Music Genre Classification Using a Simplified Neural Network

Keenan Johnson

December 7, 2013

Computer Engineering 358

Computational Intelligence

Author Biography

Keenan Johnson is an undergraduate in Computer Engineering at Missouri University of Science & Technology. He is currently a senior and will graduate in December of 2014. This class is Keenan's first experience with Neural Networks and the professional field of Computational Intelligence.

Table of Contents

1. EXECUTIVE SUMMARY

Automatically classifying musically genres is a difficult task due to loosely defined genre definitions. Neural networks have been shown to be an effective tool in automatically classifying musical genre. This paper presents a feed-forward neural network for classifying music genre. Unlike previous neural networks, which utilize a wide variety of musical characteristics, this neural network looks only at the timbre or tonal character of a section of music using Mel-Frequency Coefficients. This approach is shown to be effective, yielding an accuracy of 96.94% when classifying four genres of music and 74.68% when classifying ten genres of music.

2. INTRODUCTION

Music genres are conventional containers that society uses to classify and describe songs. Examples include genres such as "Pop," "Rock," and "Jazz." These genres arise through a complex interaction between the public, marketing efforts, historical precedence and cultural factors. Due to this complex interaction, there is no strict definition or boundaries for music genres.

The subjective and blurry definition of musical genres makes them a difficult quantity to automatically identify. However, there are significant applications that would benefit from doing just that. Musical services such as Pandora and Spotify base their business models on supplying the consumer with automatic music selections. The costumer must find these selections pleasing or will likely be unhappy with the service. Thus, an automated system to tag songs with an automatically determined musical genre would be of great use and allow companies to tailor music to customers by genre.

Music genre also becomes an increasingly important characteristic by which to sort the vast number of digital music files that exist on the Internet today.

A. Genre Characteristics

To automatically determine the genre of a song, the musical characteristics that represent the genre of a song must be identified.

One might assume that in order to correctly identify the genre of a song, many parameters such as melodic progression and rhythm must be analyzed. Both of the aforementioned characteristics require on a large sample of the song to be analyzed. This means that the system will need to process a large amount of data, and long computation times.

However, a human study performed in [1] revealed that humans determine genre with a very small portion of a musical selection. In the study, the authors sought to determine how long it takes humans to determine the genre of a song. College students were presented with variable length clips and asked to identify the clip as one of ten genres.

The students were 53% effective at identifying the genre given a 250 ms sample and 70% effective given a three second clip. If given longer than the three second clip the students were not more effective at identifying the publisher's genre. [1] This does not mean that the students were incorrect when they did not agree with the publisher's genre assignment. Instead, it quantifies the subjectivity of musical genres. Thus 70% represents the theoretical limit of genre classification correctness, since that is the rate at which humans classify genres.

The study in [1] also showed that humans make decisions about the genre of a song based on a very small time slice of the song. At the tempo of 110 beats per minute, an average song tempo, less than half of one beat will fit inside one 250 ms time slice. This means that it is impossible for humans to determine the genre of a song based upon any time dependent characteristic.

A song's timbre is its tone color or quality. It encompasses all spectral qualities in the acoustic signal of the song [1]. This quality

seems to be strongly connected to the recognition of a song's genre.

B. Representing Timbre

In order to make the processing of the audio efficient and effective, the timbre of a song clip needs to be represented in a concise manner.

Fortunately, timbral feature extraction is a standard practice in the field of speech recognition [2]. Mel-frequency cepstral coefficients provide a good way to represent the perceived frequency characteristics of an audio file as a series of coefficients. These coefficients are based on the Fourier Transform of the audio signal. The log-amplitude of the magnitude spectrum is then computed. The FFT bins are then grouped, and mapped to the Mel-frequency scale. Finally a discrete cosine transform is performed to decorrelate the resulting feature vectors [2] [3].

Figure 1 - Mel Scale [5]

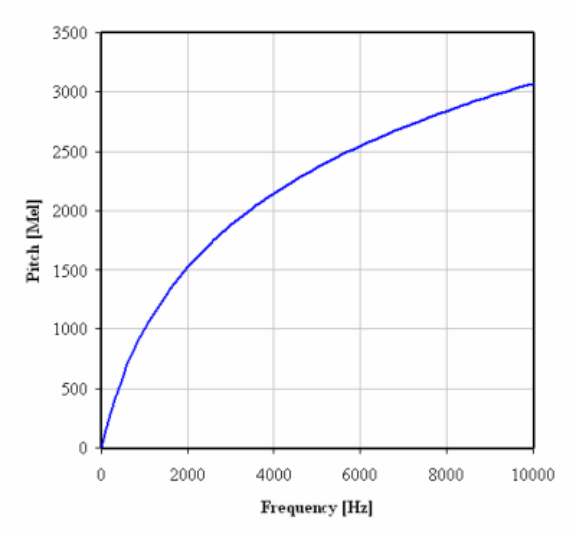


Figure 1 shows the Mel Scale. It is a mapping of perceived frequency to actual frequency since humans do not perceive frequencies linearly [5].

C. Classification Method

In the past, researchers have attempted to use standard statistical and machine learning methods to classify the genre of songs with around 60% accuracy at best[6][9].

However previous work involving neural network musical classification seems to yield far better results.

In [2] a radial basis function network was used to classify musical genre. This technique used Mel-frequency cepstral coefficients, rhythmic and pitch features as inputs to the network. This paper used the GTZAN dataset which contains ten musical genres. The researchers were able to achieve a ten genre musical classification accuracy of 71.5%.

In [9] a normal feed forward back propagation network was used to classify musical genre. The network was trained using the Levenberg-Marquardt algorithm. This network took a wide variety of features extracted from the song as input. These features are listed in Table 1.

Figure 2- [9] Network Inputs

Feature	Number of Parameters
LPC Taps	32
DFT Amplitude Value	32
Log of DFT Amplitude Values	32
IDFT of Log of DFT Amplitude Values	12
Mel-frequency Cepstral Coefficients	15
Volume	1

The above network achieved a classification of four musical genres with an accuracy of 94.8%. The dramatically increased accuracy of this network can likely be attributed to the reduced number of genres, which simplifies the problem considerably.

D. Simplified Neural Network

As shown above, there has been significant and effective previous work using neural networks to classify musical genres. However the two previous networks have used a complex set of inputs that consist of a large number or parameters.

It may be possible to achieve similar results using a simplified neural network.

3. PROJECT SPECIFICATIONS

A. Neural Network

As mentioned above, it should be possible to construct a simplified neural network to classify musical genres. Based upon the human study in [1], the only inputs to such a neural network should be the Mel-Frequency Cepstral Coefficients.

Two networks will be constructed, trained and tested in order to compare to the previous research

The networks will be simple feed-forward networks that accept Mel-Frequency Cepstral Coefficients as inputs.

These networks will be trained using the Scaled Conjugate Gradient Back propagation algorithm [7].

The first network will identify songs as one of four genres in order to compare to the network described in [2].

A second network will be constructed that will identify songs as one of ten genres in order to compare results with the results in [9].

B. Dataset

In order to train, validate and test the above networks, the GTZAN dataset will be used. This is a free, publically available dataset available here (http://marsyas.info/download/data_sets/) and used in [6].

This dataset of 1000 thirty second audio tracks was collected from the year 2000 to 2001. The tracks are all 22050Hz Mono 16-bit audio files in .au format.

The songs in the dataset are presorted by genre. There are ten genres, listed in Table 2. Each genre consists on one hundred songs.

Figure 3 - GTZAN Genres

Genres
Blues
Classical
Country
Disco
Hip-hop
Jazz
Metal
Pop
Reggae
Rock

The four genre network will classify Classical, Jazz, Metal and Pop.

The ten genre network will classify the entire dataset.

4. DETAILED DESIGN

All code for the following system is attached in the appendix.

A. Song Loading

Since the Neural Network was designed and implemented in Matlab, all of the preprocessing was also done in Matlab. To load the songs into memory, a .csv file was created containing the file paths to each song along with the correct genre of each song.

B. MFCC Coefficients

In order to input a song into a neural network, the Mel-Frequency Cepstral Coefficients must be calculated. To do this, a ten second selection is taken out of the middle of each audio file. This ten second section is then split in 20 ms long segments as shown in figure 4.

Figure 4 - Splitting Song Waveform

Cepstral Coefficients are then calculated for each 20 ms segment using the following algorithm:

1. Take the Fast Fourier Transform of the window
2. Map the frequency spectrum to the Mel Scale
3. Take the logarithm of the powers at each of the mel frequencies (20 bins)
4. Take the discrete cosine transform of the Mel log powers
5. Drop the five highest coefficients since they don't really affect the song.

The song is now represented as a matrix of MFCC's where each row is a vector of 15 elements representing a 20 ms slice of the song.

C. Neural Network Formatting

To further reduce the complexity of the MFCCs, the mean vector and covariance matrix are computed. The mean vector and top half of the covariance matrix are then combined into a single feature vector of 135 elements that represent the perceived timbre of the ten second audio clip.

This vector is now ready to be fed into the neural network and is stored in memory with the rest of the input songs.

D. Neural Network Design

As mentioned earlier, two neural networks were created. A simplified view of both networks is shown below in figures five and six.

Figure 5 - 4 Genre Network Layout

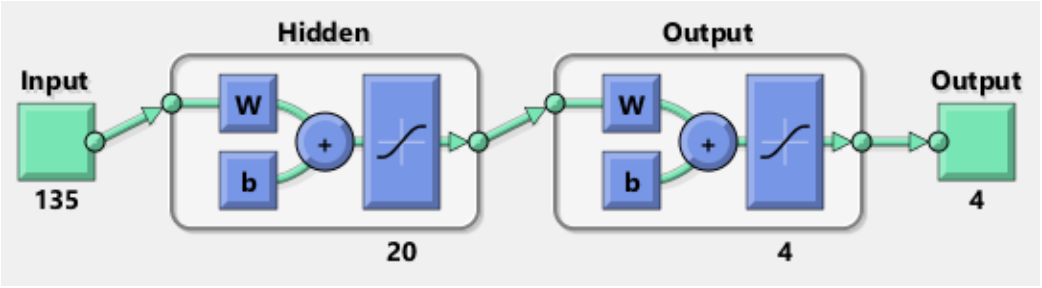
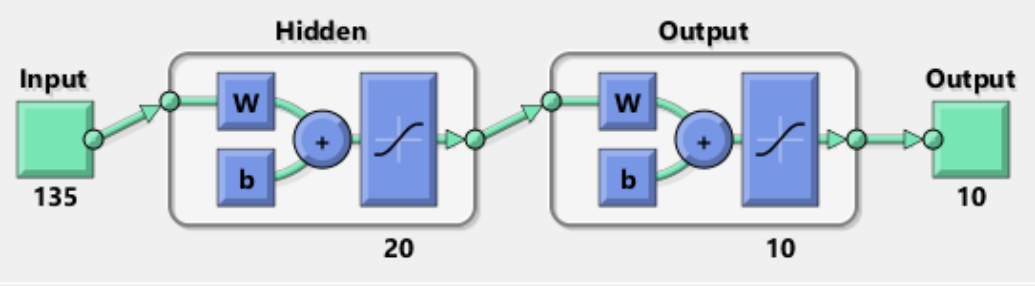


Figure 6 - 10 Genre Network Layout



Both networks contain only one hidden layer and 135 input neurons. These 135 inputs correspond to the 135 element feature vector described above. The number of output neurons corresponds to the number of genres the network is set up to detect. Each output neuron corresponds to a music genre and will output a one if that genre is detected.

In both networks 20 hidden layer nodes were chosen. These were chosen by experimentally by running each network with a variable number of hidden layer nodes. The following graphs show the results of those trials.

Figure 7 - 4 Genre Accuracy

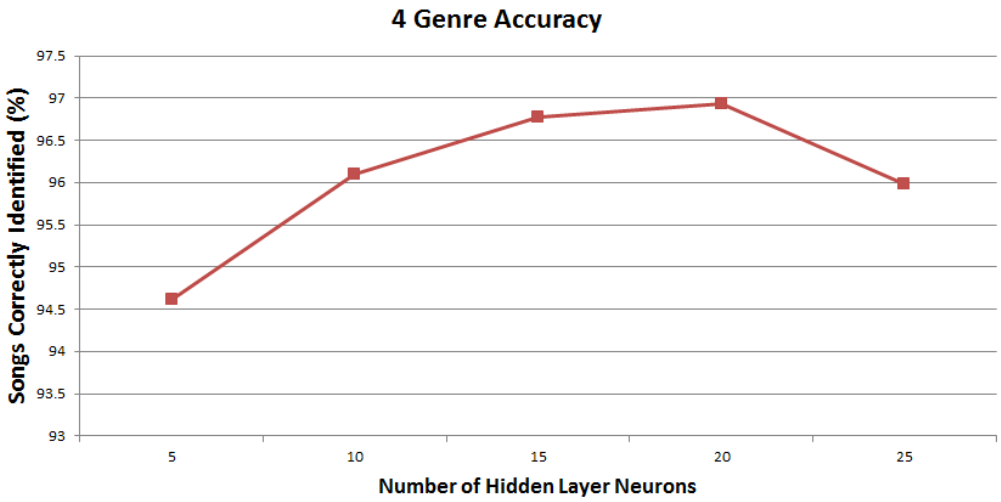
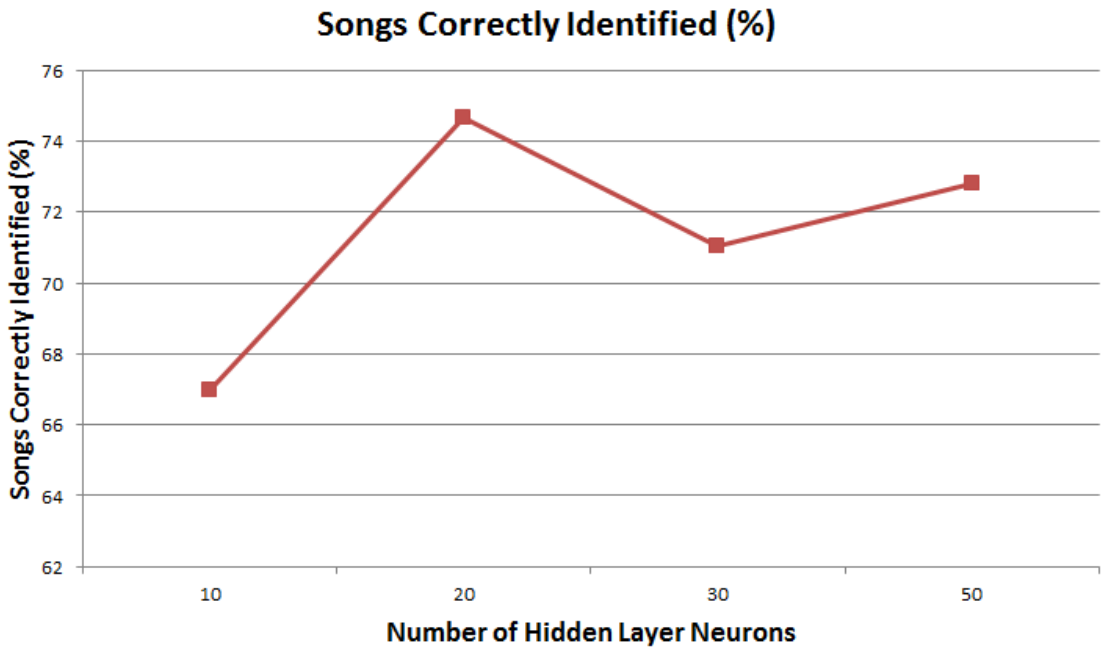


Figure 8 - 10 Genre Accuracy



In both Figure 7 and Figure 8, there exist maxima at 20 hidden layer nodes. Thus the networks were tested and analyzed with 20 hidden layer nodes.

E. Neural Network Training

The neural networks were trained using the Scaled Conjugate Gradient Back propagation algorithm. This algorithm is superior to the standard back propagation algorithm and provides at least an order

of magnitude speed-up in training time while still utilizing second order information from the network. For more information on the algorithm see [7].

Per common practice, 70% of the data was used for training, 15% for validation to prevent over training, and 15% was used for testing.

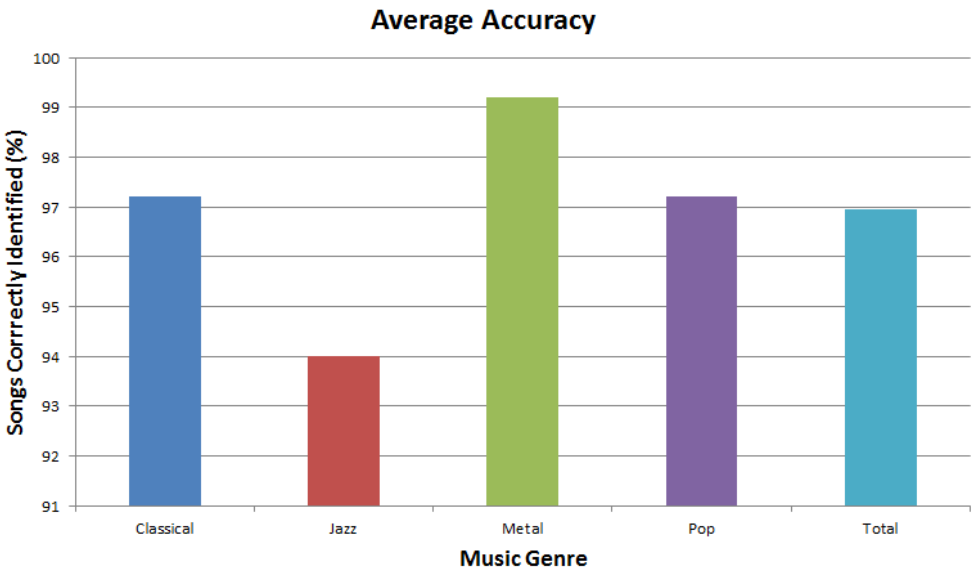
5. EXPERIMENTAL RESULTS

A. Four Genre Results

The four genre neural network was trained and tested first. The network was trained and tested five times. For complete training data and confusion matrices see Appendix C.

The average accuracy of the five trials is shown in Figure 9 below.

Figure 9 - 4 Genre Accuracy



The network has an average total accuracy of 96.94%. This is slightly better than the 94.8%.performance of the four genre network in [9].

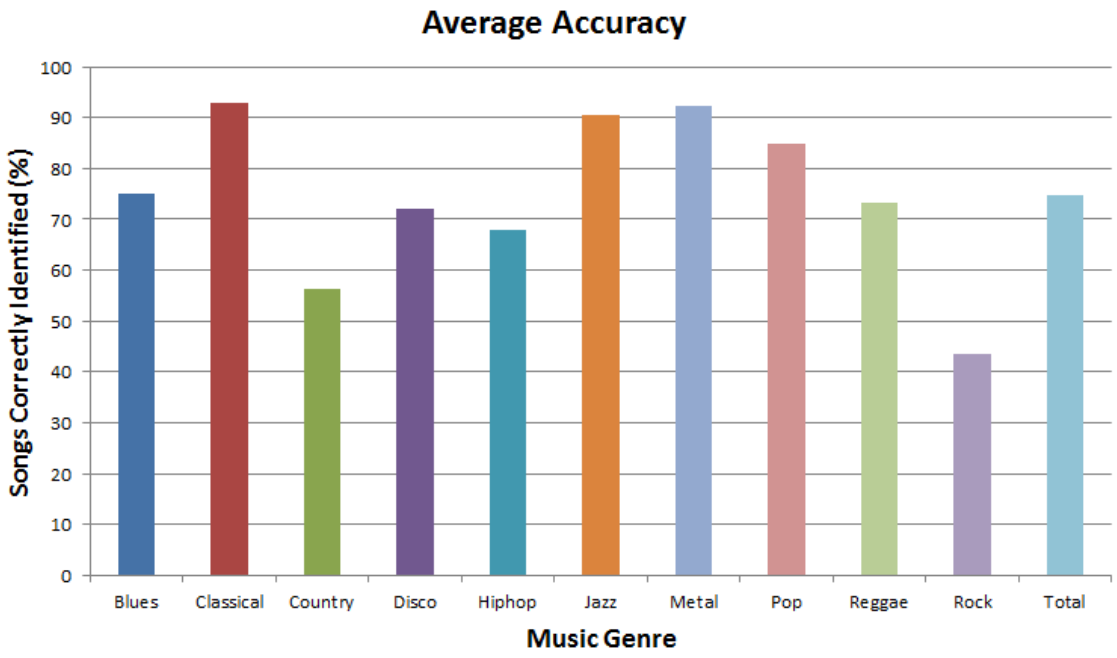
Thus, it appears that the neural network presented in this paper, using only MFCCs as inputs, classifies four musical genres as well or better than varied input neural networks.

B. Ten Genre Results

The ten genre neural network was trained and tested next. The network was trained and tested five times. For complete training data and confusion matrices see Appendix D.

The average accuracy of the five trials is shown in Figure 9 below.

Figure 10 - 10 Genre Accuracy



As shown in the diagram, the accuracy of genre classification is largely dependent on genre. Metal, for example, is classified correctly less than 50% of the time. However, Classical music is classified correctly over 90% of the time.

The total accuracy of the 10 genre network is 74.68%. This is very close to the human 10 genre classification accuracy of 70% [1] and the radial basis network's 10 genre accuracy of 71.5% [2].

Thus, it appears that the neural network presented in this paper, using only MFCCs as inputs, classifies 10 musical genres as well or better than varied input neural networks and human classification.

ACKNOWLDGEMENTS

The author would like to thank Dr. Wunsch for his support and advice.

He would also like to thank G. Tzanetakis and P. Cook for providing the GTZAN Genre Collection for free on their website.

REFERENCES

[1] Gjerdingen, Robert O., and David Perrott. "Scanning the dial: The rapid recognition of music genres." Journal of New Music Research 37.2 (2008): 93-100.

[2] Turnbull, Douglas, and Charles Elkan. "Fast recognition of musical genres using RBF networks." Knowledge and Data Engineering, IEEE Transactions on 17.4 (2005): 580-584.

[3] S. Davis and P. Mermelstein, "Experiments in syllable-based recognition of continuous speech," IEEE Trans. Acoust., Speech, Signal Processing, vol. 28, pp. 357–366, Aug. 1980.

[4] Stevens, Stanley Smith; Volkman; John; & Newman, Edwin B. (1937). "A scale for the measurement of the psychological magnitude pitch". Journal of the Acoustical Society of America 8 (3): 185–190.

[5] Image taken from DiracDelta.co.uk http://www.diracdelta.co.uk/science/source/m/e/mel/image003.gif

[6] Tzanetakis, G.; Cook, P., "Musical genre classification of audio signals," Speech and Audio Processing, IEEE Transactions on , vol.10, no.5, pp.293,302, Jul 2002.

[7] Møller, Martin Fodslette. "A scaled conjugate gradient algorithm for fast supervised learning." Neural networks 6.4 (1993): 525-533.

[8] Feng, Yazhong, Yueting Zhuang, and Yunhe Pan. "Music information retrieval by detecting mood via computational media aesthetics." Web Intelligence, 2003. WI 2003. Proceedings. IEEE/WIC International Conference on. IEEE, 2003.

[9] Scott, Paul. "Music classification using neural networks." Manuscript Class ee373a, Stanford (2001).

APPENDIX A – 4 Genre Neural Network

"csv_to_data.m"

```
%test
clear all;

%training data
CSV_FILE_PATH_training =
'C:\Users\Batman\Desktop\Dropbox\CI_Project_Files\400\gtzan_400.csv'
NUM_BINS = 22;
NUM_FRAMES = 200;
NUM_COEFF = 15;
STEP_TIME = 0.02;

mfcc_training_data400 = get_mfcc_features(CSV_FILE_PATH_training, NUM_BINS,
NUM_FRAMES, NUM_COEFF, STEP_TIME);
save mfcc_training_data400;
```

"ge_mfcc_features.m"

```
function mfcc_cells = get_mfcc_features(CSV_FILE_PATH, NUM_BINS, NUM_FRAMES,
NUM_COEFF, STEP_TIME)
% MFCC_CELLS = GET_MFCC_FEATURES(CSV_FILE_NAME, NUM_BINS, NUM_FRAMES,
%              NUM_COEFF, STEP_TIME);
%
% MFCC_CELLS is a <# songs> by 2 cell matrix, where column 1 is the means
% of the mfcc and column 2 is the covariances of the mfcc. Each row is the
% mfcc data of a song.

fid = fopen(CSV_FILE_PATH);
csv_data = textscan(fid, '%s %s %s %s', 'Delimiter', '|', 'CollectOutput',
1);
csv_data = csv_data{1};
fclose(fid);

[numRows numCols] = size(csv_data);
mfcc_cells = cell(numRows-1, 1 + numCols);

for row = 2:numRows
    fprintf('song # %d\n', row);
    cur_song = csv_data(row, :);
    song_file_name = cell2mat(cur_song(1, 1));
    mfcc_mat = mfcc(song_file_name, NUM_BINS, NUM_FRAMES, NUM_COEFF,
STEP_TIME);
    mfcc_cells{row-1, 1} = mean(mfcc_mat);
    mfcc_cells{row-1, 2} = cov(mfcc_mat);
    for col = 3:numCols + 1
        mfcc_cells{row-1, col} = csv_data(row, col-1);
    end
    fprintf('\n');
end

end
```

"mfcc.m"

```
function MFCC_matrix = mfcc(song_file_name, NUM_BINS, NUM_FRAMES, NUM_COEFF,
STEP_TIME)
% MFCC_MATRIX = MFCC(song_file_name, NUM_BINS, NUM_FRAMES, NUM_COEFF,
STEP_TIME)
%
% MFCC_MATRIX is the NUM_FRAMES-by-NUM_COEFF matrix of MFCC coefficients.
% NUM_FRAMES is how many frames (20 ms samples) of the song to use.
% NUM_BINS is how many mel frequency bins to map each frame spectrum to.
% NUM_COEFF is how many mel coefficients to keep (NUM_COEFF <= NUM_BINS).
% STEP_TIME is how often (in seconds) to capture a 20ms frame.
%
%      - Mel-Frequency Ceptrum Coefficients -
% 1. Take the Fourier Transform of a 20-30ms windowed excerpt of a signal.
% 2. Map powers of spectrum onto mel scale, using triangular overlapping
% windows.
% 3. Take the logs of the powers at each of the mel frequencies.
% 4. Take the Discrete Cosine Transform of the list of mel log powers (as
% if it were a signal).
% 5. The MFCCs are the amplitudes of the resulting spectrum.


NUM_SEC = 10;  % take a full 10-second sample (then split into 20ms frames)

format = song_file_name(end-1:end);

if (strcmp(format, 'av'))
    slength = wavread(song_file_name, 'size');
    offset = floor(slength(1,1)/3);
    [~, fs] = wavread(song_file_name, 1);
    [s, fs] = wavread(song_file_name, [offset offset+fs*NUM_SEC]);
elseif (strcmp(format, 'au'))
    slength = auread(song_file_name, 'size');
    offset = floor(slength(1,1)/3);
    [~, fs] = auread(song_file_name, 1);
    [s, fs] = auread(song_file_name, [offset offset+fs*NUM_SEC]);
elseif (strcmp(format, 'p3'))
    slength = mp3read(song_file_name, 'size');
    offset = floor(slength(1,1)/3);
    [~, fs] = mp3read(song_file_name, [offset offset+1]);
    [s, fs] = mp3read(song_file_name, [offset offset+fs*NUM_SEC]);
else
    error('Wrong filename input. Use a .mp3, .wav, or .au audio file');
end


fft_len = 512;              % length of fft
frame_time = 0.020;         % # seconds/frame
frame_len = floor(fs*frame_time);  % # samples/frame

step_len = floor(fs*STEP_TIME);    % # samples/frame step
```

```
MFCC_matrix = zeros(NUM_FRAMES, NUM_COEFF);

window = hamming(frame_len);
stop = 1+floor(fft_len/2);  % 257

[x, ~] = size(s);
begin = floor(x/2);

for i = 1:NUM_FRAMES
    first = begin + (i-1)*step_len + 1;
    last = begin + (i-1)*step_len + frame_len;
    f = s(first:last);
    heightF = size(f, 1);
    widthF = size(f, 2);
    if (heightF > widthF)
        f = f.*window;
    else
        f = f'.*window;
    end
    f = fft(f, frame_len);
    mel_bins = melbankm(NUM_BINS, fft_len, fs);
    f = abs(f(1:stop)).^2;
    m = log10(mel_bins*f);
    m = dct(m);
    m = m(1:NUM_COEFF);
    MFCC_matrix(i, :) = m';
end


end
```

"melbankm.m"

```
function [x,mn,mx] = melbankm(p,n,fs,fl,fh,w)
%
%  function  [x,mn,mx] = melbankm(p,n,fs,fl,fh,w)
%
%  Determine matrix for a mel-spaced filterbank
%
%  Inputs:   p   number of filters in filterbank
%            n   length of fft
%            fs  sample rate in Hz
%            fl  low end of the lowest filter as a fraction of fs (default =
0)
%            fh  high end of highest filter as a fraction of fs (default =
0.5)
%            w   any sensible combination of the following:
%                't'  triangular shaped filters in mel domain (default)
%                'n'  hanning shaped filters in mel domain
%                'm'  hamming shaped filters in mel domain
%
%                'z'  highest and lowest filters taper down to zero (default)
%                'y'  lowest filter remains at 1 down to 0 frequency and
%                     highest filter remains at 1 up to nyquist freqency
%
%                If 'ty' or 'ny' is specified, the total power in the FFT
%                is preserved.
%
%  Outputs:  x   sparse matrix containing the filterbank amplitudes
%                If x is the only output argument then
%                  size(x)=[p,1+floor(n/2)]
%                otherwise
%                  size(x)=[p,mx-mn+1]
%            mn  the lowest fft bin with a non-zero coefficient
%            mx  the highest fft bin with a non-zero coefficient
%
%  Usage:    f = fft(s);                % mel cep over full freq range
%            x = melbankm(p,n,fs);
%            n2 = 1+floor(n/2);
%            z = log(x*abs(f(1:n2)).^2);
%            c = dct(z); c(1) = [];
%
%            f = fft(s);                % mel cep over part of freq range
%            [x,na,nb] = melbankm(p,n,fs);
%            z = log(x*(f(na:nb)).*conj(f(na:nb)));
%
% To plot filterbanks e.g.  plot(melbankm(20,256,8000)')


%      Copyright (C) Mike Brookes 1997
%
%      Last modified Tue May 12 16:15:28 1998
%
% VOICEBOX home page:
%          http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%   This program is free software; you can redistribute it and/or modify
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation; either version 2 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public License from
%   ftp://prep.ai.mit.edu/pub/gnu/COPYING-2.0 or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

if nargin < 6
  w='tz';
  if nargin < 5
    fh=0.5;
    if nargin < 4
      fl=0;
    end
  end
end
f0=700/fs;
%f0=1000/fs;
fn2=floor(n/2);
lr=log((f0+fh)/(f0+fl))/(p+1);
% convert to fft bin numbers with 0 for DC term
bl=n*((f0+fl)*exp([0 1 p p+1]*lr)-f0);
b2=ceil(bl(2));
b3=floor(bl(3));
if any(w=='y')
  pf=log((f0+(b2:b3)/n)/(f0+fl))/lr;
  fp=floor(pf);
  r=[ones(1,b2) fp fp+1 p*ones(1,fn2-b3)];
  c=[1:b3+1 b2+1:fn2+1];
  v=2*[0.5 ones(1,b2-1) 1-pf+fp pf-fp ones(1,fn2-b3-1) 0.5];
  mn=1;
  mx=fn2+1;
else
  b1=floor(bl(1))+1;
  b4=min(fn2,ceil(bl(4)))-1;
  pf=log((f0+(b1:b4)/n)/(f0+fl))/lr;
  fp=floor(pf);
  pm=pf-fp;
  k2=b2-b1+1;
  k3=b3-b1+1;
  k4=b4-b1+1;
  r=[fp(k2:k4) 1+fp(1:k3)];
  c=[k2:k4 1:k3];
  v=2*[1-pm(k2:k4) pm(1:k3)];
  mn=b1+1;
  mx=b4+1;
```

```
end
if any(w=='n')
  v=1-cos(v*pi/2);
elseif any(w=='m')
  v=1-0.92/1.08*cos(v*pi/2);
end
if nargout > 1
  x=sparse(r,c,v);
else
  x=sparse(r,c+mn-1,v,p,1+fn2);
end
```

"NN_Input_Generator.m"

```
%generates the input for Neural Network Analysis
%clear all;
load mfcc_training_data400

NUM_COEFF = 15;
NUM_GENRES = 4;

mfcc_cells = mfcc_training_data400;


m = size(mfcc_cells, 1);

num_features = NUM_COEFF + (1 + NUM_COEFF)*NUM_COEFF/2;

nn_input = zeros(m, num_features);
nn_output = zeros(m, NUM_GENRES);


max_all = 0;

%scaling input
for i = 1 : m
    [nn_input(i, :) cur_max] = format_mfcc_to_vector(mfcc_cells{i, 1}, ...
        mfcc_cells{i, 2}, NUM_COEFF, num_features);

    if(cur_max > max_all)
        max_all = cur_max;
    end

    nn_output(i, :) = genreToVector(mfcc_cells{i, 3}(1), NUM_GENRES);


end

nn_input = nn_input./max_all;
save nn_input_training_data nn_input;
save nn_output_training_data nn_output
```

"format_mfcc_to_vector"

```
function [output cur_max] = format_mfcc_to_vector(mean_vec, cov_matrix, ...
NUM_COEFF, num_features)
% [output cur_max] = format_mfcc_to_vector(mean_vec, cov_matrix, NUM_COEFF,
% num_features
%
% output refers to the vector that takes the mean vector and top triangle
% of the cov_matrix an flatens it into 1 vector. This is designed for the
% mean-vector and cov-matrix of the mfcc data.
%
% Should be used for neural networks and SVM
    output = zeros(1, num_features);
    for j = 1:NUM_COEFF
        output(1, j) = mean_vec(1, j);
    end

    idx = NUM_COEFF+1;
    for j = 1:NUM_COEFF
        for k = j:NUM_COEFF
            output(1, idx) = cov_matrix(j, k);
            idx = idx + 1;
        end
    end
    cur_max = max(abs(output(1,:)));
end
```

"genreToVector.m"

```
function [output] = genreToVector(genre, NUM_GENRES)

genreMap = eye(4);
    if strcmp(genre, 'Classical')
        output = genreMap(1, :);
    elseif strcmp(genre, 'Jazz')
        output = genreMap(2, :);
    elseif strcmp(genre, 'Metal')
        output = genreMap(3, :);
    else
        output = genreMap(4, :);
    end

end
```

"Music_Classifier_NN.m"

```
clear
load nn_input_training_data;
load nn_output_training_data;

inputs = nn_input';
targets = nn_output';

% Create a Pattern Recognition Network
hiddenLayerSize = 25;
net = patternnet(hiddenLayerSize);


% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;


% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

% View the Network
view(net)

plotconfusion(targets, outputs)
```

APPENDIX B – 10 Genre Neural Network

"csv_to_data.m"

```
%test
clear all;

%training data
CSV_FILE_PATH_training =
'C:\Users\Batman\Desktop\Dropbox\CI_Project_Files\1000\gtzan_1000.csv'
NUM_BINS = 22;
NUM_FRAMES = 200;
NUM_COEFF = 15;
STEP_TIME = 0.02;

mfcc_training_data1000 = get_mfcc_features(CSV_FILE_PATH_training, NUM_BINS,
NUM_FRAMES, NUM_COEFF, STEP_TIME);
save mfcc_training_data1000;
```

"ge_mfcc_features.m"

```
function mfcc_cells = get_mfcc_features(CSV_FILE_PATH, NUM_BINS, NUM_FRAMES,
NUM_COEFF, STEP_TIME)
% MFCC_CELLS = GET_MFCC_FEATURES(CSV_FILE_NAME, NUM_BINS, NUM_FRAMES,
%              NUM_COEFF, STEP_TIME);
%
% MFCC_CELLS is a <# songs> by 2 cell matrix, where column 1 is the means
% of the mfcc and column 2 is the covariances of the mfcc. Each row is the
% mfcc data of a song.

fid = fopen(CSV_FILE_PATH);
csv_data = textscan(fid, '%s %s %s %s', 'Delimiter', '|', 'CollectOutput',
1);
csv_data = csv_data{1};
fclose(fid);

[numRows numCols] = size(csv_data);
mfcc_cells = cell(numRows-1, 1 + numCols);

for row = 2:numRows
    fprintf('song # %d\n', row);
    cur_song = csv_data(row, :);
    song_file_name = cell2mat(cur_song(1, 1));
    mfcc_mat = mfcc(song_file_name, NUM_BINS, NUM_FRAMES, NUM_COEFF,
STEP_TIME);
    mfcc_cells{row-1, 1} = mean(mfcc_mat);
    mfcc_cells{row-1, 2} = cov(mfcc_mat);
    for col = 3:numCols + 1
        mfcc_cells{row-1, col} = csv_data(row, col-1);
    end
    fprintf('\n');
end

end
```

"mfcc.m"

```
function MFCC_matrix = mfcc(song_file_name, NUM_BINS, NUM_FRAMES, NUM_COEFF,
STEP_TIME)
% MFCC_MATRIX = MFCC(song_file_name, NUM_BINS, NUM_FRAMES, NUM_COEFF,
STEP_TIME)
%
% MFCC_MATRIX is the NUM_FRAMES-by-NUM_COEFF matrix of MFCC coefficients.
% NUM_FRAMES is how many frames (20 ms samples) of the song to use.
% NUM_BINS is how many mel frequency bins to map each frame spectrum to.
% NUM_COEFF is how many mel coefficients to keep (NUM_COEFF <= NUM_BINS).
% STEP_TIME is how often (in seconds) to capture a 20ms frame.
%
%      - Mel-Frequency Ceptrum Coefficients -
% 1. Take the Fourier Transform of a 20-30ms windowed excerpt of a signal.
% 2. Map powers of spectrum onto mel scale, using triangular overlapping
% windows.
% 3. Take the logs of the powers at each of the mel frequencies.
% 4. Take the Discrete Cosine Transform of the list of mel log powers (as
% if it were a signal).
% 5. The MFCCs are the amplitudes of the resulting spectrum.


NUM_SEC = 10;  % take a full 10-second sample (then split into 20ms frames)

format = song_file_name(end-1:end);

if (strcmp(format, 'av'))
    slength = wavread(song_file_name, 'size');
    offset = floor(slength(1,1)/3);
    [~, fs] = wavread(song_file_name, 1);
    [s, fs] = wavread(song_file_name, [offset offset+fs*NUM_SEC]);
elseif (strcmp(format, 'au'))
    slength = auread(song_file_name, 'size');
    offset = floor(slength(1,1)/3);
    [~, fs] = auread(song_file_name, 1);
    [s, fs] = auread(song_file_name, [offset offset+fs*NUM_SEC]);
elseif (strcmp(format, 'p3'))
    slength = mp3read(song_file_name, 'size');
    offset = floor(slength(1,1)/3);
    [~, fs] = mp3read(song_file_name, [offset offset+1]);
    [s, fs] = mp3read(song_file_name, [offset offset+fs*NUM_SEC]);
else
    error('Wrong filename input. Use a .mp3, .wav, or .au audio file');
end


fft_len = 512;              % length of fft
frame_time = 0.020;         % # seconds/frame
frame_len = floor(fs*frame_time);  % # samples/frame

step_len = floor(fs*STEP_TIME);    % # samples/frame step
```

```
MFCC_matrix = zeros(NUM_FRAMES, NUM_COEFF);

window = hamming(frame_len);
stop = 1+floor(fft_len/2);  % 257

[x, ~] = size(s);
begin = floor(x/2);

for i = 1:NUM_FRAMES
    first = begin + (i-1)*step_len + 1;
    last = begin + (i-1)*step_len + frame_len;
    f = s(first:last);
    heightF = size(f, 1);
    widthF = size(f, 2);
    if (heightF > widthF)
        f = f.*window;
    else
        f = f'.*window;
    end
    f = fft(f, frame_len);
    mel_bins = melbankm(NUM_BINS, fft_len, fs);
    f = abs(f(1:stop)).^2;
    m = log10(mel_bins*f);
    m = dct(m);
    m = m(1:NUM_COEFF);
    MFCC_matrix(i, :) = m';
end


end
```

"melbankm.m"

```
function [x,mn,mx] = melbankm(p,n,fs,fl,fh,w)
%
%  function  [x,mn,mx] = melbankm(p,n,fs,fl,fh,w)
%
%  Determine matrix for a mel-spaced filterbank
%
%  Inputs:   p   number of filters in filterbank
%            n   length of fft
%            fs  sample rate in Hz
%            fl  low end of the lowest filter as a fraction of fs (default =
0)
%            fh  high end of highest filter as a fraction of fs (default =
0.5)
%            w   any sensible combination of the following:
%                't'  triangular shaped filters in mel domain (default)
%                'n'  hanning shaped filters in mel domain
%                'm'  hamming shaped filters in mel domain
%
%                'z'  highest and lowest filters taper down to zero (default)
%                'y'  lowest filter remains at 1 down to 0 frequency and
%                     highest filter remains at 1 up to nyquist freqency
%
%                If 'ty' or 'ny' is specified, the total power in the FFT
%                is preserved.
%
%  Outputs:  x   sparse matrix containing the filterbank amplitudes
%                If x is the only output argument then
%                  size(x)=[p,1+floor(n/2)]
%                otherwise
%                  size(x)=[p,mx-mn+1]
%            mn  the lowest fft bin with a non-zero coefficient
%            mx  the highest fft bin with a non-zero coefficient
%
%  Usage:    f = fft(s);                % mel cep over full freq range
%            x = melbankm(p,n,fs);
%            n2 = 1+floor(n/2);
%            z = log(x*abs(f(1:n2)).^2);
%            c = dct(z); c(1) = [];
%
%            f = fft(s);                % mel cep over part of freq range
%            [x,na,nb] = melbankm(p,n,fs);
%            z = log(x*(f(na:nb)).*conj(f(na:nb)));
%
% To plot filterbanks e.g.  plot(melbankm(20,256,8000)')


%      Copyright (C) Mike Brookes 1997
%
%      Last modified Tue May 12 16:15:28 1998
%
% VOICEBOX home page:
%          http://www.ee.ic.ac.uk/hp/staff/dmb/voicebox/voicebox.html
%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%
%   This program is free software; you can redistribute it and/or modify
%   it under the terms of the GNU General Public License as published by
%   the Free Software Foundation; either version 2 of the License, or
%   (at your option) any later version.
%
%   This program is distributed in the hope that it will be useful,
%   but WITHOUT ANY WARRANTY; without even the implied warranty of
%   MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
%   GNU General Public License for more details.
%
%   You can obtain a copy of the GNU General Public License from
%   ftp://prep.ai.mit.edu/pub/gnu/COPYING-2.0 or by writing to
%   Free Software Foundation, Inc.,675 Mass Ave, Cambridge, MA 02139, USA.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%

if nargin < 6
  w='tz';
  if nargin < 5
    fh=0.5;
    if nargin < 4
      fl=0;
    end
  end
end
f0=700/fs;
%f0=1000/fs;
fn2=floor(n/2);
lr=log((f0+fh)/(f0+fl))/(p+1);
% convert to fft bin numbers with 0 for DC term
bl=n*((f0+fl)*exp([0 1 p p+1]*lr)-f0);
b2=ceil(bl(2));
b3=floor(bl(3));
if any(w=='y')
  pf=log((f0+(b2:b3)/n)/(f0+fl))/lr;
  fp=floor(pf);
  r=[ones(1,b2) fp fp+1 p*ones(1,fn2-b3)];
  c=[1:b3+1 b2+1:fn2+1];
  v=2*[0.5 ones(1,b2-1) 1-pf+fp pf-fp ones(1,fn2-b3-1) 0.5];
  mn=1;
  mx=fn2+1;
else
  b1=floor(bl(1))+1;
  b4=min(fn2,ceil(bl(4)))-1;
  pf=log((f0+(b1:b4)/n)/(f0+fl))/lr;
  fp=floor(pf);
  pm=pf-fp;
  k2=b2-b1+1;
  k3=b3-b1+1;
  k4=b4-b1+1;
  r=[fp(k2:k4) 1+fp(1:k3)];
  c=[k2:k4 1:k3];
  v=2*[1-pm(k2:k4) pm(1:k3)];
  mn=b1+1;
  mx=b4+1;
```

```
end
if any(w=='n')
  v=1-cos(v*pi/2);
elseif any(w=='m')
  v=1-0.92/1.08*cos(v*pi/2);
end
if nargout > 1
  x=sparse(r,c,v);
else
  x=sparse(r,c+mn-1,v,p,1+fn2);
end
```

"NN_Input_Generator.m"

```
%generates the input for Neural Network Analysis
load mfcc_training_data1000

NUM_COEFF = 15;
NUM_GENRES = 10;

mfcc_cells = mfcc_training_data1000;


m = size(mfcc_cells, 1);

num_features = NUM_COEFF + (1 + NUM_COEFF)*NUM_COEFF/2;

nn_input = zeros(m, num_features);
nn_output = zeros(m, NUM_GENRES);


max_all = 0;

%scaling input
for i = 1 : m
    [nn_input(i, :) cur_max] = format_mfcc_to_vector(mfcc_cells{i, 1}, ...
        mfcc_cells{i, 2}, NUM_COEFF, num_features);

    if(cur_max > max_all)
        max_all = cur_max;
    end

    nn_output(i, :) = genreToVector(mfcc_cells{i, 3}(1), NUM_GENRES);


end

nn_input = nn_input./max_all;
save nn_input_training_data nn_input;
save nn_output_training_data nn_output
```

"format_mfcc_to_vector"

```
function [output cur_max] = format_mfcc_to_vector(mean_vec, cov_matrix, ...
NUM_COEFF, num_features)
% [output cur_max] = format_mfcc_to_vector(mean_vec, cov_matrix, NUM_COEFF,
% num_features
%
% output refers to the vector that takes the mean vector and top triangle
% of the cov_matrix an flatens it into 1 vector. This is designed for the
% mean-vector and cov-matrix of the mfcc data.
%
% Should be used for neural networks and SVM
    output = zeros(1, num_features);
    for j = 1:NUM_COEFF
        output(1, j) = mean_vec(1, j);
    end

    idx = NUM_COEFF+1;
    for j = 1:NUM_COEFF
        for k = j:NUM_COEFF
            output(1, idx) = cov_matrix(j, k);
            idx = idx + 1;
        end
    end
    cur_max = max(abs(output(1,:)));
end
```

"genreToVector.m"

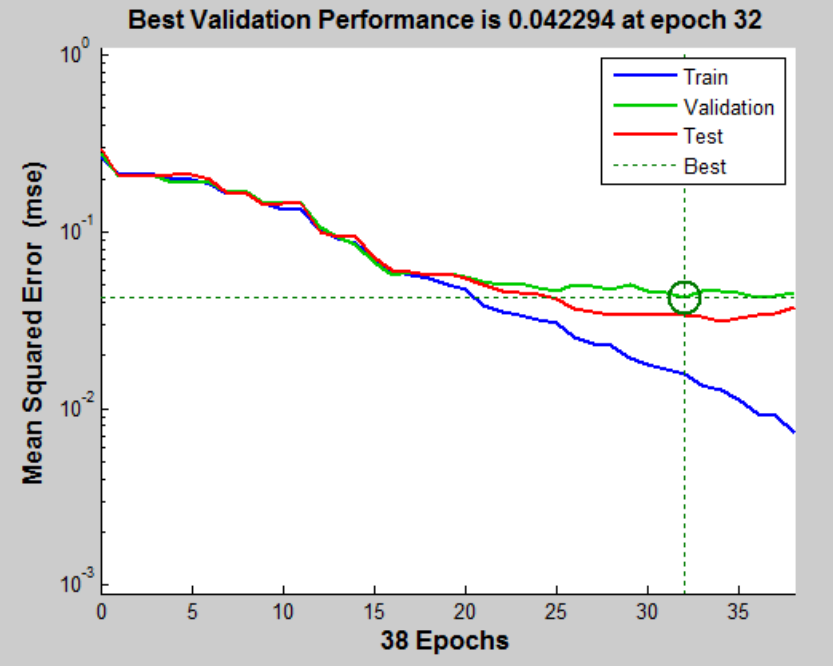
```
function [output] = genreToVector(genre, NUM_GENRES)

genreMap = eye(10);
    if strcmp(genre, 'Blues')
        output = genreMap(1, :);
    elseif strcmp(genre, 'Classical')
        output = genreMap(2, :);
    elseif strcmp(genre, 'Country')
        output = genreMap(3, :);
    elseif strcmp(genre, 'Disco')
        output = genreMap(4, :);
    elseif strcmp(genre, 'Hiphop')
        output = genreMap(5, :);
    elseif strcmp(genre, 'Jazz')
        output = genreMap(6, :);
    elseif strcmp(genre, 'Metal')
        output = genreMap(7, :);
    elseif strcmp(genre, 'Pop')
        output = genreMap(8, :);
    elseif strcmp(genre, 'Reggae')
        output = genreMap(9, :);
    else
        output = genreMap(10, :);
    end

end
```

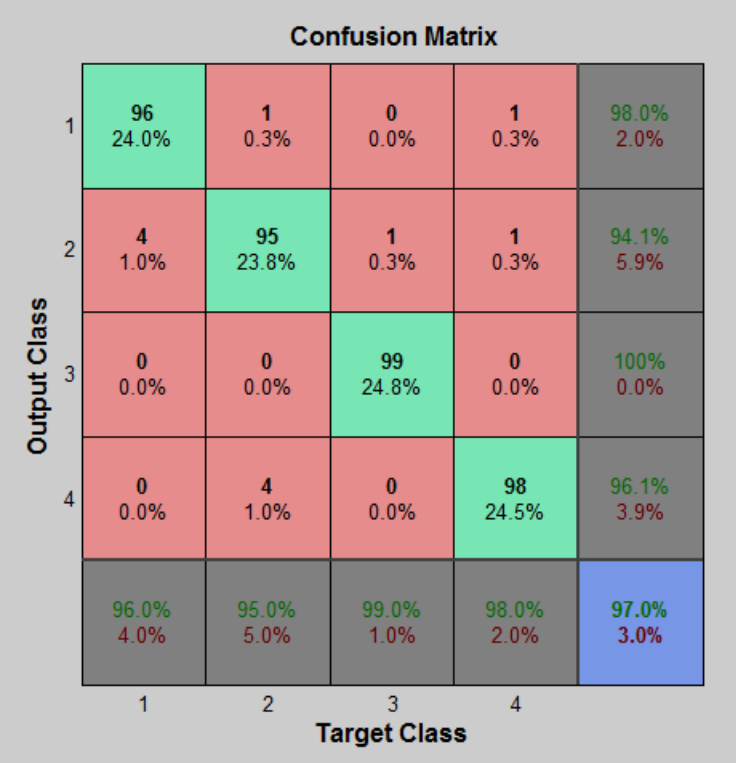
"Music_Classifier_NN.m"

```
clear
load nn_input_training_data;
load nn_output_training_data;

inputs = nn_input';
targets = nn_output';

% Create a Pattern Recognition Network
hiddenLayerSize = 25;
net = patternnet(hiddenLayerSize);


% Setup Division of Data for Training, Validation, Testing
net.divideParam.trainRatio = 70/100;
net.divideParam.valRatio = 15/100;
net.divideParam.testRatio = 15/100;


% Train the Network
[net,tr] = train(net,inputs,targets);

% Test the Network
outputs = net(inputs);
errors = gsubtract(targets,outputs);
performance = perform(net,targets,outputs)

% View the Network
view(net)

plotconfusion(targets, outputs)
```

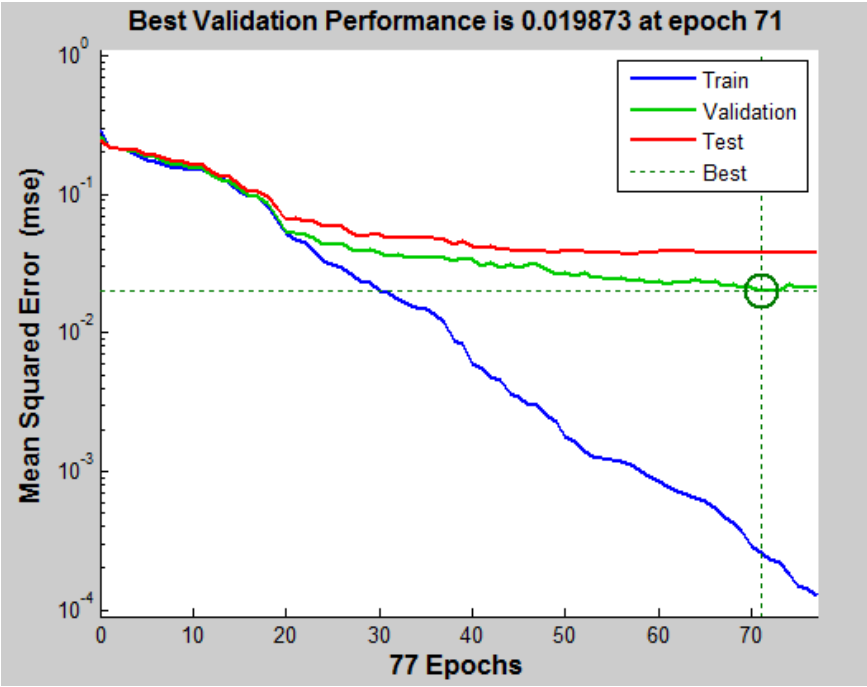
APPENDIX C – 4 Genre Neural Network Data

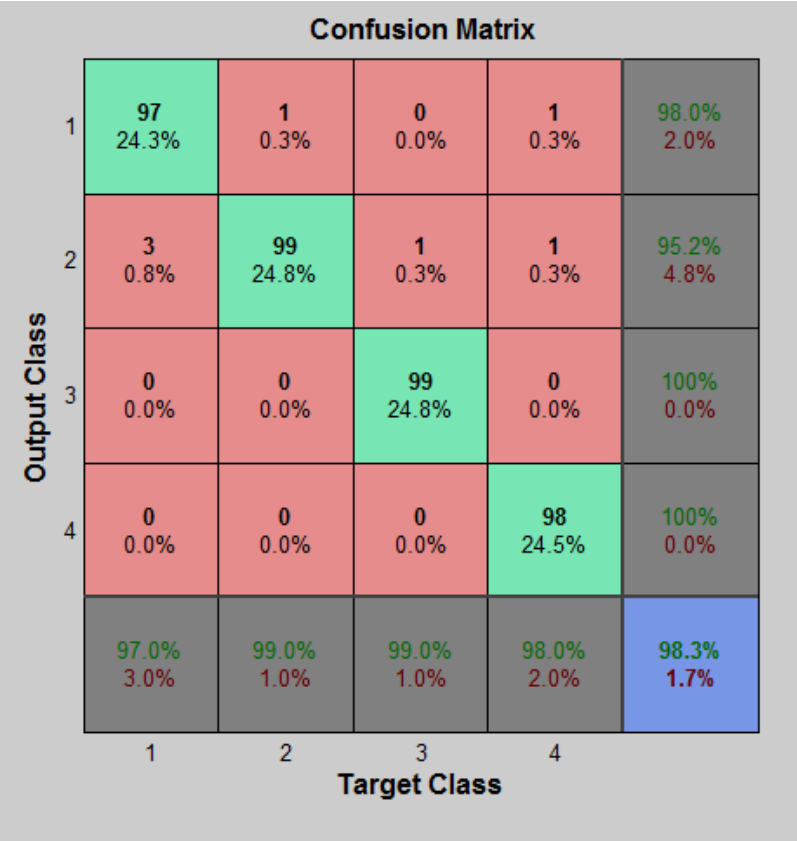
Run 1



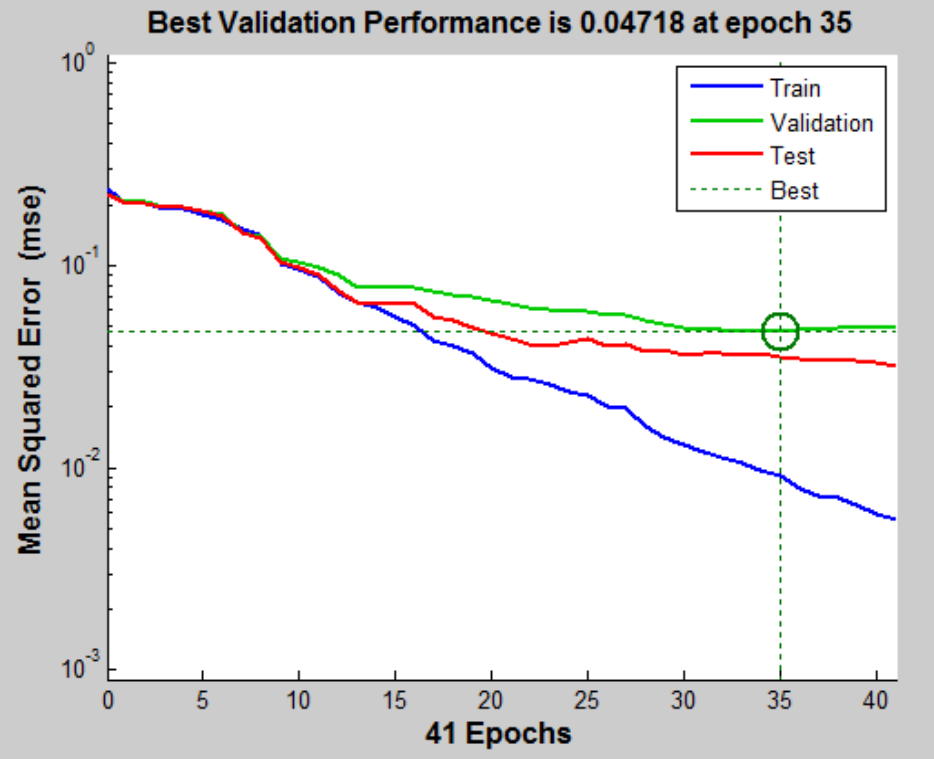


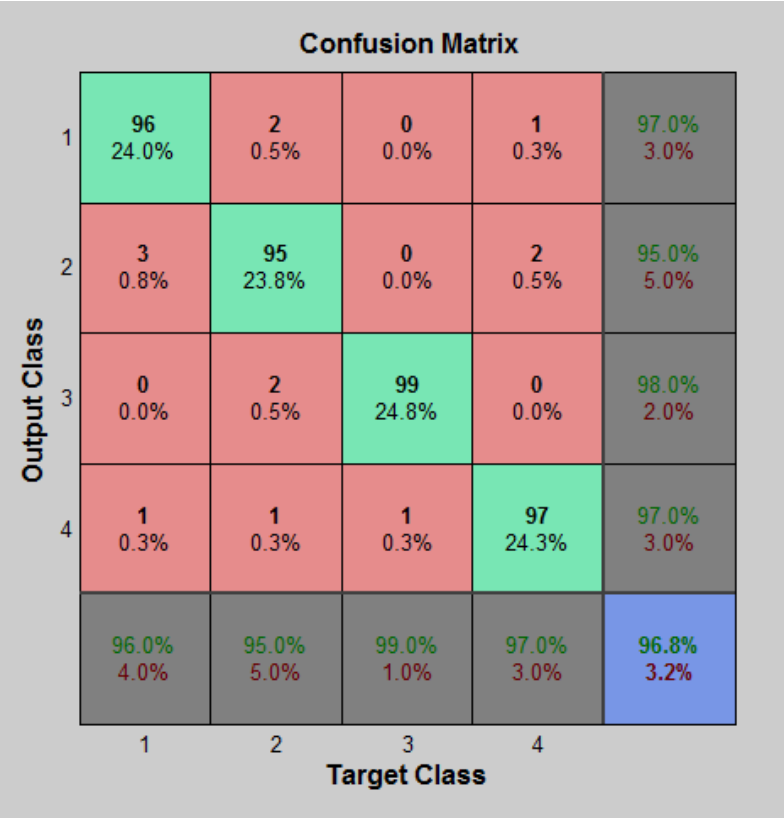
Run 2


Best Validation Performance is 0.019873 at epoch 71
Mean Squared Error (mse)
77 Epochs
Train
Validation
Test
Best

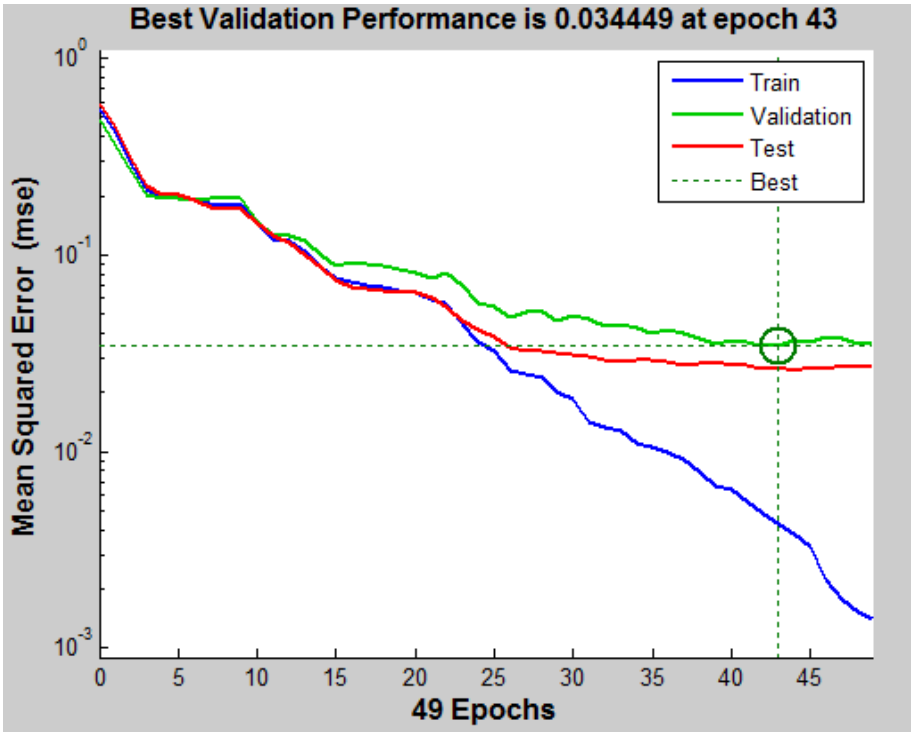

Confusion Matrix
Output Class
Target Class
97 24.3%
1 0.3%
0 0.0%
1 0.3%
98.0% 2.0%
3 0.8%
99 24.8%
1 0.3%
1 0.3%
95.2% 4.8%
0 0.0%
0 0.0%
99 24.8%
0 0.0%
100% 0.0%
0 0.0%
0 0.0%
0 0.0%
98 24.5%
100% 0.0%
97.0% 3.0%
99.0% 1.0%
99.0% 1.0%
98.0% 2.0%
98.3% 1.7%

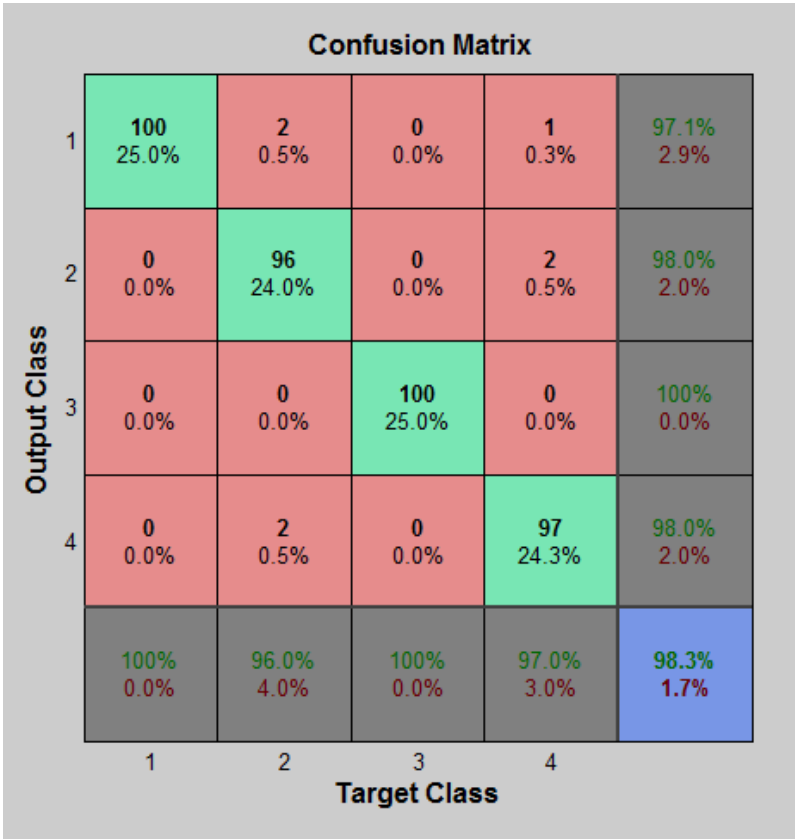
Run 3


Best Validation Performance is 0.04718 at epoch 35
Mean Squared Error (mse)
41 Epochs
Train
Validation
Test
Best

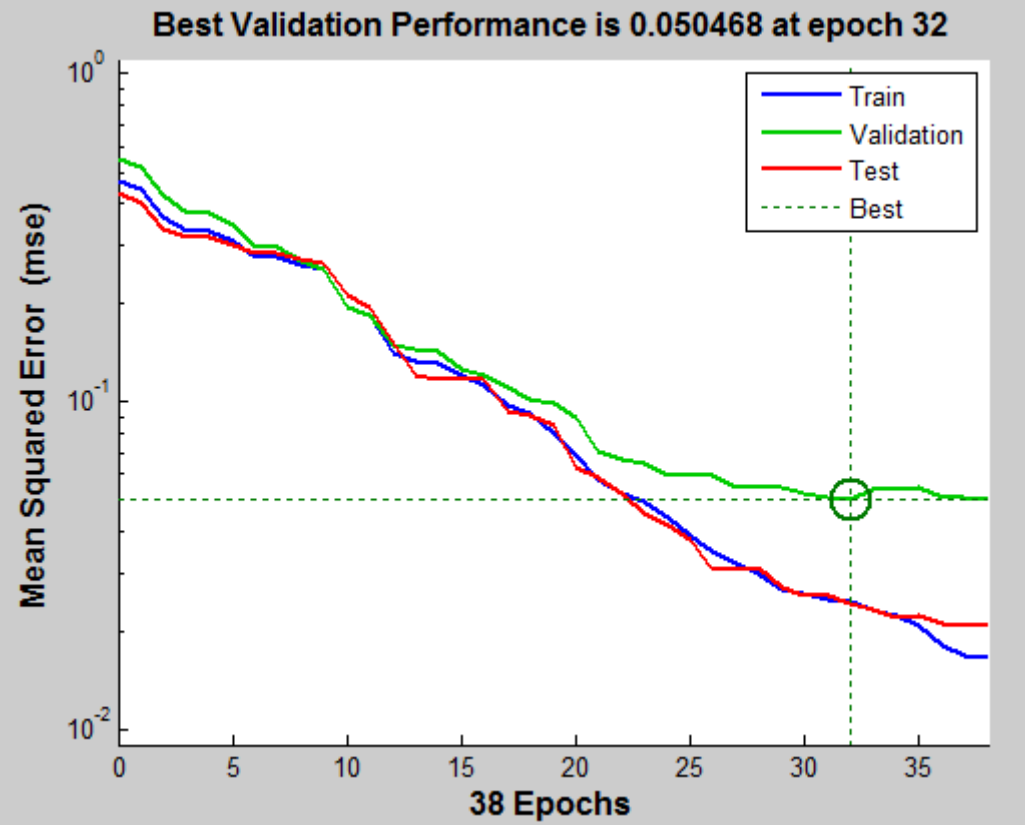

Confusion Matrix
Output Class
Target Class
96 24.0%
2 0.5%
0 0.0%
1 0.3%
97.0% 3.0%
3 0.8%
95 23.8%
0 0.0%
2 0.5%
95.0% 5.0%
0 0.0%
2 0.5%
99 24.8%
0 0.0%
98.0% 2.0%
1 0.3%
1 0.3%
1 0.3%
97 24.3%
97.0% 3.0%
96.0% 4.0%
95.0% 5.0%
99.0% 1.0%
97.0% 3.0%
96.8% 3.2%

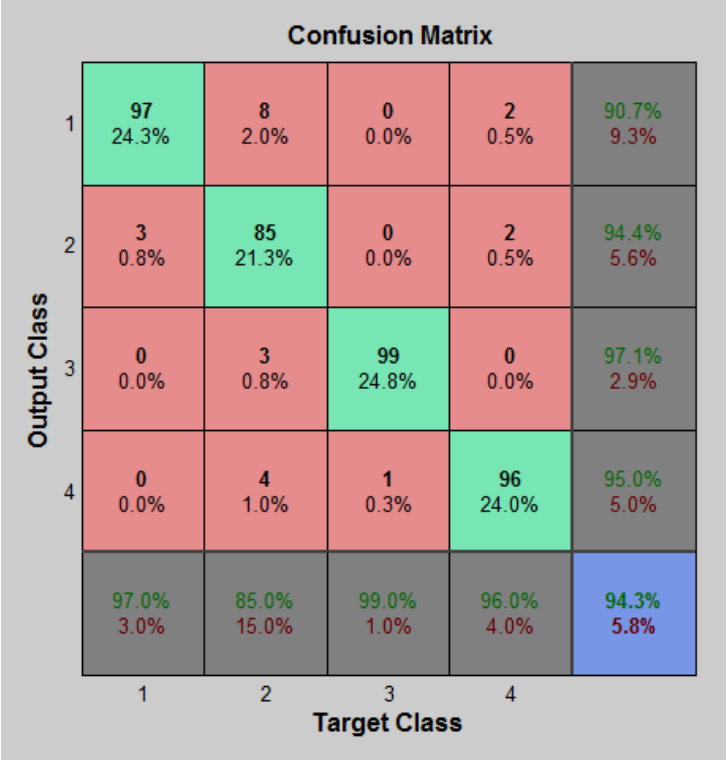
Run 4


Best Validation Performance is 0.034449 at epoch 43
Train
Validation
Test
Best
Mean Squared Error (mse)
49 Epochs


Confusion Matrix
Output Class
Target Class
100 25.0%
2 0.5%
0 0.0%
1 0.3%
97.1% 2.9%
0 0.0%
96 24.0%
0 0.0%
2 0.5%
98.0% 2.0%
0 0.0%
0 0.0%
100 25.0%
0 0.0%
100% 0.0%
0 0.0%
2 0.5%
0 0.0%
97 24.3%
98.0% 2.0%
100% 0.0%
96.0% 4.0%
100% 0.0%
97.0% 3.0%
98.3% 1.7%

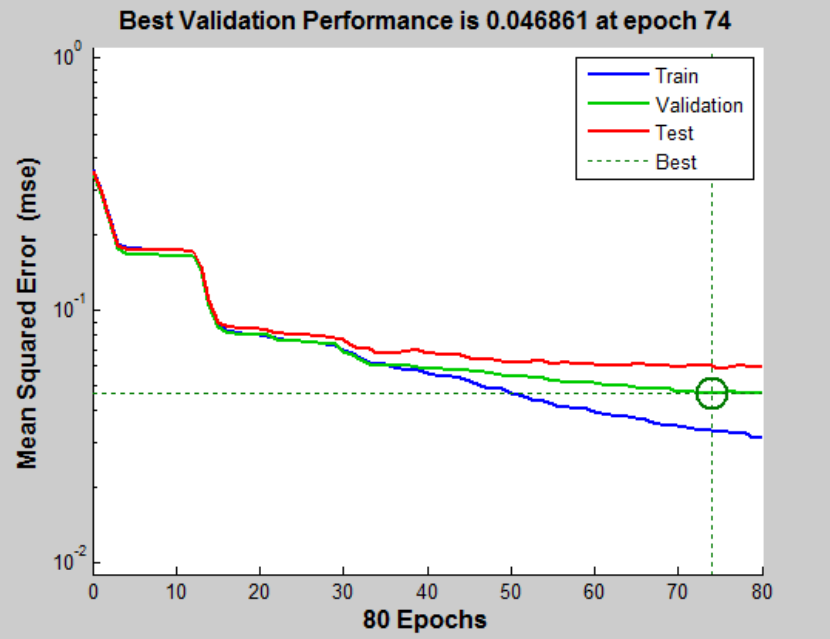
Run 5

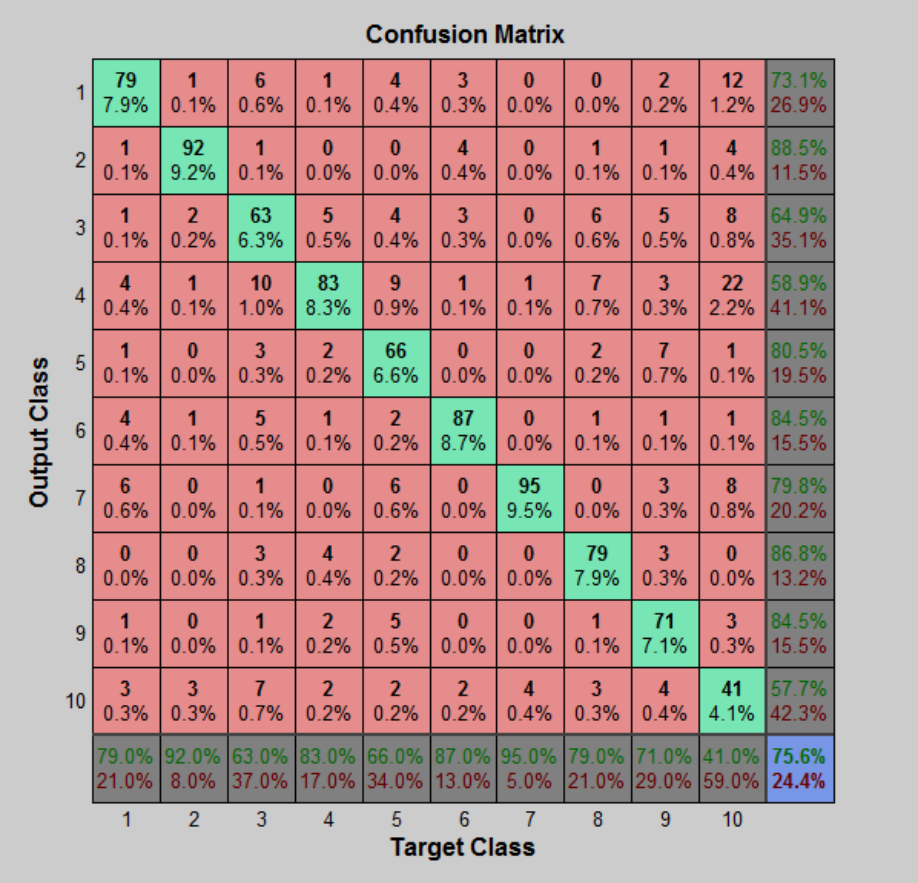

Best Validation Performance is 0.050468 at epoch 32
Train
Validation
Test
Best
Mean Squared Error (mse)
38 Epochs


Confusion Matrix
Output Class
Target Class
97 24.3%
8 2.0%
0 0.0%
2 0.5%
90.7% 9.3%
3 0.8%
85 21.3%
0 0.0%
2 0.5%
94.4% 5.6%
0 0.0%
3 0.8%
99 24.8%
0 0.0%
97.1% 2.9%
0 0.0%
4 1.0%
1 0.3%
96 24.0%
95.0% 5.0%
97.0% 3.0%
85.0% 15.0%
99.0% 1.0%
96.0% 4.0%
94.3% 5.8%

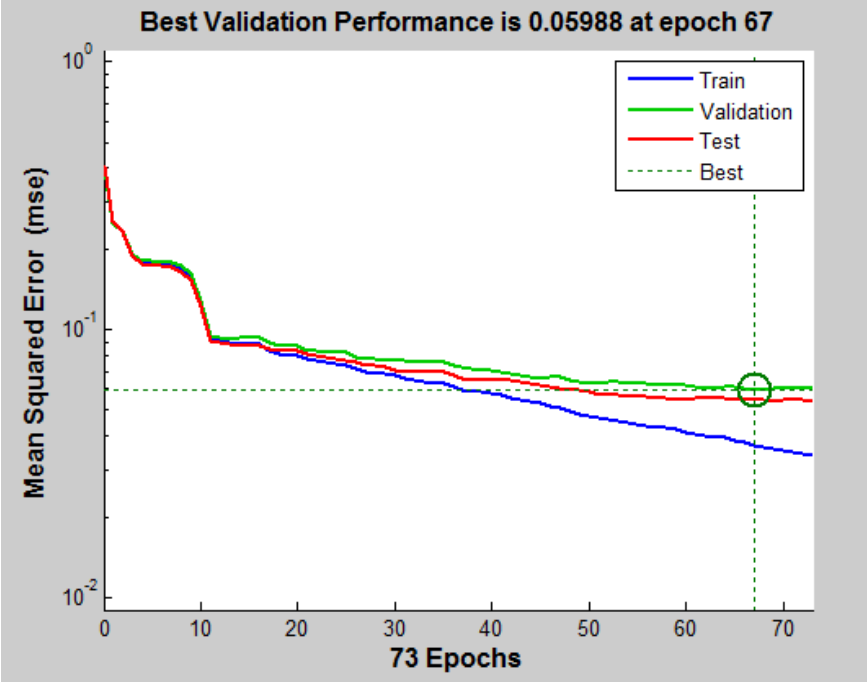
APPENDIX D – 10 Genre Neural Network Data

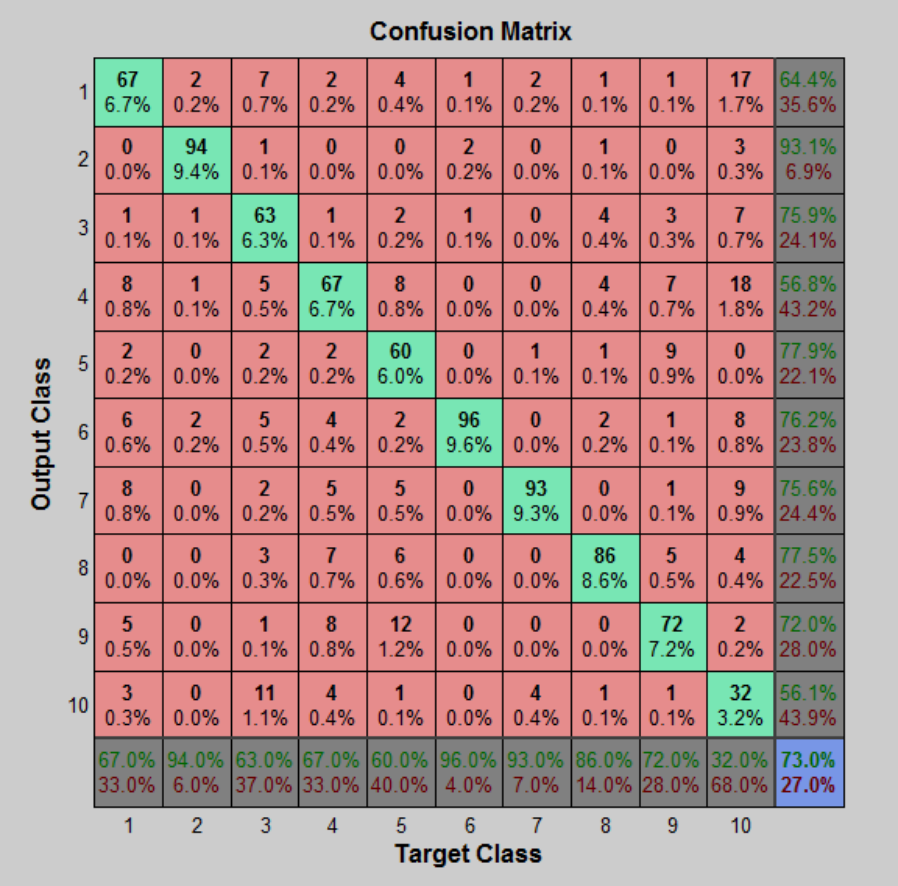
Run 1



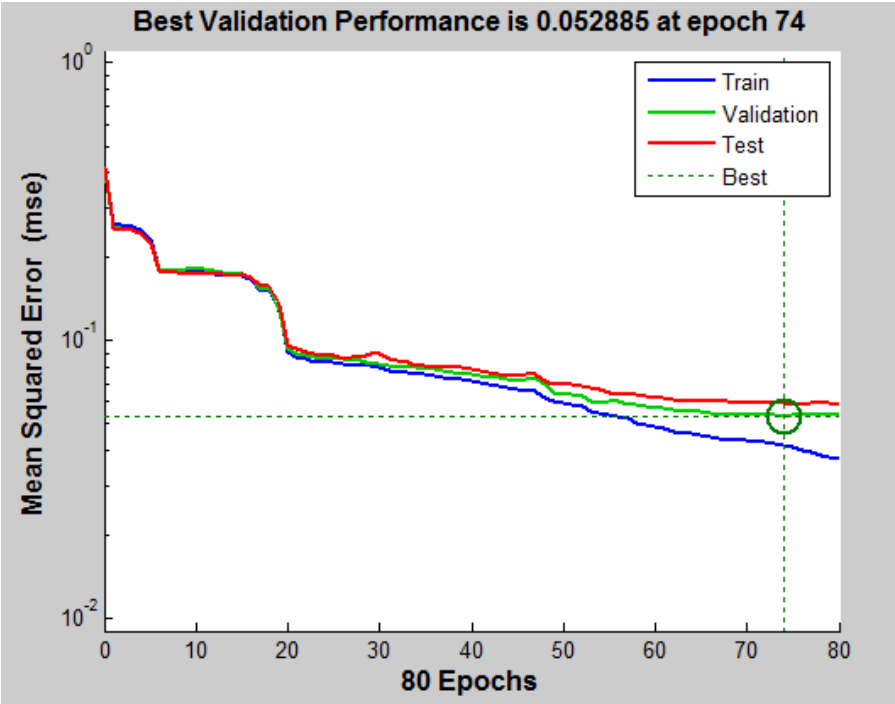


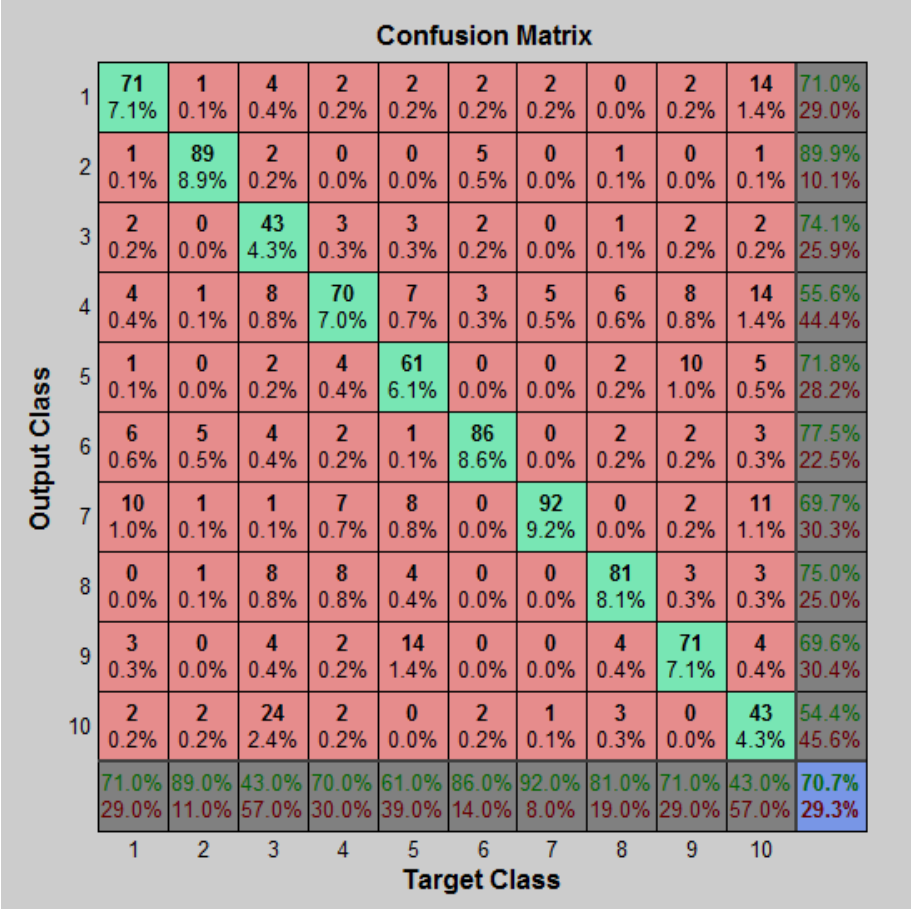
Run 2


Best Validation Performance is 0.05988 at epoch 67
Mean Squared Error (mse)
73 Epochs
Train
Validation
Test
Best

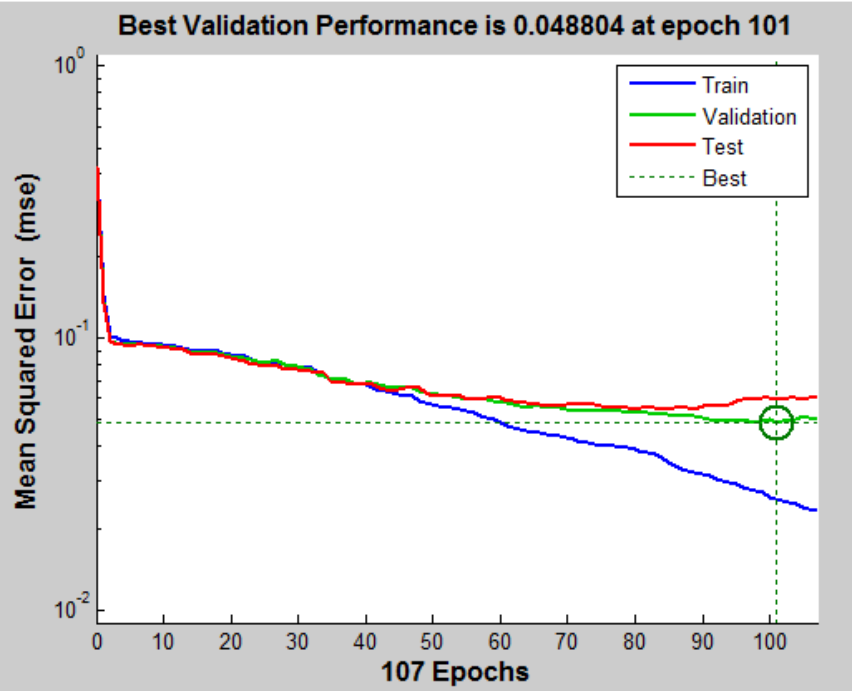

Confusion Matrix
Output Class
Target Class
67 6.7% | 2 0.2% | 7 0.7% | 2 0.2% | 4 0.4% | 1 0.1% | 2 0.2% | 1 0.1% | 1 0.1% | 17 1.7% | 64.4% 35.6%
0 0.0% | 94 9.4% | 1 0.1% | 0 0.0% | 0 0.0% | 2 0.2% | 0 0.0% | 1 0.1% | 0 0.0% | 3 0.3% | 93.1% 6.9%
1 0.1% | 1 0.1% | 63 6.3% | 1 0.1% | 2 0.2% | 1 0.1% | 0 0.0% | 4 0.4% | 3 0.3% | 7 0.7% | 75.9% 24.1%
8 0.8% | 1 0.1% | 5 0.5% | 67 6.7% | 8 0.8% | 0 0.0% | 0 0.0% | 4 0.4% | 7 0.7% | 18 1.8% | 56.8% 43.2%
2 0.2% | 0 0.0% | 2 0.2% | 2 0.2% | 60 6.0% | 0 0.0% | 1 0.1% | 1 0.1% | 9 0.9% | 0 0.0% | 77.9% 22.1%
6 0.6% | 2 0.2% | 5 0.5% | 4 0.4% | 2 0.2% | 96 9.6% | 0 0.0% | 2 0.2% | 1 0.1% | 8 0.8% | 76.2% 23.8%
8 0.8% | 0 0.0% | 2 0.2% | 5 0.5% | 5 0.5% | 0 0.0% | 93 9.3% | 0 0.0% | 1 0.1% | 9 0.9% | 75.6% 24.4%
0 0.0% | 0 0.0% | 3 0.3% | 7 0.7% | 6 0.6% | 0 0.0% | 0 0.0% | 86 8.6% | 5 0.5% | 4 0.4% | 77.5% 22.5%
5 0.5% | 0 0.0% | 1 0.1% | 8 0.8% | 12 1.2% | 0 0.0% | 0 0.0% | 0 0.0% | 72 7.2% | 2 0.2% | 72.0% 28.0%
3 0.3% | 0 0.0% | 11 1.1% | 4 0.4% | 1 0.1% | 0 0.0% | 4 0.4% | 1 0.1% | 1 0.1% | 32 3.2% | 56.1% 43.9%
67.0% 33.0% | 94.0% 6.0% | 63.0% 37.0% | 67.0% 33.0% | 60.0% 40.0% | 96.0% 4.0% | 93.0% 7.0% | 86.0% 14.0% | 72.0% 28.0% | 32.0% 68.0% | 73.0% 27.0%
1 2 3 4 5 6 7 8 9 10

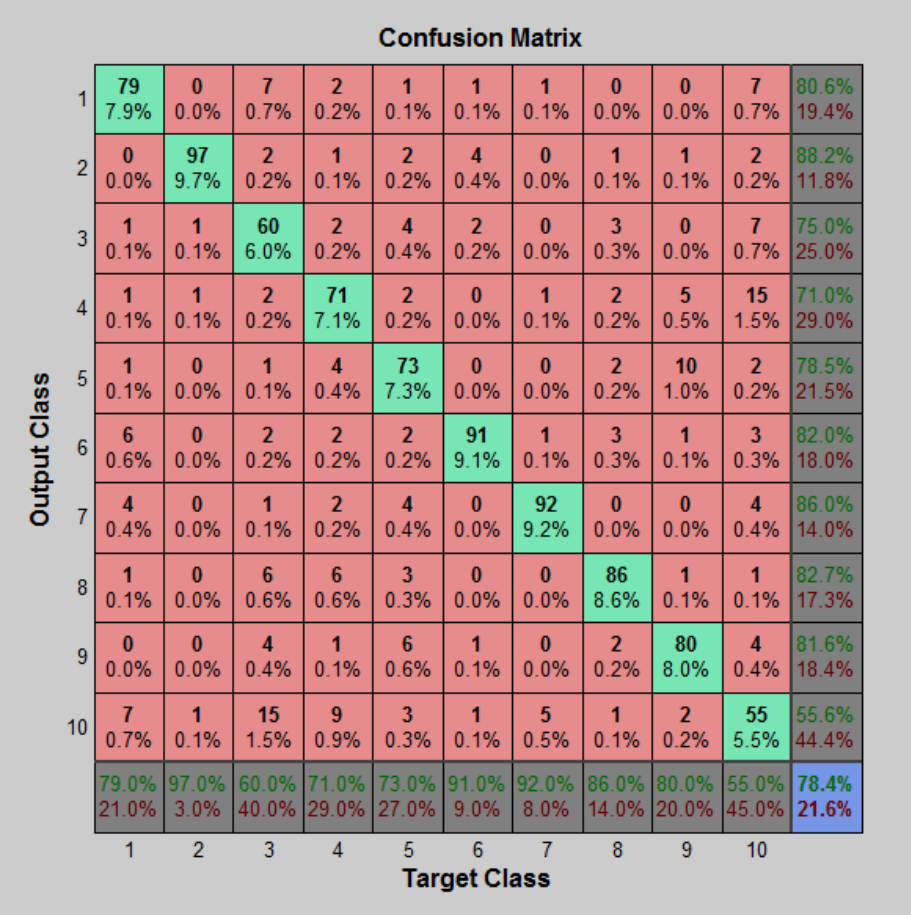
Run 3


Best Validation Performance is 0.052885 at epoch 74
Train
Validation
Test
Best
Mean Squared Error (mse)
80 Epochs

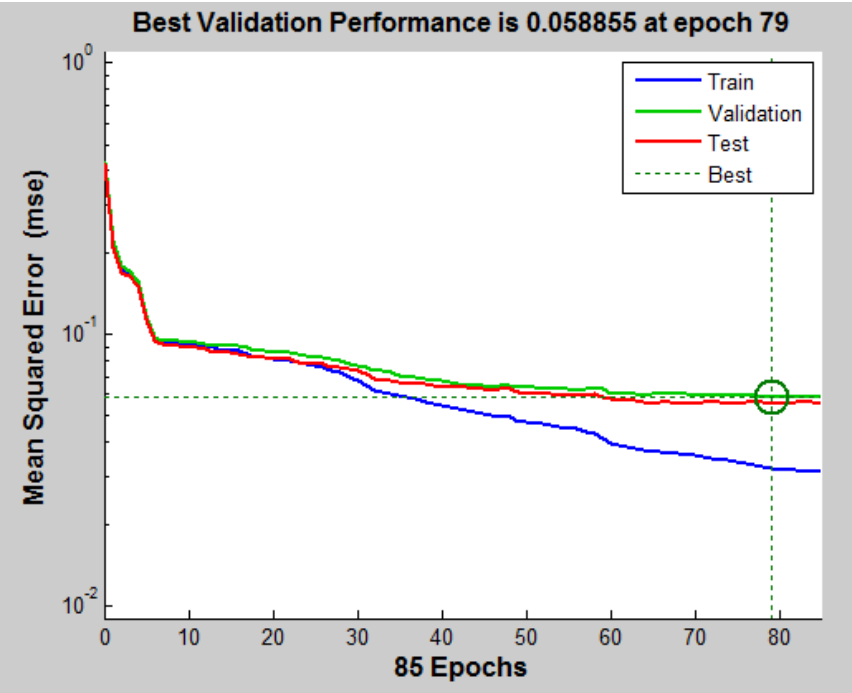

Confusion Matrix
Output Class
Target Class
71 7.1% | 1 0.1% | 4 0.4% | 2 0.2% | 2 0.2% | 2 0.2% | 2 0.2% | 0 0.0% | 2 0.2% | 14 1.4% | 71.0% 29.0%
1 0.1% | 89 8.9% | 2 0.2% | 0 0.0% | 0 0.0% | 5 0.5% | 0 0.0% | 1 0.1% | 0 0.0% | 1 0.1% | 89.9% 10.1%
2 0.2% | 0 0.0% | 43 4.3% | 3 0.3% | 3 0.3% | 2 0.2% | 0 0.0% | 1 0.1% | 2 0.2% | 2 0.2% | 74.1% 25.9%
4 0.4% | 1 0.1% | 8 0.8% | 70 7.0% | 7 0.7% | 3 0.3% | 5 0.5% | 6 0.6% | 8 0.8% | 14 1.4% | 55.6% 44.4%
1 0.1% | 0 0.0% | 2 0.2% | 4 0.4% | 61 6.1% | 0 0.0% | 0 0.0% | 2 0.2% | 10 1.0% | 5 0.5% | 71.8% 28.2%
6 0.6% | 5 0.5% | 4 0.4% | 2 0.2% | 1 0.1% | 86 8.6% | 0 0.0% | 2 0.2% | 2 0.2% | 3 0.3% | 77.5% 22.5%
10 1.0% | 1 0.1% | 1 0.1% | 7 0.7% | 8 0.8% | 0 0.0% | 92 9.2% | 0 0.0% | 2 0.2% | 11 1.1% | 69.7% 30.3%
0 0.0% | 1 0.1% | 8 0.8% | 8 0.8% | 4 0.4% | 0 0.0% | 0 0.0% | 81 8.1% | 3 0.3% | 3 0.3% | 75.0% 25.0%
3 0.3% | 0 0.0% | 4 0.4% | 2 0.2% | 14 1.4% | 0 0.0% | 0 0.0% | 4 0.4% | 71 7.1% | 4 0.4% | 69.6% 30.4%
2 0.2% | 2 0.2% | 24 2.4% | 2 0.2% | 0 0.0% | 2 0.2% | 1 0.1% | 3 0.3% | 0 0.0% | 43 4.3% | 54.4% 45.6%
71.0% 29.0% | 89.0% 11.0% | 43.0% 57.0% | 70.0% 30.0% | 61.0% 39.0% | 86.0% 14.0% | 92.0% 8.0% | 81.0% 19.0% | 71.0% 29.0% | 43.0% 57.0% | 70.7% 29.3%
1 2 3 4 5 6 7 8 9 10

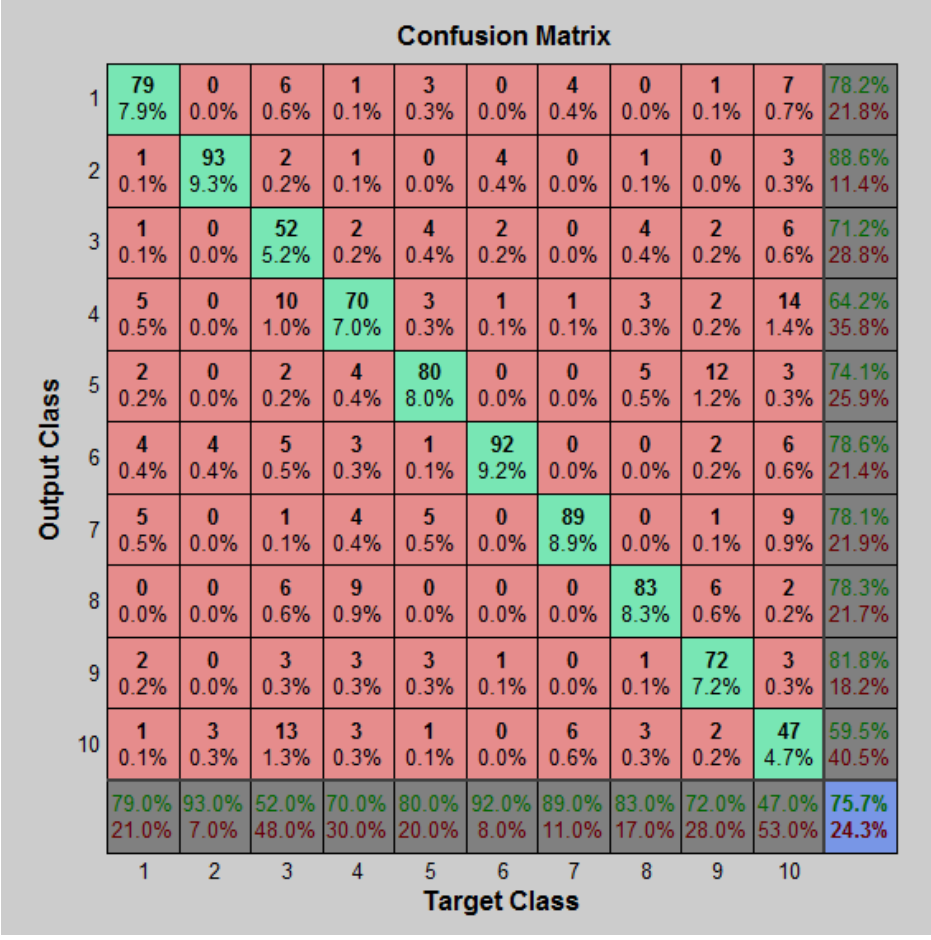
Run 4


Best Validation Performance is 0.048804 at epoch 101
Train
Validation
Test
Best
Mean Squared Error (mse)
107 Epochs


Confusion Matrix
Output Class
Target Class
79 7.9% | 0 0.0% | 7 0.7% | 2 0.2% | 1 0.1% | 1 0.1% | 1 0.1% | 0 0.0% | 0 0.0% | 7 0.7% | 80.6% 19.4%
0 0.0% | 97 9.7% | 2 0.2% | 1 0.1% | 2 0.2% | 4 0.4% | 0 0.0% | 1 0.1% | 1 0.1% | 2 0.2% | 88.2% 11.8%
1 0.1% | 1 0.1% | 60 6.0% | 2 0.2% | 4 0.4% | 2 0.2% | 0 0.0% | 3 0.3% | 0 0.0% | 7 0.7% | 75.0% 25.0%
1 0.1% | 1 0.1% | 2 0.2% | 71 7.1% | 2 0.2% | 0 0.0% | 1 0.1% | 2 0.2% | 5 0.5% | 15 1.5% | 71.0% 29.0%
1 0.1% | 0 0.0% | 1 0.1% | 4 0.4% | 73 7.3% | 0 0.0% | 0 0.0% | 2 0.2% | 10 1.0% | 2 0.2% | 78.5% 21.5%
6 0.6% | 0 0.0% | 2 0.2% | 2 0.2% | 2 0.2% | 91 9.1% | 1 0.1% | 3 0.3% | 1 0.1% | 3 0.3% | 82.0% 18.0%
4 0.4% | 0 0.0% | 1 0.1% | 2 0.2% | 4 0.4% | 0 0.0% | 92 9.2% | 0 0.0% | 0 0.0% | 4 0.4% | 86.0% 14.0%
1 0.1% | 0 0.0% | 6 0.6% | 6 0.6% | 3 0.3% | 0 0.0% | 0 0.0% | 86 8.6% | 1 0.1% | 1 0.1% | 82.7% 17.3%
0 0.0% | 0 0.0% | 4 0.4% | 1 0.1% | 6 0.6% | 1 0.1% | 0 0.0% | 2 0.2% | 80 8.0% | 4 0.4% | 81.6% 18.4%
7 0.7% | 1 0.1% | 15 1.5% | 9 0.9% | 3 0.3% | 1 0.1% | 5 0.5% | 1 0.1% | 2 0.2% | 55 5.5% | 55.6% 44.4%
79.0% 21.0% | 97.0% 3.0% | 60.0% 40.0% | 71.0% 29.0% | 73.0% 27.0% | 91.0% 9.0% | 92.0% 8.0% | 86.0% 14.0% | 80.0% 20.0% | 55.0% 45.0% | 78.4% 21.6%

Run 5


Best Validation Performance is 0.058855 at epoch 79
Train
Validation
Test
Best
Mean Squared Error (mse)
85 Epochs


Confusion Matrix
Output Class
Target Class
79 7.9% | 0 0.0% | 6 0.6% | 1 0.1% | 3 0.3% | 0 0.0% | 4 0.4% | 0 0.0% | 1 0.1% | 7 0.7% | 78.2% 21.8%
1 0.1% | 93 9.3% | 2 0.2% | 1 0.1% | 0 0.0% | 4 0.4% | 0 0.0% | 1 0.1% | 0 0.0% | 3 0.3% | 88.6% 11.4%
1 0.1% | 0 0.0% | 52 5.2% | 2 0.2% | 4 0.4% | 2 0.2% | 0 0.0% | 4 0.4% | 2 0.2% | 6 0.6% | 71.2% 28.8%
5 0.5% | 0 0.0% | 10 1.0% | 70 7.0% | 3 0.3% | 1 0.1% | 1 0.1% | 3 0.3% | 2 0.2% | 14 1.4% | 64.2% 35.8%
2 0.2% | 0 0.0% | 2 0.2% | 4 0.4% | 80 8.0% | 0 0.0% | 0 0.0% | 5 0.5% | 12 1.2% | 3 0.3% | 74.1% 25.9%
4 0.4% | 4 0.4% | 5 0.5% | 3 0.3% | 1 0.1% | 92 9.2% | 0 0.0% | 0 0.0% | 2 0.2% | 6 0.6% | 78.6% 21.4%
5 0.5% | 0 0.0% | 1 0.1% | 4 0.4% | 5 0.5% | 0 0.0% | 89 8.9% | 0 0.0% | 1 0.1% | 9 0.9% | 78.1% 21.9%
0 0.0% | 0 0.0% | 6 0.6% | 9 0.9% | 0 0.0% | 0 0.0% | 0 0.0% | 83 8.3% | 6 0.6% | 2 0.2% | 78.3% 21.7%
2 0.2% | 0 0.0% | 3 0.3% | 3 0.3% | 3 0.3% | 1 0.1% | 0 0.0% | 1 0.1% | 72 7.2% | 3 0.3% | 81.8% 18.2%
1 0.1% | 3 0.3% | 13 1.3% | 3 0.3% | 1 0.1% | 0 0.0% | 6 0.6% | 3 0.3% | 2 0.2% | 47 4.7% | 59.5% 40.5%
79.0% 21.0% | 93.0% 7.0% | 52.0% 48.0% | 70.0% 30.0% | 80.0% 20.0% | 92.0% 8.0% | 89.0% 11.0% | 83.0% 17.0% | 72.0% 28.0% | 47.0% 53.0% | 75.7% 24.3%
1 2 3 4 5 6 7 8 9 10