
Ausgabeformate	ASCII Datei
Batch- und Kommandozeilenunterstützung	Ja, Feature-Extraktion kann über die Kommandozeile gestartet werden
Details der Feature-Extraktion	Features werden mithilfe des Yale Valueseries preprocessing plugins extrahiert (siehe dort)
Details der Klassifikationsalgorithmen	Zur Visualisierung der Ähnlichkeiten der Musikstücke werden selbstorganisierende Karten verwendet
WWW	http://musicminer.sourceforge.net

Yale / Rapidminer	
Zweck des Programms	Umgebung für Maschinenlernen und Data Mining. Feature-Extraktion über ValueSeries preprocessing plugin möglich, zahlreiche Klassifikationstechniken sind implementiert
Entwickler	Artificial Intelligence Unit, Universität Dortmund
Beginn der Entwicklung	2001
Aktueller Status	stable, Ende 2007 Umbenennung von yale nach rapidminer
Programmiersprache	Java
Lizenz	GPL
Unterstützte Audioformate	Wave, mp3, ogg
Ausgabeformate	Weka ARFF und zahlreiche weitere Formate, die auch benutzerdefiniert sein können
Batch- und Kommandozeilenunterstützung	Ja, XML-Batchdateien können für die spätere Nutzung mit oder ohne GUI gespeichert werden
Details der Feature-Extraktion	das Valueseries preprocessing plugin erlaubt sowohl die Extraktion einiger Features (aktuell über 30) als auch viele Transformationen, wie z.B. FFT, Autokorrelation, Transformation in den Phasenraum, differenz-Filter, etc.
Details der Klassifikationsalgorithmen	Es stehen verschiedene Klassifikationsmethoden zur Verfügung, darunter Weka learners, SVM, Bayesian classifier, Entscheidungsbäume, association rules lerners
WWW	http://rapid-i.com

Zu Beginn dieser Arbeit wurde jAudio verwendet um Features zu extrahieren. In ersten Experimenten stellte Sich jedoch heraus, dass die Feature-Implementation von jAudio von den Definitionen in [46] signifikant abweichen. Beispielsweise waren alle Features fehlerhaft, die im Frequenzbereich ermittelt wurden, da die FFT offenbar falsch berechnet wird. Ihre Werte wiesen deutliche Abweichungen von einer Referenz-Implemenation (Matlab) auf. Für alle folgenden Versuche wurden ausschließlich die entsprechend der Definition nach [46] selber implementierten Features verwendet.

Zur Durchführung erster Klassifikationsexperimente wurde yale / rapidminer verwendet. Später erfolgte die Durchführung der Experimente mit einer eigenen Implementation des k-Nearest-Neighbor Algorithmus.

4.3. Ermittlung typischer Klassifikationsgüten

Es existiert eine Vielzahl an Veröffentlichungen, die sich bereits mit dem Thema der Klassifikation von Musikstücken befassen. Alle publizierten Verfahren unterscheiden sich hinsichtlich der Art und Anzahl der Genres, der verwendeten Art und Anzahl von Features und Klassifikatoren und der verwendeten Musikstücke (Art und Größe der Musikdatenbanken).

In Tabelle 3 sind einige Angaben aus der Literatur zusammengefasst. Die folgende Übersicht listet hierzu die in Tabelle 3 verwendeten Abkürzungen der Klassifikatoren auf:

Abkürzung Klassifikator	Bedeutung
SVM	Support Vector Machine
KNN	k-Nearest-Neighbor
GMM	Gaussian Mixture Model
MLP	Multilayer Perceptron
PCA	Principal Component Analysis
NN	Neuronal Network

Lite-ratur	Anzahl Genres	Klassifikator	Anzahl Musikstücke	Anzahl Features	Typ. erreichte Klassifikationsgüte	Besonderheiten
[16]	15	SVM	~500	5	70%	Handgenerierter Entscheidungsbaum (zunächst grobe, dann feine Unterteilung nach Genres)
[17]	2	MLP KNN 1-7	414	9	90-91% 82-90%	Klassifiziert nur einen oder drei Ausschnitte des Liedes, mit anschließender Mehrheitsentscheidung
[18]	7	GMM 4-64	175	1	79-92%	Benutzt nur die MFCC-Koeffizienten
[19]	10	PCA-NN-SVM PCA-NN-GMM PCA-NN-KNN PCA-SVM PCA-GMM PCA-KNN	1000	13	90% 81% 85% 68% 67% 68%	PCA zur Reduktion der Dimension. Hidden layer des nachfolgenden neuronalen Netzwerks als Eingangsvektor der Klassifikatoren
[20]	10 4 6 10 4 6	KNN 1-5 GMM 2-4	1000 400 600 1000 400 600	14	56-60% 70-78% 56-58% 60-61% 81-88% 62-68%	Die zehn Genres sind verschiedene Musikstile. Die vier Genres bestehen aus vier Arten Classic, die sechs Genres aus sechs Arten Jazz
[21]	2	SVM KNN 1	100	2	96% 79%	Unterscheidung nach Instrumentalmusik und Musik die Gesang enthält
[22]	4	SVM KNN GMM	100	5	93% 80% 87%	Handgenerierter zweistufiger Entscheidungsbaum mit unterschiedlichen Features an jedem Knoten. Zuerst Unterteilung in zwei Gruppen, dann nach Genres.
[23]	5	SVM KNN 3	1022	4	78% 72%	Benutzt lediglich die Mittelwerte der Features zur Klassifikation
[50]	-	-	-	7	-	lediglich Laufzeitanalyse ohne Angabe von Klassifikationsgüten

Tabelle 3 Aufstellung einiger Literaturangaben

Die typischen Klassifikationsgüten der in Tabelle 3 aufgelisteten Experimente reichen von 56% bis hin zu 96%.

Das beste Ergebnis von 96% erreicht [21] in seinem Experiment mit einem SVM-Klassifikator. Darin wird zwischen Instrumentalmusik und Musik, die Gesang enthält, unterschieden, um aus diesen anschließend eine Vorschau erstellen zu können. Bemerkenswert ist diese Klassifikationsgüte bei Nutzung von nur zwei Features, doch muss auch nur eine Trennung zwischen zwei Genres erfolgen.

Einen interessanten Ansatz zur Reduktion des Rechenaufwandes zur Extraktion der Features verfolgt [17]. In dem Experiment werden nicht die Features des gesamten Musikstückes extrahiert sondern es werden nur die Features aus einem oder drei kurzen Ausschnitten des Musikstücks zur Klassifikation herangezogen. Werden drei Ausschnitte betrachtet, so wird das Genre durch eine Mehrheitsentscheidung bestimmt. Die Klassifikationsgüte für die zwei Genres Rock und Classic liegt bei Nutzung von neun Features wieder über 80%.

Die KNN-Klassifikatoren erreichen, ebenso wie in [21], für zwei Genres wieder Klassifikationsgüten im Bereich von 80%. Einen handgenerierten Entscheidungsbaum zur Klassifikation benutzen sowohl [16] als auch [22] in ihren Experimenten. [16] unterteilt zunächst grob in drei Genres, um anschließen jeweils eine genauere Unterscheidung in einzelne Musikrichtungen vorzunehmen. Obwohl 15 Genres zu unterscheiden sind, liegt die Klassifikationsgüte bei diesem Ansatz bei 70%.

Auch [22] nutzt einen Entscheidungsbaum, doch werden die dort betrachteten vier Genres zunächst anhand von zwei Features in zwei Gruppen getrennt. Erst anschließend erfolgt mittels anderer, weiterer Features eine Trennung in die einzelnen Genres. Der Nachteil dieser beiden Ansätze, trotz der guten Ergebnisse, besteht in der starren und unflexiblen Vorgabe der Klassifikationskriterien. Um die Datenmengen durch hochdimensionale Feature-Vektoren zu reduzieren und auf die aussagekräftigsten Merkmale zu reduzieren, verwendet [19] eine Principal Component Analysis (PCA). Der so auf 36% des ursprünglichen Umfangs reduzierte Feature-Vektor wird zur Klassifikation verwendet oder erst einem Neuronalen Netzwerk zugeführt, dessen hidden layer anschließend als Eingangvektor der Klassifikatoren verwendet wird.

In [18] wird nur ein einziges Feature verwendet, die MFCC Koeffizienten, um zwischen sieben Genres zu differenzieren. Die verwendeten GMM-Klassifikatoren erreichen dabei eine Klassifikationsgüte von bis zu beachtlichen 92%.

In [50] wurde keine Klassifikation durchgeführt, da das Experiment ausschließlich der Laufzeitanalyse diente. Die in diesem Experiment verwendeten Features werden im weiteren Verlauf dieser Arbeit als „Referenz-Feature-Satz“ herangezogen.

Um diese Versuche nachzuvollziehen wurden Experimente zusammengestellt, die ähnliche Klassifikatoren und ähnliche Feature-Sätze verwenden. Die verwendeten Feature-Sätze sind in den meisten Fällen nicht vollständig, ihnen fehlen in erster Linie auf den Takt bezogene Features, die aus dem „beat histogram“ ermittelt werden. Darüber hinaus wurden heuristisch weitere Feature-Kombinationen für Experimente zusammengestellt.

Nummer	Feature-Name
F1	Spectral Centroid
F2	Spectral Rolloff
F3	Spectral Rolloff 2
F4	Spectral Flux
F5	Spectral Extend
F6	Spectral Bandwidth
F7	low Energy Windows
F8	RMS
F9	Spectral Kurtosis
F10	Sum of Correlated Components
F11	Zero Crossing Rate
F12	Relative Periodicity Amplitude Peaks (1-2)
F13	Ratio of Second and First Periodicity Peak
F14	Position of Main Peaks (1-5)
F15	Power Spectrum (0)
F16	Power Spectrum (0-1)
F17	MFCC (0-4)
F18	MFCC (0-12)
F19	Fundamentalfrequenz
F20	Amplitude of Maximum in Chromagram
F21	Pitch Intervall between two max Peaks in Chromagram

Tabelle 4 In den Experimenten verwendete Features.

Nummer	Klassifikator	Feature																			
		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21
1	KNN 1	X	X		X			X				X	X	X								
2	KNN 3	X	X		X			X				X	X	X								
3	KNN 5	X	X		X			X				X	X	X								
4	KNN 7	X	X		X			X				X	X	X								
5	RBF 4																		X			
6	RBF 8																		X			
7	RBF 32																		X			
8	KNN 1											X	X	X			X		X			
9	KNN 3											X	X	X			X		X			
10	RBF 3											X	X	X			X		X			
11	KNN 3								X			X								X		
12	KNN 1	X	X		X			X				X					X					
13	KNN 3	X	X		X			X				X					X					
14	KNN 5	X	X		X			X				X					X					
15	RBF 2	X			X	X		X				X							X			
16	RBF 3	X			X	X		X				X							X			
17	RBF 4	X			X	X		X				X							X			
18	KNN 3	X	X		X			X			X	X	X	X	X				X			
19	KNN 5	X	X		X			X			X	X	X	X	X				X			
20	RBF 3	X	X		X			X			X	X	X	X	X				X			
21	KNN 1	X		X	X			X											X			
22	KNN 3	X		X	X			X											X			
23	KNN 5	X		X	X			X											X			
24	RBF 3	X		X	X			X											X			
25	KNN 1								X		X	X	X	X					X			
26	KNN 3								X		X	X	X	X					X			
27	KNN 5								X		X	X	X	X					X			
28	RBF 2								X		X	X	X	X					X			
29	RBF 3								X		X	X	X	X					X			
30	RBF 4								X		X	X	X	X					X			

Tabelle 5
Verwendete Feature-Kombinationen in der ersten Experimental-Serie
(Referenz-Feature-Datensätze sowie heuristische Feature-Sätze)

In Tabelle 5 ist der Aufbau der durchgeführten Experimente beschrieben. In der ersten Spalte sind die Experimente mit den Ziffern durchnummeriert, mit denen sie in den folgenden Diagrammen beschriftet sind. In der zweiten Spalte stehen die im jeweiligen Experiment verwendeten Klassifikatoren. KNN steht dabei für k-Nearest-Neighbor und RBF für RBF-Network. Die Zahlen geben die Anzahl der betrachteten Nachbarn bzw. die Anzahl der verwendeten Radialen Basisfunktionen an. Die folgenden 21 Spalten geben die verwendeten Features an, die in Tabelle 4 aufgelistet sind.

Das Experiment 25 verwendet die Features des Referenz-Feature-Satzes und den einfachsten der genutzten Klassifikatoren. Es wird im weiteren Verlauf dieser Arbeit als „Referenz-Experiment“ herangezogen werden.

Die Experimente in diesem Kapitel wurden mit dem RBF Klassifikator aus yale/rapidminer und der eigenen Implementation des KNN Klassifikators durchgeführt.

Die Resultate der Experimente sind in den folgenden Diagrammen dargestellt.

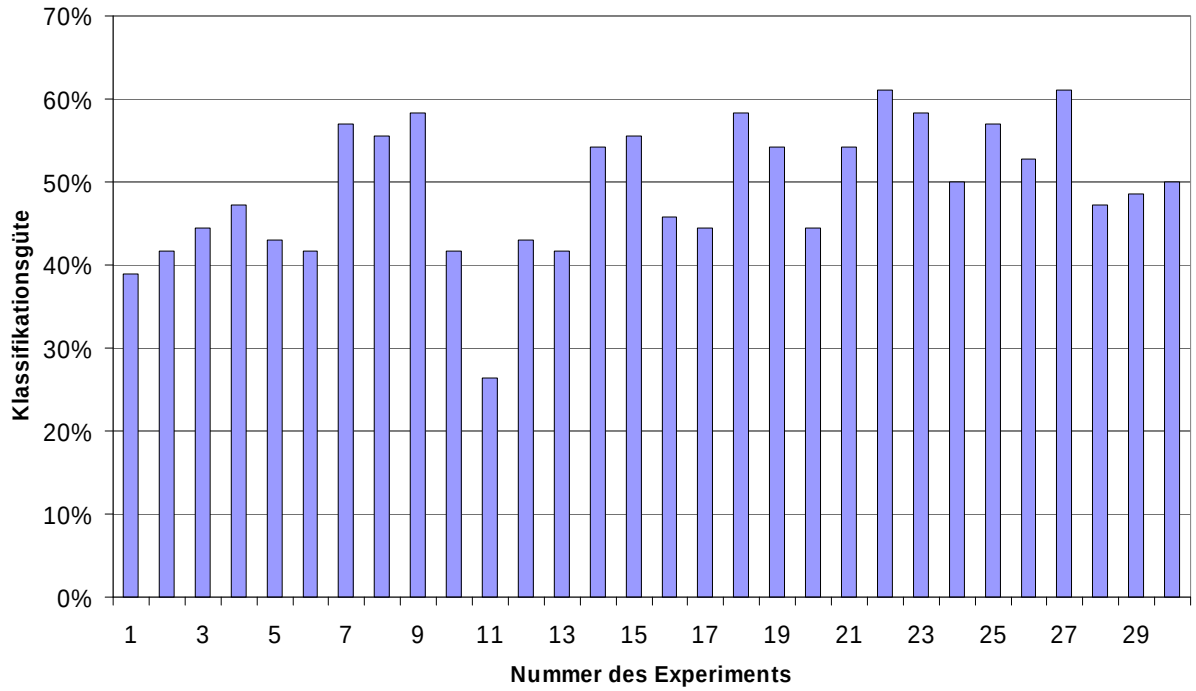


Abbildung 4-1
Klassifikationsgüten der Experimente aus Tabelle 5, für den Test-Datensatz „Satz 1“ (6 Genres, 72 Musikstücke)

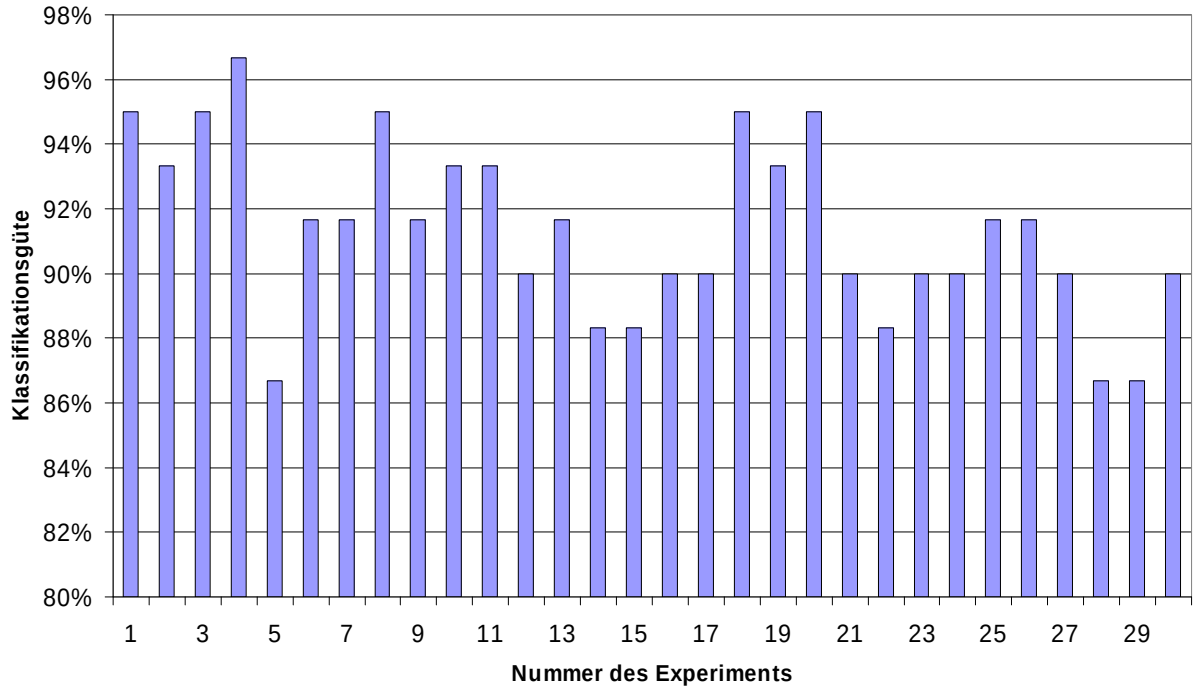


Abbildung 4-2
Klassifikationsgüten der Experimente aus Tabelle 5, für den Test-Datensatz „Satz 2“ (2 Genres, 60 Musikstücke)

In den Diagrammen Abbildung 4-1 und Abbildung 4-2 sind die Ergebnisse der durchgeführten Experimente dargestellt. Das Experiment in Abbildung 4-1 wurde auf den sechs Genres des Testdatensatzes „Satz 1“ durchgeführt, das Experiment in Abbildung 4-2 auf den zwei Genres des Test-Datensatzes „Satz 2“. Die Ergebnisse der Experimente mit sechs Genres liegen bis auf einen Ausreißer durchweg über 39% und erreichen bis zu 61%. Ihre Klassifikationsgüte ist damit bis zu dreieinhalb Mal besser als eine rein zufällige Entscheidung für ein Genre. Bei Benutzung des KNN-Klassifikators scheint sich die Klassifikationsgüte mit zunehmender Anzahl betrachteter Nachbarn zu verbessern. Die Ausnahme von dieser Regel bilden die Experimente 21-23, bei denen der Klassifikator, der die meisten Nachbarn betrachtet, wieder schlechter wird. Einen leichten Einbruch dieser zunehmenden Klassifikationsgüte zeigen die Experimente 12-13 und 25-27, bei denen der Klassifikator für drei Nachbarn leicht schlechtere Ergebnisse liefert als derjenige, der nur einen Nachbarn betrachtet.

Die Aufgabe, zwei Genres zu unterscheiden, lösen alle Versuche mit einer Klassifikationsgüte von mindestens 86%. Die steigende Klassifikationsgüte mit zunehmender Anzahl betrachteter Nachbarn lässt sich hier nur in den Experimenten 2-4 beobachten. Eine Verschlechterung der Klassifikationsgüte für den Klassifikator, der die meisten Nachbarn betrachtet, ist in den Experimenten 12-14 und 25-27 zu beobachten.

Aufgrund der stark nichtlinearen Zusammenhänge unterscheiden sich die Klassifikationsergebnisse teils deutlich von den Literaturangaben. Verglichen mit dort aufgeführten Ergebnis-

sen sind die Resultate der hier durchgeführten Experimente mit sechs Genres häufig schlechter. [16] und [19] verwenden deutlich mehr Genres, klassifizieren diese aber mit Güten, die 5% bis 25% besser sind. Allerdings verwenden beide Experimente deutlich aufwändigere Methoden zur Klassifikation. Das Experiment von [18] verwendet nur die MFCC Koeffizienten zur Klassifikation von sieben Genres, wobei das siebte Genre eine Sammlung von „sonstiger Musik“ ist, die nicht zu den sechs anderen Genres passt. Die Klassifikation erfolgt dort für die Koeffizienten eines jeden Zeitfensters, gefolgt von einer Zusammenfassung der Ergebnisse zu einem Resultat. Die Experimente fünf bis sieben stellen diesen Versuch von allen betrachteten Experimenten aus der Literatur am besten nach. Es stehen die gleichen Features zur Verfügung und das RBF-Network stellt eine gute Näherung des GMM-Klassifikators dar, jedoch erfolgt die Klassifikation anhand von Mittelwert und Varianz der MFCC Koeffizienten und nicht individuell für jedes Zeitfenster. Die beste Klassifikationsgüte dieser nachgestellten Experimente liegt bei 58%, während die Angaben in der Literatur mit mindestens 79% um etwa die Hälfte besser liegen. Die Unterschiede lassen sich möglicherweise auf eine günstigere Auswahl der Musikstücke und Genres oder auf Unterschiede in den komplexen Klassifikatoren zurückführen.

In [20] werden sechs Genres unter Verwendung von 14 Features unterschieden, wobei abhängig vom verwendeten Klassifikator eine Klassifikationsgüte von bis zu 58% bzw. 68% erreicht wurde. Unter Nutzung von nur fünf Features wird hier dagegen in einem ähnlichen Experiment und dem KNN 5 Klassifikator eine Klassifikationsgüte von 62% erreicht.

Für die Klassifikation von zwei Genres übertreffen die Klassifikationsgüten der durchgeführten Experimente die Angaben aus der Literatur. Sowohl [17] als auch [21] verwenden KNN-Klassifikatoren, deren Klassifikationsgüten im Schnitt unter den hier erzielten liegen. Nur die komplexere SVM in [21] erreicht eine Klassifikationsgüte, die mit den besten KNN mithalten kann.

Bei der Klassifikation einer größeren Anzahl an Genres gibt es deutliche Unterschiede. Ein allgemeiner Klassifikator, der ohne a priori Wissen eine Einteilung der Genres vornehmen muss, kann in der Klassifikationsgüte gegenüber den auf eine spezielle Situation ausgerichteten Klassifikatoren nicht mithalten. Bei nur zwei Genres zeigt sich aber, dass mit geeigneten Features eine gute Klassifikationsgüte erreicht werden kann.

4.4. Nichtparametrischer Statistiktest

Die Klassifikationsgüte ist offenbar sehr stark abhängig von dem verwendeten Feature-Satz. Für ein optimales Klassifikationsergebnis ist daher eine passende Feature-Auswahl entschei-

dend. Der Einfluss einzelner Features auf die Klassifikationsgüte ist allerdings weitestgehend unbekannt. Es ist daher nicht ohne weiteres möglich, einen optimalen Feature-Satz zu wählen. Insbesondere interessiert aber die Fragestellung, ob die im letzten Kapitel betrachteten Feature-Kombinationen bereits eine gute Wahl darstellen oder ob es Feature-Kombinationen gibt, die bessere Klassifikationsgüten erreichen.

Zu Bestimmung geeigneter Feature-Sätze gibt es mehrere Alternativen. Es kommen zum Beispiel Verfahren zur Pareto-Optimierung [53] in Frage, statistische Tests oder auch eine vollständige Suche.

4.4.1. Vollständige Suche

Bei Nutzung einer vollständigen Suche sind $2^n - 1$ Feature-Sätze auf die mit ihnen erreichbare Klassifikationsgüte zu untersuchen, wenn n die Anzahl der betrachteten Features darstellt. Die Laufzeit eines individuellen Suchlaufs ergibt sich durch Multiplikation mit der durchschnittlichen Dauer eines einzelnen Tests des entsprechenden Klassifikators. Sollen mehrere Klassifikatoren getestet werden, so sind für die Gesamtdauer die Laufzeiten ihrer Suchläufe zu addieren. Die Gesamte Laufzeit steigt also exponentiell mit der Anzahl der Features und annähernd linear mit der Anzahl der Klassifikatoren.

Im Rahmen dieser Arbeit wurden 19 Features implementiert. Diese sind in Tabelle 6 aufgelistet. Bei einer Laufzeit von etwa 0,06 Sekunden für einen Test (P4, 3,2 GHz) ergibt sich so für die vollständige Suche eine Gesamtlaufzeit von knapp neun Stunden. Um die optimalen Feature-Sätze zu ermitteln wurde diese Suche mit dem Test-Datensatz „Satz 1“ und einer eigenen Implementation des k-Nearest-Neighbor Klassifikators für einen, drei, fünf und sieben Nachbarn durchgeführt. Die Gesamtlaufzeit der vollständigen Suche betrug damit etwa 36 Stunden. Für jeden der Klassifikatoren ergaben sich mehrere Feature-Sätze, die eine identische beste Klassifikationsgüte aufwiesen. Für die Darstellung der Ergebnisse in Tabelle 7 wurde jeweils ein Feature-Satz ausgewählt, der sich aus möglichst wenigen Features zusammensetzt.

Nummer	Featurename
F1	Spectral Centroid
F2	Spectral Rolloff
F3	Spectral Rolloff 2
F4	Spectral Flux
F5	Spectral Extend
F6	Spectral Bandwidth
F7	low Energy Windows
F8	RMS
F9	Spectral Kurtosis
F10	Sum of Correlated Components
F11	Zero Crossing Rate
F12	Relative Periodicity Amplitude Peaks (1-2)
F13	Ratio of Second and First Periodicity Peak
F14	Position of Main Peaks (1-5)
F15	Power Spectrum (0-1)
F16	MFCC (0-12)
F17	Fundamentalfrequenz
F18	Amplitude of Maximum in Chromagram
F19	Pitch Intervall between two max Peaks in Chromagram

Tabelle 6 Die in allen Experimenten verwendeten Features

Klassifikaotr	Feature																			Klassifikationsgüte
	F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	
KNN 1		X				X	X		X							X	X			**66,7%**
KNN 3	X	X		X							X					X				**66,7%**
KNN 5		X		X		X		X				X				X	X			**70,8%**
KNN 7							X			X		X	X		X	X	X			**69,4%**

Tabelle 7 Ergebnisse der vollständigen Suche für 6 Genres

Es zeigt sich, dass die MFCC-Koeffizienten in allen Fällen zu den Feature-Sätzen gehören, die die besten Klassifikationsgüten erreichen. Offenbar sind die MFCC-Koeffizienten also ein Feature, das einen wichtigen Einfluss auf die Klassifikationsgüte hat. Bei den weiteren Features sind keine derart auffälligen Regelmäßigkeiten zu erkennen. Lediglich Spectral Centroid, Spectral Flux, Sum of Correlated Components sowie Fundamentalfrequenz treten häufiger auf, so dass vermutet werden kann, dass sie einen wichtigen Einfluss haben.

Für die hier verwendete geringe Anzahl von Features war die vollständige Suche noch durchführbar, doch durch ihr exponentielles Wachstum sind die Grenzen einer vertretbaren Laufzeit

schnell erreicht. Bereits bei 30 Features und einer (festen) Laufzeit von beispielsweise 0,1 Sekunden pro Test würde eine vollständige Suche knapp dreieinhalb Jahre dauern. Für die in [46] beschriebenen 55 Features währen es bereits mehrere Millionen Jahre.

Für umfangreichere Tests müssen folglich andere Verfahren verwendet werden. Im folgenden Kapitel wird daher ein statistischer Test vorgestellt, der nicht nur eine erheblich geringere Laufzeit aufweist als die vollständige Suche, sondern auch gegenüber einer Pareto-Optimierung Signifikanz-Hinweise für die einzelnen Features hinsichtlich ihres Einflusses auf die Klassifikationsgüte liefert.

4.4.2. Induktive Statistik

Im wissenschaftlichen Umfeld zielen die meisten Experimente darauf ab, eine Vorhersage oder Schätzung einer Eigenschaft zu unterstützen. Diese Vorhersage wird *experimentelle Hypothese* oder *Alternativhypothese* genannt, über deren Wahrheitsgehalt die induktive Statistik eine Aussage treffen will. Im gegebenen Fall lautet die

Experimentelle Hypothese: Feature A ist geeignet die Klassifikationsgüte zu verbessern.

Im Folgenden wird kurz die grundlegende Vorgehensweise in der induktiven Statistik beschrieben. Man beginnt mit einer Nullhypothese, welche die Aussage der experimentellen Hypothese umkehrt

Nullhypothese: Feature A hat keinen Einfluss auf die Klassifikationsgüte

Zur Durchführung des statistischen Tests benötigt man zwei Gruppen, eine Experimentalgruppe und eine Kontrollgruppe. In der Experimentalgruppe wird das Feature A verwendet, während es in der Kontrollgruppe nicht zum Einsatz kommt. Die Nullhypothese sagt aus, dass die Klassifikationsgüte in beiden Gruppen nicht voneinander abweicht. Kann man aber nun annehmen, dass die Klassifikationsgüte in beiden Gruppen signifikant voneinander abweicht, so lässt sich die Nullhypothese zurückweisen und die Alternativhypothese annehmen.

Abschließend wird eine Methode benötigt, um die Differenzen zwischen den Gruppen abzuschätzen.

4.4.3. Experimentalgruppen

Um die Hypothesen zu testen, benötigt man eine *Experimentalgruppe* und eine *Kontrollgruppe*. Diese könnten für jedes Feature, das getestet werden soll, einzeln definiert werden. Das Resultat wären allerdings sehr viele Experimente, so dass es günstiger erscheint, eine Experimentalgruppe und eine Kontrollgruppe zu erzeugen, die zum Testen aller Features geeignet sind. Das bedeutet, dass eine möglichst zufällige Zusammenstellung der Feature-

Kombinationen erzeugt werden muss. Auf diese Weise kann die Experimentalgruppe für jedes Feature aus den Feature-Kombinationen zusammengestellt werden, in denen das Feature verwendet wird, während die Kontrollgruppe aus denjenigen besteht, in denen das Feature nicht verwendet wird. Damit trägt das Klassifikationsergebnis jedes einzelnen Versuchs entweder zur Experimental- oder zur Kontrollgruppe für ein Feature bei. Um dies zu erreichen, benutzt man ein so genanntes *Orthogonales Array*.

Ein Orthogonales Array (OA) ist eine Matrix aus Einsen und Nullen, die in jeder Zeile und Spalte gleich häufig vorkommen. Darüber hinaus treten auch alle Kombinationen aus Einsen und Nullen, die sich z.B. über mehrere Spalten hinweg bilden lassen, gleich häufig auf.

Die Reihen des Orthogonalen Arrays repräsentieren einzelne Experimente, während die einzelnen Spalten jeweils einzelne Features darstellen, deren Einfluss untersucht werden soll.

Somit wird jedes Feature in den Experimenten gleich häufig genutzt und nicht genutzt. Des Weiteren werden in den Experimenten, in denen ein Feature aktiv ist, alle anderen Features ebenfalls gleich oft genutzt und nicht genutzt. Wenn die Matrix N Reihen hat, existieren damit $½ N$ Reihen in denen das Feature genutzt wird und $½ N$ Reihen in denen es nicht verwendet wird.

Bei Benutzung des Orthogonalen Arrays können somit Versuche durchgeführt werden, die sich im sogenannten Mann-Whitney Test für alle Features eignen. Da die Werte für alle nicht betrachteten Features gleichmäßig verteilt sind, lässt sich der Einfluss eines Features in beliebigem Zusammenhang betrachten.

Das in diesem Versuch verwendete Orthogonale Array stammt von [37].

4.4.4. Mann-Whitney Test

Da die einzelnen Experimente sehr verschiedene Feature-Kombinationen verwenden und aufgrund des begrenzten Stichprobenumfangs nicht zwangsläufig repräsentativ sind, können die Klassifikationsergebnisse stark voneinander abweichen. Es ist daher nicht angebracht, direkt mit den Ergebnissen der Experimente zu arbeiten, da (geringe) Unterschiede durch reinen Zufall entstehen können. Nichtparametrische Tests sind speziell für diese Situationen entworfen worden. Sie können die Daten untersuchen, ohne die zugrunde liegende Verteilungsfunktion zu kennen, indem sie die Testergebnisse nach der Güte sortieren und auf den Platzierungen weiterarbeiten.

Der Mann-Whitney Test ist ein bekannter Test der induktiven, nichtparametrischen Statistik.

Nachfolgend wird er an einem kurzen Beispiel erklärt.

Tabelle 8 zeigt dabei einen möglichen Versuchsaufbau und die Klassifikationsgüten für ein Klassifikationsexperiment mit elf Musik-Features. Die ersten elf Spalten stehen jeweils für

ein Feature, die zwölf Zeilen repräsentieren jeweils ein Featureset. Die zwölfte Spalte gibt die Klassifikationsgüte bei Benutzung des jeweiligen Featuresets an. In der letzten Spalte steht die Platzierung des Ergebnisses in absteigender Reihenfolge.

F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	Güte	Rang
1	1	1	1	1	1	1	1	1	1	1	68 %	6
0	1	0	1	1	1	0	0	0	1	0	58 %	9
0	0	1	0	1	1	1	0	0	0	1	47 %	12
1	0	0	1	0	1	1	1	0	0	0	78 %	1
0	1	0	0	1	0	1	1	1	0	0	54 %	10
0	0	1	0	0	1	0	1	1	1	0	73 %	3
0	0	0	1	0	0	1	0	1	1	1	64 %	7
1	0	0	0	1	0	0	1	0	1	1	72 %	4
1	1	0	0	0	1	0	0	1	0	1	62 %	8
1	1	1	0	0	0	1	0	0	1	0	70 %	5
0	1	1	1	0	0	0	1	0	0	1	51 %	11
1	0	1	1	1	0	0	0	1	0	0	75 %	2

Tabelle 8
Beispielhafter Versuchsaufbau und Ergebnisse für ein Klassifikationsexperiment mit 11 Features

Nun soll ermittelt werden, ob Feature F1 einen Einfluss auf die Klassifikationsgüte hat. Dazu wird das Experiment in Experimentalgruppe und Kontrollgruppe aufgeteilt. Die Experimentalgruppe setzt sich aus den Experimenten zusammen, in denen das Feature F1 genutzt wurde, bei denen in Tabelle 8 also eine eins in der Spalte von F1 steht. Die Kontrollgruppe setzt sich entsprechend aus den Experimenten zusammen, in denen das Feature F1 nicht verwendet wurde. Tabelle 9 zeigt die Ränge für diese Gruppen. In der letzten Zeile sind die Ränge der jeweiligen Gruppe aufsummiert, hier als T1 und T2 bezeichnet. T1 und T2 unterscheiden sich offensichtlich, doch ist der Unterschied groß genug, um auf einen signifikanten Einfluss von F1 zu schließen?

Der Mann-Whitney Test basiert auf der Summe der Ränge der Experimentalgruppe, hier also dem Wert von T1.

	Experimentalgruppe (F1 = 1)	Kontrollgruppe (F1 = 0)
	6	9
	1	12
	4	10
	8	3
	5	7
	2	11
Summe	T1 = 26	T2 = 52

Tabelle 9
Einfluss des Features F1

Im Folgenden sei zur Vereinfachung angenommen, dass der Umfang der Gruppen *N* Elemente beträgt und sich beide Gruppen nur durch das zu untersuchende Feature unterscheiden. Des Weiteren habe das Feature keinen Einfluss auf die Klassifikationsgüte, die Nullhypothese sei also wahr. In diesem Falle wäre die Rangfolge der Klassifikationsgüten rein zufällig, verursacht durch zufällige Schwankungen. Damit läge der Wert von T1 zwischen minimal $1+2+...+N=\frac{N}{2}\cdot(N+1)$ und maximal $(N+1)+...+(2N)=N^2+\frac{N}{2}\cdot(N+1)$. Werte dazwischen sind aufgrund der vielfältigeren Kombinationsmöglichkeiten wahrscheinlicher. Es kann gezeigt werden, dass T1 eine Normalverteilung besitzt, falls die Nullhypothese wahr ist, mit dem Mittelwert

$$\mu=\frac{N(2N+1)}{2}$$

und der Standardabweichung

$$\sigma=\sqrt{\frac{N^2(2N+1)}{12}}$$

Auf ein normalverteiltes T1 lassen sich nun klassische Methoden der Statistik anwenden.

Allerdings wird nicht T1 direkt betrachtet, sondern die statistische Variable *z*, welche sich aus T1 umgerechnet auf eine Standardnormalverteilung ergibt.

$$z=\frac{T_1-\mu}{\sigma}$$

mit der Verteilungsfunktion

$$Y(z)=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{1}{2}z^2}$$

welche die Wahrscheinlichkeit angibt, einen bestimmten Wert von *z* zu messen. Eine graphische Darstellung der Funktion ist in Abbildung 4-3 gegeben.

Weicht ein Wert von T1 signifikant von μ ab (bzw. der korrespondierende Wert z von Null), so kann daraus geschlossen werden, dass die Nullhypothese falsch ist, da es sehr unwahrscheinlich ist, dass sich ein solcher Wert rein zufällig ergibt.

Nimmt man die Funktion P(t) an so drückt diese die Wahrscheinlichkeit aus, einen Wert für z zu erhalten, so dass entweder $z \leq -t$ oder $z \geq t$.

$$P(t) = \left(1 - 2 \cdot \int_0^t Y(z)\right) \cdot 100\%$$

Ein Standardmaß für einen „signifikanten Unterschied“ ist gegeben durch eine Wahrscheinlichkeit von weniger als 5% einen bestimmten Wert von z zu erhalten. Dieser Grenzwert von 5% wird das Signifikanzniveau genannt. Der entsprechende Wert für t ist 1,96, wie in Abbildung 4-3 eingezeichnet. Die Wahrscheinlichkeit einen Wert von z zu erhalten, der in dem markierten Bereich liegt ist also kleiner als 5%.

Damit ist die Chance einen Fehler 1. Art zu machen, also die Nullhypothese zurückzuweisen obwohl sie stimmt, kleiner als 5%.

Zusammenfassend ermittelt der Mann-Whitney Test die Summe der Ränge für die Experimentalgruppe und daraus den entsprechenden Wert für z. Anschließend wird der Wert für $P(|z|)$ berechnet und geprüft ob dieser unter 5% liegt[4]. Ist dies der Fall, so kann geschlossen werden, dass die Experimentalgruppe signifikant von der Kontrollgruppe abweicht und die Nullhypothese wird zurückgewiesen.

Im Beispiel ist N = 6 und T1 = 28. Damit berechnet sich der Wert für z folgenderweise:

$$\mu = \frac{6(2 \cdot 6 + 1)}{2} = 39$$

$$\sigma = \sqrt{\frac{6^2(2 \cdot 6 + 1)}{12}} = \sqrt{39}$$

$$z = \frac{(26 - 39)}{\sqrt{39}} = -2{,}08$$

Da z < -1,96 ist, ist auch $P(|z|) < 5\%$. Daraus lässt sich schließen, dass das Feature F1 einen signifikanten Einfluss auf die Klassifikationsgüte besitzt. Da z negativ ist, bedeutet dies bessere Klassifikationsgüten als der Durchschnitt, das Feature F1 hat also einen positiven Einfluss.

Dieses Vorgehen kann für jedes Feature der Tabelle angewendet werden. Damit lassen sich die signifikanten Features der elf Features mit nur 12 Experimenten bestimmen.

[4] Tatsächlich kann vereinfachend überprüft werden, ob $|z| > 1{,}96$ ist, aber formal erfolgt die Überprüfung, ob $P(|z|) < 5\%$ ist.

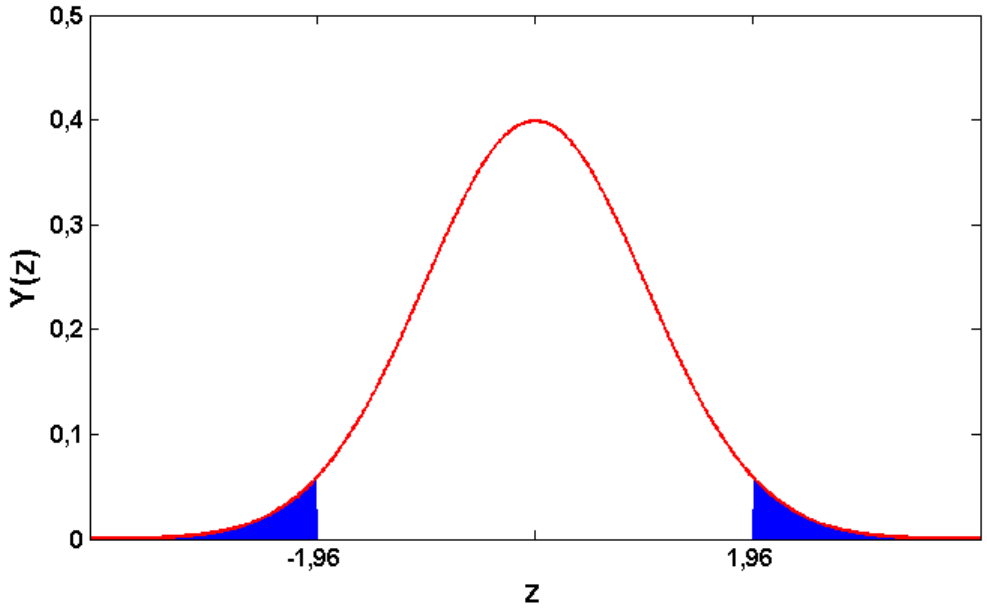


Abbildung 4-3
Normalverteilung von z

4.4.5. Stichprobenumfang

Neben dem Fehler 1. Art gibt es noch einen Fehler 2. Art, der berücksichtigt werden muss.

Der Fehler 1. Art besteht darin, die Nullhypothese zurückzuweisen, obwohl sie wahr ist. Die Chance für einen solchen Fehler wurde im vorherigen Abschnitt auf weniger als 5% festgelegt.

Ein Fehler 2. Art besteht darin, die Nullhypothese nicht zurückzuweisen, obwohl sie falsch ist. Er soll hier maximal 20 % betragen, was generell für den Mann-Whitney Test als ausreichend angesehen wird [35]. Dieser Fehler beeinflusst zusammen mit dem maximalen Fehler 1. Art den erforderlichen Umfang des Testes.

Seien z_α und z_β die z-Werte, die in $P(|z|)$ die angegebenen Fehlergrenzen ergeben. Dann berechnet sich die erforderliche Stichprobengröße M zu

$$M = \left\lceil 5 \cdot \left(z_\alpha + z_\beta \right)^2 \right\rceil$$

Für die geforderten Fehlerwahrscheinlichkeiten ergeben sich $z_\alpha = 1.96$ und $z_\beta = 1.28$, womit die Stichprobengröße mindestens 53 beträgt.

Das Orthogonale Array wird in der Größe 56x28 gewählt. Der Fehler 2. Art beträgt damit nur rund 16%.

4.4.6. Durchführung

Im Rahmen dieser Arbeit wurden 19 verschiedene Features implementiert, die in dem Mann-Whitney Test untersucht wurden. Je nach Definition umfassen diese Features jeweils eine oder mehrere Komponenten, von denen jeweils Mittelwert und Varianz gebildet wurden.

Entsprechend den Einträgen der Spalten, die den jeweiligen Optionen entsprechen, wurde die Feature-Auswahl für die einzelnen Versuche zusammengestellt. Mit Hilfe der resultierenden Klassifikationsgüten der Versuche wurden dann die signifikanten Features ermittelt. Diese so ermittelten Features wurden anschließend festgesetzt und die verbleibenden Optionen in weiteren Iterationen wieder untersucht.

Die in einer Iteration als signifikant erkannten Features verbessern die mit ihnen erreichbare Klassifikationsgüte jedoch nicht zwangsläufig, sondern haben nur unter den jeweils gegebenen Randbedingungen einen signifikanten Einfluss. Wäre z.B. in einer Iteration der optimale Feature-Satz gefunden worden, so würde der Mann-Whitney Test unter den verbleibenden Features wieder diejenigen mit einem signifikanten Einfluss auf die Klassifikationsgüte bestimmen, obwohl die Klassifikationsgüten alle schlechter geworden sind.

Insgesamt wurden fünf Iterationen durchgeführt, als Datensatz dienten die aus dem Musikdatensatz „Satz 1“ extrahierten Features. Der Mann-Whitney Test wurde dreimal durchgeführt, mit der selber implementierten Version des k-Nearest-Neighbour Klassifikator für ein, drei und fünf Nachbarn.

In Tabelle 6 sind die verwendeten Features aufgelistet. Tabelle 10 stellt die Ergebnisse des Mann-Whitney Tests dar.

In der ersten Spalte stehen die durchgeführten Iterationen, getrennt nach den Klassifikatoren. Die folgenden 21 Spalten stellen die Ergebnisse für die einzelnen Features dar. Eine fett gedruckte rote Zahl bedeutet, dass dieses Feature in der jeweiligen Iteration als signifikant erkannt wurde. Bei einer fett gedruckten schwarzen Zahl wurde dieses Feature in der Iteration festgehalten und nicht mehr untersucht, da es in einer vorherigen Iteration als signifikant erkannt wurde. Der Fall, dass ein Feature in einem anderen enthalten ist, aber nicht explizit selber erkannt wurde, ist durch eine graue fette Zahl gekennzeichnet. Bei einer eins wurde das Feature als eines erkannt, das einen positiven Einfluss auf die Klassifikationsgüte hat, bei einer null dagegen als eines, das einen negativen Einfluss ausübt. In der letzten Spalte sind die Klassifikationsgüten dargestellt, die mit dem jeweiligen Klassifikator und dem in der Iteration ermittelten Featuresets erreicht werden. Insgesamt wurden für jeden der Klassifikatoren fünf Iterationen durchgeführt.

Experiment	Iteration	Feature																			Klassifikationsgüte
		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	
	KNN 1																				
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	54,2%
2	2	0	0	1	1	0	1	0	0	0	1	0	0	0	0	0	1	1	1	0	54,2%
3	3	0	1	1	1	0	1	1	1	1	1	1	0	1	0	1	1	1	1	1	59,7%
4	4	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	56,9%
5	5	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	55,6%
	KNN 3																				
6	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	52,8%
7	2	1	0	0	1	0	1	0	0	1	1	0	0	0	0	1	1	0	0	0	58,3%
8	3	1	0	1	1	0	1	1	1	1	1	1	0	0	0	1	1	0	0	0	58,3%
9	4	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	0	0	1	61,1%
10	5	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	0	1	61,1%
	KNN 5																				
11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	55,6%
12	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	59,7%
13	3	0	1	0	1	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	61,1%
14	4	0	1	0	1	0	0	1	0	0	0	0	1	1	0	1	1	1	0	0	65,3%
15	5	0	1	0	1	0	0	1	0	0	0	0	1	1	0	1	1	1	0	0	65,3%

Tabelle 10
Die Ergebnisse des Mann-Whitney Tests

Unabhängig von dem verwendeten Klassifikator werden die MFCC Koeffizienten in der ersten Iteration des Mann-Whitney Tests als signifikante Features identifiziert. Dies ist sehr plausibel, da die MFCC-Koeffizienten auch in der Literatur als bedeutsam gelten und bereits seit langem in der Sprachverarbeitung genutzt werden. Auch bei der Signifikanz weiterer Features sind Ähnlichkeiten bei allen drei betrachteten Klassifikatoren festzustellen. Der Spectral flux, das Power Spectrum und der Spectral Rolloff werden früh als wichtige Features erkannt. Die Position of Main Peaks dagegen wird sehr unterschiedlich beurteilt. Während bei den KNN 1 und KNN 5 Klassifikatoren ein signifikant negativer Einfluss auf die Klassifikationsgüte ermittelt wird, wird dieses Feature für den KNN 3 Klassifikator als signifikant positiv erkannt. Eine ganze Reihe weiterer Features wird für die KNN 1 und KNN 3 Klassifikatoren als relevant erkannt, während sie auf die Klassifikationsgüte des KNN 5 Klassifikators keinen Einfluss zu haben scheinen. In der fünften Iteration des Mann-Whitney Test für KNN 5 wird kein zusätzliches Feature als signifikant ermittelt.

Die beste erreichte Klassifikationsgüte mit den im Mann-Whitney Test ermittelten Feature-Sätzen beträgt 65%, die mit acht Features und dem KNN 5 Klassifikator erreicht werden kann. Sie liegt damit nur 5% unter der besten Klassifikationsgüte, die in der vollständigen Suche für diesen Klassifikator ermittelt wurde.

Eine genauere Analyse der internen Zwischenergebnisse der Testläufe mit dem KNN 1 und dem KNN 3 Klassifikator zeigt, dass der Umfang des Testdatensatzes einen entscheidenden Einfluss hat. Besteht der Testdatensatz aus zu wenigen Musikstücken, so wiederholen sich Klassifikationsgüten einzelner Experimente und es kann keine eindeutige Rangfolge der Ergebnisse erstellt werden. Durch diese nicht eindeutige Rangfolge leidet in der Folge die Aussagekraft über die Signifikanz einzelner Features.

Die Laufzeit des Mann-Whitney Tests ist erheblich geringer als die Laufzeit einer vollständigen Suche. Trotz mehrerer Iterationen betrug die Anzahl der durchzuführenden Experimente für die betrachteten 19 Features weniger als ein Promille gegenüber der vollständigen Suche. Bei zunehmender Anzahl zu untersuchender Features wird dieser Anteil weiter sinken. Die Laufzeit für eine Iteration lag bei wenigen Sekunden.

Es konnten mehrere Features ermittelt werden, die einen signifikanten Einfluss auf die Klassifikationsgüte haben. Einige dieser Features gehörten zu den häufigsten Features, die auch bei der vollständigen Suche ermittelt wurden.

Eine vollständige Suche liefert immer das optimale Resultat, daher ist diese zu bevorzugen, wenn dies möglich ist. Für Tests, für die eine vollständige Suche zu aufwändig ist, kann der Mann-Whitney Test verwendet werden. Wie gezeigt wurde, liefert er bei geringer Laufzeit gute Hinweise auf signifikante Features, die bereits gute Klassifikationsergebnisse liefern und als Ausgangsbasis für weitere Untersuchungen dienen können.

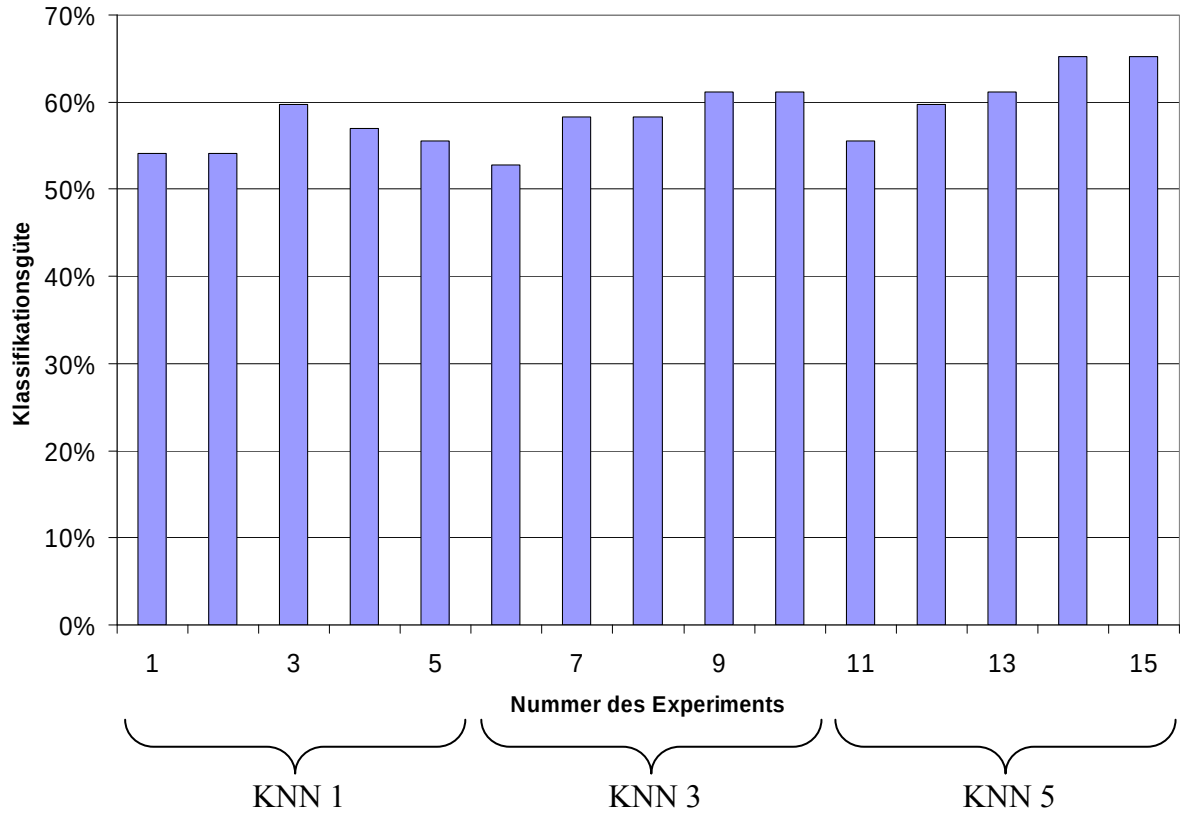


Abbildung 4-4
Klassifikationsgüten der aus dem Mann-Whitney Test ermittelten Feature-Kombinationen mit sechs Genres

Abbildung 4-4 stellt die Ergebnisse des Mann-Whitney Test graphisch dar. Auf der Abszisse sind die Experimente in der Reihenfolge durchnummeriert, wie sie in Tabelle 10 aufgelistet sind, die Experimente eins bis fünf sind also die fünf Experimente, die mit dem KNN 1 Klassifikator durchgeführt wurden.

In Abbildung 4-5 wurden die gleichen Experimente angewendet, um die zwei Genres des Test-Datensatzes „Set 3“ zu klassifizieren. Obwohl die Feature-Kombinationen nicht für diese Klassifikationsaufgabe bestimmt wurden sind die Ergebnisse sehr gut, die minimale und maximale Klassifikationsgüte liegt mit 88% und 98% höher als die in Abbildung 4-2 bestimmten Klassifikationsgüten. Der im Mann-Whitney Test ermittelte Feature-Satz liefert also auch für die Unterscheidung anderer Genres gute Ergebnisse.

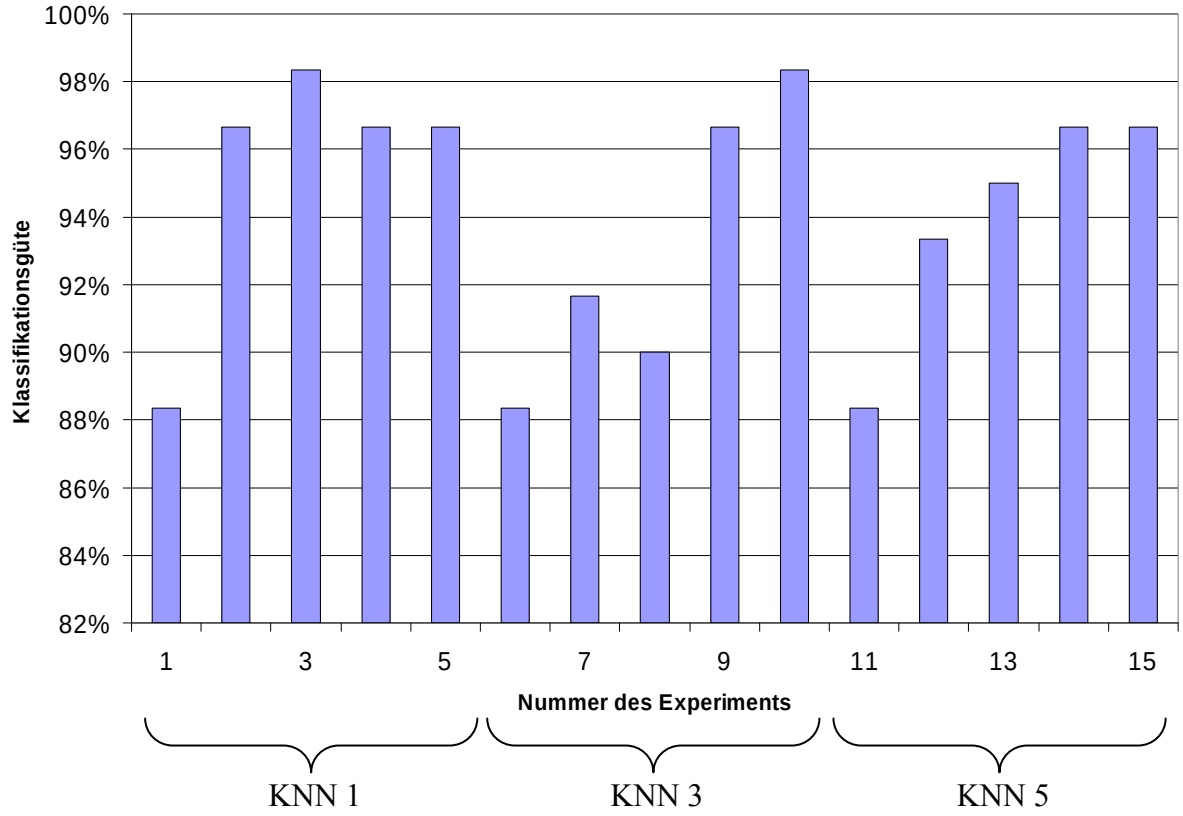


Abbildung 4-5
Klassifikationsgüten der aus dem Mann-Whitney Test ermittelten Feature-Kombinationen mit zwei Genres

4.5. Vertrauensmaße

Ein Klassifikator liefert üblicherweise diejenige Klasse, die als die wahrscheinlichste ermittelt wurde. Doch nicht immer ist die Zuordnung für den Klassifikator eindeutig, es gibt Grenzfälle, in denen eine suboptimale Entscheidung gefällt wird.

Unterscheidet sich die Ähnlichkeit zu zwei mögliche Klassen nur minimal, so könnte es sinnvoller sein, eine Zuordnung als „zu unsicher“ abzulehnen, anstatt eine zweifelhafte Entscheidung zu treffen.

Im Ergebnis wäre damit zu rechnen, dass sich der Anteil der richtig zugeordneten Lieder vergrößert, während dafür einige Lieder als „nicht sicher klassifizierbar“ übrig bleiben würden.

4.5.1. Vertrauensmaß für k-Nearest-Neighbor

Bei dem KNN-Klassifikator gilt eine Entscheidung als eindeutig, wenn unter den *k* nächstgelegenen Nachbarn im Vektorraum, der von den verwendeten Features aufgespannt wird, eine Klasse die absolute Mehrheit hat.

Doch es gibt auch Fälle, in denen mehrere Klassen gleich häufig auftreten. So könnte bei einem KNN-Klassifikator, der sieben Nachbarn betrachtet, je dreimal die Klassen A und B und einmal die Klasse C auftreten, wie in Abbildung 4-6 dargestellt. In diesen Fällen muss zwischen den Klassen der größten Häufigkeit entschieden werden, in diesem Beispiel also A und B. Es bietet sich an, von diesen beiden Klassen jeweils den Vektor zu bestimmen, der die geringste Distanz zum zu testenden Vektor hat. Ausgewählt wird dann die Klasse, dessen Vektor den geringeren Abstand hat.

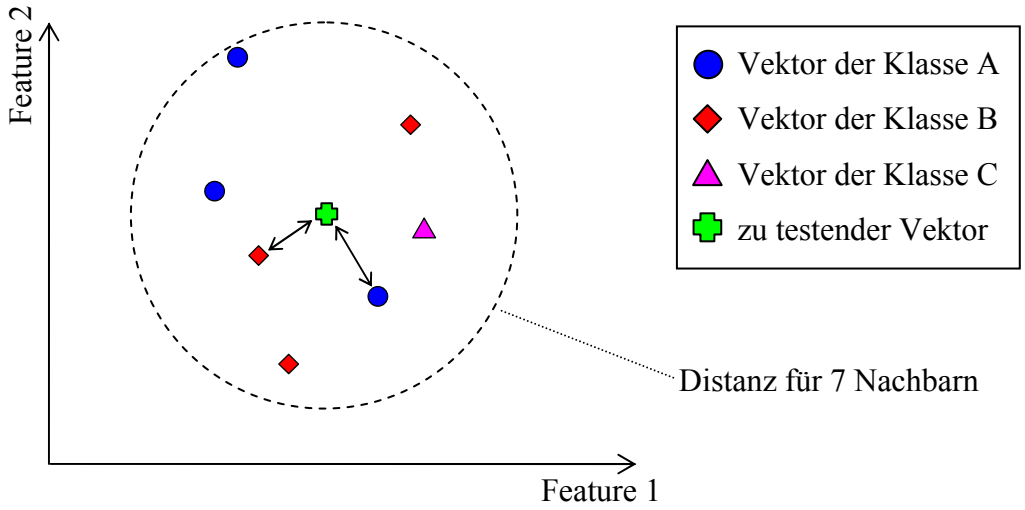


Abbildung 4-6
Kritische Entscheidung des KNN-Klassifikators bei zwei gleich häufigen Klassen

Bei dieser Auswahl stellt sich die Frage, wie sicher die Entscheidung getroffen werden kann. Um dies zu untersuchen, wurden diese kritischen Fälle für einen Klassifikationsdurchlauf betrachtet und die Abstände der potentiellen Klassen zum Testvektor miteinander verglichen.

Betrachtet wurde der KNN-Klassifikator mit sieben Nachbarn und den MFCC Koeffizienten als Features. Bei Verwendung des ersten Datensatzes gab es insgesamt 12 kritische Fälle, in denen mehr als eine Klasse die gleiche höchste Häufigkeit hatte. Die Differenz zwischen dem minimalen Abstand dieser Klassen zu dem jeweiligen Testvektor ist in Tabelle 11 eingetragen.

In der zweiten Spalte sind die Differenzen eingetragen, bei denen die korrekte Klasse die geringere Distanz hatte. In der dritten Spalte hatte eine falsche Klasse die geringere Distanz und in der vierten Spalte war die richtige Klasse nicht unter den häufigsten Nachbarn vertreten. In der letzten Zeile ist der Durchschnitt für jeden der Fälle dargestellt.

An einem kurzen Beispiel sei der erste Eintrag der Tabelle erläutert. In einem der Durchläufe der Kreuzvalidierung soll ein Musikstück klassifiziert werden. Unter den sieben nächstgelegenen Nachbarn im Feature-Raum befinden sich drei Nachbarn, die dem Genre „Classical“ angehören, drei Nachbarn, die dem Genre „Rock/Pop“ angehören, und ein Nachbar, der dem Genre „Jazz/Blues“ angehört. Es handelt sich folglich um einen kritischen Fall, da keines der Genres eine absolute Mehrheit hat. Für eine Entscheidung wird der minimale Abstand des zu testenden Musikstücks im Feature-Raum von den Vektoren der beiden in Frage kommenden Klassen bestimmt. Der nächstgelegene Nachbar des Genres „Classical“ hat einen Abstand von 0,958 während der nächstgelegene Nachbar des Genres „Rock/Pop“ einen Abstand von 0,902 hat, die Differenz der Distanzen beträgt somit 0,056. Das unbekannte Musikstück wird dem Genre „Rock/Pop“ zugeordnet, da dieses die geringere Distanz aufweist. Tatsächlich gehörte das zu testende Musikstück aber dem Genre „Classical“ an, womit eine Entscheidung für das falsche Genre getroffen wurde. Damit wird die Differenz der Distanzen in der Spalte „falsche Klasse“ eingetragen.

Nummer der kritischen Entscheidung	Distanzen		
	richtige Klasse	falsche Klasse	ganz falsch
1		0,056	
2	-0,010		
3	-0,116		
4		-0,212	
5	-0,321		
6		-0,041	
7	-0,340		
8			-0,037
9		0,003	
10		0,007	
11	-0,026		
12	-0,079		
∅	0,149	0,064	0,037

Tabelle 11
Differenzen der Distanzen bei den kritischen Entscheidungen im Testlauf
(6 Genres, 72 Musikstücke, MFCC Koeffizienten als Features)

Es fällt auf, dass bei der Wahl der korrekten Klasse die Differenz der Distanzen deutlich höher ist, als in den anderen Fällen.

Aufgrund der gewonnenen Erkenntnisse wurde eine Grenze auf 0,1 festgelegt, das sogenannte Vertrauensintervall. Die Distanzen zu den möglichen Klassen müssen sich um mindestens dieses Vertrauensmaß unterscheiden, damit eine Klassifikation noch als „sichere“ Entscheidung gilt. Unterschreitet die Differenz der Distanzen diesen Wert, so ordnet der Klassifikator den Testvektor der neuen Klasse „unsicher“ zu.

Mit dieser neuen Klassifikationsvorschrift wurde anschließend ein Experimentaldurchlauf durchgeführt, um den Einfluss der Änderungen zu testen. Tabelle 12 listet den Experimentalaufbau für die Versuche auf. Als Klassifikatoren werden nur die KNN Klassifikatoren mit drei, fünf und sieben Nachbarn verwendet. Der Klassifikator mit einem Nachbarn wird hier nicht betrachtet, da mit nur einem Nachbarn immer eine eindeutige Entscheidnung getroffen werden kann. Verwendet wurde der Test-Datensatz „Satz 1“. Die in den Experimenten verwendeten Features sind in Tabelle 4 aufgeführt. Die Resultate dieser Experimente sind in Abbildung 4-7 und Abbildung 4-8 dargestellt

Nummer	Klassifikator	Feature																				
		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21
1	KNN 3	X	X		X			X				X						X				
2	KNN 5	X	X		X			X				X						X				
3	KNN 3	X	X		X		X	X	X	X	X	X	X	X	X	X		X				
4	KNN 5	X	X		X		X	X	X	X	X	X	X	X	X	X		X				
5	KNN 3	X		X	X			X				X						X				
6	KNN 5	X		X	X			X				X						X				
7	KNN 7	X		X	X			X				X						X				
8	KNN 3	X		X	X			X				X							X			
9	KNN 5	X		X	X			X				X							X			
10	KNN 3	X		X	X			X				X										
11	KNN 5	X		X	X			X				X	X	X	X							
12	KNN 3								X		X	X	X	X	X				X			
13	KNN 5								X		X	X	X	X					X			
14	KNN 7								X		X	X	X	X					X			
15	KNN 3																		X			
16	KNN 3	X			X		X			X	X						X		X			
17	KNN 3	X		X	X		X	X	X	X	X	X					X		X			
18	KNN 3	X	X	X	X		X	X	X	X	X	X	X	X	X		X		X			X
19	KNN 3	X	X	X	X		X	X	X	X	X	X	X	X	X		X		X	X		X
20	KNN 5																		X			
21	KNN 5		X														X		X			
22	KNN 5		X		X								X				X		X	X		
23	KNN 5		X		X			X					X	X			X		X	X		
24	KNN 5		X		X			X					X	X			X		X	X		

Tabelle 12
Feature-Kombinationen der Experimente

In Abbildung 4-7 sind die Klassifikationsgüten des KNN Klassifikators mit und ohne Vertrauensmaß abgebildet. Auf der Abszisse sind die nach Tabelle 12 durchgeführten Experimente durchnummeriert Von den Balken stellt jeweils der linke das Ergebnis des normalen KNN Klassifikators dar, während der rechte Balken das Ergebnis des KNN Klassifikators mit Vertrauensmaß repräsentiert.

In Abbildung 4-8 ist der Anteil der als unsicher aussortierten Lieder dargestellt. Diesen Liedern wurden bei der Klassifikation keiner der vorgegebenen Klassen zugeordnet sondern die künstlich erzeugte Klasse „unsicher“.

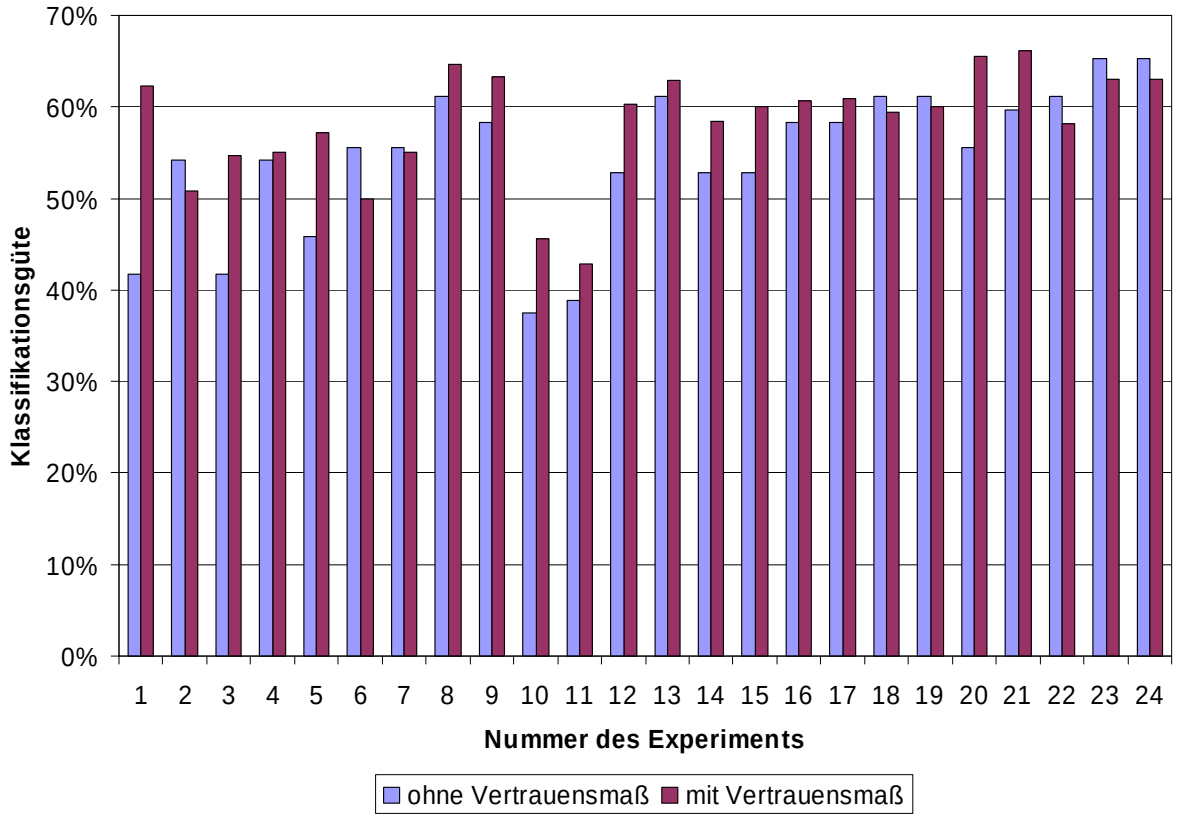


Abbildung 4-7
Klassifikationsgüten des KNN Klassifikators ohne und mit Vertrauensmaß

Die Klassifikationsgüte hat sich durch die Verwendung eines Vertrauensmaßes verbessert, im Durchschnitt über alle betrachteten Experimente um knapp 4%. Die Ausnahme bilden wenige Experimente, in dem sie sich geringfügig verschlechtert hat. Der Anteil der als „unsicher“ aussortierten Lieder liegt bei 7% bis 26%. Die beiden Fälle, in denen die meisten Musikstücke aussortiert wurden, korrespondieren mit Fällen, in denen sich die Klassifikationsgüte deutlich um 13% bzw. 20% anstieg. Die Verbesserung wird offensichtlich durch einen hohen Anteil nicht klassifizierter Lieder erkauft. Nicht immer hat ein solch hoher Anteil jedoch eine verbesserte Klassifikationsgüte zur Folge. Dann, wenn viele Lieder aussortiert werden, die andernfalls richtig klassifiziert worden wären, kann sich die Güte auch verschlechtern, wie in Experiment sieben geschehen.

Die Nutzung eines Vertrauensmaßes kann folglich zu einer deutlichen Verbesserung der Klassifikationsgüte führen. In ungünstigen Fällen kann sich die Klassifikationsgüte jedoch auch verschlechtern. Die Ergebnisse lassen es sinnvoll erscheinen das Vertrauensmaß für Klassifikatoren näher zu untersuchen und detailliertere Maße aufzustellen.

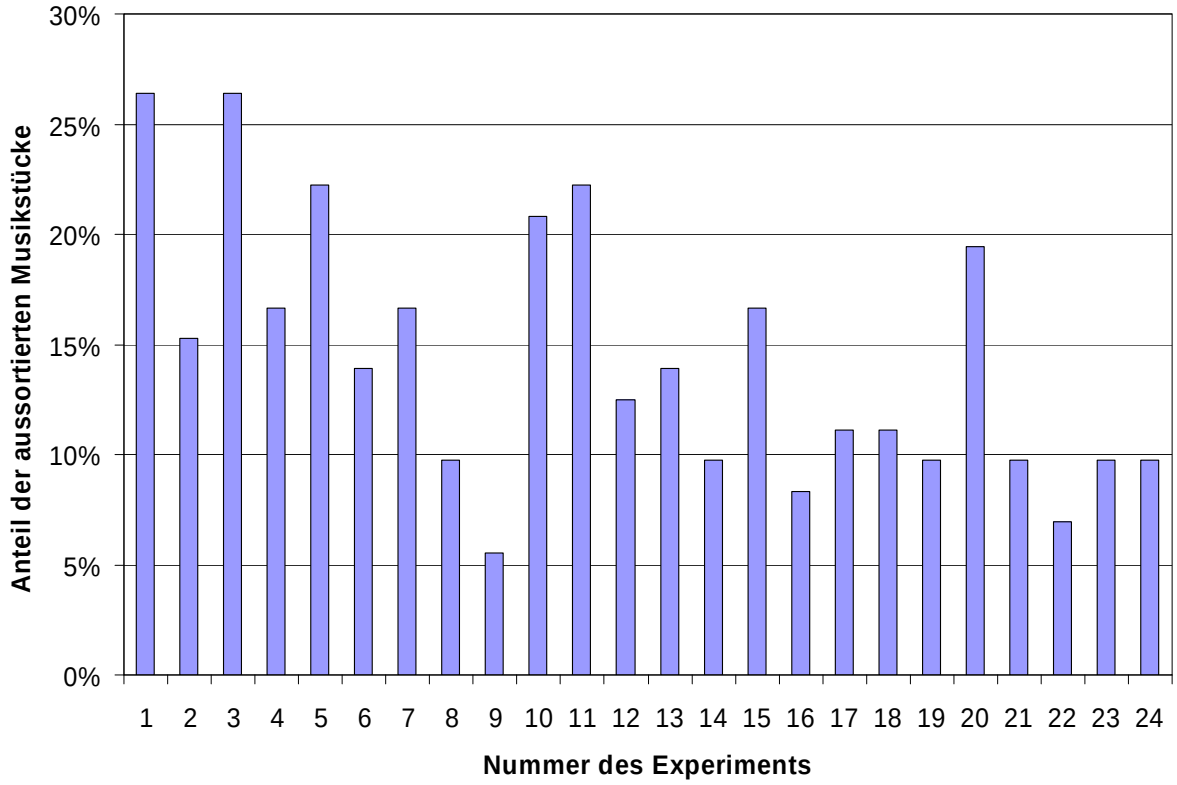


Abbildung 4-8
Anteil der als unsicher aussortierten Lieder in den Experimenten.

5. LAUFZEITANALYSE FÜR DIE MUSIKKLASSIFIKATION

Neben der reinen Klassifikationsgüte spielt auch die Laufzeit der Extraktion und Klassifikation für die Akzeptanz einer Musikklassifikation in Endgeräten eine wichtige Rolle. Auch eine sehr gute Klassifikation wird vom Anwender nicht akzeptiert werden, wenn die Laufzeit zu groß ist.

5.1. Laufzeiten der Feature-Extraktion

Die Laufzeiten wurden auf allen im Rahmen dieser Arbeit betrachteten Plattformen während der Extraktion der Features aus mehreren Fenstern aus einem kurzen Musikstück ermittelt.

Auf einem PC[5] wurde die Laufzeit für alle relevanten Funktionen separat bestimmt, indem jeweils die erforderliche Zeit für die Bearbeitung von 100.000 Datenblöcken ermittelt wurde. Dazu wurde die jeweilige Extraktions-Funktion 100.000 mal mit dem gleichen Datenblock aufgerufen und die dazu aufgewendete Zeit gemessen. Aufgrund der geringen Größe des Datenblocks (512 Samples) und der Funktionen wird im Messdurchlauf nur auf Daten zugegriffen, die sich komplett im Cache des Prozessors befinden. Die gemessene Laufzeit entspricht somit der reinen Rechenzeit, ohne durch Peripheriezugriffe beeinflusst zu werden. Die Laufzeit wurde anschließend aus fünf Messdurchläufen gemittelt, bei denen jeweils ein anderer Datenblock aus dem Musikstück verwendet wurde.

Für den SAA und den ARM-Prozessor steht jeweils ein Simulator zur Verfügung, der eine zyklengenaue Simulation des Programms ermöglicht (SAA: mmdspsim [12], ARM: ARMulator [41]). Für den SAA wurden die Laufzeiten aus den Profiling-Daten gewonnen, die bei der Bearbeitung eines Musikstückes generiert wurden, das aus insgesamt 58 Datenblöcken besteht. Die Profiling-Daten beziehen sich auf die reine Laufzeit der Funktionen, so dass keine Zeiten für Peripheriezugriffe enthalten sind. Für den ARM-Prozessor konnten die Laufzeiten aus den Statistiken des Simulators ermittelt werden.

Die Bezugsgröße von 1000 Datenblöcken entspricht bei der gegebenen Abtastrate der Musikstücke sowie der Größe der Datenblöcke einer Dauer von 23 Sekunden Musik.

[5] Pentium IV mit 3,2 GHz Taktfrequenz und 1GB RAM

Untersucht wurden die Laufzeiten der Extraktion folgender Features und Hilfsfunktionen:

Nr.	Funktion zur Berechnung von
1	FFT
2	Autokorrelationsfunktion (Zeitbereich)
3	Autokorrelationsfunktion (Frequenzbereich)
4	Drei Maxima der Autokorrelationsfunktion
5	Root Mean Square
6	Low energy windows
7	Sum of correlated components
8	Zero crossing rate
9	Relative periodicity amplitude peaks
10	Ratio of second and first periodicity peak
11	Fundamentalfrequenz
12	Magnitude of spectrum
13	Magnitude of spectrum_L
14	MFCC
15	Spectral Centroid
16	Spectral Rolloff
17	Spectral Extend
18	Amplitude of spectrum
19	Spectral Flux
20	Power Spectrum
21	Spectral Kurtosis
22	Spectral Bandwidth
23	Position of main peaks
24	Chroma vector
25	Amplitude of maximum in chromagram
26	Pitch interval between two max peaks in chromagram

Tabelle 13
Features, die hinsichtlich der korrespondierenden Extraktions-Laufzeiten untersucht wurden.

Die ermittelten Laufzeiten für die verschiedenen hier betrachteten Plattformen sind in Abbildung 5-1 bis Abbildung 5-4 abgebildet.

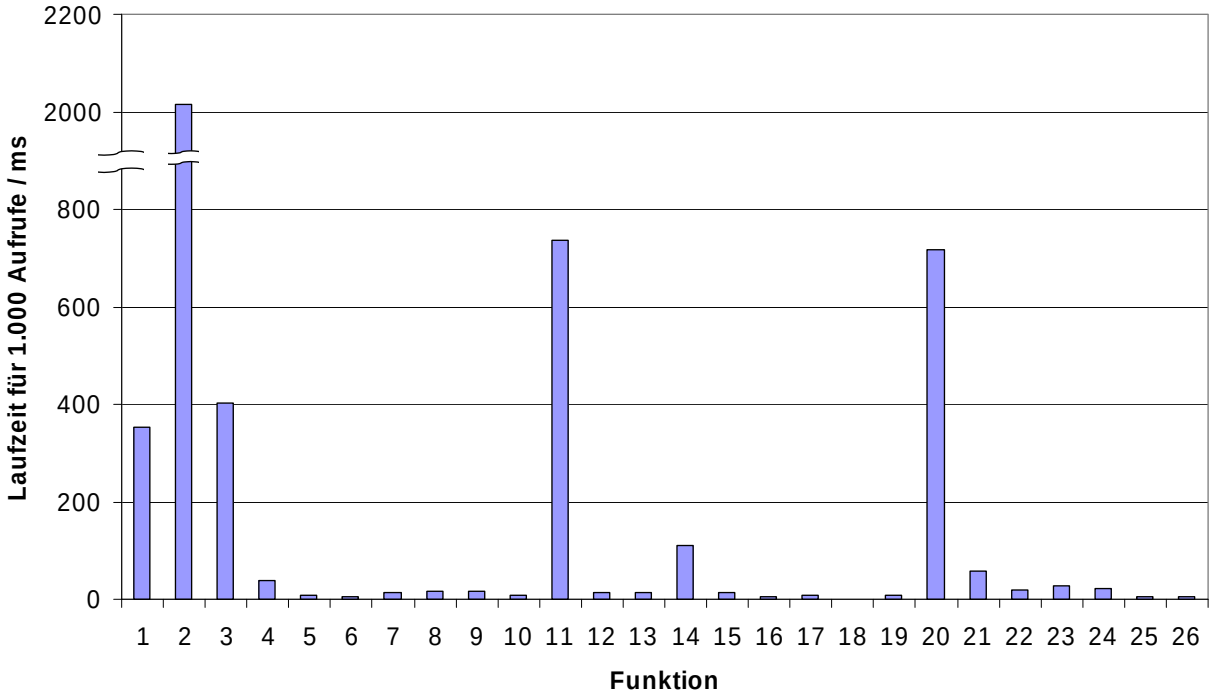


Abbildung 5-1
Laufzeiten einzelner Funktionen für die Feature-Extraktion aus 1.000 Datenblöcken auf dem SAA (133 MHz) für eine fixed-point Implementation.

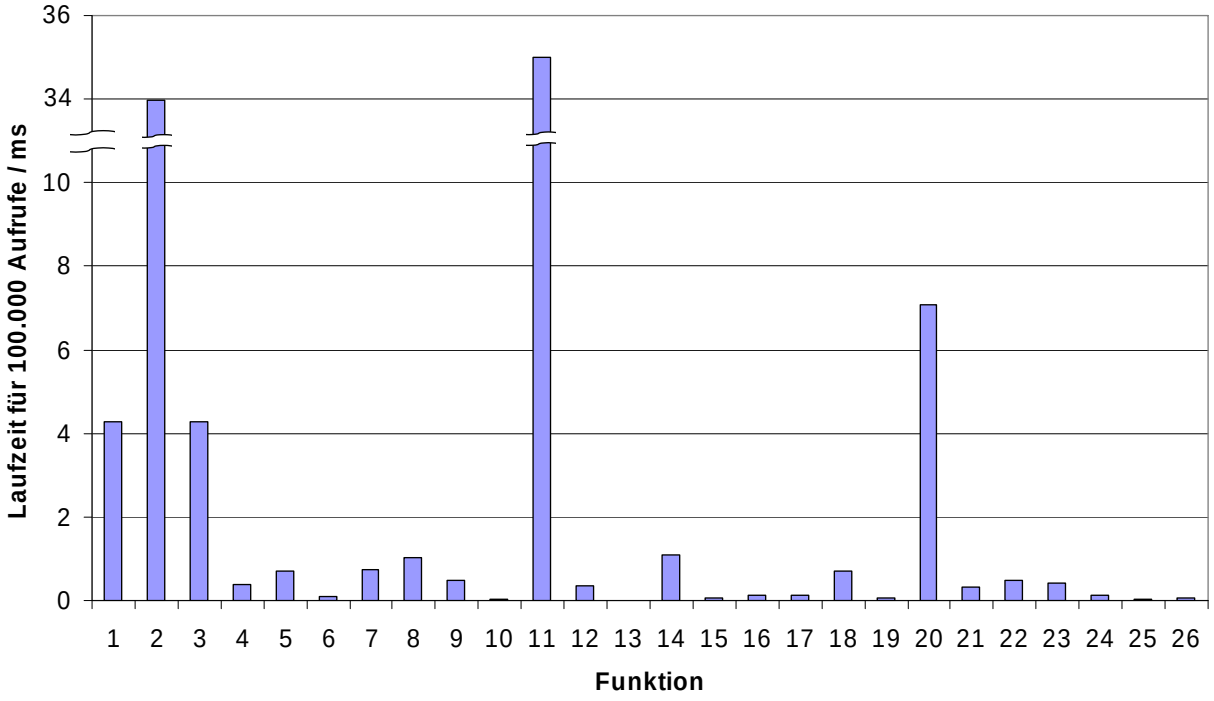


Abbildung 5-2
Laufzeiten einzelner Funktionen für die Feature-Extraktion aus 100.000 Datenblöcken auf einem PC (P4, 3,2 GHz) für eine floating-point Implementation.

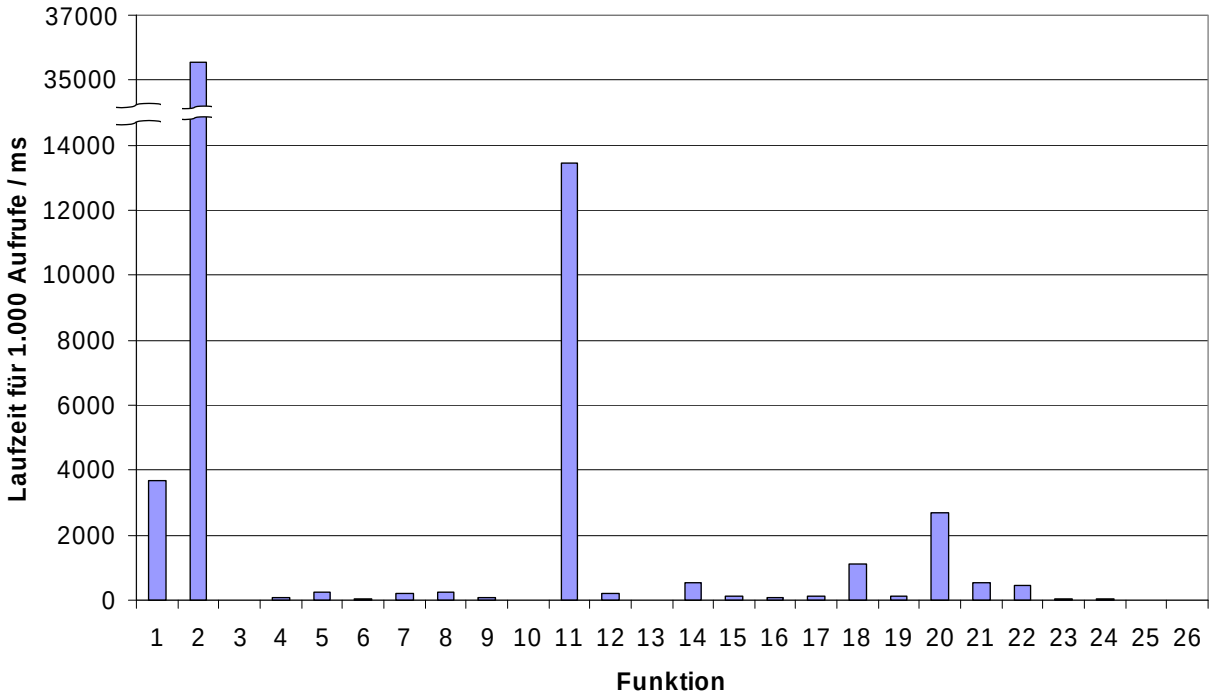


Abbildung 5-3
Laufzeiten einzelner Funktionen für die Feature-Extraktion aus 1.000 Datenblöcken auf dem ARM-Core 926EJ-S (264 MHz) für die floating-point Implementation.

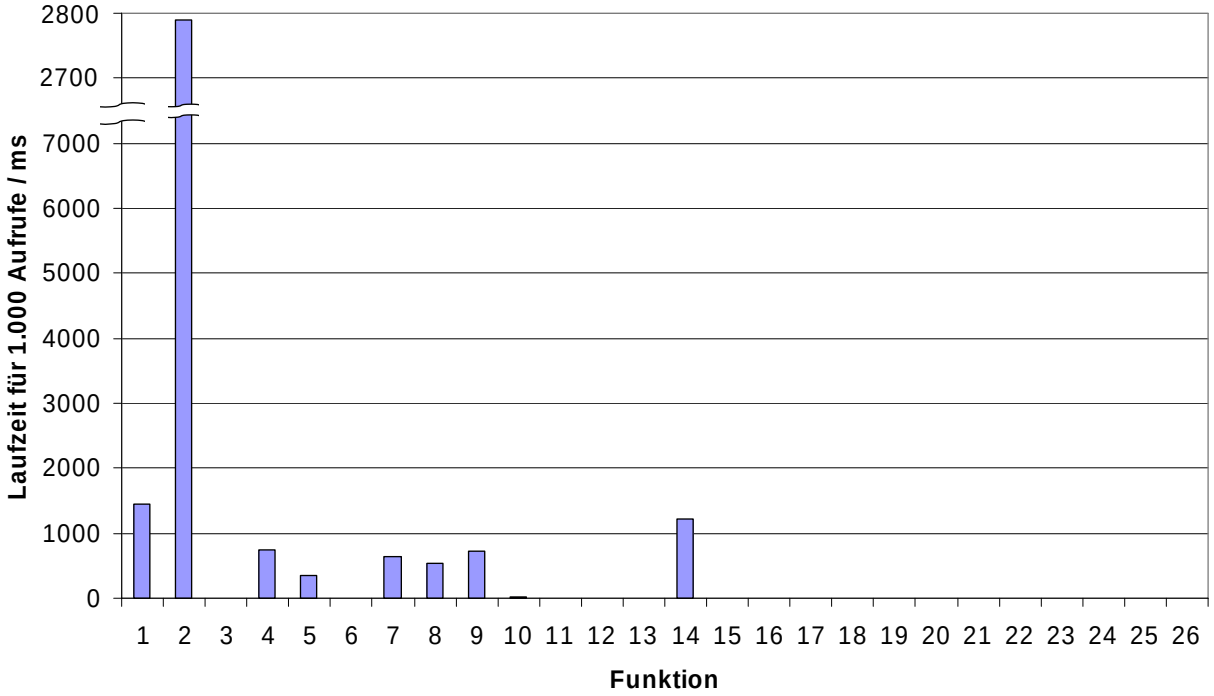


Abbildung 5-4
Laufzeiten einzelner Funktionen für die Feature-Extraktion aus 1.000 Datenblöcken auf dem ARM-Core des TI OMAP2420 (330 MHz) für die floating-point Implementation.

Insbesondere einige Funktionen stechen durch ihre hohen Laufzeiten hervor. An erster Stelle sind dies die Autokorrelationsfunktion (2), die Berechnung der Fundamentalfrequenz (11) und das „Power Spectrum“ (20). Auch die Laufzeit der FFT (1) liegt um einen Faktor von mindestens drei über der maximalen Laufzeit der verbleibenden Funktionen. Alle im Frequenzbereich ermittelten Features setzen ein bereits berechnetes Amplitudenspektrum voraus, die Laufzeit der FFT ist in ihnen nicht enthalten. Die Ausnahme bilden die Autokorrelation im Frequenzbereich und die Fundamentalfrequenz. In der Laufzeit zur Ermittelung dieser Features ist die Laufzeit für die Berechnung einer FFT enthalten.

Für die hohe Laufzeit der Funktionen zur Bestimmung der Fundamentalfrequenz und dem „Power Spectrum“ ist hauptsächlich die Berechnung des Logarithmus verantwortlich. Diese Bibliotheksfunktion wird bei der Berechnung der beiden Features sehr häufig aufgerufen.

Auf die Autokorrelation im Frequenzbereich (Funktion 3) wird näher in 5.3.2 eingegangen. Für den ARM-Core wurde sie nicht implementiert.

Um einen direkten Vergleich der Laufzeiten auf den hier betrachteten Plattformen durchzuführen, wurde ein Referenz-Experiment aus der Literatur herangezogen [50]. Für dieses Referenz-Experiment liegen auch Laufzeiten für einen TI OMAP2420 Mikroprozessor [6] vor. Die Laufzeiten für den OMAP2420 wurden auf dem ARM-Core ermittelt.

Tabelle 14 zählt die in dem Experiment verwendeten Funktionen auf. In Abbildung 5-5 sind die Laufzeiten auf den einzelnen Plattformen dargestellt. Sie wurden ermittelt für die Verarbeitung von 1.000 Datenblöcken. Dabei wird zwischen floating-Point und fixed-point Implementierungen unterschieden. Details zur floating-point nach fixed-point Umsetzung werden in Abschnitt 5.3.3 diskutiert.

Auf die Laufzeiten der floating-point Implementation auf dem SAA wird hier nicht weiter eingegangen. Aufgrund einer sehr ungünstigen Bibliothek zur Emulation der hardwaremäßig nicht unterstützen floating-point Operationen liegen die Laufzeiten um mehr als einen Faktor 60 über denen der restlichen betrachteten Laufzeiten.

[6] Der TI OMAP 2420 ist ein Mikroprozessor von Texas Instuments, der aus einer General-Purpose-CPU (ARM1136 Core mit 330 MHz, floating-point Unit nicht aktiviert) und einem DSP (TI TMS320C55x mit 220 Mhz) besteht. Eingesetzt wird er u.a. auf dem N800 Internet-Tablet von Nokia.

Autocorrelation Function
3 Maxima of the Autocorrelation Function
Zero-crossing rate
Root mean square
First relative periodicity amplitude peak
Second relative periodicity amplitude peak
Ratio of second and first periodicity peak
Sum of correlated components
Fourier Transform
MFCC

Tabelle 14
Die im Referenz-Experiment verwendeten Funktionen

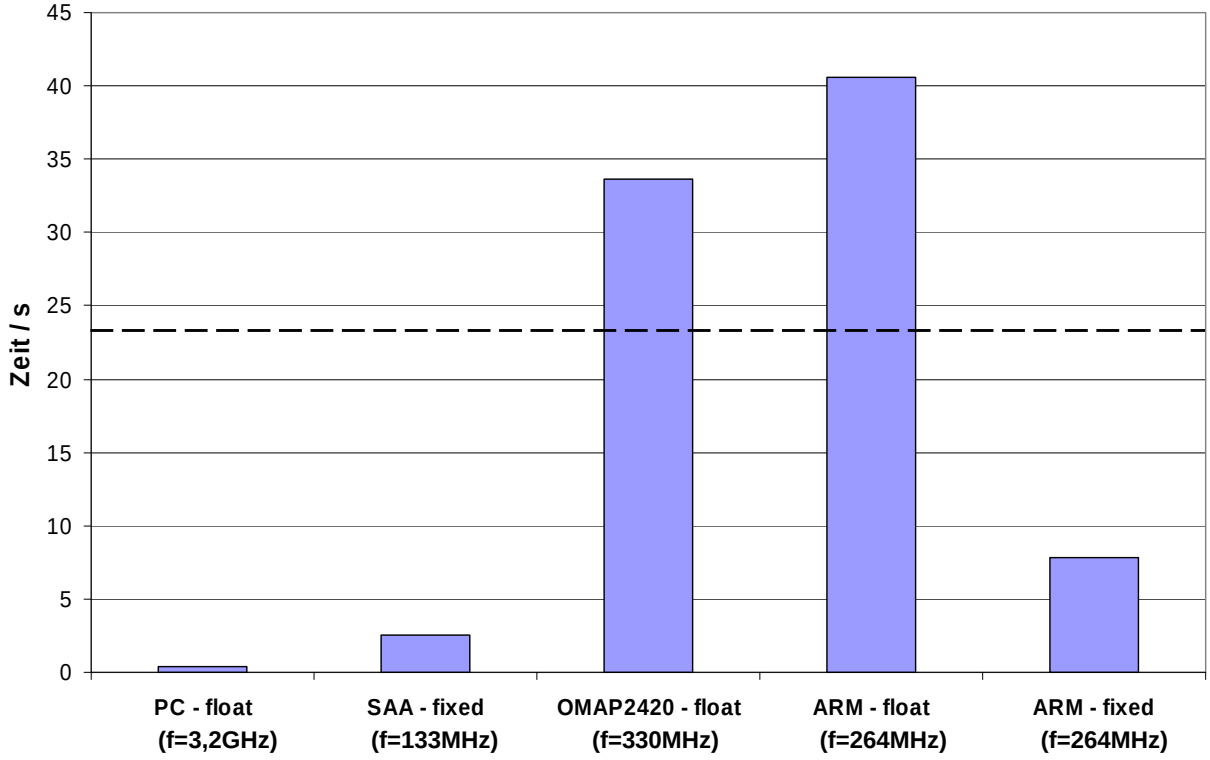


Abbildung 5-5
Laufzeiten des Referenz-Experiments, Feature-Extraktion aus 1000 Datenblöcken.
„float“ steht für eine floating-point Implementation, „fixed“ für eine fixed-point Implementation.
Markiert ist die Grenze von 23 Sekunden, denen die 1000 Datenblöcke entsprechen.

Erwartungsgemäß hat der PC durch seine hohe Taktfrequenz die geringste Laufzeit.

Sowohl der ARM-Core auf der N800 Plattform als auch der ARM-Core auf dem Nomadik benötigen zur Extraktion der Features länger, als der Musikabschnitt selber dauert. Das Verhältnis ihrer Laufzeiten entspricht annähernd dem Verhältnis ihrer Taktraten.

Auffallend ist der Unterschied in den Laufzeiten zwischen den fixed-point Implementationen auf dem SAA und dem ARM-Core. Obwohl der ARM-Core mit der zweieinhalbfachen Taktfrequenz arbeitet, benötigt er für die gleichen Aufgaben etwa dreimal so lange.

Der Unterschied ist in der unterschiedlichen Architektur der beiden Prozessoren begründet. Während der ARM-Core ein General-Purpose Prozessor ist, der lediglich über wenige Spezialbefehle verfügt, ist der SAA ein DSP mit VLIW-Architektur, der über einige Spezialfunktionen verfügt und Befehle parallel ausführen kann.

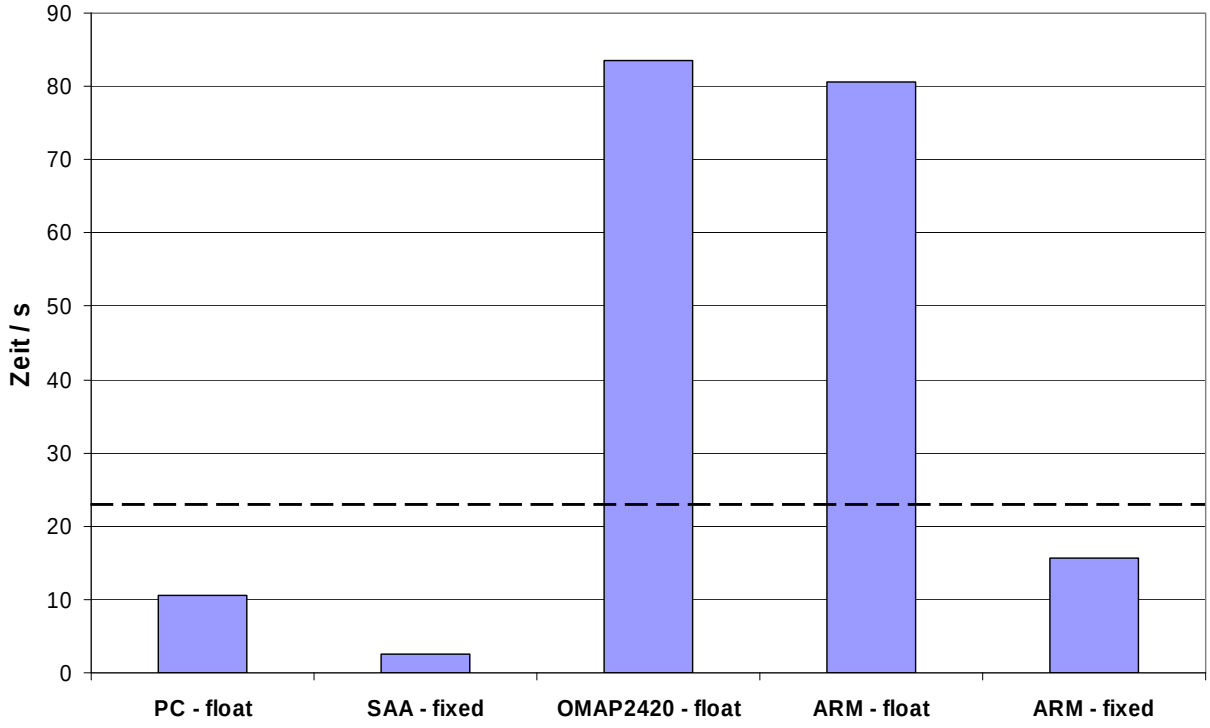


Abbildung 5-6
Laufzeiten des Referenz-Experiments, Feature-Extraktion aus 1000 Datenblöcken.
Die Laufzeiten wurden skaliert auf eine Taktfrequenz von 133 MHz, die der Taktfrequenz des SAA entspricht.

In Abbildung 5-6 wurden die Laufzeiten auf den einzelnen Prozessoren auf eine gemeinsame Taktfrequenz skaliert. Als Referenz wurde die Taktfrequenz von 133MHz des SAA gewählt. Es zeigt sich, dass die Feature-Extraktion auf dem SAA unter den hier betrachteten Implementierungsvarianten die effizienteste ist. Dies lässt sich wieder mit der speziellen Architektur des SAA begründen. Die Laufzeit des PC ist nur noch an zweiter Stelle, aber immer noch um ca. ein Drittel kürzer als die des ARM-Core mit fixed-point Zahlen. Ursache hierfür ist die deutlich höhere Komplexität der PC-CPU, die stärker auf Geschwindigkeit ausgelegt ist, und ihre größeren Caches, während für den ARM-Core das Augenmerk mehr auf einer geringen Verlustleistungsaufnahme liegt.

Der SAA verfügt über sogenannte „zero overhead loops“. Dies sind hardwareunterstützte Schleifen, die keinerlei zusätzlichen „Verwaltungsaufwand“ benötigen. Dies soll anhand des in Tabelle 15. dargestellten Beispiels verdeutlicht werden:

ARM	SAA
mov r1, <*Schleifenanzahl*> mov r2,#0 b PRUEF SCHLEIFENSTART: *[Schleifenkörper]* add r2,r2,#1 PRUEF: cmp r2,r1 blt SCHLEIFENSTART	R1h = <*Schleifenanzahl*>; loop ENDLABEL, R1h; *[Schleifenkörper]* ENDLABEL:

Tabelle 15.
Schleifen auf dem ARM-Core und dem SAA

In Tabelle 15. ist der erzeugte Assemblercode für eine Schleife auf dem ARM und dem SAA gegenüber gestellt. In beiden Fällen wird der Schleifenkörper *Schleifenanzahl* mal ausgeführt. Aus dem SAA wird dazu lediglich ein Register mit der Anzahl der Schleifendurchläufe geladen. Anschließend wird direkt die hardwareunterstützte Schleife gestartet, die keine weiteren Kontrollstrukturen benötigt. Auf dem ARM-Core dagegen wird ein weiteres Register als Zähler benötigt, das nach jedem Durchlauf inkrementiert und mit der gewünschten Anzahl der Schleifendurchläufe verglichen wird. Anhand des Ergebnisses dieses Vergleiches erfolgt ein bedingter Sprung zurück zum Anfang der Schleife. Insbesondere bei sehr häufig ausgeführten Schleifen führen diese Kontrollstrukturen zu einer deutlichen Zunahme der Laufzeit für Schleifen.

Durch seine VLIW-Struktur ist der SAA in der Lage, Befehle parallel auszuführen. So kann er neben einer Arithmetischen Operation bis zu zwei Speicherzugriffe durchführen und dabei bis zu zwei Indexregister erhöhen.

In Tabelle 16 ist als Beispiel der Assemblercode für die innere Schleife der Autokorrelationsfunktion auf dem ARM und dem SAA gegenüber gestellt. Auf dem SAA wurden einige Befehle vor und hinter die Schleife gesetzt, um die Daten für den ersten Durchlauf zu Laden bzw. die Multiplikation und Addition der Daten des letzten Durchlaufs abzuschließen. Diese sind hier nicht abgebildet, beeinflussen den Vergleich aber nicht weiter.

ARM	
ldr	r5,[r0,r3,lsl #2]
sub	r6,r3,r12
ldr	r6,[r0,r6,lsl #2]
add	r3,r3,#1
cmp	r3,r2
smlal	r14,r4,r5,r6
blt	0xbb88

SAA
R0h := *(MEM) Ax3; Ax3 = Specadd(Ax3, Ix1); XR3 = X_imulss(R0h, R0l);
R0l := *(MEM) Ax3; Ax3 = Specadd(Ax3, Ix2); XR4 = L_add(R4, R3);

Tabelle 16
Parallele Ausführung von Befehlen: Ein Schleifenkörper auf dem ARM-Core und dem SAA

Der ARM-Core führt jeden Befehl einzeln nacheinander aus. Der SAA dagegen kann manche der Befehle bis zu einem Parallelitätsgrad von 8 parallel ausführen [6]. So lädt er hier beispielsweise parallel zur Berechnung eines Zwischenergebnisses bereits die Daten für den nächsten Durchlauf und bestimmt die nächste Speicheradresse. Die Anzahl der Instruktionen des Schleifenkörpers reduziert sich dadurch deutlich auf weniger als ein drittel, wodurch sich die Abarbeitung der Schleife entsprechend beschleunigt.

5.2. Laufzeit Klassifikation

Neben den Laufzeiten für die Extraktion der Features trägt auch die Laufzeit der Klassifikatoren zur Gesamtlaufzeit bei. In diesem Kapitel wird daher die Laufzeit der Klassifikatoren untersucht.

Diese wurden wieder auf einem PC durch die Zeit für die Ausführung von 100.000 Aufrufen der Funktionen ermittelt. Ebenso wurden die Laufzeiten für den SAA und den ARM wieder im Simulator ermittelt.

Untersucht wurde der „k-Nearest-Neighbor“ Klassifikator mit 66 Trainingsvektoren und unterschiedlich vielen Features. Ermittelt wurden für einen und für drei Nachbarn jeweils die Laufzeiten bei Verwendung von zehn, 20 und 26 Features. Aus diesen ließ sich jeweils der annähernd lineare Verlauf der Laufzeit mit zunehmender Anzahl an Features ermitteln.

Aufgrund der linear ansteigenden Anzahl an Berechnungen und Vergleichen innerhalb des Klassifikators wird die Laufzeit auch bei fester Anzahl an Features und steigender Anzahl an Trainingsvektoren linear zunehmen.

Damit beträgt die Laufzeitkomplexität des k-Nearest-Neighbor Algorithmus

$$O(N \cdot M)$$

Wobei N die Anzahl der Merkmale und M die Anzahl der Trainingsvektoren darstellt.

Es fällt in Abbildung 5-7 bis Abbildung 5-9 auf, dass die Laufzeit selbst bei hoher Anzahl an Features deutlich hinter der Laufzeit zurückbleibt, die für die Extraktion der Features aus mehreren Fenstern benötigt wird. Der hier betrachtete KNN Klassifikator ist damit für die Gesamtlaufzeit zu vernachlässigen.

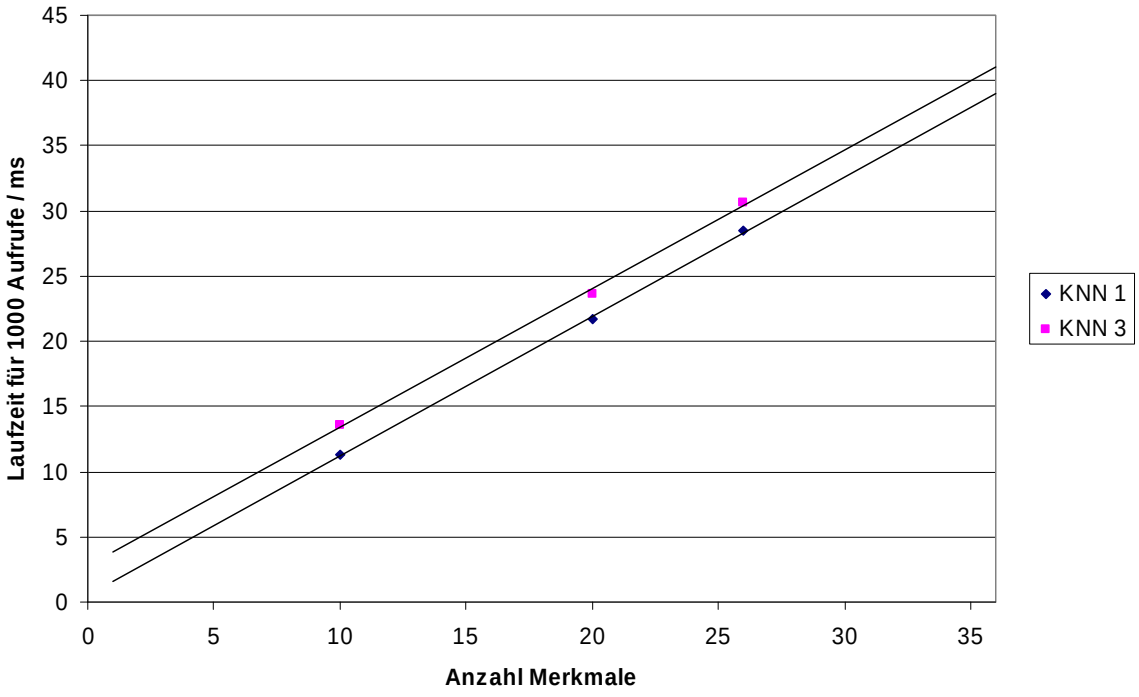


Abbildung 5-7
Laufzeit Klassifikation „k-Nearest-Neighbor“ für 1000 Aufrufe
PC (3,2 GHz), floating-point Implementation

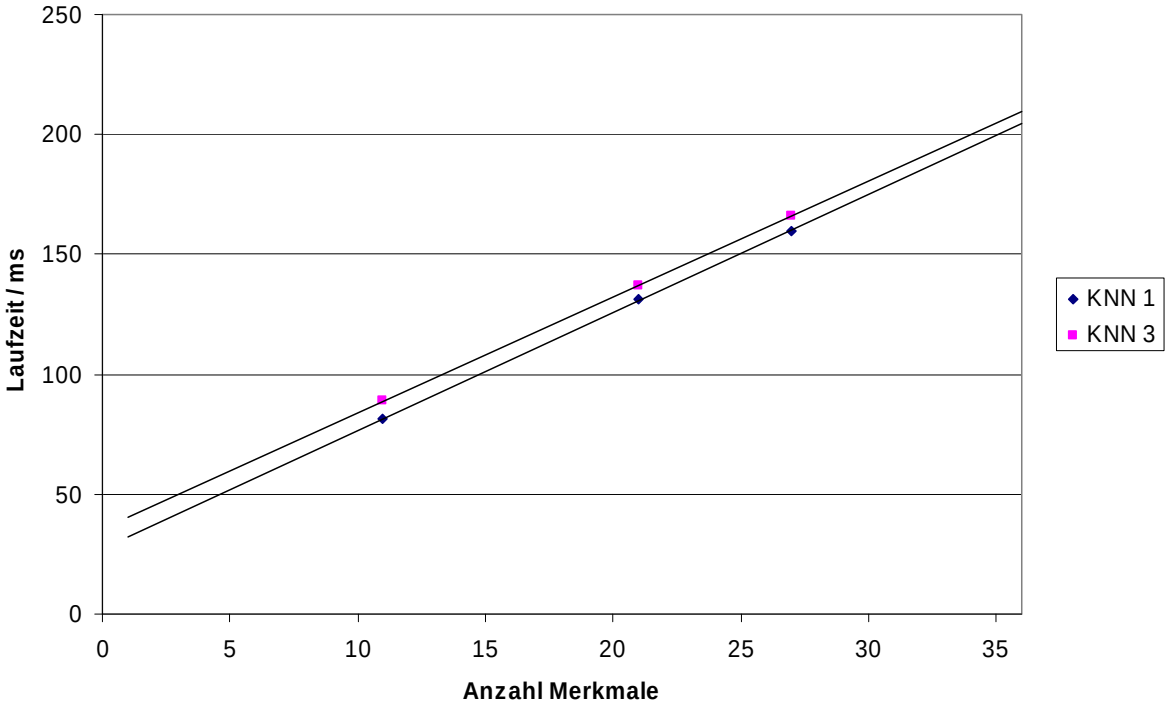


Abbildung 5-8
Laufzeit Klassifikation „k-Nearest-Neighbor“
SAA, fixed-point Implementierung

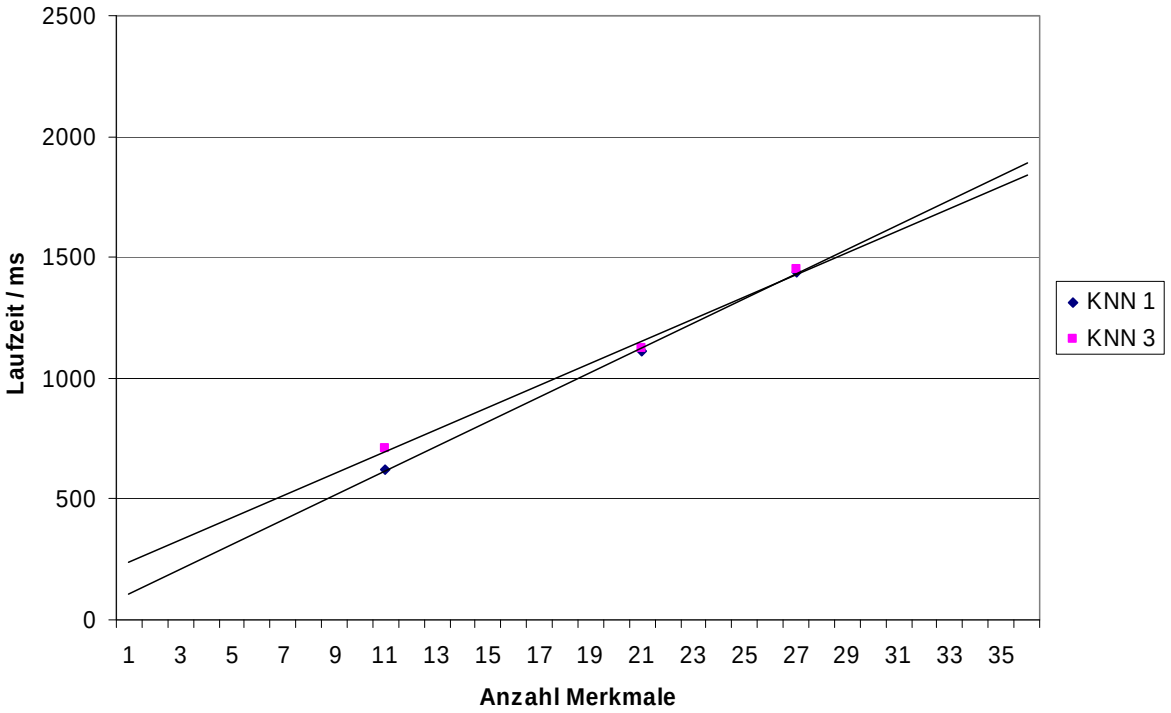


Abbildung 5-9
Laufzeit Klassifikation „k-Nearest-Neighbor“
ARM-Core, floating-point Implementierung

5.3. Optimierungsansätze

Da grundsätzlich eine geringe Laufzeit angestrebt wird, bedarf die Optimierung des Programms einer gesonderten Betrachtung.

Zur Verbesserung der Laufzeit gibt es vielfältige Ansatzpunkte. Der erste setzt an dem Code selber an und versucht, die Befehlsfolgen zu optimieren. Der zweite Ansatzpunkt betrachtet die kompletten Funktionen und versucht ihre Aufrufe bzw. ihren Ablauf zu optimieren. Als weitere Möglichkeit zur Optimierung bietet sich eine Umstellung des verwendeten Zahlenformates von floating-point nach fixed-point an.

Im Folgenden werden die verschiedenen Möglichkeiten den Code zu optimieren näher erläutert. Dabei wird speziell auf die Möglichkeiten eingegangen, die der SAA bietet.

5.3.1. Code-Optimierung

Insbesondere Schleifen kommen bei der Verarbeitung linearer Audiodaten häufig vor und benötigen einen Großteil der gesamten Laufzeit. Daher ist es wichtig, den Ablauf der Schleifen möglichst effizient zu gestalten und den Schleifenkörper klein zu halten.

Der SAA bietet dazu eine Hardwareunterstützung für Schleifen („Hardwareloops“). Normalerweise wird in einer Schleife eine Zählvariable mitgeführt und nach jedem Durchlauf inkrementiert. Anschließend wird sie mit der Endbedingung verglichen und anhand des Ergebnisses des Vergleichs ein bedingter Sprung an den Anfang der Schleife durchgeführt. Ist die Anzahl der Durchläufe vor Beginn der Schleife bekannt, so kann der SAA stattdessen eine hardwareunterstützte Schleife ausführen. Hierbei wird der ganze „Verwaltungsaufwand“ von der Hardwarelogik ausgeführt, die Schleife selber verursacht neben den Befehlen des Schleifenkörpers keinerlei zusätzlichen Rechenaufwand. Die maximale Anzahl der Ausführungen für eine Schleife darf 1023 betragen. Ist nicht sicher zu stellen, dass die Schleife mindestens einmal ausgeführt wird, so wird vom Compiler vor der Schleife eine entsprechende Sicherheitsabfrage eingebaut. Der SAA kann bis zu drei dieser Hardwareloops ineinander geschachtelt ausführen.

in Software realisierte Schleife	Hardware-unterstützte Schleife
LAB_START: *[Schleifenkörper]* jumpi LAB_COUNT; LAB_COUNT: R0l = add(R0l, 1); jumpi LAB_END; LAB_END: cmpu(R0l, <*Schleifenanzahl*>); jumpi if(<) LAB_START;	loop ENDLABEL, <*Schleifenanzahl*>; *[Schleifenkörper]* ENDLABEL:

In diesem Beispiel ist der Assemblercode von zwei Schleifen auf dem SAA gegenüber gestellt. Auf der linken Seite ist eine sehr einfache, in Software realisierte Schleife dargestellt, auf der rechten Seite eine Schleife mit Hardwareunterstützung. In der normalen Schleife wird ein Register als Schleifenzähler benötigt, das nach jedem Durchlauf inkrementiert und auf den Endwert überprüft wird. Anhand dieses Vergleichs findet ein bedingter Sprung zurück an den Schleifenanfang statt. Für die hardwareunterstützte Schleifen existieren separate Zählregister und eine Kontrolllogik. Es wird nur die Anzahl der Schleifendurchläufe und die Größe der Schleife (wird vom Compiler ermittelt) benötigt. Anschließend kann die Schleife ausgeführt werden, ohne dass zusätzliche Verzögerungen durch Kontrollstrukturen auftreten.

Der Compiler versucht selbständig zu erkennen, welche Schleifen sich für eine Hardwareloop eignen. Doch nicht immer sind die Zusammenhänge für den Compiler leicht zu erkennen, so dass sich die Anzahl der Ausführungen nicht ermitteln lässt. Für diese Fälle gibt es Compiler-Direktiven, mit denen die Anzahl der Ausführungen vorgegeben werden kann, um so die Nutzung der Hardwareloops zu ermöglichen.

```
#pragma loop minitercount (1)
#pragma loop maxitercount (512)
for(i=k; i<n; i++)
{
...
}
```

Das Beispiel zeigt eine einfache Schleife, für die der Compiler die Wertebereiche für k und n nicht auflösen können soll. Die beiden #pragma geben an, dass die folgende Schleife mindestens einmal und maximal 512 mal durchlaufen wird, so dass dennoch eine hardwareunterstützte Schleife genutzt werden kann.

Um eine Hardwareloop zu ermöglichen sind noch weitere Bedingungen zu erfüllen. Der Schleifenzähler muss nach jedem Durchlauf um eins erhöht werden. Ist dies nicht der Fall, so ist alternativ eine Hilfsvariable einzuführen und als Schleifenzähler zu verwenden. Der alte Schleifenzähler ist währenddessen weiterhin parallel zu berechnen, falls er innerhalb der Schleife benötigt wird. Funktionsaufrufe sind insbesondere in der Abbruchbedingung der Schleife zu vermeiden.

unoptimierte Schleife	optimierte Schleife
for(x=1; x<maxval(y); x=x*2) { ... }	num = maxval_neu(y); x=1; #pragma loop minitercount 1 #pragma loop maxitercount 1023 for(i=0; i<num; i++) { ... x=x*2; }

In diesem Beispiel ist auf der linken Seite eine Schleife in ihrer „Originalform“ abgebildet, die nicht beschleunigt werden kann. Auf der rechten Seite wurde diese Schleife so umgeschrieben, dass sie als Hardwareloop ausgeführt werden kann. Die Anzahl der Durchläufe wird nun vor Beginn der Schleife bestimmt, der alte Schleifenzähler wird weiterhin parallel berechnet.

Ein Aspekt, der in Schleifen zu großen Verzögerungen führt, sind häufige Speicherzugriffe. Oftmals wird in einer Schleife mehrfach auf die gleiche Speicherstelle oder auf aufeinander folgende Speicherstellen zugegriffen. Bei dem mehrfachen Zugriff auf die gleiche Speicherstelle wird jedes Mal der Wert ausgelesen und anschließend wieder zurück geschrieben, während eine Hilfsvariable viel effektiver wäre. Der Zugriff auf aufeinander folgende Speicherstellen lässt sich besser durch Pointer regeln. Diese haben den Vorteil, dass die Adresse nicht jedes Mal neu berechnet werden muss und dass sich die automatischen Adressoperatoren des SAA ausnutzen lassen. Die Adressoperatoren sind Spezialbefehle des SAA, die parallel zu einem Speicherzugriff ausgeführt werden und dabei einen Pointer auf eine nachfolgende Speicherstelle setzen. Zusätzliche Adressberechnungen entfallen damit.

unoptimierte Schleife	optimierte Schleife
for(i=0; i<n; i++) { a[n] += b[i]; }	sum = 0; pt = &b[0]; for(i=0; i<n; i++) { sum += *pt; pt++; } a[n] += sum;

In diesem Beispiel ist der „Originalcode“ auf der linken Seite dargestellt, während auf der rechten Seite der optimierte Code steht. In dem nicht optimierten Code wird in jedem Durchlauf der Schleife zunächst die Adresse von b[i] berechnet und der Wert von dieser Speicheradresse geladen. Anschließend wird der Wert von der Speicherstelle a[n] geladen, um den Wert von b[i] erhöht und wieder zurückgespeichert. In der optimierten Version wird ein Pointer auf die b[i] erstellt. Deren Werte werden in einer Hilfsvariable aufsummiert, wobei die Adressoperatoren des SAA automatisch verwendet werden. Nach der Schleife wird der Wert der Summe zu a[n] hinzuaddiert.

Neben dem Umschreiben einzelner Code-Abschnitte lohnt sich eine genaue Überprüfung, ob einzelne Funktionen nicht bereits als spezielle Hardwarebefehle vorhanden sind oder sich durch Spezialbefehle effizienter nachbilden lassen. Die Betragsbildung existiert z.B. auf dem SAA bereits als Spezialbefehl, der ein programmiertes Äquivalent mit Vergleichen und bedingten Sprüngen unnötig macht.

Als Beispiel der geschickten Nutzung vorhandener Befehle sei im Folgenden ein Abschnitt aus der Schleife zur Extraktion der Zero Crossing Rate (siehe 2.2.2.1) gegeben.

wenig optimierter Code	optimierter Code
tmp1 = SGN(val); val = *pt; pt++ tmp2 = SGN(val); tmp3 = wsub(tmp1, tmp2); tmp4 = wabssat(tmp3); zero_cr = wadd(zero_cr, tmp4);	tmp1 = winterval(val, 1); val = *pt; pt++; tmp2 = winterval(val, 1); tmp3 = wsub(tmp1, tmp2); tmp4 = wabssat(tmp3); zero_cr = wadd(zero_cr, tmp4);

Auf der linken Seite der Tabelle ist der nur wenig optimierte Code zur Bestimmung der Zero Crossing Rate abgebildet, auf der rechten Seite der optimierte Code. Die Variable *pt* ist ein Pointer auf die zu bearbeitenden Daten. Das Compiler-Makro „SGN“ berechnet das Vorzei-

chen des angegebenen Wertes, indem es überprüft ob der Wert kleiner Null ist, dann einen bedingten Sprung ausführt und entweder eine „1“ oder „-1“ zurückliefert. Der Hardwarebefehl winterval(x,y) liefert den Wert von x zurück, begrenzt auf das Intervall [0,y]. Gibt man y als „1“ vor, so wird folgende Funktion berechnet: $winterval(x,1) = \begin{cases} 0 & x \le 0 \\ 1 & für \; x > 0 \end{cases}$. Diese Funktion ist weitestgehend identisch mit der Definition der Vorzeichenfunktion, wenn man „0“ anstelle der „-1“ für negative Zahlen annimmt. Der Unterschied besteht in der Behandlung der Zahl null, die bei dieser Funktion als negative Zahl angesehen wird. Konkret gibt es dadurch Unterschiede, wenn das Musiksignal genau die Nullachse berührt, ohne sie zu überschreiten. Diese Sonderfälle lassen sich aber als sehr selten ansehen und werden sich über die Dauer eines Musikstücks herausmitteln. Durch die geschickte Nutzung dieser Funktion konnte die Laufzeit der Schleife um ein Drittel reduziert werden.

5.3.2. Funktionsoptimierung

Viele Funktionen, haben eine hohe Laufzeit, die sich nicht weiter verbessern lässt. Entweder, weil es sich um Bibliotheksfunktionen handelt, oder weil die Grenzen der reinen Befehlsoptimierung erreicht wurden. In diesen Fällen muss nach anderen Möglichkeiten gesucht werden, die Laufzeit zu verringern.

Bei Funktionen, die regelmäßig mit einer geringen Anzahl an gleichen Parametern aufgerufen werden, bietet sich die Ersetzung durch eine Wertetabelle an. Als Beispiel ist in der folgenden Tabelle ein Abschnitt aus der FFT dargestellt.

Funktionsaufruf
angle = spectra_count * PI / max_spectra_for_stage; real_correction = cos(angle); imag_correction = sin(angle);

Tabellenzugriff
angle = spectra_count * 256 / max_spectra_for_stage; real_correction = sincos_table[angle]; imag_correction = - sincos_table[angle+128];

In der oberen Hälfte ist ein Abschnitt der FFT dargestellt, in dem die sinus- und cosinus-Funktion sehr häufig aufgerufen wird. Aufgrund der hier nicht dargestellten äußeren Bedingungen werden die Funktionen nur für wenige ganz bestimmte Werte berechnet. In der unteren Hälfte des Beispiels ist der Funktionsaufruf durch den Zugriff auf eine Tabelle ersetzt

worden, der den Abschnitt erheblich beschleunigt. Durch den mathematischen Zusammenhang zwischen sinus und cosinus, $\sin(\alpha) = -\cos\left(\alpha + \frac{\pi}{2}\right)$, lässt sich die Berechnung beider Funktionen durch die gleiche Tabelle realisieren. Die Tabelle kann entweder fest im Programmcode integriert werden oder muss beim Start des Programms einmal berechnet werden.

Die Laufzeit der FFT reduzierte sich durch diese Ersetzung auf dem PC um 88%.

Für einige Funktionen existieren auch alternative Berechnungswege, beispielsweise ist die Autokorrelationsfunktion zunächst über den Zeitbereich definiert. Das gleiche Ergebnis lässt sich allerdings auch über den Frequenzbereich erreichen. Hierzu kann die folgende Relation verwendet werden

$$r_{xx} = x(t) * x(-t)$$
$$\Updownarrow$$
$$\phi_{XX} = X(f) \cdot X^*(-f) = |X(f)|^2$$

Damit lässt sich die Autokorrelationsfunktion alternativ auch berechnen, indem das Signal in den Frequenzbereich transformiert, das Amplitudenspektrum quadriert und anschließend wieder in den Zeitbereich zurück transformiert wird. Da das Amplitudenspektrum bereits für andere Features berechnet wird, fällt für die Autokorrelation somit nur die Quadrierung des Amplitudenspektrums und die Rücktransformation an.

In Abbildung 5-1 und Abbildung 5-2 sind die Laufzeiten der Berechnung der Autokorrelationsfunktionen im Zeitbereich und im Frequenzbereich für einen PC und den SAA eingezeichnet. In beiden Fällen hat sich die Laufzeit erheblich verringert.

Die Laufzeit zur Bestimmung der Autokorrelationsfunktion lässt sich auf diese Weise auf dem betrachteten PC um 87% verbessern und auf dem SAA um 80%.

5.3.3. Optimierung des Zahlenformats

Für die Repräsentation von Zahlen gibt es im Wesentlichen zwei unterschiedliche Darstellungen, das *fixed-point* Format und das *floating-point* Format.

Fixed-point Zahlen sind ganze Zahlen mit einem eng begrenzten Wertebereich aus Z. Mögliche Nachkommastellen werden durch willkürliche Setzung eines Kommas an einer beliebigen Stelle (bzw. der impliziten Annahme, die Zahl sei der Zähler eines Bruchs mit festem Nenner) festgelegt. Bei einer Multiplikation oder Division muss dieses Komma durch eine zusätzliche Korrekturoperation beachtet werden. Den effizienten Umgang mit fixed-point Zahlen beherrscht jeder Prozessor. Der große Nachteil bei der Nutzung von fixed-point Zahlen ist ihr begrenzter Wertebereich, der bei der Programmierung sehr genau beachtet werden muss. Bei

betragsmäßig sehr großen oder sehr kleinen Zahlen wird ansonsten der Wertebereich der fixed-point Zahl überschritten oder das Ergebnis einer Berechnung liefert durch nachfolgende Rundungs- oder Quantisierungsoperationen Null zurück.

Das floating-point Format nach der IEEE 754 Notation ist aufgebaut aus einer Mantisse *m* und einem Exponenten *e*, die für sich jeweils wieder ganze Zahlen sind. Sie repräsentieren eine Zahl in dem Format $0,m \cdot 2^e$, wobei diese immer so normalisiert wird, dass *m* den Anteil nach dem Komma angibt. Der Vorteil gegenüber den fixed-point Zahlen ist der deutlich größere Wertebereich sowie die höhere Genauigkeit, gegenüber einer fixed-point Zahl mit gleichem Speicherverbrauch. Der Nachteil ist der höhere Aufwand von Rechenoperationen mit floating-point Zahlen, der in Coprozessoren ausgeführt wird. Ist keine hardwaremäßige Unterstützung für die Verarbeitung von floating-point Zahlen vorhanden, so können die Operationen auch automatisch mittels spezieller Bibliotheken emuliert werden. Der Nachteil an dieser Lösung ist die dramatisch schlechtere Laufzeit gegenüber einer Hardware-Version oder gegenüber einer fixed-point Implementierung.

Die Implementation einer Funktion für die Feature-Extraktion ist im floating-point Format sehr gut möglich, da der Wertebereich und die Genauigkeit der floating-point Zahlen für diese Aufgabenstellung ausreichend ist. Doch da die Berechnung der meisten mathematischen Operationen durch die höhere Verarbeitungskomplexität in floating-point länger dauert als in fixed-point, sollte eine Implementation der Funktionen mit fixed-point Zahlen in Erwägung gezogen werden. Insbesondere auf Prozessoren, die das floating-point Format nicht direkt unterstützen, sondern die Funktionen in Software emulieren – wie dem SAA – trägt die Wahl des Zahlenformats ganz entscheidend zur Verbesserung der Laufzeit bei.

Das Ergebnis einer per fixed-point Arithmetik berechneten Funktion hat meist einen anderen Wertebereich als das Ergebnis der gleichen Funktion, die per floating-point Arithmetik berechnet wurde. Aufgrund von Rundungsfehlern ist die Genauigkeit der Ergebnisse geringer, der Verlauf und damit das Verhältnis der Werte zueinander jedoch gleich.

Im Folgenden soll an einem kurzen Beispiel die Umsetzung von floating-point nach fixed-point erläutert werden. Der Codeabschnitt ist der FFT entnommen.

floating-point Implementation
temp_real = real_correction * X[right] - imag_correction * Y[right]; X[left] += temp_real;

fixed-point Implementation
acc1 = wX_fmul(real_correction, X[right]); acc2 = wX_fmul(imag_correction, Y[right]); temp_real = wX_sub(acc1, acc2); X[left] = wh_addr_Xh(temp_real, X[left]);

In dem abgebildeten Beispiel ist in der oberen Hälfte der floating-point Code und auf der unteren Hälfte der korrespondierende fixed-point Code auf dem SAA dargestellt. Die Eingangsdaten in X[] und Y[] sind die Musikdaten mit einer Wortbreite von 16 Bit.

In der floating-point Implementierung sind diese Eingangsdaten bereits in floating-point Zahlen umgewandelt worden. Sie werden mit den Korrekturfaktoren multipliziert, von einander abgezogen und anschließend aufsummiert, ohne dass der Wertebereich beachtet werden muss.

In der fixed-point Implementation liegen die Eingangsdaten in einem 24 Bit breiten Datentyp vor, haben aber weiterhin ihren alten Wertebereich. Die Variablen *acc1* und *acc2* haben eine Wortbreite von 48 Bit und nehmen das Ergebnis der Multiplikation auf. Auch die folgende Subtraktion ist ein 48 Bit Befehl. Die Umwandlung zurück nach 24 Bit erfolgt mit dem letzten Befehl. Er addiert eine 24 Bit mit einer 48 Bit Zahl und rundet das Ergebnis auf 24 Bit ab. Die niederwerten 24 Bit gehen bei diesem Vorgang verloren, die Genauigkeit nimmt ab.

Durch die Umstellung des Zahlenformates konnte die Laufzeit der Feature-Extraktion des Referenz Experiments auf dem SAA um 99% und auf dem ARM-Core um 81% reduziert werden.

Eine fixed-point Implementierung bringt verschiedene nichtlineare Effekte mit sich, wie z.B. durch Sättigung oder Rundungen. Um die Auswirkungen dieser Effekte auf die Klassifikationsgüte zu ermitteln wurde eine Versuchsreihe durchgeführt, in der eine floating-point Implementation für Feature-Extraktion und Klassifikation mit einer fixed-point Implementation für Feature-Extraktion und Klassifikation miteinander verglichen wurden. Tabelle 17 gibt den Versuchsaufbau für die einzelnen Experimente an. Die verwendeten Features sind in Tabelle 4 aufgelistet.

Nummer	Klassifikator	Feature																				
		F1	F2	F3	F4	F5	F6	F7	F8	F9	F10	F11	F12	F13	F14	F15	F16	F17	F18	F19	F20	F21
1	KNN 3	X	X		X			X				X						X				
2	KNN 5	X	X		X			X				X						X				
3	KNN 3	X	X		X		X	X	X	X	X	X	X	X	X	X		X				
4	KNN 5	X	X		X		X	X	X	X	X	X	X	X	X	X		X				
5	KNN 3	X		X	X			X				X						X				
6	KNN 5	X		X	X			X				X						X				
7	KNN 7	X		X	X			X				X						X				
8	KNN 1								X		X	X	X	X	X				X			
9	KNN 3								X		X	X	X	X	X				X			
10	KNN 5								X		X	X	X	X					X			
11	KNN 1																		X			
12	KNN 1	X				X					X			X			X		X	X		
13	KNN 1	X			X	X					X	X		X			X		X	X		
14	KNN 1	X		X	X	X		X			X	X		X			X		X	X		
15	KNN 1	X	X	X	X	X		X			X	X		X			X		X	X		
16	KNN 3																		X			
17	KNN 3	X			X		X			X	X						X		X			
18	KNN 3	X		X	X		X	X	X	X	X	X					X		X			
19	KNN 3	X	X	X	X		X	X	X	X	X	X	X	X	X		X		X			X
20	KNN 3	X	X	X	X		X	X	X	X	X	X	X	X	X		X		X	X		X
21	KNN 5																		X			
22	KNN 5		X														X		X			
23	KNN 5		X		X								X				X		X	X		
24	KNN 5		X		X			X					X	X			X		X	X		
25	KNN 5		X		X			X					X	X			X		X	X		

Tabelle 17
In den Experimenten verwendete Feature-Kombinationen

In Abbildung 5-10 sind die Ergebnisse der Versuchsreihe dargestellt. Der linke Balken steht für die Klassifikationsgüte einer floating-point Implementation der Feature-Extraktion und des Klassifikators. Der rechte Balken stellt entsprechend die Klassifikationsgüte dar, die für eine fixed-point Implementation der Funktionen erreicht wird.

Es treten sowohl Fälle auf, in denen die eine Implementation bessere Resultate erzielt, als auch Fälle in denen die andere Implemenation besser ist oder aber die Ergebnisse gleich sind. Aufgrund der stark nichtlinearen Zusammenhänge haben die Quantisierungsoperationen der fixed-point Implemenation mal einen positiven, mal einen negativen Einfluss auf die Klassifikationsgüte. Im Mittel über die Experimente ist die fixed-point Implementation um etwa 2% besser als die floating-point Implementation.

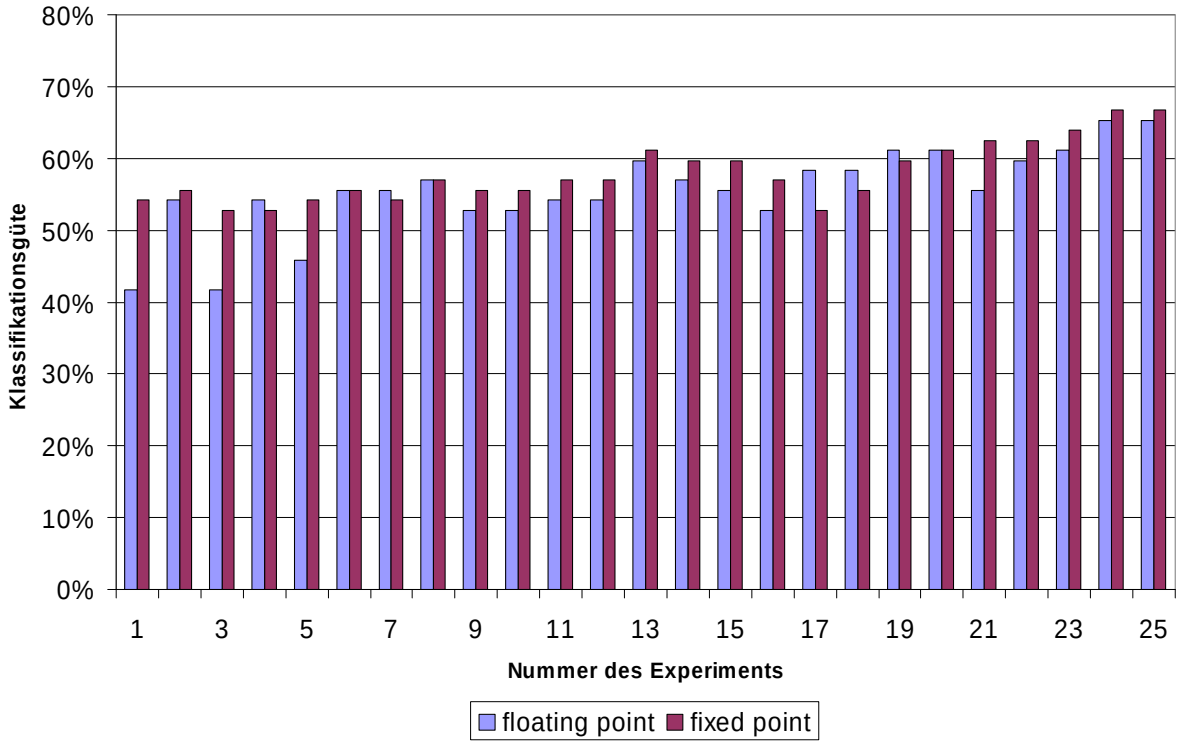


Abbildung 5-10
Klassifikationsgüten des KNN Klassifikators für eine floating-point Implementation und eine fixed-point Implementation Der jeweils linke Balken stellt die Klassifikationsgüte der kompletten floating-Point Implementation dar, der rechte Balken die einer fixed-Point Implemenation

Neben den näher erläuterten Möglichkeiten zur Laufzeitoptimierung gibt es noch weitere Optimierungsansätze, auf die im Rahmen dieser Arbeit aber nicht näher eingegangen wurde.

Eine Möglichkeit besteht in einer Unterabtastung der Musikstücke. Die Features würden nicht aus jedem der Zeitfenster extrahiert, sondern nur aus einer Untermenge der Fenster. Einer geeigneten Auswahl dieser Untermenge käme dabei eine große Bedeutung zu.

Eine andere Möglichkeit besteht in der mathematischen Approximation einzelner Funktionen. Beispielsweise lassen sich die Wurzel-Funktion oder der Logarithmus durch einfachere und damit schneller zu berechnende Funktionen approximieren. Auch für viele der komplexeren mathematischen Berechnungen zur Extraktion der Features werden sich passende Näherungen finden lassen.

Der überwiegende Teil der Musikstücke liegt im mp3-Format oder einem verwandten Format vor, das die Musik im Frequenzbereich codiert. Als weitere Möglichkeit zur Optimierung bietet es sich an, dies auszunutzen und zu prüfen, welche der Features im Frequenzbereich von einer direkten Nutzung der Musikdaten profitieren können, ohne diese zuerst komplett zu dekodieren und anschließend wieder in den Frequenzbereich zu transformieren.

5.4. Effizienzanalyse

In diesem Kapitel wird sich der Effizienz der Musikklassifikation gewidmet.

Als Effizienz ist das Verhältnis aus einem Nutzen zu den dafür aufgewendeten Kosten definiert. Je größer der Nutzen und je geringer die Kosten sind, umso größer ist die Effizienz.

Übertragen auf das Problem der Musikklassifikation entspricht der Nutzen einer Klassifikationsgüte. Die dafür aufzuwendenden Kosten lassen sich hier in erster Näherung annehmen als die erforderliche Laufzeit zur Extraktion der verwendeten Features.

In den folgenden Abbildungen sind die Laufzeiten und Klassifikationsgüten für den k Nearest-Neighbor Klassifikator auf einem PC und dem SAA für einige ausgewählte Punkte des Entwurfsraumes dargestellt. Der Klassifikator arbeitet auf den Daten des Testdatensatzes „Satz 1“, der aus 72 Musikstücken aus sechs Genres besteht, und betrachtet fünf Nachbarn. Die Extraktions-Laufzeiten wurden für zehn Minuten Musik ermittelt. In beiden Abbildungen ist die Laufzeit in Sekunden auf der linken Ordinate und der Klassifikationsgüte auf der rechten Ordinate eingetragen.

In Abbildung 5-11 sind die Daten für den betrachteten PC7 abgebildet. Der erste Balken betrachtet eine Klassifikation unter Verwendung aller 19 im Rahmen dieser Arbeit implementierten Features. Die Klassifikationsgüte dieses Versuchs liegt bei 50%. Durch Codeoptimierung konnte die Laufzeit um einen Faktor von knapp fünf gesenkt werden, während die Klassifikationsgüte unverändert blieb. Hauptsächlich kamen dabei Optimierungen zum Einsatz, die Funktionsaufrufe durch Tabellenzugriffe ersetzen (siehe 5.3.2). Diese Laufzeit stellt für die folgenden Versuche die Obergrenze dar. Als nächstes wurden die Features des Referenz-Feature-Satzes betrachtet, ihre Laufzeit ist um einen Faktor zwei besser als bei Verwendung aller Features, die Klassifikationsgüte steigt um elf Prozent. Die beste Klassifikationsgüte wurde durch die vollständige Suche ermittelt, deren Ergebnis im vierten Balken dargestellt ist. Die Dauer um den optimalen Feature-Satz zu ermitteln lag in diesem Fall bei knapp neun Stunden. Mit dem so ermittelten Feature-Satz wird eine Güte von 70% erreicht, die um 9% über derjenigen des Referenz-Feature-Satzes liegt, während die Laufzeit zur Extraktion der Features auch wieder zunimmt. Im Vergleich zur Nutzung aller Features lässt sich eine Steigerung der Klassifikationsgüte um 20% bei gleichzeitiger Reduktion der erforderlichen Laufzeit um 10% erzielen. Einen Kompromiss stellen die Ergebnisse des Mann-Whitney Tests dar, der im Gegensatz zur vollständigen Suche auch bei steigender Anzahl Features noch in handhabbarer Zeit durchführbar ist. Mit einer Dauer des kompletten Tests von weniger als einer Stunde wurde ein Feature-Satz gefunden, dessen Klassifikationsgüte zwischen der des Refe-

[7] Pentium 4, 3,2 GHz

renz-Feature-Satzes und dem in der vollständigen Suche ermittelten Feature-Satzes liegen. Die Laufzeit zur Extraktion der Features liegt allerdings wieder etwas höher als für den optimalen Feature-Satz.

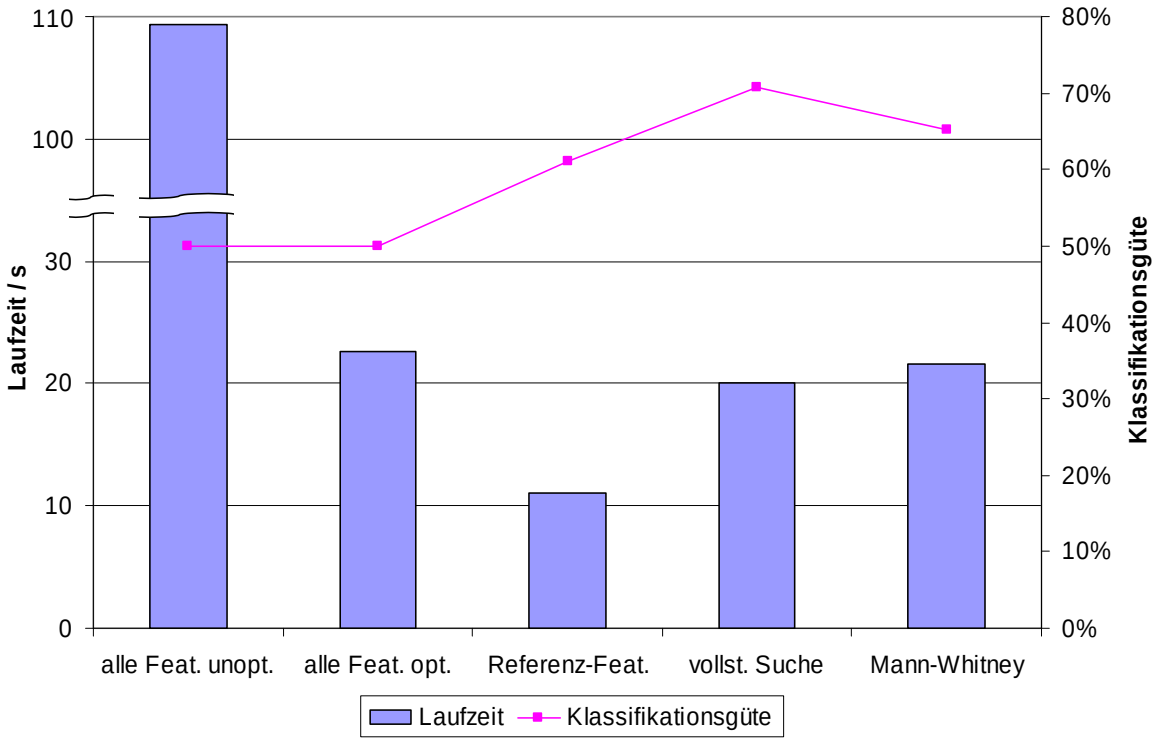


Abbildung 5-11
Laufzeit und Klassifikationsgüte des KNN5 Klassifikators auf einem PC (6 Genres, 72 Musikstücke)

In Abbildung 5-12 sind die Daten des SAA abgebildet. Die ersten beiden Balken stellen wieder die Nutzung aller Features dar. Der verhältnismäßig geringe Laufzeitunterschied zwischen den beiden Balken erklärt sich dadurch, dass viele Optimierungen bei der Portierung des Codes vom PC bereits übernommen wurden. Die weiteren Verbesserungen wurden erreicht durch die weitere Optimierung von Schleifen, die Nutzung von Hardwareloops und durch die Ausnutzung der Möglichkeiten des SAA manche Befehle parallel auszuführen. Die Nutzung des Referenz-Datensatzes führt wieder zu einer um etwa 10% verbesserten Klassifikationsgüte, während die Laufzeit um knapp 40% sinkt. Die vollständige Suche ermittelt einen Feature-Satz, der die beste erzielbare Klassifikationsgüte aufweist, welche um 10% über der des Referenz-Feature Satzes liegt und um 20% über der, die für die Nutzung aller Features erreicht wird. Gegenüber der Nutzung aller Features sinkt die erforderliche Laufzeit zur Extraktion der Features um 20%. Als Kompromiss liefert der Mann-Whitney Test wieder einen Feature-Satz, dessen Klassifikationsgüte zwischen der des Referenz-Feature-Satzes und der vollständigen

Suche liegt, während die Laufzeit gegenüber den Features der vollständigen Suche wieder ansteigt.

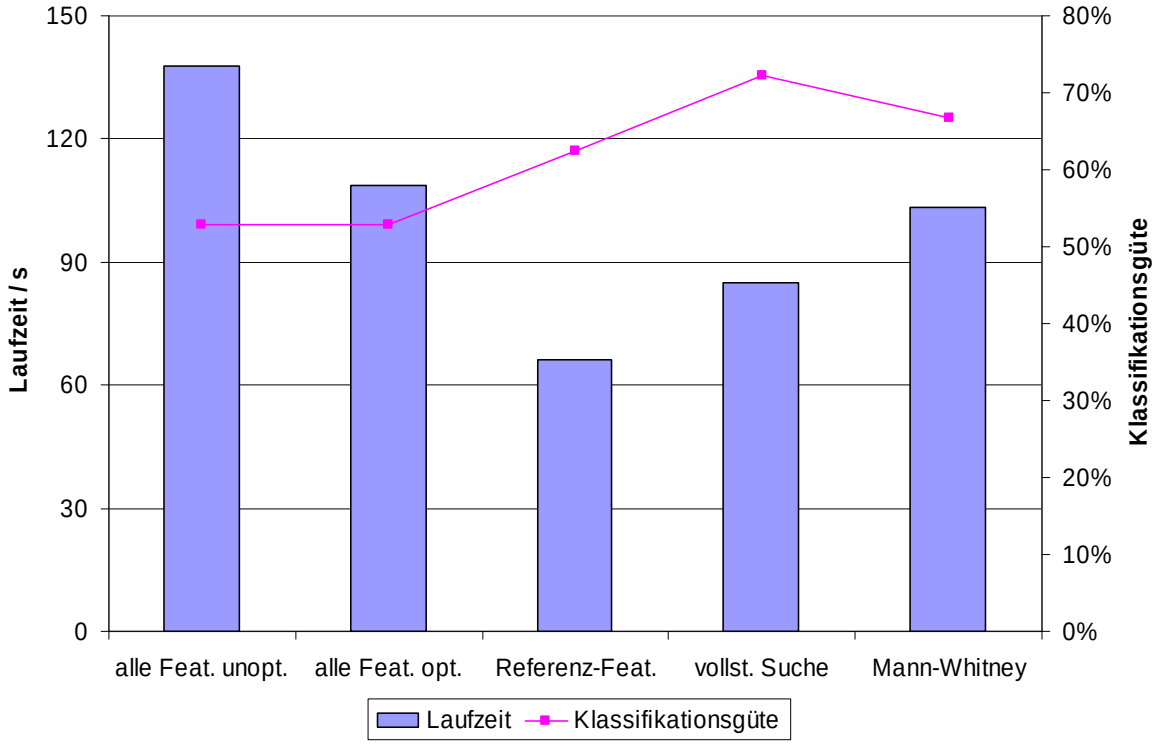


Abbildung 5-12
Laufzeit und Klassifikationsgüte des KNN5 Klassifikators auf dem SAA (6 Genres, 72 Musikstücke)

Mit weiteren Optimierungen, z.B. durch spezielle Hardware-Befehle (siehe Kapitel 6) wird die Laufzeit weiter sinken, während die Klassifikationsgüte durch weitere Features und bessere Algorithmen noch steigen wird. Eine Laufzeit von ca. einer Minute für die Extraktion der Features aus zehn Minuten Musik scheint realistisch erreichbar zu sein. Damit würde die Bearbeitungszeit je Musikstück bei 10-30 Sekunden liegen. Zusammen mit einer Klassifikationsgüte von 70% für sechs Genres wird bald die Grenze erreicht sein, an der eine automatisierte Musikklassifikation eine vom Anwender akzeptierte Laufzeit und Klassifikationsgüte erreicht.

6. INSTRUKTIONSANALYSE

Aufgrund des hohen Rechenaufwandes bei gleichzeitig sehr regelmäßig abzuarbeitenden Programmteilen bietet es sich an, den Code auf dem DSP auf sein mögliches Optimierungspotential durch geeignete, programmspezifische Hardware-unterstützte Befehle (Custom Instructions) hin zu untersuchen. Die Betrachtung kann sowohl feingranular als auch grobgranular erfolgen.

In der feingranularen Betrachtungsweise wird das Programm auf der Grundlage der einzelnen ausgeführten Befehle betrachtet. Der Code des Programms wird dazu auf häufig ausgeführte Instruktionsfolgen untersucht, die als Ansatzpunkte für mögliche Optimierungen dienen. Hierfür werden aus einer Befehlsfolge des Programms alle Befehlsbäume extrahiert. Aus diesen werden die häufig ausgeführten Befehlsfolgen ausgewählt und auf ihre Umsetzbarkeit in eine Hardware-Instruktion sowie den erzielbaren Laufzeitgewinn hin untersucht.

Der grobgranulare Ansatz dagegen betrachtet keine einzelnen Befehlsfolgen sondern ganze Funktionen. Hier stellt sich die Frage, welche Funktionen sich mit welchen Laufzeiten komplett in Hardware realisieren lassen, und welches Optimierungspotential sich dadurch ergibt.

Zunächst wird in den folgenden beiden Abschnitten dem feingranularen Ansatz nachgegangen und Befehlsbäume extrahiert und untersucht. Im darauf folgenden Abschnitt wird das Optimierungspotential durch ganze Funktionen untersucht.

6.1. Extraktion geeigneter Befehlsbäume

Zur Extraktion der Befehlsbäume wurde ein Skript in der Sprache *perl*, das für diesen Zweck in [39] erstellt wurde, an die verwendeten Formate und Gegebenheiten angepasst.

Auf Basis des Profiling-Durchlaufes über die Feature-Extraktion aus einem Musikstück liegt eine Folge der Befehle vor. Diese ist auf die lineare Abfolge innerhalb einzelner Basisblöcke[8] beschränkt. Für jeden Befehl liegt die Anzahl der Taktzyklen vor, die zu seiner Ausführung

[8] Basisblöcke sind Programmabschnitte, die linear von Anfang bis Ende durchlaufen werden. Am Ende steht eine Verzweigung, die den Programmablauf an einem von zwei möglichen Zielen fortsetzt.

benötigt wurde. Da auf dem betrachteten DSP alle Befehle nur einen Taktzyklus zur Ausführung benötigen[9], ist diese auch gleichbedeutend mit der Häufigkeit der Ausführung.

Das Skript durchläuft die Profiling-Datei und betrachtet dabei jeden Befehl als möglichen Ausgangspunkt für einen neuen Befehlsbaum. Dazu wird zu jedem Operanden des aktuellen Befehls diejenige Instruktion gesucht, in der dieser Operand berechnet wird. Von jedem der gefundenen Befehle wird die Suche rekursiv fortgesetzt.

Die Suche wird durch zwei Variablen gesteuert, die maximale Baumtiefe und die maximale Suchtiefe. Die maximale Baumtiefe gibt die Tiefe der Instruktionsbäume an, ab der Operanden eines Zweiges nicht mehr weiter verfolgt werden. Die maximale Suchtiefe gibt an, wie viele vorherige Zeilen nach der Berechnung eines Operanden durchsucht werden, bevor die Suche abgebrochen wird. Wird die Grenze eines Basisblocks oder ein Speicherzugriff erreicht, so wird die Suche ebenfalls beendet.

Wurde ein Befehlsbaum gefunden, so wird die Anzahl der Zyklen des Ausgangsbefehls als Ausführungshäufigkeit des Baumes angenommen. Der neue Instruktionsbaum wird mit allen bereits gefundenen verglichen. Existiert er bereits, so wird nur der Zähler der Ausführungshäufigkeit erhöht, ansonsten wird er der Liste hinzugefügt.

Im Folgenden ist der Ablauf an einem Beispiel erläutert. Gegeben sei der in Abbildung 6-1 auf der linken Seite angegebene Assemblercode. Als Ausgangspunkt der Suche diene der *sub*-Befehl in Zeile n. Die Eingangsoperanden dieses Befehls sind die Register R2 und R4. Nun wird, beginnend von Zeile n, in den vorhergehenden Zeilen nach den Befehlen gesucht, die diese Register berechnen. Fündig wird man in Zeile n-1, in der zwei Befehle parallel ausgeführt werden[10], die genau diese beiden Register berechnen. Ausgehend von diesen beiden Befehlen wird nun wieder rekursiv nach den Instruktionen gesucht, die die verwendeten Operanden berechnen. Für den *asl*-Befehl ist dies die Zeile n-3 für das Register R2. Für den *add*-Befehl lässt sich der letzte Schreibzugriff auf das Register R3 in der Zeile n-2 finden. Damit kann aus dem angegebenen Assemblercode der in Abbildung 6-1 auf der rechten Seite abgebildete Befehlsbaum extrahiert werden.

[9] Die Ausnahme bilden Sprungbefehle, die zwei Taktzyklen benötigen. Da diese aber den sequentiellen Programmfluss unterbrechen und sich nicht zur Auslagerung eignen, beeinträchtigen sie die weitere Betrachtung nicht.

[10] Tatsächlich kann der MMDSP diese beiden Befehle nicht parallel ausführen, sie dienen hier nur zur Demonstration.

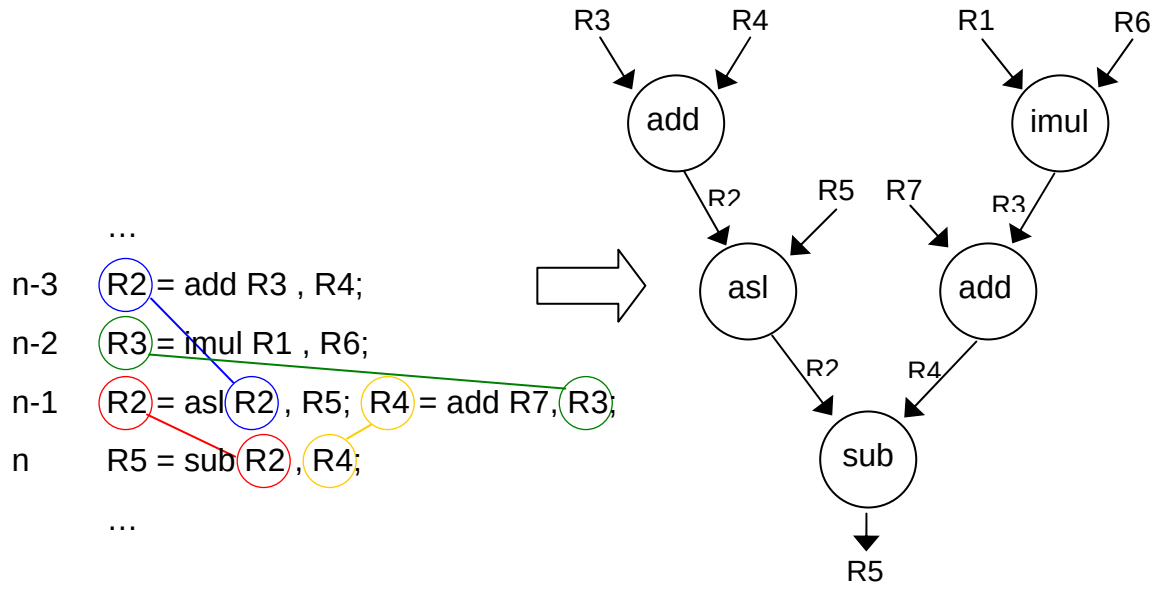


Abbildung 6-1
Extraktion eines Instruktionsbaumes.

Am Ende des Skripts werden die gefundenen Bäume in folgender Form in einer Datei ausgegeben:

```
Neuer Baum Nr. 19:
        4)add(2)        3)imul(1)
        2)asl(0)        1)add(0)
        0)sub(-1)
Anzahl Aufrufe: 2958
Erster Aufruf: 174
```

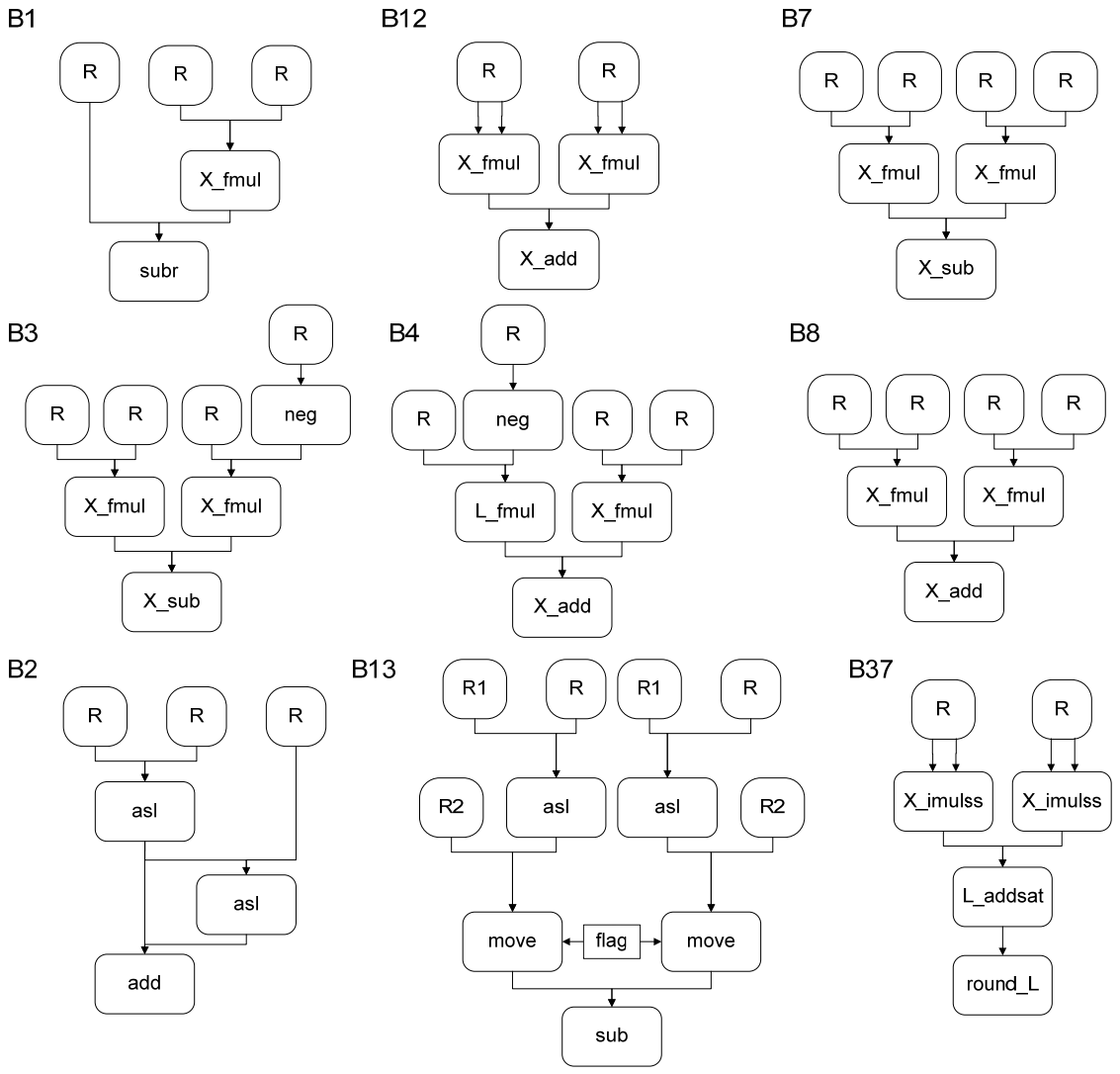
Die Verknüpfung der Befehle untereinander zu einem Befehlsbaum lässt sich über die Zahlen hinter den Befehlen nachvollziehen. Sie geben die Nummer des Befehls an, der im Instruktionsbaum das Ergebnis des Befehls erhält. So besagt in diesem Beispiel die 0 nach dem *asl*, dass der *sub*-Befehl das Ergebnis dieser Berechnung als Eingang erhält.

Des Weiteren sind die Anzahl der Aufrufe des Instruktionsbaumes und die Zeilennummer des ersten Vorkommens vermerkt.

6.2. Analyse der Befehlsbäume

Aus insgesamt 21 einzelnen Funktionen, die die Basis für die Extraktion der wichtigsten Features bilden, wurden mehrere hundert Befehlsbäume extrahiert. Aus diesen wurden zunächst 41 verschiedene Befehlsbäume ermittelt, für die eine nähere Betrachtung sinnvoll erschien. Bei der Auswahl ignoriert wurden Bäume, die keine Optimierungsmöglichkeiten bieten, wie z.B. aufeinender folgende Kopieroperationen zwischen Registern oder Ladeoperationen gefolgt von einem Befehl.

In der folgenden Zeichnung sind von diesen Bäumen diejenigen abgebildet, die im weiteren Verlauf näher betrachtet wurden.



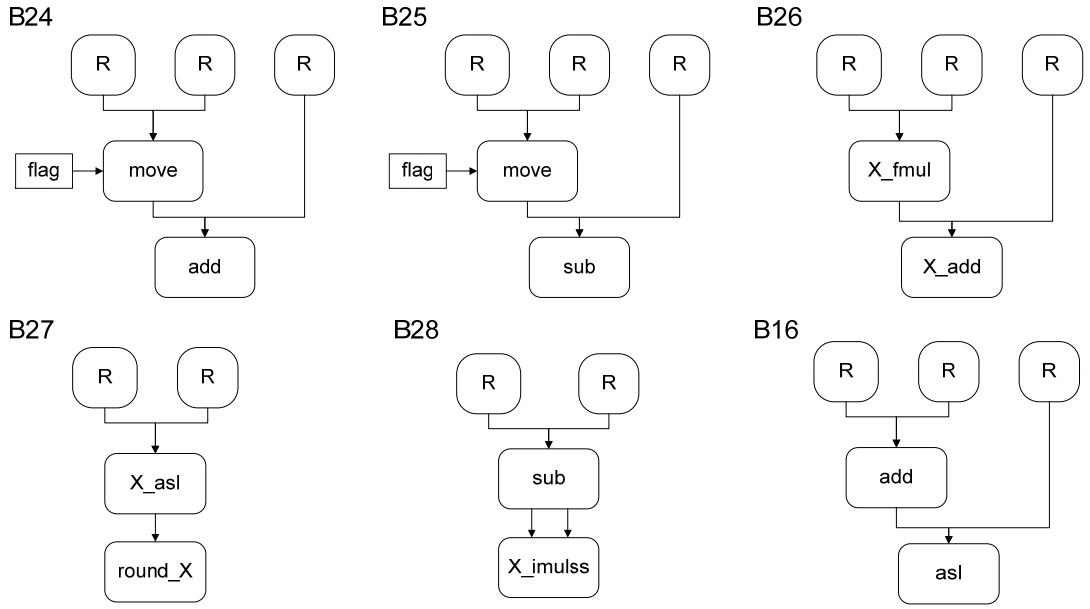


Abbildung 6-2
Einige betrachtete Instruktionsbäume.

In Abbildung 6-2 sind einige der betrachteten Instruktionsbäume abgebildet. Die Nomenklatur B<x> entspricht dabei der internen Kennzeichnung der Bäume. „R“ steht für ein Register, das einen Eingangswert für den Befehlsbaum darstellt. Der doppelte Pfeil, z.B. in B12, gibt an, dass beide Eingangswerte des Befehls identisch sind, der Wert im Falle von B12 also quadriert wird.

Aus diesen Bäumen ergaben sich 12 verschiedene Instruktionsbäume, die als Hardware-Funktion attraktiv realisierbar erschienen. Zusätzlich zu den betrachteten Instruktionsbäumen wurden drei spezielle Multiplikationen und die Division aufgenommen, da der SAA nicht über diese als Hardware-Befehl verfügt. Zur näheren Analyse des Optimierungspotentials einer möglichen Einführung weiterer Hardware-unterstützter Custom Instruktionen wurde für die Bäume der erzielbare Verbesserungsfaktor der Laufzeit ermittelt.

Annahme: Verbesserung um Faktor r(100%) für
p · 100% der Befehle wirksam
(1-p) · 100% der Befehle nicht wirksam

$$r_{ges} = \frac{\tau_{CPU,ohne\ Verbesserung}}{\tau_{CPU,mit\ Verbesserung}} \geq 1$$

$$r_{ges} = \frac{1}{(1-p) + \frac{p}{r(100\%)}} \leq \frac{1}{1-p}$$

Nach Amdahl's Law berechnet sich der Verbesserungsfaktor r_{ges} als Quotient aus der Laufzeit ohne Verbesserung und der Laufzeit mit Verbesserung. Als Laufzeit ohne Optimierungsmaßnahmen wurde die Laufzeit der in diesem Kapitel betrachteten 21 Funktionen gewählt. Als Laufzeit der Befehlsbäume wurde jeweils die Laufzeit der Befehle unter Berücksichtigung eventueller Parallelität bestimmt. Für die Hardware-Version der Befehlsbäume wurde jeweils eine Laufzeit von einem Takt angesetzt. Multipliziert mit der Häufigkeit des Auftretens der Bäume ergeben sich so die Laufzeiten vor und nach der Umsetzung auf Hardware-Befehle.

Die so ermittelten Verbesserungsfaktoren dieser Bäume sind in der folgenden Abbildung dargestellt.

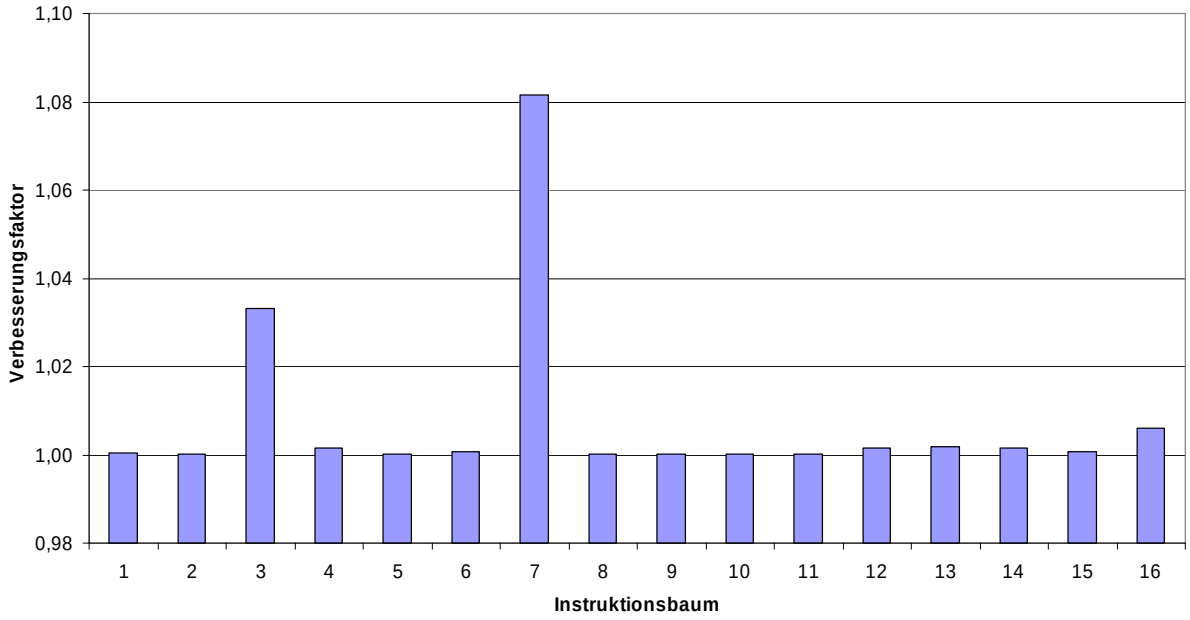


Abbildung 6-3 Verbesserungsfaktoren der Instruktionsbäume.

In Abbildung 6-3 sind die erzielbaren Verbesserungsfaktoren für die einzelnen Bäume abgebildet. Auf der Ordinate sind die Verbesserungsfaktoren aufgetragen, während auf der Abszisse die Befehlsbäume durchnummeriert sind. Der Verbesserungsfaktor der zusätzlich betrachteten Division ist unter der Nummer 16 dargestellt. Es zeigt sich, dass sich hauptsächlich zwei Bäume für eine Hardware-Implementation anbieten. Ihre Verbesserungsfaktoren betragen 3% und 8%. Ob sich die Einführung von Custom Instructions für diese beiden Befehle lohnt, hängt von ihrem Kosten-Nutzen Verhältnis ab. Es muss die zu erwartende Laufzeitverbesserung gegen Chipfläche, veränderte Verlustleistung und Implementationsaufwand abgewogen werden. Eine solche Betrachtung wurde im Rahmen dieser Arbeit nicht durchgeführt.

In der folgenden Abbildung sind die beiden Instruktionsbäume in einer hardwarenahen Darstellung abgebildet.

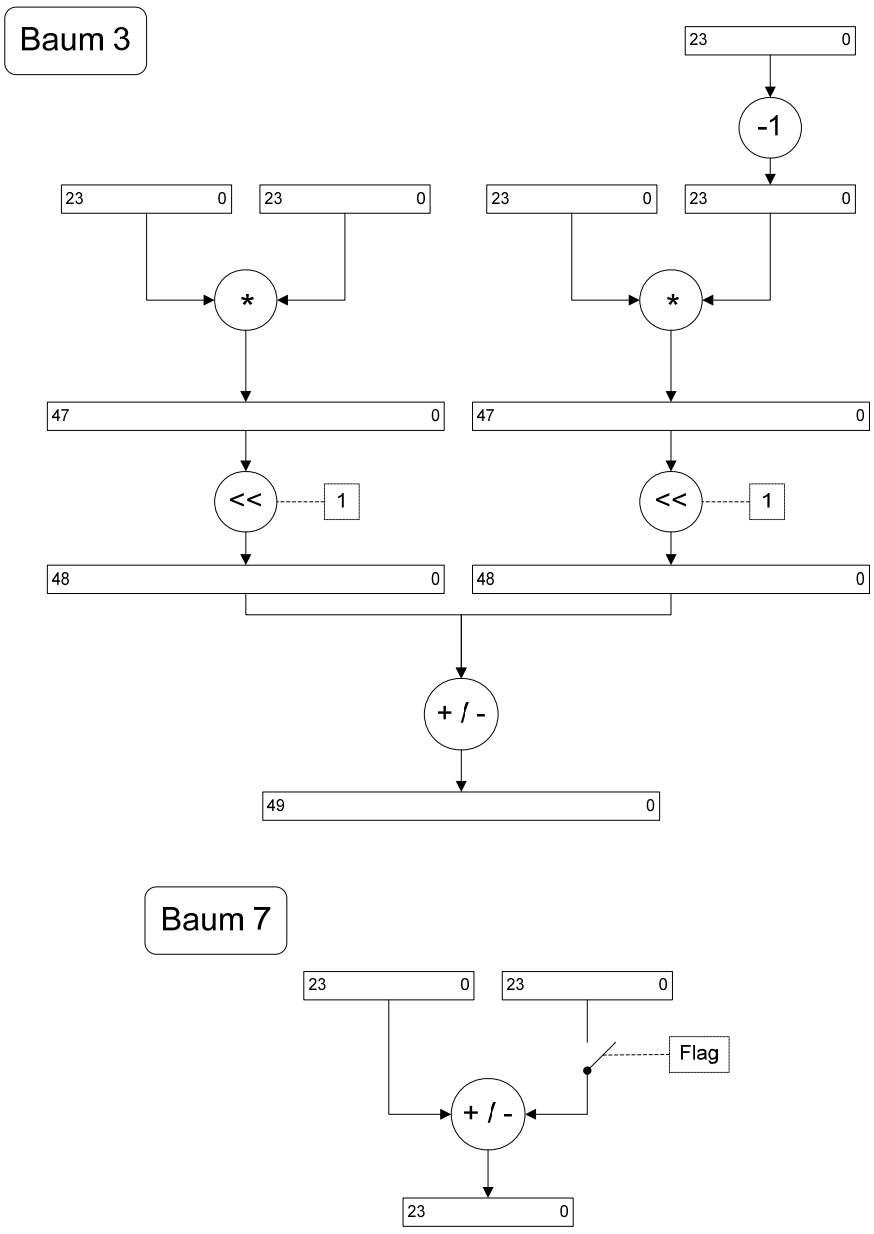


Abbildung 6-4
Hardwarenahe Darstellung der Instruktionsbäume mit dem besten Verbesserungsfaktor

Abbildung 6-4 stellt ein detailliertes Flussdiagramm der beiden Bäume dar. Die Eingangsdaten beider Bäume haben eine Wortbreite von 24 bit, ihre Ausgangswerte haben eine Wortbreite von 56 bit für den Baum 3 bzw. 24 bit für den Baum 7. Beide Bäume sind durch Zusammenfassung von Instruktionsbäumen entstanden, die sich nur durch eine Addition oder Subtraktion unterschieden. In einer Hardware-Implementation unterscheiden sich beide Operationen kaum. Sie können ohne Laufzeitunterschied von einem Addierer durchgeführt werden,

der in der Abbildung durch den Knoten „+/-“ dargestellt ist. Der dargestellte Baum 3 ist aus den in Abbildung 6-2 abgebildeten Bäumen B3 und B4 hervorgegangen, der Baum 7 aus den Bäumen B24 und B25. Der Knoten „-1“ in Baum 3 stellt eine Negation dar, der Knoten „<<“ im selben Baum einen Linksshift. Baum 3 berechnet damit die Addition (oder Subtraktion) von zwei Zahlen, die jeweils aus einer speziellen fixed-point Multiplikation hervorgegangen sind. In Baum 7 wird der zweite Operand der Instruktion abhängig von einem Flag verwendet oder durch Null ersetzt. Das Flag ist ein Statusflag des Prozessors, das durch einen Vergleich gesetzt wird. Der Befehlsbaum stellt damit eine bedingte Addition bzw. bedingte Subtraktion dar.

Bei den beiden ausgewählten extrahierten Bäumen handelt es sich um Instruktionsfolgen, die in einer 130 nm Technologie bei einer Taktfrequenz von 133 MHz (Technologie und Taktfrequenz des SAA) in einem Taktzyklus abgearbeitet werden können.

6.3. Optimierung auf Funktionsebene

Neben der Optimierung einzelner Befehlsbäume durch passende Hardware-Befehle ließe sich auch durch die Auslagerung ganzer Funktionen in eine Hardware-Funktion eine Beschleunigung erzielen. Eine Untersuchung der gemeinsamen Laufzeit aller implementierten Features zeigt, dass die Aufrufe der FFT fast 40% der Laufzeit ausmachen. Damit stellt die FFT einen sehr attraktiven Kandidaten für eine Hardware-Implementation dar.

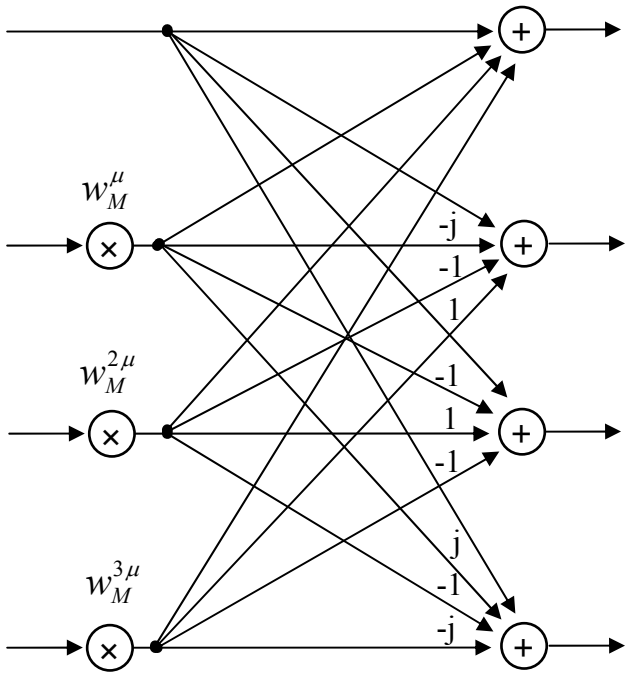
In Abbildung 6-6 ist das Blockdiagramm einer möglichen Implementation der FFT als Hardware-unterstützte Funktion abgebildet. In diesem Diagramm steht S für einen (Zwischen-) Speicher und R4 und R2 für die Butterfly-Operation eines Radix-4 bzw. Radix-2 Algorithmus. Die Zahlen geben Verzögerungen in der Verarbeitung zwischen den einzelnen Stufen an.

Zunächst wird der Radix-4 Algorithmus kurz erläutert, bevor weiter auf das Blockdiagramm der möglichen Implementation der FFT eingegangen wird.

Der Radix-4 Algorithmus arbeitet ähnlich wie der Radix-2 Algorithmus (siehe Abschnitt 2.2.3.1). Hier werden die Eingangsdaten allerdings nicht in zwei, sondern in vier Teilgruppen aufgeteilt.

Die dadurch erforderliche Korrekturoperation erfordert wieder nur wenige Multiplikationen gefolgt von Additionen. Abbildung 6-5 zeigt den Aufbau einer solchen Butterfly-Operation sowie die Berechnungsvorschrift als Matrix-Multiplikation. Durch die kürzere Länge der Teilgruppen benötigt eine FFT nach dem Radix-4 Algorithmus weniger Stufen zur Berech-

nung des Ergebnisses als der Radix-2 Algorithmus. Allerdings ist der Algorithmus auf FFTs beschränkt, deren Länge eine Potenz von 4 ist.



$$\begin{bmatrix} X(0) \\ X(1) \\ X(2) \\ X(3) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -j & -1 & j \\ 1 & -1 & 1 & -1 \\ 1 & j & -1 & -j \end{bmatrix} \begin{bmatrix} W_M^0\, x(0) \\ W_M^{\mu}\, x(1) \\ W_M^{2\mu}\, x(2) \\ W_M^{3\mu}\, x(3) \end{bmatrix}$$

Abbildung 6-5
Die radix-4 Butterfly-Operation.

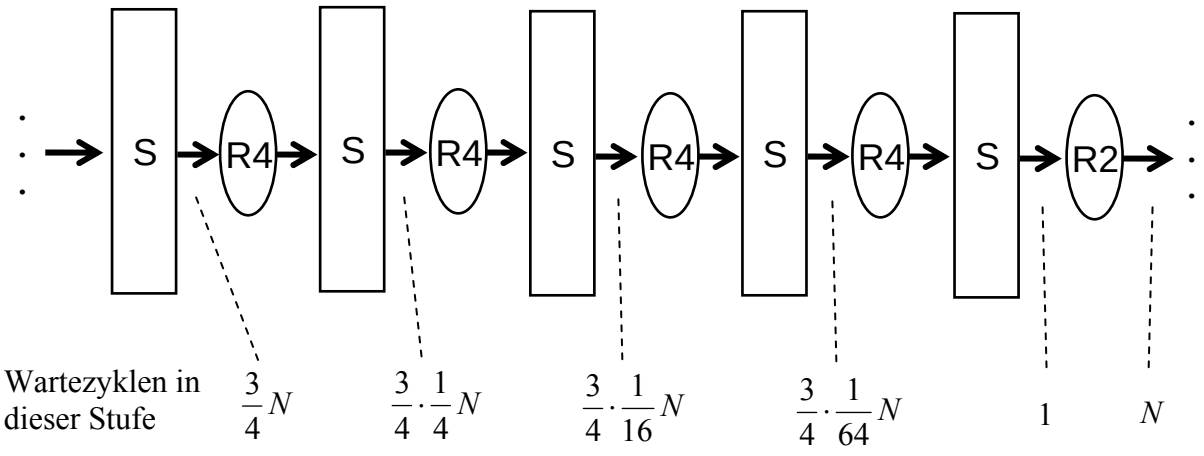


Abbildung 6-6
Blockdiagramm einer möglichen Implementierung der FFT als Hardwarefunktion.

In Abbildung 6-6 ist das Blockdiagramm für eine Implemetierung der FFT als Hardware-Funktion abgebildet. [52] Die Daten werden linear an die Funktion übergegeben. Nach $\frac{3}{4}N$ Takten liegen genug Daten vor, damit die erste Radix-4 Butterfly-Operation den ersten Ausgangswert berechnen kann. Dieser wird in den zweiten Zwischenspeicher geschrieben. In jedem folgenden Takt wird ein weiterer Ausgangswert berechnet, der ebenfalls in den zweiten Zwischenspeicher geschrieben wird. Nach $\frac{3}{4}\cdot\frac{1}{4}N$ weiteren Takten liegen im zweiten Zwischenspeicher genug Daten vor, damit die zweite Radix-4 Butterfly-Operation den ersten Ausgangswert berechnen kann. Dieser wird in den folgenden Zwischenspeicher geschrieben. Mit den beiden folgenden Stufen verhält es sich ebenso, hier nehmen die Verzögerungen aufgrund des geringeren Umfangs der jeweiligen Teiloperationen weiter ab. In der letzten Stufe folgt eine Radix-2 Butterfly-Operation, da eine FFT der Länge $N = 512$ mittels eines Radix-4 Algorithmus in dieser Stufe nicht möglich ist. Die letzte Verzögerung gibt die Zeit an, die vergeht, bis das komplette Ergebnis der FFT ausgegeben ist.

Die Gesamtlaufzeit dieser Hardware-FFT (für eine Fenstergröße von N = 512) lässt sich damit mit 1023 Takten abschätzen.

Die im Rahmen dieser Arbeit implementierte Software-FFT ist angelehnt an eine optimierte Software-Implementation in jAudio (siehe Abschnitt 4.2). Für die fixed-point Implementation der FFT auf dem SAA wurde in Abschnitt 5.1 eine Laufzeit von 46900 Taktzyklen ermittelt. Mit dieser Abschätzung ist die Hardware-FFT ca. um einen Faktor 50 schneller als die per Software ausgeführte FFT.

In Abschnitt 5.1 wurden die Laufzeiten für die Extraktion aller Features auf dem SAA ermittelt. Die Laufzeit zur Ermittelung der Autokorrelationsfunktion im Zeitbereich wird im Folgenden nicht betrachtet, da die Berechnung effizienter über den Frequenzbereich auszuführen ist. Als Laufzeit ohne Verbesserung wurde die Gesamtlaufzeit dieser Funktionen gewählt, die insgesamt drei Aufrufe der FFT enthält, welche damit 41% der Laufzeit ausmachen. Die Laufzeit mit Verbesserung wurde ermittelt, indem wieder die Summe der Laufzeiten gebildet wurde, diesmal allerdings unter Annahme der ermittelten 1023 Takte für die FFT.

Nach Amdahl's Law beträgt der Verbesserungsfaktor bei Nutzung der Hardware-FFT somit 1,67.

7. ZUSAMMENFASSUNG

Bei einem stetigen Wachstum der Größe digitaler Musikarchive gewinnen Methoden zur automatischen Analyse und Sortierung der Musikstücke zunehmend an Bedeutung. Insbesondere auf Mobilgeräten rückt diese Aufgabenstellung zunehmend ins Interesse von Entwicklern und Anwendern. Im Rahmen dieser Arbeit wurde die Klassifikation von Musikstücken anhand von akustischen Merkmalen betrachtet, die aus den Abtastwerten der Musikstücke extrahiert wurden.

Auf einem PC (P IV, 3,2 GHz) wurde eine Referenz-Implementation zur Extraktion einer Vielzahl Musik-Features vorgenommen und diese Feature-Extraktoren wurden anschließend auf den Smart Audio Accelerator (SAA) der Nomadik-Plattform, eines digitalen Signalprozessors für mobile Endgeräte, portiert. Weiterhin wurden Vergleiche zu Implementierungen auf einem OMAP 2420 DSP durchgeführt.

Zunächst wurden Experimente aus der Literatur nachgestellt und eigene Experimente mit heuristischen Feature-Sätzen durchgeführt. Die Klassifikationsergebnisse für sechs Genres lagen dabei im Durchschnitt zwischen 40% und 60%.

Um die optimalen Feature-Sätze zu ermitteln wurde eine vollständige Suche über alle möglichen Feature-Sätze und vier unterschiedliche Klassifikatoren durchgeführt, die für die hier betrachteten 19 Features noch in einer akzeptablen Laufzeit durchführbar war. Die vollständige Suche konnte die Klassifikationsgüte um weitere 10% gegenüber einem Referenz-Feature-Satz steigern und gleichzeitig die dazu erforderliche Laufzeit um ca. 25% gegenüber der Nutzung aller Features reduzieren. Aufgrund ihrer exponentiell wachsenden Laufzeit wird die vollständige Suche für eine größere Anzahl zu untersuchender Features nicht mehr durchführbar sein. Daher wurde ein Mann-Whitney Test als ein statistischer Test angewendet, der in sehr kurzer Zeit eine Aussage darüber treffen kann, welche Features einen signifikanten Einfluss auf die Klassifikationsgüte haben. Die Ergebnisse zum Mann-Whitney Test zeigen auf, dass im Vergleich zur vollständigen Suche eine nur um 5% schlechtere Klassifikationsgüte bei nur um 15% schlechterer Rechenzeit gefunden wurde. Die Zeit zum Auffinden dieses Nebenoptimums betrug dafür nur ca. 5% der Zeit der vollständigen Suche.

Zur Verbesserung der Klassifikationsgüte wurde darüber hinaus ein Vertrauensmaß diskutiert, das eine Aussage darüber trifft, wie „sicher“ eine Entscheidung für eine Klasse getroffen werden kann. Durch Zurückweisung von „zu unsicheren“ Entscheidungen ließ sich die Klassifikationsgüte der klassifizierten Musikstücke verbessern.

Anschließend wurden die Laufzeiten der Feature-Extraktion näher untersucht und für die verschiedenen Plattformen gegenüber gestellt. Anhand konkreter Beispiele wurden Möglichkei-

ten zur Optimierung der Laufzeit vorgestellt. Hierbei wurde insbesondere auf die Eigenschaften des SAA eingegangen.

Zur Untersuchung der Laufzeitgewinne einer Hardware-Beschleunigung wurde eine Instruktionsanalyse durchgeführt, die häufige Befehlsfolgen ermittelt. Diese Instruktionsfolgen wurden auf ihre Umsetzbarkeit und den möglichen Verbesserungsfaktor untersucht, der sich durch eine Ersetzung der Instruktionsfolge durch einen neuen Hardware-Befehl (Custom Instruction) ergeben würde. Als weitere Möglichkeit der Hardware-Unterstützung wurde die Umsetzung ganzer Funktionen am konkreten Beispiel der FFT dargelegt.

Im Rahmen dieser Arbeit konnte nur auf eine begrenzte Auswahl von Features und Klassifikatoren eingegangen werden. Für zukünftige Untersuchungen wären weitere Features zu implementieren und auf ihren Einfluss auf die Klassifikationsgüte hin zu untersuchen. Auch weitere Rahmenbedingungen der Feature-Extraktion blieben in den Untersuchungen bisher unberücksichtig, wie z.B. die Blocklänge, eine Überlappung der Zeitfenster oder eine Beschränkung auf wenige ausgewählte Zeitfenster. Für genauere Aussagen zur Eignung einzelner Feature-Sätze sind weitere Test-Datensätze zu erstellen und mit den gleichen Features zu testen. Des Weiteren sollten zusätzliche Klassifikatoren hinsichtlich ihrer Laufzeit und ihrer Eignung zur Musikklassifikation untersucht werden.

Die automatisierten Analyse und Klassifikation von Musikstücken eines umfangreichen Musikarchivs wird – vor allem in der Unterhaltungsindustrie – in Zukunft mehr an Bedeutung gewinnen. Mit der Identifikation geeigneter Features und weiteren Optimierungen auf Software- und Hardwareebene wird die Musikklassifikation bald eine vom Anwender akzeptierte Leistung erreichen.

8. LITERATURVERZEICHNIS

[1] http://magnatune.com/

[2] STMicroelectronics: „Nomadik - Open multimedia platform for next generation mobile devices", TECHNICAL ARTICLE TA305, 2004.

[3] STMicroelectronics: „Filter Lab MMDSP+ Compiler", 2004.

[4] STMicroelectronics: „FlexGDB", 2006.

[5] STMicroelectronics: „Flexcc2 Compiler for the MMDSP+ Architecture", 2005.

[6] STMicroelectronics: „MMDSP+ Core Datasheet", 2004.

[7] STMicroelectronics: „MMDSP+ Compiler User Manual", 2006.

[8] STMicroelectronics: „NDK-10B Core Board User Manual", 2006.

[9] STMicroelectronics: „Nomadik STn8810 Data Brief", 2006.

[10] STMicroelectronics: „Nomadik STn8810 Advanced Datasheet", 2006.

[11] STMicroelectronics: „Programming User Guide MMDSP+ Tools Set", 2005.

[12] STMicroelectronics: „MMDSP+ tool set for the MMDSP+ compiler", 2005.

[13] STMicroelectronics: „mmdspsim MMDSP+ instruction set simulator", 2006.

[14] STMicroelectronics: „Nomadik STn8810 Design Advanced Data ", 2001.

[15] Geldmacher, J.: „Nomadik Audio MMDSP+ Firmware Overview", Nokia, 2006.

[16] Brecheisen, S., Kriegel, H.-P., Kunath, P., Pryakhin, A.: „Hierarchical Genre Classification for Large Music Collections", IEEE International Conference on Multimedia and Expo, pp 1385 - 1388 , 2006.

[17] Costa, C.H.L., Valle, J.D., Jr., Koerich, A.L.: „Automatic Classification auf Audio Data", 2004 IEEE lntemational Conference on Systems, Man and Cybernetics, pp 562 - 567, 2004.

[18] Pye, D.: „Content Based Methods for the Management of Digital Music", IEEE International Conference on Acoustics, Speech, and Signal Processing, Volume 6, pp 2437 - 2440, 2000.

[19] Shen, J., Shepherd, J., Ngu, A. H. H.: „Towards Effective Content Based Music Retrieval", IEEE Transactions on Multimedia, VOL. 8, NO. 6, pp 1179 - 1189, 2006.

[20] Tzanetakis, G., Cook, P.: „Musical Genre Classification of Audio Signals", IEEE Transactions on Speech and Audio Processing, VOL. 10, NO. 5, pp 293 - 302, 2002.

[21] Changsheng Xu, Maddage, N.C., Xi Shao: „Automatic Music Classification and Summarization", IEEE Transactions on Speech and Audio Processing, VOL. 13, NO. 3, pp 441 - 450, 2005.

[22] Changsheng Xu, Maddage, N.C., Xi Shao, Fang Cao, Qi Tian: „Musical Genre Classification using Support Vector Machines", Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on, pp 429 – 432, 2003.

[23] Yibin Zhang, Jie Zhou: „A Study on Content Based Music Classification", Signal Processing and Its Applications, 2003. Proceedings. Seventh International Symposium on , pp 113 – 116, 2003.

[24] K. Jensen: „Timbre Models of Musical Sounds", PhD thesis, Datalogisk Institut, Copenhagen University, Denmark, 1999

[25] Rapid-I: „Rapidminer 4.0 - User Guide, Operator Reference, Developer Tutorial", 2007.

[26] Scherer, S.: „MFCC Implementierung", Universität Ulm, 2006.

[27] Dasgupta, S.: „Learning Mixtures of Gaussians", "Foundations of Computer Science, 1999. 40th Annual Symposium on", pp 634 - 644 , 1999.

[28] Moore, A.W.: „Clustering with Gaussian Mixtures“, Carnegie Mellon University, 2004.

[29] Maillard, E.P., Gueriot, D.: „ RBF Neural Network, Basis Functions and Genetic Algorithm“, “Neural Networks,1997., International Conference on”, pp 2187 - 2192, 1997.

[30] Orr, M.J.L.: „Introduction to Radial Basis Function Networks“, University of Edinburgh, 1996.

[31] Orr, M.J.L.: „Recent Advantages in Radial Basis Function Networks“, University of Edinburgh, 1999.

[32] Orr, M.J.L.: „Optimising the Widths of Radial Basis Functions“, University of Edinburgh, 1998

[33] Schneider, M.: „Seminararbeit Maschinelles Lernen in Data Mining und Information Retrieval“, http://www.kbs.uni-hannover.de/~schneide/c45.html

[34] Ruggieri, S.: „Efficient C4.5“, IEEE Transactions on Knowledge and Data Engineering, Vol 14, No. 2, pp 438-444, 2002

[35] Haneda, M.; Knijnenburg, P.M.W.; Wijshoff, H.A.G.: „Code Size Reduction by Compiler Tuning“, SAMOS 2006, pp 186 - 195.

[36] Schröder, H.: “Mehrdimensionale Bildverarbeitung”, 1998

[37] http://www.research.att.com/~njas/oadir/

[38] http://www.digitalreview.com.ar/normaldistribution/

[39] Korb, M.: „Analyse von Kopplungsmechanismen von softwareprogrammierbaren Prozessorkernen und eFPGA-Makros“, Diplomarbeit am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der RWTH Aachen, April 2006.

[40] Tessendorf, B.: „Performance- und Verlustleistungsanalyse einer heterogenen Nomadik-Prozessorarchitektur für verlustleistungs-kritische Applikationen der mobilen Multimedia-Kommunikation“, Diplomarbeit am Lehrstuhl für Allgemeine Elektrotechnik und Datenverarbeitungssysteme der RWTH Aachen, Dezember 2006.

[41] ARM Limited, „Application Note 93 – Benchmarking with ARMulator“, Lit.-Nr.: ARM DAI 0093A, März 2002.

[42] ARM Limited, „ARM926EJ-S Product Overview“, Lit.-Nr.: ARM DVI 0035B, 2001.

[43] ARM Limited, „ARM Developer Suite – Developer Guide“, Lit.-Nr.: ARM DUI 0056DB, 1999-2001.

[44] ARM Limited, „ARM Developer Suite – Assembler Guide“, Lit.-Nr.: ARM DUI 0068B, 2000-2001.

[45] ARM Limited, „Multi Ice User Guide", Lit.-Nr.: ARM DUI0048F, 2002.

[46] Nokia: „Definitions of Musik Features“, Bochum 2007

[47] I. Vatolkin, W. Theimer: „Introduction of Methods for Automatic Classification of Music Data“, Nokia Research Center Technical Report NRC-TR-2007 2006

[48] Vary, P.: „Digitale Sprachverarbeitung 1“, 2005

[49] CM Laboratory, National Taiwan University, DSP Group: „Fast Fourier Transform (FFT) “

[50] Nokia: „Performance measure on the embedded platform N800“, 2007

[51] http://weka.sourceforge.net/wekadoc/index.php

[52] Frauke Horstmann: „Effiziente FFT-Architekturen“, interne Kommunikation.

[53] von Livonius, J.; Blume, H.; Noll, T. G.: „Flexible Umgebung zur Pareto-Optimierung von Algorithmen - Anwendungen in der Videosignalverarbeitung“, Tagungsband der ITG Fachtagung "Elektronische Medien", Dortmund, 20.-21.3.2007, pp. 157-162