Assignment 1 — Foreground Segmentation and Poisson Blending

Moshe Zeitouny (211388913) and Daniel Bar lev (211992425).

1. Introduction:

In our assignment, we implemented foreground segmentation using the Grab Cut algorithm and applied the Poisson Blending technique to integrate cut objects into provided backgrounds. Our results generally demonstrated high precision with Grab Cut. However, two primary challenges arose. Firstly, when foreground and background color palettes were similar, their Gaussian distributions also became alike, making it difficult to accurately classify pixel masks. Secondly, initial masks that substantially exceeded the desired foreground object size (e.g., Cross.jpg) led to more erroneous foreground pixel classifications.

Regarding Poisson Blending, optimal outcomes were achieved when the subject's color palette closely matched that of the target image. This alignment facilitated the algorithm's ability to subtly adjust the source image to better harmonize with its surroundings in the target image. Conversely, mismatches in colors (e.g., Banana1.jpg and table.jpg) resulted in blended images appearing washed out or shadowed due to discordant color tones.

2. Grab Cut Algorithm:

In our implementation of the Grab Cut algorithm, we utilized K-Means and Gaussian Mixture Model (GMM) algorithms from Sklearn. We initialized the GMM with low variance Gaussian components to improve segmentation accuracy. For GMM model calculations, we primarily relied on NumPy. Additionally, to solve the max-flow-min-cut problem, we utilized the I-graph library.

We changed the file structure and implemented the Grab Cut algorithm as a class with methods and static functions. This way the code is clearer and conforms with the OOP paradigm. Also, we separated constants and utility functions between files to maintain logical structure. To run our code now an object initialization is required by calling GrabCut builder with the following signature:

```
grabcut = GrabCut(image=img, initial_rect=rect, n_iter=20, gmm_components=5, min_energy_change=100,
lamda=1)
```

Followed by a call to the Grab Cut method with following signature:

```
grabcut.grabcut()
```

Below is a summary of all changes to the given template. additional help-functions were omitted for conciseness:

```
def grabcut(img, rect, n_iter=5):
```

```
def grabcut(self) -> np.ndarray:
```

def grabcut is utilizing the class structure to read its params from the class attributes.

```
def initalize_GMMs(img, mask):
```

```
def initalize_GMMs(self, trimap: tuple[np.ndarray, np.ndarray]) -> tuple[GaussianMixture, GaussianMixture]:
```

def initialize_gmms called when an object is constract, passing foreground and background binary mask to initialize the GMMs.

```
def update_GMMs(img, mask, bgGMM, fgGMM):
```

```
def update_GMMs(self):
```

def update_GMMs is utilizing the class structure to read and update its params from the class attributes.

```
def calculate_mincut(img, mask, bgGMM, fgGMM):
```

```
def calculate_mincut(self,
                     N_link_edges: np.ndarray,
                     N_link_weights: np.ndarray,
                     k) -> tuple[list[tuple[int, int]], list[tuple[int, int]]]:
```

def calculate_mincut is utilizing the class structure to calculate T-link wights and getting N-link as arguments. Eventually returning the vertices separation.

```
def update_mask(mincut_sets, mask):
```

```
@staticmethod
def update_mask(mincut_sets: tuple[list[tuple[int, int]], list[tuple[int, int]]],
                image_mask: np.ndarray) -> np.ndarray:
```

def update_mask is almost the same, just change mask name to image_mask.

```
def check_convergence(energy):
```

```
def check_convergence(self, old_pixels_count):
```

def check_convergence is utilizing the class structure to read past energy values and consider the number of updated pixels in the current iteration.

```
def cal_metric(predicted_mask, gt_mask):
```

```
@staticmethod
def cal_metric(predicted_mask: np.ndarray, ground_truth_mask: np.ndarray) -> tuple[float, float]:
```

def cal_metric is almost the same, just change gt_mask to ground_truth_mask.

In general, the project is well documented for easy editing, reading, and evaluating.

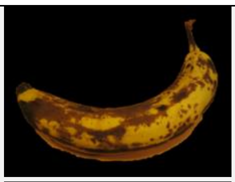
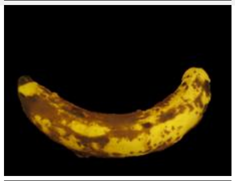
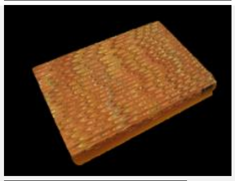
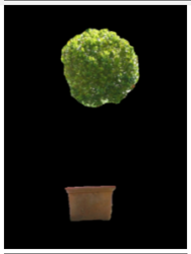
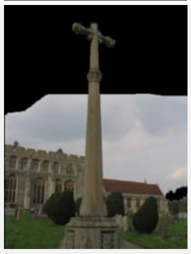
2.1 Algorithm results:

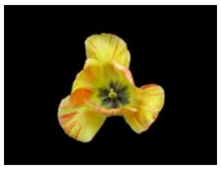
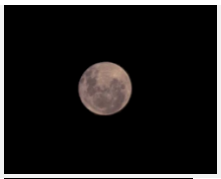
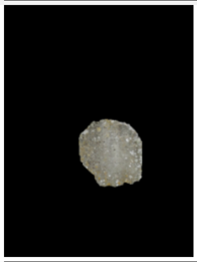
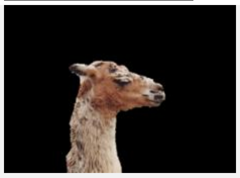
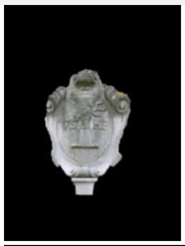
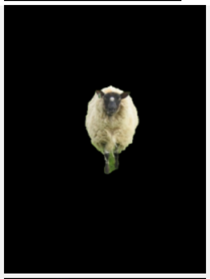
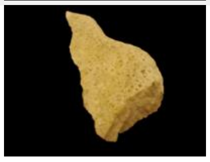
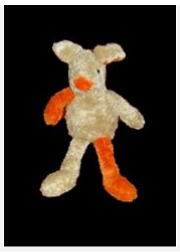
We experimented with various initialization parameters and found that using 2 components yielded the best average results. This decision improved the segmentation quality for the bush and Banana1 images (which we struggled to achieve satisfying results) but slightly reduced accuracy for the grave mask (We decided to sacrifice it for better overall accuracy). This outcome suggests that fewer Gaussian components were more effective in interpreting the smaller color variations present in the images.

In total, we achieved an average convergence time of 28.32 seconds for the given images, with an average Jaccard score of 92.69. This average run time was achieved due to a decision took in the middle of the implementation, where most of our code was implemented as for loop. When it became apparent that our running times are not optimal, we utilized cProfile library to monitor our "bottleneck" functions and NumPy fast vector operation to reduce running times substantially. (Approximately x2000 for some functions).

While most images produced good segmentation results within a reasonable time frame using our parameters, some images (like Cross) did not converge as quickly, and a few others yielded poorer results.

Table 1 – Grab Cut results

Image name	*Qualitative result*	*Accuracy*	*Jaccard*
banana1.jpg		89.03	94.19
banana2.jpg		94.51	97.18
book.jpg		94.42	97.13
bush.jpg		77.29	87.19
cross.jpg		57.05	72.65

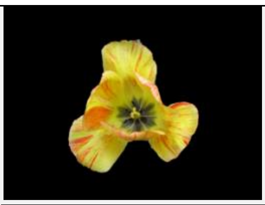
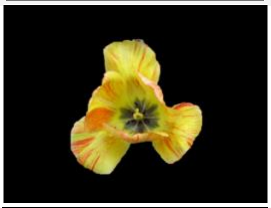
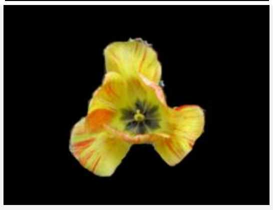
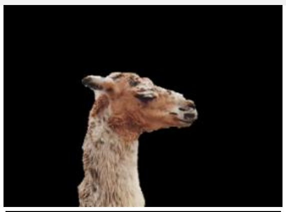
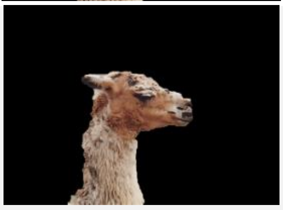
flower.jpg		98.28	99.13
fullmoon.jpg		92.50	96.10
grave.jpg		60.89	75.69
llama.jpg		93.70	96.74
memorial.jpg		92.08	95.88
sheep.jpg		92.80	96.26
stone2.jpg		98.71	99.35
teddy.jpg		95.14	97.51

As can be seen in the table, most images had good results with both Accuracy and Jaccard value above 92%.

2.1.1 Failure Cases:
The Grab Cut algorithm can indeed produce unsatisfactory results when the subject in the image, such as 'banana1.jpg', closely resembles the background in color. Moreover, if the initial bounding box around the subject includes a significant portion of the background, it may prolong the convergence time, particularly if the object's shape is not well-defined within the bounding box. These observations highlight the algorithm's sensitivity to initial conditions and color differentials between foreground and background.

2.1.2 Effects of Blur:
We compared the effects of different blur intensities on the Grab Cut algorithm. In our test we used two types of blur kernels: 13x13 and 5x5 of 1's. We also used a gaussian blur filter which produced similar results to the bigger mask, so it is not displayed here. Filtering had benefits with consistent background images but failed with very dark or very bright backgrounds. This is compatible with logic since (? Help Actual findings are presented in *Table 2.*

Table 2 - Effects of blur

Image name	*Blur*	*Result*	*Accuracy*	*Jaccard*
flower	No blur		98.09	99.03
	Low blur		97.86	98.91
	High blur		94.98	97.42
llama	No blur		93.95	96.88
	Low blur		92.36	96.03

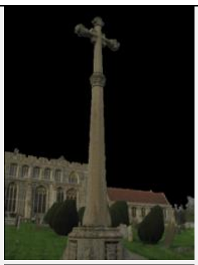
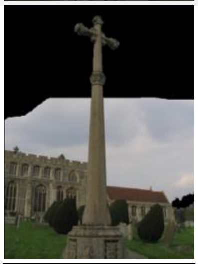
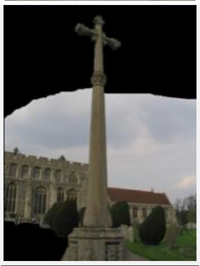
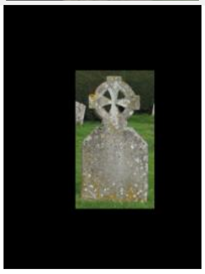
	High blur		88.99	94.17
teddy	No blur		92.82	96.27
	Low blur		96.32	98.12
	High blur		86.32	92.65

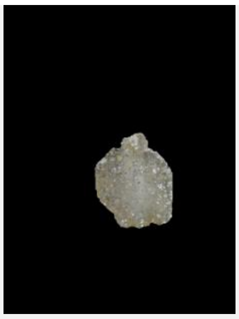
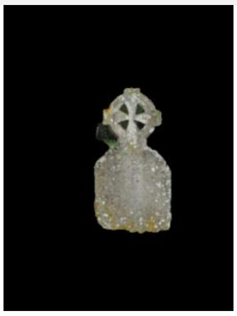
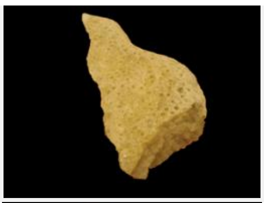
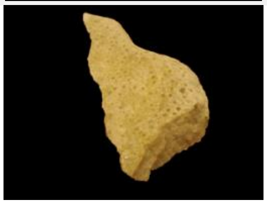
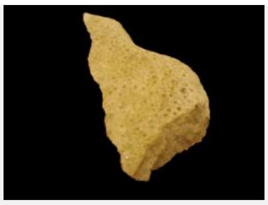
2.1.3 Effects of GMM Components Count:

More components in the GMM can be advantageous when dealing with complex color distributions in both the subject and background. They allow for a finer representation of color variations, which can improve segmentation accuracy, especially in scenarios where both foreground and background exhibit diverse colors. However, in cases where there are fewer distinct colors in either the subject or the background, using fewer components tends to yield better segmentations. This approach helps in capturing the essential color nuances more effectively, leading to clearer distinctions between foreground and background regions. Therefore, the choice of the number of components in the GMM should be tailored to the specific color complexities present in the image being segmented.

Actual findings are presented in *Table 3*.

Table 3 - Effects of GMM components count

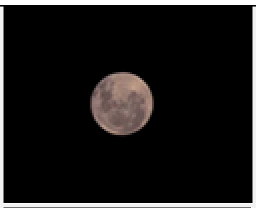
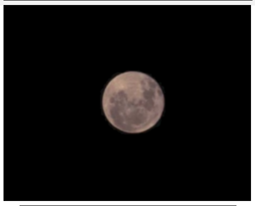
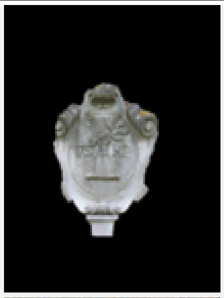
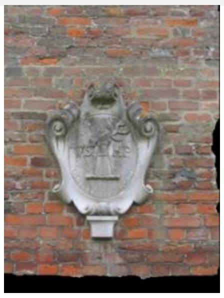
Image name	*Components*	*Result*	*Accuracy*	*jaccard*
cross	1		96.91	98.43
	2		57.34	72.89
	5		48.81	65.60
grave	1		57.01	72.62

	2		64.31	78.27
	5		89.82	94.63
Stone2	1		98.67	99.33
	2		98.67	99.33
	5		98.66	99.33

2.1.4 Effects of Bounding box initialization:
We conducted a comparison of the effects of the initial bounding box size on the Grab Cut algorithm. Our findings indicate that using a tighter bounding box around the object generally yields better segmentation results and improves the convergence time. Our test shows that even with a simple image like fullmoon, the box initialization is critical for the final result.

Actual findings are presented in *Table 4*.

Table 4 - Effects of box initialization

Image name	*Box*	*Result*	*Accuracy*	*Jaccard*
fullmoon	Tight box		92.5	96.1
	Loose box		85.76	92.33
memorial	Tight box		92.08	95.88
	Loose box		19.81	33.07

3. Poisson blending:
Our implementation of the Poisson Blending algorithm follows the formula discussed in class. We compute the Laplacian operator (matrix A) and the current state vector (vector b), and then solve the Poisson equation using the SciPy library. Our results vary in quality; some appear quite realistic and plausible, while others seem mismatched with both the source and target images.

3.1 Blending results:
To present the blending outcomes, we used the true masks of the provided images and blended them with the background images. We noticed varying levels of success: results were generally better when using the grass mountains background but less satisfactory when using the table background.

Upon observing the outcomes, it becomes apparent that pasting an image onto a light background can cause it to appear unclear or even overexposed due to the blending technique that changes the source colors to match the background. Additionally, as seen in the bush image, the color of the pasted subject seems to be blended with the background color for the same reason**.**

Table 5 - Poisson Blending results

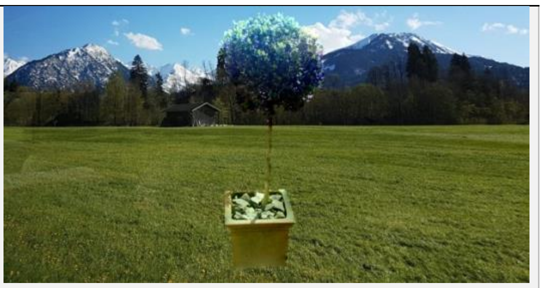
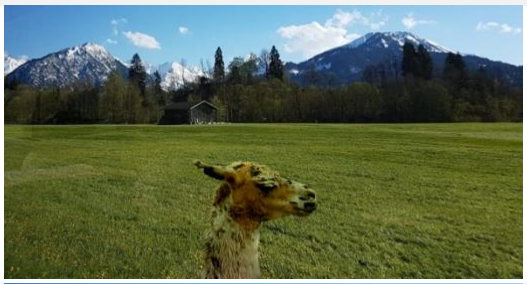
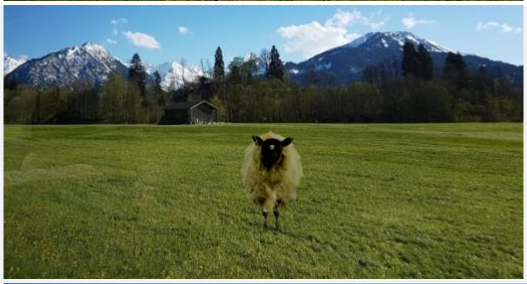
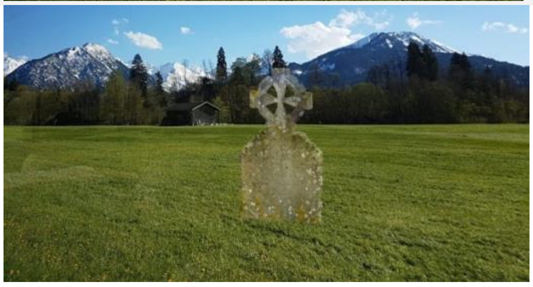
Target image	*Source image*	*result*
Grass mountains	Bush	
	Llama	
	Sheep	
	Grave	

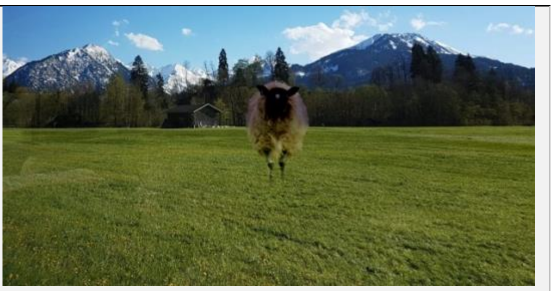
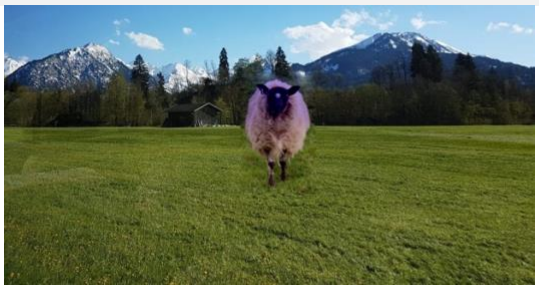
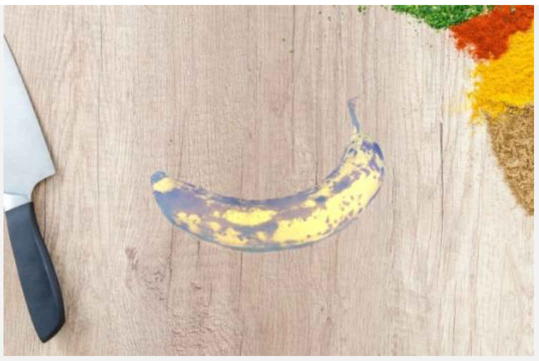
table	Banana1	
	Teddy	
	Banana2	
	Book	
wall	Cross	

	Stone2	
	Flower	
	Fullmoon	
	Memorial	

3.2 Effects of Mask Tightness:

We compared the effect of a tight mask around the object vs a simple box by running the algorithm on the sheep image as source and the grass mountains as target. Once using the tight true mask and once with the box for the Grab Cut algorithm. We found that when using a box around the object, a slight tint is caused to the subject, but the image looks a bit better. This is probably because there is additional weight to the source colors.

Table 6 - Effects od mask tightness

Target, source	*Mask type*	*result*
Grass mountains, sheep	True mask	
	Box mask	
Table, banana1	True mask	
	Box mask