Project 1 Q&A

Jonathan Krause
Outline

• GrabCut Review
• Error metrics
• Code Overview
• Project 1 Report
• Project 1 Presentations
Outline

• GrabCut Review
• Error metrics
• Code Overview
• Project 1 Report
• Project 1 Presentations
GrabCut

Input: some of this is foreground

Output: definitely background
The GrabCut Model

- **Energy Minimization:**

\[
E(\alpha, k, \theta, z) = U(\alpha, k, \theta, z) + V(\alpha, z)
\]

- $\alpha$: in $\{0,1\}$, foreground/background label for each pixel
- $k$: GMM component for each pixel
- $\theta$: GMM parameters
- $z$: vector of pixels
Unary Terms

\[ U(\alpha, k, \theta, z) = \sum_{n} D(\alpha_n, k_n, \theta, z_n) \]

\[ D(0, k_n, \theta, z_n) = -\log\text{(prob) of background GMM} \]

\[ D(1, k_n, \theta, z_n) = -\log\text{(prob) of foreground GMM} \]

Note: Only use a single GMM component
Pairwise Terms

\[ V(\alpha, z) = \gamma \sum_{(m,n) \in C} [\alpha_m \neq \alpha_n] \exp \left( -\beta \| z_m - z_n \|^2 \right) \]

\( \gamma \): a magic constant set to 50
\( \beta \): chosen heuristically based on average color difference
\( C \): set of all (pixel,pixel) neighboring pixels
Optimization

**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 $z$:
   
   $$\theta := \arg\min_{\theta} U(\alpha, k, \theta, z)$$

3. *Estimate segmentation:* use min cut to solve:
   
   $$\min_{\{\alpha_n: n \in T_U\}} \min_k E(\alpha, k, \theta, z).$$

4. Repeat from step 1, until convergence.

5. Apply border matting (section 4).

**User editing**
- *Edit:* fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap $T$ accordingly. Perform step 3 above, just once.
Set GMM components for each pixel

Trick: initialize components with K-means
- K-means uses hard labels
- Saves initial GMM setting
- Don’t need to do 1. in first iteration

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 $z$:
   $$\theta := \arg \min_{\theta} U(\alpha, k, \theta, z)$$
3. Estimate segmentation: use min cut to solve:
   $$\min_{\{\alpha_n : n \in T_U\}} \min_k \{E(\alpha, k, \theta, z)\}.$$
4. Repeat from step 1, until convergence.
5. Apply border matting (section 4).

User editing
- Edit: fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap $T$ accordingly. Perform step 3 above, just once.
Optimization

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 $z$:
   $$\theta := \arg \min_{\theta} U(\alpha, k, \theta, z).$$

3. Estimate segmentation: use min cut to solve:
   $$\min_{\{\alpha_n: n \in T_U\}} \min_k E(\alpha, k, \theta, z).$$

4. Repeat from step 1, until convergence.

5. Apply border matting (section 4).

User editing

- Edit: fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap $T$ accordingly. Perform step 3 above, just once.

Learn GMM parameters
- Components set, no EM needed
Optimize foreground/background labeling with min cut

- Submodular pairwise terms, exact
- Only need to optimize over pixels in bounding box.
- Trick: Just set large unary terms for pixels outside bounding box.

**Optimization**

**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 $z$:
   
   $\theta := \arg\min_{\theta} U(\alpha, k, \theta, z)$

3. Estimate segmentation: use min cut to solve:
   
   \[
   \min_{\{\alpha_n : n \in T_U\}} \min_k E(\alpha, k, \theta, z). 
   \]

4. Repeat from step 1, until convergence.

5. Apply border matting (section 4).

**User editing**
- *Edit:* fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap $T$ accordingly. Perform step 3 above, just once.
Optimization

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

User editing
- *Edit*: fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap $T$ accordingly. Perform step 3 above, just once.

Repeat until segmentation doesn’t change or hit iteration limit
Optimization

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 $z$:
   \[ \hat{\theta} := \arg\min_{\theta} U(\alpha, k, \theta, z) \]
3. Estimate segmentation: use min cut to solve:
   \[ \min_{\{\alpha_n: n \in T_U\}} \min_k E(\alpha, k, \hat{\theta}, z). \]
4. Repeat from step 1, until convergence.

5. Apply border matting (section 4).

User editing
- Edit: fix some pixels either to $\alpha_n = 0$ (background brush) or $\alpha_n = 1$ (foreground brush); update trimap $T$ accordingly. Perform step 3 above, just once.

Optional for this project
Energy Minimization

• Our Energy function

\[ E(\alpha, k, \theta, z) = U(\alpha, k, \theta, z) + V(\alpha, z) \]

• How can we optimize this with Min Cut?
Convert to Min Cut

- Key Idea: Convert energy function into graph

\[ E(\alpha, k, \theta, z) = U(\alpha, k, \theta, z) + V(\alpha, z) \]

- Each pixel becomes a node
- Add a “foreground” and “background” node
- Unary terms -> weighted edges between foreground/background nodes and pixel nodes
- Pairwise terms -> edges between pixels nodes
- Energy of foreground/background labeling = cost of cut
GrabCut Questions?
Outline

• GrabCut Review
• Error metrics
• Code Overview
• Project 1 Report
• Project 1 Presentations
Measuring Segmentation

Need a way to quantify that this:

is better than this:
Metric 1: Accuracy

$$100 \cdot \frac{1}{n} \sum_{i=1}^{n} [\alpha_i = f_i]$$

$\alpha_i$: estimated foreground/background label

$f_i$: ground truth foreground/background label
Problem with accuracy

Want to segment:

This has accuracy 66.7%:

This has accuracy 74.4%:
Metric 2: Jaccard Similarity

$$100 \cdot \frac{\sum_{i=1}^{n} [\alpha_i = 1 \land f_i = 1]}{\sum_{i=1}^{n} [\alpha_i = 1 \lor f_i = 1]}$$

$\alpha_i$: estimated foreground/background label

$f_i$: ground truth foreground/background label

Intersection over union of GT and predicted foreground
Accuracy vs Jaccard

Accuracy  | Jaccard similarity
----------|---------------------
66.7%     | 43.5                
74.4%     | 0                   

Accuracy vs Jaccard similarity
Outline

• GrabCut Review
• Error metrics
• Code Overview
• Project 1 Report
• Project 1 Presentations
What We Provide

• **grabcut.m**: Loads an image, gets bounding box coordinates from user click.
• **data_GT.zip**: The images
• **seg_GT.zip**: Segmentations
• **bboxes.zip**: (somewhat loose) bounding boxes for each image. (x1 y1 x2 y2)
BK_matlab

A simple Matlab Min Cut wrapper

Relevant Functions:
• `h = BK_Create(num_nodes, num_edges)`
  • *h* is a handle to the graph
• `BK_SetUnary(h, [source_weights; sink_weights])`
• `BK_SetPairwise(h, edge_weights)`
• `BK_Minimize(h)`
  • Actually do Min Cut
• `labels = BK_GetLabeling(h)`
  • Read out binary labels
• `BK_Delete(h)`
  • Delete the graph/free memory
Code Subtleties

• If Matlab, use matrix operations where possible
  • Going to be way too slow otherwise

• Handle empty clusters for GMMs and k-means

• Singular covariance matrices
  • Add e.g. 1e-8 * identity

• Do log probabilities analytically
  • i.e. don’t compute the probability and take the log
What You Need To Do

1. Implement GrabCut
   1. Initialization
   2. Assigning GMM components
   3. Fitting GMMs
   4. Setting up energy minimization + calling graph cut procedure
   5. Iterate

2. Measure performance with provided bounding boxes (accuracy and Jaccard similarity)
   • Our baseline GrabCut has average accuracy=94.5%, Jaccard=82.4, 17 sec/image. Should be able to get approximately that (maybe slower).
What You Need To Do

Q: Do I have to use the Matlab starter code?
A: No! But ask the TAs if you want to use another language (Python is ok)

Q: Do I need to turn in my code?
A: Yes. There should be a script we can call that’ll e.g. run your method on an image without any/much modification.
Outline

• GrabCut Review
• Error metrics
• Code Overview
• Project 1 Report
• Project 1 Presentations
Project 1 Report

• Write-up template provided on website (link)
• Use CVPR LaTeX template
• No more than 5 pages
• Rough sections:
  1. Overview of the field (i.e. image segmentation)
  2. The algorithm (how GrabCut works)
  3. Any changes/extensions made
  4. Code README
  5. Results
Project 1 Report

Overview of the field

• What is the problem
• What is the general scope of methods we’ve talked about in class
• Mini-summary of class papers
• Cite papers!
Project 1 Report

The algorithm

• Your understanding of how GrabCut works
• Don’t simply copy from the paper!
• Describe in a way e.g. you would have liked it to be presented, or in a way that makes the most sense to you
Project 1 Report

Your algorithm

• Did you make any changes?
  • Either simplifications or improvements
• Any details missing in the paper
  • How did you resolve them?
• Any extensions (will be reflected in results)
Project 1 Report

Code

- A README for your code
- What are the key files/functions?
- Where is the entry point?
- How can the TAs reproduce your results?
Project 1 Report

Results

- Quantitative and qualitative results
- For project 1, need average accuracy and Jaccard similarity on all 30 images
- Also show 6 images (w/accuracy and Jaccard)
  - 4 good examples, 2 bad examples
- Plots/numbers that demonstrate your extensions
Extensions

• Need at least one or two extensions (depending on scope)

• Go beyond the baseline GrabCut!

• Focus on insights!

• Plenty of possibilities
Possible Extensions

• Number of iterations of GrabCut
  • Plot performance as function of number of iterations

• Number of GMM components
  • Is more better?

• Soft labels for GMMs/do EM?
  • How much does this affect speed and performance?

• Other ways to represent probabilities
  • Histograms? Different mixture model?
Possible Extensions

- 4- or 8-pixel neighborhood
  - Does 8 improve? How much more expensive is it?
- Choice of gamma
  - Hard coded, or adapt per image?
- Alternative ways of setting beta?
  - Can you come up with a better heuristic?
- Tight or loose bounding box
  - How much does performance improve/degrade?
Possible Extensions

- Better ways of using bounding box
  - e.g. Lempitsky et al. or your own!
- Different ways of representing pixels
  - Other color space?
- Co-segmentation
  - Something like Guillaumin et al. or your own
- Other segmentation signals
  - Saliency? Center prior?
Outline

• GrabCut Review
• Error metrics
• Code Overview
• Project 1 Report
• Project 1 Presentations
Project 1 Presentations

• These happen the day before the report/code is due.

• Every team should submit 4-5 slides to me (jkrause@cs) by 5 pm the day before (Sun Apr. 19)

• Reminder: Teams of 1 or 2 people
  • If two people, make sure both present!

• Randomly pick ~10 teams to present.
Anatomy of a Presentation

Things to include:
- Any extensions you made
- Subtleties/things you didn’t expect
- A few sample results
- Any insights!
Grading

- 35%: Technical Approach and Code
  - Is your code correct? Do anything cool?
- 35%: Experimental Evaluation
  - Performance, insights, thorough evaluation
- 20%: Write-up
  - Contains everything, formatted well, etc.
  - Cite papers!
- 10%: Project Presentation
  - Clarity, Content.
Submitting

• Submit via CourseWork
• One submission per team
• We’ll use cheating-detection software
  • Please don’t make this an issue!
Piazza

- Should already be signed up for Piazza
- If not, sign up and join CS 231B
- Ask most questions there!
  - Your question may already be answered there
  - Helps other students
  - Fast response time (better than waiting for office hours)
Late Days

• You have 7, split between the three projects any way you want
• But your project presentation itself still needs to be on time (in class). Late days only apply to write-up/code submission
Working in Groups

• You can work with up to one other person
• Shared code/report.
• We’ll grade fairly regardless of team size
Important Dates

- April 19 (5 pm): Send presentations to jkrause@cs
- April 20 (in class): Presentations
- April 21 (5 pm): Reports due
Other Questions?