GrabCut

> Interactive Foreground Extraction using Iterated Graph Cuts

Foreground extraction refers to the process of employing image segmentation techniques and algorithms to extract the foreground (desired) and discard the background (undesired) part of an image or video feed. **GrabCut** is an interactive foreground extraction algorithm with minimal user interaction. It is used to accurately segment the foreground of an image from the background.

GrabCut Algorithm

1. GrabCut works by attributing a label to each pixel in the image (or image segmentation). Each pixel is assigned an alpha value based on whether it is in the foreground region (referred to as T_F where $\alpha = 1$) or the background region (referred to as T_B where $\alpha = 0$) of the image. The rest of the pixels are unknown (referred to as T_U), having $0 \leq \alpha \leq 1$, and the algorithm iteratively this α value.
2. The user starts by manually drawing a rectangular box around the region of interest in the image. Once the region is selected, everything outside the box is hard segmented as the background region (T_B assigned $\alpha = 0$), while everything in the box is unknown (T_U).
3. A gaussian mixture model (GMM) is applied over the image and this model understands the user input and starts creating labels for unknown pixel values. The unknown pixels are labelled either probable foreground or probable background depending on thier relation with the other hard-labelled pixels in terms of color statistics.
4. The problem is modeled as a task to minimize the following Gibbs energy:

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

   where,

   $$U[\alpha, k, \theta, z] = \sum_n D(\alpha_n, k_n, \theta, z_n)$$

$$D(\alpha_n, k_n, \theta, z_n) = -\log\left(p(z_n|\alpha_n, k_n, \theta)\pi(\alpha_n, k_n)\right)$$

$p(.)$ is Gaussian probability distribution, and $\pi(\cdot)$ are mixture weighting coefficients

and,

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

where C is a set of neighboring vertices and β is a constant.

5. Based on the above pixel distribution, a graph is generated where these pixels are considered as nodes. Apart from this, we also have two other nodes :

    - **Source** node : It will be connected to all the foreground pixels after mincut.
    - **Sink** node : It will be connected to all the background pixelsa after mincut.

6. The edges connecting the pixels with the source node or the sink node contains the weights (which is basically the probability of whether the pixel is a foreground one or background one). These weights between the pixels are defined by the edge information or pixel similarity. Mathematically they relate to the Gibbs Energy model as:

    - The edges between two pixel nodes are defined as the entropy term, i.e. for two neighbour nodes n and m, weight will be,

    $$\gamma\ exp(-\beta\ ||z_n - z_m||^2)$$

    - The edges between source and sink nodes respectively to the pixel nodes would be calculated from the GMM models trained for foreground and backgrounds or based on user interactions. Hence weights from source node to a pixel i would be,

    $$\text{if } i \in T_U \text{ then, } wt = -\log(score_{BG\_GMM}(i))$$

    $$\text{if } i \in T_B \text{ then, } wt = 0$$

    $$\text{if } i \in T_F \text{ then, } wt = \text{high\_weight}$$

    Hence weights from sink node to a pixel i would be,

    $$\text{if } i \in T_U \text{ then, } wt = -\log(score_{FG\_GMM}(i))$$

$$\text{if } i \in T_F \text{ then, } wt = 0$$

$$\text{if } i \in T_B \text{ then, } wt = \text{high\_weight}$$

7. Now using the mincut algorithm, we cut graph into two parts such that source and sink are in separate partitions with weights of edges cut is minimum. The weights of the edges cut will contribute to the final Gibbs Energy of the segmentation. Hence, the image is segmented into two parts.

8. Now, we have an image where the background and foreground are separated, although the separation is not completely accurate. The user can further imporove this segmentation by manually specifying parts of the image as either background, foreground, probable background, or probable foreground with the help of brush strokes. We run the entire iterative optimisation and energy minimization again after user refinement.

9. This process is repeated until we get the extracted foreground image with desired accuracy.

Model Hyperparameters

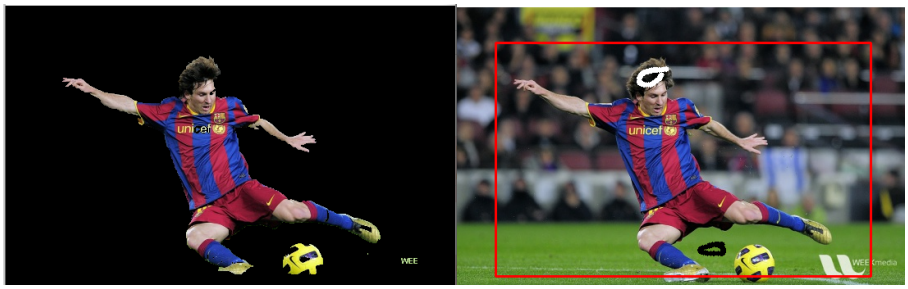
1. The parameter $\beta = \dfrac{1}{2 < \|z_m - z_n\|^2 >}$ (where z_m, z_n are the neighbouring pixels in the graph), and is calculated by using *eight-connectivity* by default.
2. The weight of the edges connecting sure background pixels to **sink** and sure foreground pixels to the **source** is set as 10^6 to prevent these edges from getting removed by the mincut algorithm.
3. To create a connection between all neighbouring pixels (even if they are at the object boundary), we omit the indicator term from the smoothness energy $V(\underline{\alpha}, \mathbf{z})$.
4. For each segmentation, we run 1 iteration of the GrabCut algorithm with $\gamma = 50$ (as our default values).

Results

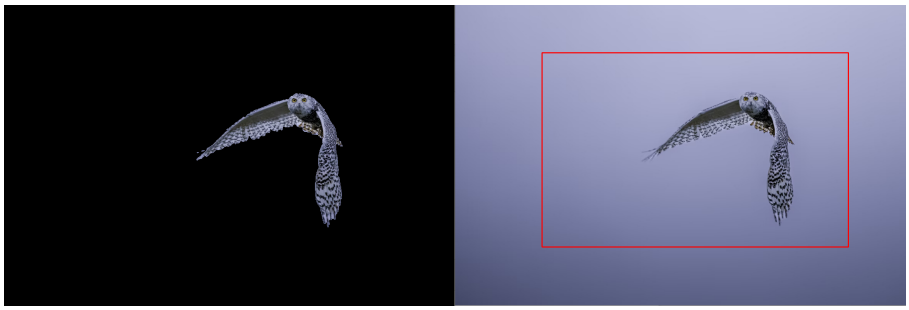
> Refer to the output folder for more predictions of our model on the testset.

In the following sections, we'll try to *qualitatively* analyze the performance of our model varying over the hyperparameters to understand their effects on the performance.
The following demonstrations show that our model (with default paramters) can handle complex scenarios with backgrounds and foreground objects pretty accurately in most situations.

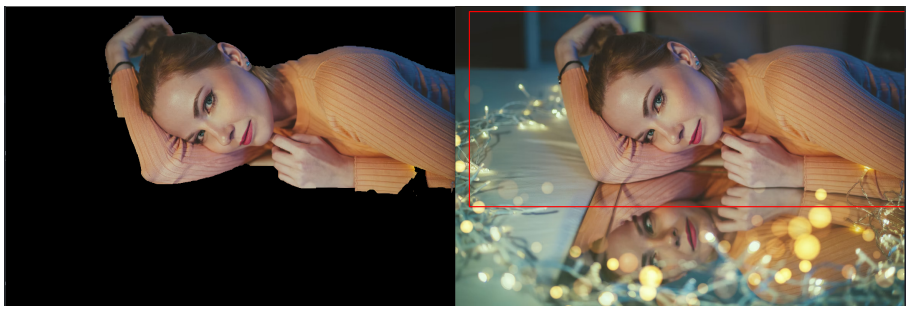
1. Footballer



2. Owl



3. Lady on a mirror

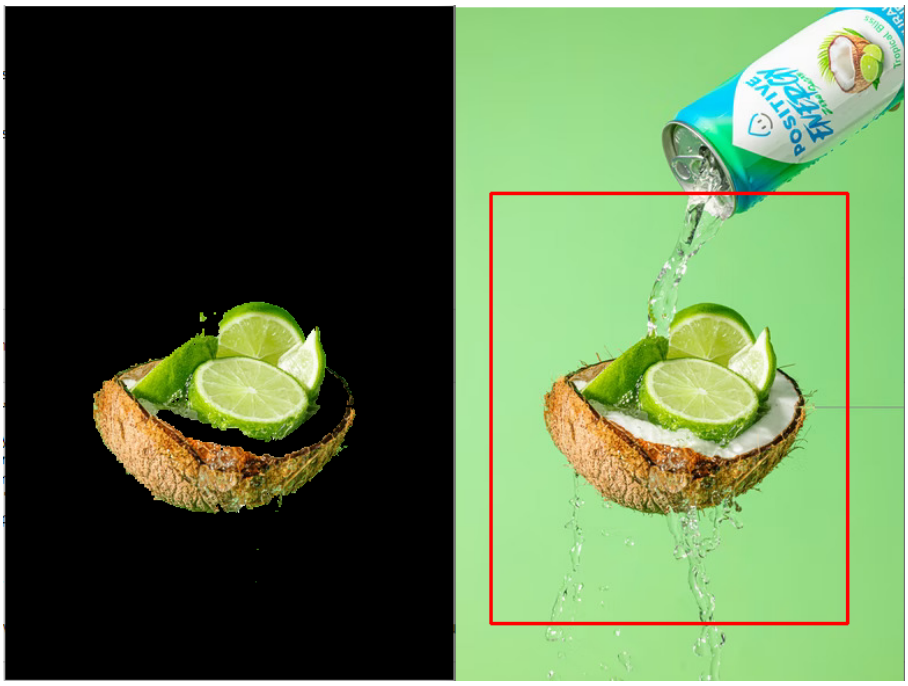


User Interaction

We'll stick with the default parameters here and try to analyze our model on the selected background and foreground pixels.

Coconut

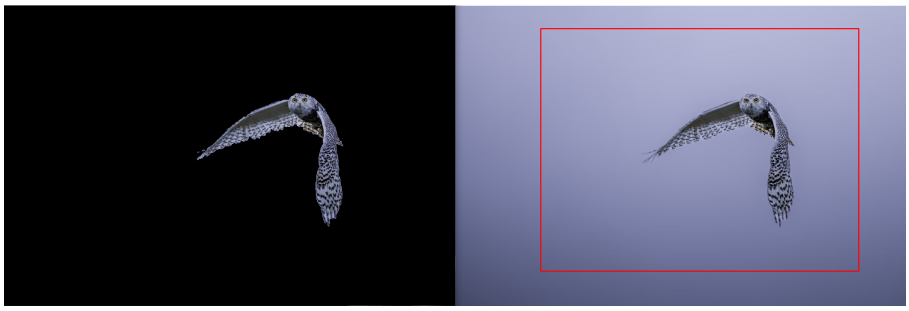
- **Without** user interaction, we observe that although our model extracts the coconut from the contrasting *green-colored background*, it isn't able to recognize the *white* portion.



- **With** user interacting to mark the *white* portion of coconut as **foreground**, we see that our model detects the whole component based on the selection. This displays the robusteness of our model on handling auxillary user input accurately.

Owl

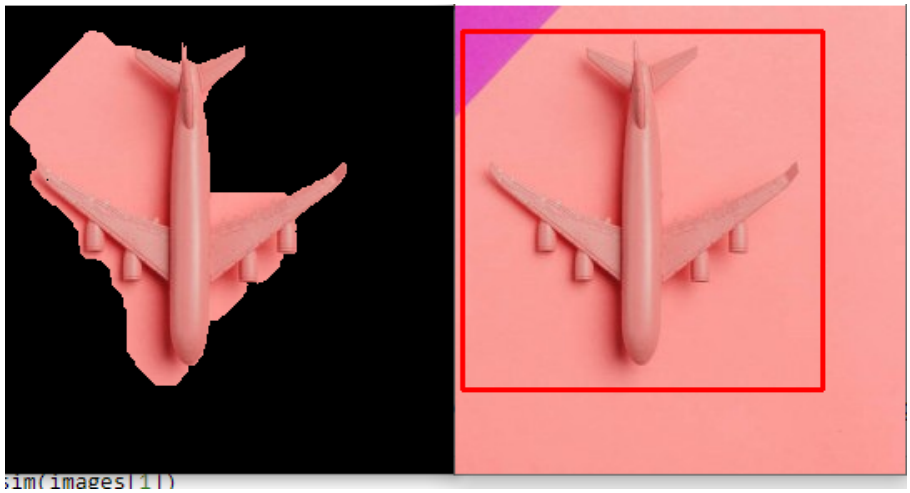
We see that our model works quite accurately without any additional user input.



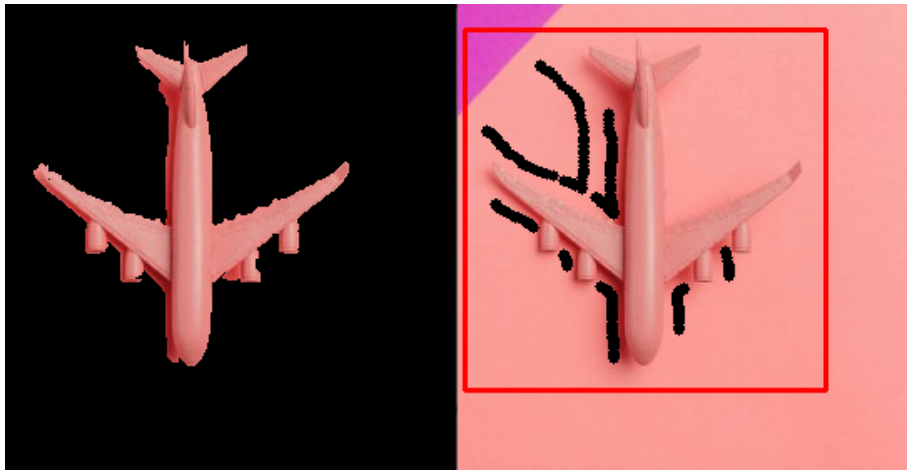
Toy Aeroplane

The given image is *challenging* owing to the similarity in colors for our foreground object and the background.

- **Without** user interaction, our model outlines the *aeroplane* loosely while inaccurately extracting the left portion of the image. This happens because of the color similarity between our object and the background.



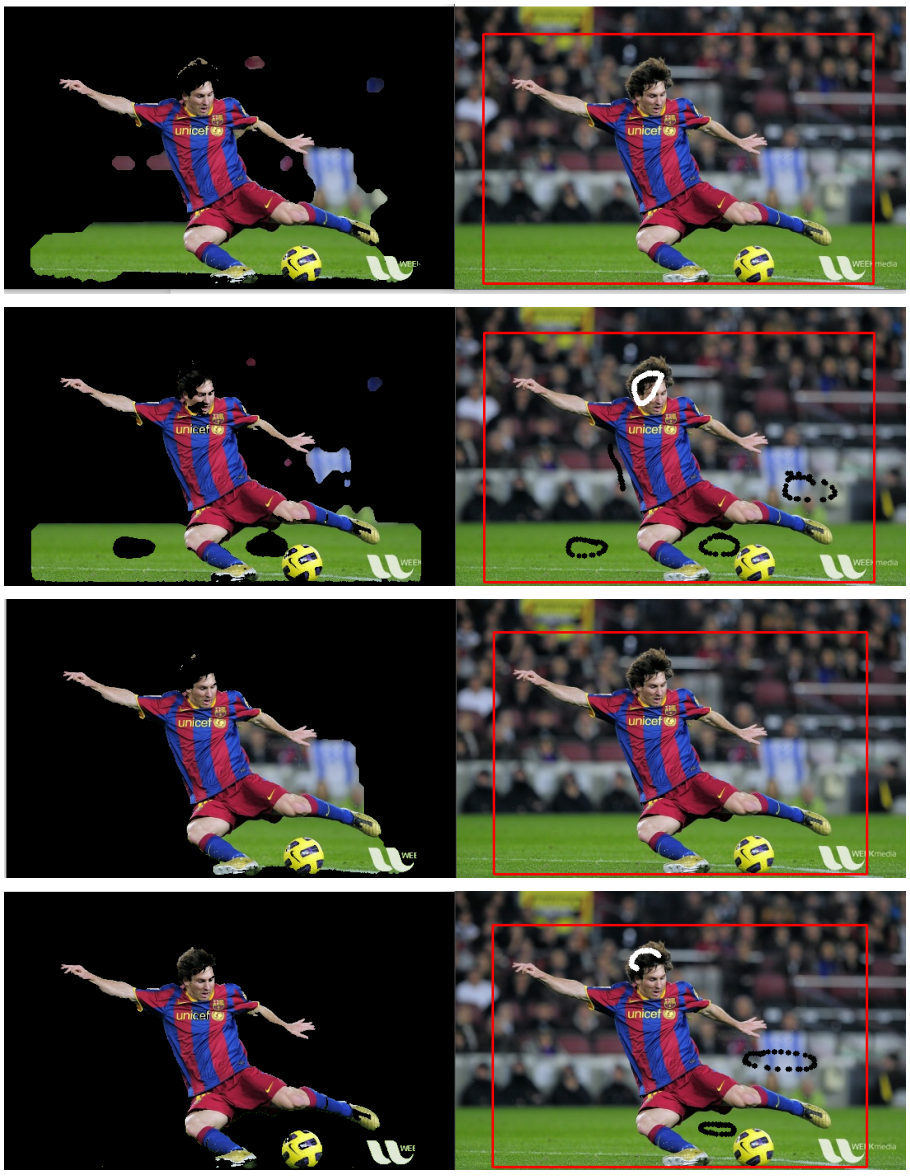
- **With** user interaction, we notice that our model still loosely fits the foreground object outline with some rugged edges.

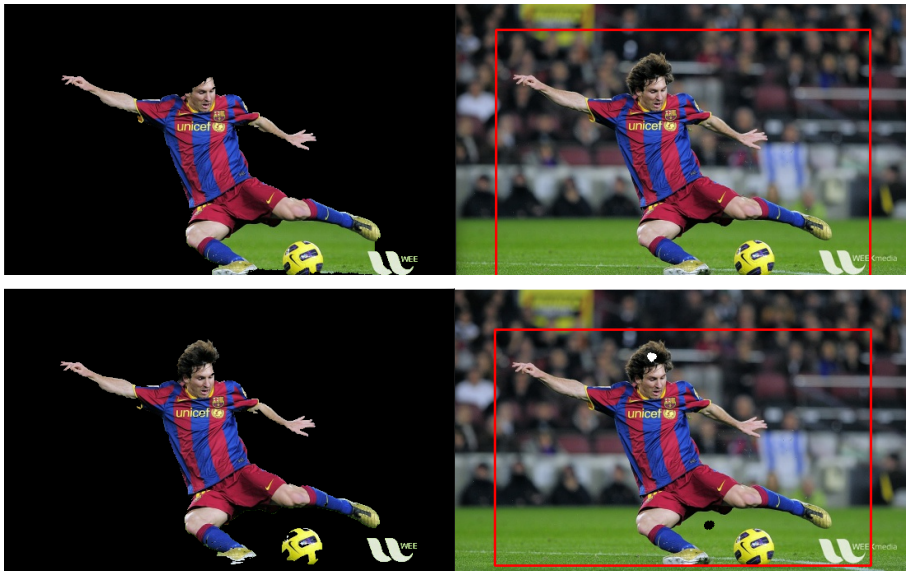


Experiments on Hyperparameters

Modifying γ

The hyperparemeter γ controls how the affinity of same coloured pixels to stay together. The following are 6 images with gamma varying in [5,15,100] in that order, and with and without interaction. We can see when gamma is 5, the even after applying background strokes to the green background, the complete backgronud did not get remove due to loose bonds between same colored pixels. Whereas, when gamma is 100 then, a minimal interaction on the hair of the player was able to recover all hair in final image, similarly a minimal interaction on background removed it completely.





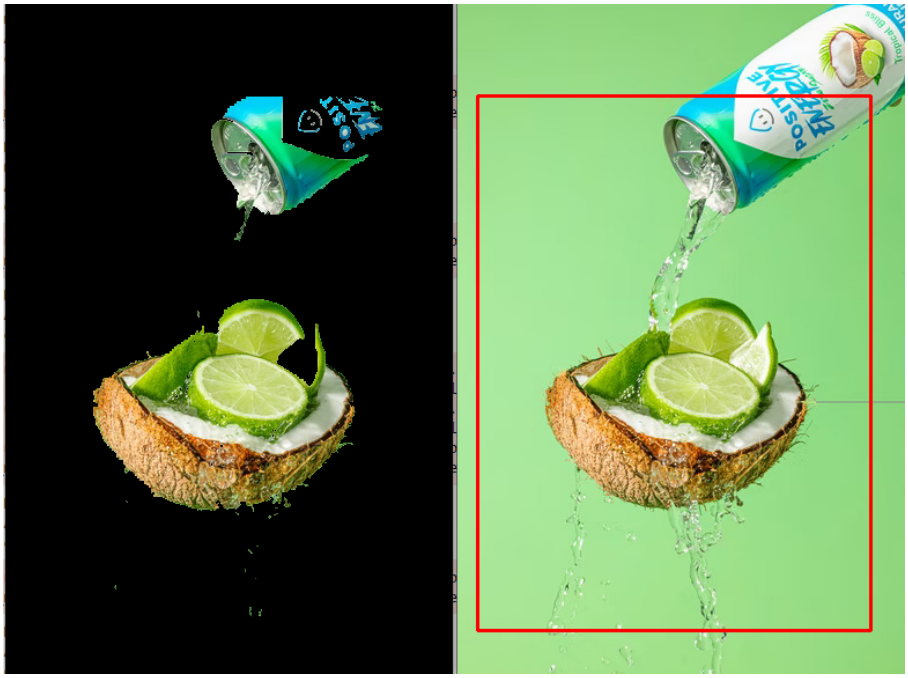
Number of GMM Components

For the **GrabCut** model, we encorporate 2 *Gaussian Mixture Models*, one for the foreground pixels and the other for the background ones. Each of these GMMs represent a distribution of colors both for the foreground and background pixels like the green background on the *coconut* test image for background while having *brown* coconut shell for the foreground along with green and white contents (with different shades) on it. Such a mixture model allows learning complex color distributions for our foreground and background pixels.
Although increasing the components for our respective mixture models gives our model power to learn more diverse foreground and background distributions, these might also give rise to competing models regarding .

Coconut

In the following comparision, we notice that having number of components equal to 1 makes our model less expressive than the image's color distribution.

$\#COMPONENTS = 1$

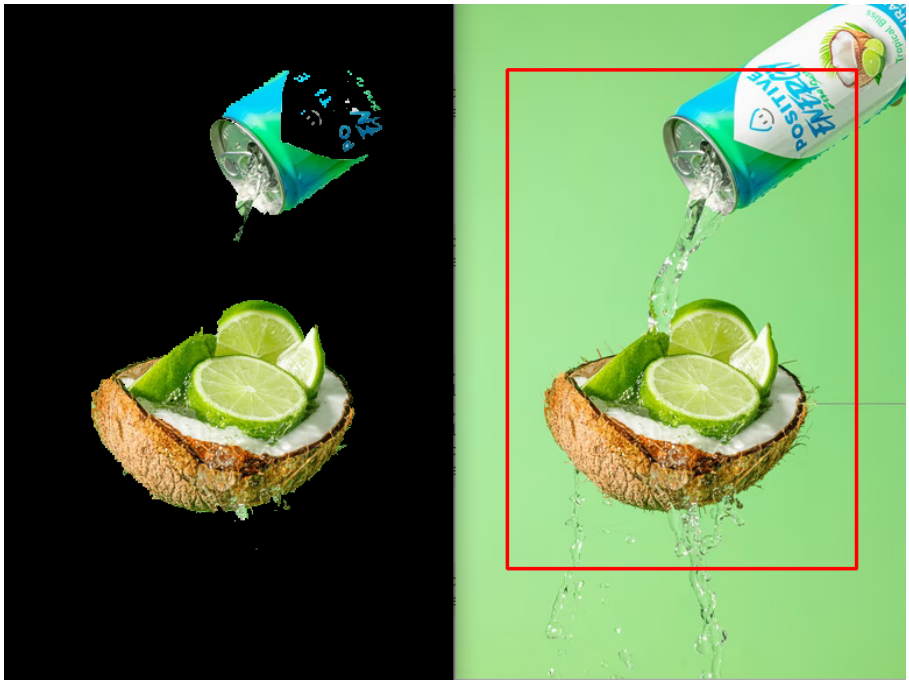


We notice that in the following experiments our model is expressive enough to model the color distributions giving rise to accurate foreground extraction.

$\#COMPONENTS = 5$

We observe no observable benefit on increasing the number of components.

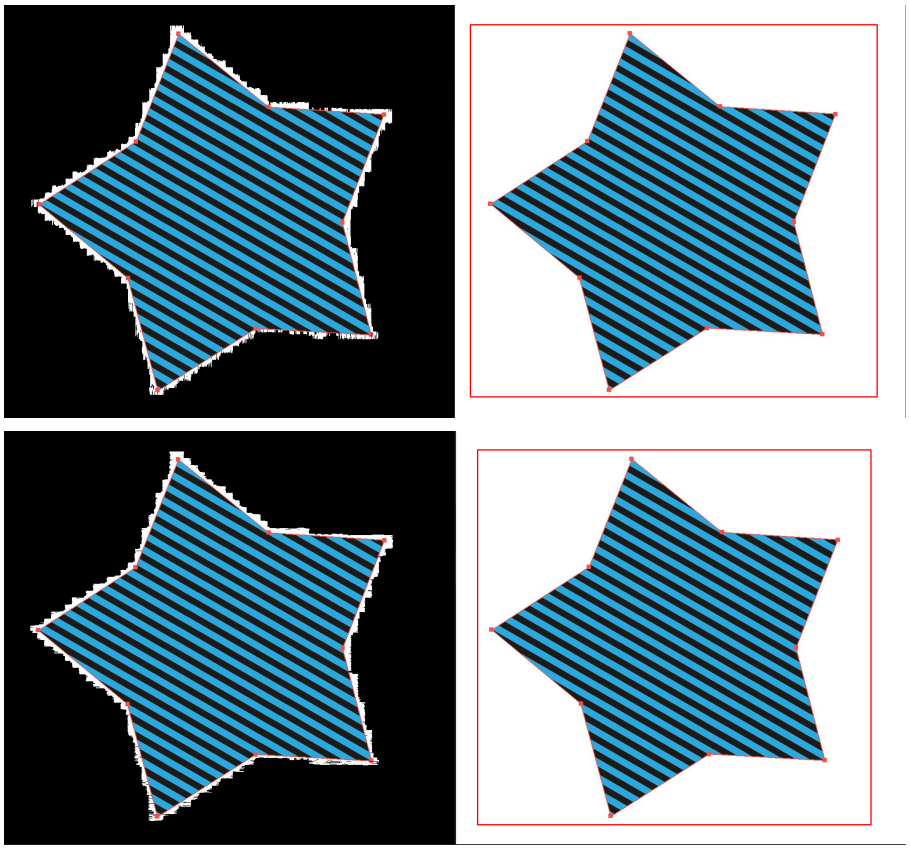
$$\#COMPONENTS = 10$$



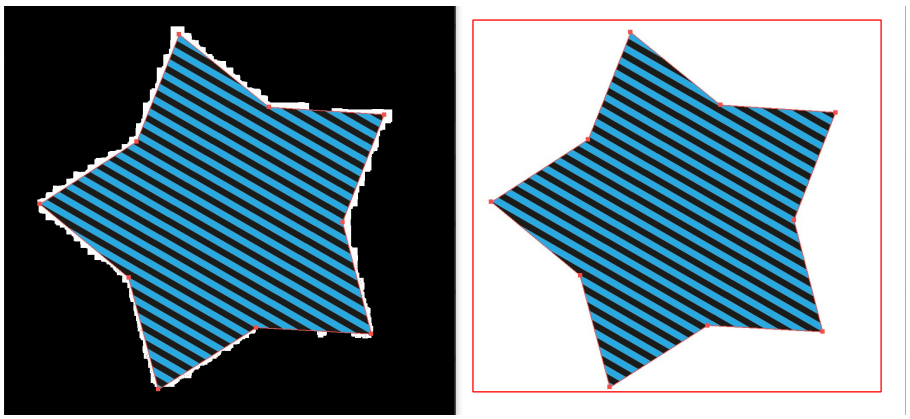
Neighbour Connectivity

The algorithm includes an entropy term which sums over neighbourhood over a pixel. We can have different ways to define this neighbourhood by by default is defined as all the pixels at one manhattan distance. We tried different variations, to define this neighbour like having only two neighbours per pixel, like only rows-wise edges, column-wise edges, 4 way connections in vertical and horizontal directions and the default 8-way connections. We can see in the below examples, many images had a problem in segmentation in the direction where the particular direction of edges were there. Like the given example, the star has diagonal borders which we could not segment smoothly if the we don't have diagonal edges in the graph.

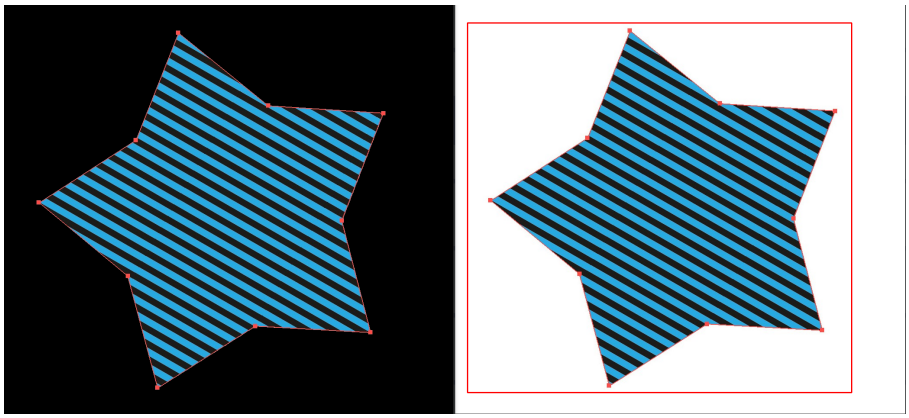
Two connectivity



Four connectivity



Eight connectivity



Work Distribution

- **Window Manager** (UI) : Vedansh and Utkarsh.
- **GMM** : Aman Rojjha.
- **Graphcut** : Bhaskar and Vedansh.
- **Grabcut** : Bhaskar and Utkarsh.
- **Experimentation**: Aman and Vedansh.
- **Report Preperation** : Aman, Bhaskar, Utkarsh, Vedansh.