DISCRIMINANT PROCESSING IN MULTI-CLASS PATTERN RECOGNITION SYSTEMS

by

SOUMITRO SWAPAN AUDDY

Presented to the Faculty of the Graduate School of
The University of Texas at Arlington in Partial Fulfillment
of the Requirements
for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2013

Acknowledgements

This thesis is the culmination of the efforts of several individuals. I would like to thank Dr. Michael Manry, my supervising professor, for his constant input and guidance. I would also like to thank Rohit Rawat and Kanishka Tyagi, for their valuable inputs during this process. I thank Nayana and Sheetal for keeping me focused on the task at hand; my roommates Anup, Madhukar, Neeraj and Shantanu; members of House 140/141, Dilip, Raghu, Asha, Shwetha, Gautam and Karthik; my friends in India, Faris, Nishant and Akshay. Last, but definitely not the least, I would like to thank my parents, Swapan and Monika Auddy, for their unwavering belief in me and my decisions. None of this would have been possible without them.

25th Nov, 2013

Abstract

DISCRIMINANT PROCESSING IN

MULTI-CLASS PATTERN RECOGNITION

SYSTEMS

Soumitro Swapan Auddy, M.S.

The University of Texas at Arlington, 2013

Supervising Professor: Michael Manry

Multi-Layer Perceptron neural network classifiers face problems when applications have numerous output classes. A major problem is the fact that the MLP discriminant values given by the MLP differ considerably from the posterior probabilities of the Bayes decision rule. A non-linear mapping technique is developed in this thesis, which warps the neural network outputs into posterior probabilities. A second problem is that when the neural network is given inputs for classes it is not trained to handle, the output discriminant values become very noisy, as compared to the values seen for correct inputs. Variance based methods are investigated for detecting unanticipated classes. A method is developed for detecting cases where a class is confused with another. In this case, a follow on chapter helps clear up the confusion.

Table of Contents

List of Illustrations

List of Tables

List of Acronyms

Acronym	Definition
2D-DFT	2 Dimensional Discrete Fourier Transform
LM	Levenberg-Marquardt
LPR	License Plate Recognition
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
NFL	No Free Lunch
NNC	Nearest Neighbor Classifier
OCR	Optical Character Recognition
OR	Output Reset
ROI	Region of Interest
SIFT	Scale Invariant Feature Transform
SURF	Speeded Up Robust Features
SVM	Support Vector Machine

List of Symbols

Symbol	Definition
$\mathbf{F}_d$	Original image
$\mathbf{f}_d$	Character image
$\mathbf{X}$	Input vector
$\mathbf{T}$	Desired output vector
N	Number of inputs
M	Number of outputs
N_h	Number of hidden units
$\mathbf{W}$, $\mathbf{W}_{oi}$, $\mathbf{W}_{oh}$	Weight matrices
$\mathbf{G}$, $\mathbf{G}_{oi}$, $\mathbf{G}_{oh}$	Gradient matrices
$\mathbf{w}$	Weight vector
$\mathbf{g}$	Gradient vector
$\mathbf{d}_1$	Original discriminant vector
$\mathbf{S}$	Discriminants after softmax operation
$\mathbf{d}_2$	Remapped discriminants
c	Constant

Chapter 1

Introduction

1.1 Basics of Pattern Recognition

A standard image contains vast amounts of information, most of which, is redundant for applications in object detection, machine vision, etc. This redundancy creates a need for an efficient way to extract crucial information from an image, while discarding the redundant information. This method is called Pattern Recognition. Pattern recognition is defined as “the automatic recognition, description, classification, and grouping of patterns” by Jain et al. [1]. In essence, it is a process of distinguishing between images of two or more classes, using the information unique to all images within a particular class, from the images belonging to a different class [2].

Numerous computer vision applications rely on the correctness of the pattern recognition engine for accurate identification, tracking, etc. of objects. Pattern recognition is used in face tracking [3] for selective photography used in digital cameras. Video tracking is the process of locating a moving object, found in applications like medical imaging [4], video editing [5], etc. Patterns found in 3D point clouds are used for motion tracking and gesture recognition [6]. Features like SIFT [7] and SURF [8] are used for reconstruction of 3D environments using 2D images, obtained from standard cameras. Pattern recognition is used in the field of health care, to detect cardiovascular problems such as atrial fibrillation, isochemic beats, etc. by analyzing ECG signals [9] and to analyze mammograms to find micro calcifications, an indicator to possible breast cancer [10]. IC fabrication techniques often lead to defects on the silicon surface [11], and the processing steps to remedy each defect are different, making the classification of the defects essential. Feature extraction and pattern recognition techniques are used extensively for this purpose [11]. Given a sufficient amount of training data, it is possible to use pattern recognition to

predict annual weather patterns accurately [12], to determine the daily electrical load patterns. Optical character recognition (OCR) is of particular interest, due to the numerous approaches available to the implementer.

OCR systems are used in parcel and mail sorting facilities [13], to read the addresses, delivery priorities, etc. electronically and sort them accordingly. OCR systems are now used as teaching aids in schools [14], where students trace characters using a stylus, with the objective of improving their handwriting skills. OCR is also used in license plate recognition (LPR) systems for speed cameras [15], toll roads [16], parking lots [17], etc. Some methods attempt to reconstruct the license plate image [18], while others simply find the location of the license plate [19] and use various methods to recognize each character.

1.2 Feature Extraction

One of the first steps in pattern recognition, for all applications, is feature extraction which can be defined as the process of generating features which can be used as inputs to a classifier, which can then be used to classify the image into one of several possible classes [20]. Feature extraction focuses on the unique properties of an image and extracts the relevant information from the same. Different feature types focus on different aspects of the data, to extract features based on unique properties.

Some methods use transforms such as the Fourier transforms to concentrate on the unique frequency components of the data at different frequency ranges [21]. Other frequency based features attempt to classify information of shapes, such as those seen in fingerprints, etc., for classification purposes [22]. Wavelet transform based feature sets [23], including Gabor filters [24], have been used for applications with large image databases. Some methods use the edge information of images to detect patterns such as finger gestures [25]. Features such as SIFT and SURF features may also be used for

gesture recognition [26], 3D reconstruction [8] of environments using 2D images, etc. Features like geometric moment features [27] are used due to their properties of scale and rotation invariance. Feature extraction is also used for detection of tampering in video footage, such as that from security cameras, by combining feature extraction and feature fusion techniques [28].

1.3 Classification

Pattern recognition systems can be classified as to the type of learning algorithms implemented, like supervised learning, unsupervised learning, reinforcement learning and evolutionary learning [29]. Recognition is also dependent on the type of classifier used, i.e. two class classifiers or multi-class classifiers, etc.

The most basic working principal of a classifier comes from the Bayes decision rule. It assigns posterior probabilities to each possible output class, and the class with the maximum probability is accepted as the correct class, while all other classes are rejected [2]. This approach, however, is not robust, as it does not accommodate for noisy estimates of the posterior densities. As a result, a Bayes classifier is a theoretical concept, which is never used for practical applications. There are, however, other classification structures, which are more easily implemented and which approximate the Bayes classifier arbitrarily well.

A k-means Nearest Neighbor classifier (k-NNC) is a simple, yet efficient machine learning algorithm, used for many practical applications [30]. In a nearest neighbor classifier, in order to classify an unknown input vector, its nearest neighbors are found from a sample vector field. The unknown input vector is assigned the class number to which a majority of the neighboring sample vectors belong [31]. It has been shown that the k-Nearest Neighbor classifier approaches the Bayes classifier's performance as the number

of sample vectors approaches infinity [32]. When there are many sample vectors, however, this is a computationally taxing method of classification.

A Bayes-Gaussian classifier is a standard Bayes classifier, where the input vectors have a Gaussian distribution. The classifier calculates the posterior probabilities of the output classes and the highest probability is assumed to indicate the correct class. This classifier potentially has a lower implementation time than a nearest neighbor classifier, as it has only one mean vector per output class. The final decision is also independent of missing input values, as these values are ignored while the posterior probabilities are being computed, thus making the Gaussian-Bayes classifier highly robust [33]. One problem with this classifier occurs if some input features are linearly dependent on others for a given class. In this case, the required inverse covariance matrix for that class cannot be calculated. A second, more serious problem is that most applications do not generate Gaussian feature vectors.

A support vector machine (SVM) is a two class classifier, where the classification is performed by separating data by a hyperplane [34]. The goal of SVM training is to find the hyperplane with the largest margin. The generalization of the classifier is directly proportional to its margin size [2].

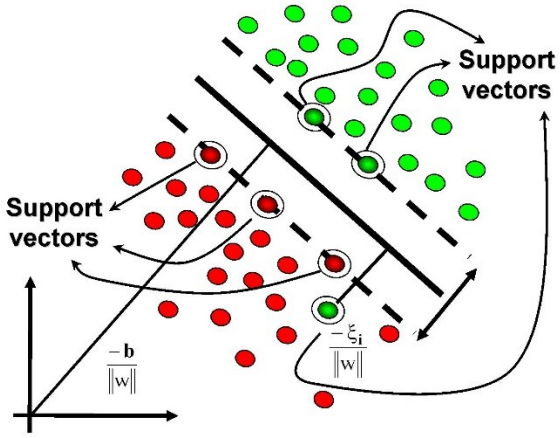

Figure 1-1 Support Vector Machine hyperplane

One advantage of an SVM is the fact that its training is not adversely affected if the number of training patterns for one class is far greater than those of another class. For applications with multiple output classes, multi-class SVMs may be used, but they commonly suffer from combinatorial explosion. Also, given a simple statistical model for testing input vectors in terms of the given training vectors, it can be shown that SVMs reduce the probability of classification error and therefore, may approximate Bayes classifiers [34].

Neural network classifiers are based on either the multilayer perceptron (MLP) structure, or, the radial basis function structure. The structure consists of multiple layers, i.e. an input layer, an output layer and one or more hidden layers [35]. The layers are connected to each other using matrices of weights or coefficients. An MLP based neural network is a non-linear approximator which uses non-linear activation functions in the hidden layers [36]. As opposed to a Bayes-Gaussian classifier, the performance of a neural network is not adversely affected by the presence of linearly dependent input vectors [2]. Neural networks are naturally multiclass classifiers, making them more flexible than two class classifiers like the SVM [37]. The neural network outputs approximate the posterior

probabilities of each output class [2] [38], and thus neural networks approximate Bayes discriminants.

1.4 License Plate Recognition

License plate recognition systems are typically multistage systems, with each successive stage is dependent on the previous one, akin to an instruction pipeline. Broadly, the stages are divided as (i) Plate Finding; (ii) Plate Segmentation; (iii) Character Recognition, as shown in Figure 1-2. Each of these stages is further divided into multiple steps and depending on the resources available, multiple algorithms can be used to perform the same tasks.

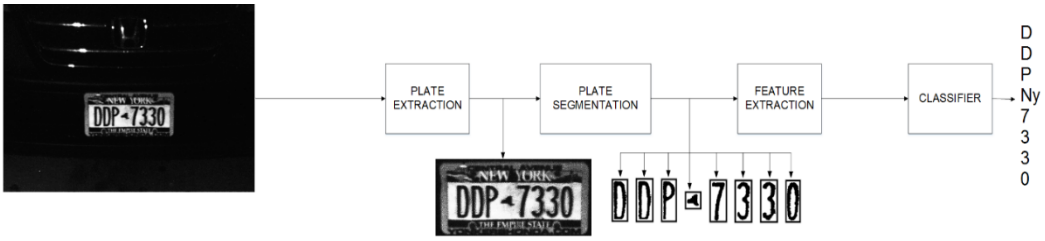

Figure 1-2 Multi-stage License Plate Recognition System

License plate recognition systems are seen in applications such as toll road cameras [19], parking lot monitoring systems [17], traffic signal cameras [19], vehicle weigh stations [39] etc. The accuracy of the systems, when placed in such diverse environments [40], with such diverse scenarios, is of critical importance. LPR systems may need to predict whether or not a given plate is correctly classified. This probability depends directly on the error probabilities of the shapes found during segmentation.

1.4.1 Damaged and New Characters

License plate characters are not always perfect. Often, the characters are damaged due to dents in the license plate. The camera angle with respect to the license

plate may be such that the characters may be incomplete due to shadows and various obstructions such as those due to license plate frames, random glints and other artefacts shaped like bars [41].

The license plates may also have various ancillary characters, such as depictions of wheel chairs, logos of specific companies, etc., which the classifier is not trained to recognize. In such cases, the classifier is almost guaranteed to misclassify the character. Characters may also be damaged during the process of segmentation [42], wherein, a single region of interest may contain portions of multiple characters. Incorrect thresholding during the binarization stage [43] may lead to erroneous artefacts connected to the correct character image.

1.4.2 Confused Characters

Characters with similar shapes and dimensions, characters which, due to the angle of the camera, appear to be other valid characters, etc. also cause problems in LPR systems. In such cases, a two classifier system may be used, where the second classifier is specific to reclassifying a group of characters which may be confused.

1.4.3 State Recognition

License plates usually show of the state of registration of the vehicle. This state information, however, is displayed differently for each state. Some plates contain a state map, while some carry the full name of the state. The character fonts for the states is also not uniform and may be in a mixture of upper and lower case, or may only be in upper case format.

Such differences cause problems in the segmentation step of an LPR system. State specific solutions may be implemented to avoid this. But for a more general, universal LPR solution, it is a challenge to accommodate the numerous different state information styles seen on license plates.

1.5 Work and Thesis Organization

In this thesis, we develop neural network based discriminants for a license plate character recognition application. Chapter 2 reviews the multilayer perceptron classifier used in the thesis. It details the MLP training algorithm and the decision rules it follows. Also covered are relevant neural network theorems. Chapter 3 details the steps involved in a license plate recognition system. It describes various plate extraction methods, plate segmentation algorithms and feature extraction methods. In chapter 4, problems with MLP neural network classifier discriminants are discussed and a non-linear mapping scheme is detailed, to remedy them. Methods for detecting bad characters are discussed in chapter 5. Chapter 6 proposes a method to identify confused input images and proposes a two classifier solution to correct it, thus improving the error percentage.

Chapter 2

Neural Network Classification

Artificial neural networks attempt to mimic the biological model, for applications in pattern recognition and machine learning. The smallest building block of an artificial neural network is the hidden unit which is modeled on the structure of the standard biological neuron. The dendrites are equated to inputs, the axon is akin to the output and the decision making nucleus is represented by numerical weights. A positive weight simulates a neural excitation, while a negative weight simulates neural inhibition. An activation function controls the magnitude of the output, to help mimic the actual working of a biological neural network.

2.1 Structure of a Multi-Layer Perceptron

Artificial neural networks are structured so as to replicate biological neural networks. They consist of an input layer, an output layer and one or more hidden layers. The output layer consists of nodes, with each node corresponding to a different output class. Nodes in a layer are connected to nodes in another layer by unique numerical values, termed as weights. Nodes within the same layer are not connected to each other. A fully connected MLP with one hidden layer is shown in Figure 2-1. The input to the MLP consists of N elements. However, one input is added for every input vector, p, and this input is valued at 1, to function as a biasing threshold. As a result, the input is a vector, $\mathbf{x}_p$ with dimension 1 × (N + 1). The MLP has M output classes, making the output, $\mathbf{d}_1$, a vector of dimension 1 × M. The input and output nodes are connected to each other through a series of hidden unit nodes. An MLP contains N_h such hidden units. The input layer is connected to the hidden layer through a network of weights, represented by the input weight matrix **W** which is N_h × (N + 1). The weights connecting the nodes in the hidden layer to the output

nodes is the output weight matrix $\mathbf{W_{oh}}$, having dimensions M × N_h. Some of the input nodes are directly connected to output layer nodes through the bypass weight matrix $\mathbf{W}_{oi}$, which has dimensions M × (N + 1).

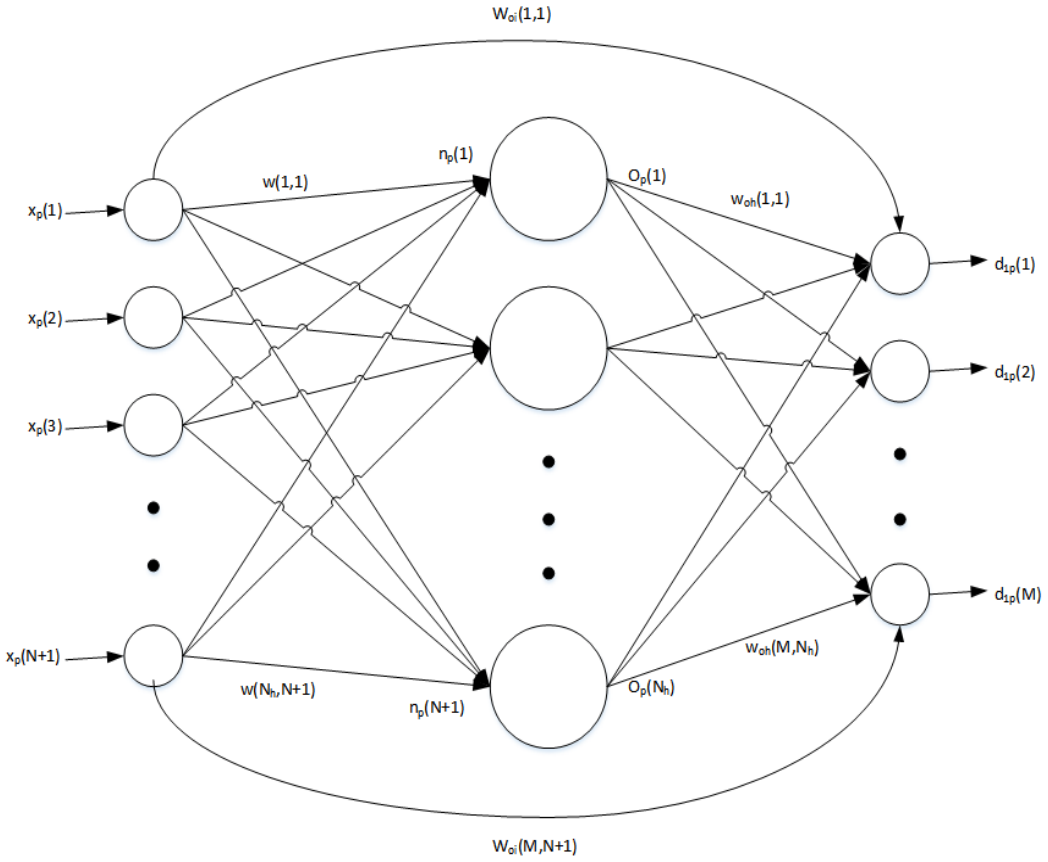

Figure 2-1 Fully Connected Multi-Layer Perceptron Structure

The output of an MLP is a function of the input vector ***x***, the weight matrices and the hidden unit activation vector, ***O***. The non-linearity of a neural network's hidden layer is what helps it mimic biological neural networks [44] and this non-linearity is achieved through a combination of the input to the layer and a sigmoid activation function. The net function at the hidden units is given by (1)

$$n(k) = \sum_{n=1}^{N+1} w(k,n)x(n) \tag{1}$$

The activation vector ***O***, for the p^{th} training pattern, is given by (2)

$$O_p(k) = f(n(k)) = \frac{1}{1 + e^{-n(k)}} \tag{2}$$

The expression for the output of the classifier, for the p^{th} training pattern, thus becomes,

$$d_1(i) = \sum_{k=1}^{N_h} w_{oh}(i,k)O_p(k) + \sum_{n=1}^{N+1} w_{oi}(i,n)x_p(n) \tag{3}$$

The output values for each class is called a discriminant and the class with the highest discriminant value is considered to be the correct class. As such, the correct class number is i_c and the estimated class decision, i'_c is given by

$$i_c{}' = argmax_i(d_1(i)) \tag{4}$$

2.2 Training a Neural Network

The process of determining the weights of a neural network, to achieve optimum performance, is called training. Training involves feeding the neural network inputs with known output classes. A constant comparison and minimization of the mean squared error of the known output class and the predicted output class forms the basis of neural network training [45].

$$E = \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} \left[t_p(i) - d_{1p}(i)\right]^2 \tag{5}$$

The objective here is to find the most optimum weights such that the actual output of the classifier, $\boldsymbol{d_1}$ is almost equal to the desired output ***t***, for the p^{th} training pattern. The neural networks are trained using different gradient techniques, all focusing on optimizing the weights using the calculated gradients.

2.2.1 First Order Training Methods

Numerous first order methods are available for training a neural network. Each method has its own trade-offs, with respect to ease of implementation, speed of learning, robustness to noise, etc. [45]. The following algorithms can be implemented on a fully connected MLP structure with a single input layer, a single hidden unit layer and a single output layer. The term fully connected implies that all the nodes in the MLP are connected to all other nodes in different layers by weights.

The first order methods take the first order derivatives of the error function with respect to the weight matrices $\boldsymbol{W_{oi}}$,$\boldsymbol{W}$ and $\boldsymbol{W_{oh}}$. The corresponding negative gradient matrices for these individual regions are given by

$$\boldsymbol{G} = \frac{-\partial E}{\partial \boldsymbol{W}}, \tag{6}$$

$$\boldsymbol{G}_{oi} = \frac{-\partial E}{\partial \boldsymbol{W}_{oi}}, \tag{7}$$

and

$$\boldsymbol{G}_{oh} = \frac{-\partial E}{\partial \boldsymbol{W}_{oh}} \tag{8}$$

The negative gradient vector g, for the network is

$$\boldsymbol{g} = vec(\boldsymbol{G}, \boldsymbol{G}_{oi}, \boldsymbol{G}_{oh}) \tag{9}$$

where the vec() function maps G, Goi and Goh to a column for convenience. The weights W, Woi and Woh are remapped to form a column vector w, as

$$\boldsymbol{w} = vec(\boldsymbol{W}, \boldsymbol{W}_{oi}, \boldsymbol{W}_{oh}) \tag{10}$$

The weights are updated in each iteration, k, using a learning factor z and the negative gradient vector, g. The initial value of z is a small constant value and this value is increased by a predefined step, if the error function for the current iteration is less than that

seen in the previous iteration, else, it is decreased. In back propagation [46], for the (k + 1)th iteration, wk representing the weights in the kth iteration and wk+1 representing the weights in the (k + 1)th iteration, the weights are updated as,

$$\boldsymbol{w}_{k+1} = \boldsymbol{w_k} + z \cdot \boldsymbol{g}_k \tag{11}$$

Back-propagation, despite being a preferred method of training neural networks, has a slow convergence rate and is heavily dependent on the selection of the initial weights [47], making it unsuitable for many large applications.

The conjugate gradient method is a related first order training method, where the gradient gk is used to update a direction vector pk for the kth iteration. The direction vector is updated as

$$\boldsymbol{p}_{k+1} = -\boldsymbol{g}_k + B_1 \cdot \boldsymbol{p}_k \tag{12}$$

using a factor B1 calculated as,

$$B_1 = \frac{\boldsymbol{g}_{k+1}{}^T \boldsymbol{g}_{k+1}}{\boldsymbol{g}_k{}^T \boldsymbol{g}_k} \tag{13}$$

This direction vector, in turn, updates the weights as

$$\boldsymbol{w}_{k+1} = \boldsymbol{w_k} + z \cdot \boldsymbol{p}_k \tag{14}$$

Since the weights are updated using the direction matrix as opposed to the gradients themselves, the direction of descent is superior to that seen in the conjugate gradient method. Thus the conjugate gradient method converges in fewer iterations than the steepest descent method [45].

2.2.2 Second Order Training Methods

Second order algorithms are preferred for training neural networks due to their faster convergence [48], but many second order training algorithms lead to memory limitation problems [49], since the Jacobian matrix must be calculated and stored. Second order training methods work on two basic assumptions

2.2.2.1 Assumption 1:

The error function is approximately quadratic in ***w*** for small weight changes [50].

2.2.2.2 Assumption 2:

The output for the p^{th} training pattern is approximated as a first degree function of ***w*** [50]**.**

The following method is Newton's second order training algorithm for a fully connected MLP. The first partial derivative of the error function with respect to the weights is given by

$$\frac{\partial E}{\partial w(j,k)} = -\frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}[t_p(i) - d_{1p}(i)]\frac{\partial d_{1p}(i)}{\partial w(j,k)} \tag{15}$$

The Gauss-Newton approximation of the second partial of the error function with respect to the weights, is

$$\frac{\partial^2 E}{\partial w(j,k)\partial w(u,v)} = \frac{2}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{M}\frac{\partial d_{1p}(i)}{\partial w(j,k)}\frac{\partial d_{1p}(i)}{\partial w(u,v)} \tag{16}$$

As described in **Error! Reference source not found.**, the weight matrix is remapped into column major format, giving the weight vector ***w***. The remapping of *w* from a two dimensional to a one dimensional vector ***w*** is performed as

$$\boldsymbol{w} = vec(w) \tag{17}$$

Using this mapping approach, the second order derivatives of (16) become elements of the Hessian matrix ***H*** is defined as

$$h(m,n) = \frac{\partial^2 E}{\partial w(m)\partial w(n)} \tag{18}$$

Newton's update vector ***e*** is found by solving,

$$\boldsymbol{H} \cdot \boldsymbol{e} = \boldsymbol{g} \tag{19}$$

The new weight vector ***w'*** is updated as

$$\boldsymbol{w}' = \boldsymbol{w} + \boldsymbol{e} \tag{20}$$

The problem with Newton's training method is the likelihood that the Hessian matrix ***H*** is singular [50]. As a result of which, it may not be possible to calculate the update vector ***e***. Hence, the Levenberg-Marquardt training algorithm is proposed, which eliminates this possibility.

2.2.3 Levenberg-Marquardt Algorithm

The Levenberg-Marquardt's algorithm is an iterative technique used to find the local minima of a multivariate function. This algorithm is an interpolation between Newton's algorithm and steepest descent, which leads to a guaranteed convergence at a slower speed [51]. The weight vector ***w***, in the LM algorithm, is updated as

$$\boldsymbol{w} = \boldsymbol{w} + [\boldsymbol{H} + \lambda \boldsymbol{I}]^{-1}\boldsymbol{g} \tag{21}$$

which implies

$$[\boldsymbol{H} + \lambda \boldsymbol{I}]\boldsymbol{e} = \boldsymbol{g} \tag{22}$$

The term λ controls the LM training towards either the first order or the second order methods [51]. If the error function *E* increases, λ is increased, making the algorithm mimic the steepest descent method. Else, the term λ is decreased. The equation in (22) is solved using orthogonal least squares (OLS) [52]. LM may be used on all the network weights, or only on the hidden layer weights. The problem with LM is that the number of calculations involved in it mean it cannot be used effectively in larger networks. Thus, LM is generally used to train smaller networks.

2.2.4 Output Reset

For the MLP, it is desirable to minimize the probability of classification error, but it is more practical to minimize the MSE *E*. The error function puts certain restrictions on the classification error. If the outputs have a unique constant bias, E may be increased or decreased, but the classification error remains unaffected. In the event that an output has a magnitude greater than the output bias, E will increase, leaving the classification error unchanged or lesser [53].

The output reset (OR) algorithm assigns each desired output a bias which reduces the error function and sets the desired output equal to the actual output, when the output has the correct sign but too great a magnitude [53]. The resulting modified error function *E'* is expressed as

$$E' = \frac{1}{N_v}\sum_{p=1}^{N_v}\sum_{i=1}^{N_c}[t'_p(i) - d_1(i)]^2 \tag{23}$$

where,

$$t'_p(i) = t_p(i) + a_p + d_p(i) \tag{24}$$

By varying a_p, $d_p(i)$ or $t'_p(i)$ for each training pattern p, it is possible to decrease E'. If i_d is an incorrect class number, the methods must adhere to two conditions,

i. The difference $\left|t'_p(i_c) - t'_p(i_d)\right|$ must be greater than, or equal to *1*.

ii. All changes made to these parameters must reduce, or not affect *E'*.

The first method equates the first derivative of *E'*, with respect to a_p, equal to zero. The expression for a_p is then obtained as,

$$a_p = \frac{1}{M}\sum_{i=1}^{M}\left[y_p(i) - t_p(i) - d_p(i)\right] \tag{25}$$

The second method attempts to modify the term $d_p(i)$, such that

$$d_p(i) = y_p(i) - t_p(i) - a_p \quad (26)$$

with the constraints that $d_p(i_c) \geq 0, d_p(i_d) \leq 0$. By implementing OR, the outputs corresponding to the classes with lower probabilities are forced to be lower and those with higher probabilities of being correct are assigned higher values. This effectively increases the number of negative discriminants at the output of the MLP and increases the magnitude of the maximum discriminant value, corresponding to the correct class.

For a given input vector which can belong to one of 36 possible output classes, class 1 being the correct class, Figure 2-2 shows the discriminant values obtained at the output of an MLP trained without implementing OR. The same input vector, when passed through an MLP trained with OR, gives the discriminant values seen in Figure 2-3 It is seen in Figure 2-2 and Figure 2-3 that most output classes have either positive or negative discriminant values. The discriminant value corresponding to the correct class has a much higher value at the output of the MLP trained with OR, compared to the MLP trained without OR. The significance of increasing the number of negative discriminant values without affecting the overall performance of the network will be seen in **Error! Reference source ot found.**.

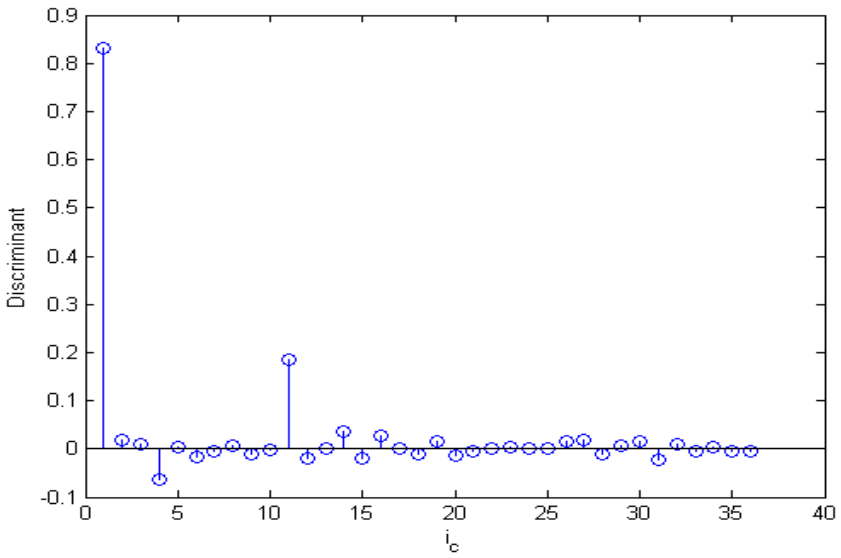

Figure 2-2 Sample discriminant values for a network trained without OR

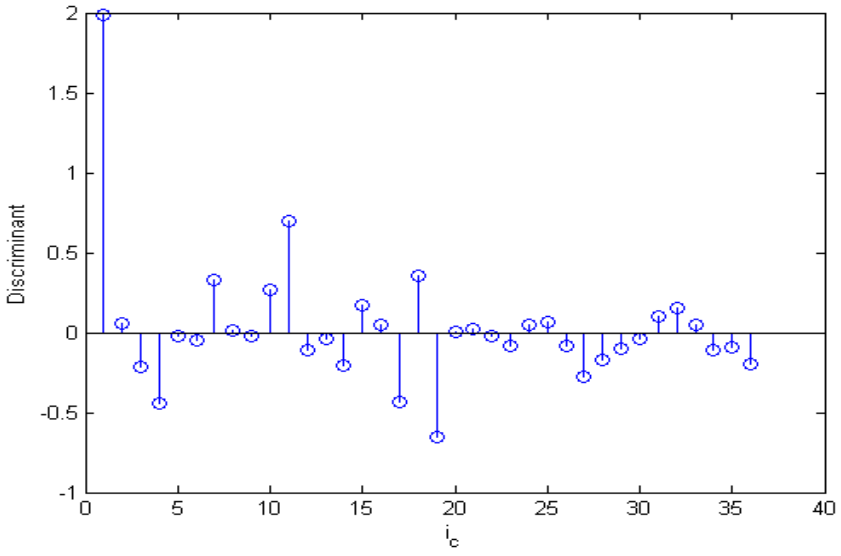

Figure 2-3 Sample discriminant values for a network trained with OR

2.3 Properties of the Multi-Layer Perceptron

2.3.1 Modelling a Noisy Discriminant

From Figure 2-2and Figure 2-3, the discriminant output of an MLP is not ideal, i.e. it has noise. Thus, the i^{th} discriminant is modeled as

$$d_1(i) = d(i) + n_d(i) \tag{27}$$

where the discriminant vector ***d*** sums to *1* and the noise component has a mean m_n. We analyze some of the properties of this model in the following two lemmas.

2.3.1.1 Lemma 1

The expected sample mean of $d_1(i)$ converges to the mean of the additive noise as the number of classes M increases.

$$E\left[\frac{1}{M}\sum_{i=1}^{M} d_1(i)\right] = E\left[\frac{1}{M}\sum_{i=1}^{M}\left(d(i) + n_d(i)\right)\right] \tag{28}$$

$$= \frac{1}{M} + E\left[\frac{1}{M}\sum_{i=1}^{M} n_d(i)\right]$$

$$= \frac{1}{M} + m_n$$

As M increases, the discriminant sequence's mean approaches m_n.

2.3.1.2 Lemma 2

If the desired output for the i^{th} class is $t_p(i) = \delta(i-i_c)$ where i_c denotes the correct class and the noisy discriminant d_1 has a bias a, found through regression, the noise component of $d_1(i)$ is zero mean.

$$E = E\left[\frac{1}{M}\sum_{i=1}^{M}\left[t_p(i) - d_1(i) - a\right]^2\right] \tag{29}$$

$$\frac{\partial E}{\partial a} = -2 \cdot E\left[\frac{1}{M}\sum_{i=1}^{M}\left[t_p(i) - d_1(i) - a\right]\right] = 0 \tag{30}$$

$$\therefore a = E\left[\frac{1}{M}\sum_{i=1}^{M}[t_p(i) - d_1(i)]\right] = \frac{1}{M} - \frac{1}{M}\sum_{i=1}^{M}E[d_1(i)] \tag{31}$$

Absorbing *a* into $d_1(i)$ and using lemma 1,

$$\sum_{i=1}^{M} d_1(i) = 1, \tag{32}$$

$$m_n = 0 \tag{33}$$

Thus, the sum of the discriminants of an MLP classifier sum up to *1*, as would posterior probabilities.

2.3.2 Approximating Bayes Posterior Probabilities

A Bayes discriminant [54] which minimizes the probability of error P_e, can be expressed in any of the following three forms

$$f(\boldsymbol{x}|i)P(i) \tag{34}$$

$$g(f(\boldsymbol{x}|i)P(i)) \tag{35}$$

$$P_b(i|\boldsymbol{x}) \tag{36}$$

In (ii) *g(.)* is either an increasing or decreasing function. MLP classifiers have been used successfully for numerous applications, based on their proven performance. This good performance is partially due to the following result.

Theorem: "*When MLP classifiers are trained to minimize the mean-squared error, the MSE approaches a constant value plus the expected squared error between the*

classifier output and Bayes discriminant, as the number of training patterns approaches infinity [55]."

In other words, if the Bayes discriminant vector ***b***, has elements as described in (36) then as the number of training patterns approaches infinity,

$$\lim_{N_v \to \infty} \frac{1}{N_v} \sum_{p=1}^{N_v} \sum_{i=1}^{M} [t_p(i) - d_{1p}(i)]^2 = \sum_{i=1}^{M} E\left[\left(b(i) - d_1(i)\right)^2\right] + c \tag{37}$$

where *c* is a constant, independent of the p^{th} training pattern. Experiments have shown that an MLP classifier often gives the same accuracy as conventional non-parametric Bayes classifiers [54]. The Bayes posterior probability for an input vector ***x***, of the p^{th} training pattern, belonging to a class *i* within the output discriminant vector $\boldsymbol{d}_1$ is given by

$$d_1(i) = P_b(i|\boldsymbol{x}_p) \tag{38}$$

The training of an MLP classifier involves the minimization of the MSE function *E* as seen in (16), which is minimized over all the training patterns. Thus, *E* is the ensemble error surface for a pattern recognition problem [54]. The input training vectors are drawn from a common, known distribution of vectors, the number of training vectors is effectively proportional to the a priori probability of any given class [54]. Since the training minimizes *E* with respect to the weights vector ***w***, the MLP is effectively a minimum mean squared error approximation to the Bayes optimal discriminant function [54]. It must be noted that the accuracy of the MLP in approximating the Bayes posterior probabilities is dependent on the weights vector ***w***. If the number of hidden units is too small, the approximation of the MLP will not be a good approximation of the Bayes posterior probabilities well.

2.3.3 Memorization

A nonlinear neural network has the ability to memorize its input patterns during training. The number of patterns it can memorize is called information capacity [56]. The ability of a neural network to memorize relates to its ability to form arbitrary shapes in the weight space, and attests to the usefulness of the training algorithm used on the neural network [57]. However, in training a neural network, memorization must be avoided as it leads to the possibility of overfitting [58]. Overfitting ultimately negatively affects the validation and testing errors of the MLP. Thus, it becomes essential to find the upper bound on memorization, in order to design an optimum classifier.

For a given application, the parameters known to us beforehand are, the number of inputs, N, the number of input training patterns, N_v and the number of output classes, M. Thus, to design an optimum classifier, the only parameters under our control are the numbers of hidden units, N_h and the training iterations, N_{it}. The upper bound C_{MLP} [59], also called the storage capacity, must satisfy

$$C_{MLP} \leq \frac{N_w}{M} \tag{39}$$

where N_w is the total number of weights,

$$N_w = (N+1)N_h + N_h M + (N+1)M \tag{40}$$

In order to prevent memorization and promote generalization, N_v must be much greater than C_{MLP}.

$$N_V \gg \frac{N_w}{M} \tag{41}$$

From this expression, the number of hidden units should be chosen to satisfy

$$N_h \ll \frac{M(N_v - N - 1)}{(N + 1 + M)} \tag{42}$$

This upper bound of (39) is independent of the activation function used and is also valid for most feed forward neural network structures, irrespective of the connectivity of the network [57].

2.3.4 Universal Approximation Theorem

A question that arose in the late 1980s is, 'How many hidden layers are needed to sufficiently approximate an arbitrary, continuous function?' the answer is given by the universal approximation theorem. The universal approximation theorem for the nonlinear input-output mapping of Figure 2-1 is stated as follows [35] [60]:

Theorem: "*Let f be a non-constant, bounded and monotonically increasing function. Let* ***x*** *be the input with dimensionality, 1 × N. If each of the input vectors are drawn from a specific distribution and ε > 0, there exist a number, M, and real valued constants* ***x****(N + 1),* $\boldsymbol{W_{oh}}$ *and* ***W****, such that an output* $\boldsymbol{d_1}$*, with dimensionality 1 × M, can be defined as given in (3).", which is an approximate realization of the desired output* ***t***", i.e.

$$|d_1(i) - t(i)| < \varepsilon, for\ 0 < i \leq M$$

For the application of license plate character recognition, the input vectors ***x*** are selected from a continuous distribution. A sigmoidal activation function (2) is used to make the outputs monotonically increasing and bounded, as required in the theorem. The MLP used has one hidden layer with N_h hidden units. The theorem justifies the approximation capabilities of an MLP with a single hidden unit layer and these criteria are adhered to by the MLP used for this classification problem. Hence, as confirmed by the universal approximation theorem, the single hidden layer MLP used here for classification is a good approximator. It however, does not state that the MLP has the optimum learning time, the easiest of implementations or that it is the most general classifier available for all applications [35].

2.3.5 No Free Lunch Theorem

As seen in previous chapters, there are different types of classifiers, some suitable for two class systems, some for multi class systems. There are also many available training algorithms, which can be used interchangeably with these classifiers. The focus now turns to the optimization of a classifier, irrespective of the training and structure of the algorithm, a "black-box" optimization algorithm, as labeled by Wolpert and Macready [61], as the No Free Lunch theorem. The theorem assumes the existence of a search algorithm, *a*, which relies on either deterministic or stochastic extrapolation of information from an existing set of *m* points. The performance of the algorithm is decided by the histogram *c*, of the cost values of the algorithm, decided by a cost function, *F*. Let *F'* represent every possible input vector, from every classification problem. The cost function is a function of the number of patterns, N_v, in the population and the output discriminant values, $\boldsymbol{d}_1$, which is a vector of length *M*. Assuming the existence of F_1, a subset of *F'* input vectors, for which a search algorithm a_1 outperforms an algorithm a_2 for a second subset of *F'*, called F_2. The reverse is true also, i.e., a_2 outperforms a_1 for the set F_2. To perform a comparison of the two search algorithms, the sum over all *F'* of $P(c|f,m,a_1)$ is compared to the sum over all *f* of $P(c|f,m,a_2)$. Since the comparison sums over all of *F'*, it implies that the comparison is independent of the search algorithms, a_1 and a_2, since they are implemented on F_1 and F_2, which are effectively subsets of *F'*. The theorem is thus stated as,

Theorem [62]**:** For any pair of algorithms, a_1 and a_2,

$$\sum_f P(C|F', N_v, a_1) = \sum_f P(C|F', N_v, a_2) \tag{43}$$

The No Free Lunch theorem implies that if nothing is known about *f*, then $P(C|N_v,a)$, which is the probability of obtaining a histogram *C* after N_v evaluations is independent of the algorithm used. Thus,

$$P(C|N_v, a) = \sum_f P(C|F', N_v, a)P(F'|N_v, a)$$
$$= \sum_f P(C|F', N_v, a)P(F') \quad (44)$$

If we have no prior knowledge of the type of input for the classifier being designed, then all *F'* are equally likely.

This theorem implies that a selected algorithm may be an optimum solution for a particular application but it may or may not be optimum for a new application. It is possible that a different algorithm may be optimum for the new case. Thus, it is not possible to determine a single algorithm that is optimum for all possible applications. It also implies that every learning algorithm must make certain assumptions beyond the given data, in order to form a reasonable degree of generalization [63]. Given a lack of such assumptions, the ability of a learning algorithm to correctly predict the class for an input is analogous to random guessing [64].

The multilayer perceptron structure used in this thesis should, according to this theorem, be as competent at predicting correct class numbers for the input feature vectors, as random guessing [64]. But the theorem does not hold true for a multilayer perceptron, because the algorithm is designed making certain assumptions, as discussed in **Error! eference source not found.**.

2.3.5.1 Assumption 3:

The basic criteria for training a multilayer perceptron is the adjustment of weights to minimize the error function, E, The input vectors used to train the MLP belong to a certain distribution [65] and the weights are adjusted to minimize the error for all inputs belonging to the same distribution. Any input vectors which do not belong to the distribution, i.e. new

character classes the MLP has not been trained for, cannot be handled by the classifier. This effectively makes an assumption that all inputs to the classifier belong to a particular distribution, thus imposing a limit on the types of inputs the classifier can handle.

2.3.5.2 Assumption 4:

The hidden layer of the neural network uses a sigmoid activation function, as discussed in **Error! Reference source not found.**, to replicate the non-linear decision aking process of biological neurons. This also serves the purpose of making the outputs continuous, thus imposing a criteria that both the inputs and the activation function, belong to a continuous distribution. We therefore have a second assumption on the working of the classifier. These two assumptions put on the MLP are enough to refute the claims of the No Free Lunch theorem.

2.3.6 Alternate Justification of the MLP Classifier

A second approach can be constructed, that shows that the NFL theorem does not apply to the MLP. First, most classifiers, including the MLP, SVM, Bayes-Gaussian classifier, nearest neighbor classifier (NNC) etc., have continuous discriminant functions, which are discussed in the following lemma.

2.3.6.1 Lemma 3:

The output discriminant functions of the MLP, SVM, NNC and Bayes-Gaussian classifiers are continuous.

For the MLP, as discussed in **Error! Reference source not found.**, the activation unction is a sigmoid function, which is bounded, monotonically increasing and continuous. As a result of this, the discriminant output vector $\boldsymbol{d}_1$ of the MLP is also continuous. The other, linear layers cascaded with the activation layers are also continuous. The SVM is an

MLP with continuous Gaussian activations, and therefore has continuous output discriminants. For a Bayes-Gaussian classifier, each discriminant is a quadratic function of the input vector $\boldsymbol{x}$, making the discriminant continuous.

The i^{th} class discriminant of an NNC is

$$d_i(\boldsymbol{x}) = min_k(d(\boldsymbol{x}, \boldsymbol{m}_{ik}))$$

where $d(\cdot)$ denotes the distance from $\boldsymbol{x}$ to the k^{th} center vector from the i^{th} class, $\boldsymbol{m}_{ik}$. As $\boldsymbol{x}$ changes continuously, the distances $d(\boldsymbol{x},\boldsymbol{m}_{ik})$ change continuously as well. If the two smallest distances for the i^{th} class change place, so that a different one is the minimum, there is no discontinuity in $d_i(\boldsymbol{x})$. This occurs because the two distances are infinitesimally different when they change places.

Second, universal approximation implies that the MLP can approximate these continuous discriminants arbitrarily well. Thus, the MLP can perform at least as well as the best alternative classifier. Given these results, we have decided to use the MLP classifier in this research.

Chapter 3

License Plate Recognition

3.1 Introduction to License Plate Recognition

Automatic license plate recognition is the process by which, a computer based system acquires vehicle images from a camera, locates the license plate and recognizes the license plate number. Depending on the place of application of the LPR system, such as unattended parking lots [17], automatic toll collection [16], speed and traffic cameras [15], etc., the steps to perform this will vary, to accommodate for the numerous restrictions imposed on the images captured. These restrictions come in the form of vehicle speed, ambient lamination, camera image quality, ancillary text on the vehicle, etc. [66]. Although this can be done manually, it is the least efficient brute force method available. A more practical solution is to automate the license plate recognition system using one of the many image processing and pattern recognition methods available to us. As such, there are numerous practical approaches to implement LPR systems. They are, usually, multi-stage applications, with the more generally accepted breakdown being plate finding [66], plate segmentation [67], feature extraction [68] and character classification [61].

3.2 License Plate Finding

This is the first step in any LPR application. Given an image F_d of an environment, the location of the license plate must first be correctly established. Several methods exist to find the location of the license plate from F_d. This step is usually divided into finding the region of interest (ROI) using area based grayscale variations, using histogram analysis, morphological operations, etc. [66] and subsequently isolating the exact license plate using methods like using edge detection followed by Hough transform [69]. Some of the common

problems encountered in these methods is their inability to distinguish between license plates and other forms of text visible on the vehicle, such as bumper stickers, advertisements, etc. Often, glints caused by light reflecting off chromed grills, high beam headlights manage to yield false results for the license plate location. Methods such as block-wise binarization using traditional thresholding algorithms such as Otsu's thresholding [67] to remove spurious high frequency noises. Other methods such as divisive normalization [70] to enhance the contrast in the image, thus facilitating the process of license plate extraction.

Several approaches to license plate finding are currently available in the market. These approaches are based on unique properties of the license plate, such as color [71], shape [40] [70], spatial frequency [24], variance, etc.

3.2.1 Statistical Window Binarization Approach to finding a License Plate

A typical preprocessing step is to binarize the complete image. One commonly used approach is to resize the image to a fixed dimension. Two concentric windows, *A* and *B*, are passed across the image, $\boldsymbol{F}_d$, moved pixel by pixel, from the top to bottom and left to right. If the ratio of the statistical measurements corresponding to these windows, either the mean value or the standard deviation of the regions *A* and *B*, M_A and M_B respectively, is greater than a threshold, *T*, the center pixel of the window is set to logic 1, indicating the pixel is present in the ROI, else, it is cleared to logic 0, indicating it is not part of the ROI [40]. The binarized image $\boldsymbol{F}_{bin}$ derived from this relation is represented in (45).

$$F_{bin}(m,n) = \begin{cases} 0, if\ \frac{M_B}{M_A} \leq T \\ 1, if\ \frac{M_B}{M_A} > T \end{cases} \quad (45)$$

Once the probable region is located, image morphological, *Dilation* [40] and *Erosion* [40] operators are used to remove smaller blobs within the ROI. This method yields the shape of the license plate within the image which is then processed further.

3.2.2 Hue, Saturation and Intensity Based Methods to Locate a License Plate

Some license plate recognition systems are region specific and tend to exploit certain properties unique to certain parts of the world. The most commonly exploited property among these is the method of combining the algorithms for edge detection and summed area table images to find the license plate.

The algorithm [71] first converts the image $\boldsymbol{F}_d$, initially in the red, *R*, green, *G*, blue, *B*, format, to a hue, *H*, saturation, *S* and intensity, *V*, format using the following conversion criteria:

$$\begin{cases} V(m,n) = \max(R(m,n), G(m,n), B(m,n)) \\ S(m,n) = \begin{cases} [V(m,n) - \min(R(m,n), G(m,n), B(m,n))] \times \frac{255}{V(m,n)}; V(n \\ 0 \; otherwise \end{cases} \\ H(m,n) = \begin{cases} (G(m,n) - B(m,n)) \times \frac{60}{S(m,n)}; V(m,n) = R(m,n \\ 180 + (B(m,n) - R(m,n)) \times \frac{60}{S(m,n)}; V(m,n) = G(n \\ 240 + (R(m,n) - G(m,n)) \times \frac{60}{S(m,n)}; V(m,n) = B(n \end{cases} \\ H(m,n) = H(m,n) + 360; if \; H(m,n) < 0 \end{cases} \quad (4\text{-}6)$$

After converting the image to the HSV color space, its integral image is extracted [72]. Using the integral image, the sum of any rectangle can be obtained using the values of the sum at the four corners of the rectangle *A*, *B*, *C* and *D*, shown in Figure 3-1. For illustrative purposes, the calculation of the area within rectangle *D* is described.

The area within rectangle *D* is obtained from the points *1*, *2*, *3* and *4*. The value at *1*, A_1, is the sum of pixels in *A*; the value at *2*, A_2, is the sum of pixels in *A* and *B*; the value

at *3*, A_3, is given by the sum of pixels in *A* and *C*; and the value at *4*, A_4, is given by the sum of pixel values in *A*, *B*, *C* and *D*. The area within region *D* is thus given by,

$$area(D) = A_4 + A_1 - (A_2 + A_3) \quad (47)$$

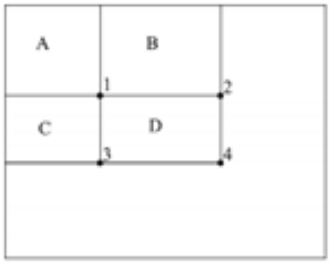

Figure 3-1 Integral image

The edges of the image are extracted by using a Sobel filter and the subsequent image is dilated [73]. As a result, several brighter regions are highlighted in the image. By performing a logical AND operation on the HSV image with the Sobel-dilated image, probable license plate regions are located. A connected component analysis of the rectangular areas from the ANDed image and the integral image yields an accurate license plate.

3.3 License Plate Segmentation

Once the general location of the license plate is identified, it must be segmented, i.e. each individual character must be isolated from the other characters, for the purpose of further processing. This step is challenging due to the inconsistencies in the formatting of the license plate characters and backgrounds. License plates in different states have different, often multicolored backgrounds. Some have holographic emblems and characters, which, depending on the lighting conditions in the vicinity, cause glints and other light artefacts which partially obscure the neighboring characters. The angle of the

camera, on certain occasions, may be too steep, causing the view of the characters to be blocked, partially, by portions of the car, etc.

Several methods have been devised for character segmentation, from image morphological techniques, namely dilation and erosion [74]; using wavelet based methods like the 2D Haar Wavelet transform, as shown by Jeffery et al. [75]; or using robust texture analysis methods using Gabor filters [76], which abandons traditional edge detection and thresholding techniques, in favor of multi-directional filter responses. Some of the methods of license plate segmentation are mentioned below.

3.3.1 Blank Space Detection Based Approach

A paper on character segmentation by Shan [77] uses the most computationally simple method for segmentation. The necessary preprocessing step of binarization is performed on the license plate image. The sum of the pixel values for each column is then calculated. The gap between characters appears as a region with a high value, as compared to the other values, which are low, due to the presence of dark characters. These valleys and peaks are used to segment the characters vertically. A similar step, taking the sum of pixel values in each row, within the vertically segmented plate, is used to create horizontal boundaries, thus segmenting the characters from the plate image.

3.3.2 Neural Network Based Approach

A paper by Garris and Wilson [42] describes a neural approach which combines the tasks of character segmentation and recognition into a single neural network. A rectangular window is passed over the probable location of the license plate characters, assuming a fixed size of the license plate characters. The image within the rectangle are passed through a neural network, to decide whether the image is a valid character, or, if it contains portions of two separate characters.

A second, two network approach is also proposed in the same paper [42]. In this method, the first neural network is tasked with determining whether the image segment at the input is a valid character, or an invalid one. The second neural network determines the correct class of the character, assuming the first neural network deems the image as a valid character.

3.4 Feature Extraction

A survey conducted by Trier, Jain and Taxt [68] discusses the different feature extraction methods for character recognition. The paper covers multiple methods for the segmented characters obtained after most common preprocessing steps, like grayscale images, binarized images and skeletal edge images.

Depending on the type of preprocessing steps used, the feature extraction methods change drastically. This is done to ensure the correct features are the focus of the extraction method being used.

Table 1 List of feature sets for different image types [68]

Gray Scale Subimage	Binary		Vector (skeleton)
	Solid Symbol	Outer Contour	
Template Matching	Template Matching		Template Matching
Deformable Templates			Deformable Templates
Unitary Transforms	Unitary Transforms		Graph Description
	Projection Histograms	Contour Profiles	Discrete Features
Zoning	Zoning	Zoning	Zoning
Geometric Moments	Geometric Moments	Spline Curve	
Zernike Moments	Zernike Moments	Fourier Descriptors	Fourier Descriptors

3.4.1 Filter Based Feature Extraction

Wang, Ding and Liu [24], in their paper, discuss the use of Gabor filters for the process of feature extraction. Gabor filters can simulate the behavior of simple cells in the

human eyes [78]. Gabor filters also have an optimal joint spatial-frequency localization property [24], thus making them suitable filters for applications like texture analysis, handwritten character recognition, etc. A 2-dimensional Gabor filter is a complex, sinusoidal modulated Gaussian function with the responses in the spatial domain and spatial-frequency domain [24]. The directional properties of the filters to detect character strokes in different directions. To accomplish this, the character image first needs to be normalized and the must pass through a bank of Gabor filters. To accommodate for background noise and non-uniform lighting, the outputs of the filter bank pass through an adaptive regulator. The system extracts histogram features by counting the positive and negative real parts of the output of each Gabor filter. This histogram forms input vectors, which is put through a histogram feature extraction process, before finally being compressed into a single feature vector $\boldsymbol{x}$. This is a time consuming and memory intensive feature extraction method, which cannot be implemented easily on simple hardware.

3.4.2 2D-DFT Based Feature Extraction

A simpler feature set relying on the information obtained from an image through frequency domain analysis is the 2D-DFT feature extraction method. The feature vectors, which ultimately are the inputs to the MLP used in this thesis, are extracted using this method. The 2D-DFT is an image processing tool which extracts the sine and cosine components of the image [79]. Any image, when finite in the spatial domain, has an infinite number of components in the frequency domain. The Fourier transform of an image gives a point corresponding to the frequency at that spatial region of the image. The discrete Fourier transform samples this point cloud and retains only a set of points. It, thus, does not contain the complete frequency information stored in an image. It, however, stores enough low frequency points, so as to faithfully represent the spatial information contained in the image [80].

For a given image $M \times N$, $f_d(m,n)$, the 2D-DFT gives the cosine and sine, i.e. real, $\boldsymbol{R}$, and imaginary, $\boldsymbol{I}$, components of the image, up to a limit L, as follows

$$R(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f_d(m,n) \cos\left(2\pi \left(\frac{mi}{M} + \frac{nj}{N}\right)\right) \tag{48}$$

$$I(i,j) = \sum_{m=0}^{M} \sum_{n=0}^{N} f_d(m,n) \sin\left(2\pi \left(\frac{mi}{M} + \frac{nj}{N}\right)\right) \tag{49}$$

The resulting vectors $\boldsymbol{R}$ and $\boldsymbol{I}$ are matrices, with dimensions $(2L+1) \times (L+1)$, consisting of the low frequency information contained in the image. This data, despite being compressed, still has redundant information. This is because, for a 2D signal, the information contained in the first and fourth quadrants of the Fourier domain are mirrors of the information contained in the second and third quadrants. As a result, only the information contained in the former is retained, while that in the latter is discarded. The remaining data is stored in the form of an array which is mapped to the input feature vector $\boldsymbol{x}$ for the classifier.

.

Chapter 4

Interpreting MLP Discriminants

As mentioned in previous chapters, the output discriminants of an MLP structure are approximate posterior probabilities [38], as described by the Bayes decision rule [60]. However, in practice, the outputs of an MLP do not map to real probabilities [81]. A histogram of MLP outputs for *8719* validation patterns, obtained from segmented and binarized license plate character images, and *36* output classes is shown in Figure 4-1. The largest discriminant corresponds to the estimated class, while the other values correspond to estimated incorrect classes.

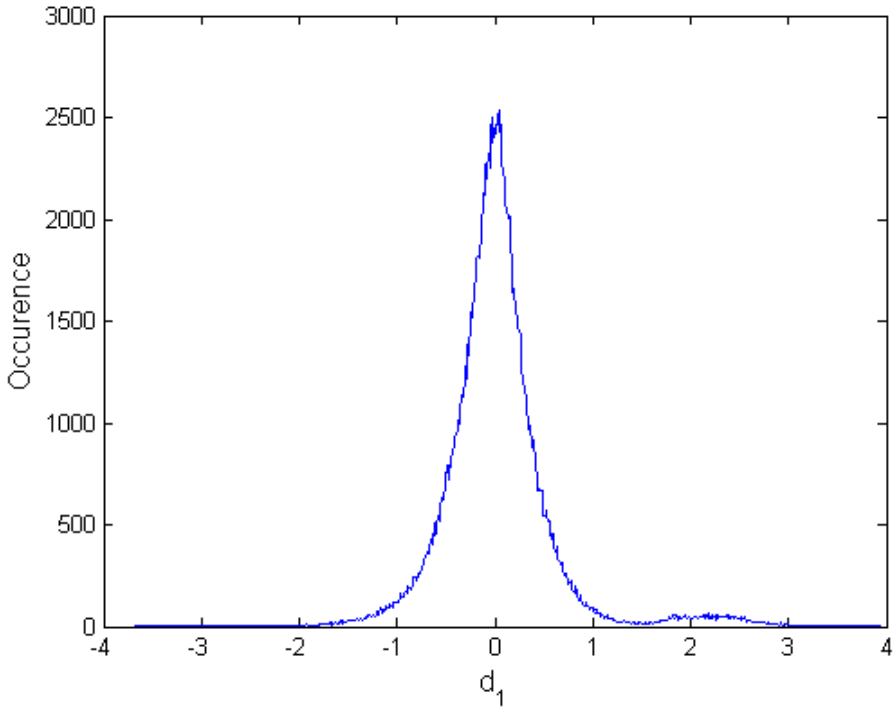

Figure 4-1 Histogram of discriminant values of an MLP

The output of the MLP, as evident, consists of both positive and negative values. By *lemma 2*, the output discriminant values of an MLP do however, approximately sum up

to 1, thus, partially buttressing the above claim. Considering the initial assumption that the MLP outputs are posterior probabilities, as discussed in **Error! Reference source not ound.**, it should be possible to remap the discriminants in a way to get the classifier to approximate posterior probabilities [81]. This chapter proposes an approach, which, based on a combination of data restructuring and non-linear mapping, attempts to warp the discriminants such that we derive a vector $\boldsymbol{d}_2$ which adheres to the following constraints

$$0 \leq d_2(i) \leq 1; \tag{50}$$

$$\sum_{1 \leq i \leq M} d_2(i) = 1; \tag{51}$$

$$d_2(i) \approx P(i|\boldsymbol{x}) \tag{52}$$

4.1 Discarding MLP Discriminants

As seen in **Error! Reference source not found.**, the output of an MLP is not just single value, which denotes the class to which the input has been assigned. It is a series of outputs, which should approximate the *M* posterior probabilities. As seen in (4), the correct class number is determined by the maximum element in the discriminant vector. The other discriminant values do not play a role in the final decision process of the MLP, but give an approximation of the validity of the other classes. Based on this knowledge, it is safe to assume that, for an MLP giving both positive and negative valued discriminants, there is no likelihood that the negative valued outputs corresponds to valid, correct classes. As mentioned in **Error! Reference source not found.**, by implementing OR, the iscriminants which earlier had negligible positive values are often assigned significantly negative values. This helps eliminate those discriminant values which are not required and these negatively valued outputs are, hence, equated to zero. This process gives us a new discriminant vector $\boldsymbol{d'}_1$.

$$d'_1(i) = \begin{cases} d_1(i), if\ d_1(i) \geq 0 \\ 0, otherwise \end{cases} \quad (53)$$

This step takes care of all negative valued discriminants, thus helping us achieve the lower bound of probability, i.e. *0*, seen in (50). But this raises a problem, since the outputs now consist of zeros and positively valued discriminants, the sum of the discriminants, which was initially *1*, now sum up to a much higher value, thus contradicting the upper bound of probability, i.e. *1*.

4.2 SoftMax

At this point, the discriminants, ***d'***$_1$, sum to a value greater than 1. As evident from Figure 4-1, the positive discriminant values vary over a large range.

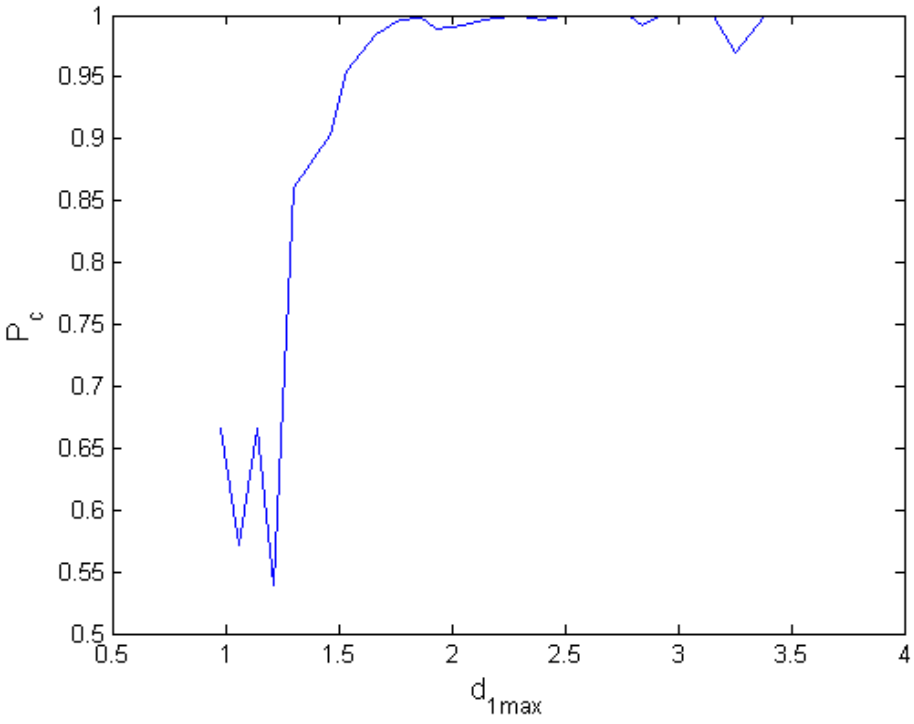

Figure 4-2 Plot of Pc v/s maximum discriminant values

From Figure 4-2, it is evident that the maximum value of the discriminant may take numerous values and the probability of correct classification for a majority of these maximum discriminant values is very high. A softmax operation is implemented to scale these values [82] so that they sum to *1*. The softmaxed discriminant vector ***S*** is described as follows

$$S(i) = \frac{d'_1(i)}{\sum_{j=1}^{M} d'_1(j)} \tag{54}$$

The elements of ***S*** now have values ranging between *0* and *1*, as described in (50), and the sum of the vector's elements is now *1*. Thus, a second criteria for the MLP output discriminants to equate to posterior probabilities is now achieved. A new plot of P_c versus the maximum softmax discriminant S_{max} in Figure 4-3 shows a smoother mapping, which now exists within the bounds of 0 and 1.

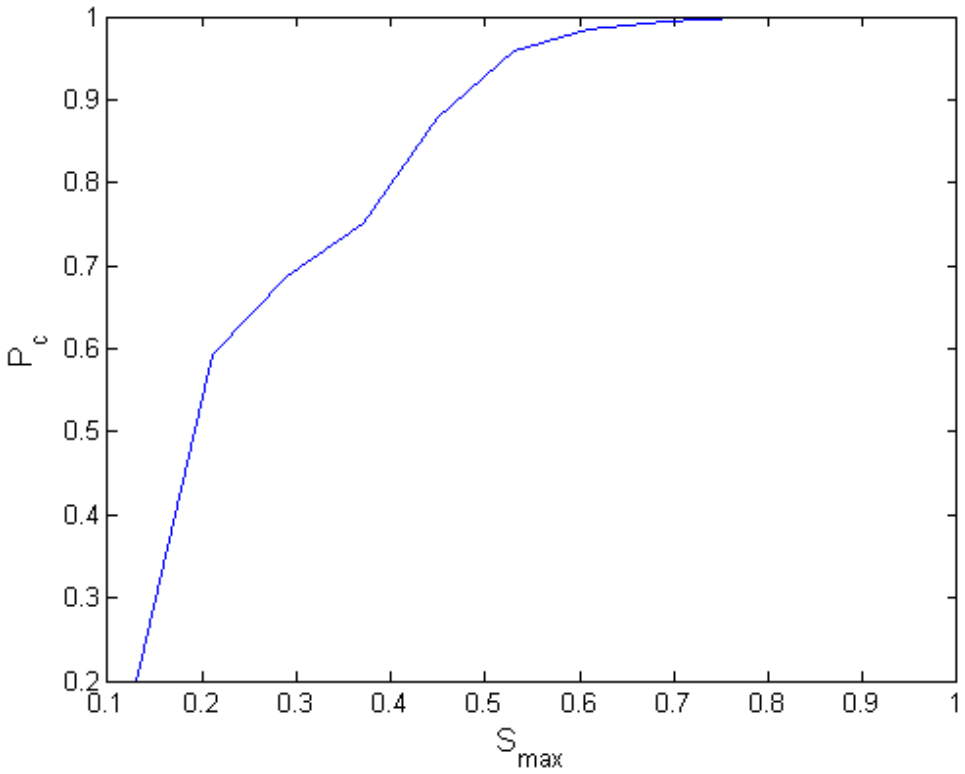

Figure 4-3 Plot of Pc after softmax v/s maximum discriminant values

Despite the softmax operation, the discriminant values do not, yet approximate posterior probabilities. As evident in Figure 4-3, numerous maximum discriminant values of the vector ***S***, have values in the mid-range of 0 to 1, but have higher actual posterior probabilities, *P (i|**x**)*. Ideally, if the MLP discriminants are posterior probabilities, the graph should be perfectly linear with a one to one mapping between the true posterior probabilities and S_{max}. Another detail of importance is that P_c is approximately *1* for the S_{max} discriminant values between *0.69* and *0.92*. This suggests that, given an appropriate non-linear mapping technique, the discriminants may be mapped to the posterior probabilities [81].

4.3 Proposed Mapping Scheme

The plot in Figure 4-3 shows that P_c versus S_{max} is not a straight line. A group of discriminants, with values in the upper mid-range of 0 to 1 have very high probabilities, while some discriminants with lower values also have sufficiently high probabilities of being correct. Any mapping scheme which is proposed, must take into account the non-linearity evident in Figure 4-3. The piecewise mapping proposed here uses a different mapping scheme for $S_{max} < S_t$ and for $S_{max} \geq S_t$, where S_t is assumed to be *0.68*.

The actual probabilities corresponding to S_{max} values greater than S_t lie between $P_{min} = 0.98$ and $P_{max} = 1$. For the current application, this range is fixed at *0.98* to *1*. The corresponding discriminant values must hence, map to values within this range. This mapping is performed by finding coefficients, *a* and *b*, which modify the value of the discriminants, making them fall within the range P_{min} to P_{max}. The coefficients are calculated by solving the equations

$$P_{min} = a + (S_t \cdot b) \tag{55}$$

$$P_{max} = a + (\max(S_{max}) \cdot b) \tag{56}$$

The term $max(S_{max})$ refers to the maximum observed value among the maximum softmaxed discriminant values S_{max} for all patterns. The new maximum discriminant values shall be referred to as d_{2max} and for S_{max} less than S_t, the value is given by,

$$d_{2max} = a + (S_{max} \cdot b) \tag{57}$$

This mapping scheme now yields values of d_{2max} which lie within the range of P_{min} to P_{max}, making the values approximate the posterior probabilities. Since this mapping procedure attempts to map values in a specified range, it cannot be used to map the discriminant values S_{max} which are lower than S_t. These discriminant values, however,

need to be mapped to values with P_{min} being the highest allowable value. A simple mapping for this region is proposed, where the square root of the value is multiplied by a constant, c. Thus, the new maximum discriminant value for $S_{max} < S_t$ is given by

$$d_{2max} = c \cdot \sqrt{S_{max}} \quad ; \quad c = \frac{P_{min}}{\sqrt{S_t}} \tag{58}$$

This mapping scheme warps the S_{max} values so that the P_c versus d_{2max} curve of Figure 4-4 better approximates a straight line.

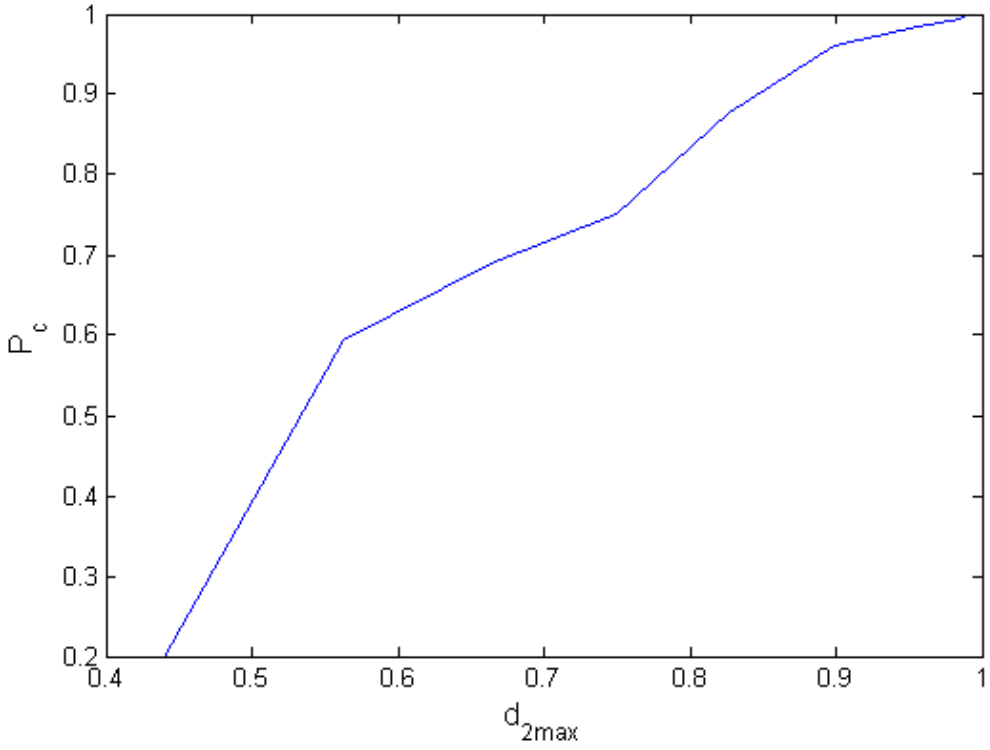

Figure 4-4 Plot of Pc v/s non-linearly mapped discriminant values

Between Figure 4-3 and Figure 4-4, the difference is evident. The new curve in Figure 4-4 is much closer to being linear. Therefore the maximum discriminants

approximate posterior probabilities more accurately. The remaining discriminants $d_2(i)$ which are less than d_{2max} are accordingly adjusted to approximate the posterior probabilities by multiplying the remaining discriminants with a factor *c* calculated as

$$c = \frac{1 - d_{2max}}{d}; d = \sum_{\substack{j=1; \\ j \neq i'_c}}^{M} S(j) \tag{59}$$

The adjusted discriminant vector $\boldsymbol{d_2}$ is given as

$$d_2(i) = c \cdot S(i); \;\; i \neq i'_c \tag{60}$$

This ensures that the sum of the discriminants in $\boldsymbol{d_2}$ is equal to *1*, thus mapping all discriminant values to the approximate posterior probabilities.

Chapter 5

Detecting Bad Inputs

The LPR character classifier is trained to correctly classify input feature vectors for valid characters. However, there may be cases where the characters obtained after segmentation are not valid, good characters. This may be caused by a non-robust license plate segmentation software, or by certain license plate artefacts which fuse with the valid characters and are not distinguishable, even to good plate segmentation software. **Error! eference source not found.**Figure 5-1 shows some examples of bad characters caused by faulty segmentation code.

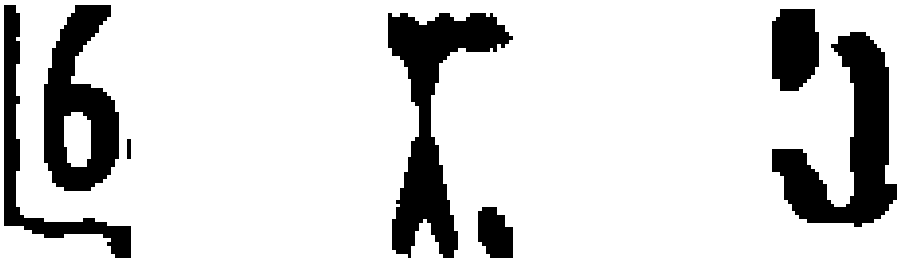

Figure 5-1 Samples of Bad Characters

Since the classifier treats all input feature vectors as valid and tries to place them into the correct class, it becomes necessary to detect and isolate these bad character cases. This chapter proposes a method to use the discriminant outputs $\boldsymbol{d}_1$ of an MLP classifier to detect such bad characters.

5.1 Effect of Bad Inputs on Output Discriminants

The output discriminants of an MLP classifier sum up to *1*. When a feature vector $\boldsymbol{x}$ from a valid, good character is input to the MLP, the discriminant vector $\boldsymbol{d}_1$ usually has only one large element and all other values are small, as seen in Figure 5-2. If a feature vector $\boldsymbol{x}$, corresponding to a bad character is input to the classifier, the discriminant vector $\boldsymbol{d}_1$ has multiple elements with values that are nearly the same magnitude, as seen in Figure

5-3, where two classes have very similar, high discriminant values and many other classes have moderately high values, in sharp contrast to Figure 5-2. This behavior of the discriminants can be exploited to detect probable bad characters in 0.

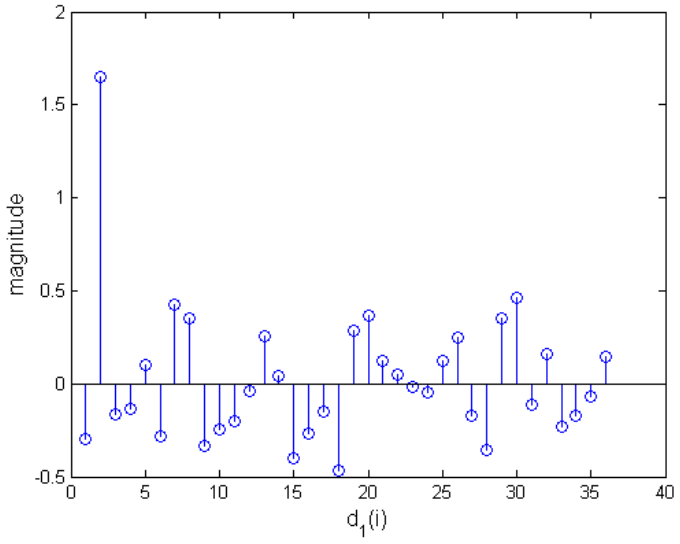

Figure 5-2 Plot of discriminants for a good character

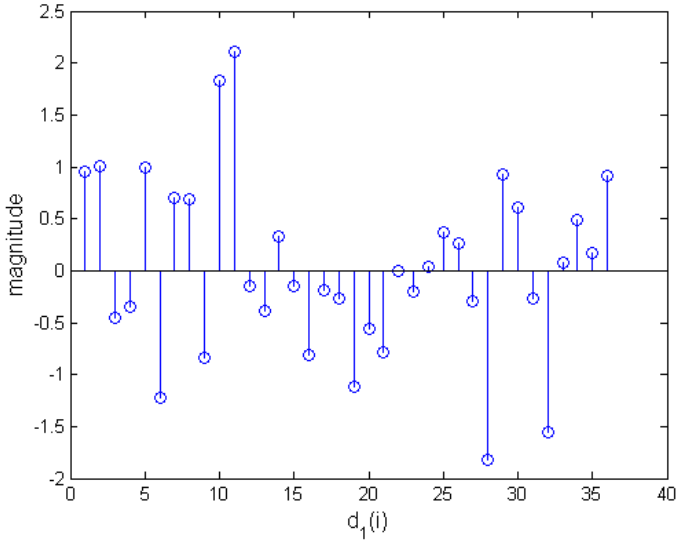

Figure 5-3 Plot of discriminants for a bad character

5.2 Using Discriminant Variances to Detect Bad Inputs

5.2.1 Single Variance Method

As seen in 0, there is an obvious contrast in the behavior of the discriminants $d_1(i)$, for good and bad characters. The proposed method exploits the differences in the variance of the discriminant vector to detect bad characters. In the single variance method, the variance of the discriminant vector excluding the maximum discriminant value $\boldsymbol{d}_{1m}$ is calculated as

$$V_1 = \frac{1}{M-1} \sum_{i=1; d_1(i) \neq \max(\boldsymbol{d}_1)}^{M} (d_1(i) - E(d_{1m}))^2 \tag{61}$$

where $E(d_{1m})$ is the expected value of the discriminant vector $\boldsymbol{d}_{1m}$ excluding the maximum discriminant value. Assuming each element of the discriminant vector $\boldsymbol{d}_{1m}$ is equiprobable, the expected value is given by

$$E(d_1) = \frac{1}{M-1} \sum_{i=1; i \neq i\prime_c}^{M-1} d_1(i) \tag{62}$$

If the value of the variance V_1 is greater than a certain threshold T_1, the character is labeled as a bad character. To decide the thresholds for the single variance method, the first step was to calculate the variance V_1 (61) for all the bad characters. The lower and upper bounds on the threshold T_1 were the minimum and maximum observed thresholds for the bad characters. For the current application, the performance of the algorithm has been compared over a range of thresholds T_1, starting from *0.07* up to *1.17* with increments of *0.1*. Table 2 shows the number of patterns each threshold removes, from the total of *8719* patterns of which *112* are bad characters.

The performance is measured on the basis of three main parameters, (i) the number of misclassified characters which are not removed is shown in Figure 5-4; (ii) the number of bad characters which are not removed is shown in Figure 5-5; (iii) the number of good characters which are incorrectly detected as bad characters is shown in Figure 5-6. The overall error percentage is the performance measure of this method, compared over the different thresholds and shown in Figure 5-7. From the results seen, a threshold of *0.27* is an optimum threshold, since it identifies a majority of the misclassified characters and the number of good characters it incorrectly identifies is relatively low.

Table 2 Number of Patterns removed and corresponding thresholds for Single Variance Method

Threshold (T_1)	**#Patterns Removed (Good and Bad)**
0.07	6283
0.17	1428
0.27	277
0.37	85
0.47	34
0.57	16
0.67	7
0.77	4
0.87	1
0.97	1
1.07	1
1.17	1

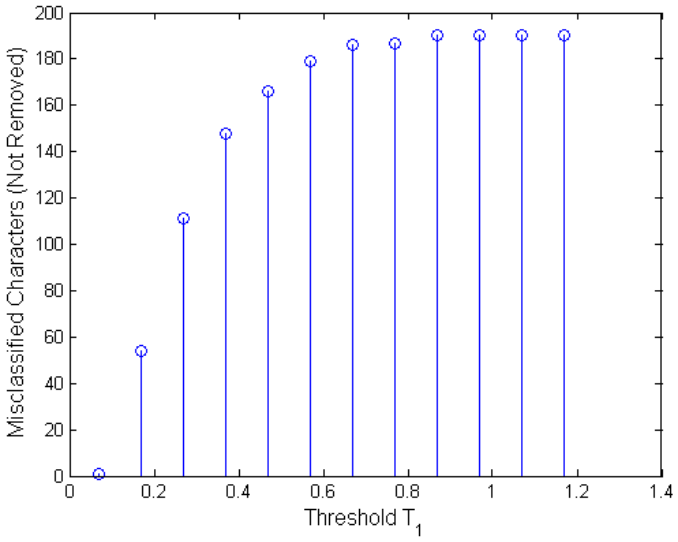

Figure 5-4 Plot of Number of Misclassified Characters (Not Removed) v/s threshold

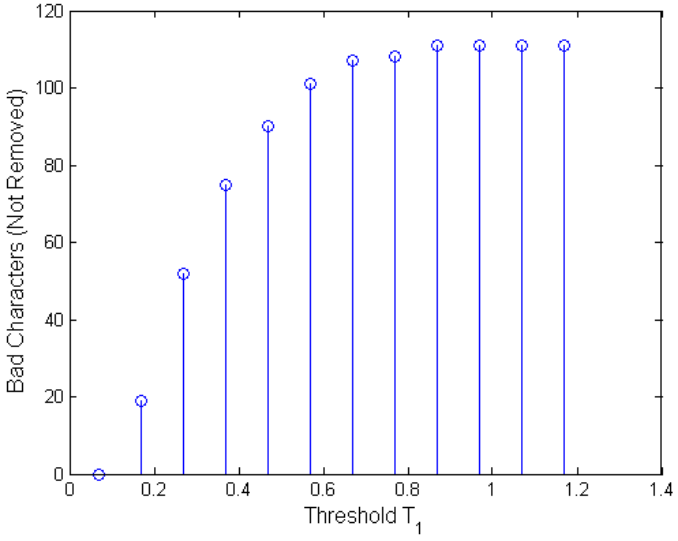

Figure 5-5 Plot of Number of Bad Characters (Not Removed) v/s threshold

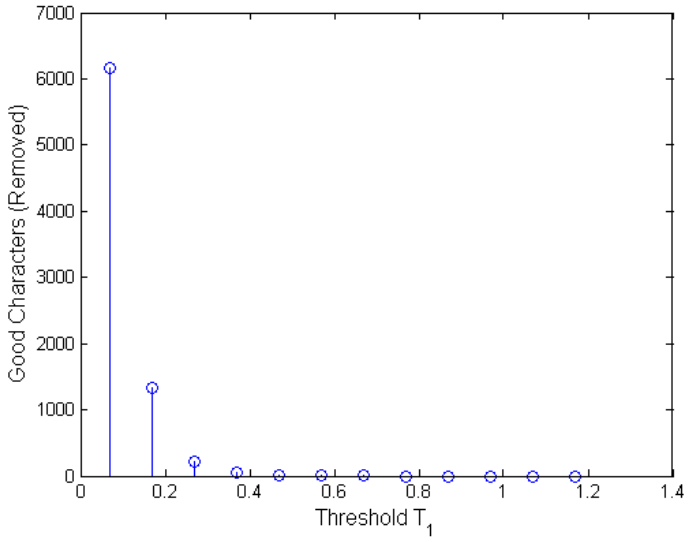

Figure 5-6 Plot of Number of Good Characters (Removed) v/s threshold

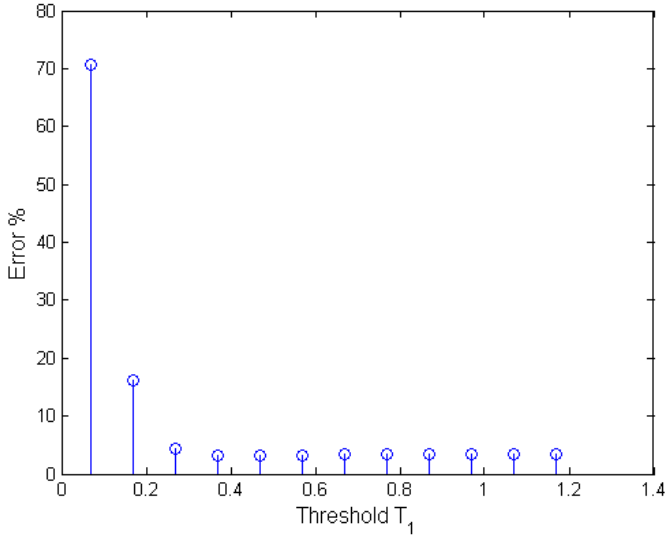

Figure 5-7 Plot of Pe v/s threshold for Single Variance method

5.2.1 Two Variance Method

A second proposed method is to take the variance of the discriminant vector twice and take the ratio of the two variances. The first step is to take the variance, V_1, of all the

discriminants at the output of the MLP, excluding the maximum discriminant, as seen in (61). Next, the variance, V_2, of all the discriminants are calculated, as seen in (63)

$$V_2 = \frac{1}{M}\sum_{i=1}^{M}(d_1(i) - E(d_1))^2 \quad (63)$$

where the term $E(d_1)$ is the expected value of the complete discriminant vector $\boldsymbol{d}_1$ given as

$$E(d_1) = \frac{1}{M}\sum_{i=1}^{M} d_1(i) \quad (64)$$

The ratio of V_2 to V_1 is calculated for the discriminant vector, $\boldsymbol{d}_1$, for every input of the classifier and a character is considered bad if this ratio is greater than a particular threshold T_2. The variances V_1 and V_2 (61) and (63), respectively, are calculated and their ratio is observed. The lower and upper bounds for the threshold T_2 are the minimum and maximum observed ratios for the bad characters. For the purposes of this application, a performance comparison is shown for thresholds, having values *0.4*, *0.5*, *0.6*, *0.7*, *0.8* and *0.9*. As in section **Error! Reference source not found.**, the performance is measured on he basis of three main parameters, (i) the number of misclassified characters which are not removed in Figure 5-8; (ii) the number of bad characters which are not removed in Figure 5-9; (iii) the number of good characters which are incorrectly detected as bad characters in Figure 5-10. The overall error percentage for this method is shown in Figure 5-11. Table 3 shows the number of patterns each threshold removes, from the total of *8719* patterns, of which *112* are bad characters. From the observed results, a threshold of *0.8* is optimal since it detects a majority of the misclassified and bad characters. The number of good characters it removes is only marginally higher than the higher thresholds.

Table 3 Number of Patterns removed and corresponding thresholds for Two Variance Method

Threshold (T_2)	#Patterns Removed (Good and Bad)
0.4	6023
0.5	3902
0.6	2104
0.7	851
0.8	272
0.9	44

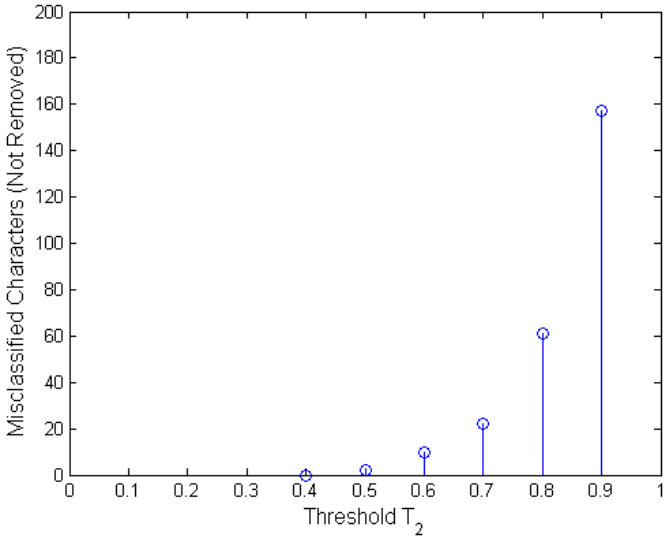

Figure 5-8 Plot of # Misclassified Characters (Not Removed) v/s threshold

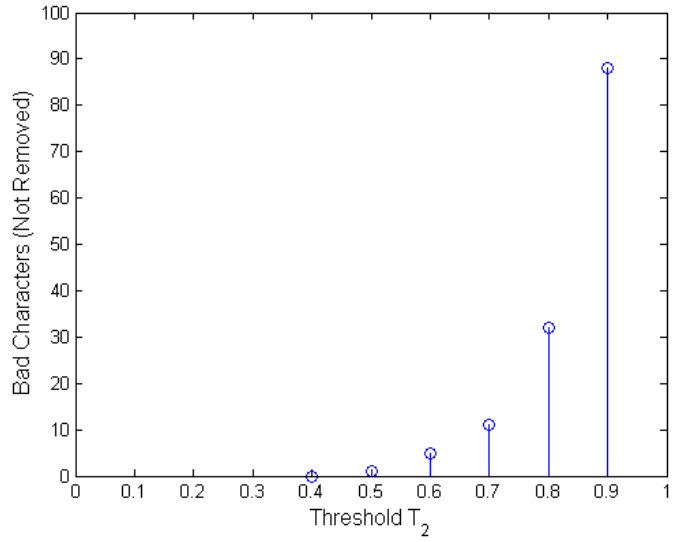

Figure 5-9 Plot of # Bad Characters (Not Removed) v/s threshold

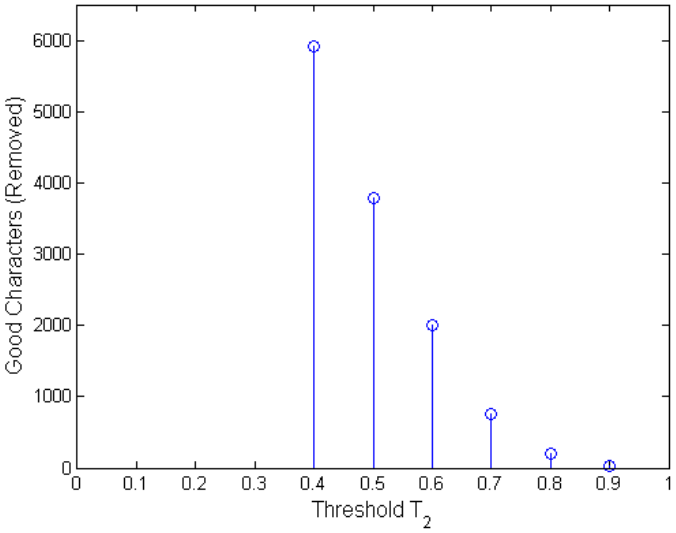

Figure 5-10 Plot of #Good Characters (Removed) v/s threshold

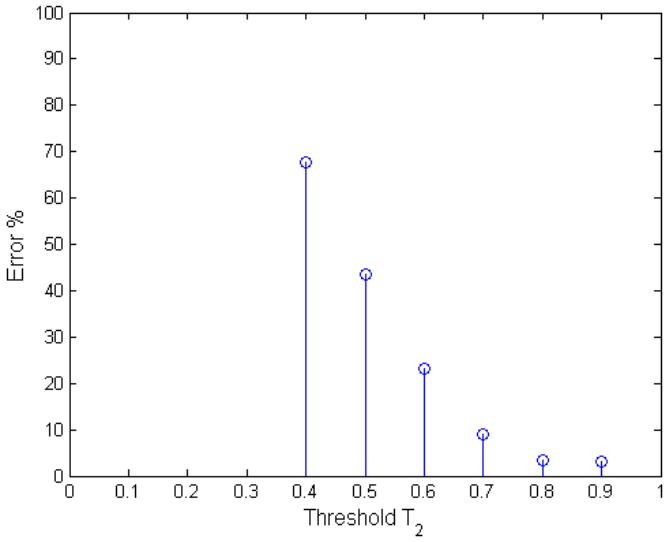

Figure 5-11 Plot of P_e v/s threshold for the two variance method

5.3 Comparison of Results

Table 4 Thresholds and Error Percentages for Single Variance Method

Threshold (T_1)	Error Percentage
0.07	70.78%
0.17	16.14%
0.27	4.35%
0.37	3.10%
0.47	3.07%
0.57	3.26%
0.67	3.38%
0.77	3.38%
0.87	3.45%
0.97	3.45%
1.07	3.45%
1.17	3.45%

Table 5 Thresholds and Error Percentages for Two Variance Method

Threshold (T_2)	Error Percentage
0.4	67.79%
0.5	43.51%
0.6	23.07%
0.7	8.98%
0.8	3.26%
0.9	3.03%

The error percentages for the different thresholds used in both the single variance case and the two variance case are shown in Table 4 and **Error! Reference source not found.** respectively. For the single variance case, the threshold values *0.27* and greater give the lowest error percentages, while for the two variance case, threshold values *0.8* and greater give low error percentages. However, if we take a look at Table 2, it is evident that the number of patterns actually detected by the algorithm is lower than those seen in Table 3. This implies that the two variance method, with a threshold of *0.8,* is better at detecting the bad characters as compared to the single variance method with a threshold

of *0.37*. The number of characters detected by the single variance method seen in Table *2* is far lower, compared to that of the two variance method, as seen in Table *3*. Hence, despite the similar low error percentages, the two variance method is better in terms of practical usability than the single variance method, since the number of bad patterns detected by the latter is greater than the former.

Chapter 6

Detection and Correction of Bad Inputs

Here we assume that all training and validation images are undamaged. The input vector is usually sufficient to classify a character image correctly. However, there are characters which look remarkably similar to each other. It is common to see a classifier misclassify these images. The final output decision of the MLP is based on the class with the maximum discriminant value. It is, however, possible to detect confused images by comparing the maximum discriminant values to the others. This chapter details a method to identify such misclassifications and rectify them.

6.1 The Confusion Matrix

As mentioned earlier, a character may be confused with a small number of other characters. This behavior is displayed using a confusion matrix [11], which is a detailed report of the performance [83] of the MLP classifier. The row numbers denote the correct class number for a given input vector and the column numbers denoted the predicted class number. During the validation stage, we have information of the correct and predicted classes for each input vector. By running a comparison of these two pieces of information, it is possible to generate a confusion matrix. For a set of validation patterns, the confusion matrix gives the number of times a character was classified correctly and the number of times it was incorrectly classified. It also gives the incorrect classes the input vector was assigned to.

Table 6 Confusion matrix for characters ‘4’; ‘6’; ‘A’; ‘W’; ‘X’

	Estimated														
Actual	Classes	0	4	5	6	7	A	D	G	K	M	N	V	W	X
	4	1	471	1	0	0	0	0	0	0	0	0	0	0	0
	6	0	0	3	424	0	0	0	4	1	0	0	0	0	0
	A	0	0	0	1	0	89	0	0	0	0	0	0	1	1
	W	0	0	0	0	0	0	1	0	0	0	2	1	133	0
	X	1	0	0	0	0	0	0	0	0	1	0	0	0	102

6.2 Detecting a Confusion between Multiple Classes

Examining the confusion matrix shown in Table 6, it is possible to identify the classes which may be confused. For example, *‘6’* is clearly being confused with *‘5’* and *‘G’*. Also, *‘W’* is being confused with *‘N’*.

To detect the possibility of a confusion, the first step is to find the discriminant vectors for the classes that are most likely to be confused. For a correctly classified character, the discriminants for the other classes should, ideally, be much lower than the maximum discriminant. However, for inputs where the classifier has difficulty correctly classifying the input character’s feature vector, it will give a higher discriminant value for at least one of the other possible classes. From the discriminant vector $\boldsymbol{d}_1$, the discriminant values corresponding to the classes which the estimated class is confused with are saved to a vector $\boldsymbol{d}_{1t}$. For a threshold T_3, if any of the discriminant values are greater than or equal to the product of T_3 and the maximum discriminant value, as seen in (65), the input vector is termed as a confused input and is put through further processing, discussed in section 0. $\boldsymbol{x}$ is a confused input vector if

$$T_3 \leq \frac{d_{1t}(i)}{d_{1max}} < 1, for\ any\ 'i' \tag{65}$$

The threshold T_3 is varied between *0.2* and *0.9* for the purpose of detecting the confused characters. It is assumed that for a confused case, there will be at least one

discriminant value in d_1' with a value which is some fraction of the maximum discriminant value. However, threshold values lower than *0.2* detect nearly all correct patterns and label them as confused patterns and hence, these lower thresholds have been ignored.

This assumption has been tested out for different threshold values and the performance of this algorithm is based on the number of confused patterns detected by each threshold and the number of correct patterns detected by the thresholds, as shown in Table *7*. The data set contains *8607* patterns with *56* patterns being the confusion cases.

Table 7 Performance comparison for different thresholds

Threshold (T_3)	**Number of Confused Patterns Detected**	**Number of Correct Patterns Detected (False Positives)**
0.2	42	1978
0.3	40	1211
0.4	36	713
0.5	34	352
0.6	31	138
0.7	30	56
0.8	23	19
0.9	11	8

6.3 Reclassifying Probable Misclassifications

One of the reasons for a misclassification is the large number of output classes in the original classifier. By reducing the number of output classes the network must train for, the accuracy of the network increases. The proposed solution for the case of confused characters involves the use of a second classifier, to reclassify the input feature vector. A cascaded classifier has been used in applications to detect pedestrians at road crossings [84]. In this case, the first classifier is a statistical learning classifier, to detect approximate ROIs and the second classifier is an SVM, used to locate pedestrians. Cascaded classifiers

have also been used in OCR systems [85], where principal component analysis is used to detect possible misclassifications, which are then reclassified using a second classifier.

For this thesis, the second classifier is also an MLP classifier and this second classifier is trained with only those classes which a particular class may be confused with. The classifier is trained on the same feature set as the primary classifier. As a result, the same input vector ***x*** is now put through a classifier with a significantly reduced number of classes, which improves its performance. If any of the characters classified correctly in the first classifier are passed to this stage, the characters are simply reclassified into the same correct class. This is because, by reducing the number of output classes, we are not adversely affecting the performance of the classifier for the correctly classified characters.

For illustrative purposes, a performance comparison is given for the group of possible confused characters, *'4'*, *'6'*, *'A'*, *'W'* and *'X'*. The comparison in performances for this collective group is presented in Table *8* and the corresponding error percentages are presented in Table *9* .

Table 8 Performance comparison for a group of confused characters in both stages

Character Group	**Number of Misclassifications (M = 36)**	**Number of Misclassifications (M = 5)**
'4'; '6'; 'A'; 'W'; 'X'	19	3

Table 9 Comparison of Error Percentage for each individual stage

Character Group	**Percentage Error (Single Classifier System)**	**Percentage Error (Two Classifier System)**
'4'; '6'; 'A'; 'W'; 'X'	1.534%	0.242%

6.4 Discussion

From Table *8*, we see that the second classifier drastically reduces the number of misclassifications due to confusion. This is because the second MLP deals with a fewer classes and hence, the weights are more optimized to classify these particular sets of inputs. Any correct characters which are put through this second classifier are also correctly classified, since the accuracy of the MLP does not decrease when the number of output classes is reduced. Hence, the overall effect of such a two classifier system is positive, since it greatly reduces the error percentage of the system. As seen in Table *9*, the error percentage for the given cluster of characters, in a single classifier system is *1.534%*, whereas, after the implementation of the proposed confusion detection algorithm, the error percentage drops to *0.242%*.

From Table *7*, it is evident that a threshold values of *0.7* and greater give an optimum results. They detect fewer confused characters than the thresholds lower than *0.7*, but the number of correct characters labeled as confused and passed on to the second classifier is also lower in the thresholds greater than *0.7* and this is the major performance metric to be considered here. Although the thresholds lower than *0.7* detect more confused characters, they also detect large numbers of correct characters too. As a result of this, the number of characters which must be processed by the second classifier increases drastically. This will adversely impact the overall speed of a practical LPR system. Hence, the optimum threshold is *0.7*.

Chapter 7

Conclusions and Future-Work

The methods to detect bad characters and confused characters; and the remapping scheme to approximate MLP outputs to Bayes posterior probabilities, proposed in this thesis have been validated using the application of license plate character recognition. But these concepts are not limited to a single application. Since the proposed methods are implemented solely on the discriminant outputs, they should work for many additional applications.

The non-linear remapping of the outputs is not a perfect algorithm, since the resultant plot is not an ideal straight line with a one-to-one mapping with the true probabilities. However, it is a good approximation of the probabilities for the current LPR application. Given a different application with a different set of input feature vectors, a more linear graph may be achieved.

The detection of bad characters using discriminant variances is a method whose robustness depends on the type of segmentation error generated by the preprocessing steps. For cases where an improper segmentation results in a shape resembling a character *'D'*, while the actual character on the license plate was a *'P'*, the algorithm may not detect the character as a bad one, simply because, the incorrect segmentation ends up producing a character that is valid in all aspects. Thus, there are certain limitations to the detection of bad characters which must still be explored in detail.

The method for detecting confused characters can be improved by increasing the number of patterns used for validation, effectively increasing the data contained in the confusion matrix. The two classifier system greatly improves the accuracy of the overall system, since it is trained to handle a smaller number of output classes. But if the classifier encounters a character which it confuses with a character not seen in the confusion matrix,

the misclassification may not be caught by the proposed algorithm, since the discriminant value corresponding to the estimated class will not be checked. A method to detect such cases must also be explored.

References

[1] A. K. Jain, R. P. W. Duin and J. Mao, ""Statistical Pattern Recognition: A Review"," *IEEE Transactions on Pattern Analysis and Machine Intelligenct,* vol. 1, no. 22, pp. 4-37, 2000.

[2] R. O. Duda, P. E. Hart and D. G. Stork, ""Pattern Recognition Systems"," in *Pattern Classification*, John Wiley & Sons, 2002, pp. 9-19.

[3] A. Colmenarez, B. Frey and T. S. Huang, ""Detection and Tracking of Faces and Facial Features"," in *International Conference on Image Processing*, 1999.

[4] P. Mountney, D. Stoyanov and G.-Z. Yang, ""Three-Dimensional Tissue Deformation Recovery and Tracking: Introducing Techniques based on Laproscopic or Endoscopic Images"," *IEEE Signal Procesing Magazine,* vol. 7, pp. 14-24, July 2010.

[5] L. Mihaylova, P. Brasnett, N. Canagarajah and D. Bull, ""Object Tracking by Particle Filtering Techniques in Video Sequences"," *Advances and Challenges in Multisensor Data and Information Processing,* vol. 8, pp. 260-268, 2007.

[6] G. J. Brostow, J. Shotton, J. Fauqueur and R. Cipolla, ""Segmentation and Recognition Using Structure from Motion Point Clouds"," *Computer Vision-ECCV 2008,* vol. 5302, pp. 44-57, 2008.

[7] L. Wang, S. Ma, H. Xue and Z. Hou, ""An Improved SIFT Algorithm for Feature Points Matching of Dairy Cow Images"," in *World Automation Congress (WAC), 2010*, 2010.

[8] J. Moonrinta, S. Chaivivatrakul, M. N. Dailey and M. Ekpanyapong, ""Fruit Detection, Tracking, and 3D Reconstruction for Crop Mapping and Yield Estimation"," in *11th International Conference on Control Automation Robotics & Vision (ICARCV)*, Singapore, 2010.

[9] N. Maglaveras, T. Stamkopoulos, K. Diamantaras, C. Pappas and M. Strintzis, ""ECG Pattern Recognition and Classification using Non-Linear Transformations and Neural Networks: A Review"," *International Journal of Medical Informatics,* vol. 52, no. 1, pp. 191-208, 1998.

[10] B. C. Subhash, S. Bagui, K. Pal and N. R. Pal, ""Breast Cancer Detection using Rank Nearest Neighbor Clasification Rules"," *Pattern Recognition,* vol. 36, no. 1, pp. 25-34, 2003.

[11] A. S. Godbole, ""Silicon Defect Recognition"," 2013.

[12] A. S. Dehdashti, J. R. Tudor and M. C. Smith, ""Forecasting of Hourly Load by Pattern Recognition: A Deterministic Approach"," *IEEE Transactions on Power Apparatus and Systems,* pp. 3290-3294, 1982.

[13] A. Singh, A. Sangwan and J. H. Hansen, ""Improved Parcel Sorting by Combining Automatic Speech and Character Recognition"," in *IEEE International Conference on Emerging Signal Processing Applications (ESPA)*, 2012.

[14] A. K. S. Alvaro, R. L. D. Dela Cruz, D. M. T. Fonseca and M. J. C. Samonte, ""Basic Handwriting Instructor For Kids Using OCR as an Evaluator"," in *International Conference on Networking and Information Technology*, 2010.

[15] K. Yamaguchi, Y. Nagaya, K. Ueda, H. Nemoto and M. Nakagawa, ""A Method for Identifying Specific Vehicles using Template Matching"," in *IEEE/IEEj/JSAI International Conference on Intelligent Transportation Systems, 1999*, 1999.

[16] P. Davies, N. Emmott and N. Ayland, ""License Plate Recognition Technology for Toll Violation Enforcement"," in *IEE Colloquium on Image Analysis for Transport Applications*, 1990.

[17] T. Sirthinaphong and K. Chamnongthai, ""The Recognition of Car License Plate for Automatic Parking System"," in *Proceedings of the Fifth International Symposium on Signal Processing and its Applications*, 1999.

[18] T. Yushuang, K.-H. Yap and Y. He, ""High Resolution Vehicle License Plate Reconstruction using Soft Recognition Learning"," in *8th International Conference on Information, Communications and Signal Processing (ICICS)*, 2011.

[19] H.-J. Lee, S.-Y. Chen and S.-Z. Wang, ""Extraction and Recognition of License Plates of Motorcycles and Vehicles on Highways"," in *Proceedings of the 17th International Conference on Pattern Recognition, 2004*, 2004.

[20] S. Theodoridis and K. Koutroumbas, "Pattern Recognition", Fourth ed., Academic Press, 2009.

[21] L.-G. Durand, P. D. Stein, M.-C. Grenier, J. Henry, R. Inderbitzen and D. W. Weiting, ""In Vitro and In Vivo Low Frequency Acoustic Analysis of Bjork-Shiley Convexo-Concave Heart Valve Opening Sounds"," in *IEEE Seventh Symposium on Computer-Based Medical Systems, 1994*, 1994.

[22] F. You, Y. Shi and P. Engler, ""Fingerprint Pattern Recognition for Medical Uses- A Frequency Domain Approach"," in *IEEE Nineteenth Annual Northeast Bioengineering Conference*, 1993.

[23] W.-Y. Ma and B. Manjunath, ""A Comparison of Wavelet Transform Features for Texture Image Annotation"," in *International Conference on Image Processing*, 1995.

[24] X. Wang, X. Ding and C. Liu, ""Gabor Filters-Based Feature Extraction Methods for Character Recognition"," in *Pattern Recognition*, 2005.

[25] R. Feris, M. Turk, R. Raskar, K.-H. Tan and G. Ohashi, ""Recognition of Isolated Fingerspelling Gestures using Depth Edges"," *Real-Time Vision for Human-Computer Interaction,* pp. 43-56, 2005.

[26] A. S. Ghotkar, R. Khatal, S. Khupase, S. Asati and M. Hadap, ""Hand Gesture Recognition for Indian Sign Language"," in *2012 International Conference on Computer Communication and Informatics (ICCCI)*, 2012.

[27] R. Bailey and M. Srinath, ""Orthogonal Moment Features for Use With Parametric and Non-Parametric Classifiers"," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 18, no. 4, pp. 389-299, 1996.

[28] G. Chetty and M. Lipton, ""Multimodal Feature Fusion for Video Forgery Detection"," in *13th Conference on Information Fusion (FUSION)*, 2010.

[29] S. Marsland, in *Machine Learning: An Algorithmic Perspective*, CRC Press, 2011, pp. 6-7.

[30] J. Laaksonen and E. Oja, ""Classification with Learning k-Nearest Neighbors"," in *IEEE International Conference on Neural Networks*, 1996.

[31] E. Bax, ""Validation of Nearest Neighbor CLassifiers"," *IEEE Transactions on Information Theory,* vol. 46, no. 7, pp. 2746-2752, 2000.

[32] L. Jiang, H. Zhang and J. Su, ""Learning k-Nearest Neighbor Naive Bayes For Ranking"," in *Advanced Data Mining and Applications*, Springer Berlin Heidelberg, 2005, pp. 175-185.

[33] A. F. M. Hani, H. A. Nugroho and H. Nugroho, ""Gaussian Bayes Classifier for Medical Diagnosis and Grading: Application to Diabetic Retinopathy"," in *IEEE EMBS Conference on Biomedical Engineering and Sciences(IECBES)*, 2010.

[34] L. Zhang and Y.-G. Xi, ""An Improved Multi-Class Algorithm for SVMs"," in *Proceedings of 2004 International Conference on Machine Learning and Cybernetics, 2004*, 2004.

[35] S. Haykin, "Neural Networks and Learning Machines", 3, Ed., Pearson Education, pp. 197-198.

[36] K. Hara and K. Nakayama, ""Comparison of Activation Functions in Multilayer Neural Network for Pattern Classification"," in *IEEE World Congress on Computational Intelligence*, 1994.

[37] J. Li and S. Sun, ""Nonlinear Combination of Multiple Kernels for Support Vector Machines"," in *20th International Conference on Pattern Recognition (ICPR)*, 2010.

[38] M. D. Richard and R. P. Lippmann, ""Neural Network Classifiers Estimate Bayesian A Posteriori Probabilities"," *Neural Computation,* vol. 3, no. 4, pp. 461-483, 1991.

[39] H. Rakha, B. Katz and A. Al-Kaisy, ""Field Evaluation of Weigh-In Motion Screening on Truck Weigh Station Operations"," in *IEEE Intelligent Vehicles Symposium, 2003*, 2003.

[40] H. Haiqi, M. Gu and H. Chao, ""An Efficient Method of License Plate Location in Natural-Scene Image"," in *Fifth International Conference on Fuzzy Systems and Knowledge Discovery*, 2008.

[41] D. Desrochers, Z. Qu and A. Saendeejing, ""OCR Readability Study and Algorithms for Testing Partially Damaged Characters"," in *International Symposium on Intelligent Multimedia, Video and Speech Processing*, 2001.

[42] M. D. Garris and C. L. Wilson, ""A Neural Approach to Concurrent Character Segmentation and Recognition"," in *Southcon Conference Record*, 1992.

[43] J. B. Broadwater and R. Chellappa, ""Adaptive Threshold Estimation via Extreme Value Theory"," *IEEE Transactions on Signal Processing,* vol. 58, no. 2, pp. 490-500, 2010.

[44] S. Chen and S. A. Billings, ""Neural Networks for Non-Linear Dynamic System Modelling and Identification"," in *Advances in Intelligent Control*, London, UK, Taylor and Francis, 1991, pp. 319-346.

[45] M. Forouzanfar, H. R. Dajani, V. Z. Groza, M. Bolic and S. Rajan, ""Comparison of Feed-Forward Neural Network Training Algorithms for Oscillometric Blood Pressure Estimation"," in *$th International Workshop on Soft Computing Applications*, Arad, Romania, 2010.

[46] R. Hecht-Nielsen, ""Theory of the Backpropagation Neural Network"," in *International Joint Conference on Neural Networks*, 1989.

[47] N. M. Nawi, R. S. Ranshing and M. R. Ransing, ""An Improved Conjugate Gradient Based Learning Algorithm for Back Propagation Neural Networks"," *International Journal of Computational Intelligence,* vol. 4, no. 1, pp. 46-55, 2007.

[48] H. Yu and B. Wilamowski, ""Neural Network Training with Second Order Algorithms"," in *Human-Computer Systems Interaction: Backgrounds and Applications 2*, 2012, pp. 463-476.

[49] M. T. Hagan and M. B. Mohammad, ""Training Feedforward Networks with the Marquardt Algorithm"," *IEEE Transactions on Neural Networks,* vol. 5, no. 6, pp. 989-993, 1994.

[50] M. D. Robinson and M. T. Manry, ""Two-Stage Second Order Training in Feedforward Neural Networks"," in *The Twenty-Sixth International FLAIRS Conference*, 2013.

[51] K. Tyagi, ""Second Order Training Algorithms for Radial Basis Function Neural Network"," 2011.

[52] M. I. A. Lourakis, ""A Brief Description of the Levenberg-Marquardt Algorithm Implemented by levmar"," institute of Computer Science-Foundation for Research and Technology, 2005.

[53] J. Li, M. T. Manry, L.-M. Liu, C. Yu and J. Wei, ""Iterative Improvement of Neural Classifiers"," in *FLAIRS*, 2004.

[54] D. W. Ruck, S. K. Rogers, M. Kabrisky, M. E. Oxley and B. W. Suter, ""The Multilayer Perceptron as an Approximation to a Bayes Optimal Discriminant Function"," *IEEE Transactions on Neural Networks,* vol. 1, no. 4, pp. 296-298, 1990.

[55] A. A. Abdurrab, M. T. Manry, J. Li, S. S. Malalur and R. G. Gore, ""A Piecewise Linear Network Classifier"," in *International Joint Conference on Neural Networks*, 2007.

[56] Y. Abu-Mostafa and J. St. Jacques, ""Information Capacity of the Hopfield Model"," *IEEE Transcations of Information Theory,* vol. 31, no. 4, pp. 461-464, 1985.

[57] P. L. Narasimha, M. T. Manry and F. Maldonado, ""Upper Bound on Pattern Storage in Feedforward Networks"," *Neurocomputing,* vol. 71, no. 16, pp. 3612-3616, 2008.

[58] S. A. Cannas, D. Stariolo and F. A. Tamarit, ""Learning Dynamics of Simple Perceptrons with Non-Extensive Cost Functions"," *Network: Computation in Neural Systems,* vol. 7, no. 1, pp. 141-149, 1996.

[59] C. Mazza, ""On the Storage Capacity of Non-Linear Neural Networks"," *Neural Networks,* vol. 10, no. 4, pp. 593-597, 1997.

[60] K. Hornik, ""Approximation Capabilities of Multilayer Feedforward Networks"," *Neural Networks,* vol. 4, no. 2, pp. 251-257, 1991.

[61] D. H. Wolpert and W. G. Macready, ""No Free Lunch Theorems for Optimization"," *IEEE Transcations on Evolutionary Computation,* vol. 1, no. 1, pp. 67-82, 1997.

[62] D. H. Wolpert and W. G. Macready, ""No Free Lunch Theorems for Search"," 1995.

[63] P. Domingos, ""A Few Useful Things to Know about Machine Learning"," *Communications of the ACM,* vol. 55, no. 10, pp. 78-87, 2012.

[64] D. H. Wolpert, ""The Lack of A Priori Distinctions Between Learning Applications"," *Neural Computation,* vol. 8, no. 7, pp. 1341-1390, 1996.

[65] T. Hastie, R. Tibshirani and J. Friedman, "The Elements of Statistical Learning", Springer.

[66] F. Faradji, A. H. Rezaie and M. Ziaratban, ""A Morphological- Based License Plate Location"," in *IEEE International Conference on Image Processing*, 2007.

[67] E. Haneda and C. A. Bouman, ""Text Segmentation for MRC Document Compression"," *IEEE Transactions on Image Processing,* vol. 20, no. 6, pp. 1161-1626, 2011.

[68] O. Due Trier, A. K. Jain and T. Taxt, ""Feature Extraction Methods for Character Segmentation and Recognition," 1996.

[69] T. D. Duan, D. A. Duc and T. L. H. Du, ""Combining Hough Transform and Contour Algorithm for Detecting Vehicles' License-Plates"," International Symposium on Intelligent Multimedia, Video and Speech Processing, 2004.

[70] S. Lyu and E. P. Simoncelli, ""Nonlinear Image Representation using Divisive Normalization"," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[71] W.-S. Kim and R.-H. Park, ""Color Image Palette Construction Based on the HSI Color System for Minimizing the Reconstruction Error"," in *InternationI Conference on Image Processing*, 1990.

[72] P. Viola and M. Jones, ""Robust Real-Time Object Detection"," *International Journal of Computer Vision",* vol. 4, 2001.

[73] W. Dingyun, Z. Lihong and L. Yingbo, ""A new algorithm for license plate recognition based on improved edge detection and Mathematical morphology"," 2nd International Conference on Information Science and Engineering, 2010.

[74] S. Ozbay and E. Ercelebi, ""Automatic Vehicle Identification by Plate Recognition"," *World Academy of Science, Engineering and Technology,* vol. 9, no. 41, pp. 222-225, 2005.

[75] Z. Jeffery, R. Soodamani and N. Bekooy, ""Real-Time DSP-Based License Plate Character Segmentation Algorithm using 2D Haar Wavelet Transform"," *Advances in Wavelet THeory and Their Applications in Engineering, Physics and Technology,* pp. 953-979, 2012.

[76] F. Kahraman, B. Kurt and M. Gokmen, ""License Plate Character Segmentation based on the Gabor Transform and Vector Quantization"," *Computer and Information Sciences-!SCIS 2003,* pp. 381-388, 2003.

[77] B. Shan, ""License Plate Character Segmentation and Recognition based on RBF Neural Network"," in *Second International Workshop on Education Technology and Computer Science (ETCS), 2010*, 2010.

[78] J. G. Daugman, ""Uncertainty Relation for Resolution in Space, Spatial Frequency and Oientation Optimized by Two-Dimensional Visual Cortical Filters"," *Optics and Image Science,* vol. 2, no. 7, pp. 1160-1169, 1985.

[79] R. C. Gonzalez and R. E. Woods, "Digital Image Processing: Third Edition", Prentice Hall, 2008.

[80] Y. Tao, V. Muthukkumarasamy, B. Verma and M. Blumenstein, ""A Texture Extraction Technique using 2D-DFT and Hamming Distance"," in *Fifth International Conference on Computational Intelligence and Multimedia Applications*, 2003.

[81] T. Kurita, H. Asoh and N. Otsu, ""Nonlinear Discriminant Features Constructed by Using Outputs of Multilayer Perceptron"," in *1994 International Symposium on SPeech, Image Processing and Neural Networks*, Hong Kong, 1994.

[82] C. Cadieu, M. Kouh, A. Pasupathy, C. E. Connor, M. Riesenhuber and T. Poggio, ""A Model of V4 Shape Selectivity and Invariance"," *Journal of Neurophysiology,* vol. 98, no. 3, pp. 1733-1750, 2007.

[83] L. Chen and H. Tang, ""Improved Computation of Beliefs based on Confusion Matrix for Combining Multiple Classifiers"," *Electronics Letters,* vol. 40, no. 4, pp. 238-239, 2004.

[84] Y. W. Xu, X. B. Cao, H. Qiao and R. Y. Wang, ""A Cascaded Classifier for Pedestrian Detection"," in *Intelligent Vehicles Symposium, 2006 IEEE*, 2006.

[85] J. Duong and H. Emptoz, ""Cascade Classifier: Design and Application to Digit Recognition"," in *Eighth International Conference on Document Analysis and Recognition*, 2005.

Biographical Information

Soumitro Auddy was born in Lucknow, Uttar Pradesh, India, in 1990. He completed his schooling in the Utpal Sanghvi School, Mumbai, Maharashtra, India. He pursued his Bachelor of Engineering degree in Electronics Engineering in the Mumbai University, India, and received his degree in 2011. He began his Masters in Electrical Engineering at the University of Texas at Arlington, USA, in 2011. He is currently pursuing his Master's thesis in the IPNNL at UTA. In the summe;r of 2013, he interned at Microsoft Corp. During his internship, he developed methods to detect screen defects caused by various manufacturing processes on different devices. After graduating, he will join Microsoft Corp. as a full time Hardware SDE.