**EECS**

**University of Tennessee**

**Pattern Recognition – ECE 571**

### Project 4 – Color Image Compression Using Unsupervised Learning (Clustering)

**Submitted by:**

Mahendra Duwal Shrestha

[mduwalsh@vols.utk.edu](mailto:mduwalsh@vols.utk.edu)

**Abstract**

Since many display devices support only handful number of colors to be displayed on, true color images may need to be compressed within limited number of colors to save storage and to display. In this project, image of 480x480 pixel of true color was compressed to display with 256, 128, 64 and 32.

The objective of this project was to implement clustering algorithms like k-means, winner takes all and Kohonen map to represent image using only limited number of colors. Also the performance of each implementation was measured using difference in original image and pseudo color image generated using clustering algorithms.

The dataset of 480x480 full color image was provided. It was treated as matrix of three dimensions with RGB values in each pixel. The image was read using readImage() method and introduced to clustering algorithms to represent each pixel with a cluster center in presence of limited number of clusters. Thus, the image was compressed and displayed using 256, 128, 64 and 32 colors and difference in original and pseudo image were displayed for each number of color representation.

**Introduction**

In machine learning, the problem of unsupervisedlearning is that of trying to find hidden structure in unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning [1]. Clustering is one of the methods of unsupervised learning.

Since many display devices support only handful number of colors to be displayed on, true color images may need to be compressed within limited number of colors to save storage and to display. In this project, image of 480x480 pixel of true color was compressed using clustering algorithms like k-means, winner takes all and Kohonen map and the image was displayed with limited number of colors (256, 128, 64 and 32).

.

**Technical Approach**

**k-means**

k-means is one of the clustering algorithms which assumes there are k clusters and starts by initializing cluster centers to arbitrary values.

Step 1: Begin with an arbitrary set of cluster centers and assign samples to nearest clusters

Step 2: Calculate sample mean of each cluster using samples in each cluster

Step 3: Reassign each sample to the cluster with the nearest cluster mean

Step 4: If assignment of sample to cluster does not change cluster, then stop; else repeat from step 2

**Winner takes all**

Winner takes all is improvement in k-means algorithm to increase speed of convergence. The winner (cluster center which is nearest to a sample) is updated on the fly immediately after a sample is classified to the cluster by pulling cluster towards the sample.

Step 1: Begin with an arbitrary set of cluster centers *i*

Step 2: Calculate sample mean of each cluster using samples in each cluster

Step 3: For each sample **x**, find the nearest cluster center , which is called the winner and assign the sample to the cluster

Step 4: Modify  using new = old + (**x**- old) where is learning parameter and typically small value in order of 0.001

Step 5: If assignment of sample to cluster does not change cluster, then stop; else repeat from step 2

**Kohonen Feature Map**

It is an extension of the winner-take-all algorithm and is also called self-organizing feature maps. In this algorithm, a problem dependent topological distance is assumed to exist between each pair of cluster centers. When the winner cluster center is updated so are other cluster centers as its neighbors in the sense of topological distance.

The winning cluster center and its neighbors are trained based on the following formula:

are cluster centers

As k increases, decreases. (However, for this project it is chosen small value constant)

are coordinates of cluster centers

are coordinates of winning cluster center

The last expression tells that closer the neighbors are in topology, more affected they are.

**Experiments and Results**

**Original image**

****

**k-means**

k-means algorithm was implemented for the image data set to represent image using 256, 128, 64 and 32 colors. These were made configurable from command line arguments while executing program. The resulting image and difference image are shown below.

256 color representation: in 181 epochs

****

128 color representation: in 77 epochs



64 color representation: 59 epochs



32 color representation: 40 epochs of iteration



**Winner takes all**

Winner takes all algorithm was implemented for the image data set to represent image using 256, 128, 64 and 32 colors. The learning rate used was 0.0001. The resulting image and difference image are shown as below:

256 color representation: in 125 epochs



128 color representation: in 92 epochs



64 color representation: in 51epochs



32 color representation: in 43 epochs



**Kohonen map**

Kohonen map algorithm was implemented for the image data set to represent image using 256, 128, 64 and 32 colors. The learning rate used was 0.00001. Topological space of 2 dimension of 64x4 was assumed. The resulting image and difference image are shown as below:

256 color representation: in 200 epochs



128 color representation: in 200 epochs



64 color representation: in 128 epochs



32 color representation: in 20 epochs



**Discussion**

In this project, we’ve implemented k-means algorithm, winner takes all algorithm and kohonen map algorithm. So from this project we learned how to implement clustering in unsupervised learning. Moreover, we also learned difference in representing image with higher number of colors and lower number of colors. From this project, it can be said that higher number of color gives better representation of color image. Winner takes all improves convergence speed by lowering number of epoch iterations due to online updating or winner cluster center. Also kohonen map did reduce convergence faster than winner takes all but convergence is not guaranteed and it highly depends on cluster initialization and parameter values.

**References**

1. <http://en.wikipedia.org/wiki/Unsupervised_learning>

2. Lectures notes from class ECE 471/571 Pattern Recognition (Prof. Qi)

**Appendix**

**/lib/kmeans.cpp**

#include "Matrix.h"

#include "Pr.h"

#include <iostream>

#include <cstdlib>

#include <cmath>

using namespace std;

Matrix kmeans(Matrix &S, Matrix &C, int max\_val)

{

int nrS=S.getRow();

int ncS=S.getCol();

int nrC=C.getRow();

int ncC=C.getCol();

int d=0;

Matrix dist(1,nrC);

Matrix sdist(1,nrC);

int pos;

double minD;

Matrix lookup(nrS,1); // lookup table for sample belonging to which cluster

Matrix count(nrC,1); // to store count for each cluster center

Matrix sum(nrC,3);

// initialize cluster means

for(int i=0;i<nrC;i++)

for(int j=0;j<ncC;j++)

{C(i,j)=rand()%(max\_val+1);}

//Matrix x1,y1; // for storing points to be used calculate distance

for(int i=0;i<nrS;i++)

{

lookup(i,0)=0;

}

bool change;

// while change of assignment to centers occurs

do{

cout<<d++<<endl;

change=false;

// clear values in count and cluster mean

for(int i=0;i<nrC;i++)

{

for(int j=0;j<ncC;j++)

{

sum(i,j)=0;

count(i,0)=0;

}

}

for(int i=0;i<nrS;i++)

{

// calculate distance between sample data and each cluster mean

minD=0;

pos=0;

for(int k=0;k<ncC;k++)

minD+=(S(i,k)-C(0,k))\*(S(i,k)-C(0,k));

for(int j=0;j<nrC;j++)

{

dist(0,j)=0;

for(int k=0;k<ncC;k++)

dist(0,j)+=(S(i,k)-C(j,k))\*(S(i,k)-C(j,k));

// identify positional value of cluster mean that will have minimum distance

if(dist(0,j)<minD)

{

pos=j;

minD=dist(0,j);

}

}

// if lookup table value for this doesn't matches cluster mean's position then update

if(((int)lookup(i,0))!=pos)

{

change=true;

lookup(i,0)=pos;

}

// calculate sum of values in each dimension belonging to each cluster

for(int j=0;j<ncC;j++)

{

sum((int)lookup(i,0),j)+=S(i,j);

}

// increase count for the cluster mean by 1

count((int)lookup(i,0),0)++;

}

// update mean for each cluster

if(change==true)

{

// divide sum by count and update mean

for(int i=0;i<nrC;i++)

{

if(count(i,0)!=0)

{

for(int j=0;j<ncC;j++)

{

C(i,j)=(sum(i,j)/count(i,0));

}

}

}

}

} while(change==true);

return lookup;

}

**/lib/winnertakesall.cpp**

#include "Matrix.h"

#include "Pr.h"

#include <iostream>

#include <cstdlib>

#include <cmath>

using namespace std;

/\*\*

\*\*/

Matrix winnerTakesAll(Matrix &S, Matrix &C, int max\_val)

{

int nrS=S.getRow();

int ncS=S.getCol();

int nrC=C.getRow();

int ncC=C.getCol();

int d=0;

Matrix dist(1,nrC);

int pos;

double minD;

Matrix lookup(nrS,1); // lookup table for sample belonging to which cluster

Matrix count(nrC,1); // to store count for each cluster center

Matrix sum(nrC,3);

// initialize cluster means

for(int i=0;i<nrC;i++)

for(int j=0;j<ncC;j++)

{C(i,j)=rand()%(max\_val+1);}

for(int i=0;i<nrS;i++)

{

lookup(i,0)=0;

}

bool change;

// while change of assignment to centers occurs

do{

cout<<d++<<endl;

change=false;

// clear values in count and cluster mean

for(int i=0;i<nrC;i++)

{

for(int j=0;j<ncC;j++)

{

sum(i,j)=0;

count(i,0)=0;

}

}

for(int i=0;i<nrS;i++)

{

// calculate distance between sample data and each cluster mean

minD=0;

pos=0;

for(int k=0;k<ncC;k++)

minD+=(S(i,k)-C(0,k))\*(S(i,k)-C(0,k));

for(int j=0;j<nrC;j++)

{

dist(0,j)=0;

for(int k=0;k<ncC;k++)

dist(0,j)+=(S(i,k)-C(j,k))\*(S(i,k)-C(j,k));

// identify positional value of cluster mean that will have minimum distance

if(dist(0,j)<minD)

{

pos=j;

minD=dist(0,j);

}

}

// if lookup table value for this doesn't matches cluster mean's position then update

if(((int)lookup(i,0))!=pos)

{

change=true;

lookup(i,0)=pos;

//cout<<lookup(i,0);

}

// update value that cluster mean

for(int k=0;k<ncC;k++)

{

C(pos,k)+=0.00001\*(S(i,k)-C(pos,k));

}

// calculate sum of values in each dimension belonging to each cluster

for(int j=0;j<ncC;j++)

{

sum((int)lookup(i,0),j)+=S(i,j);

}

// increase count for the cluster mean by 1

count((int)lookup(i,0),0)++;

}

// update mean for each cluster

if(change==true)

{

// divide sum by count and update mean

for(int i=0;i<nrC;i++)

{

if(count(i,0)!=0)

{

for(int j=0;j<ncC;j++)

{

C(i,j)=(sum(i,j)/count(i,0));

}

}

}

}

} while(change==true);

return lookup;

}

**/lib/kohonenmap.cpp**

#include "Matrix.h"

#include "Pr.h"

#include <iostream>

#include <cstdlib>

#include <cmath>

using namespace std;

Matrix kohonenMap(Matrix &S, Matrix &C, int max\_val, int kmax, int map\_width, double sig, double e)

{

int nrS=S.getRow();

int ncS=S.getCol();

int nrC=C.getRow();

int ncC=C.getCol();

int d=0;

Matrix dist(1,nrC);

Matrix sdist(1,nrC);

int pos;

double minD;

Matrix lookup(nrS,1); // lookup table for sample belonging to which cluster

Matrix count(nrC,1); // to store count for each cluster center

Matrix sum(nrC,3);

Matrix g(nrC,3);

int x=0;

double sigma=1;

double phi=0;

// initialize cluster means and map co-ordinates

for(int i=0;i<nrC;i++)

{

for(int j=0;j<ncC;j++)

{

C(i,j)=rand()%(max\_val+1);

}

}

int y=0;

do{

for(int k=0;k<map\_width;k++)

{

g(y,0)=y;

g(y,1)=x;

g(y,2)=k;

y++;

if(y>=nrC)

{

break;

}

}

if(y<nrC)

x++;

}while(y<nrC);

// calculate sigma

/\*double maxD=0;double gdist;

for(int i=0;i<nrC;i++)

{

gdist=pow(g(i,1)-g(0,1),2)+pow(g(i,2)-g(0,2),2);

if(gdist>maxD)

{

maxD=gdist;

}

}\*/

sigma=sig;

//Matrix x1,y1; // for storing points to be use calculate distance

for(int i=0;i<nrS;i++)

{

lookup(i,0)=0;

}

bool change;

// while change of assignment to centers occurs

do{

change=false;

cout<<d++<<endl;

// clear values in count and cluster mean

for(int i=0;i<nrC;i++)

{

for(int j=0;j<ncC;j++)

{

sum(i,j)=0;

count(i,0)=0;

}

}

for(int i=0;i<nrS;i++)

{

// calculate distance between sample data and each cluster mean

minD=0;

pos=0;

for(int k=0;k<ncC;k++)

minD+=(S(i,k)-C(0,k))\*(S(i,k)-C(0,k));

for(int j=0;j<nrC;j++)

{

dist(0,j)=0;

for(int k=0;k<ncC;k++)

dist(0,j)+=(S(i,k)-C(j,k))\*(S(i,k)-C(j,k));

// identify positional value of cluster mean that will have minimum distance

if(dist(0,j)<minD)

{

pos=j;

minD=dist(0,j);

}

}

// if lookup table value for this doesn't matches cluster mean's position then update

if(((int)lookup(i,0))!=pos)

{

change=true;

lookup(i,0)=pos;

}

for(int j=0;j<nrC;j++)

{

phi=exp(-(pow(g(pos,1)-g(j,1),2)+pow(g(pos,2)-g(j,2),2))/(2\*pow(sigma,2)));

// update value all cluster means

for(int k=0;k<ncC;k++)

{

C(j,k)+=e\*phi\*(S(i,k)-C(j,k));

}

}

// calculate sum of values in each dimension belonging to each cluster

for(int j=0;j<ncC;j++)

{

sum((int)lookup(i,0),j)+=S(i,j);

}

// increase count for the cluster mean by 1

count((int)lookup(i,0),0)++;

}

// update mean for each cluster

if(change==true)

{

// divide sum by count and update mean

for(int i=0;i<nrC;i++)

{

if(count(i,0)!=0)

{

for(int j=0;j<ncC;j++)

{

C(i,j)=(sum(i,j)/count(i,0));

}

}

}

}

} while(d<kmax && change==true);

return lookup;

}

**/example/testUnsup.cpp**

#include <iostream>

#include <fstream>

#include <cmath>

#include <cstdlib>

#include "Matrix.h"

#include "Pr.h"

using namespace std;

#define Usage "Usage: ./testUnsup image\_file pixel1 pixel2 color\_represent clustering\_method \n\t image\_file: the file name for image in ppm format \n\t pixel1: number of pixel in one dimenion \nt pixel2: number of pixel in other dimension \n\t color\_represent: no. of colors to be represented by \n\t clustering\_method: 1-> k-means; 2-> winner takes all; 3-> kohonen map \n"

int main(int argc, char \*\*argv)

{

// check to see if the number of argument is correct

if (argc ^ 6) {

cout << Usage;

exit(1);

}

// define no. of color in a pixel

//int r=3; // (R,G,B)

// read values from argument

int row=atoi(argv[2]);

int col=atoi(argv[3]);

int clr=atoi(argv[4]);

int c\_method=atoi(argv[5]);

int \*rImg,\*cImg;

rImg=&row;

cImg=&col;

// read in data from the data file

Matrix mTr = readImage(argv[1], rImg, cImg);

int nrTr = mTr.getRow(); // get the number of rows

int ncTr=mTr.getCol();

// define arbitrary clr cluster means

Matrix C(clr,ncTr);

// call kmeans to get lookup table that refers each pixel in original pic to updated cluster mean pixel

Matrix lookup;

switch(c\_method)

{

case 1:

lookup=kmeans(mTr,C, 255);

break;

case 2:

lookup=winnerTakesAll(mTr,C,255);

break;

case 3:

lookup=kohonenMap(mTr,C,255, 200,4,4,0.00001);

}

Matrix kTr(nrTr,ncTr), dTr(nrTr,ncTr);

for(int i=0;i<nrTr;i++)

{

for(int j=0;j<ncTr;j++)

{

kTr(i,j)=C(lookup(i,0),j);

}

}

writeImage("flower2.ppm", kTr, row, col);

for(int i=0;i<nrTr;i++)

{

for(int j=0;j<ncTr;j++)

{

dTr(i,j)=mTr(i,j)-kTr(i,j);

}

}

writeImage("diff\_flower\_kmeans.ppm", dTr, row, col);

return 0;

}