Image similarity based on color and texture

Introduction

Content-based image retrieval is a technique where images stored in a database are retrieved based on a query image. The algorithms look for similarity in the images based on certain characteristics such as color, texture and shape. The aim of this project was to develop a program in C#, investigate and develop techniques to achieve reasonable quality in searches, particularly for images of paintings. Texture and color similarity are performed on these images and a combination of these similarity results is used to retrieve similar images.

Algorithms

Texture can be defined as that where "there is a significant variation in intensity levels between nearby pixels; that is, at the limit of resolution, there is non-homogeneity" [1]. Principal component analysis (PCA) is a statistical technique used to reduce the dimension of the data set for classification problems. Since images are array of data points with each point representing color, PCA can be used for reducing the image data (extracting features) to smaller dimension to represent the image qualities [2]. The reduced feature represents the spatial distribution of the pixel gray values. In the application, the images are divided to smaller blocks (e.g. 32x32 pixel) and the features are extracted creating a projection using eigenvectors created using the entire sample. The distance between the features provides a basis for similarity of the image data. Computation of Eigen vectors for larger dataset are computationally expensive in terms of time and space. Incremental PCA computation is used to compute the eigen vectors when the database size is large [3].

Color is another property of the image that can be used for similarity. Similar to texture, the image data is reduced dimensionality based on color values of the pixels. The data is quantized in LAB color space using a discrete set of common color codes. The quantized image is split into regions each having a dominant color such that the pixels are homogeneous with respect to color within the region. The size of the region can vary depending on the color distribution. A distribution of the dominant color is generated (in 3% of total pixel intervals) and a similarity value is obtained by comparing the distance (in LAB space) [4].
Since color and texture are both important qualities of color images, a combination technique involving color and texture should give a better similarity search instead of just texture or color. This project attempts to consolidate the results of these comparison to provide a better similarity searches.

Program Description
The program is written in C# for Microsoft .NET platform. A sample screenshot is shown below:

There are 4 major sections on the application screen:

A. Query image (on left) and matching image (on right)
B. Search parameters and commands.

Commands:
Open File: opens the query image, can be one of JPG, BMP, GIF
Compare with Dir: runs the search against the pre-precessed directory of images. On completion, the search results are displayed in C in descending order of relevance.
Pre-process Dir: computes the projections of the images in the database with the pre-computed eigen vectors and stores them in files (with extension .ser, e.g db16.ser indicates values for 16x16 blocksize).
Compute EV: computes the eigen vectors of all blocks from all images in the database (and stores them in ev16.ser for 16x16 blocksize). This option can be used for small databases.
Compute EV incrementally: computes the eigen vectors for all blocks from all images but incrementally. Initial eigen vectors are calculated for the first image and then each block from
subsequent images are added incrementally. Additional files added to the folder will be 
processed selectively if selected again.

**Options:**

*Search Type - Color:* With this option, similarity search is done using Color property only. 
The EV calculation buttons are irrelevant for this case.

*Search Type - Texture:* Similarity search using Texture property only.

*Search Type - Combined:* When this option is selected, pre-process is done for both color 
and texture. Combination of these values can be used to vary importance of texture or color 
in the similarity search. The co-efficients can be changed by the user.

*Display Graph:* This check box toggles the display of graphs.

*Normalize:* When this option is checked, the values of texture and color are normalized.

**Settings:**

*Image Directory:* Directory where the database images are stored

*Rejection factor:* Percent of values rejected during Hausdorff distance calculation.

*Block Size:* Image block size (in pixels) that will be used for texture similarity

*No. of PCA components:* Dimensionality of PCA computation

*Distance:* Hausdorff distance calculation, Directed (both way) or symmetric (one way)

*Query Image (0-100%):* % of image that is used as query image. Eg. 50% specifies to use 
only the middle 50% portion of the image

*Distance Zone (0-1):* If a value of 0.2 is specified, the distance computation is done using 
points within a zone around it that is 20% of the mean of height and width of the image.

*Alpha (incrPCA):* $\alpha$ in incremental PCA algorithm [3].

C. The first 50 matching images are displayed with the most relevant as the first. The similarity 
values are displayed with each image. The first value is for Color, followed by Texture and combined.

D. The graphs give a better view of the similarity among the various images in the database. The X 
axis is the image no. and Y axis is the similarity value.

To start using the program:

1. Copy all images to a folder.
2. Start program, specify the directory in the program.
3. Select Combined search type (if you will use both color and texture)
4. Compute EV incrementally (if large database, else Compute EV)
5. Run Pre-Process directory
6. Select query image
7. Run Compare with Dir
Conclusions:
It has been shown that the combination of texture and color can be used to give reasonable results in image similarity. The incremental PCA eigen vector calculation works well for large database sizes, and was tested for image databases of around 1000 images. For this collection of images, reasonable search results resulted in most cases. Future work can focus on techniques to improve the speed of the searches and testing other variations of these algorithms.

References:
Sample Results from the 1000 image database: