

Neuronale Netze

(Beleg)

Dokumentation

Antje Niederlein

(Matrikel-Nr.: 11782)

Inhalt

1	Aufgabenstellung.....	1
2	Theoretische Grundlagen.....	1
2.1	Was ist Sound?.....	1
2.2	Tonhöhe und Lautstärke.....	1
2.3	Digitalisierung von Sound.....	3
2.4	Das Dateiformat WAV.....	4
2.5	Frequenzspektrum.....	4
3	Realisierung.....	7
3.1	Vorüberlegungen.....	7
3.2	Die Aufnahme.....	7
3.3	Analyse der Soundkurve.....	7
3.4	Vorverarbeitung der Daten.....	10
3.5	Das Neuronale Netz.....	12
3.6	Interpretation der Ausgabe.....	13
3.7	Darstellung der Ergebnisse.....	13
3.8	Übersicht über alle Hilfsklassen.....	14
3.8.1	BbArray.....	14
3.8.2	CMatrix.....	14
3.8.3	clist und myelem.....	15
3.8.4	CWave.....	16
3.8.5	CPlaySound.....	17
3.8.6	CCaptureSound.....	17
3.8.7	Complex.....	18
3.8.8	Fft.....	18
3.9	Übersicht aller implementierten Klassen.....	19
3.9.1	CSoll.....	19
3.9.2	CListEntry.....	20
3.9.3	CNeuron.....	21
3.9.4	CBProp.....	22
4	Die Software.....	23
4.1	Der initiale Zustand.....	23
4.2	Das Hauptmenü.....	24
4.2.1	Datei – Importieren.....	24
4.2.2	Datei – Neue Aufnahme.....	24
4.2.3	Datei – Entfernen.....	25
4.2.4	Datei – Programm beenden.....	25
4.2.5	Dataset – Neu.....	25
4.2.6	Dataset – Laden.....	25
4.2.7	Dataset – Speichern.....	26
4.2.8	Dataset – Speichern unter.....	26
4.2.9	Netz – Neues Netz.....	26
4.2.10	Netz – Laden.....	27
4.2.11	Netz – Speichern.....	27
4.2.12	Netz – Speichern unter.....	27
4.2.13	Netz – Konfigurieren.....	27

4.3	Der Dateibaum.....	28
4.4	Die Anzeige der Daten.....	28
4.4.1	Tabfenster – Backpropagation-Netz.....	29
4.4.2	Tabfenster – Sollwerte.....	31
4.4.3	Tabfenster – Frequenzanalyse.....	32
5	Anleitung zum Lernen.....	33
6	Eigene Erfahrungen und Tests.....	34
7	Erweiterungen / Verbesserungen.....	35
8	Bekannte Fehler.....	35
9	Systemanforderungen / Entwicklungsumgebung.....	35
10	Quellen.....	36

Grafiken

(Beleg)	1	
Grafik 2.1:	Klang – zusammengesetzte Kurve.....	2
Grafik 2.2:	Soundkurven mit verschiedenen Abtastraten (oben 8000 Hz/ unten 48000 Hz).....	3
Grafik 2.3:	Waveform-Audio RIFF-Dateistruktur (Quelle: Vorlesungsskript Multimedia-Technologien Prof. Bruns)	4
Grafik 2.4:	Soundkurve – stark horizontal gestaucht.....	5
Grafik 2.5:	Das Prinzip der Fast Fourier Transformation.....	6
Grafik 3.1:	Entwurfsschema.....	7
Grafik 3.2:	Samples eines tiefen Tones / FFT dieser Samples.....	8
Grafik 3.3:	Samples eines hohen Tones / FFT dieser Samples.....	8
Grafik 3.4:	Zwei hohe Töne wurden analysiert.....	9
Grafik 3.5:	Zwei tiefe Töne wurden analysiert.....	9
Grafik 3.6:	Verfahren zur Ermittlung eines charakteristischen Spektrums.....	11
Grafik 3.7:	Mittelwert aus sieben Frequenzspektren.....	11
Grafik 3.8:	genormte Intensitäten.....	12
Grafik 3.9:	quadrierte Werte.....	12
Grafik 3.10:	Ausgabe-Werte (Ton 16 würde als "wahr" gewertet).....	13
Grafik 4.1:	Die Software.....	23
Grafik 4.2:	Neue Aufnahme.....	24
Grafik 4.3:	Neues Netz.....	26
Grafik 4.4:	Konfigurieren.....	27
Grafik 4.5:	Dateibaum.....	28
Grafik 4.6:	Informationen zur Datei + Möglichkeiten zum Abspielen und Aufnehmen.....	29
Grafik 4.7:	Testergebnisse.....	29
Grafik 4.8:	Ausgabe der Noten.....	30
Grafik 4.9:	Informationen zum Neuronalen Netz.....	30
Grafik 4.10:	Analyse zur Bestimmung der Sollwerte.....	32
Grafik 4.11:	Anzeige des Frequenzspektrums vertikal und horizontal.....	32
Grafik 6.1:	Lernergebnis 1.....	34

1 Aufgabenstellung

Ziel des Projektes war die Entwicklung einer Software, welche Notenerkennung umsetzt. Als Grundlage dafür sollte ein neuronales Netz dienen, welches trainiert und angewendet werden kann. Als Eingabe waren die Daten von unkomprimierten Sounddateien (also im WAV-Format) vorgesehen. Die Eingabe sollte über ein Keyboard oder Klavier mit 61 Tasten (entspricht 5 Oktaven mit je 12 Tönen + 1 Ton) erfolgen. Die Ausgabe sollte in Form eines Notenblattes erfolgen.

2 Theoretische Grundlagen

2.1 Was ist Sound?

Sound – physikalisch Schall genannt – begleitet uns in unserem Leben ständig. Für Musik, Stimmen und Geräusche (und sei es nur das Summen des Rechners) sind Schallwellen verantwortlich. In der Akustik werden alle mechanischen Schwingungen eines elastischen Mediums als Schall definiert. In solch einem Medium, wie beispielsweise der Luft, breitet sich Schall in Form von Längswellen aus. Als Schallquelle kann zum Beispiel die Saite einer Gitarre angenommen werden, die ihre Schwingungen auf das umgebende Medium überträgt. Die Teilchen des Mediums beginnen dadurch um ihren Ruhepunkt herum zu schwingen. Dabei stoßen sie benachbarte Teilchen an und geben ihre Energie weiter. Dieser Prozess erzeugt so periodische Druck- und Dichteänderungen.

Gelangen Schallwellen ins Ohr, so verstärken sie sich in den Windungen der Ohrmuschel. Die verstärkten Wellen treffen schließlich auf das Trommelfell am Ende des äußeren Gehörgangs und versetzen dessen Membran in Schwingung mit der entsprechenden Intensität und Frequenz. Im Innenohr werden die mechanischen Reize dann in elektrische Energie umgewandelt und über Nervenbahnen zum Gehirn weitergeleitet, welches die Informationen dann verarbeitet.

2.2 Tonhöhe und Lautstärke

Was macht aber nun die Tonhöhe und dessen Lautstärke aus?

Die Anzahl der Druckschwankungen der Schallwelle in einem bestimmten Zeitraum charakterisiert die Tonhöhe. Der Fachbegriff dafür lautet Frequenz. Die Frequenz wird in Hertz (Hz) gemessen, wobei die Schwingungen pro Sekunde angegeben werden. Um so größer die Frequenz, um so höher nimmt der Mensch den Ton wahr.

Das menschliche Gehör (eines jungen Menschen) nimmt Frequenzen von 16 bis 20.000 Hz wahr. Frequenzen bis 16 Hz bezeichnet man als Infraschall und wird nur als Vibrieren empfunden. Frequenzen über 20.000 Hz werden als Ultraschall bezeichnet und ist für den Menschen nicht wahrnehmbar.

Die musikalische Skala des Menschen ist logarithmisch aufgebaut, d.h. eine Verdopplung der Frequenz entspricht einer Oktave. Diese Tatsache ist an späterer Stelle noch wichtig, da es eine Erklärung liefert, warum höhere Töne besser erkannt werden können.

Folgende Frequenzen werden als "c" wahrgenommen:

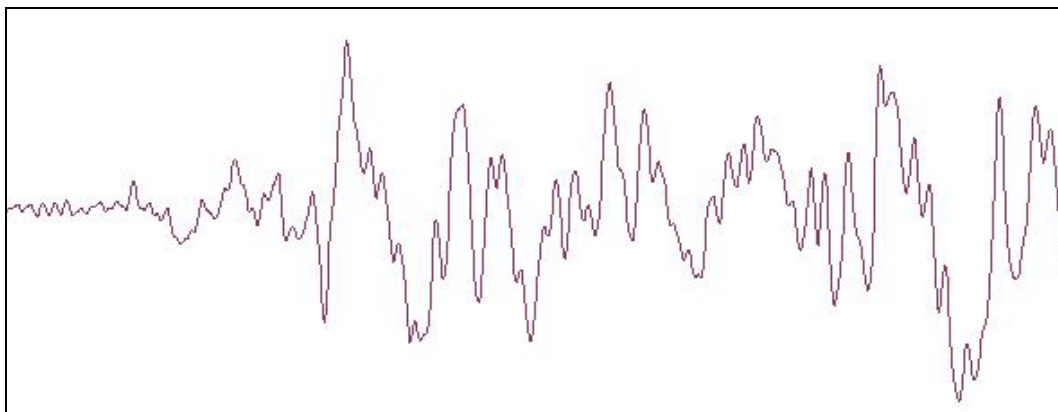
- 66 Hz
- 132 Hz
- 264 Hz
- 528 Hz
- 1056 Hz
- 2112 Hz

Die Aufzählung könnte man beliebig weiterführen. Jedoch sind Frequenzen über 4200 Hz in der Musik nur noch als Klangfarbe wahrzunehmen.

Die Lautstärke eines Tones ist nun von den Druckschwankungen abhängig, die die Schallwelle erzeugt. Der Schalldruckpegel wird in Dezibel (dB) angegeben. Dies ist keine Maßeinheit, sondern lediglich die Angabe des Drucks (gemessen in Pascal) in logarithmischem Maßstab. Die Schmerzgrenze liegt beim Menschen bei 130 dB. Im folgenden wird statt Lautstärke, der Begriff Intensität verwendet.

Die wahrgenommene Lautstärke hängt aber auch stark vom menschlichen Ohr ab. Dieses ist für den Schalldruck im Frequenzbereich von 2000 bis 5000 Hz weitaus empfindlicher als für tiefere oder höhere Frequenzen. Das bewirkt z.B. ein Absinken der Schmerzgrenze auf etwa 115 dB in diesem Bereich.

Mathematisch lassen sich reine Töne als Sinusschwingungen darstellen. In der Natur werden jedoch nie reine Töne erzeugt, sondern nur Klänge. Ein Klang besteht aus Tönen unterschiedlicher Frequenzen und Amplituden, bleiben aber dennoch periodische Schwingungen. Der niedrigste Ton bestimmt hierbei die Klanghöhe, die höheren Töne (Obertöne) die Klangfarbe.



Grafik 2.1: Klang – zusammengesetzte Kurve

2.3 Digitalisierung von Sound

Um einen Sound digital abzuspeichern muss die komplexe analoge Soundkurve nachgebildet werden. Das analoge Signal muss also abgetastet werden und die abgetasteten Werte (im folgenden als Samples bezeichnet) müssen die ursprüngliche Kurve genügend genau rekonstruieren können. Hierbei spielen drei Parameter eine wichtige Rolle für die Qualität des aufgenommenen Sounds:

Bits pro Sample:

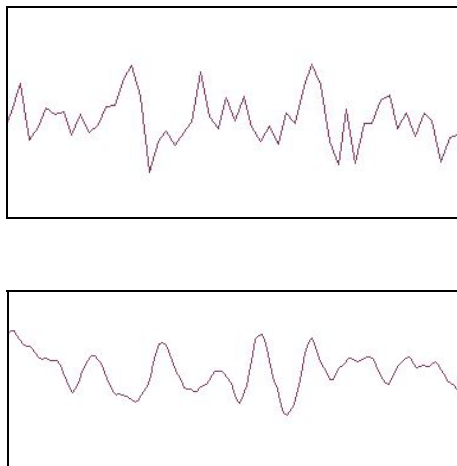
Die Amplitudenwerte werden mit einer festzulegenden Genauigkeit gespeichert. 8 Bit reichen erfahrungsgemäß für eine gute Qualität nicht aus, daher sind wenigstens 16 Bit pro Sample empfohlen (CD-Qualität).

Samples pro Sekunde:

Die Abtastrate wirkt sich ebenfalls auf die Qualität des Sounds aus. Wird das Signal in zu großen Zeitabständen abgetastet, so gehen wichtige Informationen über Schwingungen eventuell verloren. Die Samples pro Sekunde geben als an, wie oft innerhalb einer Sekunde das Signal abgetastet wird. Soll CD-Qualität des Sounds erreicht werden, so sollte mit 44100 Hz (Samples pro Sekunde) abgetastet werden. Aber auch 22050 Hz geben ein recht gutes Ergebnis (Radio-Qualität).

Anzahl der Kanäle:

Abgesehen von komfortablen Surround-Sound-Techniken, die heutzutage das Klangerlebnis schlechthin bilden, wird im Allgemeinen mit einem (mono) oder zwei (stereo) Kanälen gearbeitet. Dies wirkt sich in erster Linie auf die Menge der zu verarbeitenden Daten aus.

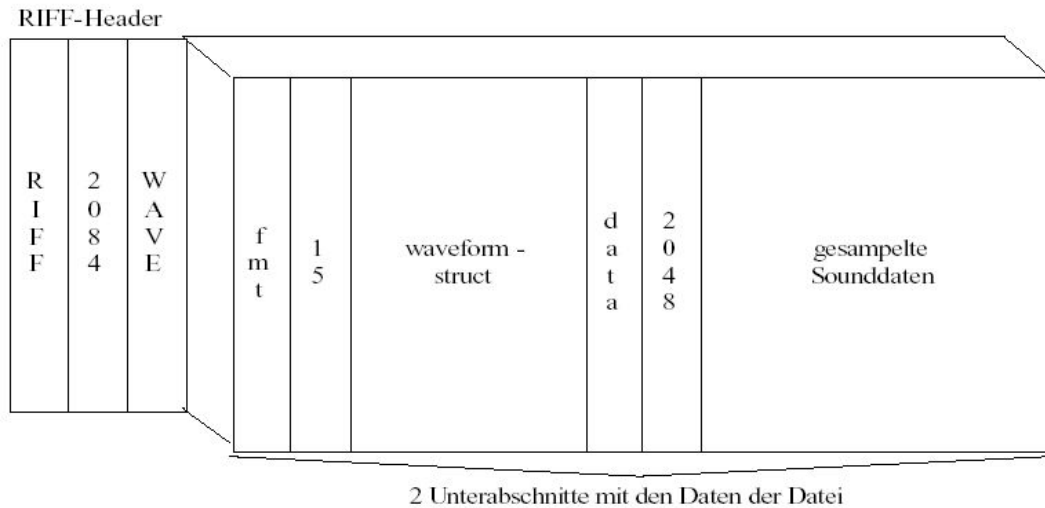


Grafik 2.2: Soundkurven mit verschiedenen Abtastraten (oben 8000 Hz/ unten 48000 Hz)

2.4 Das Dateiformat WAV

Es gibt verschiedene Dateiformat-Standards zum Abspeichern von Sound. Hier soll nur das WAV-Format betrachtet werden, da das Projekt ausschließlich dieses Format verarbeiten kann.

Eine Wav-Datei ist nach dem RIFF-Format (Resource Interchange File Format) aufgebaut.

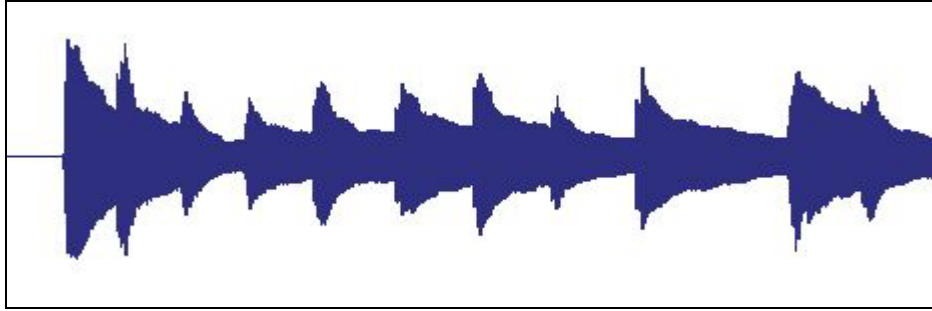


Grafik 2.3: Waveform-Audio RIFF-Dateistruktur (Quelle: Vorlesungsskript Multimedia-Technologien Prof. Bruns)

Eine Wav-Datei besteht also aus einem Informationsteil (gekennzeichnet mit "fmt"), welcher Informationen über die Abtastrate, die Anzahl der Kanäle und die Bits pro Sample enthält, und einem Datenteil (gekennzeichnet mit "data"), welcher die Samples in unkomprimierter Form enthält.

2.5 Frequenzspektrum

Der Verlauf der Soundkurve gibt ohne Transformation der Daten nicht genügend Informationen über Tonhöhen und deren Lautstärken her. In der folgenden Grafik – eine stark horizontal gestauchte Soundkurve – kann man zwar noch einzelne Töne ausmachen, jedoch keine Aussage über deren Höhe treffen.



Grafik 2.4: Soundkurve – stark horizontal gestaucht

Da sich Sound aus verschiedenen Frequenzen mit verschiedenen Intensitäten zusammensetzt und eine einzelne Frequenz durch eine Sinusschwingung beschrieben werden kann, ergibt sich die Soundkurve also als Summe unterschiedlicher Sinusschwingungen.

Also ist es naheliegend nach einer Art Rücktransformation zu suchen, welche aus einer vorliegenden Soundkurve die mathematische Beschreibung als Aufsummierung verschiedener Sinuskurven berechnet. Eine solche Berechnung kann mit der **Fast Fourier Transformation (FFT)** realisiert werden.

Da Musik im Zeitverlauf verschiedene Klänge und Lautstärken aufweist, verändern sich die Sinuskomponenten aus denen sich die Soundkurve zusammensetzt permanent. Um die charakteristischen Frequenzen an einer bestimmten Stelle herauszufiltern, wird von dieser Position aus nur ein kleiner Teil der Soundkurve betrachtet. Die FFT soll schließlich die Sinuskomponenten so berechnen, dass dieser Kurventeil rekonstruiert werden könnte. Die FFT wird also immer blockweise ausgeführt.

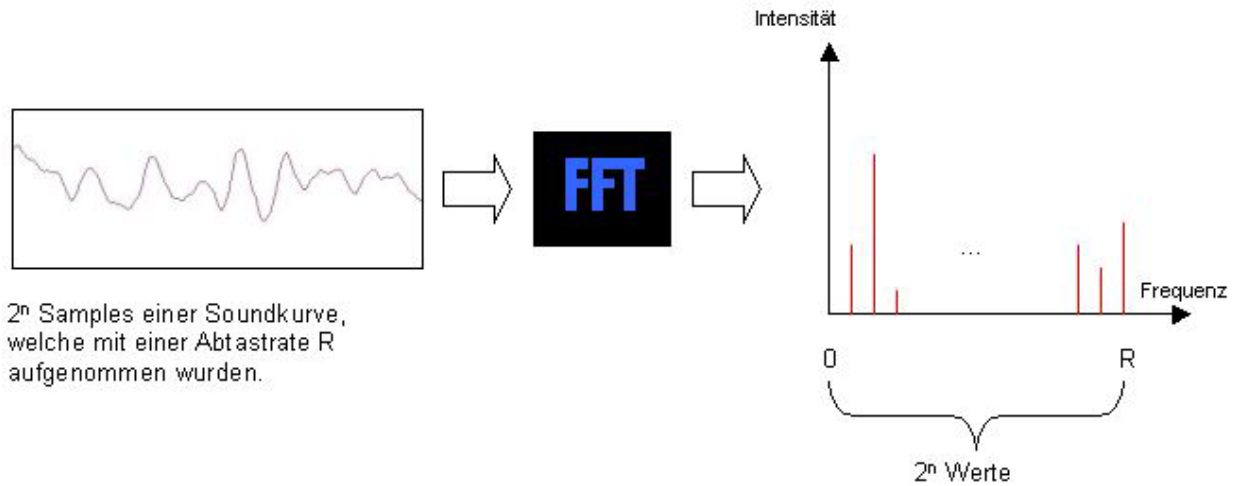
Auf die genaue Berechnung wird in dieser Dokumentation nicht eingegangen, da sie nicht realisiert werden musste. Die FFT wird daher als Blackbox angenommen. Lediglich ein paar Eigenschaften der Fast Fourier Transformation werden im folgenden erklärt.

Die höchste Frequenz, deren Anteil die FFT berechnen kann entspricht der Abtastrate des Signals. Wird also die Soundkurve mit 22050 Hz abgetastet, so kann die FFT die Sinuskomponenten bis zur Frequenz 22050 bestimmen. Die Genauigkeit der Berechnung bestimmt die Größe des Blocks (z.B. 2048 Samples). Die Größe eines Blocks muss aus Berechnungsgründen immer einer Zweierpotenz entsprechen. Die Anzahl der Sinuskomponenten die berechnet werden entspricht der Blockgröße. Wird zum Beispiel ein Block mit 512 Samples und einer Abtastrate von 44100 der FFT übergeben, so werden 512 Frequenzen im Intervall von Null bis 44100 berechnet, was einem Abstand von ca. 86 Hz entspricht. Dieser Abstand ist recht groß, wenn man bedenkt, dass sich die Frequenzen tiefer Töne (im musikalischen Sinne) nur um wenige Hertz unterscheiden.

Die Wahl eines größeren Blocks verdichtet zwar die Berechnung (bei einer Blockgröße von 2048 Samples würden in dem selben Intervall 2048 Werte berechnet, welche einen Abstand von rund 21 Hertz aufweisen), jedoch würde damit auch ein größerer Zeitraum einbezogen werden (512 Samples entsprechen bei einer Abtastrate von 44100 Samples pro Sekunde rund 11 Millisekunden, 2048 Samples entsprechen schon rund 46 Millisekunden).

Die Wahl einer kleineren Abtastrate würde das Intervall auch verdichten (dann würden zum Beispiel 512 Berechnungen auf dem Intervall von Null bis 22050 erfolgen, was einem Abstand von 43 Hz entspräche), allerdings würde die Qualität der Sounddatei darunter leiden.

Die folgende Grafik, soll das Prinzip der FFT noch mal veranschaulichen.

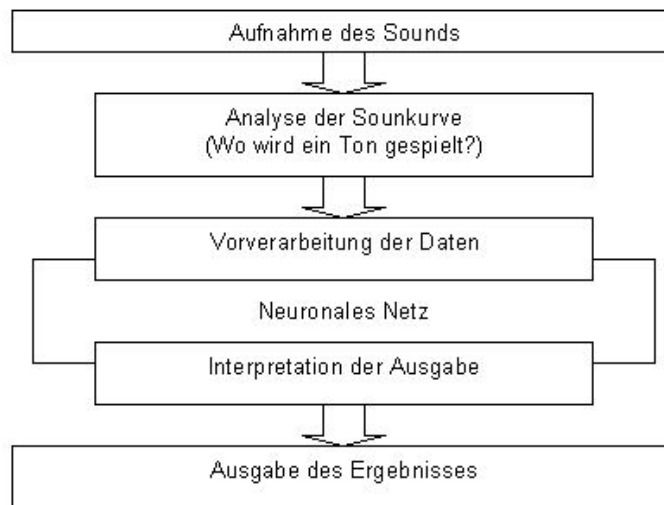


Grafik 2.5: Das Prinzip der Fast Fourier Transformation

3 Realisierung

3.1 Vorüberlegungen

Die Implementierung sollte grob folgendem Schema entsprechen:



Grafik 3.1: Entwurfsschema

Zu jedem Punkt gibt es verschiedene Problemstellungen, die in den folgenden Kapiteln erörtert werden.

3.2 Die Aufnahme

Im folgenden kann davon ausgegangen werden, dass Methoden und Klassen zur Verfügung stehen, die das Auslesen von Sounddaten, so wie Abspielen und Aufnehmen realisieren, bereits vorhanden sind und nicht mehr implementiert werden müssen

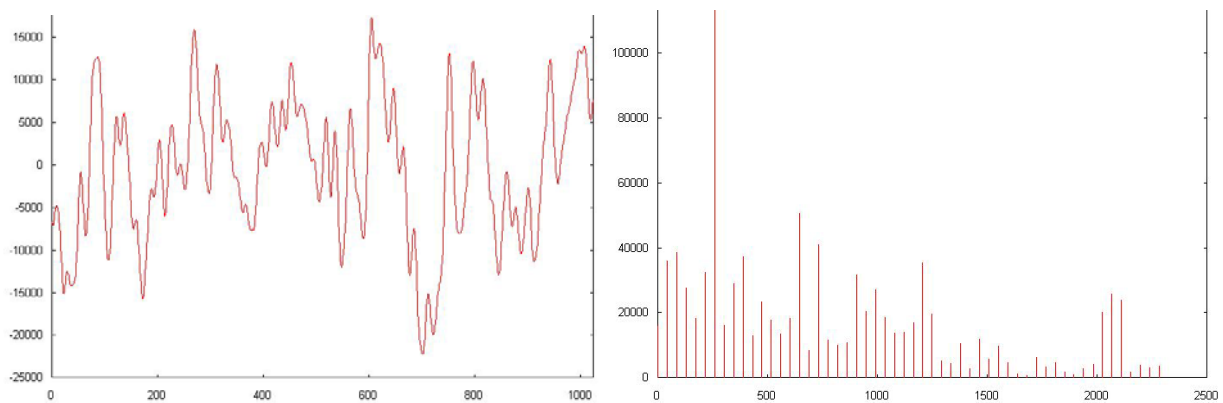
3.3 Analyse der Soundkurve

Um an aussagekräftige Daten für das neuronale Netz zu kommen, müssen zu erst die angespielten Töne gefunden werden. Da sie nicht ohne weiteres aus der Soundkurve zu bestimmen sind, musste hier bereits die FFT eingebracht werden. Die Grundidee hier ist ein Durchlauf der Soundkurve in geeignet kleinen Schritten. An jeder Position wird das Frequenzspektrum für eine geeignete Blockgröße berechnet und mit

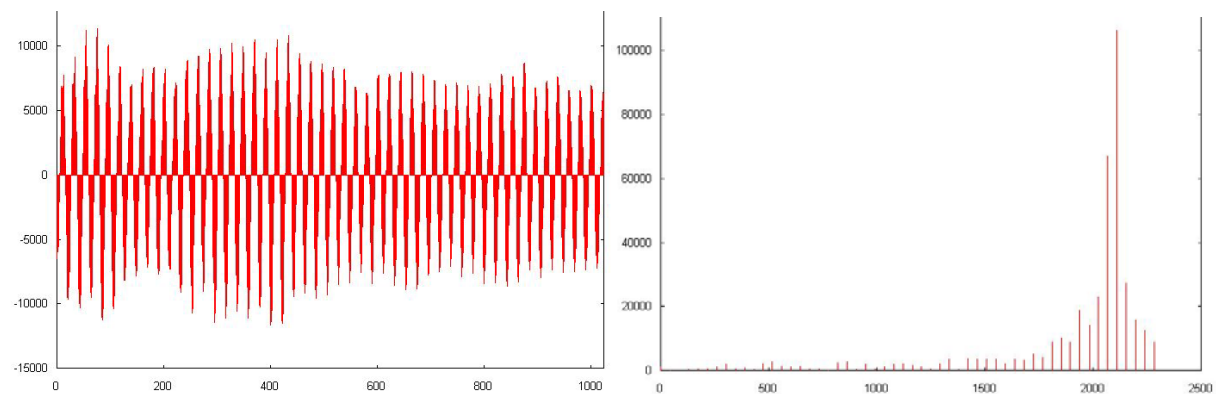
dem vorherigen verglichen. Aus einem starken Anstieg der Intensität einer Frequenz kann auf einen Ton geschlossen werden.

Der Algorithmus, der die Analyse letztlich umsetzen soll, ist um einiges komplizierter.

Grund dafür sind die unterschiedlichen Verhalten von tiefen und hohen Tönen. Während hohe Töne ein sehr klares, eindeutiges und konstantes Frequenzspektrum aufweisen, streuen tiefe Töne ziemlich stark und schwankend auf Nachbarfrequenzen. Das heißt unmittelbar nach dem der tiefe Ton angespielt wurde (und die Intensität sich damit erhöht hat) kann bei der nächsten Berechnung der Wert schon wieder drastisch abfallen, nur um genauso schnell wieder anzusteigen. Die Ursachen hierfür sind mir unbekannt.



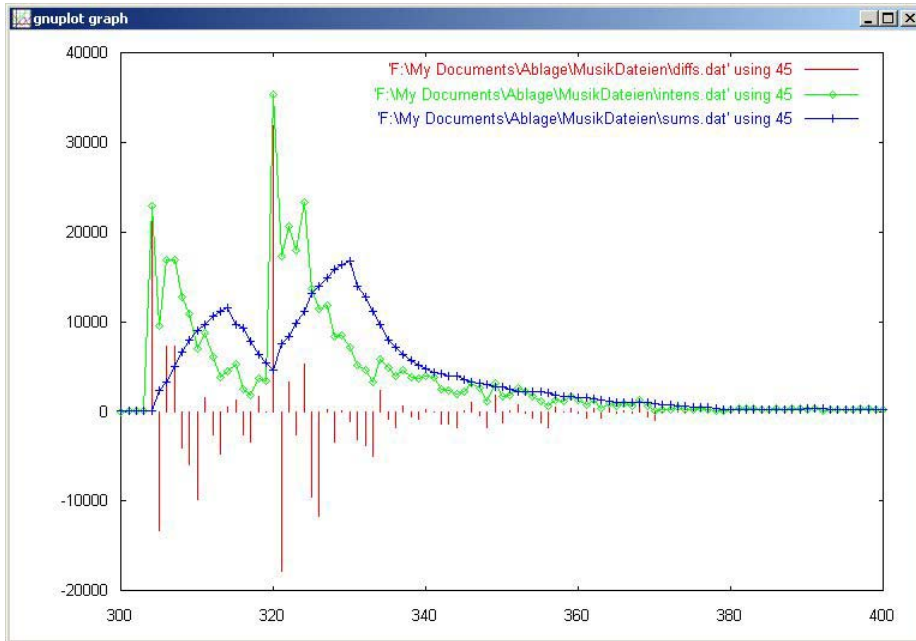
Grafik 3.2: Samples eines tiefen Tones / FFT dieser Samples



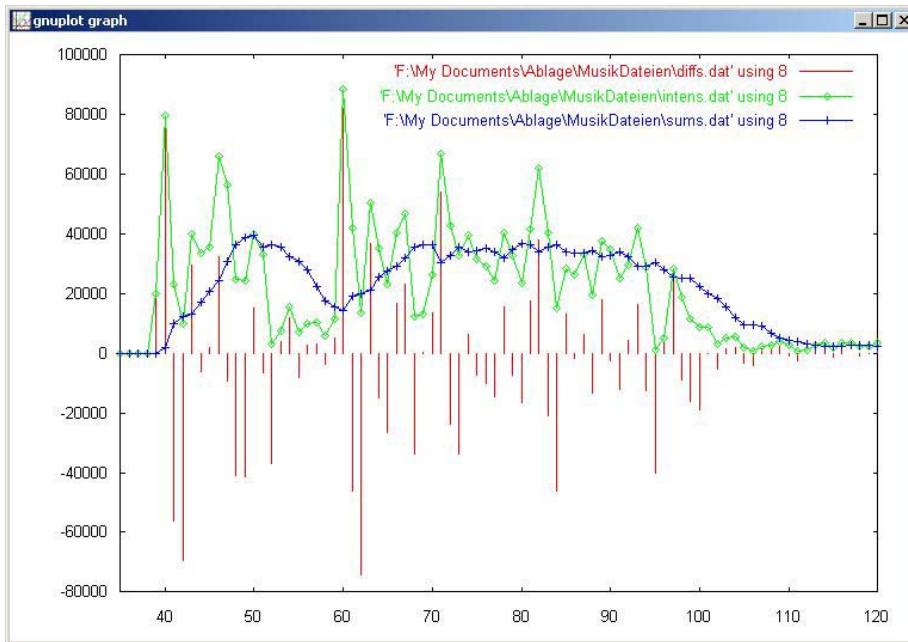
Grafik 3.3: Samples eines hohen Tones / FFT dieser Samples

Dieses Verhalten bereitete Probleme. Da die Frequenzspektren bei tiefen Tönen so sehr schwanken, kann es vorkommen, dass der Ton mehrfach hintereinander erkannt wird. Eine mögliche Lösung dafür lag in der Wichtung mehrerer vorangegangener Spektren. Anstatt immer nur das aktuell berechnete Spektrum mit dem zuletzt Ermittelten zu vergleichen, wird es mit dem Mittelwert von mehreren vorher berechneten Spektren verglichen.

Folgende Grafiken zeigen 3 Kurven. Die grüne Kurve zeigt den Verlauf der Intensität einer bestimmten Frequenz über den gesamten Zeitraum. Die roten Werte geben die Differenzen von Berechnung zu Berechnung an. Die blau Kurve schließlich ergab sich aus der Kombination und Bewertung der anderen Werte. Man erkennt, dass die blaue Kurve sehr viel eindeutiger anzeigt, dass genau zwei Töne gespielt wurden.



Grafik 3.4: Zwei hohe Töne wurden analysiert



Grafik 3.5: Zwei tiefe Töne wurden analysiert

Die optimale Schrittgröße kann nur durch Tests herausgefunden werden. Mit 50 Millisekunden wurden recht gute Ergebnisse erzielt. Die Blockgröße ist abhängig davon, wie viele Input-Neuronen das Netz hat, denn die FFT liefert ja eine feste Anzahl von Werten zurück.

Die Weiterverarbeitung nach der Analyse erfolgt mit den Werten der FFT bis zur Frequenz 2300 Hz. Das Vernachlässigen der Anteile der höheren Frequenzen macht Sinn, da die höchste Grundfrequenz eines Keyboards mit 61 Tasten bei ca. 2100 Hz liegt. Dadurch sind alle höheren Frequenzen für das neuronale Netz als Information eher störend und würden außerdem den Rechenaufwand erheblich erhöhen.

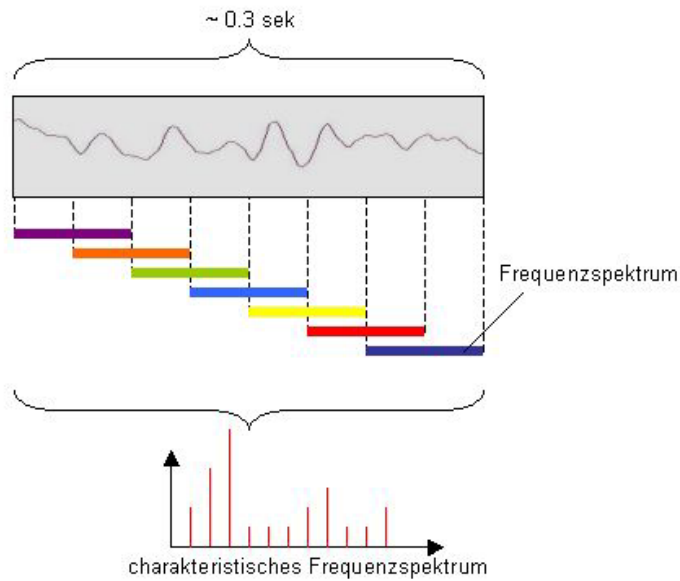
Nachteile:

Obwohl die Tonerkennung recht gut funktioniert, passiert es doch noch hin und wieder, dass tiefe Töne mehrfach erkannt werden. Außerdem können sehr schnelle Tonfolgen eventuell nicht aufgelöst werden, da nach jedem gefundenen Ton eine halbe Sekunde übersprungen wird (eine weitere Maßnahme um Mehrfacherkennung zu unterbinden). Der Algorithmus bietet also noch genügend Verbesserungsmöglichkeiten.

3.4 Vorverarbeitung der Daten

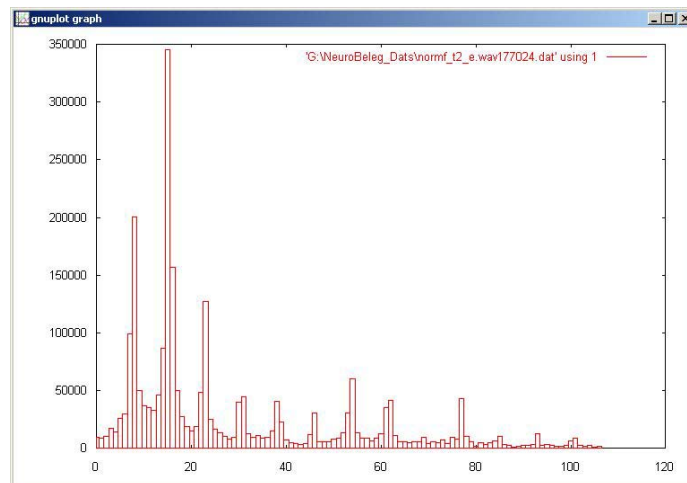
Das Ergebnis der Analyse ist vorerst nur ein Zeitpunkt, zu dem angenommen wird, dass ein Ton gespielt wurde. Da sich Töne durch ein charakteristisches Frequenzspektrum auszeichnen, soll dieses als Eingabe für das neuronale Netz dienen. Ein Problem dabei ist die logarithmische Verteilung der Töne auf dem Frequenzspektrum. D.h. während hohe Töne auf dem Frequenzband sehr viel "Platz" haben und dadurch womöglich eindeutiger sind, sind tiefe Töne nur durch einen sehr geringen Abstand voneinander getrennt. Die Frequenzspektren tiefer Töne ähneln sich dadurch sehr. Hinzu kommt das unstete Verhalten der tiefen Töne, so dass es letztendlich sehr schwierig ist charakteristische Daten herauszufiltern.

Die Lösung kann in einer Mittelwertbildung mehrerer aufeinanderfolgender Spektren liegen. Hierbei würden die Frequenzen, die den Ton charakterisieren, hervorgehoben, während der Einfluss der Nachbarfrequenzen abnehmen würde.

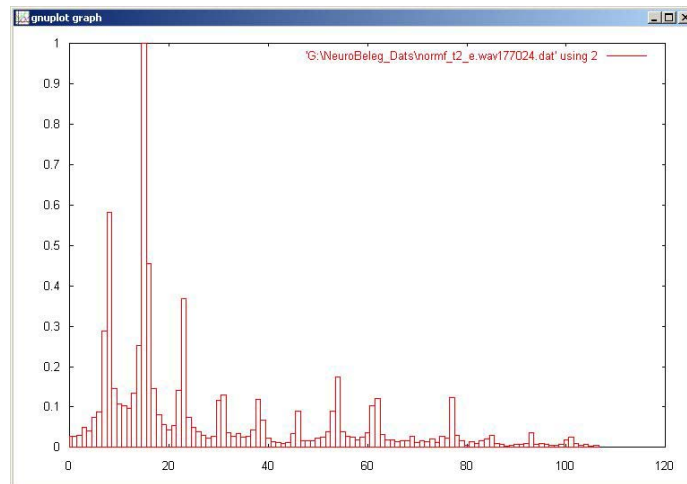


Grafik 3.6: Verfahren zur Ermittlung eines charakteristischen Spektrums

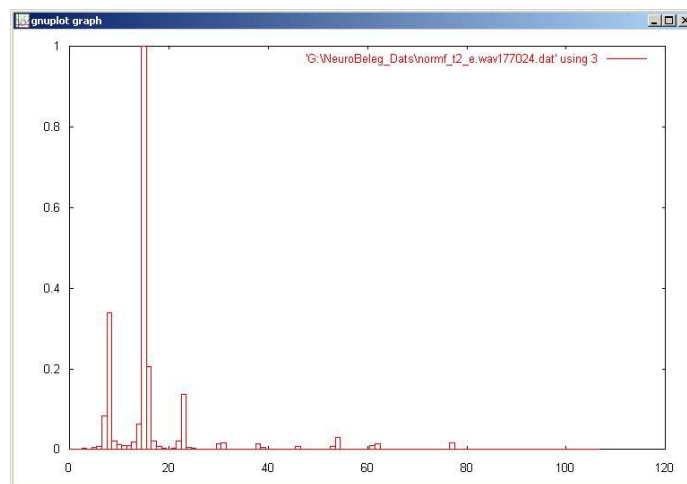
Dieses gemittelte Spektrum hat nun noch den Nachteil, dass die Daten in keiner genormten Form vorliegen. Die Intensitäten müssen also noch auf einem geeigneten Wertebereich abgebildet werden. Ein günstiger Wertebereich liegt im Intervall von Null bis Eins. Dabei sollte die höchste Intensität des vorliegenden Spektrums den Wert 1.0 bekommen um Lautstärkenunterschiede auszuschließen. Außerdem können die wichtigen Werte noch besser hervorgehoben werden, indem man alle Werte quadriert. So treten hohe Intensitäten deutlich hervor und niedrige Intensitäten werden verschwindend klein.



Grafik 3.7: Mittelwert aus sieben Frequenzspektr



Grafik 3.8: genormte Intensitäten



Grafik 3.9: quadrierte Werte

Zusammenfassend werden folgende Schritte bei der Vorverarbeitung ausgeführt:

- Berechnung der Frequenzspektren innerhalb von rund 0,3 Sekunden (überlappend)
- Bildung eines Mittelwertspektrums
- Normierung der Intensitäten auf einen geeigneten Wertebereich (z.B. [0,1])
- Quadrierung der Werte

3.5 Das Neuronale Netz

Die aus der Vorverarbeitung der Daten hervorgegangenen Daten werden nun als Inputwerte an ein neuronales Netz gegeben.

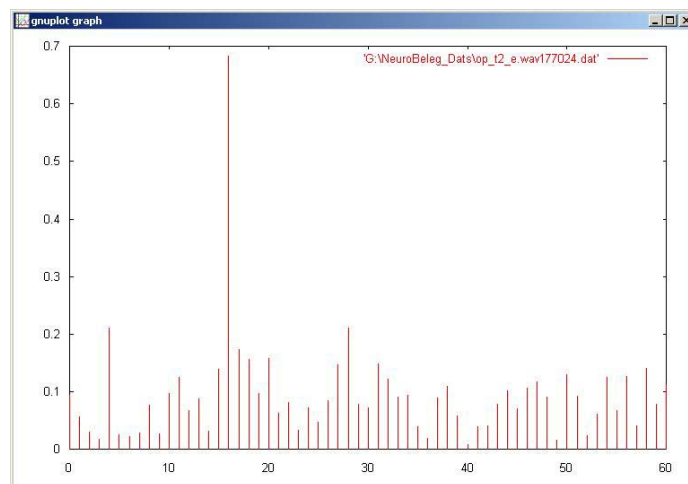
Als Netz soll ein klassisches Backpropagation-Netz verwandt werden. Das Netz besteht aus einer Inputschicht, die Anzahl der Input-Neuronen bestimmt sich aus der verwendeten Blockgröße der FFT.

Werden 2048 Samples an die FFT übergeben, so erhält man 2048 Werte des Spektrums. Davon sind jedoch nur 107 Werte, die die Intensitäten bis 2300 Hz enthalten. Also muss dieses Netz 107 Input-Neuronen besitzen. Die Anzahl der Hiddenschichten und die Anzahl der Hidden-Neuronen pro Schicht ist frei wählbar. Das Netz besitzt 61 Output-Neuronen, von denen jedes für eine Taste des Keyboards steht.

Lernrate, Momentum-Konstante und Aktivierungsfunktion sollen vom Nutzer festgelegt werden. Das Lernen erfolgt online, d.h. nach jedem Lernschritt werden die Gewichte aktualisiert. Die Präsentation der Muster erfolgt für jeden Lernzyklus in der gleichen Reihenfolge.

3.6 Interpretation der Ausgabe

Die 61 Ausgabe-Neuronen geben Auskunft über den gespielten Ton, indem sie als Information "wahr" oder "falsch" beinhalten. D.h. das Netz wird dahingehen trainiert, dass es dem Output-Neuron des entsprechenden Tones den Wert 0.9 für "wahr" und 0.1 für "falsch" zuordnet. So können theoretisch auch Akkorde erkannt werden, denn dann brauchen nur mehrere Ausgabe-Neuronen den Wert für "wahr" enthalten. Da das Netz nie so perfekt wird, dass es genau diese Werte ausgibt, reicht es oft auch aus, alle Ausgabewerte unter 0.5 als "falsch" und alle Werte über 0.5 als "wahr" zu interpretieren.



Grafik 3.10: Ausgabe-Werte (Ton 16 würde als "wahr" gewertet)

3.7 Darstellung der Ergebnisse

Die Ergebnisse liegen nun in Form von mehreren Vektoren vor, mit je 61 Elementen, wobei jedes Element die Information "wahr" oder "falsch" enthält. Diese Vektoren können nun grafisch als Noten auf den entsprechenden Notenlinien dargestellt werden.

3.8 Übersicht über alle Hilfsklassen

Zur Realisierung des Projekts waren einige Hilfsklassen notwendig, die bereits unabhängig von dem Projekt implementiert wurden. Im folgenden sollte diese kurz vorgestellt werden. Die einzelnen Methoden und Membervariablen werden hier nicht erklärt.

3.8.1 BbArray

Quelle: Sony Electronics Distributed Research Lab

Beschreibung:

Die Klasse realisiert komfortable Arrays, deren Typ über Templates festgelegt wird.

```

template <class T>
class BbArray

BbArray(unsigned long initialSize = 0)
BbArray(const BbArray<T>& t)
~BbArray()

unsigned long getNum()
void setNum(unsigned long n)
void append(const T& v)
void remove(unsigned long i, unsigned long n=1)
void insert(unsigned long i, const T& v)
void insert(unsigned long i, const T *v, unsigned long n)
void clear()
void setValue(unsigned long i, const T& v)

BbArray<T>& operator =(const BbArray<T>& t)
T& operator [] (unsigned long i)
const T& operator [] (unsigned long i) const

void openSpace(unsigned long i, unsigned long n = 0)
void closeSpace(unsigned long i, unsigned long n = 0)
T& getValue(unsigned long i)
unsigned long getMaxSize()

T *myArray
unsigned long mySize, myMaxSize

```

3.8.2 CMatrix

Quelle: selbst geschrieben

Beschreibung:

Diese Klasse realisiert Matrizen und einfache Matrix-Operationen.

```

class CMatrix

CMatrix()
CMatrix(int r, int c)
CMatrix(CMatrix &m)
~CMatrix(void)

int GetCols()
int GetRows()
int GetLength()
void Dim(int r, int c)
void set(int r,int c,double val)
double get(int r,int c)

CMatrix operator *(CMatrix &m)
CMatrix& operator =(CMatrix &m)

CString Display()

void SetArray(BbArray<double> *array)
BbArray<double> *GetArray()

BbArray<double> matrix
int cols
int rows

```

3.8.3 clist und myelem

Quelle: selbst geschrieben

Beschreibung:

Die Klasse myelem repräsentiert ein Listenelement, dessen Typ per Template festgelegt wird. Die Klasse clist realisiert eine doppelt verkettete Ringliste dieser Elemente.

```

template <class type>
class myelem

myelem()
myelem(myelem<type> *pNext,type* pItemIns)
~myelem()

myelem<type>* pNxt
myelem<type>* pPrev
type* pItem

```

```

template <class type>
class clist:public myelem<type>

clist()
~clist()

bool IsEmpty()

```

```

int InsertHead(type* pItemIns)
int InsertTail(type* pItemIns)
int InsertBefore(type* pItemIns)
int InsertBehind(type* pItemIns)
int RemoveItem()
type* GetSelected()
type* GetFirst()
type* GetLast()
type* GetNext()
type* GetPrev()
type* GetIdx(int index)
int GetNumb()
int FindItem(type* pItemfind)
int AddItemToList(type* pItem, int(*fcmp)(type* pItList,type* pItNew))

myelem<type>* pCurrent
    
```

3.8.4 CWave

Quelle: Skripte aus der Lehrveranstaltung Multimedia-Technologien

Beschreibung:

Realisiert die Schnittstelle zu Wav-Dateien (unter Verwendung von DirectX 8.1)

```

class CWave

CWave()
virtual ~CWave()

bool Open(char* fname, CString *msg)
bool Create(char* fname, int Channels, int BitsPerSample, int SamplesPerSec)
bool Truncate(char *fname)
bool Close()

DWORD Read(DWORD offset, void *lpvPtr1, DWORD dwBytes1, void *lpvPtr2, DWORD
dwBytes2)
DWORD Write(DWORD offset, void *lpvPtr1, DWORD dwBytes1, void *lpvPtr2, DWORD
dwBytes2)

void Draw(CDC *dc, CRect rect, DWORD offset, char channel)
bool GetFormat(int &Channels, int &BitsPerSample, int &SamplesPerSec)
void GetTrackLength(int time[3])
void CalcTime(DWORD offset, int time[3])
void GetAbsBytes(DWORD &abs)
int GetSamplesOfPos(int num, BbArray<double> *lch, BbArray<double> *rch, DWORD
myoffset)

HMMIO        hmmio
MMCKINFO     ciParentChunk
MMCKINFO     ciSubChunk
LONG         nFmtSize
LONG         nDataSize
BYTE*        pFmt
LONG         seek_offset
WAVEFORMATEX *w
    
```

3.8.5 CPlaySound

Quelle: Skripte aus der Lehrveranstaltung Multimedia-Technologien / eigene Erweiterungen

Beschreibung:

Objekte dieser Klasse ermöglichen ein Abspielen von Wav-Dateien unter Verwendung von DirectX 8.1.

```
class CPlaySound
{
public:
    CPlaySound()
    virtual ~CPlaySound()

    bool Create(LPTSTR filename, WORD bufferSize, CString *msg)
    bool LoadBuffer()
    bool Stop()
    bool Start(int *p)
    bool Pause()
    bool ChkPlay()

    void GetCurrentPlayTime(int time[3])
    void GetCurrentBytes(DWORD &bytes, DWORD &absbytes)
    DWORD GetOffset()
    bool IsPlaying()

private:
    LPDIRECTSOUND          m_pDSC
    LPDIRECTSOUNDBUFFER   m_pDSB
    PCMWAVEFORMAT          m_PCMWF
    DWORD                  m_bufferSize

    CWave m_wave

    int m_buffnr
    DWORD m_bytesRead
    DWORD m_fileOffset
    bool playing
    bool pause
    bool stopflag
    DWORD prevpos
    bool thread
};
```

3.8.6 CCaptureSound

Quelle: Skripte aus der Lehrveranstaltung Multimedia-Technologien / eigene Erweiterungen

Beschreibung:

Objekte dieser Klasse werden zur Aufnahme von Wav-Dateien unter Verwendung von DirectX 8.1 genutzt.

```
class CCaptureSound
{
public:
    CCaptureSound(void)
    ~CCaptureSound(void)
};
```

```

bool Create(LPTSTR filename, WORD bufferSize,
           WORD channels, WORD bitsPerSample, DWORD samplesPerSec, bool de)
bool Start()
bool SaveBuffer()
bool Stop()
bool IsRecording()

```

```

LPDIRECTSOUNDCAPTURE m_pDSC
LPDIRECTSOUNDCAPTUREBUFFER m_pDSBCapture
PCMWAVEFORMAT m_PCMWF
DWORD m_bufferSize
CWave m_wave
int m_buffnr
DWORD m_bytesWritten
DWORD m_fileOffset
bool recording

```

3.8.7 Complex

Quelle: <http://www.relisoft.com/Freeware/fft.zip>

Beschreibung:

Diese Klasse realisiert komplexe Zahlen (werden für die Klasse FFT benötigt)

```

class Complex
{
public:
    Complex ()
    Complex (double re): _re(re), _im(0.0)
    Complex (double re, double im): _re(re), _im(im)

    double Re () const
    double Im () const
    double Mod () const

    void operator += (const Complex& c)
    void operator -= (const Complex& c)
    void operator *= (const Complex& c)
    Complex operator- ()

private:
    double _re
    double _im
}

```

3.8.8 Fft

Quelle: <http://www.relisoft.com/Freeware/fft.zip>

Beschreibung:

Diese Klasse realisiert die Fast Fourier Transformation (Bezeichnung vom Autor: "decimation-in-time in-place FFT algorithm")

```

class Fft

Fft (int Points, long sampleRate)
~Fft ()

int      Points () const
void     Transform ()
void     CopyIn (BbArray<double> *samples)
void     PutAt ( int i, double val )

int      DetPtsToFreq(int freq)
double   GetIntensity (int i) const
int      GetFrequency (int point) const
int      HzToPoint (int freq) const
int      MaxFreq() const
int      Tape (int i) const
int      GetTPoints()

int      _Points
long     _sampleRate
int      _logPoints
double   _sqrtPoints
int      *_aBitRev
Complex  *_X
Complex  **_W
double   *_aTape
int      _TPoints
    
```

3.9 Übersicht aller implementierten Klassen

Im folgenden werden alle Klasse vorgestellt, die eigens für dieses Projekt entwickelt wurden. Keine Beachtung finden hier die Dialog-, Dokument- und Applikations-Klassen der MFC. Die Dokumentation der Methoden und Membervariablen dieser Klassen ist der Kommentierung des Quelltextes zu entnehmen.

3.9.1 CSoll

Objekte dieser Klasse speichern den Sollwert (Noten) zu einem bestimmten Zeitpunkt. Sie werden in der Klasse CListEntry verwendet. Sollwerte sind notwendig für den Lernvorgang. Damit das Netz lernen kann, müssen ihm zu bestimmten Input-Werten die gewünschten Ausgabe-Werte vorliegen.

<pre> class CSoll : public CObject </pre>	
<pre> CSoll(void) ~CSoll(void) virtual void Serialize(CArchive& ar) void SetOffset(DWORD o) DWORD GetOffset() void SetNote(int idx, bool set) bool GetNote(int idx) </pre>	<pre> - Konstruktor - Destruktor - Methode zum Speichern und Laden eines Objektes - Setzt den Zeitpunkt des Sollwertes - Gibt den Zeitpunkt zurück - Setzt eine Note (anhand des Index) auf wahr oder falsch) - Liefert zurück, ob die Note (anhand des Index) gesetzt ist </pre>

<code>bool IsEmpty()</code>	- Überprüft, ob zu dem Zeitpunkt keine Note gesetzt ist
<code>DWORD offset</code> <code>BbArray<bool> *noten</code>	- Gibt den Zeitpunkt an - Array, welches 61 boolsche Werte enthält (für jede Note einen)

3.9.2 CListEntry

Objekte dieser Klasse repräsentieren einen Datensatz im Dataset. Der Begriff des Datasets wird später noch erklärt. Die Objekte beinhalten den Dateinamen, sowie dessen Verwendungszweck und im Falle einer Lern- oder Testdatei die Sollwerte (erst nach der Analyse natürlich).

<code>class CListEntry : public CObject</code>	
<code>CListEntry(void)</code> <code>CListEntry(CString p, CString fn, int pl)</code> <code>~CListEntry(void)</code> <code>bool CreateWav(void)</code> <code>void SetTreeHandle(HTREEITEM h)</code> <code>HTREEITEM GetListHandle(void)</code> <code>CString GetFilename(void)</code> <code>CString GetPath(void)</code> <code>virtual void Serialize(CArchive& ar)</code> <code>CPlaySound * GetSoundHandle(void)</code> <code>CCaptureSound *GetCSoundHandle(void)</code> <code>int GetPlace()</code> <code>void SetPlace(int p)</code> <code>void SetNewCSoundHandle()</code>	- Default-Konstruktor - Konstruktor - Destruktor - Erzeugt das Sound-Objekt (CPlaySound) - Speichert die Position im Dateibaum - Liefert die Position im Dateibaum - Liefert den Dateinamen zurück - Liefert den kompletten Pfad zurück - Methode zum Speichern und Laden eines Objektes - Liefert Zeiger auf das Sound-Objekt - Liefert Zeiger auf Sound-Aufnahme-Objekt - Liefert die Information zu welcher Kategorie die Datei gehört (Lern-, Test- oder Anwendungsdatei) - Setzt die Kategorie-Information - Erzeugt ein neues Objekt zur Sound-Aufnahme
<code>CPlaySound snd</code> <code>CCaptureSound *csnd</code> <code>CString filename</code> <code>CString path</code> <code>int place</code> <code>HTREEITEM listhandle</code> <code>BbArray<CSoll*> *values</code>	- Objekt zum Abspielen der Wav-Datei - Objekt zur Aufnahme einer Wav-Datei - Dateiname der Wav-Datei - kompletter Pfad der Datei - Information über Kategorie der Datei (0 = Trainingsdatei, 1 = Testdatei, 2 = Anwendungsdatei) - Position im Dateibaum - Sollwerte der Datei bzw. Ergebnis der Anwendung

3.9.3 CNeuron

Objekte dieser Klasse stellen ein einzelnes Neuron des neuronalen Netzes dar. Das gesamte Netz wird mit der Klasse CBProp realisiert. Da alle Informationen bezüglich Lernrate, Gewichte, Aktivierungsfunktion etc. für jedes Neuron einzeln festgelegt sind, könnte ein anders implementiertes Netz (auf Grundlage dieser Klasse CNeuron) für jedes Neuron einzeln konfigurierbar sein.

<pre>class CNeuron : public CObject CNeuron(void) ~CNeuron(void) CNeuron(int n, int p, BbArray<CNeuron*> *pr, int act = LOGISTIC, double l = 0.28, double a = 0.9) void InitWeights() void SetOutput(double o) double GetOutput() double GetGradient() double GetWeight(int idx) void CalcNewOutput() void CalcGradient(bool o, double lo, int idx = 0) void UpdateWeights() void Activate() double Derivative() SetNextLayer(BbArray<CNeuron*> *n) void SetPrevLayer(BbArray<CNeuron*> *p) void SetLearn(double l) void SetMomentumAlpha(double a) void SetActType(int t) virtual void Serialize(CArchive& ar)</pre>	<ul style="list-style-type: none"> - Default-Konstruktor - Destruktor - Konstruktor (Erzeugt ein Neuron, mit -p- Vorgängern, -n- Nachfolgern, den VorgängerNeuronen -pr- (Pointer), dem Aktivierungstyp -act-, der Lernrate -l- und der Momentumkonstanten -a-) - Initialisiert die Gewichte der eintreffenden Verbindungen mit Zufallswerten - Setzt den Output eines Neurons (wichtig für Input-Neuronen, deren Output nicht berechnet wird, sondern als Input-Werte kommt) - Liefert das Output des Neurons - Liefert den zuletzt berechneten Gradienten des Neurons - Liefert ein Gewicht anhand des Index - Berechnet das Netto-Input und schickt dieses an die Aktivierungsfunktion - Berechnet den Gradienten mit -o- als Flag, ob Output-Neuron oder Hidden-Neuron, -lo- zu lernender Output (bei Output-Neuron) und -idx- für Auffinden der benötigten Gewichte - Berechnet die Gewichte neu - Berechnet mit dem -net_in- den Output -out- mittels Aktivierungsfunktion - Berechnet den Wert der Ableitung der Aktivierungsfunktion von -net_in- - Setzt die Pointer zum nächsten Layer - Setzt die Pointer zum vorigen Layer - Parameter-Set-Methoden... - Methode zum Laden und Speichern des Objektes
<pre>BbArray<CNeuron *> *next BbArray<CNeuron *> *prev int nextneurons int prevneurons BbArray<double> *weights BbArray<double> *deltaweights BbArray<double> *olddeltaweights</pre>	<ul style="list-style-type: none"> - Pointer zu Neuronen nächster Schicht - Pointer zu Neuronen voriger Schicht - Anzahl der Folgeneuronen - Anzahl der Vorgängerneuronen - Gewichte der Input-Connections - Delta-Werte "--" - Alte Delta-Werte "--"

<pre>double net_in double out double gradient int activation_type double learn double alpha</pre>	<ul style="list-style-type: none"> - Nettoinput - Output - Gradient - Aktivierungsfunktion (0 = tanh , 1 = logistisch) - Lernfaktor - Konstante für Momentum-Term
--	---

3.9.4 CBProp

Diese Klasse repräsentiert nun ein komplettes neuronales Backpropagation-Netz. Sie baut auf der Klasse CNeuron auf.

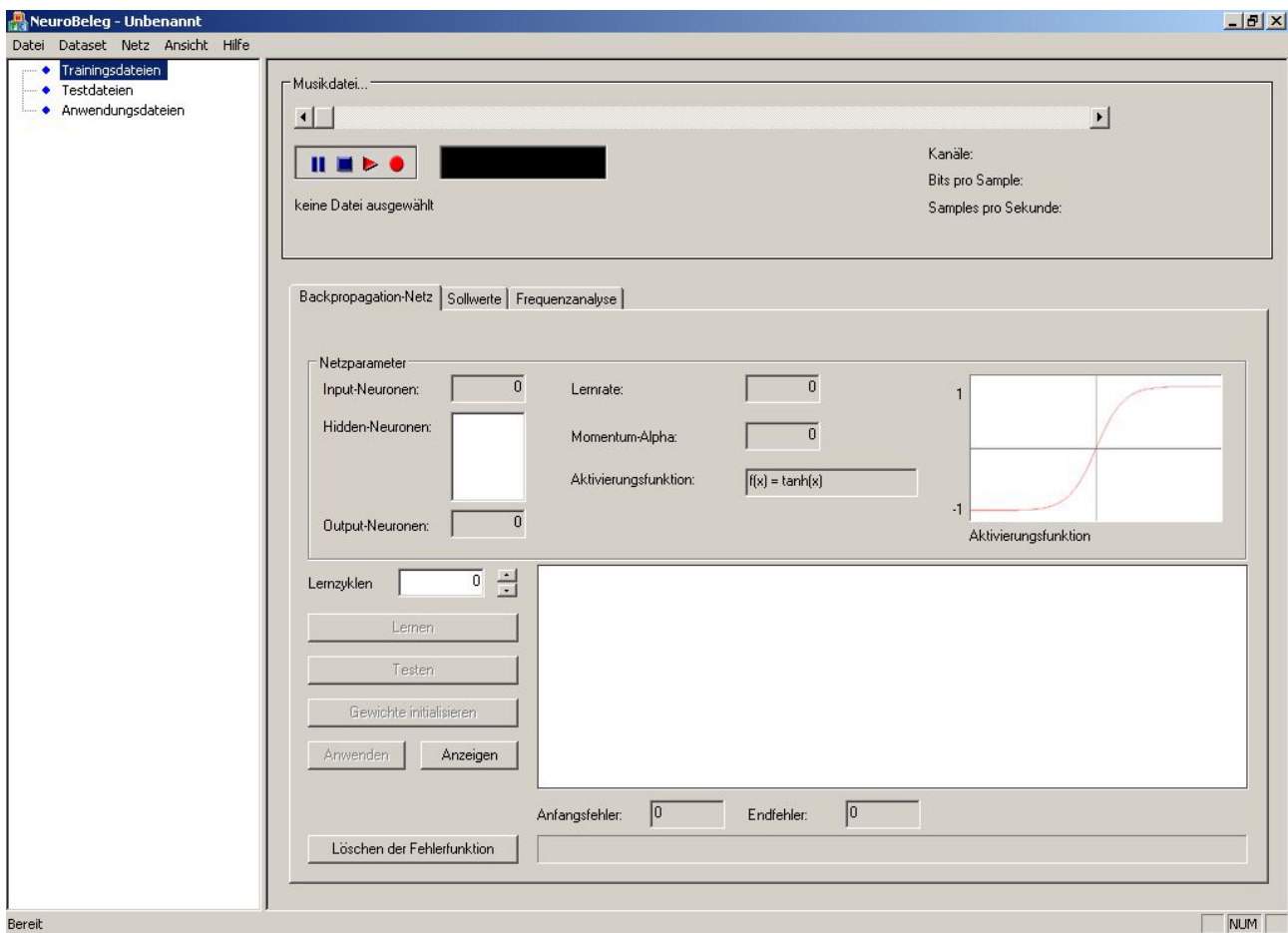
<pre>class CBProp : public CObject CBProp(void) ~CBProp(void) virtual void Serialize(CArchive& ar) void CreateNet(int in, BbArray<int> *h, int out) void InitializeWeights() double Learn(BbArray<double> *input, BbArray<double> *loutput) void ForwardPass(BbArray<double> *input) void BackwardPass(BbArray<double> *loutput) void UpdateWeights() void CalculateOutput(BbArray<double> *input, BbArray<double> *outp) double GetError(BbArray<double> *loutput) void SetLearn(double l) void SetMomentumAlpha(double a) void SetActType(int t) double GetLearn() double GetMomentumAlpha() int GetActType() int GetInNeurons() int GetOutNeurons() BbArray<int> *GetHiddenLayers() int in_neurons BbArray<int> *hidden_neurons</pre>	<ul style="list-style-type: none"> - Konstruktor - Destruktor - Methode zum Laden und Speichern des Objektes - Legt Netz an mit -in- Input-Neuronen, -h1,h2,...,hn- Hidden-Neuronen (schichtweise) und -out- Output-Neuronen - Belegt die Gewichte mit zufälligen Werten (Intervall [-0.7; 0.7]) - Trainiert das Netz mit einem Trainingsdatensatz bestehend aus Input-Werten und Soll-Output-Werten - Berechnet das Output anhand von Input-Werten (nur intern zu verwenden) - Berechnet die Fehler (--> Bestimmung der Gradienten) (nur intern zu verwenden) - Aktualisiert die Gewichte nach Lernschritt (nur intern zu verwenden) - Liefert das Output, anhand eines Inputs (zum Testen / Anwenden des Netzes) - Liefert den Fehler des letzten Forward-Schrittes (nur intern zu verwenden /bzw. nach CalculateOutput) - Parameter-Set-Methoden... - Parameter-Get-Methoden - Liefert die Anzahl der Input-Neuronen - Liefert die Anzahl der Outp.-Neuronen - Liefert Array mit Hidden-Neuronen - Anzahl der InputNeuronen - Anzahl der Hidden-Neuronen pro
--	---

<pre>int out_neurons int hidden_layers BbArray<CNeuron*> m_in BbArray<CNeuron*> *m_out BbArray<void*> *m_hidd</pre>	<p>Schicht</p> <ul style="list-style-type: none"> - Anzahl der Output-Neuronen - Anzahl der Hidden-Layer - Array mit Input-Neuronen - Array mit Output-Neuronen - Array mit Arrays mit Neuronen der Hidden-Schichten
---	---

4 Die Software

4.1 Der initiale Zustand

Wird die Software gestartet, so erhält man zunächst folgendes Bild:



Grafik 4.1: Die Software

4.2 Das Hauptmenü

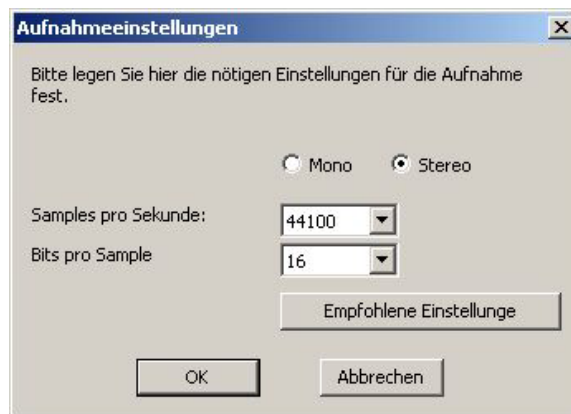
Im folgenden werden die einzelnen Menüpunkte "Datei ...", "Dataset ..." und "Netz ..." beschrieben.

4.2.1 Datei – Importieren

Dieser Menüpunkt ermöglicht es WAV-Dateien in das Dataset zu importieren. Dabei öffnet sich ein Dialog zum Auswählen einer oder mehrerer Dateien. Wird der Dialog erfolgreich beendet, so werden die Dateien unter einem der drei Verzeichnispunkte im Dateibaum eingefügt. Ausschlaggebend ist dabei, welches Verzeichnis dabei gerade selektiert ist, bzw. zu welchem Verzeichnis die aktuell selektierte Datei gehört.

4.2.2 Datei – Neue Aufnahme

Mit diesem Menüpunkt kann der Nutzer eine neue WAV-Datei erstellen. Dazu öffnet sich folgender Dialog für die Aufnahmeeinstellungen:



Grafik 4.2: Neue Aufnahme

Für die neue Datei kann man die Anzahl der Kanäle festlegen (also Mono oder Stereo). Es können die Samples pro Sekunde gewählt werden. Die Auswahl beschränkt sich auf 22050 oder 44100 Samples pro Sekunde. Und letztlich kann noch die Anzahl der Bits pro Sample eingestellt werden, wobei zwischen 8 und 16 gewählt werden kann.

Der Button "Empfohlene Einstellungen" stellt die Werte auf obige Einstellung zurück.

Wird der Dialog mit "Ok" beendet, so wird die Datei unter dem daraufhin zu wählenden Namen angelegt und in den Dateibaum eingefügt nach dem selben Prinzip wie beim Importieren von Dateien. die Datei steht nun zur Aufnahme bereit.

4.2.3 Datei – Entfernen

Dieser Menüpunkt ist nur aktiv, wenn eine Datei im Dateibaum selektiert ist. diese kann dann aus dem Dataset entfernt werden. Das ist genauso möglich, indem man mit der rechten Maustaste ein Popup-Menü öffnet und dort "Entfernen" wählt.

4.2.4 Datei – Programm beenden

Dieser Menüpunkt beendet das Programm.

4.2.5 Dataset – Neu

Ein Dataset ist als eine Art Datensammlung zu verstehen. Wurde eine optimale Auswahl an Dateien zum Lernen, Testen und/oder Anwenden getroffen, so kann diese Auswahl als Dataset gespeichert werden. Der Vorteil liegt desweiteren darin, dass Sollwerte (später erklärt) ebenfalls erhalten bleiben und die Dateien bei einer späteren Anwendung nicht mehr analysiert werden müssen bevor gelernt oder getestet werden kann. Um die Dataset-Datei aktuell zu halten, sollten die vermerkten Musikdateien ihren Pfad nicht verändern, bzw. i, selben Verzeichnis wie die Dataset-Datei zu finden sein. Ansonsten schlägt ein Laden des Datasets fehl. Folgende Informationen beinhaltet das Dataset:

- Anzahl der vermerkten Dateien
- Zu jeder Datei:
 - komplette Pfadangabe
 - Dateiname
 - Information zur Position im Dateibaum (Lern-, Test- oder Anwendungsdatei)
 - Anzahl der Sollwerte (später erklärt)
 - zu jedem Sollwert
 - Zeitpunkt
 - für jede Note die Information, ob gesetzt oder nicht

Mit dem Menü-Punkt "Neu" kann ein neues Dataset angelegt werden, d.h. sämtliche alte Sounddateien werden aus dem Dateibaum gelöscht.

4.2.6 Dataset – Laden

Das Laden eines Datasets wird ermöglicht, indem man eine zuvor gespeicherte ".dts"-Datei auswählt. Diese Datei enthält Informationen über die ausgewählten Sounddateien und ihre Sollwerte (Erklärung später), sofern denn welche vorliegen. Die Dateien werden genauso wieder in den Dateibaum eingefügt, wie als das Dataset gespeichert wurde.

4.2.7 Dataset – Speichern

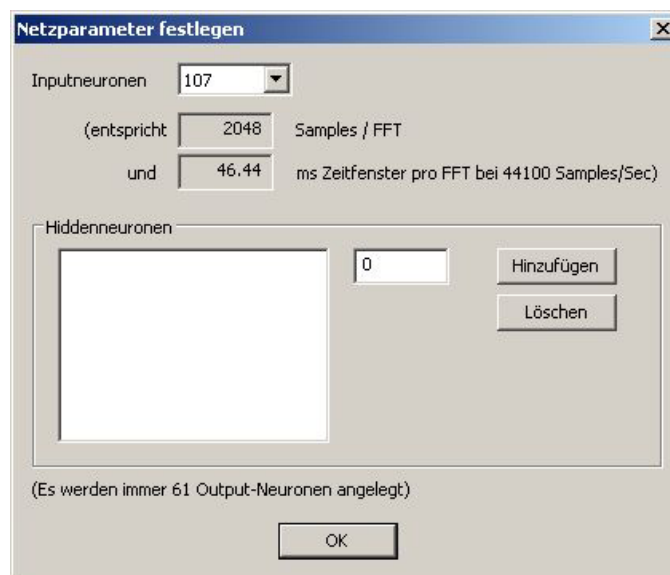
Hier kann ein Dataset gespeichert werden. Besitzt es noch keinen Dateinamen, so erfolgt automatisch ein "Speichern unter". Die Datei wird als ".dts" abgelegt.

4.2.8 Dataset – Speichern unter

Mit diesem Menüpunkt kann ein Dataset unter einem gewünschten Namen abgespeichert werden.

4.2.9 Netz – Neues Netz

Mit diesem Menüpunkt kann ein neues Netz angelegt werden. Dazu wird als erstes ein Dateiname für das Netz ausgewählt. Dann erscheint folgender Dialog:



Grafik 4.3: Neues Netz

Die Wahl der Input-Neuronen ist beschränkt auf 27, 54, 107 und 214. Das liegt daran, da die Eingabedaten von der Frequenzanalyse stammen. Die Wahl der Inputneuronen erzeugt zu Information die Anzeige, wie viel Samples dafür für die FFT benötigt werden und welchem Zeitfenster diese Anzahl entspricht. Die Angabe des Zeitfensters gilt jedoch nur für den Fall, dass die Datei mit 44100 Samples pro Sekunde aufgenommen wurde.

Die Anzahl der Output-Neuronen ist festgelegt auf 61, da ein normales Keyboard über 61 Tasten verfügt.

Die Hiddenschichten können angelegt werden, indem man die gewünschte Anzahl von Neuronen pro Schicht in das kleine Editfeld eingibt und den Button "Hinzufügen" betätigt. Die angelegte Schicht erscheint

nun im Tabellenfenster, wo sie wieder ausgewählt und entfernt ("Löschen") werden kann. So können beliebig viele Schichten angelegt werden.

Für jedes Netz muss mindestens eine Hidden-Schicht angelegt werden.

Wird nun der Dialog mit "Ok" beendet, so wird das Netz angelegt und im Tab-Fenster "Backpropagation-Netz" angezeigt.

4.2.10 Netz – Laden

Hier kann ein bereits als ".net"-Datei gespeichertes Netz mit allen Einstellungen wieder geladen werden. Es wird daraufhin im Tab-Fenster "Backpropagation-Netz" angezeigt.

4.2.11 Netz – Speichern

Ein konfiguriertes Netz kann mit diesem Menüeintrag gespeichert werden. Hat das Netz noch keinen Dateinamen, so findet "Speichern unter" Anwendung.

4.2.12 Netz – Speichern unter

Hier kann ein Netz unter einem gewählten Namen abgespeichert werden.

4.2.13 Netz – Konfigurieren

Dieser Menüeintrag ermöglicht die Veränderung von Parametern des Netzes. Veränderungen in der Struktur sind hierbei ausgeschlossen. Dafür müsste ein neues Netz angelegt werden. Wird dieser Eintrag ausgewählt, so erscheint folgender Dialog:



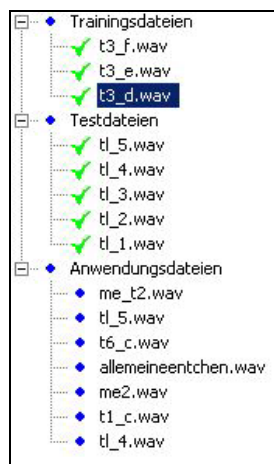
Grafik 4.4: Konfigurieren

Die Parameter "Lernrate" und "Momentum-Alpha" können als Kommazahl eingegeben werden. Bei der Aktivierungsfunktion kann zwischen der logistischen Funktion und Tanh(x) gewählt werden. Die optimalen Einstellungen bedingen sich durch Netz-Struktur und Eingabedaten.

Wird mit "Ok" beendet, so werden diese Daten für das geöffnete Netz übernommen.

4.3 Der Dateibaum

Der Dateibaum auf der linken Seite zeigt immer das aktuelle Dataset an, mit dem gearbeitet wird. Dateien, die unter dem Verzeichnis "Lerndateien" stehen, werden zu Lernzwecken genutzt, Dateien unter "Testdateien" zu Testzwecken und Dateien unter "Anwendungsdateien" stehen zur endgültigen Anwendung des fertig trainierten Netzes zur Verfügung.

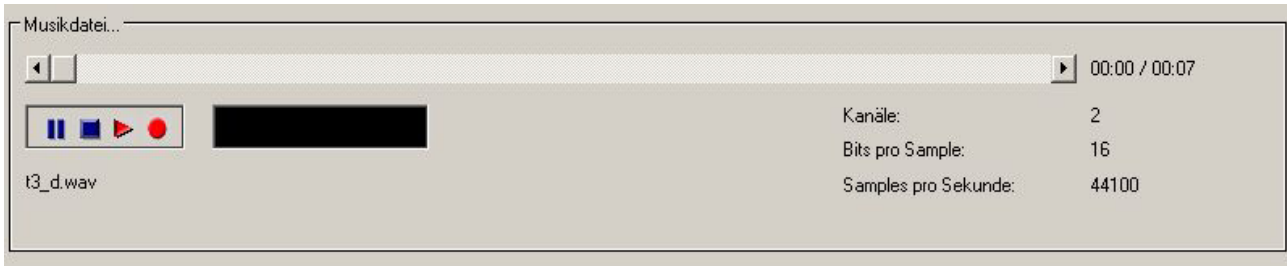


Grafik 4.5: Dateibaum

4.4 Die Anzeige der Daten

Auf der rechten Seite der Anwendung finden sich jede Menge Anzeigen von Daten, weshalb hier vom "Anzeige-Fenster" gesprochen wird. Das Anzeige-Fenster gliedert sich in zwei Teile – den oberen Teil, der zum Aufnehmen und Abspielen von Sounddateien dient, und den unteren Teil, der ein Tabulator-Control enthält mit verschiedenen Tab-Fenstern.

Als nächste folgt die Beschreibung des oberen Teils des Anzeige-Fensters. Hier findet sich eine Laufleiste, die den Fortschritt beim Abspielen einer Datei angibt. Darunter finden sich vier kleine Grafiken, die in der Reihenfolge die Funktionen "Pause", "Stop", "Abspielen" und "Aufnehmen" bilden. In dem schwarzen Kasten rechts daneben, blinkt mit roter Schrift der Hinweis "Aufnahme", wenn aufgenommen wird. Des weiteren finden sich Angaben zu den Eigenschaften der ausgewählten Datei.



Grafik 4.6: Informationen zur Datei + Möglichkeiten zum Abspielen und Aufnehmen

4.4.1 Tabfenster – Backpropagation-Netz

Dieses Fenster bietet alle Informationen zum Neuronalen Netz. Der obere Teil zeigt Struktur und Parameter des Netzes an. Der untere Teil bietet verschiedenen Funktionen.

Der Button "Lernen" ruft den Lernvorgang auf. dabei müssen Dateien im Dateibaumordner "Lerndateien" vorhanden und analysiert sein. Dann werden die vorhandenen Muster gelernt. Die Anzahl der Lernzyklen, gibt an, wie oft die Muster präsentiert werden. Der Fehler wird dabei grafisch ausgegeben. Es ist der summierte Wert über einen Zyklus. Unter der grafischen Ausgabe zeigt eine Fortschrittsanzeige an, wie weit bisher gelernt wurde. Außerdem werden die Fehler nach dem ersten sowie letzten Lernzyklus angegeben.

Der Button "Testen", gibt alle analysierten Dateien aus dem Dateibaumordner "Testdateien" an das Netz. Als Ausgabe erfolgt ein Fenster, welches die einzelnen Muster und ihre Fehler enthält. So kann überprüft werden, welche Muster noch nicht optimal gelernt wurden.

The screenshot shows a window titled 'Testergebnisse' with a table of test results. The table has two columns: 'Muster' and 'Fehler'. The data is as follows:

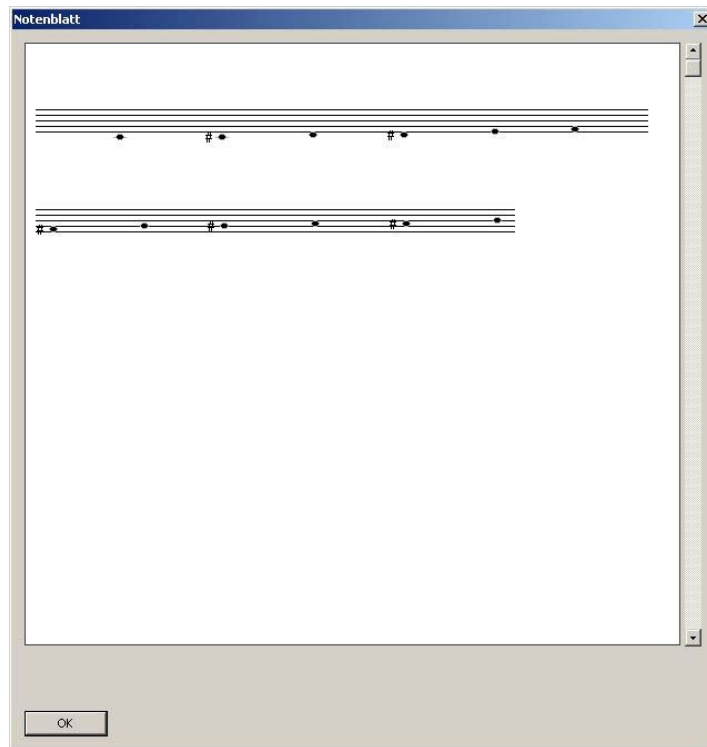
Muster	Fehler
tl_5.wav - Pattern 1	0.0760475467
tl_5.wav - Pattern 2	0.0628859208
tl_5.wav - Pattern 3	0.0627882890
tl_5.wav - Pattern 4	0.0794311237
tl_5.wav - Pattern 5	0.0641727513
tl_5.wav - Pattern 6	0.0701924793
tl_5.wav - Pattern 7	0.0656080644
tl_5.wav - Pattern 8	0.0688415307

An 'OK' button is located at the bottom of the window.

Grafik 4.7: Testergebnisse

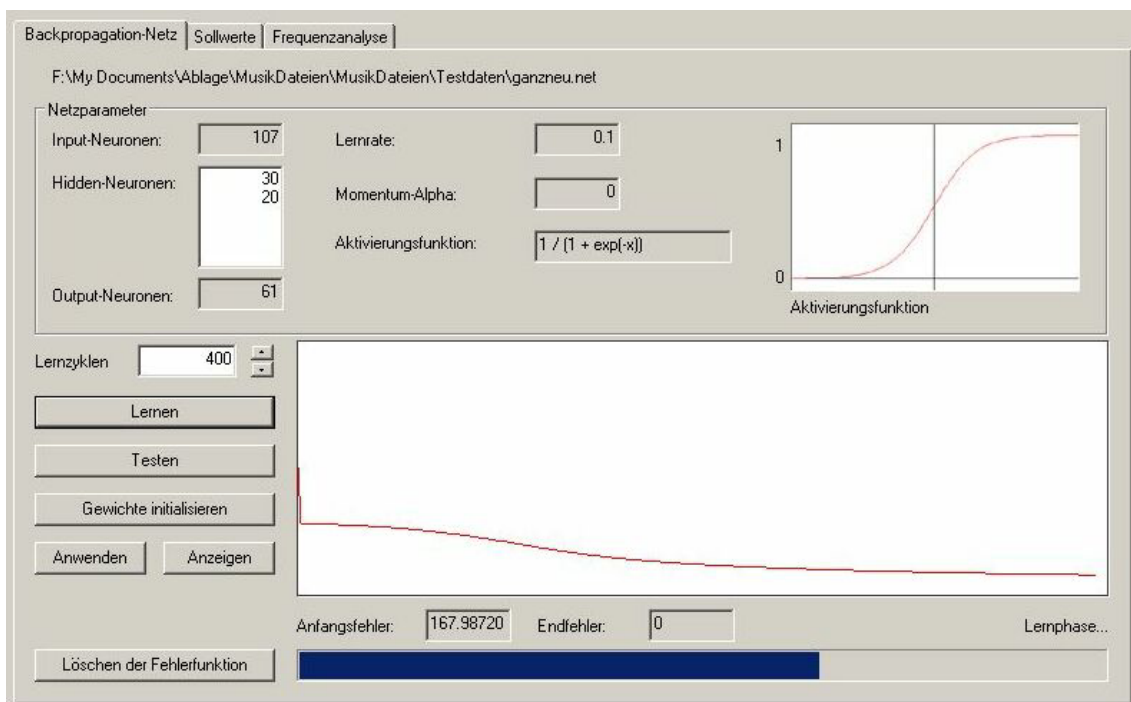
Der Button "Gewichte initialisieren" berechnet neue Zufallsgewichte für das Netz. Damit geht jeglicher bisheriger Lernerfolg verloren.

Der Button "Anwenden" schickt die ausgewählte Anwendungsdatei durch eine Analyse. Die Daten der gefundenen Töne werden an das neuronale Netz weitergegeben und dessen Ausgabe gespeichert. Der Button "Anzeigen" zeigt dann diese Ausgabe in Form von Noten an.



Grafik 4.8: Ausgabe der Noten

Der Button "Löschen der Fehlerfunktion" löscht den aufgezeichneten Fehler.



Grafik 4.9: Informationen zum Neuronalen Netz

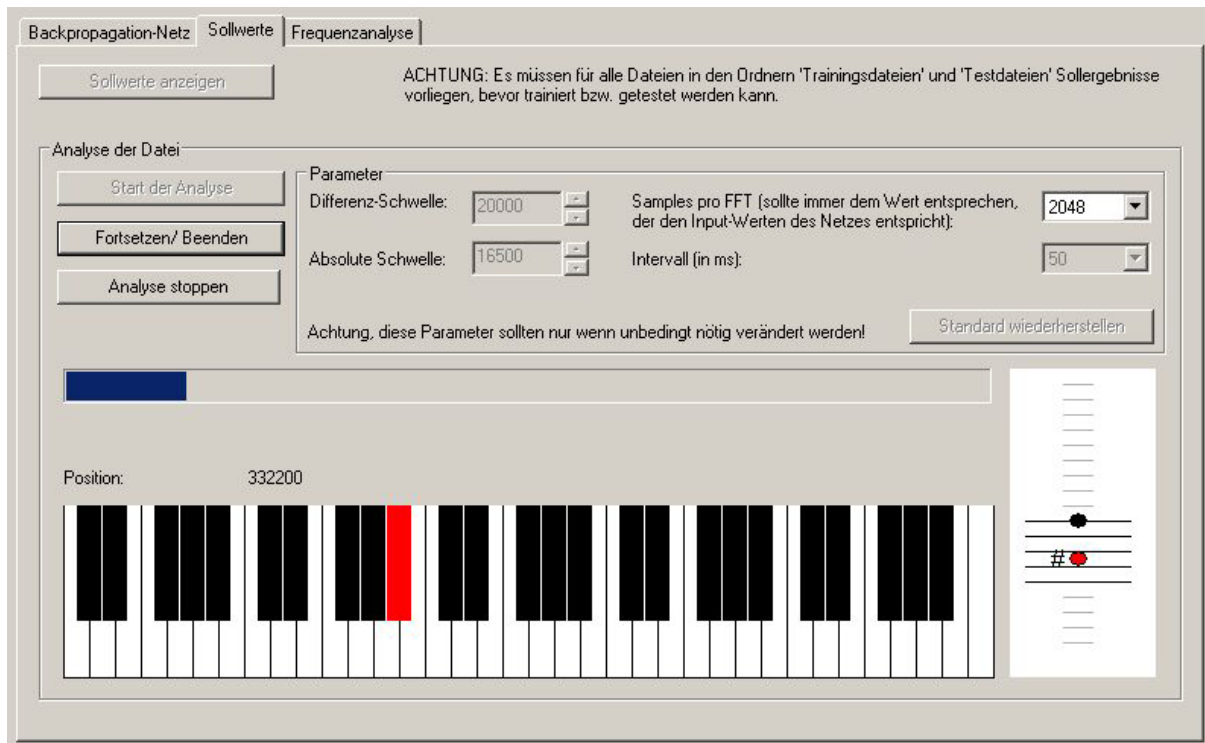
4.4.2 Tabfenster – Sollwerte

Dieses Fenster ermöglicht die Angabe der Sollwerte der Test- und Lerndateien. Sollwerte werden zum Testen und Lernen benötigt. Um das Netz zu trainieren muss man ihm Eingabedaten und zugehörige Ausgabewerte präsentieren. Beim Testen werden Inputdaten an das Netz gesendet und die Ausgabe mit den erwartete Ausgabewerten verglichen. Diese Ausgabewerte werden hier als Sollwerte bezeichnet. Um einer Datei Sollwerte zuzuordnen muss diese analysiert werden, d.h. es müssen die Stellen entdeckt werden, an denen eine Note gespielt wird und zu diesen Stellen muss der Nutzer den tatsächlich gespielten Ton/ Akkord angeben.

Analysierte Dateien werden im Dateibaum mit einem grünen Häkchen dargestellt, die anderen mit einem roten Kreuz. Wurden bereits Sollwerte analysiert, so können diese mit dem Button "Sollwerte anzeigen" in Form von Noten angezeigt werden.

Der Analyse liegen verschiedene Parameter zu Grunde. Die Schwellenwerte können verändert werden, aber das ist lediglich zu Testzwecken bezüglich des Analysealgorithmus sinnvoll. Die Analyse von Anwendungsdateien läuft mit festen Werten (die den bereits eingestellten Werten entsprechen). Die Anzahl der Samples pro FFT sollte am besten so gewählt werden, wie sie auch für das Netz benötigt wird. Unterschiedliche Werte können zu unterschiedlichen Erkennungspunkten führen und daher größere Fehler erzeugen. Das Intervall ist mit 50 Millisekunden ebenfalls ein Wert, der so belassen werden sollte.

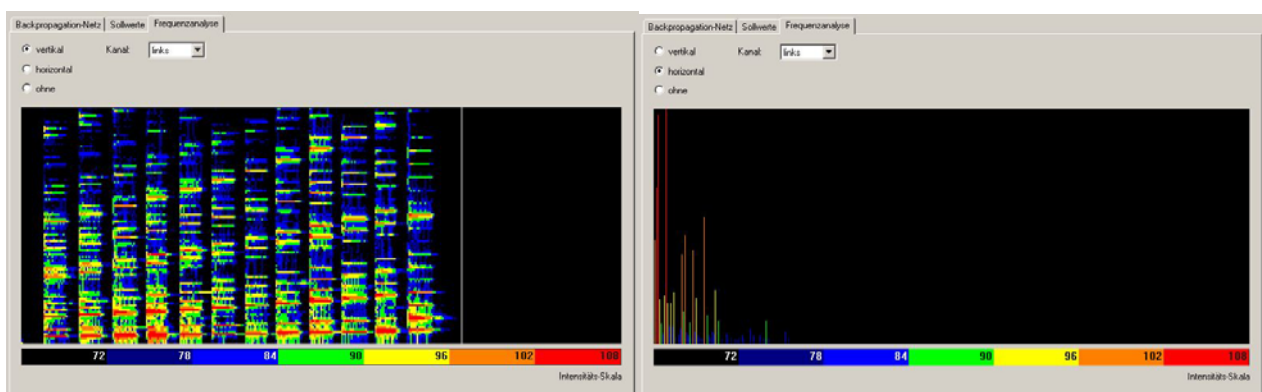
Mit dem Button "Start der Analyse" kann nun eine ausgewählte Datei analysiert werden. Dabei gibt eine Fortschrittsanzeige an, wie weit die Analyse ist. Wird ein Ton gefunden, bleibt die Analyse stehen. Wenn man nun mit der Maus eine Klaviertaste anklickt, so wird diese Note in die Notenlinien gezeichnet. Nochmaliges Klicken löscht diese Note wieder. Auf diese Weise können ein oder mehrere Noten bestimmt werden, die der Position entsprechen sollen. Dann wird mit dem Button "Fortsetzen/ Beenden" die Analyse fortgesetzt. Es ist auch möglich keinen Ton auszuwählen und gleich fortzusetzen, wenn z.B. ein tiefer Ton doch doppelt erkannt werden sollte. Wird kein Ton mehr gefunden, gilt die Analyse als beendet. sie kann auch jederzeit mit dem Button "Analyse stoppen" beendet werden.



Grafik 4.10: Analyse zur Bestimmung der Sollwerte

4.4.3 Tabfenster – Frequenzanalyse

Dieses Fenster bietet die Möglichkeit die Frequenzwerte während des Abspielens anzuzeigen. Dabei kann der Kanal gewählt werden und die vertikale oder horizontale Anzeige. Bei der horizontalen Anzeige liegen die Frequenzen über den "Horizont" verteilt und Impulslinien geben deren Intensität an. Bei der vertikalen Anzeige, sind die Frequenzen von unten nach oben verteilt und die Farben geben Informationen über deren Intensitäten. Die Frequenzen der vertikalen Anzeige werden nur bis 2300 Hz angezeigt.



Grafik 4.11: Anzeige des Frequenzspektrums vertikal und horizontal

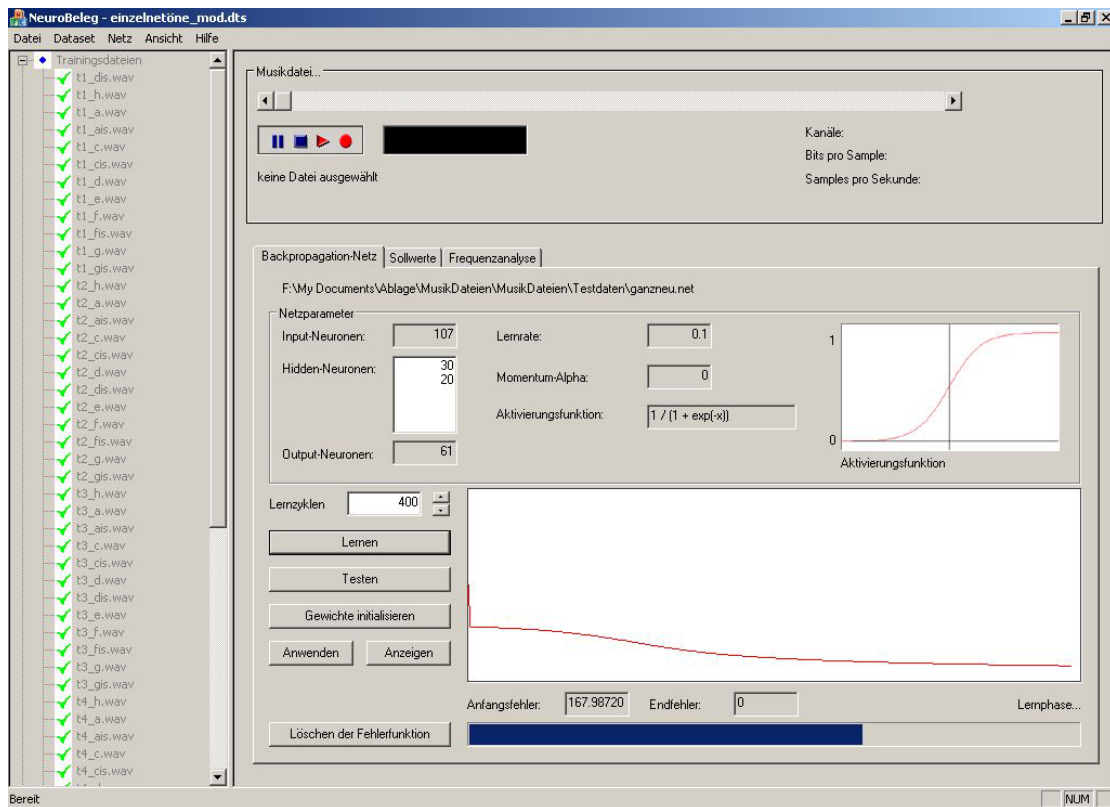
5 Anleitung zum Lernen

Im folgenden werden stichpunktartig die Schritte genannt, die zum Trainieren eines Netzes benötigt werden:

- Anwendung öffnen
- Musikdateien importieren, aufnehmen oder als Dataset öffnen
- Eine noch nicht analysierte Lern- oder Testdatei auswählen
- Tabfenster "Sollwerte" aktivieren
- Analyse starten
- Bei gefundenem Ton mit der Maus auf dem Klavier die Note(n) auswählen (Kontrolle im Fenster rechts daneben) und die Analyse fortsetzen
- Kontrolle der analysierten Töne durch die Anzeige der Sollwerte
- Wenn alle Dateien analysiert wurden → Netz öffnen oder erstellen (Achtung: Die Anzahl der Samples der Analyse sollte mit denen im Lernvorgang übereinstimmen → Anzahl der Eingabe-Neuronen danach wählen)
- Parameter gegebenenfalls einrichten
- Anzahl der Lernzyklen eingeben
- Lernen lassen
- Netz abspeichern

6 Eigene Erfahrungen und Tests

Tests mit der Lernsoftware waren relativ schwierig durchzuführen, da der Lernvorgang oft einen erheblichen Zeitaufwand bedeutete. Folgender Screenshot zeigt einen Anwendungsfall:



Grafik 6.1: Lernergebnis 1

Als Lerndaten, wurden hier 61 Dateien verwendet, die jeweils 5 Töne der selben Note enthielten. Somit lagen 305 Muster bereit. Die Daten des Netzes können der Grafik entnommen werden. Für 400 Lernzyklen benötigte das Netz etwa eine Stunde (Pentium III – 500 MHz). Ein Endfehler von 36.6 entspricht einem Fehler von 0.12 pro Muster, was schon recht gut ist, da nicht jedes Muster optimal ist. Gerade tiefe Töne werden schlecht gelernt. Die Gründe wurden bereits besprochen. Ein Nachtrainieren bestimmter Töne kann manchmal Erfolg bringen.

Ein weiteres Problem sind Dateien, die sehr "fließend" klingen. Hier versagt die Analyse, da sie nur noch schlecht Töne erkennen kann.

Außerdem bringen zu leise Dateien manchmal Probleme in der Analyse mit sich. Fehlen also Töne komplett bei der Anwendung, so kann es sein, dass die Töne zu leise sind.

7 Erweiterungen / Verbesserungen

Verbesserungswürdig ist zum einen die Analyse der Sounddatei. Man kann den Algorithmus bezüglich der Erkennung von Noten sicherlich noch optimieren.

Zum anderen gibt es etliche Verbesserungen, die an der Software vorgenommen werden können. Dazu gehören z.B. eine komfortablere Darstellung der Noten (mit Drucken evtl.) und die Möglichkeit Lern-, Test- und Anwendungsvorgänge anzuhalten.

Des weiteren könnte man darüber nachdenken, ob eine andere Art von Netz eventuell einen größeren Lernerfolg versprechen würde.

8 Bekannte Fehler

- Der Fortschrittsbalken beim Anwenden fängt mitunter einfach noch mal von vorn an.
- Die Aufnahme von einer Datei klappt nur einmal. Danach hängt der Sound an bestimmten Stellen.
- Beim Speichern des Datasets erfolgt das nicht immer automatisch als ".dts"-Datei.

9 Systemanforderungen / Entwicklungsumgebung

Das Projekt wurde mit Microsoft Visual C++ .NET / Microsoft .NET Framework 1.0 entwickelt. Die Programmiersprache ist C++, die Oberfläche wurde mit MFC (Microsoft Foundation Classes) gestaltet.

Um die Software auch auf Windows Systemen ohne .NET Framework ausführen zu können, wurde ein Setup-Projekt realisiert, welches die Software samt Dokumentation installiert. Das Erstellen des Setup-Projektes, sowie die endgültige Generierung der Anwendung wurden mit Microsoft Visual C++ .NET / Microsoft .NET Framework 1.1 vorgenommen.

Der Rechner sollte außerdem über mindestens 500 MHz Rechenleistung verfügen, da Lern- und Anwendungsvorgänge sehr rechenintensiv sind.

10 Quellen

- V. Sirotin, V. Debeloff, Y. Urri – "DirectX-Programmierung mit Visual C++, Addison-Wesley Longman Verlag, 1999
- S. Heitsiek – "Das Einsteigerseminar, Windows-98-Programmierung mit Visual C++", bhv-Verlag, 1999
- URL: <http://www.relisoft.com/>
- URL: <http://www.uni-koblenz.de/~odsgroe/schallph.htm>
- URL: http://www.kjr-muenchen-stadt.de/publikationen/pdf/schall_jena.pdf