Image Quantization Analysis

T055

# GetDistinctColorsList(ImageMatrix):

private static List<int> GetDistinctColorsList(RGBPixel[,] ImageMatrix)

{

HashSet<int> distinctColors = new HashSet<int>();

foreach (RGBPixel pixel in ImageMatrix)

distinctColors.Add(RGBPixel.Hash(pixel)); // O(N^2) if C# handles resizeing well

// Can we do better?

List<int> colorsList = distinctColors.ToList();

colorIndices = new Dictionary<int, int>(distinctColors.Count);

for (int i = 0; i < colorsList.Count; i++) // O(D)

colorIndices.Add(colorsList[i], i);

return colorsList;

}

Final Order: O(N^2)

# Prim(List<int> distinctColors):

public static int[] Prim(List<int> distinctColors)

{

int V = distinctColors.Count;

int[] parent = new int[V];

int[] key = new int[V];

bool[] mstSet = new bool[V];

//initialize all keys as infinite

for (int i = 0; i < V; i++) //O(D)

{

key[i] = int.MaxValue;

mstSet[i] = false;

}

key[0] = 0;

parent[0] = -1;

for (int i = 0; i < V - 1; i++) //O(D^2)

{

int u = GetMinimumKey(key, mstSet);

mstSet[u] = true;

//relax all edges connected to u

for (int v = 0; v < V; v++) //O(D)

{

int distance = ColorQuantization.GetWeight(

distinctColors[u],

distinctColors[v]

);

if (distance != 0

&& mstSet[v] == false

&& distance < key[v]

)

{

parent[v] = u;

key[v] = distance;

}

}

}

return parent;

}

Final Order: O(D^2)

# ConstructEdges(List<int> distinctColors, int[] parent):

public static void ConstructEdges(List<int> distinctColors, int[] parent)

{

edges = new List<Edge>();

int V = distinctColors.Count;

for (int i = 1; i < V; i++) // O(D)

{

int weight = ColorQuantization.GetWeight(distinctColors[i], distinctColors[parent[i]]); //O(1)

sum += ColorQuantization.GetDistance(weight); //O(1)

Edge e = new Edge(parent[i], i, weight, ColorQuantization.GetDistance(weight));

edges.Add(e);

}

}

Final Order: O(D)

# ClusterEdges(int num\_of\_clusters):

public static List<Edge> ClusterEdges(int num\_of\_clusters)

{

if (edges == null)

{

throw new Exception("MST is not constructed");

}

edges.Sort(); //O(DlogD)

int numOfEdgesBefore = edges.Count;

for (int i = 0; i < num\_of\_clusters - 1; i++)

//O(K) where K is num\_of\_clusters <= D

{

int lastIndex = edges.Count - 1;

edges.RemoveAt(lastIndex); //O(1)

}

return edges;

}

Final Order: O(DlogD)

# Forest(List<Edge> Edges):

public Forest(List<Edge> Edges)

{

Trees = new Dictionary<int, List<int>>();

foreach (Edge edge in Edges) //O(D-K)

{

addEdge(edge.from, edge.to); //O(1)

addEdge(edge.to, edge.from); //O(1)

}

}

Final Order: O(D-K)

// EXPECTED TO CHANGE

# BFS\_HELP(int s, List<bool> visited):

public static List<RGBPixel> BFS\_HELP(int s, List<bool> visited)

{

List<RGBPixel> cluster = new List<RGBPixel>();

List<int> q = new List<int>();

q.Add(s);

visited.RemoveAt(s);

visited.Insert(s, true);

while (q.Count != 0)

{

int f = q[0];

q.RemoveAt(0);

cluster.Add(IToPixel(f)); //O(1)

if (Trees.ContainsKey(f))

{

foreach (int iN in Trees[f]) O(D)

{

int n = iN;

if (!visited[n])

{

visited.RemoveAt(n);

visited.Insert(n, true);

q.Add(n);

}

}

}

}

return cluster;

}

Final Order: O(D-K)

// EXPECTED TO CHANGE

# BFS(distinctColorsList.Count):

public static Dictionary<int, List<RGBPixel>> BFS(int V)

{

List<bool> visited = new List<bool>();

Dictionary<int, List<RGBPixel>> clusters = new Dictionary<int, List<RGBPixel>>();

for (int i = 0; i < V; i++) //O(D)

{

visited.Insert(i, false);

}

int index = 0;

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

{

if (!visited[i])

{

var cluster = BFS\_HELP(i, visited);

index++;

clusters.Add(index, cluster);

}

}

lastkey=index;

return clusters;

}

Final Order: O(1)

# ReduceImageColors(ImageMatrix, ColorPallette, ClusterIndices):

// EXPECTED TO CHANGE

private static void ReduceImageColors(RGBPixel[,] ImageMatrix, Dictionary<int, RGBPixel> ColorPallette, Dictionary<int, short> clusterIndices)

{

int rows = ImageOperations.GetHeight(ImageMatrix);

int columns = ImageOperations.GetWidth(ImageMatrix);

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

{

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

{

RGBPixel currentColor = ImageMatrix[i, j];

int currentColorIndex = colorIndices[RGBPixel.Hash(currentColor)];

int currentColorClusterIndex = clusterIndices[currentColorIndex];

RGBPixel newColor = ColorPallette[currentColorClusterIndex];

ImageMatrix[i, j] = newColor;

}

}

}

Final Order: O(N^2)

# Bonus 1 (Finding K using MSDR):

// EXPECTED TO CHANGE.

GetColorPallette(clusters):

public static Dictionary<int, RGBPixel> GetColorPallette(Dictionary<int, List<RGBPixel>> clusters)

{

// for every member of cluster sum all values and get the mean for the sum

Dictionary<int, RGBPixel> colorPallete = new Dictionary<int, RGBPixel>();

foreach (int clusterIndex in clusters.Keys)

{

int sumRed = 0, sumGreen = 0, sumBlue = 0;

int numberOfColorsInCluster = clusters[clusterIndex].Count;

foreach (RGBPixel pixel in clusters[clusterIndex])

{

sumRed += pixel.red;

sumBlue += pixel.blue;

sumGreen += pixel.green;

}

sumRed = (int)Math.Ceiling((double)sumRed / numberOfColorsInCluster);

sumGreen = (int)Math.Ceiling((double)sumGreen / numberOfColorsInCluster);

sumBlue = (int)Math.Ceiling((double)sumBlue / numberOfColorsInCluster);

byte red = Convert.ToByte(sumRed);

byte green = Convert.ToByte(sumGreen);

byte blue = Convert.ToByte(sumBlue);

RGBPixel representitaveColor = new RGBPixel(red, green, blue);

colorPallete.Add(clusterIndex, representitaveColor);

}

return colorPallete;

}

Final order: **Ø(N)**