USING ARTIFICIAL LIFE TO DESIGN MACHINE LEARNING ALGORITHMS FOR DECODING GENE EXPRESSION PATTERNS FROM IMAGES

By

Shaza B. Zaghlool

Thesis submitted to the faculty of the Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of

Master of Science
In
Biomedical Imaging

Christopher L. Wyatt, Chair
Jason Xuan
William T. Baumann

April 30, 2008

Alexandria, Egypt

Keywords: Genotype, Phenotype, Biomorph, Classifier

USING ARTIFICIAL LIFE TO DESIGN MACHINE LEARNING ALGORITHMS FOR DECODING GENE EXPRESSION PATTERNS FROM IMAGES

Shaza B. Zaghlool

ABSTRACT

Understanding the relationship between gene expression and phenotype is important in many areas of biology and medicine. Current methods for measuring gene expression such as microarrays however are invasive, require biopsy, and expensive. These factors limit experiments to low rate temporal sampling of gene expression and prevent longitudinal studies within a single subject, reducing their statistical power. Thus methods for non-invasive measurements of gene expression are an important and current topic of research. An interesting approach (Segal et al, Nature Biotechnology 25 (6) 2007) to indirect measurements of gene expression has recently been reported that uses existing imaging techniques and machine learning to estimate a function mapping image features to gene expression patterns, providing an image-derived surrogate for gene expression. However, the design of machine learning methods for this purpose is hampered by the cost of training and validation.

My thesis shows that populations of artificial organisms simulating genetic variation can be used for designing machine learning approaches to decoding gene expression patterns from images. If analysis of these images proves successful, then this can be applied to real biomedical images reducing the limitations of invasive imaging. The results showed that the box counting dimension was a suitable feature extraction method yielding a classification rate of at least 90% for mutation rates up to 40%. Also, the box-counting dimension was robust in dealing with distorted images. The performance of the classifiers using the fractal dimension as features, actually, seemed more vulnerable to the mutation rate as opposed to the applied distortion level.

TABLE OF CONTENTS

LIST OF FIGURES

Number *Page*

LIST OF TABLES

ACKNOWLEDGMENTS

The author wishes to express sincere appreciation to Professor Wyatt for his assistance in the preparation of this thesis. In addition, special gratitude goes to the author's family for their constant love, support, and faith.

ABBREVIATIONS

IFS: Iterated Function System

kNN: k- Nearest Neighbors

LVQ: Learning Vector Quantization

NN: Neural Network

RBF: Radial Basis Function

ROC: Receiver Operator Characteristic

SMO: Sequential Minimal Optimization

SVM: Support Vector Machine

C h a p t e r 1

INTRODUCTION AND BACKGROUND

1.0 Introduction

It was discovered that the global gene expression of patterns of some human tissues can correlate with their dynamic image features [1]. In other words, extraction of certain image characteristics can be used to draw conclusions regarding the genetic content of the tissue being imaged. This was a very important discovery as the usage of non-invasive imaging might no longer necessitate invasive surgery, tissue procurement, or cell destruction. The gene expression level of a tissue can indicate substantial information about the disease or therapy. The downside is that present techniques for molecular profiling involve some form of invasive surgery thus limiting their usage. Also, along with being extremely expensive, current profiling methods cause the destruction of cells and so only provide single snapshots in time. Therefore, using non-invasive imaging in the decoding of global gene expression is a promising direction.

The difficulty in designing, training, and validating algorithms using real data, results from the complexity of the association required to link the imaging features with gene expression patterns. The imaging features must be extremely visible in order to be extracted for their incorporation in an association mapping them to their gene expression patterns. The extraction of the appropriate features can be quite difficult with such data. As an alternative, images of artificial life, known as "biomorphs", can be used to model real data.

The concept of biomorphs was first introduced by Richard Dawkins in "The Blind Watchmaker" [2]. Biomorphs are visual representations of a set of genes. Although they are not real living organisms, they possess simplified life-like characteristics and they are easy to obtain as they can be generated quickly and in vast quantities with a computer program. The physical appearance of a biomorph is based on a number of genes that

indicate the figure's morphology. Using examples of artificial life, particular genetic features regarding the organism's genotype, can be inferred.

Random mutation followed by non-random selection can lead to interesting, complex forms. Starting with a small number of genes determining the phenotype, billions of combinations can be included in the genetic space. Instead of using regular shapes or lines as Dawkins did, fractal parameters can be associated with biomorph appearance. Fractal parameters can be incorporated within the genome of the artificial organism and represented visually as a fractal structure. The incorporation of fractal structure in the phenotype of an artificial organism could be used to model biological tissues or structures such as blood vessels or trabecular bone [3]. Also the bronchi of human lungs appear to possess fractal properties by being self-similar over 15 successive bifurications [3]. Moreover, recent discoveries in brain research claim that a hexagonal fractal structure represents the organization of receptive fields in the visual cortex [3]. Fractal analysis of bones [4] has been done to differentiate between healthy and diseased bone tissue. In essence, a blood vessel network or bone could be the "organism" the biomorph would be modeling, which could be a good precursor to later work in actual animal models.

The No Free Lunch theorem [5] states that without prior knowledge any learning algorithm would perform just as well as random guessing across all problems. Therefore, it is important in machine learning for the model to match the real system. Because fractals are a good model of bone and other biological tissues, they are also a good match to the problem of decoding gene expression from imaging data.

My thesis states that by using artificial life, actual biological tissues or structures could be modeled. This model can serve as an aid to help design suitable machine learning algorithms for decoding the gene expression from those biological tissues. Being able to decode the gene expression of biological matter is beneficial because no longer will invasive surgery, with all its drawbacks, be necessary.

The remainder of this thesis is presented as follows. The rest of this chapter contains the literature review and background for this work. The second chapter contains details

regarding the methods used, specifications, and parameter values. The third chapter describes the experimental setup involved in the various experiments carried out and illustrates the obtained results. Finally, the fourth chapter provides an analytical review of the results and suggests interpretation and reasoning behind the obtained experimental results.

1.1 Related Work

A great deal of work has been carried out in the field of biomedical imaging over the past couple of decades. This section covers several of the most relevant contributions of other researchers working in fields particularly related to this thesis. Related work is presented in order of most relevance.

A study done by D. Ashlock and J. Golden (2000) [6] was performed where fractals were generated by driving iterated function systems (IFS) with uniform random numbers. The IFS was driven by a biased source that imitated DNA and produced both DNA fragments with and without stop codons. An evolutionary algorithm was used to generate derived fractals that were viewed graphically and represent the open reading frame (part encoding the physical expression) of the DNA. A number of biological issues were used when designing the biological representation for a fractal. The map from nucleic acid to protein read DNA in triples, to produce 64 codons, which in turn were taken by a many-one map (the genetic code) onto the 20 amino acids and a "stop" codon. This stop codon specified the end of transcription of a certain sequence of DNA.

The data structure or chromosome used to hold the evolvable fractal had two parts; a list of similitudes (models having similarity with the real application) and an index of DNA triples into that list of similitudes. This permitted smooth use of the fractal on DNA, DNA triplets, or amino acids by simply modifying the way the DNA or amino acids were interpreted by the indexing function. The index list is simply a sequence of 64 integers that denote, for each of the 64 possible DNA codon triplets, which similitude to apply when that triplet is come across. In order to derive a fractal from DNA, the DNA was separated into triplets with a specific reading frame. These triplets were then used, via the

index portion of the gene, to choose a similitude to apply to the moving point. This permitted evolution to choose the shape of the maximal fractal (the one we would see if we drove the process with data chosen uniformly at random) and which DNA codon triplets were associated with the use of each similitude.

When applying the evolutionary algorithm, to the chromosome, the variation operators were specified. The variation operators included a crossover operator and a mutation operator. Two sets of simulations were performed. The first used a pair of random sources, one of which had no stop codons and the other of which generated DNA with stop codons. The second set of simulations used the genome of mycobacterium tuberculosis, in two different reading frames, cyclically, in place of the random sources. For a given fractal, fitness was estimated by calculating the average position of the moving point for each type of data: with/without stop codons or in-frame/out of frame natural DNA. The claim of the concept for an evolvable fractal representation proved successful despite the less than perfect fitness function. This work is similar to what was done in my thesis in the derivation of evolvable fractals from DNA.

Also, Jayalalitha and Uthayakumar (2007) [7] classified skin cancer by fractal models, which are based in vitro and by approaching the cells potential in a hierarchical way. The classification framework was probabilistic and computerized. The framework included feature extraction of features such as irregular border, color and diameter in a robust manner. Malignant melanoma, a skin cancer, appears visually as a dark lesion, most usually with an irregular boundary. The degree of irregularity was an important diagnostic indicator. The Box counting method [8] and the Sausage method [7] were used to find out the dimensions of the affected cells in an accurate manner. This fractal approach led to very promising results which improved the determination and examination of skin cancer. This was related to my work by using the same feature extraction method, and applying classification strategies for diagnostic purposes.

Moreover, Samarabandu et al (1993) [9] have applied mathematical morphology for fractal analysis of bone x-ray images. The digitized gray level image was treated as a three-dimensional surface whose fractal dimension was calculated by performing a series

of dilations on this surface and plotting the area of the resulting set of surfaces against the size of the structuring element. This approach gave the added advantage of encoding structural information via the use of a structuring element. The algorithm was applied to several bone radiographs and the results showed that the fractal dimension using mathematical morphology produced a robust texture measure of trabecular bone structures. This relates to my thesis by the use of fractal analysis techniques for analyzing biological material having fractal properties.

In a book by D. Ashlock and J. Golden (2002) [10] fractals were used to express several types of information including shape and color in a single picture. They reported on a standard method for visualizing DNA or other sequence data with a fractal algorithm and then generalized the technique to get two types of evolvable fractals. Both were forms of IFS, collections of randomly driven or data driven contraction maps. The first, an indexed IFS, used incoming data to choose which contraction map will be applied next. The second, called a chaos automata, drove a finite state machine with incoming sequence data and associated a contraction map with each state. Both evolvable fractals were tested on their ability to differentiate between DNA from distinct microbial genomes. Fitness functions were designed since that is a crucial concern in trying to make fractals that look good and express information about the sequences driving them. It was verified that chaos automata were superior to the indexed IFS on the test problem. They also discussed possible enhancements in the fractal chromosomes, fitness functions, and issues to be resolved to obtain useful applications. The concept of visualizing DNA relates to my thesis in the evolutionary process and biomorph generation.

Randomized fractal algorithms were used to visualize DNA. The central idea basically involved replacing random numbers with DNA bases in an algorithm that generated a fractal and the resulting modified fractal was a visualization of the DNA. The fractal quality of DNA, with its significant component of non-randomness, forced the DNA driven fractals to look distinct from fractals produced from streams of random numbers. Three technologies for producing DNA driven fractals were presented. The first was a modification of the well known chaos game and did not require evolutionary

computation. It served as a point of departure for evolvable fractal generating algorithms. The second technology was an evolvable form of iterated function system (IFS). The third technology was also evolvable and combined IFS technology with a finite state automata to produce a structure that could respond to long range effects in DNA. The goal was to use a data driven fractal to provide a visual representation of sequence data since it would be nice if this fractal representation could work efficiently with DNA, protein, and codon data. Although these sequences were derived from one another, they had varying amounts of information and appeared at different levels of the biological process that run a cell.

M. Baldoni et al (2000) [11], also used IFS codes for learning 2D isolated –object classifying systems. It was shown that fractal features obtained from IFS encodings contained the sort of information that was required by learning systems and, hence, allowed the successful classification of 2D images of objects. Recognition of complex images is a tough and computationally expensive task, primarily because it is very difficult to automatically confine with a few features the necessary discriminant information. Given that such features were available, an appropriate learning system was able to be trained to distinguish images of different kinds of objects, starting from a set of labeled examples. The classification task becomes more tractable as the number of input features is smaller. Working at a more abstract level of detail is better, extracting from the raw data the most meaningful properties of the represented object. These properties usually correspond to properties of parts of the image or of the image as a whole (colors, brightness, etc.) and to relations between the different parts (position, similarity, etc.). M. Baldoni et al (2000) [11] also presented a fractal feature extraction algorithm and reported the classification results obtained on two extremely different test-beds by applying Machine Learning techniques to sets of encoded images. The practice of extracting the most meaningful characteristics from fractal images and performing machine learning directly relates to my work.

More work was done by A. Murugan and K. Easwarakumar (2007) [12] where they introduced the SInsDelP system with single contextual insertion and deletion. They

applied this SInsDelP system for the creation of fractal images which was performed in two stages. A DNA sequence for the fractal image was generated first by a sequence of SInsDelP operations, and then the respective image was produced through the specialized recognition algorithm. Only insertion and deletion operators were used in their approach for the construction of the DNA sequence corresponding to a fractal. Every use of these operations produced a new sequence, and applying them in succession produced a related sequence of fractals. The number of times a particular operation could be repeated was under no restriction. However, it was obvious that performing a sequence of such operations would surely enlarge the DNA sequence. By looking at a given DNA sequence, the fractal it represents cannot be known. Consequently, a recognition algorithm was required, and so a generalized recognition algorithm was proposed. The recognition algorithm read the final DNA sequence and constructed the fractal image depending on the different nucleotides present in the sequence. The strategy used to generate related fractals was similar to the strategy used in this thesis although the variation operators used were quite different involving insertion and deletion as opposed to mutation and crossover.

The Hausdorff distance [13] had been used for fractal image recognition by calculating the distance between sets of points. Sometimes, when dealing with image compression, the aim would be able to recover the class of the original image rather than reproducing the same high-fidelity image. Because fractals are iterative functions, if an appropriate method was found for capturing the fundamentals of the image, it made sense that the amount of information needed to be extracted is fewer than other images. G. LeChiara and L. Saitta [14] used a series of 9 classes of trees along with a 3-layer artificial neural network (NN) performing the back-propagation algorithm. The approach used was reconstructing the image by covering its surface with smaller and smaller copies of itself, through an approximate fractal generation rule. This rule let a point P(x,y) correspond to another point P'(x',y') in the image, by means of an affine transformation. When an image is complex, it is likely that one approximate rule is not sufficient and different parts of the image have to be covered by different fractal rules. The computed set of coefficients constitutes the image features. High recognition rates were achieved using comparatively

few features describing the fractal images. In my thesis, iterative fractals were used to model biomedical structures. Since fractals contained few representative features, a small number of features were needed for the machine learning process.

For calculating the dimension of a 2D (binary representation) fractal, the box-counting method [8] is the most commonly and easily used. This dimension provides a measure of how much of the surface the fractal fills. A. Conci and E. Nunes (2001) [15], extended the box-counting algorithm to grey-scale images by the assumption that the pixel intensity was the third coordinate. Thus the boxes that were used in the 2D fractal images were replaced by cubes. Variation in the intensity of an image represents the texture of the image. Furthermore, to model RGB images, 5 coordinates were used for each point. Because image information can be handled in four color spaces – grayscale, RGB, HLS, and HSB/HSV, for the calculation of fractal dimension, only binary images were used. Many criteria were varied to derive many different binary fractals. Therefore, various fractal dimensions were obtained for the same image. The different bands were finally combined and the fractal dimension was calculated using box-counting. This was similar to the box-counting approach used in my thesis that was applied to RGB and grayscale images.

Fractal dimension is a parameter that measures the geometric complexity of the shape of an object. Shape is obviously not the only characteristic of image texture. Other factors such as the size and distribution of a textural feature and its spatial relations to other features may also be an important part in differentiating one type of texture from another. Basically, the use of fractal dimension alone may not be sufficient in characterization of image textures. Mandelbrot (1982) [16] noted that different fractal sets may share the same fractal dimension but still have remarkably different appearances or textures. The computed fractal dimension values of different image types were found to be overlapping or even the same.

Lacunarity analysis is a technique presented by R. Plotnick et al (1982) [17] to deal with fractal objects having the same dimension but different textural appearances. Lacunarity is linked to the distribution of gap sizes: objects having low lacunarity have similar or

same gap sizes and therefore appear homogeneous, whereas high lacunarity objects are heterogeneous. Lacunarity analysis was originally developed for fractal objects; however, it has also been proposed to be a general method for the analysis of nonfractal and multifractal patterns (Plotnick et al. 1996) [17]. Image segmentation using lacunarity or a combination of fractal dimension and lacunarity yielded good results as showed by Keller et al. (1989) [18] and Dong (2000) [19]. It seemed that incorporating lacunarity measures into the classification procedure was another promising approach to improving classification results.

The methods of computing the fractal dimension must be robust, consistent, and have the ability to distinguish visually different textures, in order for the fractal dimension to be a valuable parameter. There can be considerable variations in the calculated fractal dimension introduced by computational methods, as research to date has shown. Consequently, the choice of method is an important issue in the computation of the fractal dimension for different data. Different methods presented in the literature and the biases that are related with a particular method provide different performance. Blind use of a method without being aware of its applicable fractal dimension ranges and potential errors may lead to poor or even erroneous results.

1.2 Background

This section discusses the scientific background for the different phases in this thesis. The first phase includes the data generation. Then the second phase deals with some preprocessing and feature extraction. Finally, the third phase involves the training and testing using several classifiers. The following paragraphs discuss the background knowledge essential for each phase.

The first phase of this thesis involved the generation of fractal-like biomorphs. A fractal is a "rough or fragmented geometric shape that can be subdivided into parts each of which is (at least approximately) a reduced-size copy of the whole"[17]. Thus, fractals possess the property of being self-similar. Moreover, they have a simple and recursive definition. There are numerous techniques for generating fractals such as using a

recurrence relation at each point in a complex plane, using stochastic processes, or by using a fixed geometric replacement rule. The last of these is called Iterated Functions Systems (IFS) and it is the technique used in this thesis.

IFSs are methods of constructing fractals which always result in self-similar structures. The fractal structure consists of the union of several copies of itself; however, each copy is transformed by a function (the function system). These functions bring the points closer together and make the shapes smaller, composing the shape of an IFS of numerous, possibly overlapping, smaller copies of itself, each of which is also made up of copies of itself. The formal definition of an IFS [20] is shown in Equation 1 where $S \subset \Re^n$ and $f_i : \Re^n \to \Re^n$ are the functions to be iterated. S is the fixed point of a Hutchinson operator, which is the union of the functions f_i.

$$S = \bigcup_{i}^{N} f_i(S)$$

Equation 1

A number of populations of fractal biomorphs were generated for the experiments. These populations followed an evolutionary strategy [21]. The evolutionary strategy uses mechanisms inspired by biological evolution such as reproduction, mutation, and recombination. Reproduction is the biological process by which new individual organisms are produced, mutations are changes in the genetic material of an organism, and recombination is the process where the genetic material is broken and joined to other genetic material and is also referred to as crossover. Both mutation and recombination are the key factors in providing variation during the reproduction process. Figure 1 describes the evolutionary algorithm used in the population generation process.

Evolutionary Strategy:

1. Initialize parent population P.
2. Generate N offsprings where each offspring n is generated by:
    i. Randomly select 2 parents from P.
    ii. Recombine the selected parents to form a recombinant r.
    iii. Mutate the recombinant r.
3. Select new parent population from N and P.
4. Go to Step 2 until termination criterion fulfilled.

Figure 1 Evolutionary Algorithm

Once the fractal biomorphs were generated, the necessary features needed to be extracted from the images. Feature extraction was done using one of the numerous fractal analysis techniques [22]. It is necessary to reduce the dimensionality of the extracted feature to avoid problems such as the curse of dimensionality [5]. One of the most popular fractal analysis techniques is the well-known box counting algorithm [23]. This algorithm measures the fractal dimension which is a statistical quantity that gives an indication of how completely a fractal appears to fill space, as one zooms down to finer and finer scales. The box-counting algorithm works by picturing the fractal is lying on an evenly-spaced grid, and counting how many boxes are necessary to cover the set. Then the box-counting dimension is calculated by seeing how this number changes as the grid is made finer. A log-log plot of all the points of the number of boxes covering the fractal versus the box size is then drawn. Finally, a line of best fit is drawn through these points and the slope of that line is computed as the fractal dimension. Each point on the log-log plot, where N(s) is the number of grid boxes that cover the fractal structure and s is the size of the box size, is described in Equation 2.

$$D = \frac{\log(N(s))}{\log(1/s)}$$

Equation 2

Choosing the box counting algorithm as a method for feature extraction is well-justified. First of all, the algorithm is simple and very easily implemented. Secondly, it provides a significant reduction in dimensionality which is essential to the efficiency of any classifier. The complexity of the box counting algorithm is $O(n \log \sqrt{n})$, where n is the total number of pixels. The complexity of the box counting algorithm makes up for the considerable reduction of data space. Not only should the feature vector size be minimized enough to where each element only contains genuine information on the dataset, but complete loss of some crucial information has to be avoided.

Once the features of the fractal biomorphs are extracted, the last phase is the machine learning phase. There are various types of classifiers and machine learning algorithms used in different classification problems. When looking at different learning algorithms or classifier models, there is no reason to prefer one over the other for a given application. This concept based on having no prior information about the problem, is described as the No Free Lunch Theorem [5]. Also, the Ugly Duckling Theorem[5] states that the number of predicates shared by any two different patterns is constant and does not depend on the choice of the two objects, given that a finite set of feature values are used to tell apart the patterns under consideration. In other words, we need some kind of bias to prefer certain categories over others. These two theorems emphasize the necessity for proper features and matching the algorithm to the data distribution, since there is no "best" learning algorithm or feature representation that is independent of the classification problem.

There are two types of models that could be used for pattern recognition – parametric, and non-parametric models [5]. An example of a parametric classifier could be for instance a Gaussian Mixture Model, where maximum likelihood would be used to estimate the model parameters. In some applications, it is difficult to find the best model, particularly when the feature dimension is high. Non-parametric approaches don't assume any model; rather, they let the available data produce its own distribution over the feature space. There are two main techniques, the parzen-window classifier [24], and the nearest-neighbor classifier [5]. In the parzen-window classifier, the probability density function of each class is approximated by applying a symmetric window over each training data

point in the feature space. A spherical Gaussian is usually used. The parzen approach is also known as the probabilistic neural network. The main disadvantages of the probabilistic neural network are that it suffers from excessive storage and high computational costs.

The second type of non-parametric models is the nearest neighbor classifier. In this type of classifier, the training data for each class is stored as centroids or prototypes. A test pattern is assigned to the class that contains the nearest prototype to that test pattern. This is what is referred to as the nearest neighbor rule. Any type of distance may be used, but commonly, Euclidean distance is the default. An extension of the nearest neighbor classifier is the k-nearest neighbor classifier (kNN). kNN works by finding the nearest k prototypes to the test pattern. The next step is finding the class identity that occurs the most in k identities. An edited version of the nearest neighbor classifier is the condensed nearest neighbor classifier [25]. This model suggests that not all patterns are stored. Only those prototypes which actually form the decision boundaries would mainly be needed, not those deep within the class probability density function. Systematically removing the deeper prototypes produces a more compact classifier reducing the computational costs and storage requirements. The algorithm is described in Figure 2.

1) Start by removing one prototype from any class. Use it as a test pattern. If it is classified correctly then remove it. If not, put it back.
2) Repeat over all prototypes for all classes.
3) Go back to Step 1. Stop when no more prototypes can be removed.

Figure 2 Edited kNN

If the remaining prototypes are kept in their original location, then the system is non-parametric. However, if the prototypes are adapted, then the system is considered semi-parametric. Hence, some criterion was used to enhance the classifier performance by

adapting the locations of the prototypes (their means). This approach is the one used in Linear Vector Quantization (LVQ) Neural Networks [5] and many other systems. LVQ is a technique that trains competitive layers in a supervised manner. In supervised learning the classifier is trained to predict the class label of an unseen input based on knowledge acquired previously from pairs of input and targets. The competitive layer automatically learns to classify input vectors. Yet, the determined classes are dependent only on the distance between the input vectors. So if two vectors are too similar, the competitive layer will assign them to the same class. LVQ networks learn to classify inputs into targets specified by the user. The adaptation in LVQ is based on a discriminative training (as opposed to a maximum likelihood) criterion.

There are many criteria used for discriminative training which are different in concept from maximum likelihood. In maximum likelihood, the probability density function of each class is approximated as closely as possible (both at the boundaries and at the depth of the class). However, in discriminative training, the focus is on minimizing the probability of error in classification and that involves all classes simultaneously.

As for LVQ neural network models, the initial state may be an edited nearest neighbor classifier or just a k-means [26] with k centroids for each class. The LVQ adapts the locations of the centroids to obtain better classification performance. LVQ uses the nearest neighbor approach (competitive network). In other words, for any input pattern, the centroids of each class compete and only one winner is used to represent the class.

The most popular type of Neural Network (NN) classifiers [27] is the multi-layered NN. It is constituted of several linear perceptrons or neurons. Because a single layer of perceptrons poses the limitation of only being able to classify linearly separable data, multi-layer networks provide an optimal solution for classification of data that isn't linearly separable. However, at base, multi-layer networks do implement linear discriminants, but in a space where the inputs have been mapped non-linearly. Although such networks use rather simple algorithms, they are widely applicable to real-world problems. The most common method of training these networks is based on gradient descent in error – the backpropagation algorithm. The backpropagation algorithm is an

extension of the Least Mean Square algorithm. It is a supervised learning method that can calculate the desired output for any given input.

To solve non-linear problems, multilayers of neurons are used. The role of the first hidden layer in a NN is to split the problem space into regions (convex hulls). If a second hidden layer exists in the NN, it combines regions that were split by the first hidden layer creating arbitrary decision boundaries. Having 2 hidden layers of neurons can solve any nonlinear problem. However, caution needs to be taken when deciding the number of neurons to include in each layer and that can be practically determined by validation. Moreover, the number of inputs depends on the feature length, and the number of outputs depends on the number of classes.

Another type of classifiers is the radial basis function networks or RBF networks [28]. These classifiers aim to solve non-linear problems by projecting the feature space to a higher dimension. In that higher dimension space, the classification is transformed into a linear problem. At first, polynomials of higher degrees were used to solve the non-linear problems linearly using a linear perceptron for instance. Later, mixtures of local experts were defined to cover a certain region in the input space and produce a high output only for that region. The outputs would then be linearly combined. Gaussians, for example, could be used to produce a high output when a test set happens to lie in its region of influence. The weights would be changed to optimize the performance of the classifier.

A type of RBF networks is Support Vector Machines (SVMs)[29], where the input is mapped by a nonlinear function to a high dimensional space. Basically, the data is preprocessed to represent patterns in a dimension higher than the original feature space. The original data is transformed to that new space using a kernel function. SVM features and kernel methods are unique in the fact that they are not affected by local minima and they do not suffer from the curse of dimensionality [30]. The generalization problem makes it easy to overfit in high dimensional spaces since regularities can be found in the training set that would not be found again in the testing set.

A nonlinear map to high dimension is used to separate data from two categories by a hyperplane. SVMs work by maximizing the margin around the separating hyperplane. The decision function is fully specified by a subset of training samples, the support vectors, which are the data points that lie closest to the decision surface. These vectors are the most difficult to classify because they have to represent the whole class. Of course, there may be many possible hyperplanes providing good solutions to any classification problem. SVMs find the most optimal solution maximizing the margin. There are a number of specialized algorithms for solving the problem that arises from SVMs, mostly depending on breaking the problem down into smaller parts. One common method is Sequential Minimal Optimization [31] (SMO).

The SMO algorithm breaks a large quadratic programming optimization problem down into 2 dimensional sub-problems that may be solved analytically. This eliminates the need for a time consuming numerical optimization algorithm. The amount of memory needed for SMO is linear in the training set size, allowing SMO to deal with large training sets. Compared to the standard chunking SVM algorithm, which scales between linear and cubic, SMO scales between linear and quadratic in the training set size for different test problems. This is due to the fact that matrix computation is avoided.

C h a p t e r 2

METHODS

2.0 Data Preparation

The creation of a single fractal biomorph image was done using Fractal Time [32], a Visual C++ program by Daniel D. Gonzalez. The generated fractal biomorphs were described as being IFS and had a number of parameters that could be modified. Such parameters include the visualization range, the number of iterations of the algorithm which indicates the number of times the algorithm applies the Hutchinson's operator to the function system, the number of functions the system has, and the function system matrices and their corresponding probabilities.

The biomorph genome was composed of a vector of 35 real numbers encoding the fractal parameters. Each dataset used in the experiments performed for this thesis consisted of 2 populations of fractal biomorphs. Using selection, the genomes of the biomorphs were driven to 2 certain "interesting" points over several generations. A large number of subjects (1000 subjects) were then generated for each population. Each population followed an evolutionary strategy. The evolutionary algorithm used Gaussian mutation and crossover as operators bringing about variation within the population. Mutation was performed by adding a normally distributed random value to the vector components. The standard deviation of the normal distribution represented the mutation strength. The program implementing the evolutionary algorithm was coded using Matlab 7.0 [33].

A total of 11 datasets, each including 2 populations, were generated having increasing levels of evolutionary variance. The datasets were generated to contain increasing mutation rates of 1,5,10…50%. For subsequent experiments, 5 different versions of the original 11 datasets were generated to contain noise and blur modeling the actual imaging process. The blurring was first done using a Gaussian filter. The kernel sizes for the progressively increasing levels of blur were 2x2, 4x4, 6x6, 8x8, and 10x10. Following the

blurring, progressively increasing levels of Gaussian noise, whose means were 0, were added to the images. These increasing levels of noise had increasing variances of 0.02, 0.04, 0.06, 0.08, and 0.1 in normalized units.

Figures 3-6 show some samples from each population for some of the generated datasets. The first, fourth, eighth, and eleventh datasets had mutation rates of 1%, 15%, 35%, and 50% respectively. Figure 7 shows how a given sample is affected by adding 5 different levels of distortion. Distortion includes blurring followed by noise addition.

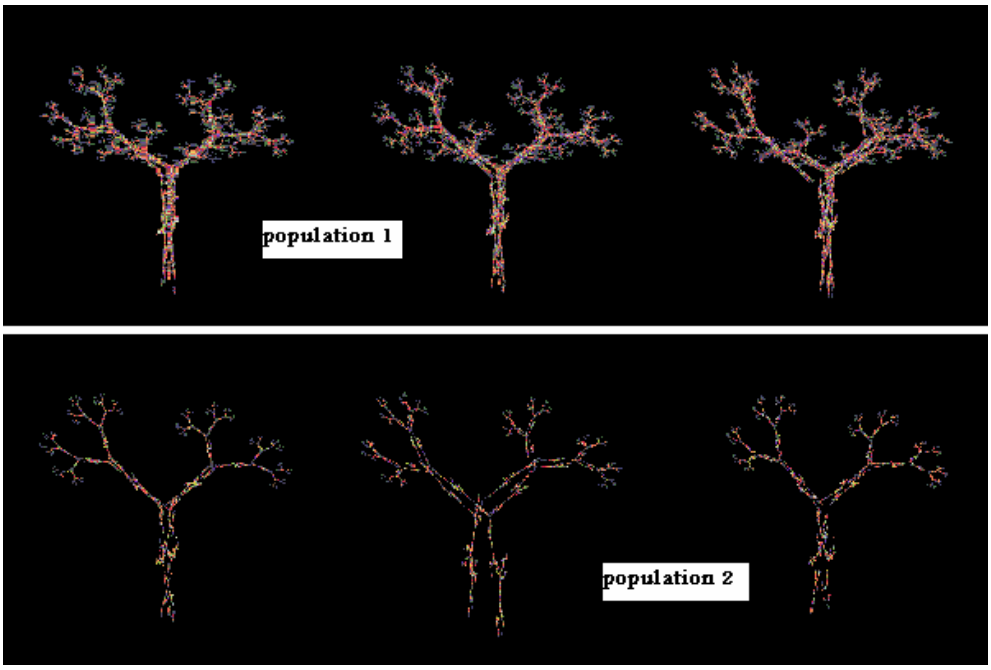

Figure 3 Sample of Dataset # 1 with 1% mutation rate

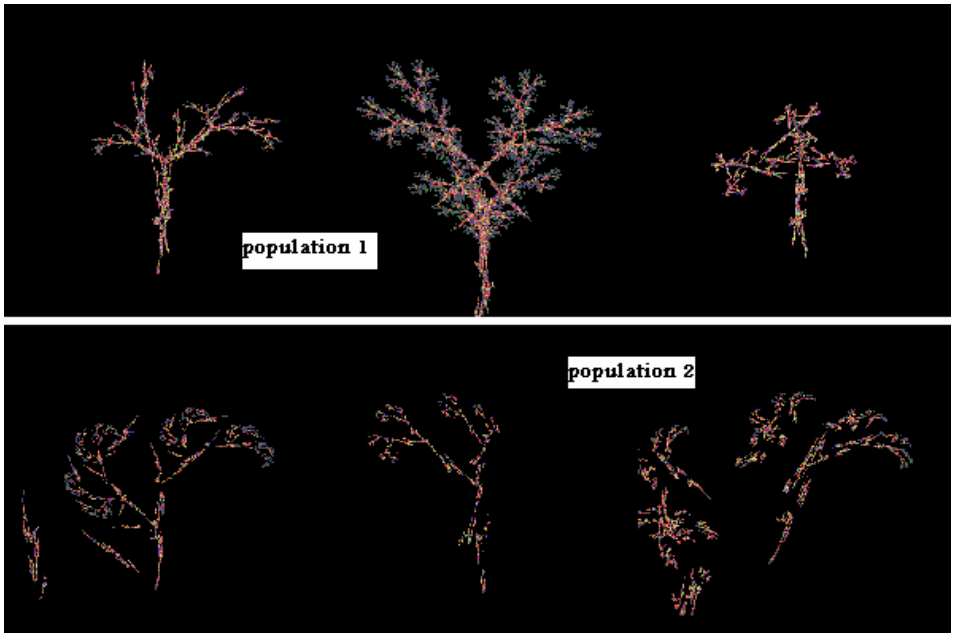

Figure 4 Sample of Dataset # 4 with 15% mutation rate

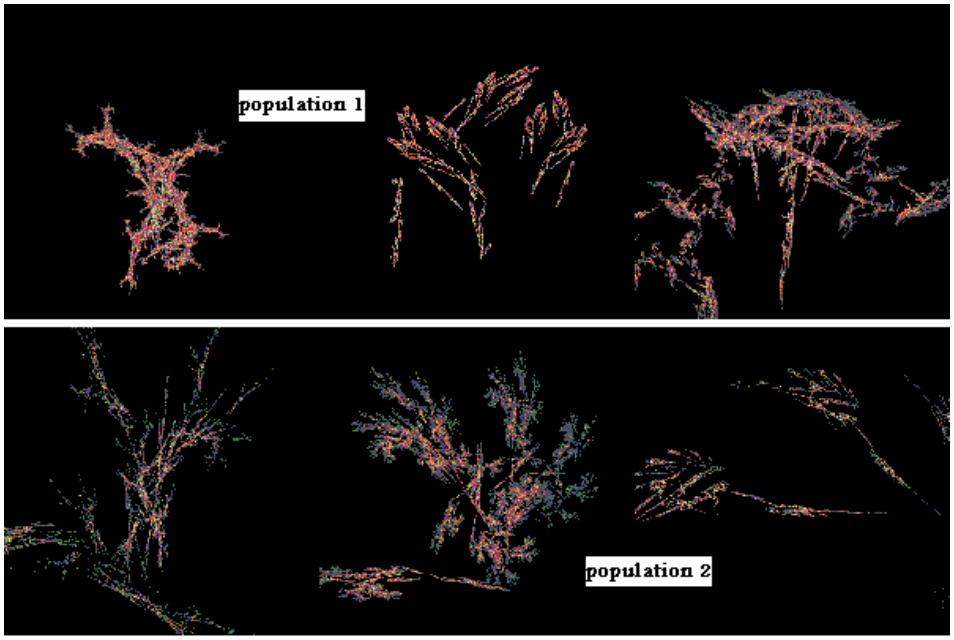

Figure 5 Sample of dataset # 8 with 35% mutation rate

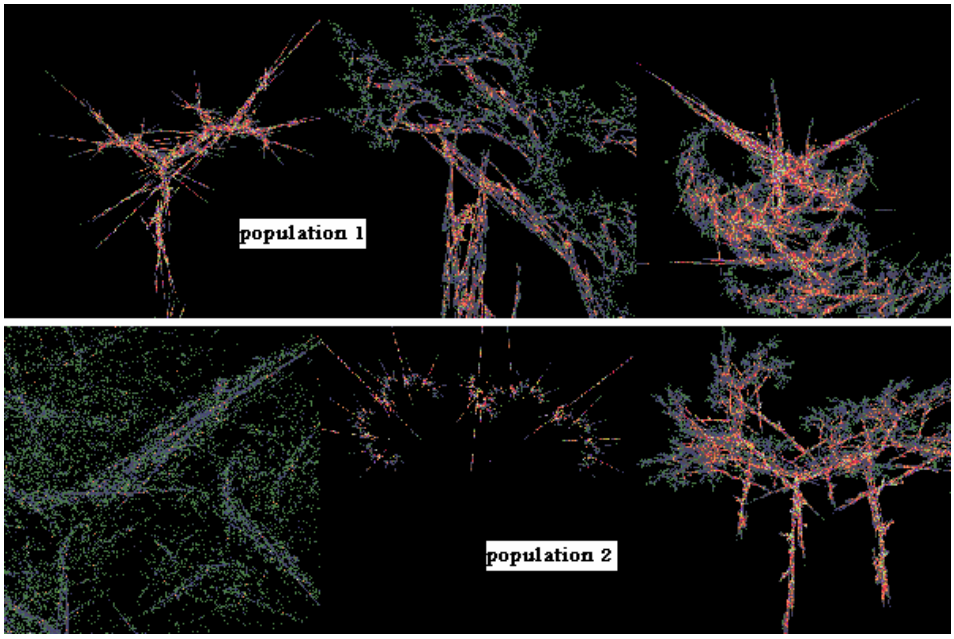

Figure 6 Dataset # 11 with 50% mutation rate

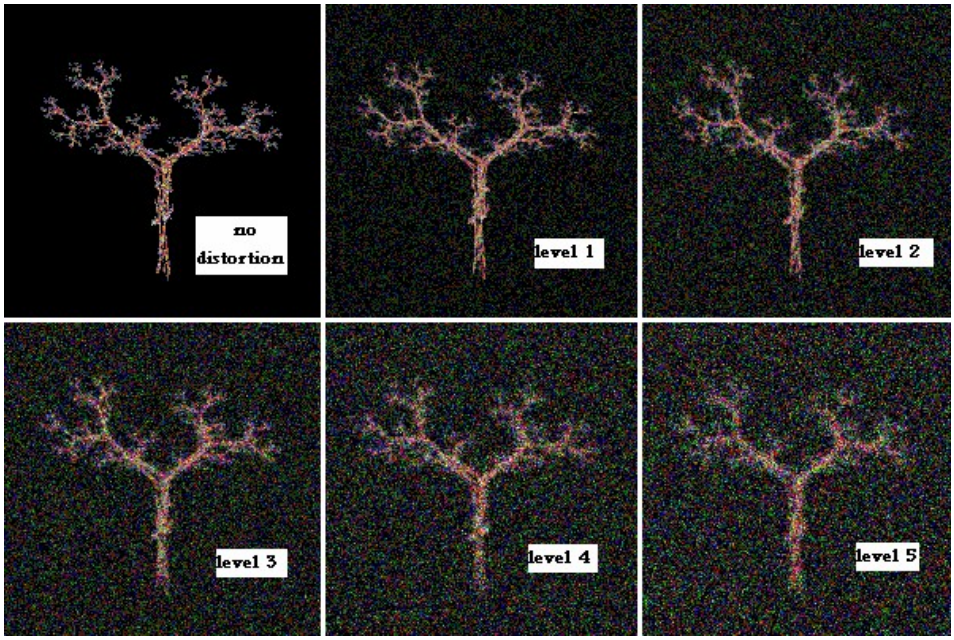

Figure 7 Effect of adding noise and blur on images

2.1 Preprocessing and Feature Extraction

In order to prepare the images for feature extraction, some simple preprocessing was first performed on the images, also using Matlab. The original images were in RGB format of size 512x512 pixels. These images needed to be converted to a binary format. This was done by first converting the pixels of the images to grayscale where each pixel was given an intensity value ranging between 0 and 255. By then taking a threshold, the grayscale intensities were further converted into binary values. A 0 value was assigned for each black pixel while a 1 value was assigned for each white pixel. Figure 8 shows a sample image passing through the preprocessing stage.

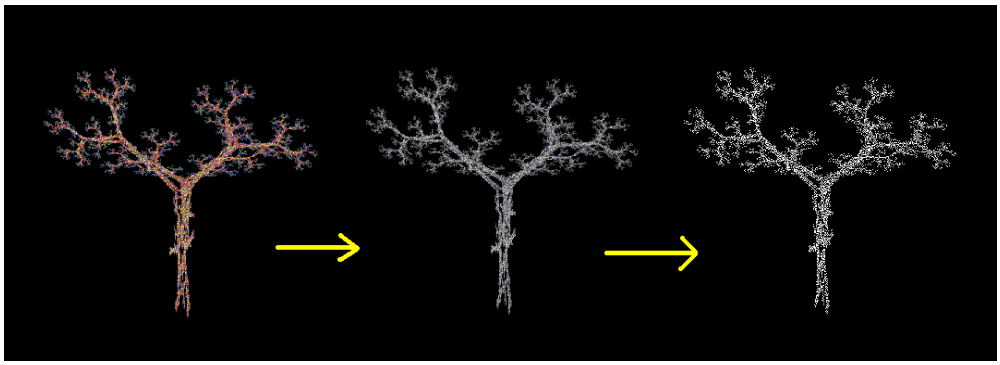

Figure 8 Preprocessing Stage: Conversion from RGB → Grayscale → Binary

Once the datasets were prepared, feature extraction was performed using a fractal analysis technique. The box counting algorithm was used to extract 10 features from each image. The box sizes used to form the grids were powers of two (1,2,4,…512). Once the counts were available for all images and for all box sizes, the features had to be normalized with respect to one another. All counts were normalized so that all values lied between 0 and 1. This algorithm was also implemented using Matlab. Figure 9 shows a sample feature vector extracted from one of the images. However, these values are before the normalization process.

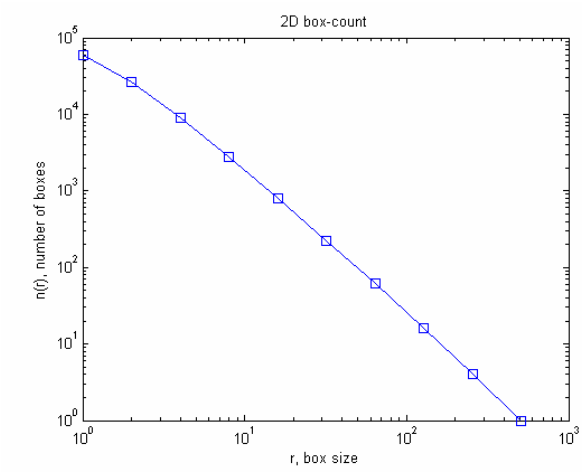

Figure 9 Sample Feature Vector for an Organism

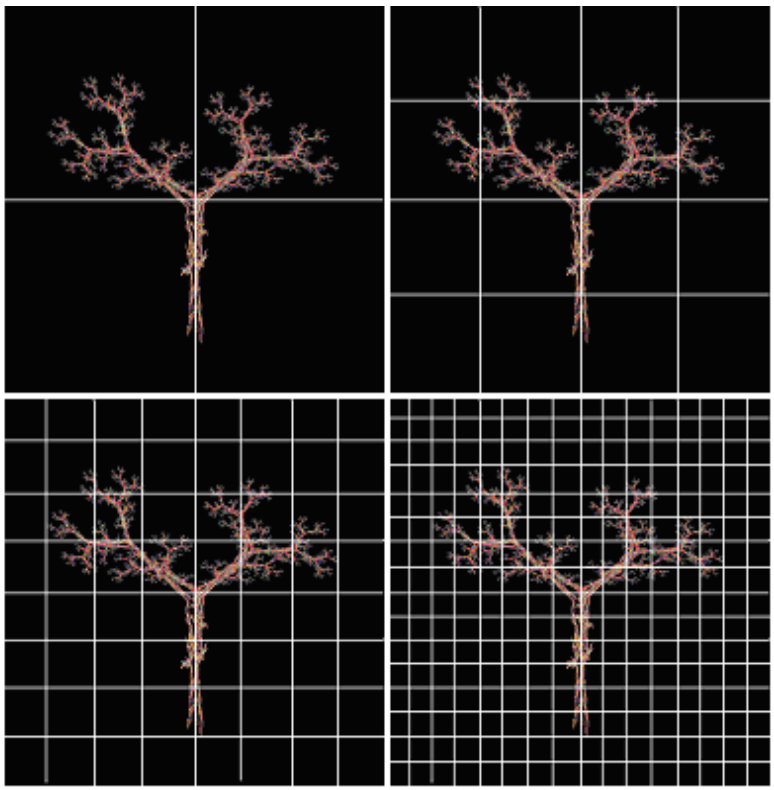

Figure 10 Visualization of the Box-Counting Algorithm

Figure 10 shows some samples of how the box size is varied. The 10 different chosen box sizes vary by powers of 2, which is the standard statistically. However, more box sizes could have been taken into account, but that would have increased the feature vector length depending on how many more box sizes were used. This would have an effect on the classifier performance depending on whether or not true (not redundant) information was added. Also, depending on which box sizes were chosen to add between the original 10, the performance of the classification would also be affected.

When comparing the feature vectors of the 2 classes of each dataset, the features seemed to overlap a great deal. In other words, in order to differentiate between the 2 classes, a non-linear decision boundary had to be drawn. A visualization of the feature space for one of the data sets is shown in Figure 11. Clearly, the features from each class are not linearly separable, which had to be taken into account when designing the classifiers to solve this classification problem.

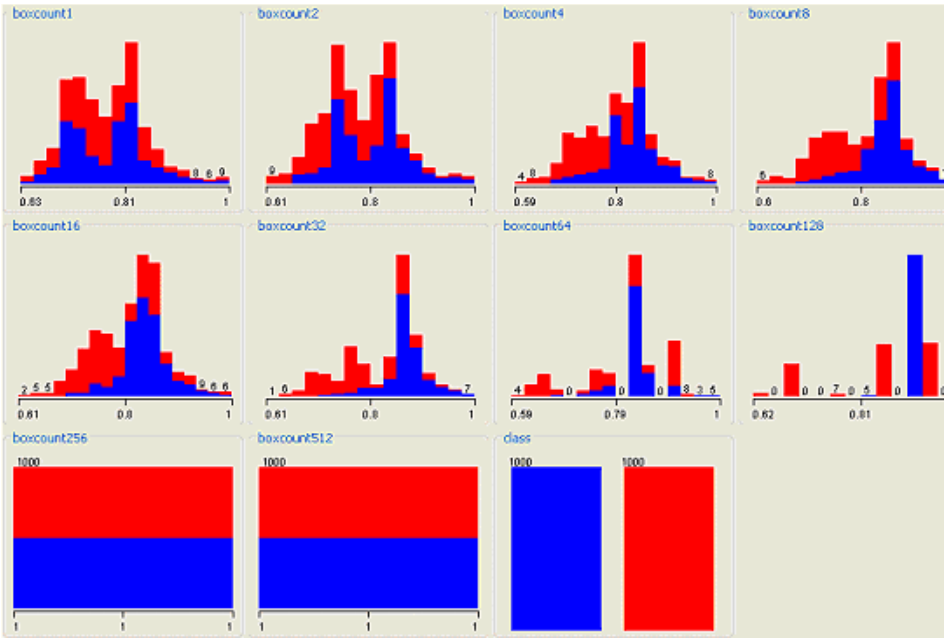

Figure 11 Visualization of Feature Distribution for the 2 classes

All box counts were normalized with respect to one another before the training process, and that is why the maximum count did not exceed 1 in this visualization. The means and standard deviations of each dataset varied depending on the mutation level and the distortion amount applied to the image.

2.2 Machine Learning

Once the essential features in the datasets were extracted, Weka[34], a machine learning package, was used to apply different machine learning techniques. Several classifiers and their structures were specified, namely, NNs, SVMs, and kNN. Moreover, a voting scheme combined the outputs from the preceding classifiers to come up with a single decision that included an un-weighted average of the probability estimates of those base classifiers.

A Multilayer Perceptron NN used the back-propagation algorithm to train. The NN structure shown in Figure 12 was composed of a size 10 input layer corresponding to the length of the feature vector, 2 hidden layers consisting of {6, 4} perceptrons, and an output layer consisting of 2 neurons corresponding to the number of classes in the datasets. The number of neurons in the hidden layers was settled on after a series of trial and error and then selecting the structure that yielded the best performance.

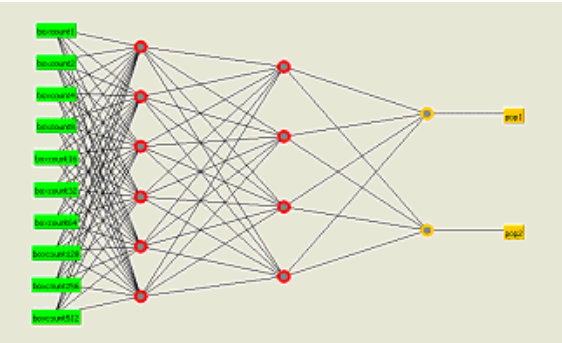

Figure 12 Neural Network Structure

All the nodes in the NN are sigmoid. The inputs are provided from the green labels on the left. The red nodes are the hidden layer nodes. The orange nodes are the output nodes. The labels on the right show the class the node represents

The SVM used was built using Weka. It implemented John Platt's SMO algorithm for training the support vector classifier. A polynomial kernel whose cache size was set to full was used to map the attributes to a higher dimensional space. The polynomial kernel was K(x,y) = <x,y>^p or K(x,y)=(<x,y>+1)^p. The exponent for the polynomial kernel was specified as 3. These were the optimal values obtained after a series of trial and error. Figure 13 shows how a simple example of how support vector would separate 2 classes using 2 support vectors after the problem has been transformed to linear problem. A problem may have many support vectors maximizing the margin between the hyperplane forming the separation as in the case of the experiments in this thesis.

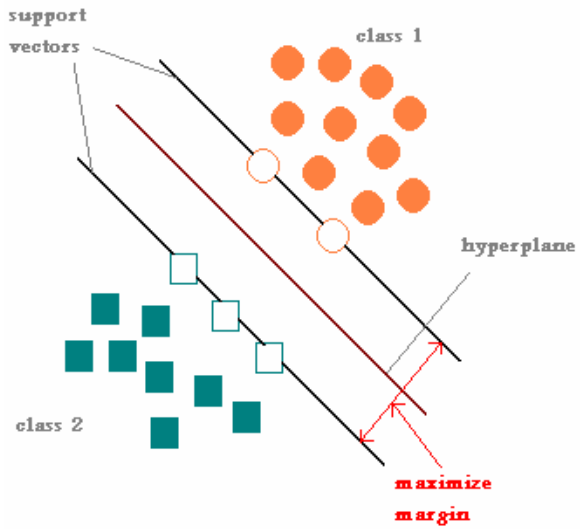

Figure 13 Illustration of SVM learning method

One of the most famous classifiers in machine learning is the kNN. Nearest neighbor classifiers are considered lazy learning memory-based methods. Although they are simple, they are very powerful non-parametric classifiers. They work by taking an input

pattern and computing the k closest prototypes to it using some distance metric that the user defines. The classifier then does a majority vote among the k nearest prototypes to decide which label will be assigned to the input. If k=1 the classifier simply the label of the nearest prototype to the input pattern. The parameters of a kNN to be estimated during the learning stage are usually the prototypes but can sometimes be the distances. The prototypes are usually computed by storing the whole training set as the set of prototypes. The drawback to that would be high storage and computational requirements which lead to the search for simpler solutions.

As for the kNN classifier used in the experiments, the best value of k was selected using hold-one-out cross-validation. There was no limit to the number of training instances allowed in the training pool, meaning when new instances were added the old instances were not removed. There are several nearest neighbor search algorithms. The one used in the experiments was the linear nearest neighbor search algorithm. It performs a brute force search for the nearest neighbor. The distance function used for finding neighbors was the Euclidean distance [35]. The Euclidean distance between point P= (p_1, p_2,…p_n) and Q=(q_1, q_2,…q_n) is defined as:

$$\sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + ... + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$$

Equation 3

Chapter 3

RESULTS

3.0 Experiment 1

In the first experiment, the data generated included 11 populations, each consisting of 2 classes. Each successive population had an increasing amount of evolutionary variation within the population pair. The variation was brought about by applying a nonlinear mutation followed by a crossover. Each class consisted of 1000 images. Feature extraction was done using the box counting dimension which was calculated for varying box sizes of 1,2,4,8…512. This was performed for all 22,000 images. The dataset of each class was split into 66% for training and 33% for testing. Training was done using NN (using the back-propagation algorithm), a SVM (using the SMO algorithm), and a kNN (using cross-validation to select appropriate values for k and calculating the Euclidean distance using a linear brute force search algorithm.) Figure 14 shows a comparison of performance among the 3 different classifiers.

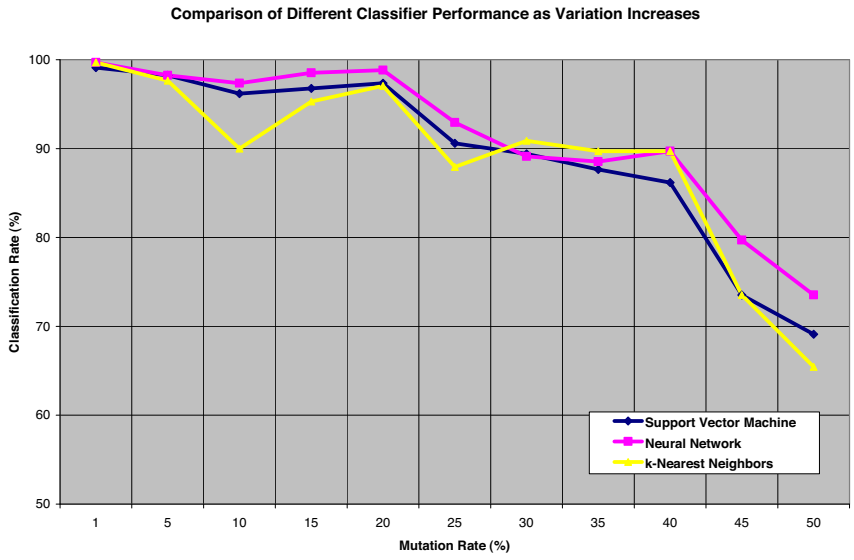

Figure 14 Comparison of Classifiers Performance vs. Variation

The following graphs show the Receiver Operating Characteristic (ROC) [36] [37] curves for the 3 classifiers. The ROC curve is a graphical plot of the sensitivity versus the specificity for a binary classifier as its discrimination is varied. It basically plots the true positives against the false positives as the criterion changes. For a two-class (binary) classification there are 4 possible outcomes from a binary classifier. If the outcome of a prediction is the actual value then it is called a true positive. However, if the actual value is different from the prediction, it is said to be a false positive. By drawing the false positives and true positives on the x and y axes respectively, relative trade-offs can be illustrated between true positives (sensitivity) and the false positives (specificity). Each point on a ROC curve represents an instance of a confusion matrix in a classification problem. A 45° line passing through the origin is the line of no discrimination, where a random guess would be just as good as the classification process. A point lying along this diagonal line would indicate that the classifier is useless. On the other hand, the best possible prediction method would yield a point in the upper left corner of the ROC space. Mainly, the further the points lie away from the so-called line of no-discrimination, the better the prediction method and classifier performance. Figures 15, 16, 17, and 18 show the ROC curves for mutation levels 1, 15, 30, and 50% respectively.

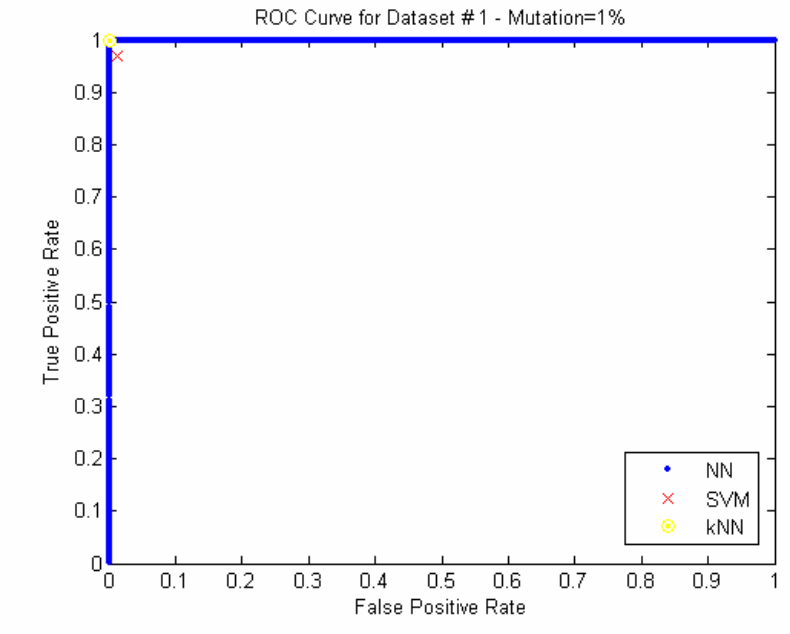

Figure 15 ROC curve for first dataset

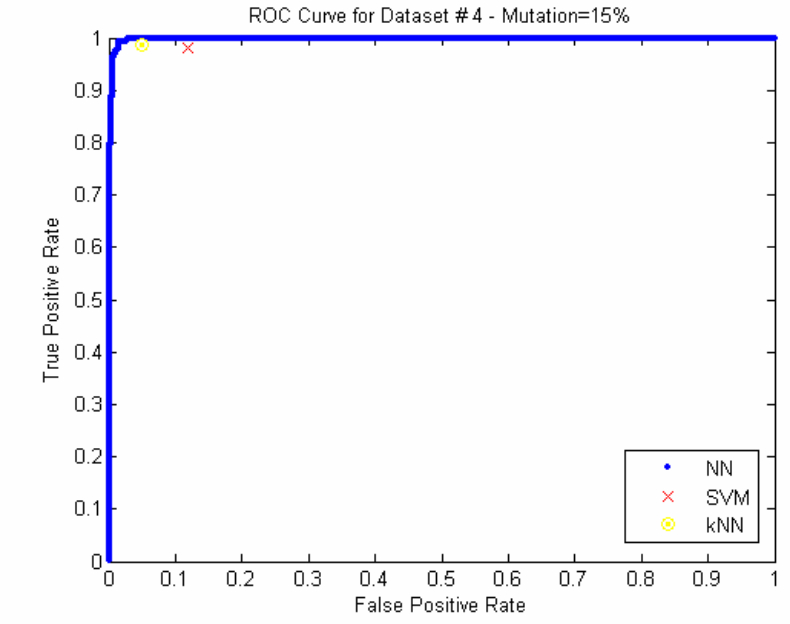

Figure 16 ROC curve for fourth dataset

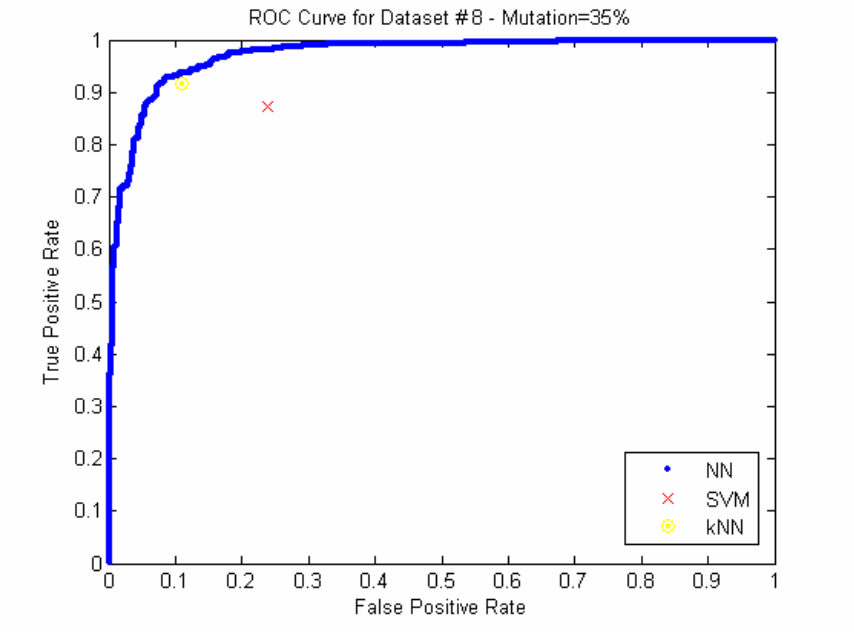

Figure 17 ROC curve for eighth dataset

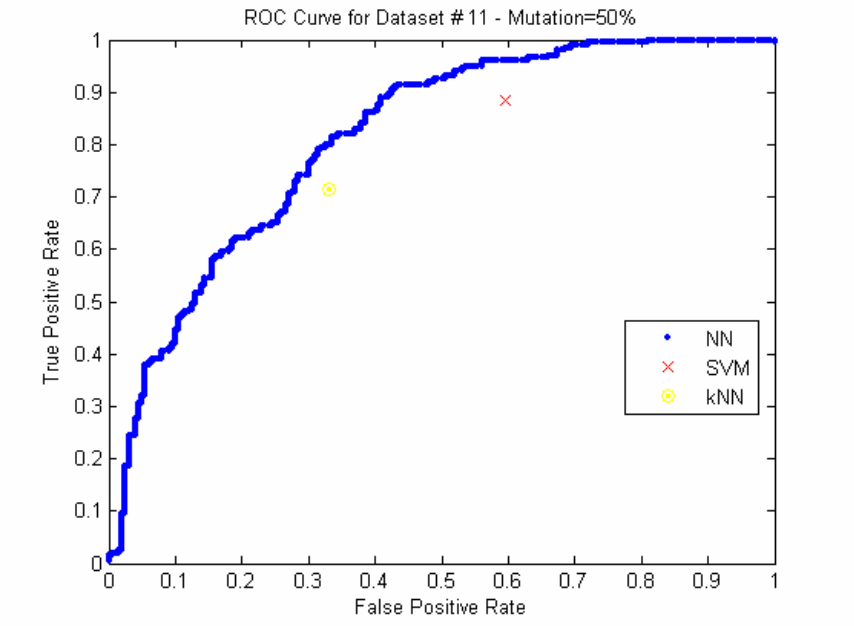

Figure 18 ROC curve for eleventh dataset

3.1 Experiment 2

Experiment 2 is a repetition of Experiment 1; however, outputs from all three classifiers were combined to come up with a single decision (voting). This combination involved using an un-weighted average of the probability estimates of each base classifier (SVM, NN, kNN). Figure 19 shows the effect of combining the 3 classifiers' decisions to come up with a cooperative decision.

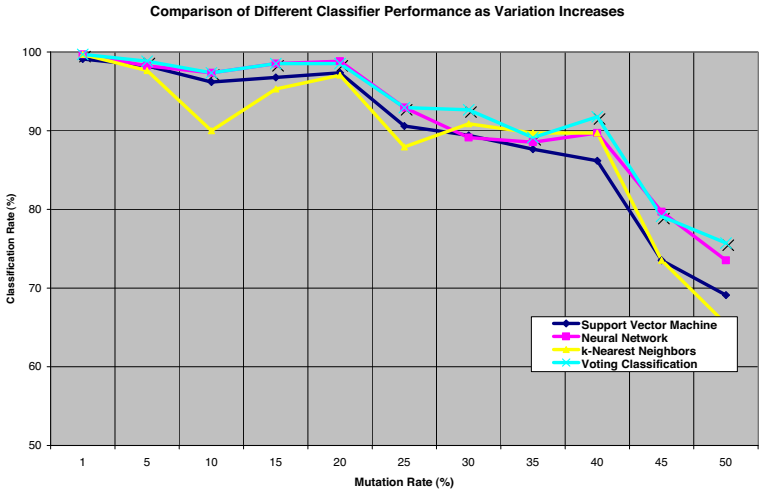

Figure 19 Comparing Voting with individual classifiers

3.2 Experiment 3

The same 11 populations used in Experiment 1, each consisting of 2 classes, were used again. This time 5 different levels of noise and blur were added to the 22,000 images. Following that, the box counting dimension was calculated for varying box sizes of 1,2,4,8…512. This was performed for all 5 noisy and blurred versions the 22,000 images. Again, the dataset of each class was split into 66% for training and 33% for testing. Training was done using a NN, a SVM, and a kNN. Using the populations containing differing levels of noise and blur, created in this experiment, testing was performed. This entire experiment was repeated 10 times, from the simulation up, and the box and whisker

plots [38] were drawn for each type of classifier from 2 perspectives – once with respect to the mutation rate, and once with respect to the distortion level.

The box and whisker plot is a convenient way of graphically depicting groups of numerical data through their five-number summaries (the smallest observation, the lower quartile Q1, the upper quartile Q3 and the largest observation). It also shows if there happen to be any outliers. The reason box plots were drawn was to portray the differences between populations without making any assumptions of the underlying statistical distribution. These plots indicate information regarding the degree of dispersion and skewness in the data.

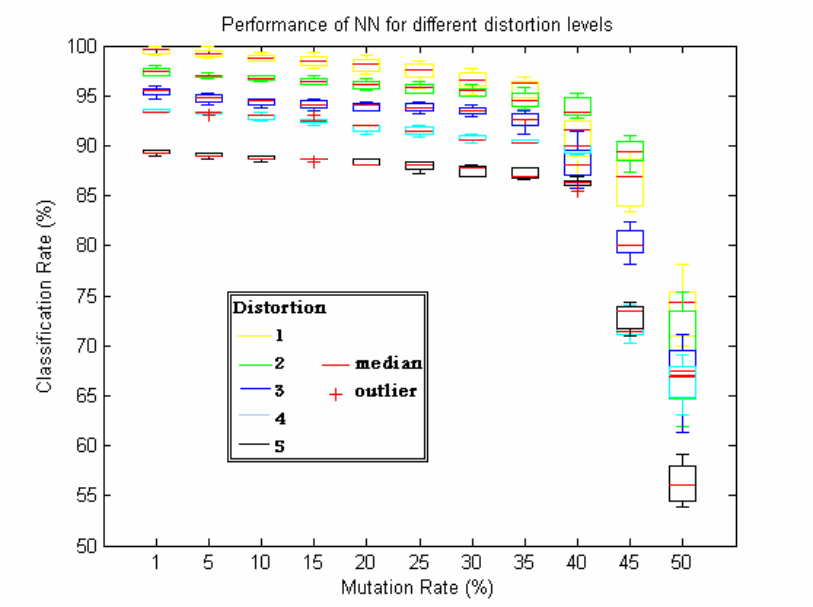

Figure 20 Performance of NN for different distortions

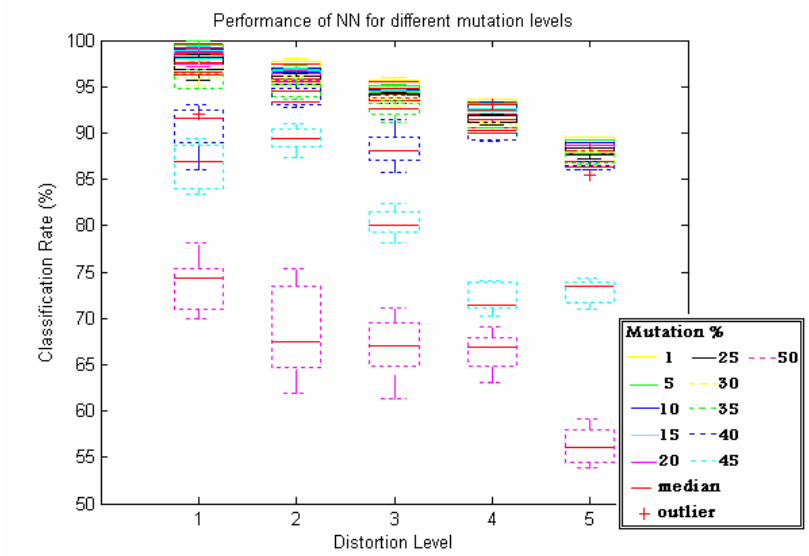

Figure 21 Performance of NN for different mutations

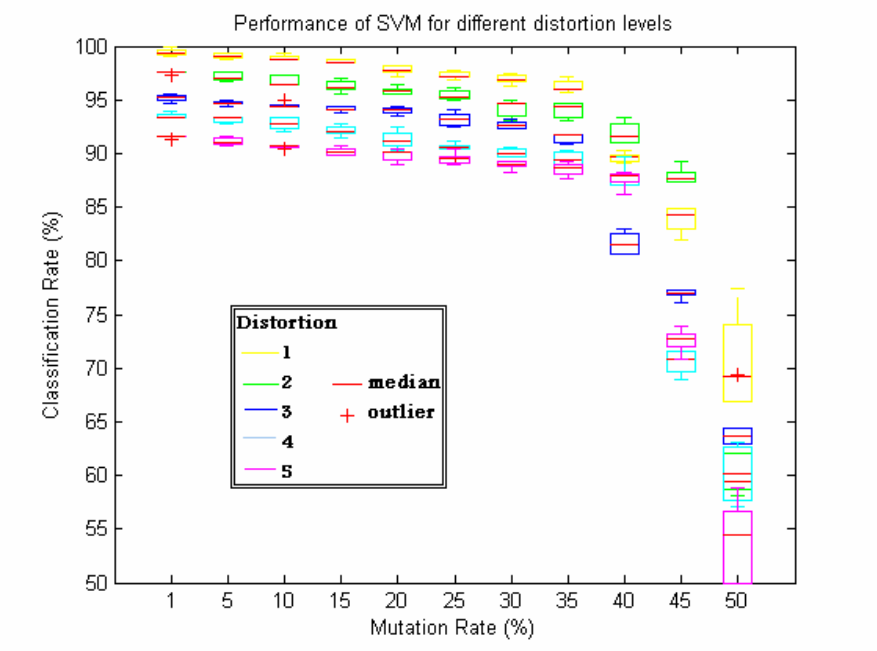

Figure 22 Performance of SVM for different distortions

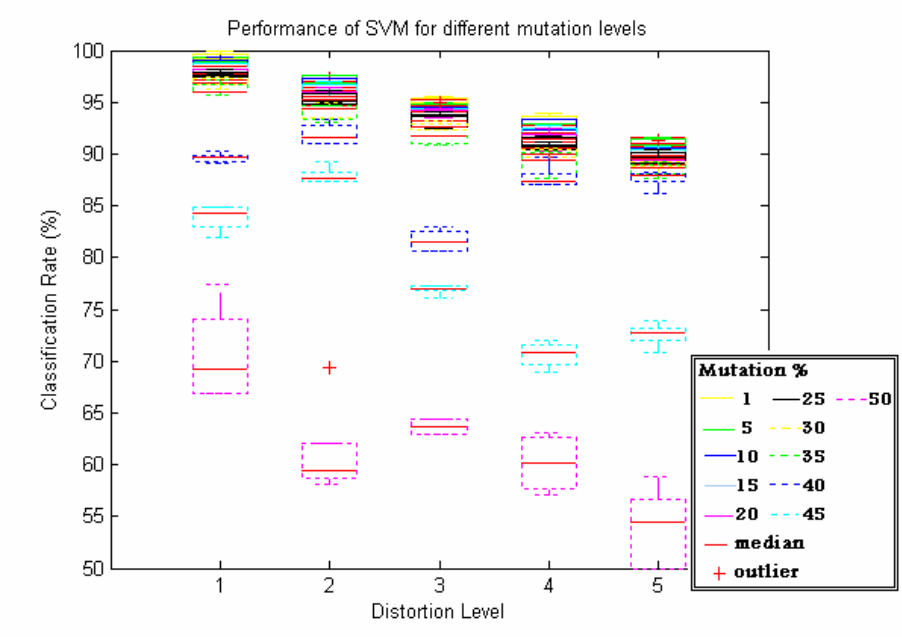

Figure 23 Performance of SVM for different mutations

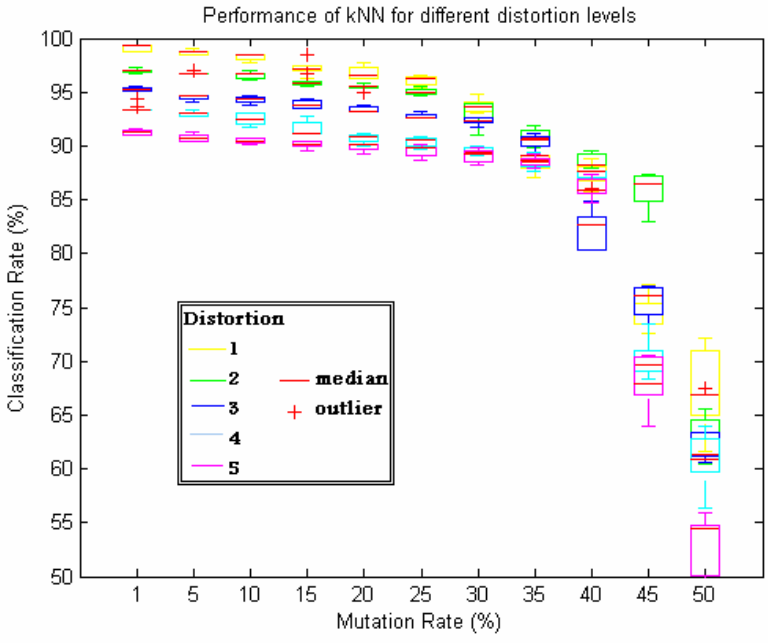

Figure 24 Performance of kNN for different distortions

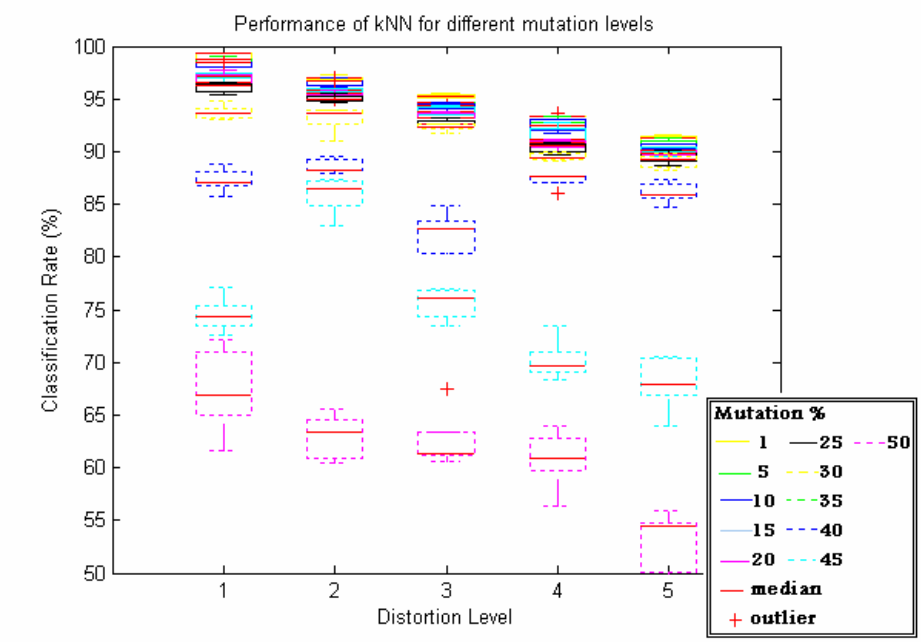

Figure 25 Performance of kNN for different mutations

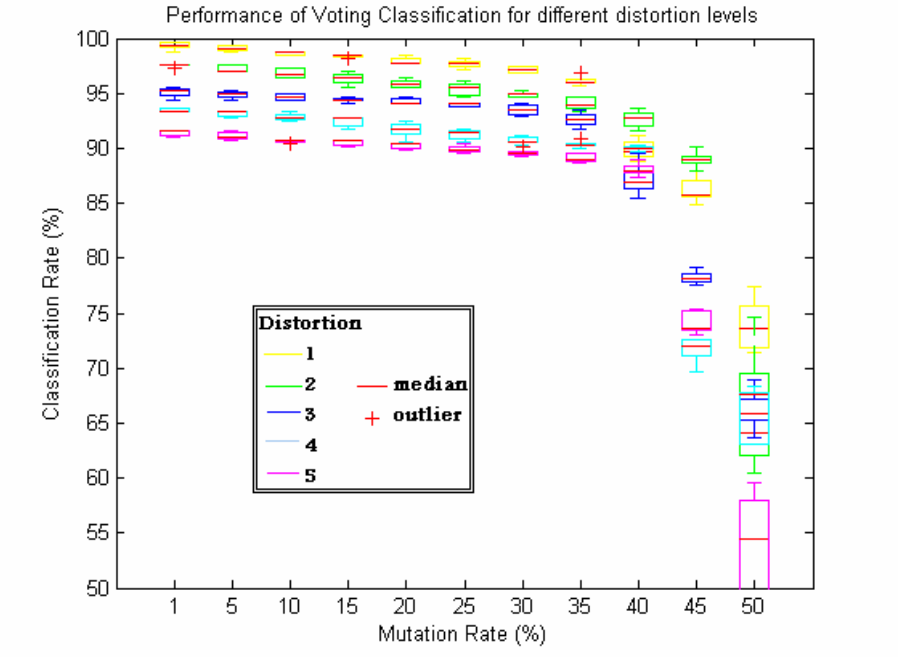

Figure 26 Performance of Voting for different distortions

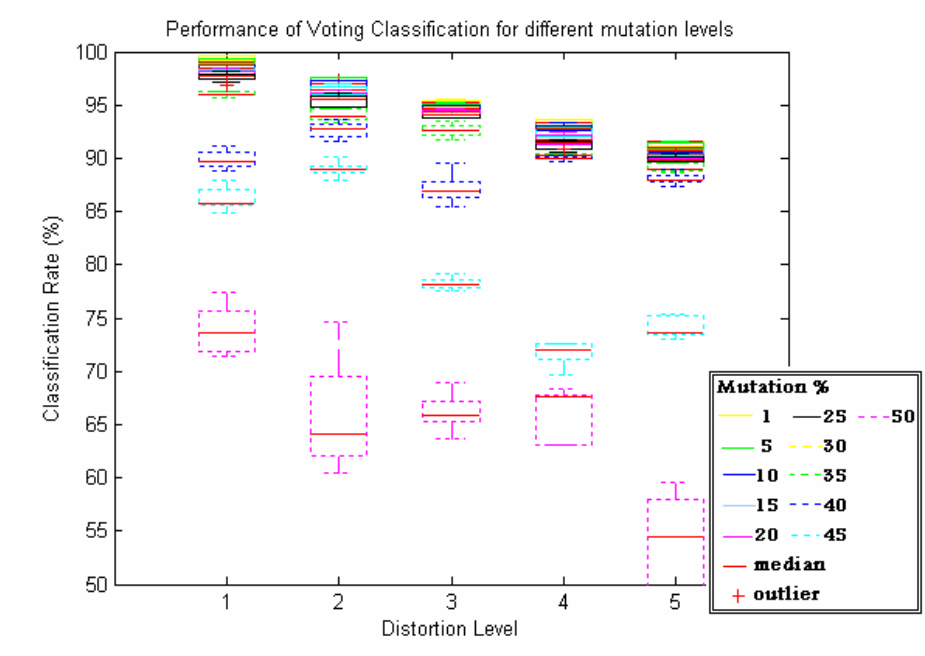

Figure 27 Performance of Voting for different mutations

Chapter 4

DISCUSSION AND CONCLUSIONS

4.0 Discussion

This section discusses, interprets, and provides justification for the observations from each experiment shown in the previous chapter. Each one of the four experiments' observations will be analyzed in turn.

In Experiment 1, NNs, SVMs, and kNN classifiers were used to classify 2 classes from 11 datasets. Figure 14 shows a general trend of the classification rate falling as the mutation rate increases. The reasoning behind this is that when the mutation rate increases, the amount of variation within a given class also increases. An increase in mutation rate increases the variance in the classes, thus making the classification problem harder leading to poorer recognition rates. The figure also shows that NNs, in general, performed slightly better than its counterparts. SVMs seem to rank second in terms of performance for mutation rates between 1% and 25% while kNN classifiers seem to rank second for mutation rates between 30% and 50%.

Also in Experiment 1, Figures 15-18 of ROC curves were drawn for the 3 classifiers for a number of mutation rates. The first figure, having a mutation rate of 1% showed an almost perfect classifier for NNs and slightly lower (although still very high) performance for SVMs and kNN classifiers. As the mutation rate was increased from 1% to 15%, then to 30%, and finally to 50%, the ROC curves for NNs showed that the curves get closer to the line of no-discrimination, indicating degradation in the classifier's performance. The same applies to the SVMs and kNN. The points on their ROC curves got closer and closer to the line of no-discrimination as the mutation rate was increased. This makes perfect sense since it is justified that as the mutation rate increases, so do the intra- class differences, making the classification problem more complex. Moreover, NNs seemed to have better ROC curves compared to the other types of classifiers. The kNN seems to

have better ROC curves than SVMs for mutation rates 1%, 15%, and 30%. However, SVMs showed a better ROC curve compared to kNN for mutation rate 50%.

Experiment 2, combined the decisions of NNs, SVMs, and kNN to come up with a single collective decision. Figure 19 shows that this voting scheme resulted in performance just as good as the performance of the best single classifier, if not, better. Since NNs appeared to have the best performance when compared with SVMs and kNN, the voting classification followed the NN's performance for mutation rates 1%-25%. This could be justified by the fact that for these mutation rates, the classification rates were fairly high to start with. However, when the classification rates started to descend (for mutations 30%-50%), the voting classification did a good job reasonably improving the decision of the best single classifier.

In Experiment 3, the same 11 populations in Experiment 1 were used and 5 different levels of noise and blur were added to the 22,000 images. This entire experiment was repeated 10 times, from the simulation up, as a means of validating the results. Then box and whisker plots were drawn for each type of classifier from 2 perspectives – once with respect to the mutation rate, and once with respect to the distortion level. Figure 20 shows the performance of NNs for different distortion levels as the mutation rate is varied. For a given distortion level (boxes of the same color), the general trend was that, as the mutation rate was increased, the classification rate declined. Also, when comparing a given distortion level (a whole set of boxes of the same color), to the other levels of distortion, the higher distortion levels yielded lower classification rates than lower ones. For the different distortion levels, the decrease in performance seemed steady as the mutation rate increased, until mutation rate 35%. After that, there was a sharp decline in performance for mutation rates 40% to 50%. Along with this sharp descent, the boxes for these higher mutation rates became taller as the mutation rate increased. Taller boxes indicate that different classification readings for the same level of distortion and mutation had higher dispersion (higher range which is the difference between the maximum and minimum readings) than the shorter boxes. Also, for these higher mutation levels, the

distances between the boxes of different distortion levels increased as the mutation rate did.

As for the second box and whisker plot for NNs in Figure 21, showing the classification rate of different distortion levels for all mutation rates, the results seem to follow the results of the previous figure. As the distortion level was increased, the classification rates steadily declined for a given mutation rate. Also for a given distortion level, mutation rates 1% through 35% seemed to slightly decrease the classification rate (the boxes lie very closely together yet maintaining their expected order). However, for mutation rates 40% to 50%, there were increasingly bigger jumps of the box median locations, and the boxes showed higher dispersion for these higher mutation levels.

It follows that the box and whisker plots for the SVM, kNN, and Voting classification (Figures 22-27) show trends very similar to the NN box and whisker plots, though with slight differences. The first difference is the classification rate for a given mutation rate and distortion level for the different classifiers. Tables 1 and 2 show the average classification rates of classifiers for the first and fifth levels of distortion respectively only for mutation rates 1%, 40% and 50%.

Table 1 Classification Rates for First Level of Distortion

Classification Rate (%) for First Level of Distortion			
	Mutation Rate 1%	Mutation Rate 40%	Mutation Rate 50%
NN	99.6	91.6	74
SVM	99.3	89.7	69.2
kNN	99.3	87	66
Voting	99.6	91.7	74

Table 2 Classification Rates for Fifth Level of Distortion

Classification Rate (%) for Fifth Level of Distortion			
	Mutation Rate 1%	Mutation Rate 40%	Mutation Rate 50%
NN	91.3	86.3	55.1
SVM	91.6	88	54
kNN	91.3	85.9	54
Voting	91.6	88	57

From the average figures above, it is clear that NNs ranked first compared to SVM and kNN when considering the first and fifth levels of distortion for all mutation rates. SVM and kNN then came in. The Voting Classification done in Experiment 4 yielded slightly improved performance compared to using any single classifier.

4.1 Conclusion

This purpose of this thesis was to use artificial life to investigate the process of using machine learning to decode genetic information. Using artificial life models can be beneficial because the same concepts can be applied to biological material. This will enhance the diagnosis, examination, and treatment of patients in the future providing an alternative to existing methods such as MRI and bioluminescence.

The set of experiments performed in this thesis allowed a number of conclusions to be drawn. The first conclusion is that, given a large population of genetically related organisms, the box counting dimension was an appropriate feature extraction method allowing a classification rate of 90% or higher for mutation rates less than 40%. A "break" in the performance seemed to occur after mutating the genome by a rate over

40%. Secondly, NNs performed better than SVM and kNN classifiers. Combining the decisions of these 3 classifiers using a voting scheme, fairly improved the overall classification rate to a 92% before another break would occur. Third, even with the addition of distortion to images, which in reality is part of any imaging process, a NN still performed better than its counterparts. Finally, the box-counting dimension, as a feature extraction technique, seems to be robust under different distortion levels. The real degradation in performance happened to occur at higher mutation rates rather than at higher distortion levels. In other words, the classification rate was more sensitive to the mutation rate as opposed to the distortion level.

These conclusions could help one to design a system for real images in several aspects. Firstly, in a real system, combining classifiers would be necessary to combine the benefits of various classifiers. Secondly, although the distortion is a natural part of any imaging process, reducing the distortion has a positive effect on the classifier performance. Thus, one could use high resolution/quality images in a real application to lessen the distortion effect. Also, one could consider using noise filters or blur reduction methods as part of the preprocessing prior to the classification process to lessen the effect of distortion.

The above conclusions support my thesis statement that using artificial life can be useful in designing machine learning techniques for gene decoding from images. The conclusion that the box-counting dimension is an appropriate feature extraction method when analyzing images having fractal properties supports the usage of such a feature extraction method, especially if the artificial life images greatly resemble the real images. The conclusion that NNs performed better than SVMs and kNNs supports my thesis in the fact that different classifiers will perform differently depending on the problem nature, the datasets, and other factors. Thus, according to the No Free Lunch Theorem it wouldn't be wise to blindly choose a certain classifier for a given problem. Using voting can combine the benefits of several classifiers improving the overall classification effectiveness.

4.2 Future Work

The work in this thesis can be extended in a number of ways. First of all more types of classifiers can be tested and compared with the classifiers used in this thesis. Also, other types of feature extraction techniques (fractal analysis methods) could be tried out and compared to the box counting algorithm which has a high time complexity. In addition, real biomedical images can be used instead of the artificial life models, or other fractal models can be used to model other types of biological structures.

BIBLIOGRAPHY

[1] E. Segal, C. Sirlin, C. Ooi, A. Adler, J. Gollub, X. Chen, B. Chan, G. Matcuk, C. Barry, H. Chang, and M. Kuo, "Decoding global gene expression programs in liver cancer by noninvasive imaging", *NATURE BIOTECHNOLOGY*, **25**, 6, 2007.

[2] R. Dawkins, *The Blind Watchmaker*, New York: W. W. Norton & Company, 1986.

[3] A. Keyserling and R. C. L., Chance and Choice, School of Wisdom Series, Vol. 2, 1999.

[4] S. Mishra, A. Tami, and M. Tate, "Fractal Analysis of Bone Cell Syncytium in Normal and Diseased Bone", Bioengineering Conference, Florida, 293-294, 2003.

[5] R. Duda, P. Hart, and D. Stork, *Pattern Classification*, Second Edition, John Wiley & Sons Publishers, 2000.

[6] D. Ashlock and J. Golden, "Iterated Function System Fractals for the Detection and Display of DNA Reading Frame", IEEE Proceedings of the 2000 Congress on Evolutionary Computation, 2, 1160-1167, 2000.

[7] G. Jayalalitha and R. Uthayakumar, "Estimating the Skin Cancer using Fractals", *International Conference on Computational Intelligence and Multimedia Applications*, 2007.

[8] J. Feng, W. Lin, and C. Chen, "Fractional Box-Counting Approach to Fractal Dimension Estimation", *Proceedings of ICPR '96*, 1996.

[9] J. Samarbandu, R. Acharya, E. Hausmann, and K. Allen, "Analysis of Bone X-Rays Using Morphological Fractals", *IEEE Transactions on Medical Imaging*, 12, 3, 1993.

[10] D. Ashlock and J. W. Goldin III, Evolutionary Computation and Fractal Visualization of Sequence Data, Chapter 11, in Evolutionary Computation in Bioinformatics, Gary B. Fogel and David W. Corne eds., Morgan Kauffman, Invited Publication, 2002.

[11] M. Baldoni, C. Baroglio, and D. Cavagnino, "Use of IFS Codes for Learning 2D Isolated-Object Classification Systems", *Computer Vision and Image Understanding*, 77, 371-387, 2000.

[12] A. Murugan and K. S. Easwarakumar, "DNA Algorithms for Fractal Construction – an Application of the SInsDelP system", *International Journal of Computer Mathematics*, Vol. 84, No. 4, 437-450, April 2007.

[13] M. James, *Topology*, Prentice Hall, 2nd Edition, 1999.

[14] G. Chiara and L. Saitta, "Using Fractals to Learn Image Descriptions by means of Artificial Neural Networks", *IEEE International Conference on Computational Intelligence*, 5, 27, 2952-2955, 1994.

[15] A. Conci and E. Nunes, "Multi-bands Image Analysis using Local Fractal Dimension", *Computer Graphics and Image Processng*, 91-98, 2001.

[16] B. Mandelbrot, *The Fractal Geometry of Nature*. W.H. Freeman and Company, 1982.

[17] R. Plotnick, R. Gardner, W. Hargrove, and K. Prestegaard, "Lacunarity Analysis: A General Technique for the Analysis of Spatial Patterns", *Physical Review E*, **53**, 5461-5468, 1996.

[18] J. Keller, S. Chen, and R. Crownover, "Texture Description and Segmentation through Fractal Geometry", Computer Vision, Graphics, and Image Processing, **45**, 150-166, 1989.
[19] P. Dong, "Test of a New Lacunarity Estimation Method for Image Texture Analysis," *International Journal of Remote Sensing*, **17**, 3369-3373, 2000.
[20] F. Kenneth, *Fractal Geometry: Mathematical Foundations and Applications*, John Wiley & Sons, Ltd., xxv, 2003.
[21] T. Back, *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford Univ. Press, 1996.
[22] Y. Tao, "Extraction of Fractal Feature for Pattern Recognition", *15th International Conference on Pattern Recognition (ICPR'00) IEEE Computer Society,* **2**, 2527-2530, 2000.
[23] H. Peitgen, H. Jurgens, and D. Saupe, *Chaos and Fractals: New Frontiers of Science*, Springer-Verlag, NewYork, 1992.
[24] E. Parzen, On Estimation of a Probability Density Function and Mode, Ann. Math. Stat. 33, 1065-1076,1962.
[25] E. Alpaydin, "Voting Over Multiple Condensed Nearest Neighbors", *Artificial Intelligence Review*, **11**, 115-132, 1997.
[26] J. MacQueen, "Some Methods for Classification and Analysis of Multivariate Observations", *Proceedings of 5th Berkeley Symposium on Mathematical Statistics and Probability*, **1**, 281-297, 1967.
[27] S. Haykin, *Neural Networks: A Comprehensive Foundation*, Prentice Hall, 1999.
[28] B. Ripley, *Pattern Recognition and Neural Networks*, Cambridge, 1996.
[29] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, 1995.
[30] R. Bellman, Eye of the Hurrican, an Autobiography, *World Scientific Publishing*, 1984.
[31] J. Platt, *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*, Microsoft Research Technical Report MSR-TR-98-14, 1998.
[32] D. Gonzalez, Fractal Time!, Career End Project: Graphic Generation of Fractals, Department of Sciences of Computation and Artificial Intelligence, University of Seville.
[33] Matlab 2007a, The Mathworks Inc. available at: http://www.mathworks.com.
[34] I. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.
[35] J. Dattorro, *Convex Optimization & Euclidean Distance Geometry*, Meboo Publishing USA, 2005.
[36] T. Fawcett, *An Introduction to ROC Analysis*, Pattern Recognition Letters, 27, 861-874.
[37] T. Lasko, J. Bhagwat, K. Zou, and L. Ohno-Machado, "The use of receiver operating characteristic curves in biomedical informatics," *Journal of Biomedical Informatics*, 38 (5), 404-415, 2005.
[38] J. Tukey, "Exploratory Data Analysis", Addison-Wesley, Reading, MA, 1977.