APPENDIX:I

// DATABASE GENERATION OF EXTRACTED FEATURE USING DAUBECHIES D4 COEFFICIENTS OF DISCRETE WAVELET TRANSFORM

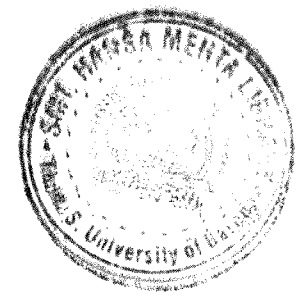
```
import java.util.*;
import java.awt.*;
import java.awt.image.*;
import java.lang.Math.*;
import utils.*;
import objects.*;
import java.io.*;

public class DaubDatabase1 {

public static void main(String[] args){
            int coeff = 50;

            if(args.length < 2 ){
                  System.out.println("usage : java DCTDatabase <source path>
<destination path>");
            }
            String sourceDir = args[0];
            String destDir = args[1];

            File sdir = new File(sourceDir);
            String[] fileNames = sdir.list();
            try{
                  FileWriter fw = new
FileWriter(destDir+"\\daubCoeff.dat",true);
                  PrintWriter pw = new PrintWriter(fw);
                  int namelength;
                  GlyDaub d = new GlyDaub();
                  for(int i=0;i<fileNames.length;i++){
                        System.out.println("fileNames["+i+"]="+fileNames[i]);

                        namelength = fileNames[i].length();
                        String extension =fileNames[i].substring(namelength-
4);

      if(!(extension.equalsIgnoreCase(".png")||extension.equalsIgnoreCase(".j
pg")||extension.equalsIgnoreCase(".bmp"))){
                              System.out.println("extension = "+extension);
                              continue;

                        }
                        int pos=0;
                        double[] a;
                        double[] c;
                        double[][] ca;
                        int maxcoeff=1024;
                        double[] b = new double[maxcoeff];
                        int length;
```

```
                        byte[] image;
                        FileInputStream in = new
FileInputStream(fileNames[i]);
                        BufferedInputStream reader = new
BufferedInputStream(in);
                        length = reader.available();
                        image = new byte[length];
                        reader.read(image, 0, length);
                        reader.close();
                        a=new double[maxcoeff];
                        for(int ii=0;ii<image.length;ii++){
                              a[ii] = (double)image[ii];
                        }

                        for ( int i1 = 0;i1<a.length;i1++)
                        {
                              if(a[i1] > 0){
                                    a[i1] = 1;}
                              else{
                                    a[i1] = 0;}
                        }
                        c = new double[maxcoeff];
                        for(int i2=0;i2<a.length;i2++)
                        {
                              c[i2] = a[i2];
                        }

                        ca = new double[maxcoeff][2];
                        b = d.daubTrans(a);
                        int count = 0;
                        for(int i3=0;i3<maxcoeff;i3++){
                              ca[i3][0] = b[i3];
                             ca[i3][1] = i3;

                        }
                        ca = GlyDaub.sort2D(ca,maxcoeff,2);
                        for(int j=0;j<maxcoeff;j++)
                              b[j] = 0;

                        for(int j1=0;j1<maxcoeff;j1++)
                        {
                          if(j1<coeff)
                          {
                              pos = (int)ca[j1][1] ;
                              b[pos] = ca[j1][0];
                          }
                      }
                      for(int i5=0;i5<maxcoeff;i5++)
                              b = d.invDaubTrans(b);
                              for(int i5 =0;i5<maxcoeff;i5++)
                              {
                                    if(b[i5] > 0.475){
                                          b[i5] = 1;}
                                    else{
                                          b[i5] = 0;}
                              }
                              for(int j3=0;j3<maxcoeff;j3++){
```

```
                                        if(b[j3] == c[j3])
                                        {
                                              count++;
                                        }

                                  }
                                  System.out.println("count"+count);
                                  System.out.println("The success rate
is"+(count*100)/maxcoeff);

                                  for(int fi=0;fi<1024;fi++){
                                        System.out.println("b["+fi+"]="+b[fi]);
                                        pw.print(b[fi]+" ");
                                  }

                                  pw.println();

                            }
                        pw.close();
                        fw.close();

                  }catch(IOException e){System.out.println("I/O Exception in
DCTDatabase.....");}
                  System.exit(0);

                  }
            }
```

// Testing program : Decompose the image and Reconstruction with full or less coefficients.

```
public class GlyDcomp6{

        D6 gc = new D6();


        public double[][] lmdcom(double[][] arr1,int dim,int index){

                double[][] LPF = new double[dim][dim];

                                double[][] HPF = new double[dim][dim];

                                double[][] d_hpf_col = new double[dim][dim/2];

                                double[][] d_lpf_col = new double[dim][dim/2];

                                double[][] LPF_0 = new double[dim][dim/2];

                                double[][] LPF_1 = new double[dim][dim/2];

                                double[][] HPF_0 = new double[dim][dim/2];

                                double[][] HPF_1 = new double[dim][dim/2];

                                double[][] d_lhpf_row = new double[dim/2][dim/2];

                                double[][] d_llpf_row = new double[dim/2][dim/2];

                                double[][] d_hhpf_row = new double[dim/2][dim/2];

                                double[][] d_hlpf_row = new double[dim/2][dim/2];

                                double[][] u_llpf_row = new double[dim][dim/2];

                                double[][] u_lhpf_row = new double[dim][dim/2];

                                double[][] u_hlpf_row = new double[dim][dim/2];

                                double[][] u_hhpf_row = new double[dim][dim/2];

                                double[][] LPR_0 = new double[dim][dim/2];

                                double[][] HPR_0 = new double[dim][dim/2];
```



```
        double[][] LPR_1 = new double[dim][dim/2];
        double[][] HPR_1 = new double[dim][dim/2];
        double[][] addcon_0 = new double[dim][dim/2];
        double[][] addcon_1 = new double[dim][dim/2];
        double[][] u_lpr_row = new double[dim][dim];
        double[][] u_hpr_row = new double[dim][dim];
int dim1 = 32;


int[][] arrOri;
arrOri = new int[arr1.length][arr1[0].length];
System.out.println("Approximation coefficients are :");
for(int i=0;i<dim;i++){
        for(int j=0;j<dim;j++){
                arrOri[i][j] = (int)arr1[j][i];
                System.out.print(""+arrOri[i][j]);
        }
        System.out.println();
}


LPF = gc.ConLPF_Row(arr1,dim);
HPF = gc.ConHPF_Row(arr1,dim);
d_lpf_col = gc.down2D_Col(LPF,dim);
d_hpf_col = gc.down2D_Col(HPF,dim);
LPF_0 = gc.ConLPF_Col(d_lpf_col,dim,dim/2);  //SIZE(LPF_0)=32*16
```

```
                HPF_0 = gc.ConHPF_Col(d_lpf_col,dim,dim/2); //SIZE(HPF_0)=32*16

                LPF_1 = gc.ConLPF_Col(d_hpf_col,dim,dim/2); //SIZE(LPF_1)=32*16

                HPF_1 = gc.ConHPF_Col(d_hpf_col,dim,dim/2); //SIZE(HPF_1)=32*16

                d_llpf_row = gc.down2D_Row(LPF_0,dim,dim/2);
                //SIZE(D_LLPF_ROW)=16*16

                d_lhpf_row = gc.down2D_Row(HPF_0,dim,dim/2); //SIZE(D_LHPF_ROW)=16*16

                d_hlpf_row = gc.down2D_Row(LPF_1,dim,dim/2); //SIZE(D_HLPF_ROW)=16*16

                d_hhpf_row = gc.down2D_Row(HPF_1,dim,dim/2);
                //SIZE(D_HHPF_ROW)=16*16


                if(index == 1)

                        return d_hlpf_row;

                else

                return d_llpf_row;

}

// Reconstruction Phase

public void ImRecon(double[][] d_llpf_row,int dim){


                System.out.println("dim="+dim);

                double[][] LPF = new double[dim][dim];

                double[][] HPF = new double[dim][dim];

                double[][] d_hpf_col = new double[dim][dim/2];

                double[][] d_lpf_col = new double[dim][dim/2];

                double[][] LPF_0 = new double[dim][dim/2];

                double[][] LPF_1 = new double[dim][dim/2];

                double[][] HPF_0 = new double[dim][dim/2];
```

```
double[][] HPF_1 = new double[dim][dim/2];
double[][] d_lhpf_row = new double[dim/2][dim/2];
double[][] d_hhpf_row = new double[dim/2][dim/2];
double[][] d_hlpf_row = new double[dim/2][dim/2];
double[][] u_llpf_row = new double[dim][dim/2];
double[][] u_lhpf_row = new double[dim][dim/2];
double[][] u_hlpf_row = new double[dim][dim/2];
double[][] u_hhpf_row = new double[dim][dim/2];
double[][] LPR_0 = new double[dim][dim/2];
double[][] HPR_0 = new double[dim][dim/2];
double[][] LPR_1 = new double[dim][dim/2];
double[][] HPR_1 = new double[dim][dim/2];
double[][] addcon_0 = new double[dim][dim/2];
double[][] addcon_1 = new double[dim][dim/2];
double[][] u_lpr_row = new double[dim][dim];
double[][] u_hpr_row = new double[dim][dim];


double[][] t = new double[dim][dim/2];
for(int i=0;i<dim;i++){
        for(int j=0;j<dim/2;j++)
                t[i][j]=0;
}

u_llpf_row = gc.up2D_row(d_llpf_row,dim/2,dim/2);
```

```
u_lhpf_row = t;
u_hlpf_row = t;
u_hhpf_row = t;
LPR_0 = gc.ConLPR_Col(u_llpf_row,dim,dim/2);
LPR_1 = gc.ConLPR_Col(u_hlpf_row,dim,dim/2);
HPR_0 = gc.ConHPR_Col(u_lhpf_row,dim,dim/2);
HPR_1 = gc.ConHPR_Col(u_hhpf_row,dim,dim/2);
for(int i=0;i<dim;i++){
        for(int j=0;j<dim/2;j++){
                addcon_0[i][j] = LPR_0[i][j] + HPR_0[i][j];
                addcon_1[i][j] = LPR_1[i][j] + HPR_1[i][j];
        }
}
u_lpr_row = gc.up2D_col(addcon_0,dim,dim/2);
u_hpr_row = gc.up2D_col(addcon_1,dim,dim/2);
u_lpr_row = gc.ConLPR_row(u_lpr_row,dim,dim);
u_hpr_row = gc.ConHPR_row(u_hpr_row,dim,dim);
for(int i=0;i<dim/2;i++){
        for(int j=0;j<dim/2;j++){
                if(d_llpf_row[i][j] >0.5)
                        d_llpf_row[i][j] = 1.0;
                else
                        d_llpf_row[i][j] = 0.0;
        }
}
```

```
                System.out.println("Reconstructed Approximation Coeff:"+dim);
                for(int i=0;i<dim/2;i++){
                        for(int j=0;j<dim/2;j++){
                                System.out.print(""+(int)d_llpf_row[i][j]);
                        }
                        System.out.println();
                }

        }

}
```

```
// IMPLIMENTATION OF MULTILAYER PERCEPTRON CLASSIFIER OF ARTIFICIAL NEURAL
   NETWORKS
import java.io.*;
import javax.swing.*;

public class MLP {
      protected int inNeurons;
      protected int hidNeurons;
      protected int outNeurons;
      protected InputLayer inLayer;
      protected HiddenLayer hidLayer;
      protected OutputLayer outLayer;
      private int patCount;
      private int validationCount;
      private double[][] validationPatterns;

      private double[][] patterns;
      private double[][] inputPatterns;
      private double[][] outputPatterns;
      private double learningRate;

      public MLP(){}
      public MLP(int inputs, int hneus, int oneus, int pts, int valids){
            inNeurons = inputs;
            hidNeurons = hneus;
            outNeurons = oneus;
            patCount = pts;
            validationCount = valids;
            hidLayer = new HiddenLayer(inputs, hneus, patCount, oneus);
            inLayer = new InputLayer(inNeurons,hidNeurons,patCount);
            outLayer = new OutputLayer(outNeurons,hidNeurons,patCount);
            hidLayer.setSources(inLayer);
            hidLayer.setTargets(outLayer);
            validationPatterns = new double[validationCount][inputs + oneus];
            patterns = new double[patCount][inputs + oneus];
            inputPatterns = new double[patCount][inputs] ;
            outputPatterns = new double[patCount][oneus] ;
      }
      public void createNet(int inputs, int hneus, int oneus, int pts){
            inNeurons = inputs;
            hidNeurons = hneus;
            outNeurons = oneus;
            patCount = pts;
            hidLayer = new HiddenLayer(inputs, hneus, patCount, oneus);
            inLayer = new InputLayer(inNeurons, hidNeurons,patCount);
            outLayer = new OutputLayer(outNeurons,hidNeurons,patCount);
            hidLayer.setSources(inLayer);
            hidLayer.setTargets(outLayer);
            patterns = new double[patCount][inputs + oneus];
            inputPatterns = new double[patCount][inputs] ;
            outputPatterns = new double[patCount][oneus] ;
            learningRate=0.2;
```

```
	}
	public void assignHiddenWeights(double[][] wa){
		hidLayer.assignInWeights(wa);
	}

	public void assignOutputWeights(double[][] wa){
		hidLayer.assignOutWeights(wa);
	}
	public void randomizeWeights(){
		hidLayer.randomizeWeights();
	}

	public void printSynapseInfo(int patn) {
		System.out.println("***** Synapses info with pattern number =
"+patn+" pattern count = "+patCount);
		System.out.println("***** Input Layer Synapses
info***************");
		inLayer.printSynapseInfo(patn);
		System.out.println("***** Hidden Layer Synapses
info***************");
		hidLayer.printSynapseInfo(patn);
		System.out.println("***** Output Layer Synapses
info***************");
		outLayer.printSynapseInfo(patn);
	}

	public void printInputs() {
		for ( int i =0; i < patCount; i++ ) {
			String patString = "Inputs of pattern "+(i+1)+"  ";
			for (int j=0; j<inNeurons; j++) {
				patString += inputPatterns[i][j]+" : ";
			}
			System.out.println(patString);
		}
	}

	public void setInputPatterns() {
		inLayer.setInputPatterns(inputPatterns);
	}

	public void setOutputPatterns() {
		outLayer.setOutputPatterns(outputPatterns);
	}


	public void setTestPatterns(int vpc, double[][] vptns) {
		int outNeurons = 0;
		validationCount = vpc;
		validationPatterns = new double[validationCount][inNeurons +
outNeurons];
		validationPatterns = vptns;
		outLayer.setValidationPatterns(vpc,validationPatterns,inNeurons);
		inLayer.setValidationPatterns(vpc,validationPatterns);
		hidLayer.setValids(vpc);
	}

	public void printDesiredOutputs() {
```

```
        for ( int i =0; i < patCount; i++ ) {
                String patString = "Outputs of pattern "+(i+1)+"  ";
                for (int j=inNeurons; j<inNeurons+outNeurons; j++) {
                        patString += outputPatterns[i][j-inNeurons]+" : ";
                }
                System.out.println(patString);
        }
}

public void forwardPass(int npat) {
        hidLayer.forwardPass(npat);
        outLayer.forwardPass(npat);
}

public void forwardPass() {
        hidLayer.forwardPass();
        outLayer.forwardPass();
}

public void forwardTestPass() {
        hidLayer.forwardTestPass();
        outLayer.forwardTestPass();
}


public void backwardPass() {
        outLayer.backwardPass();
        hidLayer.backwardPass();
}

public void setGradients() {
        outLayer.setGradients();
        hidLayer.setGradients();
}

public void train(int npat) {
        outLayer.train(npat);
        hidLayer.train(npat);
}

public void train() {
        outLayer.backwardPass();
        hidLayer.backwardPass();
}

public void printPatterns() {
        System.out.println(" Patterns for training of network:  ");
        for ( int p = 0; p<patCount; p++) {
                String s = "Pattrn "+(p+1)+" :: inputs ";
                for ( int n=0; n<inNeurons; n++)
                        s += patterns[p][n]+" ; ";
                s += " : outputs : ";
                for ( int n=inNeurons; n<inNeurons+outNeurons; n++)
                        s += patterns[p][n]+" ; ";
        }
        System.out.println(" Patterns of input layer:  ");
        inLayer.printPatterns();
```

```
        System.out.println(" Patterns of output layer:  ");
        outLayer.printPatterns();
    }

    public void printOutputs() {
        System.out.println(" Computed Outputs of output layer ");
        outLayer.printOutputs();
    }

    public void printNetwork(PrintWriter pw) {

        pw.println(inNeurons);
        pw.println(hidNeurons);
        pw.println(outNeurons);
        hidLayer.printWeights(pw);
        outLayer.printWeights(pw);
    }


    public double getError() {
        return outLayer.getError();
    }

    public void setLearningRate(){
        learningRate = learningRate/10;
    }

    public double getTestError() {
            return outLayer.getTestError();
    }

    public void readNet(String tdf ) throws IOException{
        /*String s1 = "0";
        s1 = JOptionPane.showInputDialog(" Please Enter the name of
training data file");
        */
        String dataFile = tdf;
        int inputs, hiddens, outs,patns;
        FileReader fr = new FileReader(dataFile);
        BufferedReader br = new BufferedReader(fr);
        String textLine = br.readLine();
        inputs = Integer.parseInt(textLine);
        textLine = br.readLine();
        hiddens = Integer.parseInt(textLine);
        textLine = br.readLine();
        outs = Integer.parseInt(textLine);
        textLine = br.readLine();
        patns = Integer.parseInt(textLine);

        hidNeurons = hiddens;
        outNeurons = outs;
        inNeurons = inputs;
        patCount = patns;

        createNet(inputs, hiddens, outs, patns);
```

```
            System.out.println((inNeurons+outNeurons)+" inputs =
"+inNeurons+" hidden neurons = "+hidNeurons+" outputs = "+outNeurons+"
patterns "+patCount);
            double [][] pa = new double[patCount][inNeurons+outNeurons];
            double [][] inpats = new double[patCount][inputs];
            double [][] outpats = new double[patCount][outNeurons];
            for ( int i =0; i < patCount; i++ ) {
                  textLine = br.readLine();
                  String[] tokens = textLine.split("\\s|,");
                  for (int j=0; j<(inputs+outNeurons); j++) {
                        double patij = Double.parseDouble(tokens[j]);
                        pa[i][j] = patij;
                        if( j < inputs)
                              inpats[i][j] = patij;
                        else
                              outpats[i][j-inputs] = patij;
                  }
            }

            patterns = pa;
            inputPatterns = inpats;
            outputPatterns = outpats;
            setInputPatterns();
            setOutputPatterns();
            br.close();
            fr.close();

            double [][] valPats;
            String valFile = vdf;
            fr = new FileReader(valFile);
            br = new BufferedReader(fr);
            textLine = br.readLine();
            valCount = Integer.parseInt(textLine);

            valPats = new double[valCount][inNeurons + outNeurons];
            for ( int i =0; i < valCount; i++ ) {
                  textLine = br.readLine();
                  String[] tokens = textLine.split("\\s|,");
                  for (int j=0; j<(inNeurons+outNeurons); j++) {
                        double patij = Double.parseDouble(tokens[j]);
                        valPats[i][j] = patij;
                  }
            }
            setValidationPatterns(valCount,valPats);
            br.close();
            fr.close();

      }

      public void readWeightedNet(String tdf/*, String vdf*/) throws
IOException{
            /*String s1 = "0";
            s1 = JOptionPane.showInputDialog(" Please Enter the name of
training data file");
            */
            String dataFile = tdf;
            int inputs, hiddens, outs,patns;
```

```
            FileReader fr = new FileReader(dataFile);
            BufferedReader br = new BufferedReader(fr);
            String textLine = br.readLine();
            inputs = Integer.parseInt(textLine);
            textLine = br.readLine();
            hiddens = Integer.parseInt(textLine);
            textLine = br.readLine();
            outs = Integer.parseInt(textLine);
            textLine = br.readLine();
            patns = Integer.parseInt(textLine);

            hidNeurons = hiddens;
            outNeurons = outs;
            inNeurons = inputs;
            patCount = patns;

            createNet(inputs, hiddens, outs, patns);
            System.out.println((inNeurons+outNeurons)+" inputs =
"+inNeurons+" hidden neurons = "+hidNeurons+" outputs = "+outNeurons+"
patterns "+patCount);
            double [][] pa = new double[patCount][inNeurons+outNeurons];
            double [][] inpats = new double[patCount][inputs];
            double [][] outpats = new double[patCount][outNeurons];
            for ( int i =0; i < patCount; i++ ) {
                  textLine = br.readLine();
                  String[] tokens = textLine.split("\\s|,");
                  for (int j=0; j<(inputs+outNeurons); j++) {
                        double patij = Double.parseDouble(tokens[j]);
                        pa[i][j] = patij;
                        if( j < inputs)
                              inpats[i][j] = patij;
                        else
                              outpats[i][j-inputs] = patij;
                  }
            }

            patterns = pa;
            inputPatterns = inpats;
            outputPatterns = outpats;
            setInputPatterns();
            setOutputPatterns();
            int lnks = (inputs+1);
            double[][] inWeights = new double[hidNeurons][lnks];
            for ( int n =0; n < hidNeurons; n++ ) {
                  textLine = br.readLine();
                  String[] tokens = textLine.split("\\s|,");
                 for (int j=0; j<=inNeurons; j++) {
                        inWeights[n][j] = Double.parseDouble(tokens[j]);
                        System.out.print("
inWeights["+n+"]["+j+"]="+inWeights[n][j]);
                  }
            }
            hidLayer.assignInWeights(inWeights);
            double[][] outLinkWeights = new double[outNeurons][hidNeurons+1];
            String ol=" ";
            for ( int n =0; n < outNeurons; n++ ) {
                  textLine = br.readLine();
```

```
            String[] tokens = textLine.split("\\s|,");
            for (int j=0; j<= hidNeurons; j++) {
                  outLinkWeights[n][j] = Double.parseDouble(tokens[j]);
                  ol += " : "+j+" , "+n+" = "+outLinkWeights[n][j];
            }
      }

      outLayer.assignInWeights(outLinkWeights);
      br.close();
      fr.close();
}


public void readTestNet(String tdf) throws IOException{
      String dataFile = tdf;
      int inputs, hiddens, outs,testpats,testCount;
      FileReader fr = new FileReader(dataFile);
      BufferedReader br = new BufferedReader(fr);
      String textLine = br.readLine();
      inputs = Integer.parseInt(textLine);
      textLine = br.readLine();
      hiddens = Integer.parseInt(textLine);
      textLine = br.readLine();
      outs = Integer.parseInt(textLine);
      textLine = br.readLine();
      testpats = Integer.parseInt(textLine);

      hidNeurons = hiddens;
      outNeurons = outs;
      inNeurons = inputs;
      testCount = testpats;

      createNet(inputs, hiddens, outs, testCount);
      double [][] pa = new double[testCount][inNeurons+outNeurons];
      double [][] inpats = new double[testCount][inputs];
      double [][] outpats = new double[testCount][outNeurons];
      for ( int i =0; i < testCount; i++ ) {
            textLine = br.readLine();
            String[] tokens = textLine.split("\\s|,");
            for (int j=0; j< inputs; j++) {

                  double patij = Double.parseDouble(tokens[j]);
                  pa[i][j] = patij;
            }
      }
      setTestPatterns(testCount,pa);
      int lnks = (inputs+1);
      double[][] inWeights = new double[hidNeurons][lnks];
      for ( int n =0; n < hidNeurons; n++ ) {
            textLine = br.readLine();
            String[] tokens = textLine.split("\\s|,");
            for (int j=0; j<=inNeurons; j++) {
                  inWeights[n][j] = Double.parseDouble(tokens[j]);
            }
            System.out.println();
      }
      hidLayer.assignInWeights(inWeights);
```

```
            double[][] outLinkWeights = new double[outNeurons][hidNeurons+1];
            String ol=" ";
            for ( int n =0; n < outNeurons; n++ ) {
                  textLine = br.readLine();
                  String[] tokens = textLine.split("\\s|,");
                 for (int j=0; j<= hidNeurons; j++) {
                        outLinkWeights[n][j] = Double.parseDouble(tokens[j]);
                        ol += " : "+j+" , "+n+" = "+outLinkWeights[n][j];
                  }
            }
            outLayer.assignInWeights(outLinkWeights);
            br.close();
            fr.close();
      }


      public static void main( String[] args) throws IOException{
            MLP xn = new MLP();
            String s = JOptionPane.showInputDialog(" Type '1' for training
and '2' for testing and '3' for further training");
            if ((s.trim().length() == 0 )) return;
                  int runType = Integer.parseInt(s);
            if(runType < 1 || runType > 3){
                  System.out.println("press 1 , 2 or 3 only");
                  return;
            }
            String df = "0";
            df = JOptionPane.showInputDialog(" Please Enter the name of data
file");
            String dataFileName = df;
            if(runType == 1 || runType == 3){

                  xn.readNet(dataFileName);
                  System.out.println(" network data read ");
                  FileWriter fw = new FileWriter("network.out");
                  PrintWriter networkStream = new PrintWriter(fw);

                  if(runType == 3)
                        xn.readWeightedNet(dataFileName);
                  else
                        xn.randomizeWeights();

                  int i = 0;
                  int totalEpochs = 0;

                  while ( true ) {
                        String s1 = JOptionPane.showInputDialog(" Type number
of training epochs to carry out or 0 to exit");
                        if ( (s1.trim().length() == 0 ) ) break;
                              int turns = Integer.parseInt(s1);
                        if ( (turns  <= 0) ) break;
                              totalEpochs += turns;
                        for ( ; i < totalEpochs; i ++) {
                              xn.forwardPass();
                              xn.setGradients();
                              xn.train();
```

```
                              System.out.println("After Epoch "+(i+1)+"
average error on training & test sets = "+dround(xn.getError(),4)+" ::
"+dround(xn.getTestError(),4));
                        }
                        networkStream.println();
                        networkStream.println("*** After "+(i+1)+" Epochs,
the updated weights are :");
                        xn.printNetwork(networkStream);
                  }
                  System.out.println("^^^^^^^^^^^^^^^ After "+i+"  epochs
^^^^^^^^^^^^^^^^^^");
                  networkStream.println();
                  xn.printNetwork(networkStream);

                  xn.printOutputs();
                  fw.close();
            }
            else{
                  String tf = "0";
                  tf = JOptionPane.showInputDialog(" Please Enter the name of
test file");
                  String testFileName = tf;
                  xn.readTestNet(testFileName);
                  xn.forwardTestPass();
                  xn.printOutputs();
            }
      }

      public static double dround ( double x, int n ) {
            long copy = 01;
            if ( n <= 0 ) {
                  copy = Math.round(x);
                  return (double)copy;
            }
            long multiplier = 1;
            for ( int i=0; i<n; i++)
                  multiplier *= 10;
            double y = x * multiplier ;
            copy = Math.round(y);
            return (double)copy/multiplier;
      }

}
```