HUMAN COMPUTER INTERACTION USING NEURAL NETWORKS

Project Report submitted in partial fulfillment of the requirement for the degree of

Bachelor of Technology.

in

Electronics and Communication Engineering

under the Supervision of

Ms. Pragya Gupta

By

Group **MPG-2**

Urmanpreet Singh (091012)

SahilSoni (091075)

AmanAgarwal (091081)

To

Jaypee University of Information and Technology

Waknaghat, Solan – 173234, Himachal Pradesh

Certificate

This is to certify that project report entitled "**Human Computer Interaction using Neural Networks**", submitted by **Urmanpreet Singh**, **Sahil Soni** and **Aman Agarwal** in partial fulfillment for the award of degree of Bachelor of Technology in Electronics and Communication Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

P Gupta 29/05/2013

Date: 29-05-2013

Ms. Pragya Gupta

(Sr. Lecturer)

Acknowledgement

It has been a wonderful and intellectually stimulating experience working on the **Human Computer Interaction using Neural Network** which is useful in the areas of Artificial Intelligence.

We gratefully acknowledge the Management and Administration of Jaypee University of Information Technology for providing us the opportunity and hence the environment to initiate and complete our project.

We are greatly thankful to **Ms. Pragya Gupta**, the supervisor of the project for giving us freedom to implement our creativity and at the same time guiding us at important steps of our work with attention and care. Her methodology of making the system strong from inside has taught us that output is not the end of project.

Date: 29-05-2013

Urmanpreet Singh
Sahil Soni
Aman Agarwal

Table of Contents

List of Figures

List of Tables

Abstract

Since the introduction of the most common input computer devices not a lot have changed. This is probably because the existing devices are adequate. It is also now that computers have been so tightly integrated with everyday life, that new applications and hardware are constantly introduced. The means of communicating with computers at the moment are limited to keyboards, mice, light pen, trackball, keypads etc.

These devices have grown to be familiar but inherently limit the speed and naturalness with which we interact with the computer. As the computer industry follows Moore's Law since middle 1960s, powerful machines are built equipped with more peripherals. Vision based interfaces are feasible and at the present moment the computer is able to "see". Hence users are allowed for richer and user-friendlier man-machine interaction. This can lead to new interfaces that will allow the deployment of new commands that are not possible with the current input devices. Plenty of time will be saved as well.

Recently, there has been a surge in interest in recognizing human hand gestures. Hand-gesture recognition has various applications like computer games, machinery control and thorough mouse replacement. One of the most structured sets of gestures belongs to sign language. In sign language, each gesture has an assigned meaning (or meanings).
We have used developed a neural network based human computer interaction system incorporating some open ended set of features including classification tasks such as handwritten digit recognition and gesture recognition. We have used the gestures to interface with the machine with an example application to control the basic operations of a media player. The Neural network under problem specific optimization is also capable of handling several other applications such as voice recognition, facial recognition and stock market forecast provided the features are extracted properly.

CHAPTER 1: INTRODUCTION

1.1 What is Human Computer Interaction:

It involves the study, planning, and design of the interaction between people (users) and computers. It is often regarded as the intersection of computer science, behavioral sciences, design and several other fields of study. The term was popularized by Card, Moran, and Newell in their seminal 1983 book, "The Psychology of Human-Computer Interaction", although the authors first used the term in 1980, and the first known use was in 1975. The term connotes that, unlike other tools with only limited uses (such as a hammer, useful for driving nails, but not much else), a computer has many affordances for use and this takes place in an open-ended dialog between the user and the computer.

Because human–computer interaction studies a human and a machine in conjunction, it draws from supporting knowledge on both the machine and the human side. On the machine side, techniques in computer graphics, operating systems, programming languages, and development environments are relevant. On the human side, communication theory, graphic and industrial design disciplines, linguistics, social sciences, cognitive psychology, and human factors such as computer user satisfaction are relevant. Engineering and design methods are also relevant. Due to the multidisciplinary nature of HCI, people with different backgrounds contribute to its success. HCI is also sometimes referred to as **man–machine interaction (MMI)** or **computer–human interaction (CHI)**.

1.2 Goals of Human Computer Interaction:

A basic goal of HCI is to improve the interactions between users and computers by making computers more usable and receptive to the user's needs. Specifically, HCI is concerned with:

- Methodologies and processes for **designing interfaces** (i.e., given a task and a class of users, design the best possible interface within given constraints, optimizing for a desired property such as learnability or efficiency of use).
- Methods for **implementing interfaces** (e.g. software toolkits and libraries; efficient algorithms.
- Techniques for **evaluating** and **comparing interfaces.**
- Developing **new interfaces** and interaction techniques.
- Developing descriptive and **predictive models** and theories of interaction.

A long term goal of HCI is to design systems that minimize the barrier between the human's cognitive model of what they want to accomplish and the computer's understanding of the user's task.

1.3 Applications of Human Computer Interaction:

Excellence in HCI is important for several reasons:

Quality of life: Important applications of computers in medicine are possible only if they are both useful and easy to use by doctors, nurses, and aides; similarly, use of computers in education requires that they be both useful and easy to use by students and teachers. Computers can assist disabled individuals; at the same time, special techniques are needed to allow computers to be used by some who are disabled.

National competitiveness: Information technology is one of the drivers for increased productivity. As more and more workers use computers in their jobs, training time and ease-of-use issues become economically more and more important.

Growth of the computer and communications industries: Powerful, interesting, and usable applications are the fuel for continuing growth of these industries. The current growth cycle is the direct consequence of the graphical user interface developed by Xerox and commercialized by Apple and Microsoft, and of the lower computer costs

made possible by the microprocessor. The resulting mass market supports commodity pricing for both hardware and software. Future growth cycles will in part be driven by current HCI research, which will lead to new applications that are increasingly easy to use.

National security: Computer-based command, control, communications, and intelligence systems are at the heart of our military infrastructure. Interfaces between operators and computers are found in cockpits, on the bridge, and in the field. To be effective, these systems must have high-quality human-computer interfaces.

1.4 Challenges Faced by HCI Applications:

Interactive applications pose particular challenges. The **response time** should be very fast. The user should sense no appreciable delay between when he or she makes a gesture or motion and when the computer responds. The computer vision algorithms should be **reliable** and work for different people. There are also **economic constraints**: the vision-based interfaces will be replacing existing ones, which are often very low cost.

Attention to human-machine interaction is also important because poorly designed human-machine interfaces can lead to many **unexpected problems**. A classic example of this is the **Three Mile Island accident**, a nuclear meltdown accident, where investigations concluded that the design of the human–machine interface was at least partially responsible for the disaster. Similarly, accidents in aviation have resulted from manufacturers' decisions to use non-standard flight instrument: even though the new designs were proposed to be superior in regards to basic human–machine interaction, pilots had already ingrained the "standard" layout and thus the conceptually good idea actually had undesirable results.

CHAPTER 2: ARTIFICIAL NEURAL NETWORK

2.1 What is Neural Network:

In machine learning and computational neuroscience, an **artificial neural network**, often just named a **neural network**, is a mathematical model inspired by biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation. In most cases a neural network is an **adaptive system** changing its structure during a **learning phase**. Neural networks are used for modeling complex relationships between inputs and outputs or to find patterns in data.

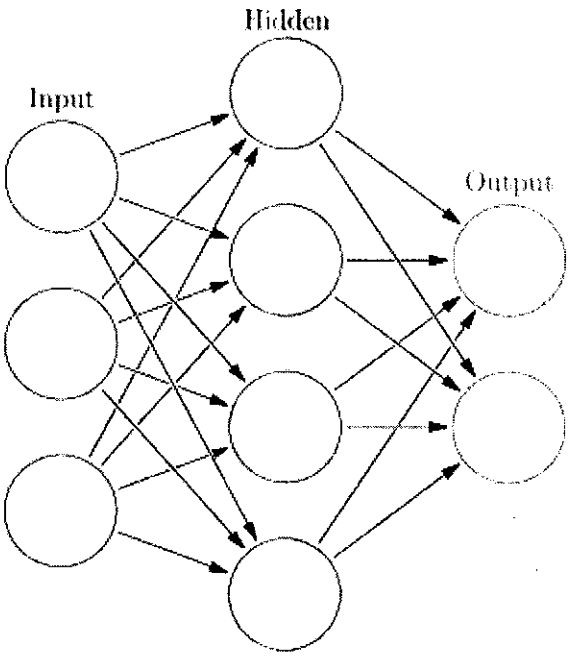


Fig 1: Representation of a simple Neural Network

2.2 Background:

The inspiration for neural networks came from examination of central nervous systems. In an artificial neural network, simple artificial nodes, called "**neurons**", "neurodes", "processing elements" or "units", are connected together to form a network which mimics a biological neural network. There is no single formal definition of what an artificial neural network is. Generally, it involves a network of simple processing elements exhibiting complex global behavior determined by the connections between the processing elements and element parameters. Artificial neural networks are used with algorithms designed to alter the strength of the connections in the network to produce a desired signal flow.

Neural networks are also similar to biological neural networks in performing functions collectively and in parallel by the units, rather than there being a clear delineation of subtasks to which various units are assigned. The term "neural network" usually refers to models employed in statistics, cognitive psychologyand artificial intelligence. Neural network models which emulate the central nervous system are part of theoretical neuroscience and computational neuroscience.

In modern software implementations of artificial neural networks, the approach inspired by biology has been largely abandoned for a more practical approach based on statistics and signal processing. In some of these systems, neural networks or parts of neural networks (like artificial neurons) form components in larger systems that combine both adaptive and non-adaptive elements. While the more general approach of such **adaptive systems** is more suitable for real-world problem solving, it has far less to do with the traditional artificial intelligence connectionist models. What they do have in common, however, is the principle of non-linear, distributed, parallel and local processing and adaptation. Historically, the use of neural networks models marked a paradigm shift in the late eighties from high-level (symbolic) artificial intelligence, characterized by expert systems with knowledge embodied in if-then rules, to low-level (sub-symbolic) machine learning, characterized by knowledge embodied in the parameters of a **dynamical system**.

2.3 Architecture of Neural Network:

The word **network** in the term 'artificial neural network' refers to the inter–connections between the neurons in the different layers of each system. An example system has three layers. The first layer has input neurons, which send data via synapses to the second layer of neurons, and then via more synapses to the third layer of output neurons. More complex systems will have more layers of neurons with some having increased layers of input neurons and output neurons. The synapses store parameters called "**weights**" that manipulate the data in the calculations.

An ANN is typically defined by three types of parameters:

1. The **interconnection pattern** between different layers of neurons
2. The **learning process** for updating the weights of the interconnections
3. The **activation function** that converts a neuron's weighted input to its output activation.

Commonly neural networks are adjusted, or trained, so that a particular input leads to a specific target output. Such a situation is shown in fig(3). There, the network is adjusted, based on a comparison of the output and the target, until the network output matches the target. Typically many such input/target pairs are used, in this ***supervised learning*** (training method studied in more detail on following chapter), to train a network.

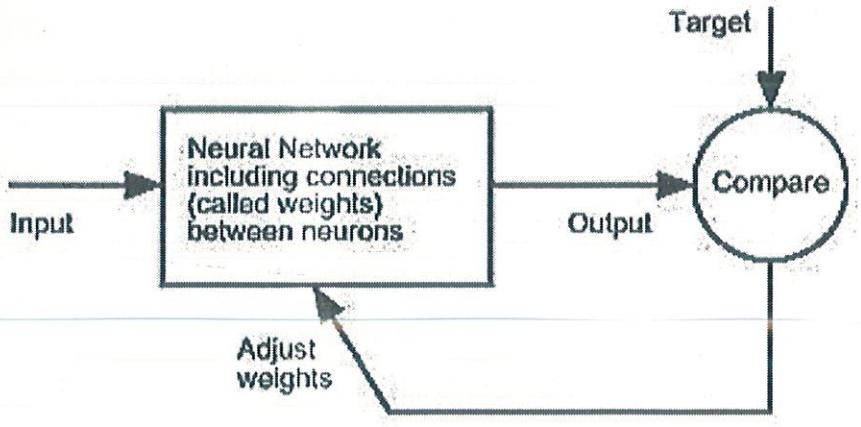


Fig 2: Neural Net block diagram

2.4 Neuron Model:

A neuron with a single scalar input and no bias is shown on the left below.

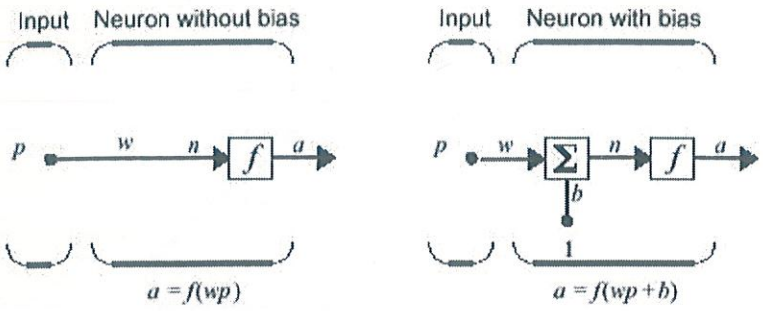


Fig 3: Neuron

The scalar **input *p*** is transmitted through a connection that multiplies its strength by the scalar **weight w**, to form the product *w*p, again a scalar. Here the weighted input *wp* is the only argument of the **transfer function f**, which produces the scalar output *a*. The neuron on the right has a scalar **bias, b**. You may view the bias as simply being added to the product *wp* as shown by the summing junction or as shifting the function *f* to the left by an amount b. The bias is much like a weight, except that it has a constant input of 1. The transfer function net input n, again a scalar, is the sum of the weighted input *wp* and the bias b. This sum is the argument of the transfer function *f*. Here *f* is a transfer function typically a step function or a **sigmoid function**, that takes the argument *n* and produces the output a. Examples of various transfer functions are given in the next section. Note that *w* and *b* are both adjustable scalar parameters of the neuron. The central idea of neural networks is that such parameters can be adjusted so that the network exhibits some desired or interesting behavior.

Thus, we can train the network to do a particular job by adjusting the weight or bias parameters, or perhaps the network itself will adjust these parameters to achieve some desired end. All of the neurons in the program written in MATLAB have a bias. However, you may omit a bias in a neuron if you wish.

2.5 Transfer Functions:

Three of the most commonly used transfer functions are shown in fig(5).

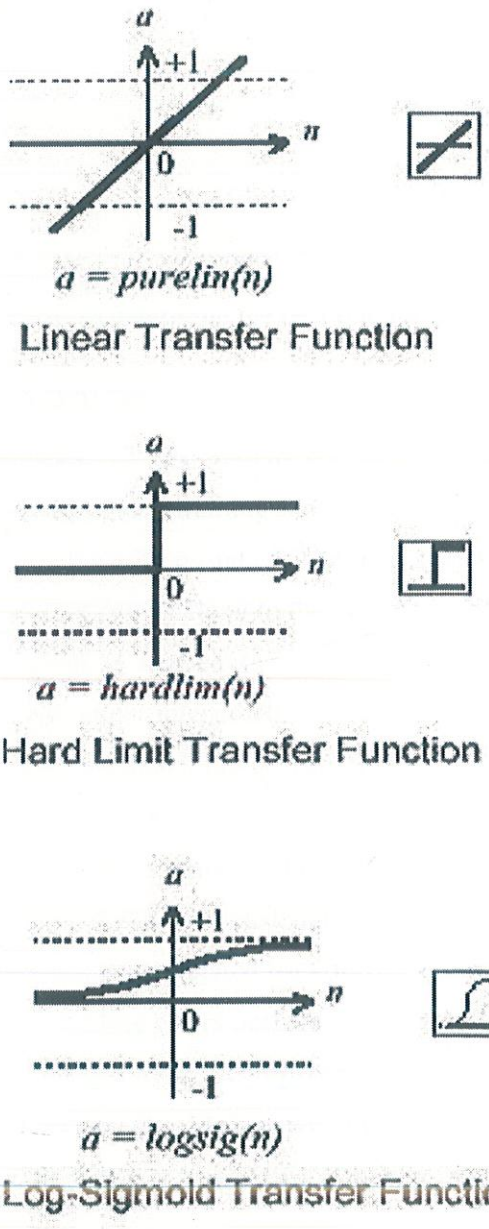


Fig 4: Transfer Functions

The hard limit transfer function shown above limits the output of the neuron to either 0, if the net input argument n is less than 0, or 1, if n is greater than or equal to 0. This is the function used for the Perceptron algorithm written in MATLAB to create neurons that make a classification decision.

2.6 Learning:

Learning is the process by which the free parameters of a neural network are adapted through a process of stimulation by the environment in which the network is embedded.

There are two modes of learning: **Supervised** and **unsupervised**. Below there is a brief description of each one to determine the best one for our problem.

2.6.1 Supervised Learning:

Supervised learning is based on the system trying to **predict outcomes** for known examples and is a commonly used training method. It compares its predictions to the target answer and "learns" from its mistakes. The data start as inputs to the input layer neurons. The neurons pass the inputs along to the next nodes. As inputs are passed along, the weighting, or connection, is applied and when the inputs reach the next node, the weightings are summed and either intensified or weakened. This continues until the data reach the output layer where the model predicts an outcome.

In a supervised learning system, the predicted output is compared to the actual output for that case. If the predicted output is equal to the actual output, no change is made to the weights in the system. But, if the predicted output is higher or lower than the actual outcome in the data, the error is propagated back through the system and the weights are adjusted accordingly. This feeding errors backwards through the network is called "back-propagation." Both the **Multi-Layer Perceptron** and the Radial Basis Function are supervised learning techniques. The Multi-Layer Perceptron uses the back-propagation while the Radial Basis Function is a feed-forward approach which trains on a single pass of the data.

2.6.2 Unsupervised Learning:

Neural networks which use unsupervised learning are most effective for **describing data** rather than predicting it. The neural network is not shown any outputs or answers as part of the training process—in fact, there is no concept of output fields in this type of system. The primary unsupervised technique is the Kohonen network. The main uses of Kohonen and other unsupervised neural systems are in cluster analysis where the goal is to group "like" cases together.

The advantage of the neural network for this type of analysis is that it requires no initial assumptions about what constitutes a group or how many groups there are. The system starts with a clean slate and is not biased about which factors should be most important.

2.7 Multilayer Perceptron:

A multilayer perceptron is a **feed forward neural network** with one or more hidden layers. The network consists of an **input layer** of source neurons, at least one middle or **hidden layer** of computational neurons, and an **output layer** of computational neurons. The input signals are propagated in a forward direction on a layer-by-layer basis.

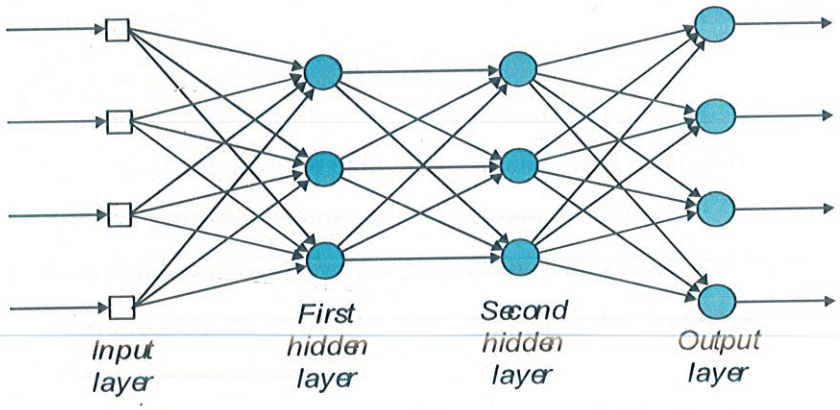


Fig 5: A Multi-layered Perceptron with 2 hidden layers

It works by using the following algorithms:

1. **Feed forward neural network algorithm**
2. **Back Propagation algorithm**

The general steps in **Perceptron training Algorithm** are:

1. Initialization of weights.
2. Activation of the perceptron by applying inputs and desired outputs.
3. Update the weights after the learning phase.
4. Increase iteration by 1 and repeat the process till minimization of cost function.

2.8 Back propagation learning Algorithm:

Back propagation, an abbreviation for "backward propagation of errors", is a common method of training artificial neural networks. From a desired output, the network learns from many inputs, similar to the way a child learns to identify a dog from examples of dogs.

For better understanding, the back propagation learning algorithm can be divided into two phases: propagation and weight update.

Phase 1: Propagation

Each propagation involves the following steps:

1. Forward propagation of a training pattern's input through the neural network in order to generate the propagation's output activations.
2. Backward propagation of the propagation's output activations through the neural network using the training pattern target in order to generate the deltas of all output and hidden neurons.

Phase 2: Weight update

For each weight-synapse follow the following steps:

1. Multiply its output delta and input activation to get the gradient of the weight.
2. Bring the weight in the opposite direction of the gradient by subtracting a ratio of it from the weight.

This ratio influences the speed and quality of learning; it is called the *learning rate*. The sign of the gradient of a weight indicates where the error is increasing, this is why the weight must be updated in the opposite direction.

Repeat phase 1 and 2 until the performance of the network is satisfactory.

As the algorithm's name implies, the **errors propagate backwards** from the output nodes to the input nodes. Technically speaking, back propagation calculates the gradient of the error of the network regarding the network's modifiable weights. This gradient is almost always used in a simple stochastic gradient descent algorithm to find weights that minimize the error. Back propagation usually allows quick convergence on satisfactory local minima for error in the kind of networks to which it is suited.

Back propagation networks employ **gradient descent algorithm**. Gradient descent is a first-order **optimization** algorithm. To find a **local minimum** of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function at the current point.

2.9 Advantages of Neural Computing:

There are a variety of benefits that an analyst realizes from using neural networks in their work.

1. Pattern recognition is a powerful technique for harnessing the information in the data and generalizing about it. Neural nets learn to recognize the patterns which exist in the

data set.

2. The system is developed through learning rather than programming. Programming is much more **time consuming** for the analyst and requires the analyst to specify the exact behavior of the model. Neural nets teach themselves the patterns in the data freeing the analyst for more interesting work.

3. Neural networks are **flexible** in a changing environment. Rule based systems or programmed systems are limited to the situation for which they were designed--when conditions change, they are no longer valid. Although neural networks may take some time to learn a sudden drastic change, they are excellent at adapting to constantly changing information.

4. Neural networks can build **informative models** where more conventional approaches fail. Because neural networks can handle very complex interactions they can easily model data which is too difficult to model with traditional approaches such as inferential statistics or programming logic.

5. Performance of neural networks is at least as good as classical statistical modeling, and better on most problems. The neural networks build models that are more **reflective** of the structure of the data in **significantly less time**.

6. Neural networks now operate well with **modest computer hardware**. Although neural networks are computationally intensive, the routines have been optimized to the point that they can now run in reasonable time on personal computers. They do not require supercomputers as they did in the early days of neural network research.

2.10 Limitations of Neural Computing:

There are some limitations to neural computing. The key limitation is the neural network's **inability to explain the model** it has built in a useful way. Analysts often want to know why the model is behaving as it is. Neural networks get better answers but they have a hard time explaining how they got there.

There are a few other limitations that should be understood. First, It **is difficult to extract**

rules from neural networks. This is sometimes important to people who have to explain their answer to others and to people who have been involved with artificial intelligence, particularly expert systems which are rule-based.

As with most analytical methods, you cannot just throw data at a neural net and get a good answer. You have to **spend time** understanding the problem or the outcome you are trying to predict. And, you must be sure that the data used to train the system are appropriate and are measured in a way that reflects the behavior of the factors. If the data are not representative of the problem, neural computing will not product good results. This is a classic situation where "garbage in" will certainly produce "garbage out."

Finally, it can take time to train a model from a very complex data set. Neural techniques are **computer intensive** and will be slow on low end PCs or machines without math coprocessors. It is important to remember though that the overall time to results can still be faster than other data analysis approaches, even when the system takes longer to train. Processing speed alone is not the only factor in performance and neural networks do not require the time programming and debugging or testing assumptions that other analytical approaches do.

2.11 Applications of Neural Networks:

1. Medical Diagnosis:

Eg: Classification of Breast Cancer Cells

Input: 17 morphometric features including object size, object shape, object sum density, object average density, object texture, angular, second moment contrast, difference moment, sum variance, difference variance (Fisher), difference entropy, information measure B, maximum correlation coefficient, difference variance, diagonal moment, second diagonal moment.

Outputs: Well differentiated, moderate, poor, benign

2. Business Applications:

Eg: Credit Scoring Goal: Determine whether a load should be approved based on features extracted from applicant's information.

Inputs: Own/Rent your home, Years with Employer, Credit Cards, Store Account, Bank Account, Occupation, Previous Account, Credit Bureau

Outputs: Credit scores: delinquent, charged-off, or paid-off

3. Energy Cost Prediction:

Eg: Natural gas price prediction

Inputs: Quarter of the year, season, NNG's sales commodity rate last year, NNG's market sensitive price last year, Nat. Gas Week price index last month, Nat. Gas Week price index last year, Degree-days last month

Output: Gas index next month

4. Data processing, including filtering, clustering, blind source separation and compression.

5. Classification, including pattern and sequence recognition, novelty detection and sequential decision making.

6. Function approximation, or regression analysis, including time series prediction, fitness approximation and modeling.

CHAPTER 3: GESTURE RECOGNITION

3.1 What is Gesture Recognition:

Gesture recognition is a topic in computer science and language technology with the goal of interpreting human gestures via mathematical algorithms. Gestures can originate from any bodily motion or state but commonly originate from the face or hand. Current focuses in the field include emotion recognition from the face and hand gesture recognition. Many approaches have been made using cameras and computer vision algorithms to interpret sign language. However, the identification and recognition of posture and human behaviors is also the subject of gesture recognition techniques.

Gesture recognition can be seen as a way for computers to begin to understand human body language, thus building a richer bridge between machines and humans than primitive text user interfaces or even GUIs (graphical user interfaces), which still limit the majority of input to keyboard and mouse. Gesture recognition enables humans to communicate with the machine (HMI) and interact naturally without any mechanical devices. Using the concept of gesture recognition, it is possible to point a finger at the computer screen so that the cursor will move accordingly. This could potentially make conventional input devices such as mouse, keyboards and even touch screens redundant. Gesture recognition is useful for processing information from humans which is not conveyed through speech or type. As well, there are various types of gestures which can be identified by computers. Gesture recognition can be conducted with techniques from **computer vision** and **image processing**.

The literature includes ongoing work in the computer vision field on capturing gestures or more general human pose and movements by cameras connected to a computer.Hand gestures can be classified in two categories: **static** and **dynamic**. A static gesture is a particular hand configuration and pose, represented by a single image. A dynamic gesture

is a moving gesture, represented by a sequence of images. We will focus on the **recognition of static images**.

3.2 Applications of Gesture Recognition:

Recently, there has been a surge in interest in recognizing human hand gestures. Hand gesture recognition has various applications like -

- **Sign language recognition**. Just as speech recognition can transcribe speech to text, certain types of gesture recognition software can transcribe the symbols represented through **sign language** into text.
- **Directional indication through pointing**. Pointing has a very specific purpose in our society, to reference an object or location based on its position relative to ourselves. The use of gesture recognition to determine where a person is pointing is useful for identifying the context of statements or instructions. This application is of particular interest in the field of **robotics**.
- **Control through facial gestures**. Controlling a computer through facial gestures is a useful application of gesture recognition for users who may not physically be able to use a mouse or keyboard. **Eye tracking** in particular may be of use for controlling cursor motion or focusing on elements of a display.
- **Immersive game technology**. Gestures can be used to control interactions within video games to try and make the game player's experience more interactive or immersive.

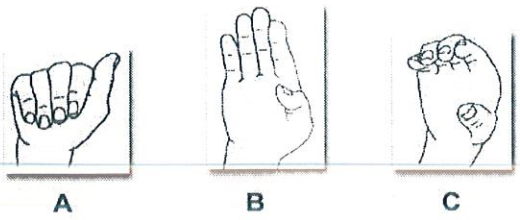


Fig 6: ASL Examples

The main goal of the project is implementation of a **Human Computer Interaction** system having the following open ended set of features-

- **Handwritten Character Recognition**
- **Gesture Recognition**

and finally integrating these set of features under a GUI and testing it in a robotic platform and interacting with the machine in a natural way, the way we interact among ourselves in our day to day lives.

In other words, the objective of our project is **two-fold:**

Firstly, to develop a system able to **recognize static hand gestures/handwritten characters** using **neural networks**. We would train our system in the learning phase and then test our system with test examples and calculate the accuracy of the system.

Secondly, to **control** any day to day used computer application like **Windows Media Player** using our system, i.e. we can control its functioning by static hand gestures or characters as the case may be.

CHAPTER 4: METHODOLOGY

The basic methodology in any supervised learning problem starts with the collection of training dataset i.e. samples containing both input and output values. In case of digit recognition we started by gathering a dataset of 5000 samples of handwritten digits from 0 to 9 and then trained the neural network using backpropagation algorithm as a multi class classifier. Similarly in case of Gesture recognition we started by gathering a data set of five gestures. From each gesture we extracted a feature vector of length 19 using orientation histogram method. This feature vector was used to train the neural network . When the neural network got trained, we stored the network parameters and used it to recognize gesture. Finally different instructions were given to the media player corresponding to different hand gestures to play, pause, stop the music.

4.1 Collection of Image Database:

The starting point of the project was the creation of a database with all the images that would be used for training and testing. The image database can have different formats. Images can be either hand drawn, digitized photographs or a 3D dimensional hand. Photographs were used, as they are the most realistic approach.

Images came from internet database and they have different sizes, different resolutions and at times almost completely different angles of shooting. Two operations were carried out in all of the images. They were converted to grayscale and the background was made uniform. The internet databases already had uniform backgrounds.

The database itself was constantly changing throughout the completion of the project as it was it that would decide the robustness of the algorithm. Therefore, it had to be done in such way that different situations could be tested and thresholds above which the algorithm didn't classify correct would be decided.

The construction of such a database is clearly dependent on the application. If the application

is a crane controller for example operated by the same person for long periods the algorithm doesn't have to be robust on different person's images. In this case noise and motion blur should be tolerable. The applications can be of many forms and since we were not developing for a specific one we have tried to experiment for many alternatives.

We can see figure 7. In the first row are the training images. In the second row are the testing images.

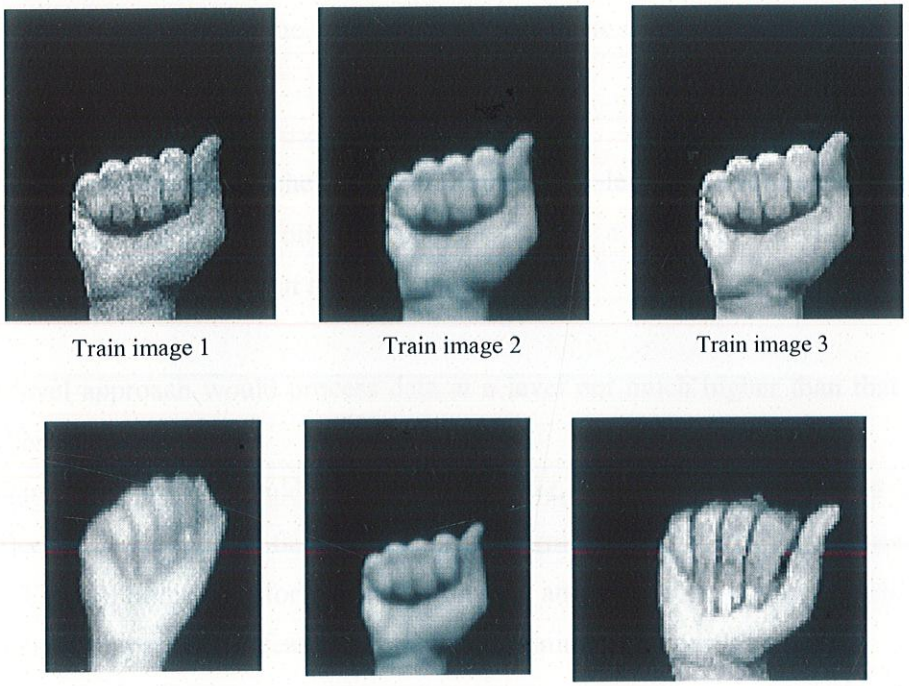


Figure 7: Train – Test images

For most of the gestures the training set originates from a single gesture. Those were enhanced in MATLAB using various filters. The reason for this is that we wanted the algorithm to be very robust for images of the same database. If there was a misclassification to happen it would be preferred to be for unknown images.

The final form of the database is as follows:

Train set:

Five training sets of images, each one containing twenty five images. Each set originates from a single image for testing.

Test Set:

The number of test images varies for each gesture. There is no reason for keeping those on a constant number. Some images can tolerate much more variance and images from new databases and they can be tested extensively, while other images are restricted to fewer testing images.

The system could be approached either in high or low-level. The former would employ models of the hand, finger, joins and perhaps fit such a model to the visual data. This approach offers robustness, but at the expense of speed.

A low-level approach would process data at a level not much higher than that of pixel intensities.

Although this approach would not have the power to make inferences about occluded data, it could be simple and fast. The pattern recognition system that will be used can be seen in Fig (8). Some transformation T, converts an image into a feature vector, which will be then compared with feature vectors of a training set of gestures.

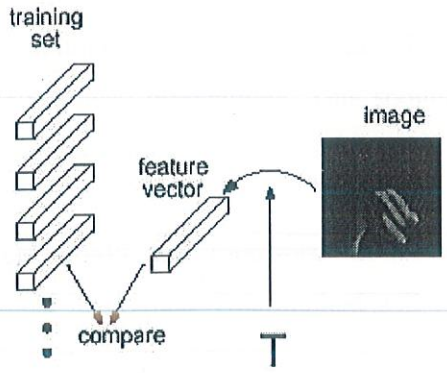


Figure 8: Pattern Recognition System

We will be seeking for the simplest possible transformation T, which allows gesture recognition.

Histogram orientation has the advantage of being robust in lighting change conditions. If we follow the pixel-intensities approach certain problems can arise for varying illumination. Taking a pixel-by-pixel difference of the same photo under different lighting conditions would show a large distance between these two identical gestures. For the pixel-intensity approach no transformation T has been applied. The image itself is used as the feature vector. In Fig (9) we can see the same hand gesture under different lighting conditions.

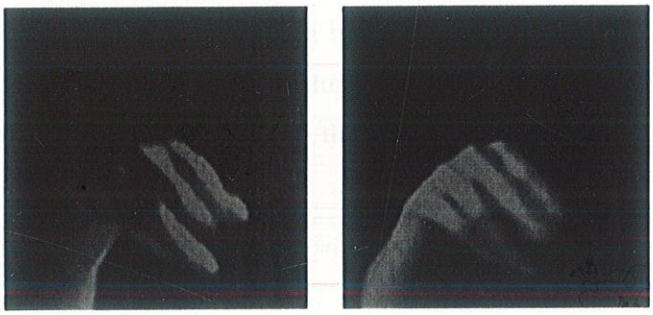


Figure 9: Illumination Variance

Another important aspect of gesture recognition is translation invariance. The position of the hand within the image should not affect the feature vector. This could be enforced forming a local histogram of the local orientations. This should treat each orientation element the same, independent of location.

Therefore, orientation analysis should give robustness in illumination changes while histogram will offer translational invariance. This method will work if examples of the same gesture map to similar orientation histograms, and different gestures map to substantially different histograms.

4.2 Feature Extraction through Orientation Histogram:

We want gestures to be the same regardless of where they occur with the images boarders. To achieve this we will ignore position altogether, and tabulate a histogram of how often each orientation element occurred in the image. Clearly, this throws out information and some distinct images will be confused by their orientation histograms. In practice, however, one can choose a set of training gestures with substantially different orientation histograms from each other.

One can calculate the local orientation using image gradients. We used two 3 – tap x and y derivative filters. The outputs of the x and y derivative operators will be dx and dy. Then the gradient direction is ***atan (dy/dx).*** We had decided to use the edge orientation as the only feature that will be presented to the neural network. The reason for this is that if the edge detector was good enough it would have allowed me to test the network with images from different databases. Another feature that could have been extracted from the image would be the gradient magnitude using the formula below

$$\sqrt{dx^2 + dy^2}$$

This would lead though to testing the algorithm with only similar images. Apart from this the images before resized should be of approximately the same size. This is the size of the hand itself in the canvas and not the size of the canvas. Once the image has been processed the output will be a single vector containing a number of elements equal to the number of bins of the orientation histogram.

Figure 10 shows the orientation histogram calculation for a simple image. Blurring can be used to allow neighboring orientations to sense each other.

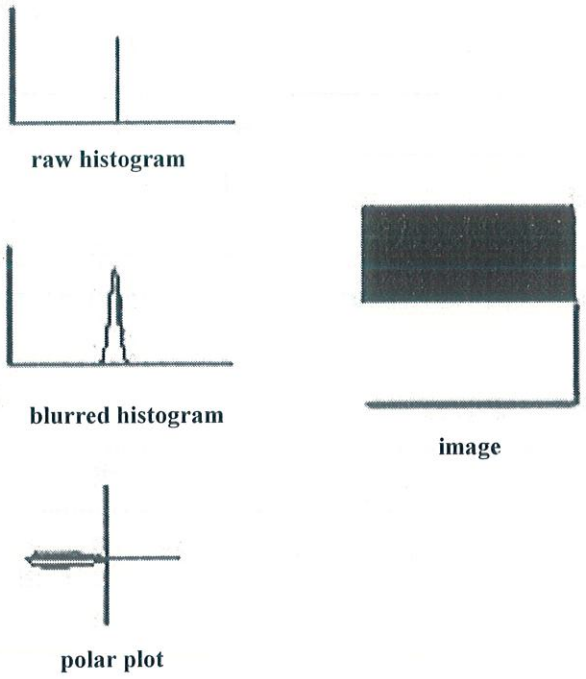


Figure 10: Orientation histogram

4.2.1 Operation:

The program can be divided in six steps. Let us examine them one by one.

Step1:

The first thing for the program to do is to **read the image database**. A *for* loop is used to read an entire folder of images and **store them in MATLAB's memory**. The folder is selected by the user from menus. A menu will firstly pop-up asking you whether you want to run the algorithm on test or train sets. Then a second menu will pop-up for the user to choose which ASL sign he wants to use.

Step2:

Resize all the images that were read in Step1 to **150x140 pixels**. This size seems the optimal for offering enough detail while keeping the processing time low.

Step3:

Next thing to do is to **find the edges**. Two filters were used.

For the x direction x = [0 -1 1] and For the direction y = [0 1 -1]'

Step 4:

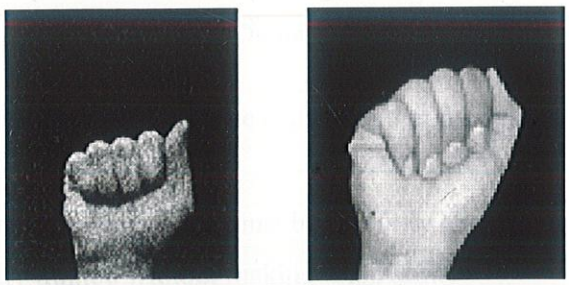
Dividing the two resulting matrices (images) *dx* and *dy* element by element and then taking the atan (dy/dx). This will give the **gradient orientation**.

Step 5:

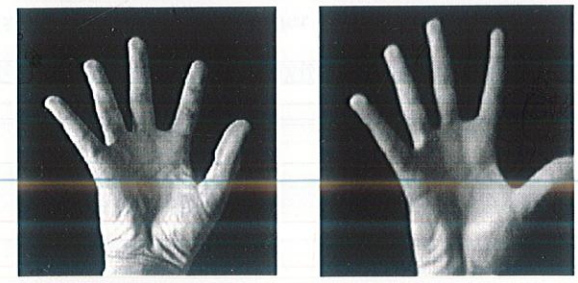
Orientation Binning: It aims that local object appearance and shape within an image can be described by the distribution of edge directions. It is achieved by dividing the images into small connected regions called cells and for each cell compiling a histogram of gradient orientations for pixels within the cell.

Step 6:

Input this feature vector in the ANN.



Original image a_1 **Original image a_2**



Original image 5_1 **Original image 5_2**

Figure 11 : Training image

CHAPTER 5: RESULT

5.1 Complexities involved:

Under this heading we are going to mention the complexities we faced while building this project:

- **Designing the basic structure of our Neural Network system**: There are various factors taken into consideration while designing the system such as the number of hidden layer nodes and the activation function of the neuron.

- **Initializing the weights of the system**: If the weights are assigned large initial values, neurons are driven into saturation whereas if weights are too small, algorithm operates around the origin.

- **Algorithm to be used for feature extraction**: After testing for different algorithms, we found that Orientation Histogram provided the best results.

- An **accurate dataset** must be chosen so that the system is properly trained.

- The **number of iterations** must be carefully chosen so that the system can be effectively trained without making it too slow.

- The **learning rate** should neither be too large nor too small. Hence choosing the learning rate is another complexity we need to counter.

5.2: Analysis:

- We have successfully trained our artificial neural network with the suitable dataset for **Digit Recognition** and **Static Hand Gesture Recognition**.

- The testing accuracy for gesture recognition system had been calculated for different hidden layer sizes :

Hidden layer size	Accuracy (%)
1000	88.07
5000	90.10
7500	96.00
10000	97.42
20000	97.84

Table 1: Variation of accuracy of gesture recognition system with hidden layer size

In case of digit recognition, the variation of accuracy with hidden layer size was:

Hidden layer size	Accuracy (%)
10	93.32
15	95.10
25	98.04
35	98.42
50	97.84
100	96.02

Table 2: Variation of accuracy of digit recognition system with hidden layer size

Inference: We select **optimum middle layer size to be 7500 in case of gesture recognition and 25 for digit recognition** as for lower sizes of hidden layer, the accuracy as not good and for higher higher layer sizes, the complexity of computation increases. Therefore, we select the lowermost value of layer size for which we get a desired accuracy.

- Finally we were successful in **controlling Windows Media Player functions** like play, pause, next, previous and stop using various hand gestures.

5.3 Output:

```
Do you what to continue (Y/N)
y
MEDIA PLAYER OPTIONS
A.PLAY
B.PAUSE
C.NEXT
D.PREVIOUS
E.STOP
Enter the test example you what to use:"A,B,C,D or E"
A
```

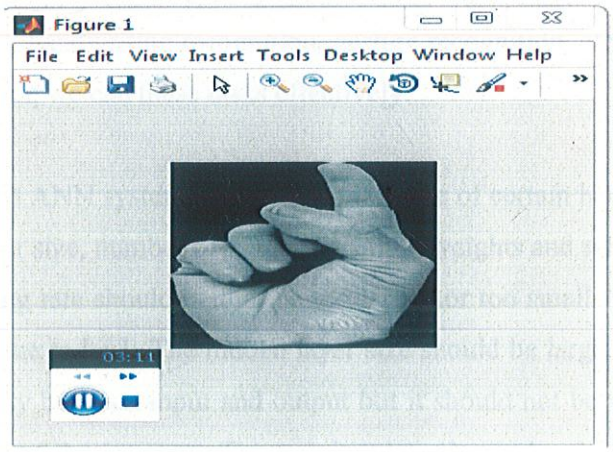


Fig 12 : Using NN to control Media Player

Here, we have selected test image "A" which contains gesture 1. We have used gesture 1 to play the current item on the playlist. Similarly, choosing other gestures we can pause and stop the music on the playlist.

CHAPTER 6: CONCLUSION AND FUTURE WORK

6.1 Conclusion:

HCI is widely being used these days in the design of user-friendly machines like smart TV, laptops, robots, smartphones etc. We have developed our interface with regard to the complexity of a general computer system and the limitations of humans. The same system can also be deployed in a robot or a smart television with minor adjustments.

With the evolution of big data, performance of Human machine interface can be subsequently improved by networking it with global repositories which keep on updating in the real time. So interaction will become even more effective as more and more training data is made available to it.

We have seen that artificial neural network can be effectively used to perform a wide variety of supervised learning tasks which form the basis of human computer interactive systems. It can be used to perform tasks like voice recognition, human emotion recognition, handwriting recognition, gesture recognition, etc. Hence by building a robust ANN system, we can these tasks effectively.

For a robust ANN system, we need to take care of certain heuristics such as learning rate, hidden layer size, number of iterations, initial weights and selection of training data.
The learning rate should neither be too large nor too small. We experimentally found its optimal value to be 1. The hidden layer size should be large enough to accommodate the non-linearity between input and output but it should not be very large as it increases the complexity of the program. The training data should be chosen such as to maximize the information content. Initial weights should be chosen as small non-zero values for symmetry breaking.
Taking care of these heuristics, we were successful in designing a generic multi class classifier able to recognize digits and hand gestures.

6.2 Future Work:

Since the scope of human computer interaction is very wide, we will extend our interface to control more daily used applications such as Windows Explorer, web browsers, start and shut down the PC using different techniques like **dynamic hand gesture recognition**, **voice command recognition**, **facial expression** and **handwriting recognition**.

We will implement our interface on a robotic platform to enable it to perform advanced cognition functionalities like vision, hearing, reading, understanding human mood and act accordingly. For example if a human near the robot is depressed, it will play a soothing music.

We will use the neural network in other domains such as finance, medical diagnostics to perform advanced prediction tasks such as stock market forecast, cancer prediction.

REFERENCES

1. Simon Haykin, "Neural Networks, A comprehensive Foundation" 2d ed., Prentice Hall
2. Duane Hanselman, Bruce Littlefield, "Mastering MATLAB, A comprehensive tutorial and reference" 1st ed., Prentice Hall
3. Christopher M. Bishop, "Neural networks for Pattern Recognition" Oxford, 1995.
4. Maria Petrou, Panagiota Bosdogianni, "Image Processing, The Fundamentals" 1st ed., Wiley
5. N. Sivanandam, S. N Deepa, "Introduction To Neural Networks using Matlab 6.0" 1st ed., Tata McGraw-Hill Education.
6. Klimis Symeonidis, "Hand Gesture Recognition Using Neural Networks"
7. http://www.kaggle.com/digit
8. http://www.coursera.com/digit
9. http://www.tk.uni-linz.ac.at/~schaber/ogr.html
10. http://vismod.www.media.mit.edu/vismod/classes/mas622/projects/hands/

APPENDIX A

Digit Recognition Code:

```
clear ;
close all;
clc;

input_layer_size  = 400;
hidden_layer_size = 25;
out_layer_size = 10;

load('data.mat');
m = size(X, 1);

Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
Theta2 = randInitializeWeights(hidden_layer_size, out_layer_size);

nn_params = [Theta1(:) ; Theta2(:)];

% Weight regularization parameter 1
lambda = 1;

J = nnCostFunction(nn_params, input_layer_size, hidden_layer_size,out_layer_size, X,
y, lambda);

options = optimset('MaxIter', 100);

lambda = 1;
```

```
costFunction = @(p)
nnCostFunction(p,input_layer_size,hidden_layer_size,out_layer_size,X, y, lambda);

[nn_params, cost] = fmincg(costFunction, nn_params, options);


Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)),
hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size +
1))):end),out_layer_size, (hidden_layer_size + 1));

pred = predict(Theta1, Theta2, X);

fprintf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) * 100);

load('test5.mat');
imshow(test5);
test_double=im2double(test5);
test_gray=rgb2gray(test_double);
test=mat2vec(test_gray');
pred1= predict(Theta1, Theta2,test);
display('The predicted output is :');
display(pred1);
```

Functions used:

nn_costfunction( )

```
function [J grad] =
nnCostFunction(nn_params,input_layer_size,hidden_layer_size,output_layer_size,X, y,
lambda)
    Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size +
1)),hidden_layer_size, (input_layer_size + 1));
    Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size +
1))):end),output_layer_size, (hidden_layer_size + 1));

m = size(X, 1);
J = 0;
Theta1_grad = zeros(size(Theta1));
Theta2_grad = zeros(size(Theta2));

a1=X';
a1=[ones(1,size(a1,2)); a1];
z2=Theta1*a1;
a2=sigmoid(z2);
a2=[ones(1,size(a2,2)); a2];
z3=Theta2*a2;
a3=sigmoid(z3);
h=a3';

Y = repmat(y,1,10)==repmat([1:10],5000,1);

for i=1:m
  for k=1:output_layer_size
    J=J+[-Y(i,k).*log(h(i,k))-(1-Y(i,k)).*log(1-h(i,k))];
  end
end
```

```
J=J/m;

 DELTA1=zeros(hidden_layer_size,(input_layer_size +1));
 DELTA2=zeros(output_layer_size,(hidden_layer_size +1));

 delta3=h-Y;
 delta2=[(Theta2'*delta3').*(a2.*(1-a2))]:
 delta2=delta2(2:end,:);

 DELTA1=DELTA1+(delta2*a1');
 DELTA2=DELTA2+(delta3'*a2');

 Theta1_grad=(1/m).*DELTA1;
 Theta2_grad=(1/m).*DELTA2;

grad = [Theta1_grad(:) ; Theta2_grad(:)]:

for j=1:hidden_layer_size
   for k=1:input_layer_size
      J=J+((lambda/(2*m)).*Theta1(j,k)^2);
   end
end

for j=1:10
   for k=1:hidden_layer_size
      J=J+((lambda/(2*m)).*Theta2(j,k)^2);
   end
end
 end
```

APPENDIX B

Gesture Recognition code:

```
clc;
clear all;
cont='Y';
while cont=='Y'||cont=='y'
fprintf('MENU: \n1.TRAINING THE NEURAL NETWORK FOR GESTURE RECOGNITION \n2.TESTING THE NEURAL NETWORK \n3.CONTROLLING THE MEDIA PLAYER WITH GESTURE RECOGNITION\n');
  choice=input('Enter your choice\n');
  if choice==1
     training_nn;
  else if choice==2
     testing_nn;
     else if choice==3
          control_media_player;
        else
           fprintf('Enter a valid choice.\n');
        end
     end
     end
cont=input('Do you want to continue (Y/N) \n','s');
end
break;
```

Functions used:

1) training_nn( )

```
clc;
clear all;
store_traindata;
feature_extraction;
load('t.mat');

m = size(X,1);

input_layer_size  =size(X,2) ;
hidden_layer_size = 5000;
out_layer_size =5;


Theta1 = randInitializeWeights(input_layer_size, hidden_layer_size);
Theta2 = randInitializeWeights(hidden_layer_size, out_layer_size);

nn_params = [Theta1(:) ; Theta2(:)];

%  regu. parameter 1
lambda = 1;

J = nnCostFunction(nn_params, input_layer_size, hidden_layer_size,out_layer_size, X,
y, lambda);

options = optimset('MaxIter', 82);

lambda = 1;
costFunction = @(p)
nnCostFunction(p,input_layer_size,hidden_layer_size,out_layer_size,X, y, lambda);
```

```
[nn_params, cost] = fmincg(costFunction,nn_params, options);

Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)),
hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size +
1))):end),out_layer_size, (hidden_layer_size + 1));

pred = predict(Theta1, Theta2, X);
save ('nnsystem.mat','Theta1','Theta2');
printf('\nTraining Set Accuracy: %f\n', mean(double(pred == y)) *100);
```

2) testing_nn( )

```
clc;
store_testdata;
feature_extraction;
load('t.mat');% loads X,y
load('nnsystem.mat');%loads Theta1,Theta2
pred = predict(Theta1, Theta2, X);
fprintf('\nTesting Set Accuracy: %f\n', mean(double(pred == y)) * 100);
```

Controlling media player functions Code:

```
clc;
clear all;
cont='y';
while cont=='y'||cont=='Y'
   fprintf('MEDIA PLAYER OPTIONS \n1.PLAY \n2.PAUSE \n3.NEXT
\n4.PREVIOUS \n5.STOP\n');
   count=1;
   c=input('Enter the test example you what to use :A,B,C,D or E:\n','s');
```

```
myFolder=strcat('C:\Users\Aman\Desktop\project\gesture recognition\music\test',c);
filePattern = fullfile(myFolder, '*.png');
jpegFiles = dir(filePattern);

for k = 1:length(jpegFiles)

    baseFileName = jpegFiles(k).name;
    fullFileName = fullfile(myFolder, baseFileName);
    imageArray = rgb2gray(imread(fullFileName));
    imageArray = imresize(imageArray,[150,140]);
    imshow(imageArray);
    T{count}=imageArray;
    count=count+1;
end

y=0;

save('data.mat','T','y');

feature_extraction;
load('t.mat');% loads X,y
load('nnsystem.mat');%loads Theta1,Theta2
pred = predict(Theta1, Theta2, X);

if pred==1
    control = actxcontrol('WMPlayer.ocx.7'); % Create Controller
    plylst= control.newMedia('C:\Users\Aman\Desktop\project\gesture
recognition\music\plylst.wpl'); % Create Media object
    control.currentMedia = plylst;
    control.Controls.play;
```

```
    fprintf(strcat('You have selected gesture '',num2str(pred),''.The current item on the
playlist is being played\n'));
  else if pred==2
    control.Controls.pause;
    fprintf(strcat('You have selected gesture '',num2str(pred),''.The current item on the
playlist has being paused\n'));

  else if pred==3
    control.Controls.next;
    fprintf(strcat('You have selected gesture '',num2str(pred),''.The next item on the
playlist is being played\n'));

  else if pred==4
    control.Controls.previous;
    fprintf(strcat('You have selected gesture '',num2str(pred),''.The previous item on the
playlist is being played\n'));
  else if pred==5
    control.Controls.stop;
    fprintf(strcat('You have selected gesture '',num2str(pred),''.The current item on the
playlist has being stopped\n'));
    else
    fprintf('Please enter a valid option.\n')
    end
    end
    end
    end
  end

cont=input('Do you what to continue (Y/N) \n','s');
end
break;
```

Feature extraction code:

```
clc
%clear all;

load('data.mat'); %variable train is loaded in its workspace

filtx = [0 -1 1];
filty = [0 1 -1]';

for i=1:length(T)
dx{i} = convn(T{i},filtx,'same');
dy{i} = convn(T{i},filty,'same');

gradient{i} = dy{i} ./dx{i};
theta{i} = atan(gradient{i});

cl{i}= im2col(theta{i},[1 1],'distinct');
N{i} = (cl{i}*180)/(22/7);

c1{i}=(N{i}>0)&(N{i}<10);
s1{i}=sum(c1{i});
c2{i}=(N{i}>10.0001)&(N{i}<20);
s2{i}=sum(c2{i});
c3{i}=(N{i}>20.0001)&(N{i}<30);
sum(c3{i});
s3{i}=sum(c3{i});
c4{i}=(N{i}>30.0001)&(N{i}<40);
sum(c4{i});
s4{i}=sum(c4{i});
c5{i}=(N{i}>40.0001)&(N{i}<50);
sum(c5{i});
```

```
s5{i}=sum(c5{i});
c6{i}=(N{i}>50.0001)&(N{i}<60);
sum(c6{i});
s6{i}=sum(c6{i});
c7{i}=(N{i}>60.0001)&(N{i}<70);
sum(c7{i});
s7{i}=sum(c7{i});
c8{i}=(N{i}>70.0001)&(N{i}<80);
sum(c8{i});
s8{i}=sum(c8{i});
c9{i}=(N{i}>80.0001)&(N{i}<90);
sum(c9{i});
s9{i}=sum(c9{i});
c10{i}=(N{i}>90.0001)&(N{i}<100);
sum(c10{i});
s10{i}=sum(c10{i});
c11{i}=(N{i}>-89.9)&(N{i}<-80);
sum(c11{i});
s11{i}=sum(c11{i});

c12{i}=(N{i}>-80.0001)&(N{i}<-70);
sum(c12{i});
s12{i}=sum(c12{i});
c13{i}=(N{i}>-70.0001)&(N{i}<-60);
sum(c13{i});
s13{i}=sum(c13{i});
c14{i}=(N{i}>-60.0001)&(N{i}<-50);
sum(c14{i});
s14{i}=sum(c14{i});
c15{i}=(N{i}>-50.0001)&(N{i}<-40);
sum(c15{i});
```

```
s15{i}=sum(c15{i});
c16{i}=(N{i}>-40.0001)&(N{i}<-30);
sum(c16{i});
s16{i}=sum(c16{i});
c17{i}=(N{i}>-30.0001)&(N{i}<-20);
sum(c17{i});
s17{i}=sum(c17{i});
c18{i}=(N{i}>-20.0001)&(N{i}<-10);
sum(c18{i});
s18{i}=sum(c18{i});
c19{i}=(N{i}>-10.0001)&(N{i}<-0.0001);
sum(c19{i});
s19{i}=sum(c19{i});

D{i}= [s1{i} s2{i} s3{i} s4{i} s5{i} s6{i} s7{i} s8{i} s9{i} s10{i} s11{i} s12{i} s13{i}
s14{i} s15{i} s16{i} s17{i} s18{i} s19{i}];
X(i,:)=D{i};
end;
save('t.mat','X','y');
```