Chapter 2. Literature Review

2.1 Overview.

In the literature, a pattern recognition problem is generally described as follows: Given an object *S*, and a set of class labels $\Omega = \{ \omega_i \mid i = 0,1,\ldots N\}$, assign the best label ω_i to *S*. It is assumed that ω_i to ω_N correspond to *N* known object classes, and ω_0 shows the "unknown" class. If the object *S* fits the criteria for class *i* better than the criteria for any other class, then ω_i is assigned to the object *S*.

In a pattern recognition system, the most important issues are:

- selection of a suitable color space,
- quantization of the color space,
- finding an appropriate classification method,
- acquisition of a model database of images [1].

These topics will be discussed in the following sections. The last two issues will be discussed in more detail because the main goal of this thesis is to obtain a better classification method. The first two issues will be discussed briefly citing the reference [1].

2.2 Color Spaces

Color is a perceptual phenomenon related to the human response to different wavelengths in the visible electromagnetic spectrum [11]. A small number of basis functions can perform good spectral approximations of most perceivable colors even though the number of basis functions needed to completely describe the full spectrum is infinite. Generally, a color is described as a weighted combination of three primary colors that form a natural basis [1]. There are many color spaces currently being used. The three color spaces most often used are *RGB*, *normalized RGB* and *HSI* spaces.

2.2.1 RGB Space

Red-green-blue (RGB) space is one of the most common color spaces representing each color as an axis. Most color display systems use separate red, green, and blue as light sources so that other colors can be represented by a weighted combination of these three components. The set of red, green, and blue can generate the greatest number of colors even though any other three colors can be combined in varying proportions to generate many different colors [12].

All colors that can be displayed are specified by the red, green, and blue components. One color is presented as one point in a three-dimensional space whose axes are the red, green, and blue colors. As a result, a cube can contain all possible colors. The RGB space and its corresponding color cube in this space can be seen in Figure 2.1. The origin represents black and the opposite vertex of the cube represents white.

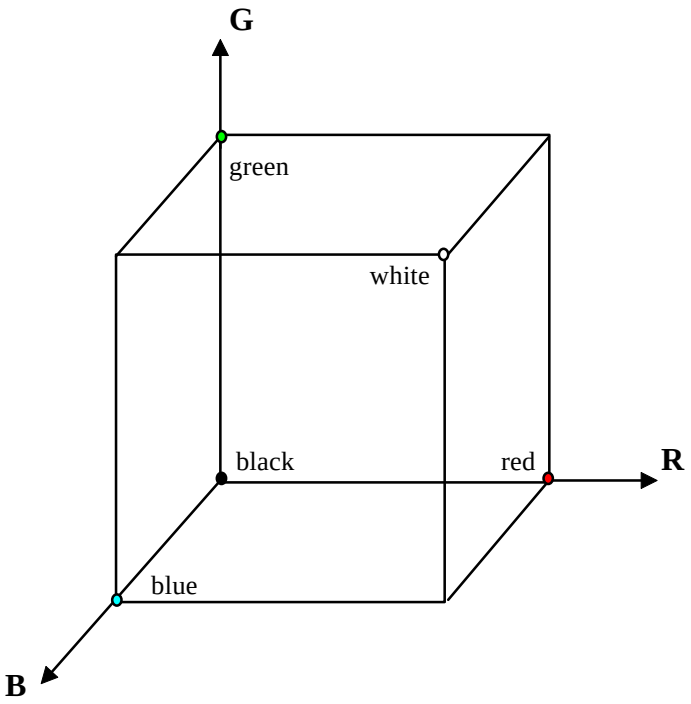

Figure 2.1. RGB color space and the color cube. Any color can be represented as a point in the color cube by (R, G, B). For example, red is (255, 0, 0), green is (0,255,0), and blue is (0, 0, 255).

The axes represent red, green, and blue with varying brightness. The diagonal from black to white corresponds different levels of gray. The magnitudes of the three components on this diagonal are equal.

The RGB space is discrete in computer applications. Generally, each dimension has 256 levels, numbered 0 to 255. In total, 256^3 different colors can be represented by (R,G,B), where R, G, and B are the magnitudes of the three elements, respectively. For example, black is shown as (0, 0, 0) while white is shown as (255, 255, 255).

2.2.2 Normalized RGB Space

Normalized RGB space is formed independently from varying lighting levels. The red, green, and blue components of normalized RGB space can be obtained from the three components of RGB space using the following formulation:

$$N_c \quad \frac{C}{(R \quad G \quad B)} \qquad \text{for } C \in \{R, G, B\} \text{ and } R + G + B \neq 0. \quad (2.1)$$

On the other hand, the components of the normalized RGB space are redundant because

$$N_r + N_g + N_b = 1. \quad (2.2)$$

The normalized RGB space is represented as:

$$Y = c_1R + c_2G + c_3B \quad (2.3)$$

$$T_1 \quad \frac{R}{(R \quad G \quad B)}$$

$$T_2 \quad \frac{G}{(R \quad G \quad B)} \quad (2.4), (2.5)$$

where c_1, c_2, and c_3 are constants chosen such that $c_1+c_2+c_3 = 1$ [13]. The luminance of the image pixel is represented by Y while T_1 and T_2 are chromatic and are independent of illumination to some extent [14].

2.2.3 HSI Space

Hue-saturation-intensity (HSI) space is also a popular color space because it is based on human color perception. Electromagnetic radiation in the range of wavelengths of about 400 to 700 nanometers is called visible light because the human visual system is sensitive to this range [12]. Hue is generally related to the wavelength of a light and intensity shows the amplitude of a light. Lastly, saturation is a component that measures the "colorfulness" in HSI space.

Color spaces can be transformed from one to another easily. A transformation from RGB to HSI can be formulated as below, although there are many alternative formulations [11], [13], [15].

$$\text{Intensity:} \quad I = R + G + B \quad (2.6)$$

$$\text{Saturation:} \quad S = 1 - \frac{3 \times \min(R, G, B)}{I} \quad (2.7)$$

Hue:
$$H = \begin{cases} 2\pi - \cos^{-1}\left\{\dfrac{[(R-G)+(R-B)]}{2\sqrt{(R-G)^2+(R-B)(G-B)}}\right\}, & B > G \\ \cos^{-1}\left\{\dfrac{[(R-G)+(R-B)]}{2\sqrt{(R-G)^2+(R-G)(G-B)}}\right\}, & \textit{otherwise} \end{cases} \quad (2.8)$$

HSI space can be considered as a cylinder as represented in Figure 2.2, where the coordinates r, θ, and z are saturation, hue, and intensity, respectively.

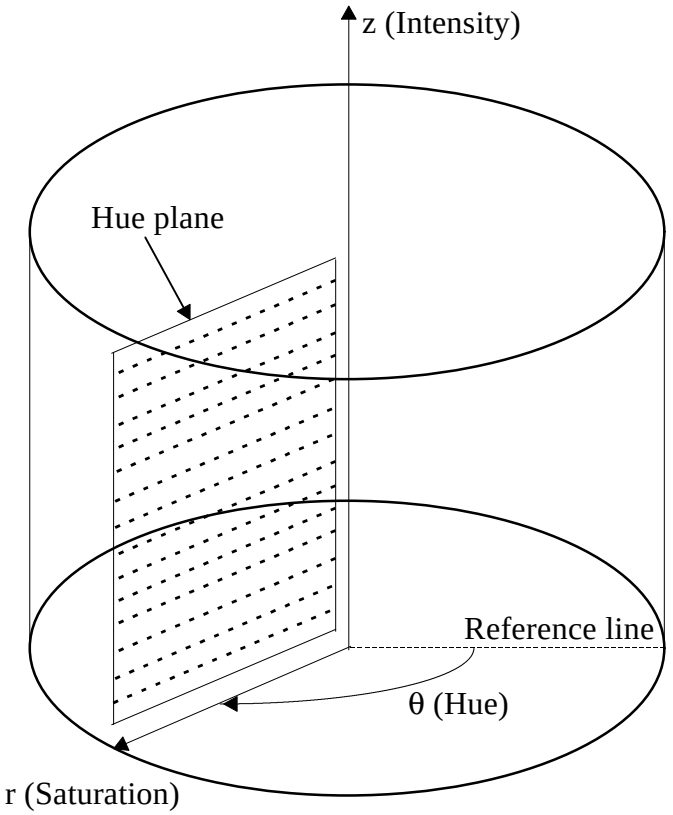

Figure 2.2. HSI color space, and the cylinder. The coordinates r, θ, and z represent saturation, hue, and intensity, respectively. The angle between the reference line and the hue plane is shown by . Hue plane is obtained by all colors that have the same angle,.

2.3 Color Quantization

Color spaces are generally large, such as 256^3 colors if a standard RGB camera is used. Having such a huge number of discrete colors is not practical if they are to be used for computing *histogram distance*. Color quantization is the term given to the effort of reducing the size of the color space by partitioning the original color space into larger cells. Since the set of colors that is obtained by the quantization process has an extraordinary impact on the performance of a color classification system, color quantization has been studied extensively.

Uniform quantization is the simplest method of color quantization. It is shown in Figure 2.3 for a two-dimensional case. Each color cell represents a cube whose edges are aligned with the color axes and all cells have the same size. The uniform quantization scheme can be written as:

$$C = \{(R_i, G_j, B_k) \mid i\,t \le R < (i+1)\cdot t,\ j\,t \le G < (j+1)\cdot t,\ k\,t \le B < (k+1)\cdot t;\ i, j, k = 0,1,.., n-1\}, \quad (2.9)$$

where (R, G, B) is the original color; C is the color space after quantization; (R_i, G_j, B_k) is a chosen color that corresponds to the (i, j, k)th cell; n is the number of quantization cells in each dimension and t is the size of quantization cells so that $n\,t = 256$. The total number of color cells is n^3 [1]. Since this thesis focuses on classification methods, uniform color quantization is used for the methods that use the *histograms* of the images, discussed in Section 2.4.1. The primary goal of this research is to find better classification methods, not colorquantization methods.

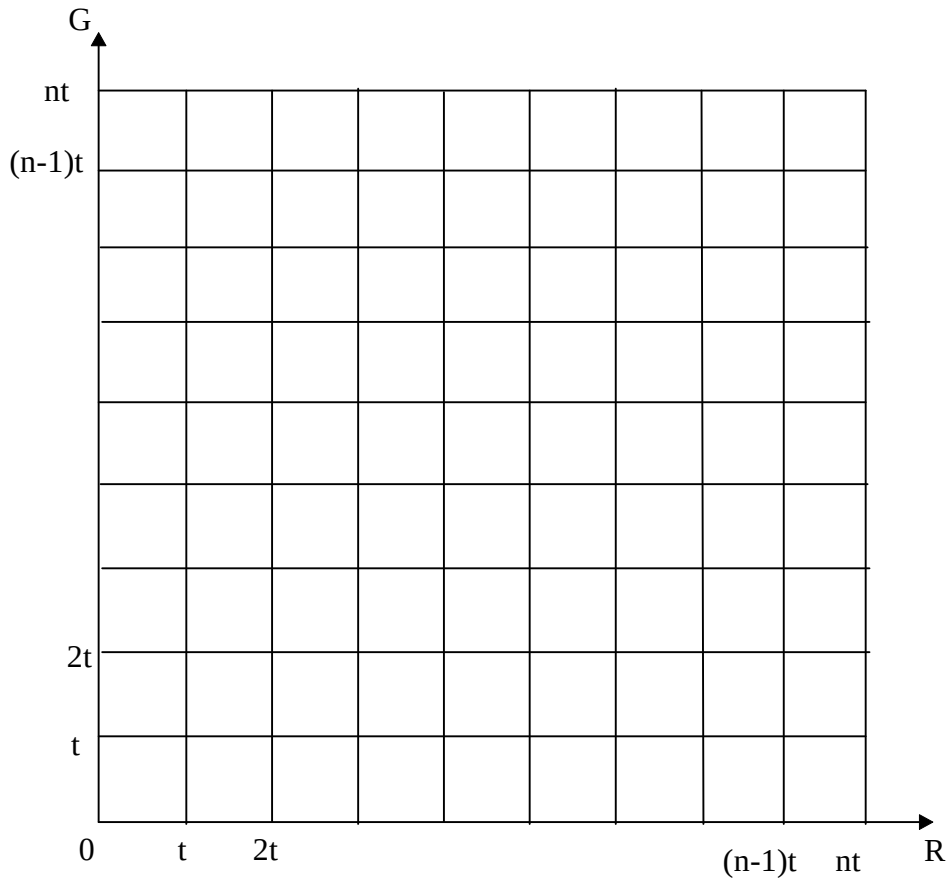

Figure 2.3. The representation of uniform quantization in two dimensions. Since this is a two dimensional case, color cells form squares having the same size. N is the number of quantization cells in each dimension. Therefore the total number of cells is n^2. The value t is calculated by $n\,t = 256$.

Nonuniform quantization gives improved performance in many systems [16], [17], [18], [19]. The color space is divided by using multiple thresholds on each color axis separately. The thresholds are obtained based on an analysis of the original distribution in the color space. In this case, the result is a set of rectangular boxes whose edges are aligned with the color axes. Nonuniform quantization in two dimensions can be seen in Figure 2.4. In [1], the mathematical interpretation of multiple threshold quantization is presented as:

$$C = \{(R_i, G_j, B_k) \mid Rt_i \quad R < Rt_{i+1}, Gt_j \quad G < Gt_{j+1}, Bt_k \quad B < Bt_{k+1}\ ; \quad i = 0,1,\ldots,r-1, \quad j = 0,1,\ldots,g-1, \ k = 0,1,\ldots,b-1\} \tag{2.10}$$

where (R, G, B) is the original color; C is the color space after quantization; Rt_i, Gt_j, and Bt_k are chosen thresholds, in which Rt_0, Gt_0, and Bt_0 are equal to 0 while Rt_r, Gt_g, and Bt_b are equal to 256. The number of cells in the three dimensions are not necessarily equal and are represented as r, g, and b. The triple (R_i, G_j, B_k) represents a chosen color that

represents the corresponding (*i*, *j*, *k*)th cell. In this method, the cells might not have the same size. The total number of cells can be calculated by $r \cdot g \cdot b$.

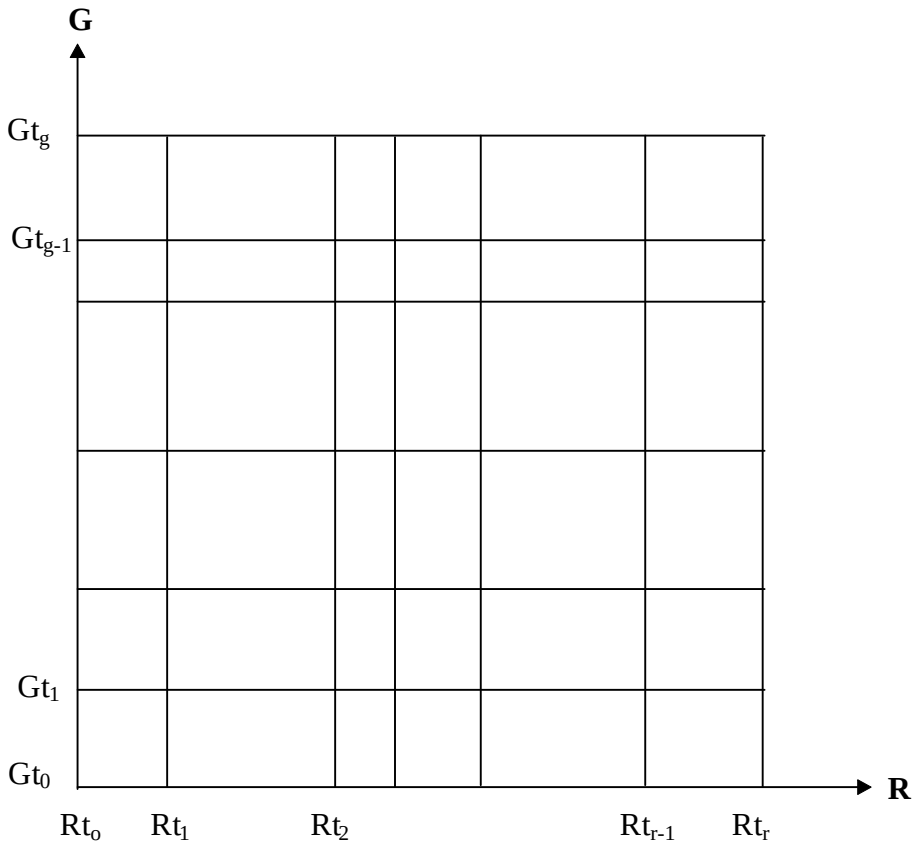

Figure 2.4. The representation of nonuniform quantization in two dimensions. The *r* and *g* are the numbers of quantization cells and they are not necessarily the same. The Rt_i and Gt_j are chosen thresholds. The minimum and maximum values that color axes can have are 0 and 256 respectively.

There are other methods [20], [21], [22] that use *clustering* and *vector quantization* methods to quantify the multispectral space. In these cases, the cells do not have to be rectangular. The resulting quantization is equivalent to the Voronoi tessellation of the color space according to class centers. The case of the two-dimensional color space can be seen in Figure 2.5.

Vector quantization is the process of mapping input vectors into a finite number of weight vectors. Additionally, it requires that a set of weighted vectors be found which can show the original multidimensional data with the least distortion. Clustering is the process of partitioning the input vectors. It is generally used in unsupervised systems to

find the “best” location for the class representatives in the multidimensional space [1]. In this work, *k*-means algorithm is employed as the clustering algorithm.

As stated in previous paragraphs, this thesis focuses on finding better classification methods instead of improving the clustering or color quantization algorithms. For this reason, methods that are presented in this thesis use the uniform quantization method to obtain the inputs of the radial basis function networks. As a clustering algorithm, the *k-means algorithm* is used. The *k*-means algorithm is used to find class centers and is explained extensively in Section 2.7.

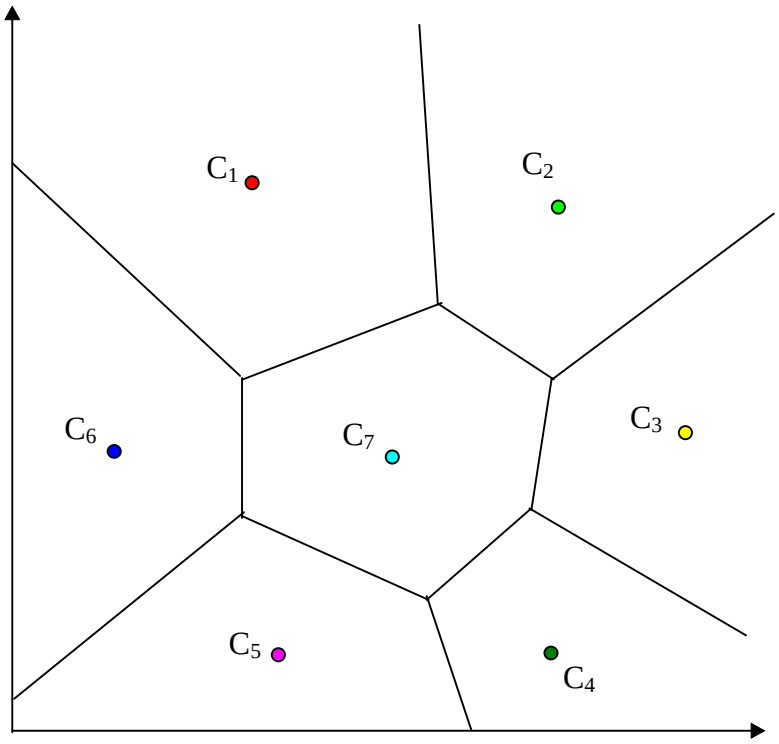

Figure 2.5. The representation of clustering or vector quantization in two dimensions. Each color cell represents one color.

2.4 Color Classification Methods

Color has been used as a crucial cue for object recognition tasks. These tasks consist of comparing a color image with a set of known color images to find the best match. Different descriptions and distance measurements can yield different results for a given pattern recognition task.

2.4.1 Extraction of Color Features

The most popular description of a color image is the centroid of the distribution of colors in a three-dimensional color space. In the RGB color space, the centroid of a color image is formulated as:

$$\begin{bmatrix} R_c \\ G_c \\ B_c \end{bmatrix} = \frac{1}{N} \sum_{i=1}^{N} \begin{bmatrix} R_i \\ G_i \\ B_i \end{bmatrix} \tag{2.11}$$

where N is the number of pixels of an image. Since this description is computationally simple, it is widely used for images with uniform texture. On the other hand, this description is not sufficient to describe an image with sharply varying texture.

Another frequently used description for an image is a *histogram*, which can provide information on the distribution of an image over a color space. Usually, a histogram has K bins which are called *cells*. The cells in a histogram do not overlap. The set of cell forms a partition of a color space. Each bin is valued by calculating the number of pixels that have the $[R, G, B]^T$ values associated with this cell. A histogram represents a k-dimensional feature of a color image, where K is the number of histogram bins. Detailed information about color features that are used in this thesis is presented in Chapter 3.

2.4.2 Classification Methods

The most commonly used classifier is the *minimum distance classifier* because it is very simple and efficient. It can be employed to classify an observed image for n unknown classes. Assume that a set of k-dimensional features $\{ \mathbf{x}_i \mid i = 1,2,...,n\}$ has been calculated from a set of training samples and stored as class models. The class models are created by choosing some of the training samples that represent a certain class best. These class models are used to identify the test samples by the following rule. An observed sample with the k-dimensional feature $\mathbf{y}$ is matched with the models, and if $d(\mathbf{x}_i, \mathbf{y}) < d(\mathbf{x}_j, \mathbf{y})$, $\forall\ i \neq j$, where d is the distance measure that is chosen, class label ω_i is assigned.

Although it is assumed that each class can be represented by one model, in most cases, a single model may not be enough to represent a class sufficiently. Each class can have more than one subsets that have their own models. If all subsets of a class are assigned a unique class label, the expression of the classification problem can be written as:

$S = \{ \omega_i \mid \min(d(\mathbf{x}_{ik,} \mathbf{y})) < \min (d(\mathbf{x}_{jk},\mathbf{y})) \; \forall \; i \neq j, \; k = 1,2,...,n_i \}$,

where $\mathbf{x}_{ik}$ is the kth model of the ith class, and n_i is the number of models of the ith class, and $\mathbf{y}$ is the test sample.

There are a number of frequently used distance measures in the literature. The most common distance measure is *Euclidean distance*. Let a k-dimensional feature be $\mathbf{x} = [x_1, x_2, ..., x_k]^T$ and a test sample be $\mathbf{y} = [y_1, y_2, ...,y_k]^T$. The Euclidean distance d_1 is expressed as:

$$d_1^2(\mathbf{x}, \mathbf{y}) = (\mathbf{x} - \mathbf{y})^T(\mathbf{x} - \mathbf{y}) = \sum_{i=1}^{k}(x_i - y_i)^2 \text{ , or} \tag{2.12}$$

$$d_1(\mathbf{x}, \mathbf{y}) = [(x_1 - y_1)^2 + (x_2 - y_2)^2 + ... + (x_k - y_k)^2]^{1/2}, \tag{2.13}$$

where x_i and y_i are the ith elements of $\mathbf{x}$ and $\mathbf{y}$ respectively.

The *city-block metric distance* is a simpler measure than Euclidean distance. The expression for the city-block metric distance is given as:

$$d_2(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{k}|x_i - y_i| \tag{2.14}$$

The k-dimensional feature is sometimes interpreted as a normalized form. In this case, a histogram approximates the probability distribution of the pixel values. Then, the *relative entropy* (Kullback Leibler distance) can be employed. The formulation for this distance measure is

$$d_3(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{k} y_i \log \frac{y_i}{x_i} \text{ , or} \tag{2.15}$$

$$d_3(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{k} y_i \log y_i - \sum_{i=1}^{k} y_i \log x_i \tag{2.16}$$

where $\sum_{i=1}^{k} x_i = \sum_{i=1}^{k} y_i = 1,$ and $x_i \geq 0, \;\; y_i \geq 0.$

Since this is not symmetric and does not satisfy the triangle inequality, it is not considered as a true metric. On the other hand, relative entropy is always positive and zero if and only if $\mathbf{x} = \mathbf{y}$. In many cases, relative entropy is considered as a distance between distributions [23].

A metric version of relative entropy can be written [24] as:

$$\begin{aligned} d_4(\mathbf{x}, \mathbf{y}) &= \sum_{i=1}^{k} y_i \log \frac{y_i}{x_i} + \sum_{i=1}^{k} x_i \log \frac{x_i}{y_i} \\ &= \sum_{i=1}^{k}(x_i - y_i)(\log x_i - \log y_i) \text{ ,} \end{aligned} \tag{2.17}$$

Since $d_4(\mathbf{x}, \mathbf{y}) = d_4(\mathbf{y}, \mathbf{x})$, this type of relative entropy is a symmetric distance measure.

2.5 Color Texture

Texture is one of the important characteristics to analyze many types of images. Although it is quite important and ubiquitous in image data, however, a precise definition or a formal approach of texture has not been given [25]. One of the problems confronted in this work is to accept a textured region, and decide to which of a finite number of classes the sample belongs [26]. The term "textured region" refers to a region defined by its texture.

Texture analysis is generally categorized into *statistical* methods and *structural* methods. Statistical methods use statistical results of the distribution of pixels and their relationship as the feature of a texture image. Although this method works well for many natural texture analyses, statistical methods are commonly based on gray level images and are not suitable to cope with multidimensional data simultaneously. Structural methods work on the assumption that a texture image consists of primitives that appear in repetitive spatial arrangements. The primitives and the placement rules must be described before describing the texture images [27].

Real visual textures are colored textures. They may have different colors, different structural patterns, or both [28]. In the computer vision literature, research has stated that color is one of the important properties that humans use in object recognition, particularly, pattern recognition in image texture [29]. In object recognition a great amount of effort has been taken to use the color and the texture features together instead of treating them separately.

An approach that encodes color, texture, and shape in a single spatio-chromatic feature space is proposed in [30]. It uses multi-scaled filtering and correlation methods to extract the features for object recognition. The color, texture, and shape are taken into account cooperatively in a systematic and interpretable fashion in this paper.

Another proposed approach is to use color histograms to represent multicolored objects [31]. Histograms are invariant to translation and rotation about the viewing axis and change only a little under change of the angle of the view, change in scale, and occlusion. A histogram backprojection algorithm is introduced in this paper. Histogram backprojection algorithm finds where the colors that belong to the object being looked for (the target). The algorithm uses color information instead of spatial information to solve the location estimation problem. In the Histogram backprojection algorithm a ratio R, defined as

$$R_i = \min\left(\frac{M_i}{I_i},\ 1\right) \tag{2.18}$$

is computed from the model (M_i) and image (I_i) histograms. It is this histogram R that is backprojected onto the image. That is, the image values are replaced by the values of R that they index [31].

There are also many other approaches that use color distribution as the cue to classify color objects [32], [33], [34]. These methods use color histograms calculated by a quantized color space and involve histogram similarity measures between test samples and model database.

Zhao examines both the color and the texture problem and uses color distribution as the cue to the classification of color and species. The color histograms are used to represent test samples. Color histograms are built on a quantized color space obtained in a training procedure using a novel color quantization method [1].

In this work, instead of color quantization, different learning and decision making mechanisms are studied. The color histograms are constructed on a quantized color space using a linear quantization method. This work differs from Zhao's work in the sense that different training methods, different clustering methods, and different classification methods are used. The *k*-means algorithm is employed because it is widely used and is an effective clustering algorithm [37], [38], [39], [40]. RBFNs are employed for the training method and classification method because of their nonlinear classification ability and speed of training. Some statistical methods are also involved in building the RBFN, such as calculating spreads of radial basis functions. Chapter 3 gives an intensive definition of RBFNs and their application on pattern the recognition problem.

2.6 Inductive Learning

Inductive learning is the process that finds general descriptions from object samples. The descriptions are generally abstract in nature although they can be temporal or spatial. Inductive learning consists of three main steps:

- Find subsets or clusters of the input objects.
- Categorize the subsets into classes.
- Find the class descriptions that cover the classes.

Unsupervised learning (learning from observation) systems contain all three steps. On the other hand, if only the last two are involved in the learning process, then the system is called *supervised learning* (learning from samples) [26].

In unsupervised learning, the training samples are not assigned with class labels at the beginning of the process. The system does not know which class each object belongs to as the process begins. Indeed, it may not know how many classes are assigned until a partition of the known objects are found. Unsupervised learning can be contemplated as a "data-driven" approach based on the observation of the input data [1].

Clustering is an often-used technique in unsupervised learning to generalize a subset of the objects. It is called "free" clustering if there is no information about the number of clusters in advance. If the number of clusters is determined in advance it is called "constraint" clustering [35].

Supervised learning uses the training examples by assigning a class label from a set of known classes at the beginning of the learning process. The main problem of supervised learning is to extract the characteristics of the objects, in order to achieve the proper descriptions of the objects, and to find a set of rules assigning each object into a class, including an *unknown* class that contains the objects that do not fit into a *known* class. Since the system or the algorithm knows what it is looking for, supervised learning can be contemplated as a "model-driven" approach. Zhao uses supervised learning in the system training procedure in which the training samples are used to extract the features of the classes and obtain the discrimination rules [1]. In this thesis, a supervised learning method is employed to determine the class centers and dilations for RBFs. Chapter 3 gives detailed information about the learning process of the RBFN. As a clustering algorithm, the *k*-means algorithm is used. Clustering is used as a part of the supervised learning.

2.7 The *K*-Means Algorithm

The *k-means algorithm* is commonly used in vector quantization (VQ) [42, 59] and clustering analysis [51, 55]. It is employed in many pattern recognition and computer vision applications [37, 54, 55, 58]. In the literature, the *k*-means algorithm is also known as the C-means algorithm, the generalized Lloyd algorithm (GLA), or the Linde-Buzo-Gray (LGB) algorithm [43 - 46]. The *k*-means algorithm is employed to define the centers of the radial basis functions in the radial basis function network Finding the centers of the RBFs is crucial because it makes the RBFN faster. The RBFN does not need any training for the centers of the RBFs since *k*-means algorithm provides centers of the classes from the training samples. The next section provides a background for *k*-means algorithm.

2.7.1 Background

The aim of the *k*-means algorithm is to divide M points in N dimensions into K clusters so that the within-cluster sum of squared errors is minimized [36]. It is not practical to require that the solution has minimal sum of squared errors against all partitions, except when M, N are small and $K = 2$. For bigger M and N, requiring the solution that has the minimal sum of squared errors causes a big increase in computational complexity. Instead of requiring the minimal sum of squared errors against all partitions, local optima are searched sufficient enough for bigger M and N. Local optima are the solutions such that no movement of a point from one cluster to another will reduce the within-cluster sum of squares. The algorithm needs as input a set of M points in N dimensions and a set of K initial cluster centers in N dimensions [47].

It is desired to partition the points so that points within clusters are close, in some sense, and points in different clusters are distant. The discordance between the data and a given partition $P(M, K)$ of M objects into K clusters is measured by an error $e[P(M, K)]$. The very large number of possible partitions makes it impractical to search through all for the minimum of e. That is why it is necessary instead to use the technique of local optimization. This technique begins with an initial partition. It then searches the set of partitions at each step, moving from a partition to that partition in its neigborhood for

which $e[P(M, K)]$ is a minimum. Hartigan declares that if the neighborhoods are very large, it is sometimes cheaper computationally to move to the first partition discovered in the neighborhood where $e[P(M, K)]$ is reduced from its present value [48].

A number of stopping rules are possible to end the algorithm. For example, the search stops when $e[(P, K)]$ is not reduced by a movement of a point from one cluster to another. The present partition is then locally optimal in that it is the best partition in its neighborhood. The following section gives mathematical details and explanations of the *k*-means algorithm.

2.7.2 Description of the *k*-means Algorithm

Before we give the mathematical equations this paragraph introduces some preliminaries. As mentioned before, there are M points in N dimensions to divide into K clusters. A *case* refers to a point in these M points in N dimensions. Each case has N variables. The ith case of the jth variable has value $A(i, j)$ $(1 \le i \le M, 1 \le j \le N)$. Let us consider the average values of red, green, and blue pixels over an image: r, g, and b. Each color can have a value between 0 and 255. Then an example of a case can be given by:

$$C_1 = \begin{bmatrix} 255 \\ 150 \\ 200 \end{bmatrix} \tag{2.19}$$

where red has the value of 255, green has the value of 150, and blue has the value of 200. In this case, $A(1, 1)$ is equal to 255, representing the 1st case (C_1) of the 1st variable (red). The partition $P(M, K)$ is composed of the clusters 1, 2, ..., K. Each of the M cases lies in just one of the K clusters. The mean of the jth variable over the cases in the lth cluster is denoted by $B(l, j)$. The number of cases in l is $N(l)$. The distance between the ith case and lth cluster is

$$D(i,l) = \sqrt{\sum_{j=1}^{N} \left[A(i, j) - B(l, j)\right]^2} \tag{2.20}$$

The error of the partition is

$$e[P(M,K)] = \sum_{i=1}^{M} D[i,l(i)]^2, \tag{2.21}$$

where $l(i)$ is the cluster containing the ith case. The general procedure is to search for a partition with small e by moving cases from one cluster to another. The search ends when no such movement reduces e. Hartigan gives a detailed description of the *k*-means algorithm step by step as follows:

> Step 1. Assume initial clusters 1,2, ..., K. Compute the cluster means $B(l, j)$ $(1 \le l \le K,\ 1 \le j \le N)$ and the initial error
>
> $$e[P(M,K)] = \sum_{i=1}^{M} D[i,l(i)]^2, \tag{2.22}$$
>
> where $D[i,l(i)]$ denotes the Euclidean distance between i and the cluster mean of the cluster containing i.
>
> Step 2. For the first case, compute for every cluster l

$$\Delta(l) = \frac{N(l)D(1,l)^2}{N(l)+1} - \frac{N[l(1)]D[1,l(1)]^2}{N[l(1)]-1}, \qquad (2.23)$$

where $\Delta(l)$ is the increase in error in transferring the first case from cluster $l(1)$, to which it belongs at present, to cluster l. If the minimum of this quantity over all $l \neq l(1)$ is negative, transfer the first case from cluster $l(1)$ to this minimal l, adjust the cluster means of $l(1)$ and the minimal l, and add the increase in error (which is negative) to $e[P(M, K)]$.

Step 3. Repeat Step 2 for the ith case $(2 \leq i \leq M)$.

Step 4. If no movement of a case from one cluster to another occurs for any case, stop. Otherwise, return to Step 2 [48].

2.7.3 Other Distances

The k-means algorithm searches for partitions $P(M, K)$ with a small error

$$e[P(M,K)] = \sum_{i=1}^{M} D^2[i, l(i)], \qquad (2.24)$$

where D is the Euclidean distance from i to the average object in $l(i)$, the cluster to which i belongs. The essential characteristics of the k-means algorithm are the search method, the changing of partitions by moving objects from one cluster to another, and the measure of distance [48].

To examine more general measures of distance, denote the ith case $\{A(i, j),\ 1 \leq j \leq N\}$ by $A(i)$, and let $B(l) = \{B(l, j),\ j = 1,2,....,N\}$ denote a set of mean values corresponding to the lth cluster. Let us define any distance, $F[A(i), A(k)]$, between the ith and kth cases. The *central case* of the lth cluster is denoted by the $B(l)$, minimizing $\sum_{i \in l} F[A(i), B(l)]$. The error of a particular partition is then

$$e[P(M,K)] = \sum_{i=1}^{M} F\{A(i), B[l(i)]\} \qquad (2.25)$$

where $l(i)$ is the cluster containing i.

Locally optimal clusters may be obtained by moving cases from one cluster to another, if this decreases e, and updating the central cases after each movement. For example, if the distance between two cases is the sum of the absolute deviations between the cases over their variables, then the central case of a cluster is the median for each variable. The contribution of a cluster to the error is the sum of absolute deviations from the median, over all objects in the cluster and over all variables [48].

Four different distance measures were discussed in Chapter 2 in detail. Every one of those distance measures can be employed in the k-means algorithm. Depending on which distance measure is used, the contribution to the error changes for each distance measure. In this thesis, the k-means algorithmm uses the Euclidean distance measure.

2.7.4 The Shape of the*k*-means Clusters

This section explains the geometrical meaning of the *k*-means algorithm. The *k*-means algorithm separates the clusters with hyperplanes as shown in Figure 2.5. The detailed mathematical explanations can be found in [48]. The following paragraphs summarize these mathematical explanations. One can visualize how the *k*-means algorithm works after reading this section.

Consider a partition that is locally optimal in the sense of equation (2.24) using the *k*-means search method. The cluster containing case i is represented by $l(i)$, and $D(i,l)$ is the distance between case i and the cases in cluster l. Let l_1 and l_2 be two different clusters. For each i in l_1, $D(i,l_1) < D(i,l_2)$, for otherwise i would be removed from l_1 during Step 2 of the algorithm. Therefore, each case i in l_1 satisfies the following conditions:

$$\sum_{j=1}^{N}[B(l_1, j) - B(l_2, j)]A(i, j) > c, \tag{2.26}$$

where

$$c = \sum_{j=1}^{N}\frac{1}{2}[B(l_1, j)^2 - B(l_2, j)^2], \tag{2.27}$$

and each case i in l_2 satisfies

$$\sum_{j=1}^{N}[B(l_1, j) - B(l_2, j)]A(i, j) < c. \tag{2.28}$$

Hartigan insists that the two conditions above prove that the cases in l_1 and l_2 are separated by a hyperplane normal to $\{B(l_1, j) - B(l_2, j)\}$ [48]. This hyperplane is a linear discriminant function for separating cases into the clusters l_1 and l_2. Figure 2.5 shows an example of the shape of the *k*-means clusters. He also states that each cluster is convex, which means that a case lies in a cluster if and only if it is a weighted average of cases in the cluster. The proof of this statement can be examined in [48].

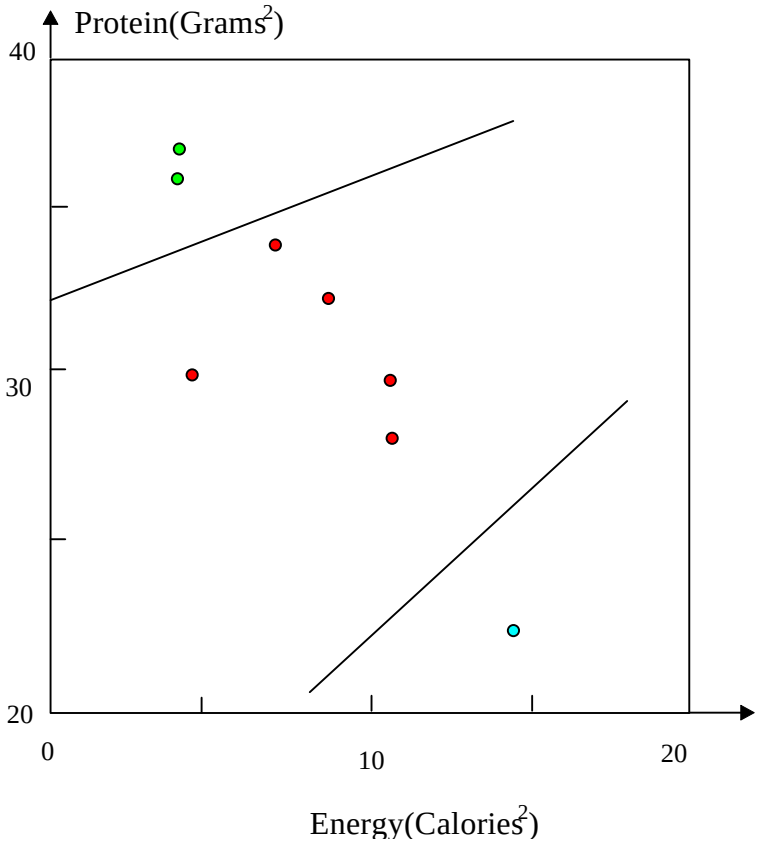

Figure 2.6. Clusters separated by hyperplanes based on the data in [48]. Each point represents a food item according to its protein weight and energy quantity.

2.7.5 Variations of the*k*-means Algorithm

There are a number of versions of *k*-means algorithm that need to be compared by sampling experiments or asymptotic analysis. The changeable components are the starting clusters, the *movement rule*, and the *updating rule* [48]. The movement rule is the rule that shows how the algorithm should move a point from one cluster to another. The updating rule defines how the algorithm recomputes the cluster means. The criteria for evaluation are the expected time of calculation and the expected difference between the local optimum and the global optimum. It is often necessary that an algorithm produce clusters that are independent of the input order of the cases. Although this requirement is not necessarily satisfied by the *k*-means algorithm, it can always be met by some initial reordering of the cases. For instance, if one reorders all cases by the first,

second, third,....., *N*th variables, in succession. This ordering will usually reduce the number of iterations in the algorithm.

Hartigan gives some starting options, movement options, and updating options in detail [48].

> The following are some starting options:
> (i) Choose the initial clusters at random. The algorithm is repeated several times from different random starting clusters with the hope that the spread of the local optima will give a hint about the likely value of the true global optimum. To justify this procedure, it is necessary to have a distribution theory, finite or asymptotic, connecting the local and global optima.
> (ii) Choose a single variable, divide it into *K* intervals of equal length, and let each cluster consist of the cases in the single interval. The single variable might be the average of all variables or weighted combination of variables that maximizes variance, the first roweigenvector.
> (iii) Let the starting for *K* be the final clusters for *K*-1, with that case furthest from its cluster mean split off to form a new cluster.
> The following are some movement options:
> (i) Run through the cases in order, assigning each case according to the cluster mean it is closest to.
> (ii) Find the case whose reassignment most decreases the within-cluster sum of squares and reassign it.
> (iii) For each cluster, begin with zero cases and assign every case to the cluster, at each step finding the case whose assignment to the cluster most decreases (or least increases) the within-cluster sum of squares. Then take the cluster to consist of those cases at the step where the criterion is a minimum. This procedure makes it possible to move from a partition which is locally optimal under the movement of single cases.
> The following are some updating options:
> (i) Recompute the cluster means after no further reassignment of cases decreases the criterion.
> (ii) Recompute the cluster means after each reassignment [48]

In the literature, there are a number of modified *k*-means algorithms in terms of the starting criteria, the movement rule, and the updating rule. The possibilistic *k*-means algorithm, the new possiblistic *k*-means algorithm, and fuzzy *k*-means algorithm are some of the latest modified*k*-means algorithms [37,38,39,40].

Krishnapuram and Keller give the possibilistic family of clustering algorithms which differ from the original algorithm in a sense that the membership of a point in a cluster is independent of all other clusters [38,39]. They use the term " possiblistic" simply because in their algorithm resulting partition and the membership values can be interpreted as degrees of possibility of the points belonging to the classes, i.e., the

compatibilities of the points with the class prototypes. Nasraoui and Krishnapuram introduce the new possibilistic C-means algorithm [40]. The new possibilistic C-means algorithm is more effective for noisy data than is the possibilistic C-means algorithm. It is, in fact, a modification of the C-means algorithm. One of the recent advanced *k*-means algorithms is introduced by Chinrungrueng, *et al.* [49]. Their algorithm approximates an optimal clustering solution with an efficient adaptive learning rate, which renders it usable even in situations where the statistics of the problem task vary slowly with time. This modification is based on the optimality criterion for the *k*-means partition stating that: all the regions in an optimal *k*-means partition have the same variations if the number of regions in the partition is large and the underlying distribution for generating input patterns is smooth.

The *k*-means algorithm is commonly used in RBFNs to obtain the RBFs' centers [50, 51, 52, 53]. In a RBFN, the problem of producing the centers of the RBFs is an important matter. A good set of centers can give very accurate and efficient results when compared to an ordinary set of centers [56]. The *k*-means algorithm has been a favorite method for researchers because it is fast, a more successful method obtaining optimal minima, and an excellent way of estimating the initial location of cluster centers [53, 57, 56].

The convergence properties of the *k*-means algorithm is studied by Tesauro, *et al.* [57]. Their results show that the *k*-means algorithm actually minimizes the sum of the squared error successfully. Keirnan, *et al.* indicates that RBFNs can be trained quickly once the centers have been initialized by the *k*-means algorithm [54]. Storer and Chon declare that a good estimation of the initial location of the cluster centers can be obtained through *k*-means clustering of a sample of the input data [56].

To summarize, the *k*-means algorithm is important for RBFNs due to the convergence properties of the algorithm. Additionally, knowing the class centers in advance makes RBFNs faster and more efficient. Otherwise, RBFNs need more RBFs to obtain the same results. This causes an increase in the computational complexity.