A Portable Computer System for Recording Heart Sounds and Data Modeling Using a Backpropagation Neural Network

Erik Mark Hudson

University of North Florida
A PORTABLE COMPUTER SYSTEM FOR RECORDING HEART SOUNDS AND DATA MODELING USING A BACKPROPAGATION NEURAL NETWORK

by

Erik Mark Hudson

A thesis submitted to the Department of Computer and Information Sciences in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences

UNIVERSITY OF NORTH FLORIDA
DEPARTMENT OF COMPUTER AND INFORMATION SCIENCES

December, 1995
The thesis "A Portable Computer System For Recording Heart Sounds And Data Modeling Using A Backpropagation Neural Network" submitted by Erik Mark Hudson in partial fulfillment of the requirements for the degree of Master of Science in Computer and Information Sciences has been approved by the thesis committee:

Approved by the thesis committee: Date

Signature Deleted

Dr. Layne Wallace
Thesis Adviser and Committee Chairperson

Signature Deleted

Dr. Charles Winton

Signature Deleted

Dr. Susan Wallace

Accepted for the Department of Computer and Information Sciences:

Signature Deleted

Dr. Charles Winton
Chairperson of the Department

Accepted for the College of Computing Sciences and Engineering:

Signature Deleted

Dr. Charles Winton
Interim Dean of the College

Accepted for the University:

Signature Deleted

Dr. Charles Galloway
Dean of Graduate Studies
ACKNOWLEDGEMENT

I would like to thank my wife Luanne for her support and understanding through the years. Also, I would like to dedicate this project to my daughter Felicity. Her heart problem is what generated my interest in this project and her strength and courage through tough times has always been an inspiration.
## CONTENTS

List of Figures ............................................... vii  
Abstract ........................................................ viii  

### Chapter 1: Introduction ................................. 1  
1.1 Cardiac Auscultation ................................. 2  
1.2 Cardiovascular Sounds ............................... 6  
1.3 The Physician Decision Process ................. 8  
1.4 Neural Networks in ECG Analysis ............... 9  
1.5 Neural Networks in Heart Sound Analysis .. 16  
1.6 Disadvantages of Neural Networks ............. 16  

### Chapter 2: Methodology ................................ 18  
2.1 Hardware ................................................ 18  
  2.1.1 Computer ........................................... 18  
  2.1.2 Stethoscope ........................................ 18  
  2.1.3 Microphone ......................................... 18  
  2.1.4 Preamplifier ....................................... 20  
  2.1.5 Sound Card ......................................... 23  
2.2 Software ................................................ 23  
  2.2.1 Wave Files and the Wave Editor .. 23  
  2.2.2 FORM_GEN ......................................... 30  
  2.2.3 DEM_FILE .......................................... 33
FIGURES

Figure 1: Data Representation & Packing in PCM Wave Files ....................... 29
Figure 2: Sample Demographic Information Form ........... 32
Table 1: Information Represented by Input Pattern ... 44
Table 2: Test Results for Files Used in Training .... 50
Table 3: Test Results for Files Reserved for Testing ......................... 50
Table 4: Test Results for Linear Regression ........ 52
ABSTRACT

Cardiac auscultation is the primary tool used by cardiologists to diagnose heart problems. Although effective, auscultation is limited by the effectiveness of human hearing. Digital sound technology and the pattern classification ability of neural networks may offer improvements in this area. Digital sound technology is now widely available on personal computers in the form of sound cards. A good deal of research over the last fifteen years has shown that neural networks can excel in diagnostic problem solving. To date, most research involving cardiology and neural networks has focussed on ECG pattern classification. This thesis explores the prospects of recording heart sounds in Wave format and extracting information from the Wave file for use with a backpropagation neural network in order to classify heart patterns.
Along with patient information and the results of diagnostic
tests, the cardiologist relies on skilled cardiac auscultation to make a diagnosis. Cardiac auscultation is the process
of listening to the heart sounds through a stethoscope. Al-
though auscultation is an effective tool in the diagnosis of
heart disorders, it does have limitations. Any improvement
in this technology will benefit a significant number of peo-
ple. It has been reported [Fyler92] that 1% of all children
in the United States will be referred to a pediatric cardiol-
ogist. Half of those children will require specialized fa-
cilities at some time.

This thesis is a pilot study for a project which will attempt
to classify heart murmurs using cardiac auscultation, digital
sound technology, and a backpropagation neural network. This
thesis researched the techniques required for cardiac auscul-
tation to be performed by a portable computer system. The
portability of this system is considered important for future
applications. For this thesis, the hardware configuration reflects the desire to make this system portable. A stethoscope, with a microphone inserted into one of its rubber tubes, was used to transmit the heart sounds to a sound board in a notebook computer. The sound board stored the sounds in a Wave file format. This data was then processed and used as input to a backpropagation neural network along with pertinent patient information, for classification. Backpropagation neural networks have been successfully used in a number of research projects involving electrocardiogram (ECG) classification (e.g. [Artis92], [Morabito92] [Bortolan92]), which is why it was selected for this thesis. However, only one study [Barschdorff91] was found that involved heart sound analysis and neural networks.

1.1 Cardiac Auscultation

The tool used for cardiac auscultation is the stethoscope. The end of the stethoscope applied to the chest is composed of two parts, a diaphragm and a bell. They allow the physician to hear different frequencies of the heart sound. The diaphragm and bell actually work like band pass filters. The bell transmits very low and higher frequency sounds. The very
low frequency sounds mask out the higher frequency sounds so that the higher frequency sounds are perceived as faint or absent. In effect, the bell works as a low pass filter during auscultation. The diaphragm attenuates very low frequencies and selectively transmits the higher frequencies. This gives the effect of a high pass filter. [Delman79]

Positioning is very important during auscultation. The various cardiac sounds are best heard in different locations. A systematic approach is taken in cardiac auscultation. There are five positions on the chest where auscultation is performed:

- the upper right sternal border
- the upper left sternal border
- the lower left sternal border
- the third space
- the apex

On the back, four additional locations are used:

- the left midaxillar line
- the right midaxillar line
- the left midpoint of the back
- the right midpoint of the back

The cardiologist will sample each position with both the bell
and the diaphragm for a minimum of five cardiac cycles each. Five cardiac cycles is usually sufficient for a cardiologist to diagnose any abnormality in the heart sounds.

The upper right sternal border is located to the patient’s right of the sternum at the second intercostal space. Auscultation at the upper right sternal border allows the cardiologist to hear the aortic valve, aortic stenosis, and the venous hum.

The upper left sternal border is located to the patient’s left of the sternum at the second intercostal space. At the upper left sternal border, the pulmonary valve, pulmonary flow, Patent Ductus Arteriosus, and a split second heart sound can all be heard.

The lower left sternal border is located to the patient’s left of the sternum at the fourth intercostal space. At this position, the tricuspid valve, a tricuspid click, a pulmonary click, a ventricular septal defect, and the Still’s Murmur can be heard.
The third space is located to the patient's left of the sternum at the third intercostal space. From this location, aortic insufficiency, pulmonary insufficiency, ventricular septal defects, and pulmonary flow can be discerned.

The apex is located at the lower left ventricle. At this position, the first heart sound, the mitral valve, an aortic valve click, a mitral valve click, and the Still's Murmur can be heard.

Delman and Stein [Delman79] describe the process of cardiac auscultation in great medical detail. Cardiovascular sounds, which can be categorized as either heart sounds or murmurs, are used by physicians to diagnose cardiac disease and defects. However, cardiac auscultation is limited by the threshold sensitivity of the ear and the stethoscope itself. The human ear is most efficient in the 1000-5000 Hz range, however, most significant cardiovascular sounds occur in the 20-500 Hz range. Because the stethoscope transmits both low (20-100 Hz) and higher (100-500 Hz) frequency sounds, lower frequency components tend to mask out higher frequency components. Therefore, higher frequency components are perceived
as faint or absent. Overcoming these limitations would improve cardiac auscultation.

1.2 Cardiovascular Sounds

A normal heart sound is composed of two distinct sounds. The first heart sound is produced by the closing of the mitral and tricuspid valves. The closing of the mitral valve can be heard through auscultation at the apex. The closing of the tricuspid valve can be heard through auscultation at the lower left sternal border. The second heart sound is produced by the closing of the aortic and pulmonary valves. These sounds occur very close together and can not be distinguished from each other unless separated by an interval of at least 0.02 seconds. This can be accomplished by having the patient inhale during auscultation. The aortic valve can be heard at the upper right sternal border and the pulmonary valve can be heard at the upper left sternal border. Both the first and second heart sounds are composed of low and high frequency sounds.

In children, a low pitched third heart sound may be heard through auscultation at the apex. This is considered a nor-
mal variant of the heart sounds. However, in the presence of other problems, it may be considered abnormal.

Murmurs are sound phenomena of significantly longer duration than the normal heart sounds [Delman79]. They vary in frequency, intensity, loudness, quality, and duration. Murmurs are produced by obstruction to the blood flow and/or abnormal hemodynamics of flow in the absence of organic disease. The intensity of a murmur is graded on a scale of one to six. A level one murmur is barely audible with a stethoscope. A level four murmur produces a high frequency vibration on the chest which can be felt by placing a hand on the chest. A level six murmur can be heard without a stethoscope.

According to J. H. Newburger [Newburger92], innocent heart murmurs occur in 50% of all children. An innocent murmur is not caused by a cardiac abnormality and is not associated with cardiovascular disease. Organic heart murmurs are associated with cardiac defects. Actuarial data on innocent murmurs has shown there is no change in the expected mortality of those affected. The two types of murmurs can be distinguished through skilled auscultation. The cardiologist takes patient past history, present symptoms, the physical examination, and
cardiac auscultation into account when making an evaluation of a murmur. The diagnosis is not determined by the heart sounds alone. Verification of a diagnosis is usually accomplished with an echocardiogram.

Due to the varying severity of heart murmurs, a correct diagnosis as early as possible is essential. With an innocent murmur, the patient needs to be reassured that his/her heart is functioning properly. It has been reported that the label of heart disease may have profound adverse effects on the patient and family [Newburger92]. If a murmur is not innocent, proper medical treatment from a cardiologist is required.

1.3 The Physician Decision Process

A. O. Esogbue and R. C. Elder have analyzed the physician decision process [Esogbue79]. They defined four general categories of information which physicians use to make a diagnosis. Those categories are:

- patient past history
- present symptoms
- signs observed upon physical examination
- results of clinical and diagnostic tests
The patient past history category may include information about occupation, age group, membership in a high incidence group, past diseases and disorders, and genetic history. Present symptoms may include the presence of disease and its stage of development. Upon physical examination, the physician must determine which signs are present or absent, and if present, the level of severity. A physician must also consider all possible results of tests. All of these factors affect the physician decision process.

1.4 Neural Networks in ECG Analysis

Automating diagnostic problem solving has been the focus of a large amount of work over the last 15 years [Reggia93]. Along with expert systems, neural networks have received a large portion of this research. In decision support and pattern classification problems, the supervised learning and error backpropagation paradigm has been most widely used. Neural networks have been used in a number of studies for ECG analysis. In the field of ECG classification, neural networks have shown satisfactory performance. Many works have shown them to perform better than statistical methods and some have shown them to perform as well or better than human experts.
M. R. Reddy [Reddy92] used a neural network to diagnose anterior myocardial infarction. A 12-lead ECG setup with a computerized ECG recorder was used to make digital recordings at a sampling rate of 250 Hz. The Q, R, and S amplitudes and the Q and S durations in each of the V2-V4 leads were used as input to the network. A multilayered, feedforward type network was used with backpropagation learning. There were 15 input neurons, 5 hidden neurons, and one output neuron. The output node was used to identify if anterior myocardial infarction was present or not. Learning was terminated after 1500 iterations. The neural network’s performance was compared to both a conventional program, which was developed at the Glasgow Royal Infirmary and used in computerized ECG recorders, and an experienced electrocardiographer. It performed 11% better than the conventional program and as well as the human expert.

W. G. Baxt [Baxt90] studied neural networks and acute myocardial infarction, a disease process that has been difficult to diagnose accurately. The neural network used was a multilayer perceptron trained with backpropagation by use of the McClelland and Rumelhart simulator. Only patients admitted to the coronary care unit were studied here. Forty-one input variables were used in the network. They included ECG information
along with information from the categories described by Esogbue and Stein [Esogbue79]. This information is routinely available to and utilized by physicians. The network performed with a detection rate of 92% and a 4% false alarm rate. The previous best computer program reported an 88% detection rate and a 26% false alarm rate. The author further challenged the neural network by training and testing it on patients without clear-cut evidence of acute myocardial infarction from the ECG data. For this test, the network performed with a detection rate of 86% and a false alarm rate of 8%. According to the author, the neural network simply discovered relationships in the data that are not immediately apparent to the physician.

Bortolan [Bortolan92] studied ECG classification with neural networks and cluster analysis. The network was used to identify one of six cardiac diseases or else report normal. The six diseases were:

- left ventricular hypertrophy
- right ventricular hypertrophy
- biventricular hypertrophy
- anterior myocardial infarction
• inferior myocardial infarction
• combined myocardial infarction

An ECG database at the University of Leuven was used to train and test the network. It consisted of 3266 12-lead ECG recordings. The actual input to the neural network was 37 standard ECG variables plus the patient's sex and age. The authors considered several neural networks by considering clusters of the original learning set and adjusting some components of the architecture. The authors found that for a single neural network, as the number of nodes and connections increased by a certain point, the performance on the learning set increased but the performance on the test set decreased. With clustering and a combination of seven neural networks, an improvement of 2.5% over a single neural network was observed. The results have shown that neural networks provide satisfactory performance in ECG classification. An interesting point about this study is that it used two diseases, biventricular hypertrophy and combined myocardial infarction, which are combinations of other diseases used in the study. The network had the most trouble identifying these diseases correctly. When it did not identify one of these correctly, it most often chose one of its component diseases as the cor-
rect output. The authors considered this to be partially correct.

Morabito [Morabito92] studied both backpropagation and Kohonen neural networks for the classification of ECG morphologies. This work deals with the problem of validating automatic ECG analyzers. An annotated database was used to compared output from a neural network with a physician’s analysis of a true event. The annotated database was the VALE DB. For the backpropagation network, 3 hidden layers of 10 neurons each with data normalized beat by beat yielded the best performance. Backpropagation turned out to be well suited for QRS waveform classification and the Kohonen network was found to extract a representative clustering of QRS morphologies.

S. G. Artis [Artis92] used a backpropagation neural network to detect atrial fibrillation from the ECG data. The network was created using StarNet on a Vax Station 2000 running BSD Unix 4.3. It had nine input nodes, twelve hidden nodes, and one output node. The data used was obtained from the MIT-BIH ECG Database, the MIT-BIH Atrial Fibrillation/Flutter Database, and the AHA Database for Evaluation of Ventricular Arrhythmias. The data required significant preprocessing to
create unique patterns due to the disease studied. This network demonstrated excellent rhythm classification with a positive predictive accuracy of 92.34%. The network also performed significantly better on the test database than on the training database.

J. M. Herbert [Herbert93] studied antepartum fetal ECG extraction and analysis. Two ECG signals were used in this study. One signal was obtained from the maternal thorax and the other was obtained from the maternal abdomen. The thoracic signal was used to locate the maternal contribution in the abdominal signal. A straight subtraction of the signals was performed. For this study, this method worked surprisingly well. The authors reported a success rate of 75%. This fetal ECG extraction and analysis is possible from week 20 on. Horner [Horner92] also studied noninvasive fetal ECG extraction. Their approach was to use a near optimal subtraction of the pure maternal ECG from an abdominal signal via a genetic algorithm. The model used here accounted for muscle and breathing movements and random noise in both the abdominal and thoracic signals. The resulting equation was then minimized using a genetic algorithm. Noninvasive procedures make it difficult to reliably observe the fetal ECG. However, this
approach was effective in removing the maternal ECG without distorting and/or adding misleading information to the fetal ECG.

According to C. A. Hernandez [Hernandez93], the three critical factors which affect the generalization capabilities of a neural network are the network architecture, the learning algorithm, and the training data set. The multilayer feedforward neural network with backpropagation learning has been proven to work well for medical diagnosis problems. For the training data, using the real distribution of data, rather than an equal distribution, is a better choice for generalization capability of the neural network. That is, if two cases are being studied and case A occurs twice as often as case B, the training set should be composed of 2/3 case A patterns and 1/3 case B patterns. This should yield better results when it comes to testing the neural network than if the training set was composed of 1/2 case A patterns and 1/2 case B patterns. A real distribution of data may be difficult to obtain in some cases.
1.5 Neural Networks in Heart Sound Analysis

Barschdorff [Barschodorf91] used a neural network to perform heart sound analysis. The input used for the network were the ECG, an earpulse, ultrasound data, doppler and the heart sound signals, body size, blood pressure, and age. The heart sound signals were acquired using a commercial stethoscope equipped with a sensitive microphone. The microphone used a preamplifier with a Maas-Weber filtering characteristic. According to the author, the signal produced by the preamplifier was as good or better than the original signal. The network was trained to recognize different types of heart failures. When tested with data not in the training set, it categorized the data with 100% accuracy.

1.6 Disadvantages of Neural Networks

Although neural networks have shown great promise in the field of pattern classification, they are not without their drawbacks. For example, backpropagation ignores a huge, pre-existing body of knowledge. Training can be slow and computationally expensive with neural networks. A large representative database, which may be difficult to obtain, is
also required for training. Neural networks have also not
performed well for multiple disorder diagnosis problems
[Reggia93]. Another potential problem not restricted to ne-
ural networks, but which may have some bearing on this study,
is the use of human observations as input. The interobserver
variability may decrease the accuracy of the system. A system
which takes data directly will always show the same accuracy
[Reddy92].
Chapter 2

METHODOLOGY

2.1 Hardware

2.1.1 Computer

A notebook computer was used to make the experiment easily portable. The computer used was a MEI Winbook. It has an Intel 486 25 MHz processor and runs Microsoft Windows 3.1.

2.1.2 Stethoscope

The stethoscope used for this study was a Proscope 640, made by the American Diagnostic Corporation. It is a Sprague Rapaport Type Professional Stethoscope with both a diaphragm and a bell. It was supplied by the University of North Florida.

2.1.3 Microphone

The microphone used was a Realistic Tie Tack Back Electret Condenser Microphone from Radio Shack (Cat # 33-1052). It has
the following technical specifications:

- Frequency Response: 50 - 15000 Hz
- Impedance: 800 ohms
- Sensitivity: -72dB+/−4dB (0dB=1 V/micro bar, 1 kHz)
- Plug: 1/8" diameter plug

The frequency response curve provided with the specifications shows that the curve is flat at 0 dB in the frequency range 20 - 10000 Hz. This works well since heart sounds are in the range of 20 - 2000 Hz.

Charlton [Charlton93] recommends using an Electret Condenser Microphone, or ECM, as the sound sensor for recording low intensity sounds. Three reasons cited for this are:

- the ECM has a frequency response that is flat through the human hearing range
- the ECM has a high signal to noise ratio
- the ECM has an output impedance suitable for connecting directly to a high gain circuit

The Realistic Tie Tack Back Electret Condenser Microphone is small enough to be inserted into one of the rubber tubes con-
necting the stethoscope head to the earpiece. It can also be attached to either the earpiece or stethoscope head with rubber medical tubing.

2.1.4 Preamplifier

Barschdorff [Barschdorff91] noted that there was a need to preamplify the heart sounds in his study. Through experimentation in this study, it was found that the heart sounds could not be recorded adequately with the stethoscope, microphone, and 8 bit sound card. When using the 8 bit sound card, the soft heart sounds were indistinguishable from the noise generated by the sound card. The heart sounds could actually be heard through amplified speakers also attached to the sound card. From this, it was determined that the microphone being used was sensitive enough to record the sounds, but the sounds were too soft for the sound card to record in their original state. A preamplification stage was needed before the sound card to make the hearts sounds loud enough for the sound card to record.

Finding a suitable preamplifier proved challenging. Initially, a commercial preamplifier could not be found. Anthony
Charlton [Charlton93] described a preamplification circuit he designed. It appeared that this circuit with a slight modification, would provide the amplification needed for this application. The circuit is sold in kit form only and was purchased for about $15.

The Detector '700, as the circuit is known, has extremely high amplification. The voltage gain is about 47000 times the input. In the original design, the Detector '700 was equipped with a high pass filter to prevent the higher frequencies from being masked by lower frequencies. However, this application is interested in the lower frequency range, that is, below 2000 Hz. Therefore, the Detector '700 was modified to have a low pass filter instead of a high pass filter. This was accomplished by changing the values of two of the circuit’s capacitors.

The Detector '700 appears to be a well-designed high gain circuit. It has circuitry which is supposed to prevent unwanted oscillation, a common problem with high gain circuits. Ultrasonic energy is shunted to ground to prevent it from swamp-out audible sounds. The LM387N Low Noise Dual Preamplifier is also used in the circuit. One of the ampli-
fiers in this integrated circuit is used to form an automatic level control, while the other is used to amplify the microphone signal. The automatic level control of the Detector '700 has a feedback mechanism for loud signals which prevents output from exceeding a certain voltage value. This allows headphones to be used without danger of ear damage from very loud input. It is also battery powered and easily portable.

Although the Detector '700 appeared well-suited to this application, it did not perform its intended purpose. The gain of the circuit appeared to be too high, causing an oscillation effect. Attempts at modifying the circuit to reduce the gain were unsuccessful.

By the time the experimentation with the Detector '700 had ended, a fully assembled, affordably priced commercial preamplifier was found. It is produced by MCM Electronics and is called a Stereo Phono Preamp (Model # 40-630). This preamplifier is small enough to be portable, but it does require A/C power. It has the following technical specifications:

• Frequency Response: 30-20000 Hz
• Input Impedance: 50000 Ohms
• Input Voltage: 6 mV
Output Voltage: 500 mV
Voltage Gain: 83.33

Since the microphone is mono, it uses just one of the preamplifier’s two channels. This preamplifier works remarkably well with the 8 bit sound card. It virtually eliminated noise and produced clear heart sound patterns in the Wave files.

2.1.5 Sound Card

The sound card used was the Audioport by Media Vision. The Audioport can record sound at 8 bits/sample with a maximum sampling rate of 22.1 kHz. This sound card is readily portable, as it connects to the parallel printer port of a computer.

2.2 Software

2.2.1 Wave Files and the Wave Editor

The Wave file is a popular Windows file format used to record sound. It is a type of RIFF file. RIFF, or the Resource Interchange File Format, is the preferred format for multimedia files [Microsoft91]. Although Wave files are widely used,
most references do not discuss the file format. Microsoft Windows Multimedia Programmer's Workbook [Microsoft91] and a reference obtained via ftp from ftp.cwi.nl provided the best discussions about the wave file format. Another book published by Microsoft Press, Microsoft Windows Multimedia Programmer's Reference, is supposed to contain an in depth discussion of the Wave file format, but it is currently out of print.

The basic building block of a RIFF file is called a chunk [Microsoft91]. A Wave file consists of two types of chunks: a "RIFF" chunk and subchunks. The "RIFF" chunk is located at the beginning of the Wave file. It consists of the following fields:

• a chunk ID - four bytes which are the ASCII characters 'R', 'I', 'F', 'F'
• the second four bytes specify the size of the data field in the chunk
• a data field - the first four bytes of which are the ASCII characters 'W', 'A', 'V', 'E'

The "RIFF" data field consists of subchunks, after the four byte WAVE identifier. A subchunk consists of three fields also:
• a four byte identification code consisting of ASCII characters
• a four byte integer value specifying the size of the subchunk’s data
• a data field

There are two subchunks which are mandatory for Wave files: the format chunk and the data chunk. The identification codes for these chunks are ‘f’, ‘m’, ‘t’, ‘ ’ and ‘d’, ‘a’, ‘t’, ‘a’, respectively. There are several variations of Wave files which include other subchunks. According to the Wave specifications, the format subchunk must occur before the data subchunk. For this project, Microsoft’s PCM Wave format, the most popular Wave format, was used.

The only two subchunks of interest to this project are the format and data chunks. The format chunk contains information on how the wave file was recorded. The data field of the format chunk consists of the following C data structure:

```c
struct PCM_format
{
    WORD wFormatTag;        // Format category
    WORD wChannels;         // Number of channels
    DWORD dwSamplesPerSec;  // Sampling Rate
    DWORD dwAvgBytesPerSec; // For buffer estimation
    WORD wBlockAlign;       // Data block size
    WORD wBitsPerSample;    // Sample size
};
```
The variable wFormatTag is an integer value which indicates the type of Wave file. For PCM, this value is one. The variable wChannels is an integer value representing the number of input channels used during recording. One indicates mono and two indicates stereo. The sampling rate is stored as a long integer representing the value in Hertz in the variable dwSamplesPerSec. Playback software uses the next two variables, dwAvgBytesPerSec and wBlockAlign. The last member of the structure, wBitsPerSample, is an integer indicating the sample size used. The most popular sampling sizes are eight and sixteen bit.

The data chunk contains the digital sound data. This represents the amplitude and frequency information as a function of time. The number of bytes contained in the data field of the data chunk is stored as a long integer in the four bytes following the data chunk identifier. For extracting the data from the data chunk, it is necessary to know the values of wChannels and wBitsPerSample from the format chunk. The data representation and data packing schemes are different for the various values of these two variables. For eight bit data, each sample is represented as an unsigned byte, representing the decimal values 0 to 255. Sixteen bit data samples are
represented by a signed two byte integer, with a decimal range of -32,768 to 32767. Sixteen bit data samples are also stored in little endian format. Little endian format stores the low-order byte ahead of the high-order byte. This must be taken into account when reading sixteen bit data. If the file has been recorded using stereo input, the data sample from the left channel is stored ahead of the data sample from the right channel. This information is illustrated well in Microsoft Windows Multimedia Programmer’s Workbook [Microsoft91] and shown here in Figure 1.

The wave editor used to record the heart sounds was Cool Edit, which is a shareware program written by David Johnston. Cool Edit runs in Microsoft Windows 3.1 and provides a graphical user interface for recording and editing sound. Cool Edit can store recorded sound in Microsoft’s standard Windows audio file format, the PCM WAVE format, as well as other popular audio formats.

Cool Edit was used to record and edit the heart sounds. Dr. Edward Bayne, a pediatric cardiologist at University Medical Center of Jacksonville, Florida, advised that five complete heart cycles are usually required to make an accurate diagno-
sis. Following this guideline, each recording was edited to five complete heart cycles. The edited wave files begin at the start of a heart beat and end just before the beginning of the sixth consecutive beat.
DATA REPRESENTATION

<table>
<thead>
<tr>
<th>Data Format</th>
<th>Maximum Value</th>
<th>Minimum Value</th>
<th>Midpoint Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit PCM</td>
<td>255 (0xFF)</td>
<td>0</td>
<td>128 (0x80)</td>
</tr>
<tr>
<td>16-bit PCM</td>
<td>32,767 (0x7FFF)</td>
<td>-32,768 (0x8000)</td>
<td>0</td>
</tr>
</tbody>
</table>

DATA PACKING

8-bit mono PCM

<table>
<thead>
<tr>
<th>Sample 1</th>
<th>Sample 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 0 (left)</td>
<td>Channel 1 (right)</td>
</tr>
</tbody>
</table>

8-bit stereo PCM

<table>
<thead>
<tr>
<th>Sample 1</th>
<th>Sample 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 0 Low-Order Byte</td>
<td>Channel 1 High-Order Byte</td>
</tr>
<tr>
<td>Channel 0 High-Order Byte</td>
<td>Channel 1 Low-Order Byte</td>
</tr>
</tbody>
</table>

16-bit mono PCM

<table>
<thead>
<tr>
<th>Sample 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 0 (left) Low-Order Byte</td>
</tr>
<tr>
<td>Channel 1 High-Order Byte</td>
</tr>
</tbody>
</table>

16-bit stereo PCM

<table>
<thead>
<tr>
<th>Sample 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Channel 0 (left) Low-Order Byte</td>
</tr>
<tr>
<td>Channel 1 High-Order Byte</td>
</tr>
</tbody>
</table>

Figure 1: Data Representation & Packing in PCM Wave Files
2.2.2 FORM_GEN

FORM_GEN is a software program written by the author to support this project. It was written in Borland C++ 4.0 to run in a DOS environment. The source code for FORM_GEN can be found in Appendix A. The command line for this program is: FORM_GEN. The purpose of this program is to generate the demographic information questionnaires required for this project, in an ASCII file format. An example of the forms generated by FORM_GEN is shown in Figure 2. FORM_GEN is a menu driven application. The options available for this program are:

- Add a heart defect to its list
- Display the list of heart defects
- Generate demographic information forms
- Quit

FORM_GEN maintains a binary data file called "DEFECTS.DAT". This data file contains the following information: the name of the heart defect, a four character abbreviation for the defect, and the number of forms generated to date for this defect. The program will allow up to 100 heart defects to be entered into the file "DEFECTS.DAT" and up to 10,000 forms can be generated for each defect. Each form is given a unique
eight character identification code, which is also used as the first eight characters of the file name. The identification code is obtained by concatenating the four character abbreviation for the defect and current number for the defect, expressed using four digits. All the demographic information files are given a FRM extension.
<table>
<thead>
<tr>
<th>PATIENT DEMOGRAPHIC INFORMATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Patient ID: TEST0000</td>
</tr>
</tbody>
</table>

**Patient Past History**

- Was Patient Born Prematurely? (Y/N): ____
- History of Respiratory Infections? (Y/N): ____
  - Pneumonia? (Y/N): ____
  - Bronchialitis? (Y/N): ____
- Age When Murmur First Discovered (Years Months): ____ ____ (-1 -1 if N/A)

**Family History**

- Any Relatives With Heart Problems? (Y/N): ____
  - Other Immediate Family Member? (Y/N): ____
- Any Syndromes Associated With Heart Disease in the Family? (Y/N): ____

**Present Symptoms**

- Lack of Tolerance for Exercise? (Y/N): ____
- Unable to Keep Up with Age Group During Play? (Y/N): ____
- Cyanosis? (Y/N): ____
- Pallor? (Y/N): ____
- Fainting or Dizziness with exercise? (Y/N): ____

**Physical Exam**

- Below Average Height on Growth Chart? (Y/N): ____
- Below Average Weight on Growth Chart? (Y/N): ____
- High Blood Pressure? (Y/N): ____
- Any Syndrome Associated with Height? (Y/N): ____
- Presence of Other Congenital Malformations? (Y/N): ____
- Left Side of Chest More Prominent Than Right? (Y/N): ____
- Pulse Too Fast? (Y/N): ____
- Pulse Too Slow? (Y/N): ____
- Respiratory Rate Too Fast? (Y/N): ____

---

Figure 2: Sample Demographic Information Form
2.2.3 DEM_FILE

DEM_FILE is a program written by the author to support this project. It was written in Borland C++ 4.0 and compiled to run in DOS. The source code for this program is listed in Appendix B.

The command line for this program is simply: DEM_FILE.

DEM_FILE will allow the user to interactively input the information obtained from the questionnaires created using the FORM_GEN program. DEM_FILE uses the ASCII file GOTO.PTS, which contains the coordinate pairs used for screen display.

The output of DEM_FILE is an ASCII file containing 28 signed integer values. These values represent the answers on the questionnaire. The output file will have an extension of DEM. The first eight characters of the output file name will be the unique identifier generated on the questionnaire by FORM_GEN.

The values in the output file will be used as input to the neural network, along with information obtained from the Wave files.
2.2.4 DIRECTRY

DIRECTRY.BAT is a DOS batch program written by the author to support this project. The source code for DIRECTRY.BAT can be found in Appendix C.

The command line for this program is: DIRECTRY <PATIENT_ID> <PATH>. This program will create a directory structure to standardize the file names of the recorded heart sounds. The directory structure is based on the two command line arguments. PATIENT_ID is the identifier created for the patient by FORM_GEN and PATH is where the directory structure is to be placed. In PATH, DIRECTRY will create a directory called <PATIENT_ID>. DIRECTRY will also create nine subdirectories, one for each of the nine auscultation positions discussed in Section 1.1, under <PATH>\<PATIENT_ID>. The names of the subdirectories are:

- UPRT_STN.BDR - for auscultation at the upper right sternal border
- UPLF_STN.BDR - for auscultation at the upper left sternal border
- LOLF_STN.BDR - for auscultation at the lower left sternal border
• 3RD_SPCE – for auscultation at the third space
• APEX – for auscultation at the apex
• LFMIDAXL.LIN – for auscultation at the left midaxillary line
• RTMIDAXL.LIN – for auscultation at the right midaxillary line
• LFMIDPT.BCK – for auscultation at the left midpoint of the back
• RTMIDPT.BCK – for auscultation at the right midpoint of the back

In each of these nine subdirectories, two zero length files are created, BELL.WAV and DIAPHRGM.WAV. This was done to avoid naming inconsistencies if the recordings are done by more than one person. DIRECTRY for each patient before starting to record the patient’s heart sounds.

2.2.5 WAVE

WAVE is a program written by the author to support this project. It was written in Borland C++ 4.0 and compiled to run in DOS. The source code for this program is listed in Appendix D.
The command line for this program is: WAVE <WAVE_FILE>. The purpose of this program is to extract information from the format and data chunks of the specified Wave file and represent the data in a form which the neural network can use. This is implemented in code using a C++ class. The class performs the following tasks on the Wave file:

- reads the RIFF chunk to confirm that the file is a RIFF WAVE file
- reads the format chunk to confirm that it is a PCM Wave file and extract the useful format information
- reads the data chunk and creates two data files - one binary data file and one ASCII file which contains offset/value pairs
- uses the ASCII data file and the demographic information file associated with the Wave file to create a training file and a test file for the neural network

Using all the data values in the Wave data chunk as input to the neural network is impractical. At a sampling rate of 22.1 kHz, the short heart sound recordings generated sixty to one hundred thousand samples each. Ignoring zero data values was tried to reduced the amount of data, but this too yielded far too much data too be useful as input to the neural network.
Representing the local maximum and minimum values of the data, or the "spikes", proved to be a suitable choice. These data values, along with their offsets from the start of the data, were used as input to the neural network with the patient demographic information. Each offset/value pair is appended to the twenty-eight values of the associated demographic information file to form a complete input vector for the neural network. Each Wave file is represented by a set of these vectors. The number of vectors in the set varies from recording to recording.

The training and test files created by WAVE both have the same file name as the command line argument. The training file has an extension of TRN and the test file has an extension of TST. The training file consists of a set of input/output vector pairs, with the output vector equal to its corresponding input vector. A test file consists only of the set of input vectors.

2.2.6 Neural Network

The neural network used for this project was written by Dr. Layne Wallace of the University of North Florida. It was written in the C programming language. Backpropagation is the
training paradigm used for this three layer neural network. The input and output layers consist of thirty neurons each and the middle layer contains fifteen neurons.

2.3 Patient Information Necessary for Diagnosis

According to Esogbue and Stein [Esogbue79], the information a physician needs to make a diagnosis comes from four basic categories, which are described in Section 1.3. The actual information used for each category varies among disorders. Dr. Edward Bayne, advised us on the information pediatric cardiologists use to diagnose heart murmurs in children aged four through ten. This information was used to create a demographic information questionnaire which is produced by the FORM_GEN program.

2.3.1 Patient Past History

When examining a patient's past history, a pediatric cardiologist is mainly concerned with two things: prematurity and respiratory infections. Children that were born prematurely have a higher incidence of heart problems than those that were
full-term. A history of respiratory infections, particularly pneumonia and bronchialitis, also may be an indication of heart problems.

Family history also plays an important part in the cardiologist’s diagnosis. Many heart disorders and defects are congenital. If a relative has a heart problem, the child has a higher risk of cardiac disease. Also, the closer the relative is, the higher the risk. An interesting fact is that a child that has a mother with cardiac disease is twice as likely to develop a heart problem as a child with a father with cardiac disease. Cardiologists are also interested in any disorders or syndromes that run in the family that are associated with heart disease. For example, more than 40% of the children with Down’s Syndrome have heart problems. Children of mothers with Marfan Syndrome are more likely to develop heart disease than other children.

When a heart murmur was first discovered is also significant to the cardiologist. Generally, the earlier a murmur is first heard, the greater the risk of congenital heart disease. Jane Newburger [Newburger92] reported that a murmur first heard at six months of age has a 1:7 risk of being congenital heart
disease, but a murmur first heard at twelve months of age has only a 1:50 risk.

2.3.2 Present Symptoms

There are several symptoms which may indicate a heart problem. One is a lack of tolerance for exercise. If a child is unable to keep up with his or her age group during play, a cardiac problem may be the reason. Another symptom is cyanosis. This is a blue coloration of the skin, caused by a lack of oxygen to the area. Pallor, a condition in which the skin takes on a white color, is another indication. Pallor is caused by insufficient blood flow. Physicians also look for fainting or dizziness with exercise.

2.3.3 Physical Examination

Upon physical examination, the child’s height, weight, and blood pressure are measured. The values for the height and weight are plotted on a growth chart, which allows the physician to compare the child to a “normal” child of the same age. High blood pressure is often associated with heart disease.
Cardiac problems can also cause a child to be significantly below the curve on the height and weight charts. Other indications of possible heart problems which are observable in the physical exam include:

• any syndromes associated with height
• the presence or absence of other congenital malformations
• the left side of the chest being more prominent than the right side - signifying an enlarged heart
• a pulse rate which is too fast or too slow
• a respiratory rate which is too fast

Also during the physical examination, cardiac auscultation will be performed.

2.3.4 Clinical and Diagnostic Tests

Diagnostic tests which cardiologists use include the electrocardiogram, the chest X-ray, the echocardiogram, and the cardiac catheterization. The electrocardiogram (also known as the ECG or EKG) is used to monitor the electrical activity of the heart. The chest X-ray allows the cardiologist to determine if the heart is enlarged. The echocardiogram utilizes sonogram technology to allow the cardiologist to exam the interior workings of the heart. This procedure is noninvasive and is
used most often to validate an initial diagnosis. The cardiac
catheterization is an invasive procedure that examines the in-
terior of the heart. It is used most often before surgery or
when the echocardiogram does not provide enough information.

2.4 Data Collection

Heart sound data was collected from subjects using the con-
figuration described in Section 2.1 to provide proof of con-
cept information for purposes of this thesis. Auscultation was
performed by the author. A demographic information question-
naire, such as the one shown in Figure 2, was also completed
for each subject.

2.4.1 Wave Sampling Rate

Cardiac sounds have a frequency range of 20 to 2000 Hz. The
Nyquist Theorem states that a waveform can be faithfully re-
produced only if the sample rate is at least twice as high as
the frequency of the highest component. Higher sampling rates
produce recordings which better approximate the original
sound. The trade-off is that higher sampling rates also pro-
duce larger files. Therefore, according to the Nyquist The-
orem, the sampling rate for recording auscultation should be at least 4 Hz. However, to obtain a quality recording, it was found through experimentation that a sampling rate of 22.1 kHz worked well. This was the sampling rate used to make the eight-bit heart sound recordings for this thesis.

2.5 Project Evaluation

For this thesis, the neural network was trained as an autoassociative network. That is, the neural network associates an input pattern with itself. The information represented by an input pattern is shown in Table 1. This information comes from two sources. The first twenty-eight items are taken from the patient questionnaires and described in Section 2.3, while the final two items represent heart sound information from a Wave file. The data model used in this thesis represents the heart sounds by considering the local maximum and minimum values, or “spikes”, in the data chunk of a Wave file. In the input pattern, the “spikes” are represented by their position in the data chunk relative to the beginning of the data chunk, and their actual value in the data chunk. One heart sound recording will correspond to a set of input patterns, with the
size of the set equal to the number of “spikes” in the data chunk of the Wave file.

<table>
<thead>
<tr>
<th>Pattern Position</th>
<th>Information Represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Patient’s Sex</td>
</tr>
<tr>
<td>2</td>
<td>Patient’s Age in Years</td>
</tr>
<tr>
<td>3</td>
<td>Patient Born Prematurely</td>
</tr>
<tr>
<td>4</td>
<td>History of Respiratory Infections</td>
</tr>
<tr>
<td>5</td>
<td>History of Pneumonia</td>
</tr>
<tr>
<td>6</td>
<td>History of Bronchialitis</td>
</tr>
<tr>
<td>7</td>
<td>Age Murmur Discovered (years part)</td>
</tr>
<tr>
<td>8</td>
<td>Age Murmur Discovered (months part)</td>
</tr>
<tr>
<td>9</td>
<td>Relatives with Heart Problems</td>
</tr>
<tr>
<td>10</td>
<td>Mother</td>
</tr>
<tr>
<td>11</td>
<td>Father</td>
</tr>
<tr>
<td>12</td>
<td>Sibling</td>
</tr>
<tr>
<td>13</td>
<td>Other Immediate Family Member</td>
</tr>
<tr>
<td>14</td>
<td>Syndromes Associated w/ Heart in Family</td>
</tr>
<tr>
<td>15</td>
<td>Lack of Tolerance for Exercise</td>
</tr>
<tr>
<td>16</td>
<td>Unable to Keep Up with Age Group at Play</td>
</tr>
<tr>
<td>17</td>
<td>Cyanosis</td>
</tr>
<tr>
<td>18</td>
<td>Pallor</td>
</tr>
</tbody>
</table>

Table 1: Information Represented by Input Pattern
<table>
<thead>
<tr>
<th>Pattern Position</th>
<th>Information Represented</th>
</tr>
</thead>
<tbody>
<tr>
<td>19</td>
<td>Fainting or Dizziness with Exercise</td>
</tr>
<tr>
<td>20</td>
<td>Below Average Height</td>
</tr>
<tr>
<td>21</td>
<td>Below Average Weight</td>
</tr>
<tr>
<td>22</td>
<td>High Blood Pressure</td>
</tr>
<tr>
<td>23</td>
<td>Syndromes Associated with Height</td>
</tr>
<tr>
<td>24</td>
<td>Presence of Congenital Malformations</td>
</tr>
<tr>
<td>25</td>
<td>Left Side of Chest More Prominent</td>
</tr>
<tr>
<td>26</td>
<td>Pulse Too Fast</td>
</tr>
<tr>
<td>27</td>
<td>Pulse Too Slow</td>
</tr>
<tr>
<td>28</td>
<td>Respiratory Rate Too Fast</td>
</tr>
<tr>
<td>29</td>
<td>Offset from Beginning of Data Chunk</td>
</tr>
<tr>
<td>30</td>
<td>Data Value from Data Chunk</td>
</tr>
</tbody>
</table>

Table 1: Information Represented by Input Pattern

The neural network operates in two modes: training and testing. The backpropagation training is controlled by the total sum of the squares of the difference error. When this value becomes less than 0.1, training is considered complete. For testing, an output error vector of size thirty is calculated for each input pattern. If the value at each position of the output error vector is less than 0.01, the output pattern was considered to be a successful match of the input pattern.
Since each heart sound recording corresponds to a set of input patterns, the percentage of the set that was successfully matched is calculated and used to determine the success of the neural network.

The success of the neural network was also compared to the success of a standard statistical approach on the same data set. For this thesis, a linear regression analysis was chosen as the statistical method for comparison.
The make up of the subjects, along with the assigned Patient IDs, was as follows:

- one twenty-seven year old male - TEST0000
- one twenty-seven year old female - TEST0001
- one twenty-four year old male - TEST0005
- one twenty-two year old female - TEST0004
- one sixteen year old female - TEST0006
- one fifteen year old male - TEST0003
- one five year old female - TEST0002

The average age of the subjects was 19.43 years. All subjects had no history of heart problems, except for the five year old female. She was born with an AV Canal defect which was surgically repaired at ten months of age. The auscultation recordings of these subjects were made by the author. Auscultation was performed using the stethoscope's diaphragm. The stethoscope was positioned to the patient's left of the sternum at its approximate midpoint. Twelve recordings of
five consecutive heart cycles were made for each subject. Of these twelve, ten were used for training the neural network and two were reserved for testing the neural net once training was complete. The FORM_GEN program was run to generate demographic information forms for this group. The identifier "TEST" was used, which generated the IDs TEST0000 through TEST0006. The questionnaires were filled out as accurately as possible without the assistance of a medical expert.

Eighty-four training and testing files were created for the neural network by running the WAVE program on each of the recordings. The first ten training files from each patient were concatenated together to form the training file for the neural network. The remaining files would be used to test the generalization capabilities of the neural net.

The objective of the training was to have the neural network train in an autoassociative manner using a backpropagation algorithm. The training set consisted of 14,029 patterns. The remaining 3018 patterns were reserved for testing the trained neural net.

The success of the trained neural network was determined by the percentage of input patterns that were correctly matched.
The criteria for a successful match is discussed in Section 2.5. For each test file, which consisted of a variable number of patterns, the number of successful matches and the total number of input vectors were counted. The ratio of those two values determined the success rate for the file. The neural network was able to successfully match 76.20% of the patterns on which it trained and 74.22% of the patterns which it had not seen before. The results of testing are shown in Table 2 and Table 3. The test results that were obtained from the patterns on which the neural net trained on and those which were reserved for testing were very similar.
A linear regression analysis was performed on this data as a comparison using the Statistical Analysis System (SAS). This method derived a linear equation for the thirtieth parameter, which is the data value taken from the Wave file, in terms of the other twenty-nine input parameters. The analysis was able
to eliminate many of the variables from the equation because they were linear combinations of other variables. The resulting equation was:

\[ v_{30} = -2.461583 + (3.477307 \times v_1) + (-0.151621 \times v_2) + (-7.709763 \times v_4) + (-2.162370 \times v_9) + (1.501670 \times v_{11}) + (0.824143 \times v_{29}) \]

For this analysis, the twenty-ninth variable, \( v_{29} \), was divided by 70,000. The variables which this analysis found to be linearly independent are:

- \( v_1 \) - the patient’s sex
- \( v_2 \) - the patient’s age in years
- \( v_4 \) - a history of respiratory infections
- \( v_9 \) - any relatives with heart problems
- \( v_{11} \) - a father with heart problems
- \( v_{29} \) - the offset from the beginning of the Wave data

The linear regression technique was unable to match any of the data successfully. Sample output from the linear regression test is shown in Table 4.
<table>
<thead>
<tr>
<th>Expected Result</th>
<th>Linear Regression Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>-125</td>
<td>0.968057</td>
</tr>
<tr>
<td>4</td>
<td>0.971377</td>
</tr>
<tr>
<td>0</td>
<td>0.972413</td>
</tr>
<tr>
<td>1</td>
<td>-0.066512</td>
</tr>
<tr>
<td>1</td>
<td>-0.066454</td>
</tr>
<tr>
<td>1</td>
<td>-0.066348</td>
</tr>
</tbody>
</table>

Table 4: Test Results for Linear Regression
Chapter 4

DISCUSSION

This thesis has shown that cardiac auscultation can be performed using a portable computer system, which uses readily available digital audio technology. A method for modeling heart sound data and required patient information for use with a neural network was also developed. This model was used to train and test a backpropagation neural network in an autoassociative manner. Encouraging results were obtained from the testing of the neural network.

The electret condenser microphone (ECM) used in this study appears to be better suited for cardiac auscultation than the human ear. In the frequency range where most significant heart sounds occur, the efficiency of human hearing declines. Human hearing perceives higher frequency sounds as faint or absent when both low and higher frequency sounds are present. The ECM does not have these problems. It has a flat frequency response throughout the frequency range of heart sounds. An-
other quality which makes it a good sound sensor is that it has a high signal to noise ratio.

The analog heart sounds were converted to digital files using an eight-bit sound card. It was found that the ECM and the eight-bit sound card alone were unable to record the heart sounds. When recordings were attempted, the result was just noise. With amplified speakers attached to the sound card, however, the heart beat could clearly be heard. From this, it was assumed that the problem was in the sound card, which was limited by the eight-bit data representation. An amplification stage was needed between the microphone and the sound card. A commercial preamplifier with a voltage gain of 83.33 was found to work well with this setup. Experimentation with a preamplifier with a much higher voltage gain was unsuccessful.

Some recordings were made using a sixteen-bit sound card and an ECM. The sixteen-bit sound card was able to record distinguishable heart sounds without the use of a preamplifier. The most likely reason why the sixteen bit sound card was able to record sounds that the eight-bit sound card couldn’t is the difference in data representation. The sixteen-bit card can
represent sound on a scale which is two hundred and fifty-six times finer than the eight-bit card. The trade-off between the two cards is that the sixteen-bit card requires twice as much space to store the data. However, the increase in recording quality is much greater than the increase in output file size.

The heart sounds were stored in Wave files. The Wave file is a popular Microsoft Windows audio file format. It was found that there are several types of Wave files. Microsoft’s PCM Wave file format was chosen for this study. A Wave file is composed of “chunks” of information. The PCM format can contain several optional chunks. However, there are two chunks which must be included. They are the format chunk and the data chunk. The format chunk contains information about how the sound was recorded and the data chunk contains the actual data values associated with the sound.

Wave files can get quite large. The size of a Wave file is a function of the length of the recording, the sampling rate, and the data size (eight- or sixteen-bit). The sampling rate can be varied in the Wave editor. Again there is a trade-off with the sample rate between the size of Wave file and the quality of the recording. Higher sample rates better approx-
imate the original sound, but they also create larger data chunks. Through experimentation it was found that a sample rate of 22.1 kHz gave the best results.

Although Wave files are a convenient way to store sound digitally, they contain too much information to be used directly as input to a neural network. Each Wave file recorded for this study was edited to contain exactly five heart beats. These five heart beats were represented by more than seventy thousand bytes of digital information. A method of representing the information in a more concise manner needed to be developed. The method developed for this study represented heart sound data as offset/value pairs. A program was written to extract information from the Wave file’s format and data chunks and create the new representation. The program, called WAVE, extracts the local maximums and minimums from the data chunk. It records the offset from the beginning of the data chunk and the actual data value in the new representation. This results in a much more manageable representation of the heart sounds.

One problem with this representation is that the number of offset/value pairs will vary from one recording to another.
The question of how the data was going to be presented as input to the neural network needed to be resolved. The solution was to append each offset/value pair to the twenty-eight values of the associated demographic information file. This formed a set of input vectors for each recording. For this study, the backpropagation neural network was trained to output the input vector.

The backpropagation training was fairly successful. After training, the neural network was able to successfully match 74.22% of the input vectors which were reserved for testing. This proved to be definitively better than the results of linear regression, a standard statistical approach, which failed to match any input.

A literature review has shown that very little work has been done involving neural networks and heart sound analysis. Only one paper [Barschdorff91] has been found which involved research in both areas. This study [Barschdorff91] also used input from other sensors such as the ECG. This study has shown that heart sound analysis using a backpropagation neural network is possible. In addition, this study was completed using readily available commercial parts which were relatively in-
expensive. The setup used is also easily portable. The only limitation on the portability is that the preamplifier needs an A/C power source.

When fully developed, this portable system could have a significant impact on reducing the time required to diagnose a heart problem. The diagnostic ability of cardiologists could be extended to other physicians with this system. It is expected that the greatest impact would be in rural areas, where there are few cardiologists. Family physicians would be able to make an initial diagnosis without the need for expensive, specialized equipment. A course of action can then be planned with a better understanding of the patient's condition. In addition to being informed of the neural network's diagnosis, a cardiologist at a distant location could also have the heart sound recording available to him through the Internet. The standard Wave format was chosen with this in mind. This format is compatible with most sound cards available for personal computers.

It is anticipated that in the future, this study will be extended to determine if a neural network can classify heart murmurs in pediatric subjects using the configuration in this
thesis. It is expected that the data will be collected from patients at the University Medical Center and/or Nemours Children’s Clinic under the supervision of Dr. Bayne. The patient information will be collected in the form of questionnaires produced by the FORM_GEN program. Heart sound data, in the form of Wave files, will be collected from children aged four to ten with one of the following:

- Still’s Murmur
- aortic valve stenosis
- subaortic stenosis
- no heart murmur

Auscultation will be performed as described in Section 1.1. The DIRECTRY program will be used to enforce a consistent naming convention for the Wave files. The diagnosis of each patient will be confirmed by echocardiogram.

In future work, it is recommended that a sixteen-bit sound card be used to more accurately record the heart sounds. This will double the size of the Wave recordings, but the increased fidelity of the recorded sound justifies this. The WAVE program can extract information from both eight- and sixteen-bit Wave files, so there will be no change required to process the
Wave files. This thesis used the Audioprobe was chosen because of its portability. At the time of this work, a portable sixteen-bit sound card was not available. It is expected that one will be available for future research. With the current setup, switching the sound cards should not require any other changes.

The statistical method of linear regression was shown to be ill-suited for this type of analysis. In future studies, the results of the neural network testing should be compared to a different statistical approach, such as nonlinear or curvilinear regression.

In conclusion, this study has shown that cardiac auscultation can be performed using current digital audio technology and that a backpropagation neural network can be trained to recognize heart sound data. These results support the suggestion that further research in this area is warranted.
References

[Ali93]

[Artis92]

[Barschdorff91]

[Baxt90]

[Bayne88]

[Bortolan91]

[Bortolan92]
[Charlton93]

[Chi92]

[Chow92]

[Cianflone91]

[Clarke93]

[Dassen89]

[Dassen92A]

[Dassen92B]
Wave files. This thesis used the Audioport was chosen because of its portability. At the time of this work, a portable sixteen-bit sound card was not available. It is expected that one will be available for future research. With the current setup, switching the sound cards should not require any other changes.

The statistical method of linear regression was shown to be ill-suited for this type of analysis. In future studies, the results of the neural network testing should be compared to a different statistical approach, such as nonlinear or curvilinear regression.

In conclusion, this study has shown that cardiac auscultation can be performed using current digital audio technology and that a backpropagation neural network can be trained to recognize heart sound data. These results support the suggestion that further research in this area is warranted.
[Habboush92]

[Heden93]

[Held92]

[Herbert93]

[Hernandez93]

[Horner92]

[Leao93]

[Leffingwell93]
[Levey90]

[Locati-Heilbron93]

[Merri89]

[Micheli-Tzanakou93]

[Microsoft91]

[Morabito92]

[Murray94]

[Nadal93]
[Newburger92]

[Reddy92]

[Reggia93]

[Rodriguez93]

[Schultz91]

[Sethi93]

[Shen93]

[Silipo93]
[Spitzer93]

[Stamkopoulous92]

[Stein93A]

Stein93B]

[Strand91]

[Suzuki92]

[Thompson83]

[Thompson93]
[Vondrak94]

[Wynbrandt91]
APPENDIX A

FORM_GEN Code Listings

/*

Written By: Erik Hudson

To Support Master's Thesis Work
"A Portable Computer System For Recording Heart
Sounds And Data Modeling Using A Backpropagation
Neural Network"

University of North Florida

Adviser: Dr. Layne Wallace

File: FORM_GEN.CPP

Written In: Borland C++ 4.0 to Run in DOS

Purpose: Generates the required demographic
information forms required for the thesis project.
Maintains a count of how many of each form has
been generated. Unique patient IDs are generated
using the counts and identifiers which are
associated with the different heart problems.

*/

// header files
#include <iostream.h>
#include <fstream.h>
#include <string.h>
#include <conio.h>
#include <ctype.h>
#include <stdio.h>
#include <dos.h>
#include <ctype.h>
#include <stdio.h>
#include <dos.h>

cost int MAX_NUM_OF_DEFECTS = 100;
// structure for keeping track of counts
struct count {
    char heart_defect[80];
    char abbreviation[5];
    int count;
} x[MAX_NUM_OF_DEFECTS];

int num_of_defects = 0;

// function prototypes
void create_file(const char *id);
void add_defect(void);
void list_defects(void);
void menu(void);
int check(char *);
void file_menu(void);
void display_list(void);

void main(void)
{
    char c; // used to hold menu choice

    // input defect data
    FILE * ifp, *ofp;

    ifp = fopen("DEFECTS.DAT","rb");

    while (fread(&x[num_of_defects], sizeof(struct count),
                 1,ifp) == 1)
    {
        ++num_of_defects;
    }

    fclose(ifp);

textcolor(YELLOW);
textbackground(BLUE);

    menu(); // display main menu for the program
while (1)
{
    gotoxy(32,20);

    cin >> c;

    c = toupper(c);

    if (c == 'A')
    {
        add_defect();
        continue;
    }
    else if (c == 'B')
    {
        display_list();
        continue;
    }
    else if (c == 'C')
    {
        file_menu();
    }
    else if (c == 'Q')
    {
        clrscr();
        cout << "\n\n*** PROGRAM TERMINATED ***\n\n";
        break;
    }
    else
    {
        cout << "\a";
    }
}
} // end while

ofp = fopen("defects.dat","wb");
for (int i = 0; i < num_of_defects; ++i)
{
    fwrite(&x[i], sizeof(struct count),1,ofp);
}
fclose(ofp);

} // end main()
// This function will display the main menu

void menu(void)
{
    clrscr();

gotoxy(24,1);
cout << "FORM GENERATOR PROGRAM MAIN MENU";

gotoxy(10,5);
cout << "A. Add A New Heart Defect To List";

gotoxy(10,8);
cout << "B. Display List Of Heart Defects";

gotoxy(10,11);
cout << "C. Generate Patient Information Forms";

gotoxy(10,14);
cout << "Q. Quit This Program";

gotoxy(20,20);
cout << "Selection: ";

} // end menu()

void file_menu(void)
{
    char defect[80];

    char fname[13];

    char abbr[5];

    int number, i,start,last;

    clrscr();

gotoxy(24,1);
cout << "FORM GENERATOR PROGRAM CREATE MENU"

gotoxy(10,5);
cout << "Name Of Defect: ";

gotoxy(10,10);
cout << "Number Of Forms Needed: ";

gotoxy(27,5);
gets(defect);

gotoxy(35,10);
cin >> number;

for (i = 0; i < num_of_defects; ++i)
{
    if (strcmpi(x[i].heart_defect, defect) == 0)
    {
        strcpy(abbr, x[i].abbreviation);
        start = x[i].count;
        x[i].count += number;
        last = x[i].count;
        break;
    }
}

for (i = start; i < last; ++i)
{
    sprintf(fname,"%s%04d", abbr, i);
    create_file(fname);
}

} // end for loop

gotoxy(10,20);
cout << "Done Creating Files ...";

sleep(2);

clrscr();
menu();

} // end file_menu()
void add_defect(void)
{
    FILE *ofp;

    ofp = fopen("DEFECTS.DAT","a+b");

    char c;

    clrscr();

gotoxy(28,1);
cout << "Add A Defect To The List";

gotoxy(5,5);
cout << "Defect Name: ";

gotoxy(5,10);
cout << "Create A 4 Character ID String For Defect: ";

gotoxy(5,15);
cout << "Add It? (Y/N): ";

gotoxy(5,20);
cout << "Add Another? (Y/N): ";

while (1)
{
    gotoxy(19,5);
    gets(x[num_of_defects].heart_defect);

    do {
        gotoxy(49,10);
        cin >> x[num_of_defects].abbreviation;
    } while (!check(x[num_of_defects].abbreviation));

    x[num_of_defects].count = 0;

    do {
        gotoxy(21,15);
        cin >> c;
        c = toupper(c);
    } while ( (c != 'Y') && (c != 'N'));

    if ( c == 'Y') // write it to file, increment count
    {

fwrite(&x[num_of_defects], sizeof(struct count), 1, ofp);
++num_of_defects;
}

do {
    gotoxy(26,20);
    cin >> c;
    c = toupper(c);
} while ((c != 'Y') && (c != 'N'));

if (c == 'Y')
    continue;
else
    break;

} // end while

fclose(ofp);
clrscr();

menu(); // return to main menu

} // end add_defect()

// check to see if abbreviation has already been used
int check(char *abbreviation)
{
    int i;

    for (i = 0; i < num_of_defects; ++i)
    {
        if (strcmpi(abbreviation, x[i].abbreviation) == 0)
            return (0); // can't use abbreviation
    } // end for

    return (1);
} // end check()
void create_file(const char *id) {
    char fname[13];
    strcpy(fname, id);
    strcat(fname, " .frm ");
    ofstream outfile(fname); // create form file
    outfile << "\t\t\tPATIENT DEMOGRAPHIC INFORMATION\n\n";
    outfile << "Patient ID: " << id;
    outfile << "\t \t Sex (M/F): ___\n";
    outfile << "\t \t Age in Years: ___\n";
    outfile << "\n\nPatient Past History\n\n";
    outfile << "\t \t Was Patient Born Prematurely? (Y/N): ____\n";
    outfile << "\n\t \t History of Respiratory Infections? (Y/N): ____\n";
    outfile << "\t \t Pneumonia? (Y/N): ____\n";
    outfile << "\t \t Bronchialitis? (Y/N): ____\n";
    outfile << "\n\t \t Age When Murmur First Discovered (Years Months): ___ ___\n";
    outfile << "\n\n\t Family History\n\n";
    outfile << "\t \t Any Relatives With Heart Problems? (Y/N): ____\n";
    outfile << "\t \t Mother? (Y/N): ____\n";
    outfile << " Father? (Y/N): ____\n";
    outfile << " Sibling? (Y/N): ____\n";
Other Immediate Family Member? (Y/N): ___

Any Syndromes Associated With Heart Disease in the Family? (Y/N): ___

Present Symptoms

Lack of Tolerance for Exercise? (Y/N): ___

Unable to Keep Up with Age Group During Play? (Y/N): ___

Cyanosis? (Y/N): ___

Pallor? (Y/N): ___

Fainting or Dizziness with exercise? (Y/N): ___

Physical Exam

Below Average Height on Growth Chart? (Y/N): ___

Below Average Weight on Growth Chart? (Y/N): ___

High Blood Pressure? (Y/N): ___

Any Syndrome Associated with Height? (Y/N): ___

Presence of Other Congenital Malformations? (Y/N): ___

Left Side of Chest More Prominent Than Right? (Y/N): ___

Pulse Too Fast? (Y/N): ___

Pulse Too Slow? (Y/N): ___
outfile <<
    "\n\tRespiratory Rate Too Fast? (Y/N): _____\n";

outfile.close(); // close form file

} // end create_file()

void display_list(void)
{
    int i, j;
    clrscr();
    gotoxy(1, 2);
    cout << "DEFECT";
    gotoxy(50, 2);
    cout << "ABBREVIATION";
    gotoxy(65, 2);
    cout << "COUNT";
    for (i = 0; i < num_of_defects; ++i)
    {
        j = i + 3;
        cout << endl << x[i].heart_defect;
        gotoxy(50, j);
        cout << x[i].abbreviation;
        gotoxy(65, j);
        cout << x[i].count;
    }
    cout << endl << endl;
    cout << "\t\tPress Any Key ...";
    getchar();
    clrscr();

    - 78 -
menu();
}  // end display_list()
APPENDIX B

DEM_FILE Code Listings

/*

Written By:  Erik Hudson

To Support Master's Thesis Work
"A Portable Computer System For Recording Heart
Sounds And Data Modeling Using A Backpropagation
Neural Network"

University of North Florida

Adviser:  Dr. Layne Wallace

File:  DEM_FILE.CPP

Written In:  Borland C++ 4.0 to Run in DOS

Purpose:  This program will receive patient
demographic input interactively and create a file
(*.dem) which will be used as input to the neural
network along with info from the Wave files

*/

// Include the appropriate header files

#include <iostream.h>
#include <fstream.h>
#include <conio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
#include <iomanip.h>

// Function Prototypes
void last_screen(void);
void first_screen(void);
void second_screen(void);
void third_screen(void);
void fourth_screen(void);
void fifth_screen(void);

const TRUE = 1;
const FALSE = -1;
const num_of_questions = 30;

void main(void)
{
    char fname[13]; // used to hold name of output file

    int question_num = 0; // keeps track of which question
    // is current

    int done = 0; // determines if another patient's info
    // will be entered

    int age; // age of the patient

    char answers[num_of_questions]; // will hold the input
    // of each question

    int points[num_of_questions][2]; // holds points at
    // which to get input

    int years, months; // used in question 8

    ifstream infile("goto.pts"); // open file with the goto
    // pts

    if (!infile) // goto.pts didn't open
    {
        cerr << "\nERROR: Could not open file GOTO.PTS\n\n";
        exit (1); // quit program
    }

    for (int i = 0; i < num_of_questions; ++i) // read in
    // the coords
    {
        infile >> points[i][0];
        infile >> points[i][1];
    }
// set the screen colors
textcolor(YELLOW);
textbackground(BLUE);

clrscr(); // clear the screen

while (!done)
{
    if (question_num == 0) // first question, filename
    {
        clrscr();
        first_screen(); // display screen
        gotoxy(points[0][0],points[0][1]);
        cin >> fname;
        ++question_num;
        continue;
    }

    else if (question_num == 1) // second question, sex
    {
        gotoxy(points[1][0],points[1][1]);
        cin >> answers[1];

        answers[1] = toupper(answers[1]);

        if (answers[1] == 'B') // backup
        {
            --question_num; // go back to previous question
            continue;
        }

        if ( (answers[1] != 'M') && (answers[1] != 'F'))
        // invalid sex
        {
            cout << "\a";
            continue;
        }

        ++question_num;
        continue;
    }

// end if second question

else if (question_num == 2) // third question, age
{
    gotoxy(points[2][0],points[2][1]);
cin >> age;

    if (age < 0) // backup key
    {
        --question_num; // go back to previous question
        continue;
    }

    if (age > 120) // invalid age
    {
        cout << "\a";
        continue;
    }

    ++question_num;
    continue;
} // end if third question

if (question_num == 3)
{
    clrscr();
    second_screen(); // display screen

    gotoxy(points[question_num][0],
            points[question_num][1]);
cin >> answers[question_num];

    answers[question_num] =
        toupper(answers[question_num]);

    if (answers[question_num] == 'B') // backup
    {
        --question_num; // go back to previous question
        clrscr();
        first_screen(); // redisplay first screen
        continue;
    }
if ( (answers[question_num] != 'Y') &&
       (answers[question_num] != 'N')) // only yes
    // or no
{
    cout << "\a";
    continue;
}
++question_num;
continue;
} // end fourth question

if (question_num == 7)
{
    gotoxy(points[question_num][0],
           points[question_num][1]);
    cin >> years >> months;
    if (((years < 0) || (months < 0)) // backup
        
    {
        --question_num; // go back to previous question
        continue;
    }
    if (((years > 120) || (months > 11))
        
    {
        cout << "\a";
        continue;
    }
++question_num;
continue;
} // end eigth question

if (question_num == 8)
{
    clrscr();
    third_screen(); // display screen
    gotoxy(points[question_num][0],
           points[question_num][1]);
    cin >> answers[question_num];
answers[question_num] =
    toupper(answers[question_num]);

if (answers[question_num] == 'B') // backup
{
    --question_num; // go back to previous question
    clrscr();
    second_screen(); // redisplay first screen
    continue;
}

if ((answers[question_num] != 'Y') &&
    (answers[question_num] != 'N')) // only yes
    // or no
{
    cout << "\a";
    continue;
}

++question_num;
continue;
} // end ninth question

if (question_num == 14)
{
    clrscr();
    fourth_screen(); // display screen

go_toxy(points[question_num][0],
        points[question_num][1]);

    cin >> answers[question_num];

    answers[question_num] =
        toupper(answers[question_num]);

    if (answers[question_num] == 'B') // backup
    {
        --question_num; // go back to previous question
        clrscr();
        third_screen(); // redisplay first screen
        continue;
    }

```cpp
if (answers[question_num] != 'Y') &&
    (answers[question_num] != 'N')) // only yes
    // or no
{
    cout << "\a";
    continue;
}

++question_num;
continue;
} // end fifteenth question

if (question_num == 19)
{
    clrscr();
    fifth_screen(); // display screen

gotoxy(points[question_num][0],
        points[question_num][1]);

    cin >> answers[question_num];

    answers[question_num] =
        toupper(answers[question_num]);

    if (answers[question_num] == 'B') // backup
    {
        --question_num; // go back to previous question
        clrscr();
        fourth_screen(); // redisplay first screen
        continue;
    }

    if (answers[question_num] != 'Y') &&
        (answers[question_num] != 'N')) // only yes
        // or no
    {
        cout << "\a";
        continue;
    }

    ++question_num;
    continue;
} // end twentieth question

if (question_num == 28)
```
{
    clrscr();
    last_screen(); // display screen

gotoxy(points[question_num][0],
       points[question_num][1]);

    cin >> answers[question_num];

    answers[question_num] =
    toupper(answers[question_num]);

    if (answers[question_num] == 'B') // backup
    {
        --question_num; // go back to previous question
    clrscr(); // go back to previous question
    fifth_screen(); // redisplay first screen
    continue;
    }

    if ( (answers[question_num] != 'Y') &&
         (answers[question_num] != 'N')) // only yes
         // or no
    {
        cout << "\a";
        continue;
    }

    ++question_num;

    // write information to the output file
    strcat(fname,".dem");
ofstream outfile(fname);

    outfile << answers[1]; // output sex
    outfile << setfill('0') << setw(3) << age;

    for (i = 3; i < 28; ++i) // write the Y/N answers
    {
        if (i == 7)
        outfile << setfill('0') << setw(2) << years
        << setfill('0') << setw(2) << months;
        else
        outfile << answers[i];
    } // end for loop
outfile.close(); // close output file
continue;
} // end twenty-ninth question

if (question_num == 29)
{
gotoxy(points[question_num][0],
        points[question_num][1]);

cin >> answers[question_num];

answers[question_num] =
toupper(answers[question_num]);

if (answers[question_num] == 'B') // backup
{
    --question_num; // go back to previous question
    continue;
}

if ( (answers[question_num] != 'Y') &&
    (answers[question_num] != 'N')) // only yes
    // or no
{
    cout << "\a";
    continue;
}

if (answers[question_num] == 'N') // do another
    // file
{
    question_num = 0;
    clrscr();
    first_screen();
    continue;
}
else
    break;
} // end thirtieth question

else
{
gotoxy(points[question_num][0],
        points[question_num][1]);

cin >> answers[question_num];

answers[question_num] =
    toupper(answers[question_num]);

if (answers[question_num] == 'B') // backup
{
    --question_num; // go back to previous question
    continue;
}

if ( (answers[question_num] != 'Y') &&
    (answers[question_num] != 'N')) // only yes
    // or no
{
    cout << "\n";
    continue;
}

++question_num;
    continue;
} // end any other question

} // end while not done

clrscr();

cout << "\n*** Exiting Program ***\n\n";

} // end of main()

/* This function will display the questions which need to
be answered from the patient information questionnaire */

void first_screen(void)
{
    gotoxy(25,1);
    cout << "PATIENT DEMOGRAPHIC INFORMATION";

    gotoxy(5,5);
    cout << "Patient ID Number: ";
gotoxy(5,8);
cout << "Patient's Sex (M/F): ";

gotoxy(5,11);
cout << "Patient's Age in Years: ";

} // end first_screen()

void second_screen(void)
{
    gotoxy(25,1);
cout << "PATIENT DEMOGRAPHIC INFORMATION"

    gotoxy(5,5);
cout << "Patient Past History"

    gotoxy(10,8);
cout << "Was Patient Born Prematurely? (Y/N): ";

    gotoxy(10,11);
cout << "History of Respiratory Infections? (Y/N): ";

    gotoxy(15,14);
cout << "Pneumonia? (Y/N): ";

    gotoxy(15,17);
cout << "Bronchialitis? (Y/N): ";

    gotoxy(10,20);
cout << "Age When Murmur First Discovered? (Years Months): ";

} // end second_screen()
void third_screen(void)
{
    gotoxy(25,1);
    cout << "PATIENT DEMOGRAPHIC INFORMATION";
    gotoxy(5,5);
    cout << "Family History";
    gotoxy(10,8);
    cout << "Relatives with Heart Disorders? (Y/N): ";
    gotoxy(15,11);
    cout << "Mother? (Y/N): ";
    gotoxy(15,14);
    cout << "Father? (Y/N): ";
    gotoxy(15,17);
    cout << "Sibling? (Y/N): ";
    gotoxy(15,20);
    cout << "Other Immediate Family Member? (Y/N): ";
    gotoxy(10,23);
    cout << "Any Syndromes Associated with Heart Disease in the Family? (Y/N): ";
}

void fourth_screen(void)
{
    gotoxy(25,1);
    cout << "PATIENT DEMOGRAPHIC INFORMATION";
    gotoxy(5,5);
    cout << "Present Symptoms";
    gotoxy(10,8);
    cout << "Lack of Tolerance for Exercise? (Y/N): ";
    gotoxy(10,11);
    cout <<"Any Syndromes Associated with Heart Disease in the Family? (Y/N): ";
}
Unable to Keep up with Age Group at Play? (Y/N): 

gotoxy(10,14);
cout << "Cyanosis? (Y/N): ";

gotoxy(10,17);
cout << "Pallor? (Y/N): ";

gotoxy(10,20);
cout << "Fainting/Dizziness with Exercise? (Y/N): ";

} // end fourth_screen()

void fifth_screen(void)
{
    gotoxy(25,1);
cout << "PATIENT DEMOGRAPHIC INFORMATION"
;

gotoxy(5,5);
cout << "Physical Exam"
;

gotoxy(10,7);
cout << "Below Average Height on Growth Chart? (Y/N): ";

gotoxy(10,9);
cout << "Below Average Weight on Growth Chart? (Y/N): ";

gotoxy(10,11);
cout << "High Blood Pressure? (Y/N): ";

gotoxy(10,13);
cout << "Any Syndrome Associated with Height? (Y/N): ";

gotoxy(10,15);
cout << 
    "Presence of Other Congenital Malformations? (Y/N): ";

gotoxy(10,17);
cout << 
    "Left Side of Chest More Prominent Than Right? (Y/N): ";

- 92 -
gotoxy(10,19);
cout << "Pulse Too Fast? (Y/N): ";

gotoxy(10,21);
cout << "Pulse Too Slow? (Y/N): ";

gotoxy(10,23);
cout << "Respiratory Rate Too Fast? (Y/N): ";

} // end fifth_screen()

void last_screen(void)
{

gotoxy(25,1);
cout << "PATIENT DEMOGRAPHIC INFORMATION";

gotoxy(5,5);
cout << "Commit Information to File? (Y/N): ";

gotoxy(5,10);
cout << "Exit the Program? (Y/N): ";

} // end last_screen()
APPENDIX C

DIRECTRY Code Listings

ECHO OFF
rem ****************************************************
rem *
rem Written By: Erik Hudson
rem *
rem Written to support Master's Thesis
rem *
rem "Pediatric Heart Murmur Classification Using
rem A Backpropagation Neural Network"
rem *
rem *
rem This is a DOS batch program
rem *
rem Filename: DIRECTRY.BAT
rem *
rem *
rem The command line is DIRECTRY PATIENT_ID PATH
rem *
rem *
rem This program will create the directory structure
rem needed for the .WAV files corresponding to
rem PATIENT_ID
rem *
rem ****************************************************
rem
rem Create a subdirectory in the path name Patient_ID
rem
MKDIR %2\%1
rem
rem Create a subdirectory under Patient_ID for the Upper
rem Right Sternal Border and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\UPRT_STN.BDR
XCOPY BELL.WAV %2\%1\UPRT_STN.BDR /E
XCOPY DIAPHRGM.WAV %2\%1\UPRT_STN.BDR /E
rem
rem Create a subdirectory under Patient_ID for the Upper
rem Left Sternal Border and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\UPLF_STN.BDR
XCOPY BELL.WAV %2\%1\UPLF_STN.BDR /E
XCOPY DIAPHRGM.WAV %2\%1\UPLF_STN.BDR /E
rem
rem Create a subdirectory under Patient_ID for the Lower
rem Left Sternal Border and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\LOLF_STN.BDR
XCOPY BELL.WAV %2\%1\LOLF_STN.BDR /E
XCOPY DIAPHRGM.WAV %2\%1\LOLF_STN.BDR /E
rem
rem Create a subdirectory under Patient_ID for the Third
rem Space and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\3RD_SPCE
XCOPY BELL.WAV %2\%1\3RD_SPCE /E
XCOPY DIAPHRGM.WAV %2\%1\3RD_SPCE /E
rem
rem Create a subdirectory under Patient_ID for the
rem Apex and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\APEX
XCOPY BELL.WAV %2\%1\APEX /E
XCOPY DIAPHRGM.WAV %2\%1\APEX /E
rem
rem Create a subdirectory under Patient_ID for the Left
rem Midaxillary Line and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\LFMIDAXL.LIN
XCOPY BELL.WAV %2\%1\LFMIDAXL.LIN /E
XCOPY DIAPHRGM.WAV %2\%1\LFMIDAXL.LIN /E
rem
rem Create a subdirectory under Patient_ID for the Right
rem Midaxillary Line and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\RTMIDAXL.LIN
XCOPY BELL.WAV %2\%1\LFMIDAXL.LIN /E
XCOPY DIAPHRGM.WAV %2\%1\LFMIDAXL.LIN /E
rem Create a subdirectory under Patient_ID for the Left
rem Midpoint of the Back and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\LFMIDPT.BCK
XCOPY BELL.WAV %2\%1\LFMIDPT.BCK /E
XCOPY DIAPHRGM.WAV %2\%1\LFMIDPT.BCK /E
rem
rem Create a subdirectory under Patient_ID for the Right
rem Midpoint of the Back and place the zero length .WAV
rem Files for both the bell and diaphragm auscultation
rem At this location.
rem
MKDIR %2\%1\RTMIDPT.BCK
XCOPY BELL.WAV %2\%1\RTMIDPT.BCK /E
XCOPY DIAPHRGM.WAV %2\%1\RTMIDPT.BCK /E
echo
echo DONE! Directory structure created at %2 for %1
echo
APPENDIX D

WAVE Code Listings

/*

Written By: Erik Hudson

To Support Master's Thesis Work
"A Portable Computer System For Recording Heart Sounds And Data Modeling Using A Backpropagation Neural Network"

University of North Florida

Adviser: Dr. Layne Wallace

Files: WAVE.CPP and WAVE.H

Written In: Borland C++ 4.0 to Run in DOS

Purpose: Analyze the Wave file given in the command line. Ensure that it is a PCM Wave file. Extract useful information from the format and data chunks of the Wave file. Create a training and a testing file for the neural network based on the associated demographic information file and the local max and min values and their offsets in the data chunk

Usage: WAVE <Wave_file>

*/
// the standard C include header files

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <iostream.h>

```
#include <fstream.h>

// header file with class WaveFile
#include "wave.h"

void main(int argc, char **argv)
{
    // check for correct number of command line arguments
    if (argc != 2)
    {
        fprintf(stderr, "\nCorrect usage: wave <wave-file-name>\n"");
        exit (1);
    }

    // create a WaveFile object, pass the wave file name to
    // the constructor
    WaveFile wfObject(argv[1]);

    // check the riff id
    if (wfObject.get_riff_id() == 0)
        exit(1);
    else
        fprintf(stderr, "\nThe RIFF id is OK\n");

    if (wfObject.get_format_id() == 0)
        exit(1);
    else
        fprintf(stderr, "\nThe Format was read correctly\n");

    if (wfObject.get_data_id() == 0)
        exit(1);
    else
        fprintf(stderr,
            "\nThe data chunk id was read correctly\n");
}
```
if (wfObject.read_data(argv[1]) == 0)
    exit (1);
else
    fprintf(stderr,
        "\nThe data was read correctly, raw file created\n");

} // end main()

// This structure will contain the general data contained in
// all wave file format chunks

typedef unsigned short WORD;
typedef unsigned long DWORD;

typedef struct waveformat_tag {
    WORD wFormatTag;  // format type - PCM
    WORD nChannels;   // mono or stereo
    DWORD nSamplesPerSec;    // sample rate in kHz
    DWORD nAvgBytesPerSec;  // avg data rate
    WORD nBlockAlign;       // block alignment
} WAVEFORMAT;

// PCM WAVE Files contain an additional field in the
// format chunk - the bits per sample, e.g. 8 or 16 bit

typedef struct pcmwaveformat_tag {
    WAVEFORMAT wf;     // general format info
    WORD wBitsPerSample; // 8 or 16 bit
} PCMWAVEFORMAT;

// Define a class for the WAVE file

class WaveFile
{
    private:
        FILE *ifp;           // ptr to the wave file
char riff_id[12]; // stores the first 12 bytes of the file
char format[4]; // holds the ID for the format chunk
char data[4]; // holds the ID for the data chunk
unsigned long data_size; // holds the number of bytes in data chunk

PCMWAVEFORMAT pcm;

public:
WaveFile(const char *name); // constructor
int get_riff_id(void);
int get_format_id(void);
int get_data_id(void);
int read_data(const char *name);
void NN_files(char *ascii_file);
~WaveFile(void) { fclose(ifp);
    fprintf(stderr,
        "\nExiting ...
" );}

};

// WaveFile constructor
WaveFile::WaveFile(const char* name)
{
    // open the input file to read binary

    if ( (ifp = fopen(name,"rb")) == NULL)
    {
        fprintf(stderr,
            "\nCould not open file: %s\n",name);

        exit (1);
    }
} // end constructor
// this method will check the first 12 bytes of the file to see if it is a RIFF WAVE file

int WaveFile::get_riff_id(void)
{
    // read the first 12 bytes of the file into riff_id array
    if (fread(riff_id, 12, 1, ifp) != 1)
    {
        fprintf(stderr,
                "\nDid not read the first 12 bytes correctly\n"");

        return (0); // return error
    }

    // the first four bytes should be RIFF and bytes 9-12 should be WAVE

    {
        // the RIFF id is not correct, this is not a RIFF WAVE File

        fprintf(stderr,
                "\nThe RIFF id is not correct, this is not a RIFF WAVE File\n"");

        return (0); // return error
    } // end if

    else // it is a RIFF WAVE File
    {
        return (1);
    }

} // end WaveFile::get_riff_id()

int WaveFile::get_format_id(void)
{
    // check for format chunk id

    - 101 -
if (fread(format,4,1,ifp) != 1)
{
    fprintf(stderr,
        "\nError reading format chunk id\n");
    return (0);
}

// check to see if id is correct
if (format[0] != 'f' || format[1] != 'm' ||
{
    fprintf(stderr,"\nFormat id is incorrect\n");
    return (0);
}

// format chunk id is correct, now read in format data
char dummy[4]; // used to hold bytes to skip

// skip 4 bytes to get at format data
if (fread(dummy,4,1,ifp) != 1)
{
    fprintf(stderr,
        "\nError skipping 4 bytes in format chunk\n");
    return (0);
}

if (fread(&pcm,sizeof(PCMWAVEFORMAT),1,ifp) != 1)
{
    fprintf(stderr,"\nFormat not read correctly\n");
    return (0);
}

// print out the useful info from the format
if (pcm.wf.wFormatTag != 1)
{
    fprintf(stderr,
        "\nThis is not a PCM WAVE File\n");
    return (0);
}
if (pcm.wf.nChannels != 1)
{
    fprintf(stderr,
        "\nThis is not a mono PCM WAVE File\n");
return (0);
}
if (pcm.wBitsPerSample != 8 &&
    pcm.wBitsPerSample != 16)
{
    fprintf(stderr,
        \"This is not an 8 or 16 bit PCM WAVE File\n\");
    return (0);
}

fprintf(stderr,"\nThis is a PCM WAVE File\n");
fprintf(stderr,
    \"Number of Channels: %d\n", pcm.wf.nChannels);
fprintf(stderr,
    \"Sample Rate: %ld\n", pcm.wf.nSamplesPerSec);
fprintf(stderr,
    \"Bits Per Sample: %d\n", pcm.wBitsPerSample);
return (1);

} // end WaveFile::get_format_id()

// This method will check the data format id and get the
// size of the data in bytes

int WaveFile::get_data_id(void)
{
    // read in data chunk id
    // there may be one or more optional chunks
    // between the format chunk and the data
    // chunk

    long chunk_length = 0;

    while (1)
    {
        if (fread(&data,4,1,ifp) != 1) // read in chunk
            // id
        {
            fprintf(stderr,
                \"Error reading data chunk id\n\");
        
            break;
        }
    }

    // get size of data
    chunk_length = ((long)pcm.wf.nSamplesPerSec *
        pcm.wf.nChannels *
        pcm.wBitsPerSample / 8);
return (0);

// check chunk id
if (data[0] != 'd' || data[1] != 'a' ||
{
    // This is not the Data chunk id
    // get the length of this chunk & advance
    // the file pointer
    // by that length

    // read in chunk id
    if (fread(&chunk_length, 4, 1, ifp) != 1)
    {
        fprintf(stderr,
                "\nError reading chunk length\n" );
        return (0);
    }
    else // move file pointer
    {fseek(ifp, chunk_length, SEEK_CUR);
        continue;
    } // end if not data chunk id

    else // is the data chunk id, break out of loop
    break;
}

// read in size of data in bytes

data_size = 0;

if (fread(&data_size, 4, 1, ifp) != 1)
{
    fprintf(stderr,
            "\nData size not read correctly\n" );
    return (0);
}

fprintf(stderr,
        "\tData size in bytes: %ld\n", data_size);
int WaveFile::read_data(const char *name)
{
    char outfile_name[128], ascii_file[128];
    strcpy(outfile_name, name);
    int length = strlen(outfile_name);
    // outfile will become name.raw
    outfile_name[length-1] = 'w';
    outfile_name[length-3] = 'r';
    strcpy(ascii_file, outfile_name);
    ascii_file[length-1] = 'c';
    ascii_file[length-2] = 's';
    ascii_file[length-3] = 'a';

    FILE *ofp, *ofpl;
    // open output file for writing binary
    if ((ofp = fopen(outfile_name, "wb")) == NULL)
    {
        fprintf(stderr, "\nCould not open raw data output file\n");
        return (0);
    }
    // open output file for writing ascii
    if ((ofpl = fopen(ascii_file, "w")) == NULL)
fprintf(stderr, "Could not open ascii data output file\n");
return (0);
}

unsigned char c;

if (pcm.wBitsPerSample == 8) // eight bit file, get
    // then put byte
{
    for (long i = 0; i < data_size; ++i)
    {
        if (fread(&c, 1, 1, ifp) != 1) // read byte
            // from input
        {
            if (feof(ifp))
            {
                fprintf(stderr, "\nReached end of data\n");
                break;
            }
            fprintf(stderr, "\nError reading the data\n");
            return (0);
        }
        else // write the byte to output file
        {
            if (fwrite(&c, 1, 1, ofp) != 1) // write the offset and the value
            {
                fprintf(stderr, "\nError writing the data\n");
                return (0);
            }
            // to the ascii file
            fprintf(ofp1,"%ld %d\n", i, c);
        }
    }
}
fclose(ofp);

- 106 -
fclose(ofp1);

) // end if

else // 16 bit Wave file
{
    unsigned char c1;
    int hold_value;
    char *hold_addr;

    for (long i = 0; i < data_size; i+=2)
    {
        // read 2 bytes from input
        if (fread(&c,1,1,ifp) != 1 ||
            fread(&c1,1,1,ifp) != 1)
        {
            if (feof(ifp))
                {
                fprintf(stderr,
                        "\nReached end of data\n");
                        break;
                }
            fprintf(stderr,
                    "\nError reading the data\n");
            return (0);
        }
        else // write the bytes to the output
            // file in reverse order
        {
            // 16 bit WAVE files store data in
            // little-endian format
            if (fwrite(&c1,1,1,ofp) != 1 ||
                fwrite(&c,1,1,ofp) != 1)
            {
            fprintf(stderr,
                    "\nError writing the data\n");
                return (0);
            }

            // need to take care of little
            // endian problem
            // reverse the bytes, use bitwise
            // shift

            hold_addr = (char*)(&hold_value);
    }
*hold_addr = cl;

// cl is in the most significant
// byte now
++hold_addr; // hold_addr points at
    // low byte
*hold_addr = c; // now 16 bit int
    // should be
    // stored correctly

// write to the ascii file
fprintf(ofp1,
    "%ld %d\n", i/2, hold_value);

}  // end else

} // end for

fclose(ofp);
fclose(ofp1);

} // end else

// create training and test files for this wave
NN_files(ascii_file);
return (1);
} // end WaveFile::read_data()

// This member function will create the training pairs file
// and the testfile for this wave recording. It takes the
// name of the ASCII that was created in read_data as a
// parameter

void WaveFile::NN_files(char *ascii_file)
{

int demographic_info[28]; // will hold the data from
// the *.dem file

FILE *ifp,*ofp1,*ofp2;

char test_name[128],train_name[128],dem_name[128];

cout << "Enter the name of the demographic info file
associated with this" << " wave file: \n";
icin >> dem_name;

int len = strlen(ascii_file);
strcpy(test_name,ascii_file);
strcpy(train_name,ascii_file);

train_name[len-3] = test_name[len-3] = 't';
train_name[len-2] = 'r';
test_name[len-2] = 's';
train_name[len-1] = 'n';
test_name[len-1] = 't';

if ( (ifp = fopen(dem_name,"r")) == NULL) {
    cout << "Error opening demographic file "
        << dem_name << endl;
    exit (1);
}

// read in the demographic info

for (int i = 0; i < 28; ++i) {fscanf(ifp,"%d", &demographic_info[i]);}

fclose(ifp); // close demographic file

// open the ascii data file for input

if ( (ifp = fopen(ascii_file,"r")) == NULL) {
    cout << "Error opening ascii data file "
        << ascii_file << endl;
}
exit (1);
}

if ( (ofp1 = fopen(train_name,"w")) == NULL) {
    cout << "Error opening training data file " 
        << train_name << endl;
    exit (1);
}

if ( (ofp2 = fopen(test_name,"w")) == NULL) {
    cout << "Error opening test data file " 
        << test_name << endl;
    exit (1);
}

long offset;

// these values will be used to determine the slope
int value;
int previous_value;
int slope;
int eight_bit_flag,same_sign;

if (pcm.wBitsPerSample == 8)
    eight_bit_flag = 1;

int previous_slope;
int prev1,prev2,prev3; // used for the running average

while ( fscanf(ifp,"%ld %d",&offset,&value) != EOF) {
    if (eight_bit_flag)
        value -= 128;

    if (offset == 0) {
        prev1 = prev2 = prev3 = value;
        previous_slope = value;

    /* code */
}
II use a running average to compute previous value

previous_value = (prev1 + prev2 + prev3)/3;

slope = value - previous_value;  // calculate the  
   // slope

same_sign = slope*previous_slope;

// ignore if not at a max or min point

if ( ((previous_slope > 0) && (slope < 0) &&  
   (value < previous_value)) ||  
   ((previous_slope < 0) && (slope > 0) &&  
   (value > previous_value)) )
   ;  // NULL statement, this is a max or min,  
   // need to use in training

else  // ignore this, get next value
   {
      prev3 = prev2;
      prev2 = prev1;
      prev1 = value;
      previous_slope = slope;
      continue;
   }

// write to the training file, input is same as  
// output

for (int j = 0; j < 2; ++j)
   {
      for (i = 0; i < 28; ++i)
         fprintf(ofp1,"%d ",demographic_info[i]);

      fprintf(ofp1,"%ld %d
",offset-1,previous_value);
   }  // end for j

// show values read from ascii file
printf("%ld %d\n",offset-1,previous_value);
// write to the test file, just gets input, no  
// output

for (i = 0; i < 28; ++i)
     fprintf(ofp2,"%d ",demographic_info[i]);
fprintf(ofp2,"%ld %d\n", offset-1, previous_value);

prev3 = prev2;
prev2 = prev1;
prev1 = value;
previous_slope = slope;

} // end while

// close output files and input file

fclose(ofp1);
fclose(ofp2);
fclose(ifp);

cout << "The files " << train_name << " and " << test_name <<
" were create for use with the neural net\n\n";

} // end WaveFile::NN_files
Erik Mark Hudson received a Bachelor of Science Degree in Aerospace Engineering from the University of Florida in May, 1991 and expects to receive a Master of Science in Computer and Information Sciences from the University of North Florida, December, 1995. Dr. Layne Wallace of the University of North Florida is serving as Erik's thesis adviser. Erik is currently employed as a software engineer for Harris Corporation's Information Systems Division. He has been involved in a research and development project involving on-line transaction processing and relational databases.

Erik's computer interests include image processing, relational databases, and artificial intelligence. Erik has programmed extensively in C, as well as C++, FORTRAN, and SQL. Erik has a wife, Luanne, and a five-year-old daughter, Felicity.