My Implementation of the GrabCut Algorithm

COMP8501 - Assignment1 Lu Jintao, 3030084309

Experiment Settings: I run the project on a laptop with 5.0GHz intel i7-10870H, nVidia RTX3070, 16G RAM, and test in Windows 11 with .Net 4.7.2 Framework, developing using C#, VS2022

Algorithm Overview:

1. External Libraries:
    a. *Accord.Net*, which supply a implementation of the standard GMM
    b. *OpenCVSharp*, which is similar to OpenCV for C++
2. Algorithm Overview:

I follow the pipeline proposed in the paper as:

Initialisation

- User initialises trimap T by supplying only T_B. The foreground is set to $T_F = \emptyset$; $T_U = \overline{T}_B$, complement of the background.
- Initialise $\alpha_n = 0$ for $n \in T_B$ and $\alpha_n = 1$ for $n \in T_U$.
- Background and foreground GMMs initialised from sets $\alpha_n = 0$ and $\alpha_n = 1$ respectively.

Iterative minimisation

1. *Assign GMM components to pixels:* for each n in T_U,
$$k_n := \arg\min_{k_n} D_n(\alpha_n, k_n, \theta, z_n).$$
2. *Learn GMM parameters* from data $\mathbf{z}$:
$$\underline{\theta} := \arg\min_{\underline{\theta}} U(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z})$$
3. *Estimate segmentation:* use min cut to solve:
$$\min_{\{\alpha_n:\, n \in T_U\}} \min_{\mathbf{k}} \mathbf{E}(\underline{\alpha}, \mathbf{k}, \underline{\theta}, \mathbf{z}).$$
4. Repeat from step 1, until convergence.
5. Apply border matting (section 4).

At the following, I will introduce each part (class *GMM*, class *Dinic*) individually, and then present my implementation of the main class *grabCut.*

Module1: GMM model

The GMM model is built with parameters componentsCount = 4 as the number of GMM components. We build two models for modeling the Foreground and Background distributions, respectively.

```
public class GMM
{
    public GaussianMixtureModel gmmForeground;
    public GaussianMixtureModel gmmBackground;
    public GaussianClusterCollection clustersForeground;
    public GaussianClusterCollection clustersBackground;

    private const int componentsCount = 4;

    public GMM()
    {
        gmmForeground = new GaussianMixtureModel(componentsCount);
        gmmBackground = new GaussianMixtureModel(componentsCount);
    }
}
```

After the definition, we optimize the parameters on each GMM component (`double[][] gmm.Gaussian[i].Means, double[][] gmm.Gaussian[i].Variance`) by the method `clustersForeground = gmmForeground.Learn(samplesForeground)`, also the same for the `gmmBackground` model. The learning for a dataset is implemented as:

```
public void Fit(Mat image, Mat mask)
{
    // Convert image to data points for the GMM
    double[][] samplesForeground = MatToJagged(image, mask,
(byte)GrabCutClasses.PR_FGD);
    double[][] samplesBackground = MatToJagged(image, mask,
(byte)GrabCutClasses.PR_BGD);

    // Re-estimate GMM parameters for the foreground
    if (samplesForeground.Length > 0)
    {
        gmmForeground = new GaussianMixtureModel(componentsCount);
        clustersForeground = gmmForeground.Learn(samplesForeground);
    }
    // Re-estimate GMM parameters for the background
    if (samplesBackground.Length > 0)
    {
        gmmBackground = new GaussianMixtureModel(componentsCount);
        clustersBackground = gmmBackground.Learn(samplesBackground);
    }
}
```

and the estimation from the gmm model is computed by the PDF as

```
public double ComputePDF(GaussianMixtureModel gmm, double[] dataPoint, bool
print = false)
```

```
{
    double probability = 0;
    for (int i = 0; i < gmm.Gaussians.Count; i++)
    {
        double likelihood = -0.01 * gmm.Gaussians[i].LogLikelihood(dataPoint);
        probability += likelihood;
    }
    return probability;
}
```

in which the `LogLikelihood` is computed by:

```
System.Math.Log(coefficients[componentIndex]) +
components[componentIndex].LogProbabilityFunction(x);
```

Module2: Graph Computing for Minimum Cut

2.1 Graph Building

We Add edge from the source/sink node to the nodes based on their pdf estimation, and set it as the Capicity of each edge, the reverse flow is also initalized with 0.

```
double[] sample = new double[] { color.Item0, color.Item1,
color.Item2 };

double bgProb = gmm.ComputePDF(gmm.gmmBackground, sample);
double fgProb = gmm.ComputePDF(gmm.gmmForeground, sample);

int bgCapacity = (int)(bgProb);
int fgCapacity = (int)(fgProb);
AddEdge(source, nodeIndex, fgCapacity);
AddEdge(nodeIndex, sink, bgCapacity);
```

We then connect each node with its neighbors based on the weight that negatively correlated with the color difference, as:

```
if (x > 0)
{
    int w = (int) leftW.At<double>(y, x);
    int neighborIndex = ComputeNodeID(x - 1, y, img);
    AddEdge(nodeIndex, neighborIndex, w);
    AddEdge(neighborIndex, nodeIndex, w);
}
if (x > 0 && y > 0)
{
    int w = (int)upleftW.At<double>(y, x);
    int neighborIndex = ComputeNodeID(x - 1, y - 1, img);
    AddEdge(nodeIndex, neighborIndex, w);
    AddEdge(neighborIndex, nodeIndex, w);
}
if (y > 0)
{
    int w = (int)upW.At<double>(y, x);
```

```
    int neighborIndex = ComputeNodeID(x, y - 1, img);
    AddEdge(nodeIndex, neighborIndex, w);
    AddEdge(neighborIndex, nodeIndex, w);
}
if (x < img.Cols - 1 && y>0)
{
    int w = (int)uprightW.At<double>(y, x);
    int neighborIndex = ComputeNodeID(x + 1, y - 1, img);
    AddEdge(nodeIndex, neighborIndex, w);
    AddEdge(neighborIndex, nodeIndex, w);
}
```

2.2 Compute Minimum Cut by Dinic Algorithm with Dinic Algorithm

As computing the maximum flow is equal to the minimum cut, I implement a *Dinic* Algorithm which use BFS to check possible Augmenting Path, and use DFS to add the flow on the augmenting path, the main procedure of the Dinic algorithm is:

```
public int Dinic()
{
    int flow = 0;
    int[] level = new int[Nodes.Length];
    while (BFS(level))
    {
        int[] start = new int[Nodes.Length];
        while (true)
        {
            int f = DFS(Source, int.MaxValue, level, start);
            if (f == 0) break;
            flow += f;
        }
    }
    return flow;
}
```

and the *BFS(level), DFS(Source, int.MaxValue, level, start)* can be detailly checked in my code.

2.3 Update the Mask

After compute the MaxFlow, the nodes that still can be approached from the source node, which means they are cut to the side of possible foreground, can be acquired after a BFS. Then the mask will be updated based on the results.

```
public void EstimateSegmentation(Graph graph, Mat mask)
{
    // Compute the MaxFlow
    int maxFlow = graph.Dinic();
    Console.WriteLine("MaxFlow "+ maxFlow);

    // BFS again and find the reachable nodes from the source
    HashSet<int> reachableFromSource = new HashSet<int>();
    Queue<int> queue = new Queue<int>();
```

```
    bool[] visited = new bool[graph.Nodes.Length];
    int source = graph.Source;

    queue.Enqueue(source);
    visited[source] = true;

    while (queue.Count > 0)
    {
        int currentNode = queue.Dequeue();
        reachableFromSource.Add(currentNode);

        foreach (Edge edge in graph.Nodes[currentNode].Edges)
        {
            if (!visited[edge.To] && edge.Capacity - edge.Flow > 0)
            {
                queue.Enqueue(edge.To);
                visited[edge.To] = true;
            }
        }
    }

    // Updating Mask after classification
    for (int y = 0; y < mask.Rows; y++)
    {
        for (int x = 0; x < mask.Cols; x++)
        {
            int nodeIndex = y * mask.Cols + x;
            if (reachableFromSource.Contains(nodeIndex))
            {
                if (y % 50 == 0 && x % 50 == 0)
                    Console.WriteLine("MinCut Updating xy "+ x+","+y);
                mask.Set<byte>(y, x, (byte)GrabCutClasses.PR_FGD);
            }
            else
            {
                mask.Set<byte>(y, x, (byte)GrabCutClasses.PR_BGD);
            }
        }
    }
}
```

Main Class GrabCut

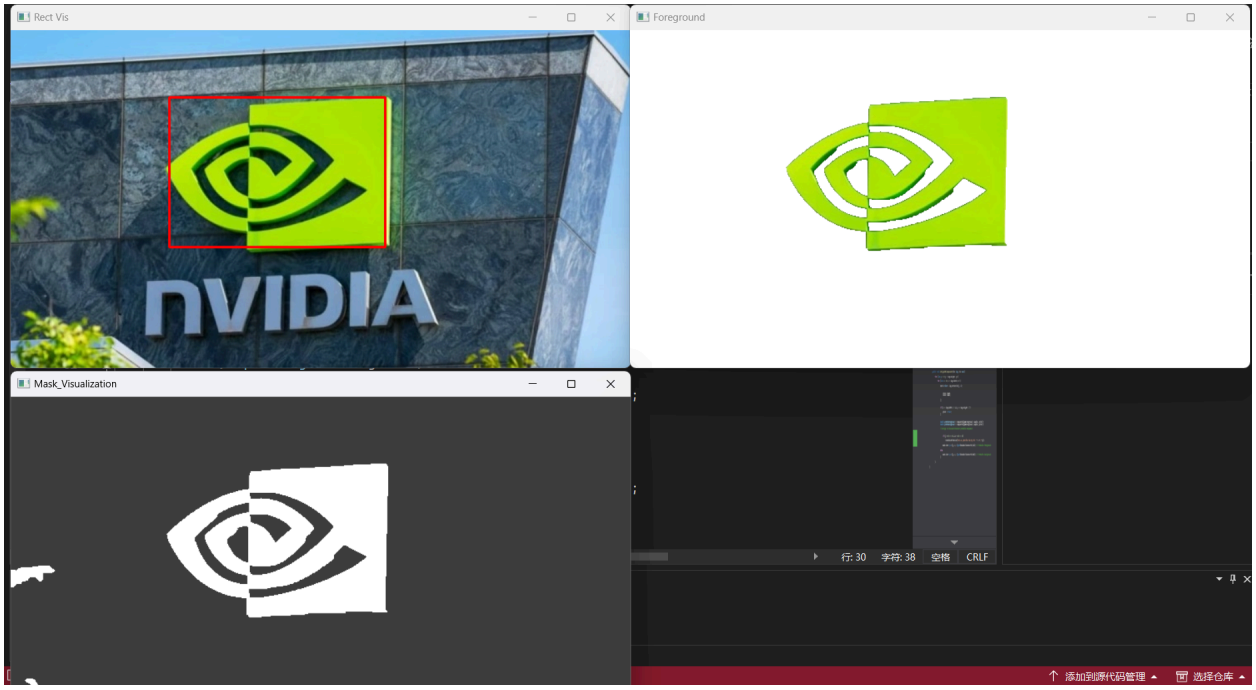
In our main class grabCut, we implement the main iterations, in each iteration the GMM assign the mask based on its prediction, and then learn the data distribution. After that, the grabCut run the Minimum Cut algorithm and update the mask again, as follows:

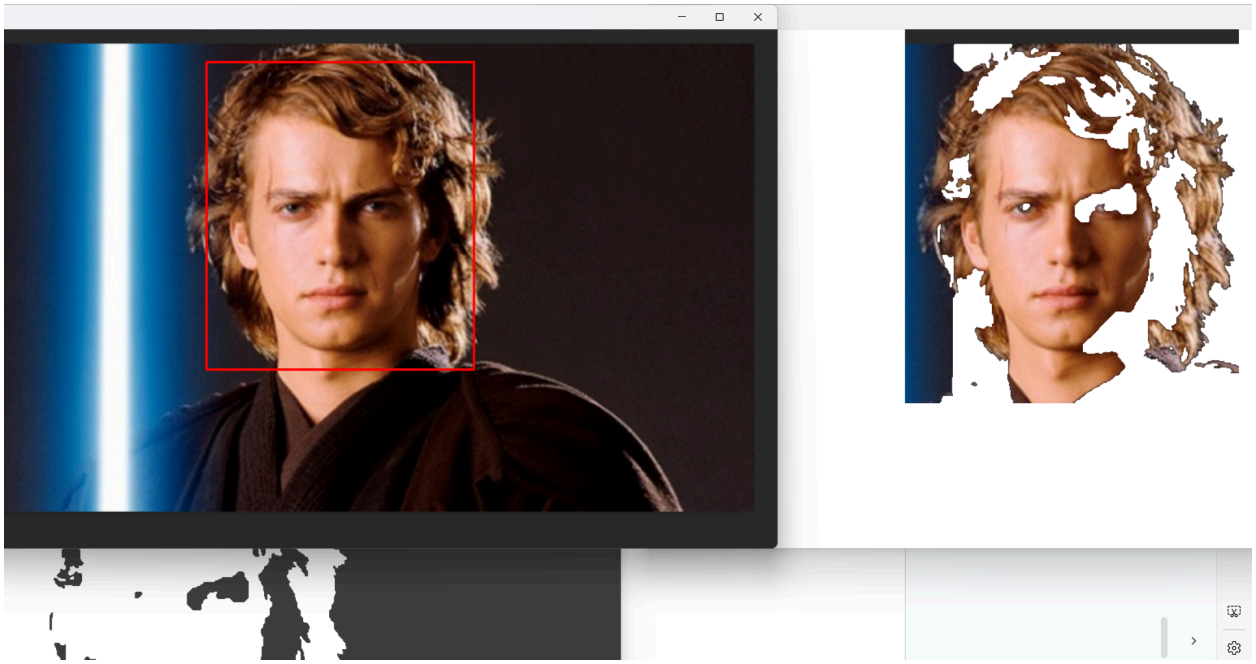
```
GMM gmmModel = new GMM();
gmmModel.Initialize(img, mask, rect);
```

```
for (int i = 0; i < iterCount; i++)
{
    // 1. Assign GMM components to pixels
    gmmModel.AssignGMMComponents(img, mask);

    // 2.Learn and Update GMM parameters
    gmmModel.Fit(img, mask);

    // 3.Construct graph
    Graph graph = new Graph(2,0,1);
    graph.ConstructGraph(img, mask, gmmModel, leftW, upleftW, upW, uprightW);
    Console.WriteLine("SourceID "+ graph.Source + " sink "+ graph.Sink);
    // 4. Estimate segmentation, and mask updating
    graph.EstimateSegmentation(graph, mask);
}
```

Results:

Implementation details: Check the *grabCut.cs, GMM.cs, Dinic.cs*
Running: grabCut_JTLU\grabCut_JTLU\bin\Debug\grabCut_JTLU.exe
Time cost: ~200 seconds for running 5 iterations with image size as 1063x640p.





In some difficult cases, when the lighting change rapidly, the grabCut may fill like this. Tuning the weights of nearby color similarity and the GMM prediction will finally make the results better, like: