Group two Project Report

Hossein Goli, Bahar Dibaeinia, Siavash Rahimi, Mahdi GhaemPanah, Ali Ansari
Department of Computer Engineering
Sharif University of Technology
Tehran, Iran

I. GHS Algorithm

In the first part of the project, we introduce GHS algorithm, a **graph-based unsupervised** method for getting **adversarially robust** for getting image segmentations. First, we will briefly discuss the preliminaries related to graph theory and unsupervised local clustering algorithms.

A. Preliminaries

1) DSU Data Structure: This data structure provides the following capabilities. We are given several elements, each of which is a separate set. A DSU will have an operation to combine any two sets, and it will be able to tell in which set a specific element is. The classical version also introduces a third operation, it can create a set from a new element.

Thus the basic interface of this data structure consists of only three operations:

$makeset(v)$ - creates a new set consisting of the new element v $union_sets(a, b)$ - merges the two specified sets (the set in which the element a is located, and the set in which the element b is located) $findset(v)$ - returns the representative (also called leader) of the set that contains the element v. This representative is an element of its corresponding set. It is selected in each set by the data structure itself (and can change over time, namely after $unionsets$ calls). This representative can be used to check if two elements are part of the same set or not. a and b are exactly in the same set, if $findset(a) == findset(b)$. Otherwise, they are in different sets.

2) Unsupervised Clustering Algorithms: Unsupervised clustering algorithms, such as K-means, hierarchical clustering, and DBSCAN, uncover hidden patterns within data without the need for labeled examples. These algorithms partition data points into distinct groups based on similarity measures, enabling insights into natural structures and relationships. By iteratively organizing data into clusters, they facilitate exploratory analysis, anomaly detection, and data compression, playing a crucial role in various fields like customer segmentation, image processing, and anomaly detection. Despite their autonomy from labeled data, fine-tuning parameters and interpreting results remain pivotal challenges in ensuring the accuracy and interpretability of clustering outcomes.

B. SuperPixel Segmentation

Superpixel segmentation algorithms offer a powerful approach to partitioning images into perceptually meaningful regions. Unlike traditional pixel-based methods, these algorithms group pixels into coherent and homogeneous clusters, known as superpixels, based on color, texture, and spatial proximity. By reducing the complexity of image data while preserving essential features, superpixel segmentation facilitates efficient processing tasks such as object recognition, image compression, and boundary delineation.

One of the popular techniques in this domain is the Simple Linear Iterative Clustering (SLIC) algorithm. SLIC efficiently generates superpixels by iteratively assigning each pixel to the nearest cluster centroid in a feature space that combines color and spatial information. This approach ensures spatial coherence while maintaining computational efficiency, making it suitable for real-time applications.

Superpixel segmentation algorithms find wide application across various domains, including medical imaging, remote sensing, and computer vision. They enable more robust and accurate analysis by providing a compact representation of image content and reducing computational overhead. However, challenges such as parameter tuning and boundary adherence remain areas of ongoing research to further enhance the effectiveness of superpixel segmentation in diverse applications.

The method we used is as follows:

The Simple Linear Iterative Clustering (SLIC) algorithm is a popular method for superpixel segmentation, offering a balance between computational efficiency and segmentation quality. SLIC operates by iteratively grouping pixels into compact and visually coherent clusters, known as superpixels. Let's delve into its mathematical underpinnings and operational steps:

1) **Initialization:** SLIC begins by initializing cluster centers in a regular grid pattern over the image. The spacing between these initial cluster centers, termed as the "step size" or "superpixel size," is a crucial parameter affecting the segmentation result.
2) **Assignment of Pixels to Clusters:** For each cluster center, SLIC assigns nearby pixels to that cluster based on both spatial proximity and color similarity. This assignment is governed by a distance metric in a combined

color-space and spatial domain. The distance metric can be formulated as:

$$D = d_{\text{color}} + \frac{m}{S} \cdot d_{\text{spatial}}$$

Here, d_{color} represents the color distance between a pixel and a cluster center in a suitable color space (e.g., LAB or RGB). d_{spatial} denotes the Euclidean distance between pixel coordinates and cluster center coordinates. S represents the step size (superpixel size), and m is a constant controlling the influence of spatial distance on the overall distance calculation.

3) **Update Cluster Centers:** After assigning pixels to clusters, SLIC updates the cluster centers by computing the mean color and position of all pixels assigned to each cluster. This step ensures that the cluster centers converge towards the true boundaries of image regions.
4) **Iterative Refinement:** SLIC iterates the assignment and update steps until convergence criteria are met. Typically, the algorithm converges after a fixed number of iterations or when the change in cluster centers falls below a predefined threshold.
5) **Post-processing:** Optionally, a boundary refinement step may follow to improve the coherence of superpixel boundaries, ensuring smoother segmentation results.

Benefits and Applications: This method offers several advantages, including computational efficiency, parameter control, and high-quality segmentation results. Its ability to generate compact superpixels makes it well-suited for applications such as image segmentation, object tracking, and image compression.

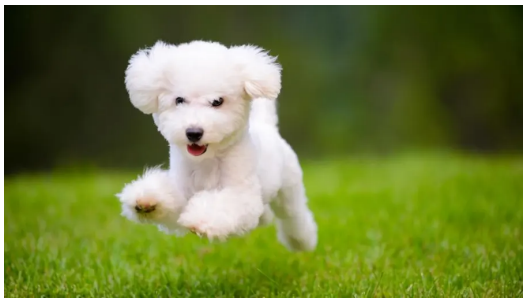


Fig. 1: Original Image

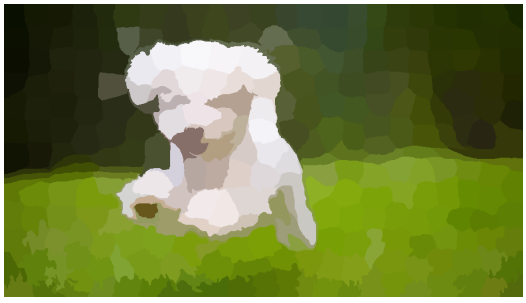


Fig. 2: Partioned-Average Image

Fig. 3: Partioned-Average Image with boundaries

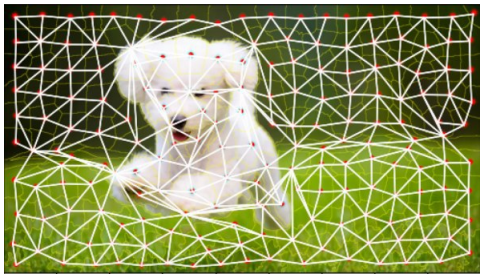


Fig. 4: Graph created using our method

C. Proposed Method

We propose a novel method for expanding locally group pixels to global groups. This method can be interpreted for many applications such as segmentation, finding background from the foreground, finding more global superpixels, Tiling images, and so on. In our method, we first find local clusters within an image as shown in Fig. 3. Then we propose the average for each group to get Fig. 2 which is a representation for each group using its mean color. We note that each group had similar pixels thus doing so is reasonable. The novelty of our idea is that we transform the image at this stage to a **Graph** and use algorithms from Graph Theory to maximize our objective function. The created graph vertices are the local cluster centers and each edge is between neighboring grouped pixels and its weight is a $f(x, y, r, g, b)$ in which the simplest case would simply be a 3-dimensional color space. we note that using different f will result in different performance for different applications and leave it as a future work.

The next step of our method is to join these different sets in a graph effectively. A famous approach to this problem is using the Disjoint-set data structure which was discussed in the previous parts. Note that these steps can and should be done iteratively to find better and better global groups at each time step and thus a scheduler is required to relax the DSU joining threshold over time. You can find the result of our method in

It is worth noting that our method is robust to noise. The theoretical reason for it is simple. each time we average noise in these clusters, their variance is divided by a factor of $\sqrt{A}$ (size of the cluster). Thus in each iteration, the perturbation is becomes weaker and weaker and our method initially doesn't need to find large clusters and finding just the right amount

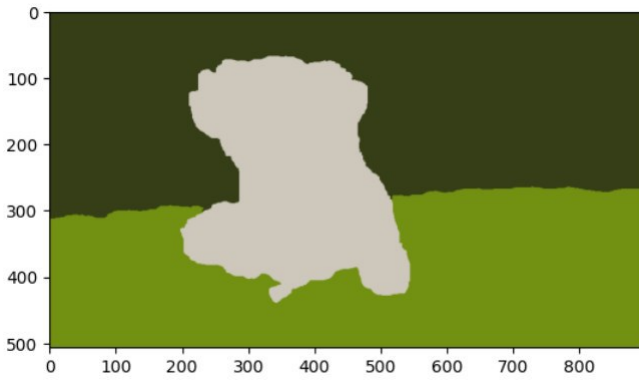


Fig. 5: GOur proposed method for segmentation

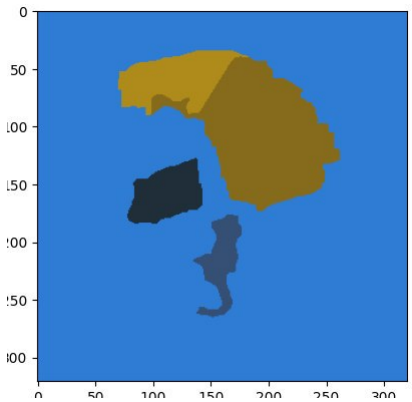


Fig. 6: Our proposed method for segmentation

of small yet local clusters will give it a good starting point.

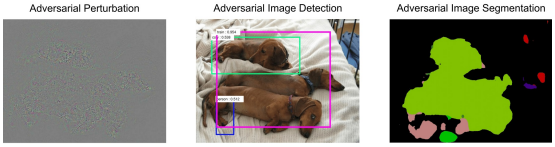
II. Tiny NeRF

We have implemented a simplified version of **Neural Radiance Fields**, a method that achieves state-of-the-art results for synthesizing novel views of complex scenes by optimizing an underlying continuous volumetric scene function using a sparse set of input views. The algorithm represents a scene using a **Multilayer Perceptron** network, whose input is the spatial location (x, y, z) and whose output is the volume density and emitted radiance at that spatial location.

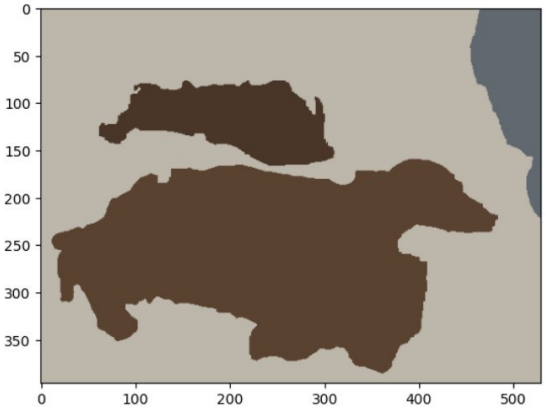
A. Background

Advancements in 3D scene representation have seen a significant shift from traditional geometric representations, such as meshes and voxel grids, to more advanced neural network-based approaches. Notably, Park et al.'s DeepSDF [1] and Mescheder et al.'s Occupancy Networks [2] have pioneered in demonstrating the effectiveness of implicit neural representations for encoding complex geometrical details. These methods utilize Multi-Layer Perceptrons (MLPs) to model continuous volumetric fields, offering enhanced precision in rendering intricate scenes.

In the realm of view synthesis, earlier techniques primarily relied on light field interpolation [3], [4]. Another



(a) Aversarial attack on SegNet



(b) Our method being robust to adversarial perturbations

Fig. 7: Creating subfigures in LaTeX.

class of methods use volumetric methods for photorealistic view synthesis from input RGB images realistically represent complex shapes and materials, are suitable for optimization, and produce fewer artifacts than mesh-based methods. Early methods directly colored voxel grids [5] using observed images, while recent approaches train deep networks on large datasets to predict and render volumetric representations from input images [6].

B. Neural Radiance Field Scene Representation

In the original NeRF approach, a continuous scene was represented as a 5D vector-valued function. The input of the function is a 3D location $x = (x, y, z)$ and 2D viewing direction (θ, ϕ), which is typically represented by a 3D Cartesian unit vector d. The output of the function is an emitted color $c = (r, g, b)$ and volume density σ. However, in the simplified version, Tiny NeRF, the input is only the 3D location. This is approximated by an MLP network $F_\theta : (x) \rightarrow (c, \sigma)$, where x represents the 3D location, and the output is the emitted color c and volume density σ.

Positional Encoding: Deep networks are biased towards learning lower frequency functions [7]. Mapping the inputs to a higher dimensional space using high frequency functions before passing them to the network enables better fitting of data that contains high frequency variation. We reformulate F_theta as a composition of two functions $F_\theta = F'_\theta \circ \gamma$. Here γ is a mapping from $\mathbb{R}$ into a higher dimensional space $\mathbb{R}^{2L}$, and F'_θ is the a regular MLP.

$\gamma(p) = (sin(2^0\pi p), cos(2^0\pi p), ..., sin(2^{L-1}\pi p), cos(2^{L-1}\pi p))$

We set $L = 6$. This function is applied separately to each of the three coordinate values in x.

Model: The MLP F'_θ consists of 4 fully-connected layers with $ReLU$ activations and 256 channels per layer. Its input is the 39-dimensional positional encoding of the 3D coordinate, x.

C. Volume Rendering with Radiance Fields

We render the color of any ray passing through the scene using principles from classical volume rendering [8]. The volume density $\sigma(x)$ can be interpreted as the differential probability of a ray terminating at an infinitesimal particle at location x. The expected color C(r) of camera ray $r(t) = o+td$ with near and far bounds t_n and t_f is:

$$C(r) = \int_{t_n}^{t_f} T(t)\sigma(r(t))c(r(t))dt$$

where $T(t) = exp(-\int_{t_n}^{t} \sigma(r(s))ds)$. We numerically estimate this continuous integral using quadrature. We partition the interval $[tn, tf]$ into N evenly-spaced bins and draw one sample uniformly at random from within each bin and use these samples to estimate C(r) with the quadrature rule discussed in the volume rendering review by Max [9].

$$\hat{C}(r) = \sum_{i=1}^{n} T_i(1 - exp(-\sigma_i\delta i))c_i$$

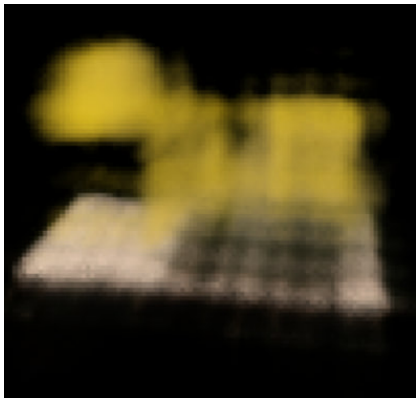
where $Ti = exp(-\sum_{j=1}^{i-1} \sigma_j\delta j)$.

D. Implementation and Results

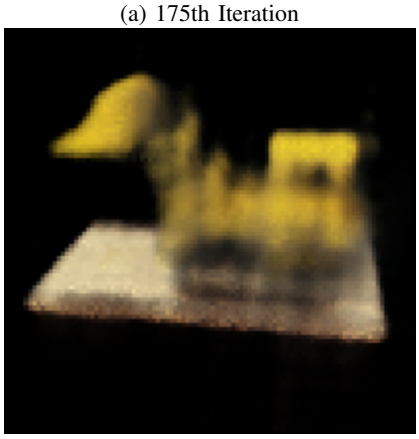
We optimize a separate neural continuous volume representation network for each scene. This requires only a dataset of captured RGB images of the scene, the corresponding camera poses and intrinsic parameters, and scene bounds. At each optimization iteration, we randomly select an image from our dataset and sample $N = 64$ points from its camera rays, corresponding to each pixel. We then use the described volume rendering procedure to render the color of each ray from our samples. Our loss is simply the total squared error between the rendered and true pixel colors:

$$l = \sum_{r\in R} ||\hat{C}(r) - C(r)||_2^2$$

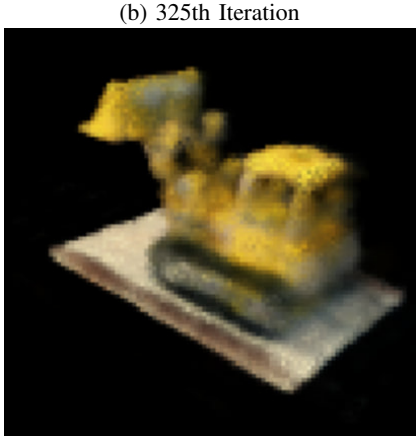
R is the set of rays of the image and $C(r)$, $\hat{C}(r)$ are the ground truth and volume predicted RGB colors for ray r respectively. We use the Adam optimizer with a learning rate of 5×10^{-3}. Our dataset is Tinynerf Lego Example. The optimization for a single scene typically take around 1000 iterations to converge on a single T4 GPU (Fig. 8).



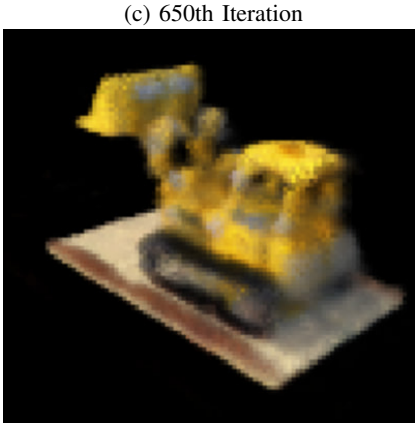
(a) 175th Iteration



(b) 325th Iteration



(c) 650th Iteration



(d) 975th Iteration

Fig. 8

REFERENCES

[1] Park, J.J., Florence, P., Straub, J., Newcombe, R., Lovegrove, S.: DeepSDF: Learning continuous signed distance functions for shape representation. In: CVPR (2019)

[2] Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., Geiger, A.: Occupancy networks: Learning 3D reconstruction in function space. In: CVPR (2019)

[3] Cohen, M., Gortler, S.J., Szeliski, R., Grzeszczuk, R., Szeliski, R.: The lumigraph. In: SIGGRAPH (1996)

[4] Davis, A., Levoy, M., Durand, F.: Unstructured light
elds. In: Eurographics (2012)

[5] Kutulakos, K.N., Seitz, S.M.: A theory of shape by space carving. International Journal of Computer Vision (2000)

[6] Flynn, J., Broxton, M., Debevec, P., DuVall, M., Fyffe, G., Overbeck, R., Snavely, N., Tucker, R.: DeepView: view synthesis with learned gradient descent. In: CVPR (2019)

[7] Rahaman, N., Baratin, A., Arpit, D., Dr. axler, F., Lin, M., Hamprecht, F.A., Bengio, Y., Courville, A.C.: On the spectral bias of neural networks. In: ICML (2018)

[8] Kajiya, J.T., Herzen, B.P.V.: Ray tracing volume densities. Computer Graphics (SIGGRAPH) (1984)

[9] Max, N.: Optical models for direct volume rendering. IEEE Transactions on Visualization and Computer Graphics (1995)