Optimization Project 1 - Image Segmentation

Sarah Stephens (sgs2623), Tara Mary Joseph (tj9763), Theresa Sushil (ts42924), Sarah Lee (cl45274)

Overview

Objective

Explore the feasibility of implementing an image segmentation tool using linear programming (LP) and max-flow/min-cut optimization techniques to compete with PhotoShop.

Background

Image segmentation is the process of partitioning an image into meaningful regions. Given familiarity with optimization and the max-flow/min-cut theorem, the idea is to scope out the approach of using linear programming to perform image segmentation, leveraging the graph-cut method.

Max-Flow Min-Cut Analysis

The max-flow min-cut theorem is a fundamental principle in computer science and optimization, particularly useful for image segmentation. The theorem states that the maximum flow from a source to a sink in a network is equal to the total weight of the minimum cut that separates them. In image segmentation, this is applied by algorithmically separating the foreground, typically the object, from the background. Each pixel in the image is represented as a node and edges between other pixels reflect its intensity similarities. This flow of similarity is taken into the algorithm to efficiently draw a line or cut where the similarity is the weakest. Ultimately, using the max-flow min-cut method, we can identify the optimal boundary of an image or a network flow that distinguishes the source (foreground) from the sink (background).

* See appendix for handwritten notes & diagrams explaining the max-flow min-cut theorem

Key Model Assumptions

1. Each pixel in an image corresponds to a node in a graph, with edges connecting neighboring pixels.
2. These edges are weighted based on the similarity between pixels, typically determined by their relative intensities. A higher edge weight indicates a stronger similarity between neighboring pixels.
3. The similarity function used to calculate these weights is as follows:

$$100\exp\left(-\frac{(I_i - I_j)^2}{2\sigma^2}\right)$$

I_i = intensity of pixel i
I_j = intensity of pixel j
σ = controls sensitivity of similarity between pixels (predefined by user)

Algorithm Overview

1. Parameter Defining and Image Processing

Our algorithm specifies key parameters by user input: a sigma value, the coordinates for the foreground and background pixels, and the image file path. The sigma value controls the sensitivity of the pixel similarity. The foreground and background pixels serve as the source and sink nodes in the graph.

Next, we process the image file, converting it to grayscale and handling different file types for a simplified intensity comparison. We also extract important details like image size and pixel count, allowing the algorithm to work with various input formats. Then, each pixel is connected to the four neighboring pixels (top, bottom, left, and right) to create the network for segmentation.

*Source and sink nodes are the starting and end points in the graph, used to compute flow and determine the optimal segmentation cuts.

2. Determining Edge Weights

As mentioned earlier, the internal edge weights reflect the similarity between neighboring pixels. The similarity between the neighbors is calculated and stored into a similarity graph dictionary.

```
# Calculate similarity between current pixel & neighbor
sim_value = np.ceil(100 * np.exp(-((pixel_intensity - neighbor_intensity) ** 2) / (2 * sigma ** 2)))

# Save similarity to dictionary
similarity_graph[(pixel_idx, neighbor_idx)] = sim_value
```

A higher weight or similarity value represents a more similarity in intensity, leading to an edge weight. To keep things efficient, we connect each pixel to its four nearest neighbors. This turns

the image into a weighted graph where similar pixels are grouped together, setting the stage for optimal segmentation cuts. Note, a larger sigma will increase the sensitivity when small intensity differences occur.

3. Compute Maximum Flow

Next, we use the max-flow min-cut algorithm to calculate the flow through the network, which represents the amount of information that moves from the source (background) to the sink (foreground) along the network of pixel edges based on the similarities.

```
# Initialize Gurobi model to calculate max flow
gurobi_model = Model('MaxFlow')

# Define decision variables for the flow on each edge in the graph
flow_values = {}
for (start, end), capacity in similarity_graph.items():
  flow_values[start, end] = gurobi_model.addVar(lb=0, ub=capacity, vtype=GRB.CONTINUOUS,
name=f"flow_{start}_{end}")

# Objective = maximize flow from source to sink
nodes_in_graph = set(range(num_pixels)) | {source_index, sink_index}
flow_from_source = sum(flow_values[source_index, neighbor] for neighbor in nodes_in_graph if (source_index, neighbor) in
flow_values)
gurobi_model.setObjective(flow_from_source, GRB.MAXIMIZE)

# Apply flow conservation for every node except source and sink
for node in nodes_in_graph:
  if node != source_index and node != sink_index:
    inflow = sum(flow_values[parent, node] for parent in nodes_in_graph if (parent, node) in flow_values)
    outflow = sum(flow_values[node, child] for child in nodes_in_graph if (node, child) in flow_values)
    gurobi_model.addConstr(inflow == outflow, name=f'flow_balance_{node}')

# Optimize!!!
gurobi_model.Params.OutputFlag = 0  # No output
gurobi_model.optimize()

# Print max flow value if optimal solution found
if gurobi_model.status == GRB.OPTIMAL:
  print(f"Max Flow: {gurobi_model.objVal}")
```

For each of these nodes/pixels, the constraint is applied to ensure the entering flow (inflow) equals the leaving flow (outflow), maintaining flow conservation. After Gurobi's optimization, the max flow is calculated, representing the strength of the connection between the background (source) and the foreground (sink).

4. Finding the Minimum Cut

By calculating the maximum flow, the algorithm identifies the minimum cut, which represents the optimal boundary between the foreground and background. Using the depth first search function, all nodes are explored to classify reachable and unreachable nodes, which determine the cut edges, the edges that connect the two nodes.

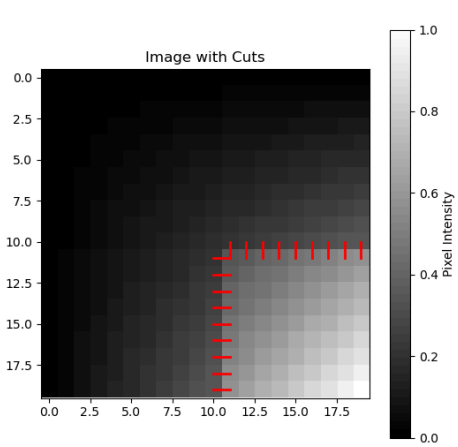
```
# Run DFS from source node
reachable_nodes = depth_first_search(residual_graph, source_index)

# Identify nodes that can't be reached from the source
unreachable_nodes = set(range(num_pixels + 2)) - reachable_nodes

# Find cut edges
cut_edges = [(i, j) for (i, j) in similarity_graph.keys() if i in reachable_nodes and j in unreachable_nodes]
print(cut_edges)
```

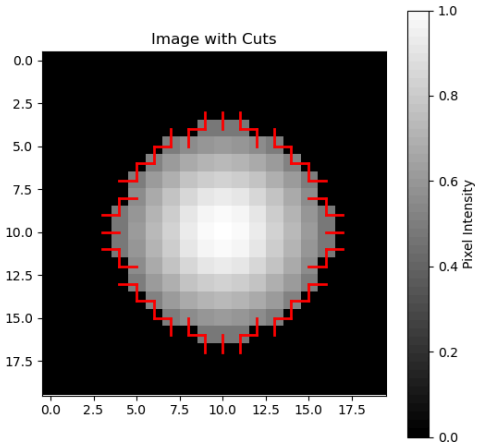
The cuts are made at the edges with the least similarity, where there is a sharp contrast between light and dark areas or the object and the background.

5. Visualizing Image Segmentation

Lastly, the algorithm reconstructs the image with an overlay of the optimal cuts, completing our image segmentation task.

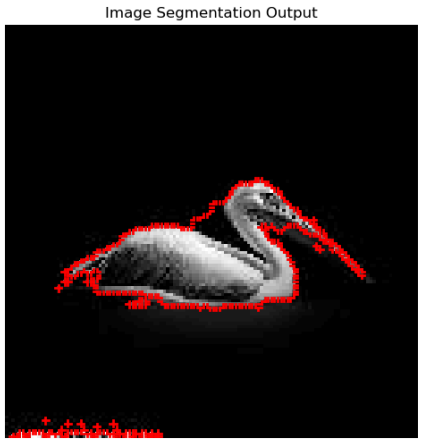
Below are examples of csv files with its segmented optimal cuts
box (left) (sigma - 0.05, background - (0,0), foreground (19,19))
oval (right) (sigma - 0.05, background - (0,0), foreground (10,10)):

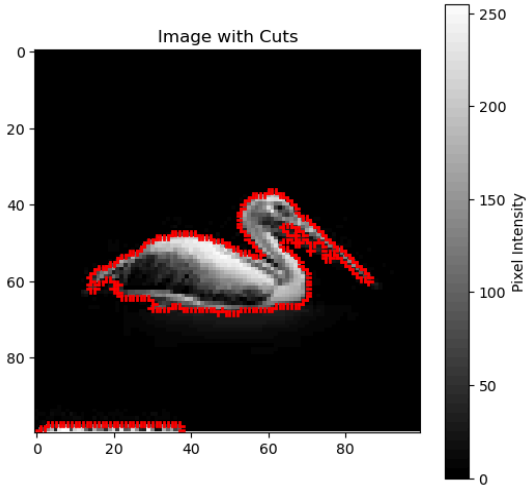




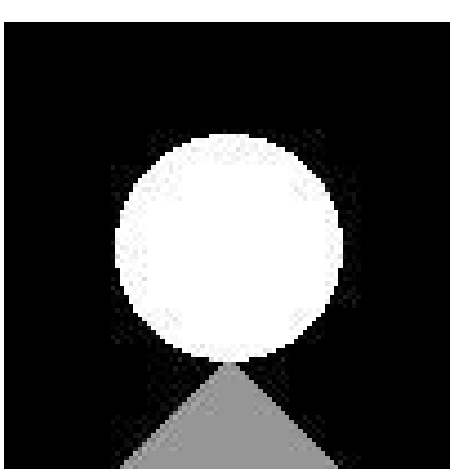
Below is the example of the pelican file with its segmented optimal cuts (sigma - 0.3, background - (0,0), foreground (64,64))

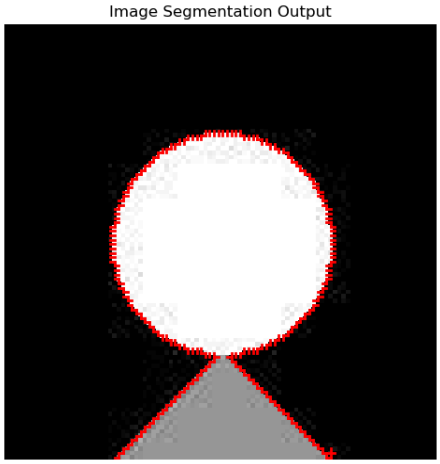
Pelican (left) - Pelican segmentation with original image size of 128*128
Pelican (right) - Pelican segmentation with original image resized to 100*100



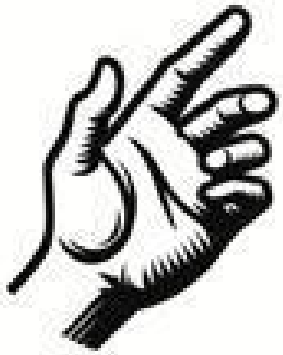


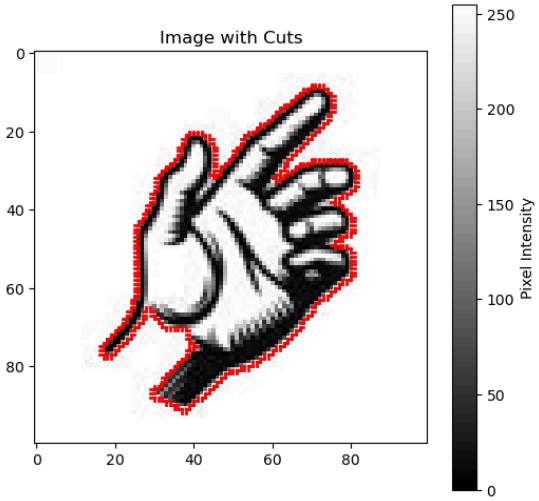
We applied the algorithm to a geometric image: the original image (left) and with its optimal cuts (right) (sigma - 1, background - (0,0), foreground (50,50))





We applied the algorithm to a hand image: the original image (left) and with its optimal cuts (right) (sigma - 0.3, background - (0,0), foreground (50,50))





Feasibility

Strengths

Applying the max-flow min-cut theorem to image segmentation presets is beneficial for the following reasons:

- **Flexibility:** With the ability to process various types of images as graphs and focus on the relative similarity of pixels, our algorithm is flexible (yet strong) enough to handle a variety of projects in the least costly way possible

- **Optimal Solutions:** Our algorithm guarantees the best separation between the foreground and background (binary segmentation) through the minimum cut by finding the edges with dissimilarity to segment or cut. LP methods guarantee globally optimal solutions, which is valuable for producing accurate boundaries.
- **Structure Alignment**: Image segmentation problems can be represented as graph problems, which align well with the capabilities of LP.

Challenges

- **Hyperparameter tuning**: The limitation of the max-flow min-cut theorem is that it requires the user to specify certain characteristics of the image, such as a background pixel and a foreground pixel, which requires knowledge of the image size and matrix. Additionally, this theorem—and the resulting similarity calculation it requires for implementation—requires the use of an optimal sigma, which can take a sequence of trial and error to find. This algorithm considers the 4 neighbors, but considering more neighbors like its diagonal neighbors may improve the accuracy of the segmentation for certain images.
- **Scalability:** Applying linear programming (LP) to large images will be compute heavy since each pixel is treated like a node in the graph, resulting in a huge number of variables. This can slow things down compared to specialized algorithms designed specifically for image segmentation.
- **Non-linearity/Efficiency:** LP works best with linear constraints, but image boundaries in real life can be non-linear and complex. To handle these with a linear approach will be inefficient.

Recommendations

The min-cut max-flow technique is a powerful method for image segmentation, effectively identifying boundaries based on pixel similarity. While it offers precision, challenges remain, particularly in computational efficiency, and further research into other techniques like neural networks is recommended.

We recommend testing various algorithms across different use cases to find the best model based on accuracy and efficiency. Combining linear programming with other methods like clustering could enhance both accuracy and speed. By integrating advanced algorithms, our approach could improve everyday business tasks and deliver high-quality results, making it a strong contender for photo editing and specialized fields like medical imaging where precision is crucial.

References

Chatgpt

[Max-flow min-cut theorem - Wikipedia]

[Ford Fulkerson Algorithm]

[Network Flows: Max-Flow Min-Cut Theorem (& Ford-Fulkerson Algorithm) video by Back to Back SWE]

Appendix