1 Methods

1.1 Color

Color segmentation is very effective as it enables the differentiation of objects based on the fact that most objects are not the same as the background of the image they are in front of. This segmentation strategy works very well when objects are well defined have have distinct color differences/edges between them.

Color segmentation works less well when objects are camouflaged in with the background or have similar colors to other objects or the background of the image.

1.2 Position

Adding position segmentation stems from the idea that pixels that are close together are likely to come from the same object and so should be grouped together. This segmentation works well on objects that have clear boundaries but might have more similar colors that color segmentation alone would struggle with.

Position segmentation does not work well when objects that should be clustered together are not near each other in the image, like when an object is obscured by another and so gets split into multiple parts in around the object that is in front of it.

1.3 Edges

Adding edge segmentation enables the detection of outlines of objects that can be used to bound segmentation as most segmentation should occur near or on the edge of an object in the image. This works well when there are clear, sharp edges in an image that can be calculated.

Edge segmentation will struggle when it is faced with blurred/smoothed images, or things without clear edges in them as the binary map of edges will either not contain any or be noisy and not accurate therefore skewing segmentation when using it.

1.4 Gradients

Gradient segmentation helps to fix the shortcoming of edge segmentation when faced with smoothing that removes some sharp edges. By detecting gradients, the segmentation algorithm can find contiguous objects that have very little gradient. It can also better find edges of objects that are not sharp or have been smoothed as even smoothed edges have a steep gradient that will get detected.

Adding gradient segmentation increases the run-time of the algorithm by a decent amount. In addition, it can occasionally bring down accuracy if fewer transforms were already working well to segment an image by undoing color/position/edge segmentation that was correct due to a gradient.

1.5 Feature Normalization

We normalize the feature vectors by forcing them all to have a mean of 0 and a standard deviation of 1. This fixes uneven scaling between different types of features allowing them to mesh together more easily. The equations for accomplishing this normalization strategy are:

$$\mu_j = \frac{1}{n}\sum_{i=1}^{n} f_{ij}$$

$$\sigma_j^2 = \frac{1}{n-1}\sum_{i=1}^{n}(f_{ij} - \mu_j)^2$$

$$\tilde{f_{ij}} = \frac{f_{ij} - \mu_j}{\sigma_j}$$

2 Visualizations

Below are several visualizations of segmentations produced by our algorithms – both successful and unsuccessful. In the evaluation section of this paper, we will analyze the primary reasons that cause these segmentations to succeed or fail.

2.1 Successful Segmentations

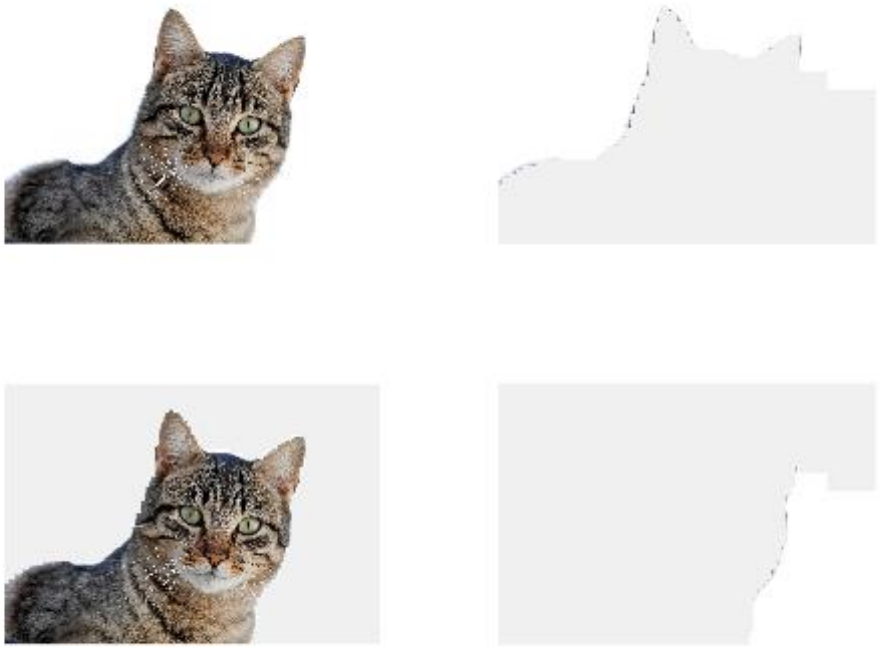

Figure 1: `cat_march.jpg`, using HAC with $k = 3$, position + color features, feature normalization, and a resize factor of 0.025.

Figure 2: `Cat_Bed.jpg`, using k-means clustering with $k = 4$, position + color features, and feature normalization.

Figure 3: `black_kitten_star.jpg`, using k-means clustering with $k = 3$, color features, and no feature normalization.

2.2 Unsuccessful Segmentations

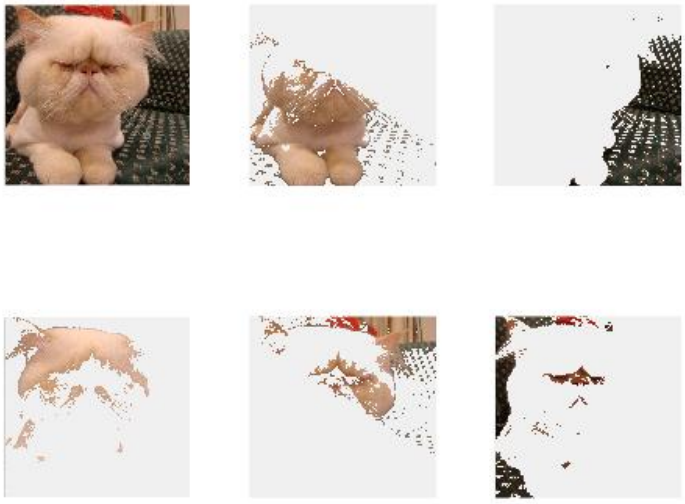

Figure 4: `cat_grumpy.jpg`, using k-means clustering with $k = 5$, position + color features, and no feature normalization.

Figure 5: `cat-jumping-running-grass.jpg`, using k-means clustering with $k = 3$, color features, and feature normalization.

Figure 6: `kitten16.jpg`, using HAC with $k = 3$, color features, feature normalization, and a resize factor of 0.25.

2.3 Composite Images

Using the script titled `GrabCat.m`, we were able to produce composite images by transferring segments from one image to another background image. This allowed us to create the two composite images shown below.

Figure 7: Input: `black_kitten_star.jpg, desert.jpg`, using k-means clustering with $k = 3$, color features, and feature normalization.

Figure 8: Input: `black_kitten.jpg, beach.jpg`, using HAC with $k = 5$, color features, feature normalization, and a resize factor of 0.2.

3 Evaluation

We evaluated in detail the effect of varying each of the segmentation parameters – feature transform, feature normalization, clustering method, number of clusters, and the resize. Our results can be seen in the table below.

Feature Transform	**Feature Normalization**	**Clustering Method**	**Number of Clusters**	**Resize (Max Pixels)**	**Mean Accuracy**
Color	Yes	K-Means	3	50000	.8341
Color	Yes	K-Means	5	50000	.8736
Color	Yes	K-Means	7	50000	.8795
Color	Yes	K-Means	15	50000	.9087
Color	Yes	K-Means	30	50000	.9228
Color	No	K-Means	5	50000	.8680
Color/Position	Yes	K-Means	5	50000	.8765
Color/Position	No	K-Means	5	50000	.8802
Color/Edges	Yes	K-Means	5	50000	.7991
Color/Edges	No	K-Means	5	50000	.8670
Color/Gradients	Yes	K-Means	5	50000	.7905
Color/Gradients	No	K-Means	5	50000	.8775
Color/Position/Edges	Yes	K-Means	5	50000	.7951
Color/Position/Edges	No	K-Means	5	50000	.8800
Color/Position/Edges/Gradients	Yes	K-Means	5	50000	.7924
Color/Position/Edges/Gradients	No	K-Means	5	50000	.8866
Color	Yes	HAC	5	1000	.8623
Color	Yes	HAC	3	1000	.8340
Color	Yes	HAC	7	1000	.8691
Color	Yes	HAC	15	1000	.8906
Color	Yes	HAC	30	1000	.9123
Color	No	HAC	5	1000	.8585
Color/Position	Yes	HAC	5	1000	.8531
Color/Position	No	HAC	5	1000	.8585
Color/Edges	Yes	HAC	5	1000	.9288
Color/Edges	No	HAC	5	1000	.8585
Color/Gradients	Yes	HAC	5	1000	.9108
Color/Gradients	No	HAC	5	1000	.8578
Color/Position/Edges	Yes	HAC	5	1000	.9256
Color/Position/Edges	No	HAC	5	1000	.8585
Color/Position/Edges/Gradients	Yes	HAC	5	1000	.9052
Color/Position/Edges/Gradients	No	HAC	5	1000	.8550
Color	Yes	K-Means	5	1000	.8610
Color	Yes	K-Means	5	10000	.8649
Color	Yes	K-Means	5	100000	.8666
Color	Yes	HAC	5	2000	.8638
Color	Yes	HAC	5	4000	.8702

Table 1: Mean Accuracy with varied Segmentation Parameters

Note: In order to better isolate the individual effects of varying the different segmentation parameters, we only tested the "most similar clusters" approach for the HAC algorithm. This would allow us to more thoroughly test the effect of other parameters on the quality of the segmentation.

3.1 Qualitative Analysis

3.1.1 Effect of Segmentation Parameters

Each of the segmentation parameters have various effects on the quality of the final segmentation:

- *Feature Transform* - The feature transform determines the qualities of points which should be considered as similar to be in a cluster. This could be color, position, gradients, or some combination of all of these. Depending on which transform is chosen, the overall quality of the segmentation can vary drastically.
- *Feature Normalization* - When features in a transform are distributed in a vastly different way, it is likely that one feature will have a higher "weight" when finding the distance between clusters. With normalization, all the features are transformed to have the same mean and standard deviation, and will allow each feature in the transform to contribute equally in segmentation.
- *Number of Clusters* - Generally, the number of clusters is a trade-off between accuracy and generalization. A higher number of clusters will be more accurate compared to the ground truth of a segmentation, especially when clusters can be chosen based on specific segments. But, as the number of clusters decreases, it is easier to have a more generalized view of the final segmentation.
- *Clustering Method* - The clustering method will have varying effects on the final segmentation, depending on the data. A HAC segmentation can be better for when k is not known, as it can simply stop when a good segmentation state is reached. Overall, the accuracy of these two methods are usually pretty similar, but HAC can also be much more accurate in certain situations despite taking a longer time to run.
- *Resize* - A lower resize parameter will reduce the quality of the final segmentation. There will be fewer overall points to cluster, thus resulting in more blocky and less accurate segmentations.

3.1.2 Effect of Parameters on Computation Speed

For the k-means clustering algorithm, increasing the number of clusters drastically increases the runtime of the segmentation. This is because the k-means algorithm needs to do more comparisons on each point, as there are more possible centers that each point can be clustered into.

For the HAC algorithm, increasing the max pixels value slows the algorithm down to a crawl and requires much more RAM to compute the segmentation. Due to the nature of the algorithm, each pixel starts in its own cluster, and this requires a large amount of memory for images with many pixels. The max pixels also affects the k-mean runtime, but not as drastically as with HAC.

In both algorithms, normalization increases the runtime of the clustering, and this effect is more noticeable in feature transforms with many points (such as the `Color / Position / Edges / Gradients` transform). The choice of feature transform also affects runtime, as transforms with more data points cause the clustering to take longer to run. Also, feature transforms which require some external calculation (such as `Edges` or `Gradients`) affect computation speed, as it is computationally expensive to initially calculate these transforms.

3.1.3 Image Properties

There are many properties of an image which can affect segmentation. Any amount of noise in an image will introduce difficulties to determine cluster distances, leading to inaccurate clusters. Also, an image which is busy, with lots of distinct objects or a complicated background, will be harder to segment. Finally, if the foreground and background of an image are visually similar, it may be hard to separate them into segments, as the cluster distance may be very close.

3.2 Quantitative Analysis

Based on our quantitative analysis seen in the table above, we saw that each of the segmentation parameters affected the quality of the final foreground-background segmentation in different ways.

- *Feature Transform* -
  The color transform performs reasonable well all by itself, with all of the mean accuracy's being above 0.83. Adding position tended to make the algorithm slightly more accurate, all other parameters being equal, increasing the accuracy from around 0.86/0.87 with K-means to 0.87/0.88. With HAC, adding position seemed to not have a noticeable affect as the accuracy was 0.86/0.85 with just color and stayed around 0.85 when the number of clusters was at 5.

  When using just color and edges, the accuracy does not notable improve and actually decreases when the values are normalized (0.79 vs the 0.83 with just color and normalized). With HAC, using just color and edges increases accuracy to 0.928 but only when normalized. If using all 3 transforms so far, color, position, and edges, the same is true for K-Means where normalization decreases accuracy but not normalizing the values increases accuracy slightly to 0.88. With HAC, it is strictly the same or better performance, not decreasing accuracy like with K-Means.

  Using just color and gradients yields the same results as with edges where normalization greatly effects if the transform helps or hurts the algorithm. Gradients follows the same pattern as edges did in this respect. When combining all the transform types with K-Means, there is no notable difference between just color/position and using them all together as both versions yield accuracy around 0.87-0.88. With HAC, something very different happens. Color/position yields an accuracy around 0.85, however adding edges increases this to 0.92 (when normalized). If position and edges are used similar results are generated seeming to imply that edges have more of an affect on HAC than position does. If gradients are added, the accuracy decreases to around 0.90 and so adding all the transforms together actually has a detrimental affect on the algorithm as it slows down processing and makes the accuracy worse. Despite making the accuracy worse than other combinations of fewer transforms, using all of them still increases accuracy over just color from 0.86 to 0.90.

- *Feature Normalization* -
  When using solely color, normalization increases the accuracy from 0.868 to 0.874. With all the rest of K-Means, having normalization actively makes the accuracy worse across basically all combinations of transforms. 0.87 vs 0.88 for color/position, 0.79 vs 0.86 for color/edge, 0.79 vs 0.87 for color/gradient, 0.79 vs 0.88 for color/position/edges, and 0.79 vs 0.89 for all of them together.

  When using solely color, normalization of HAC makes the accuracy slightly worse 0.86 to 0.85 without it, but this behavior does not continue when adding other transforms. This exact behavior is inverted when using HAC. The presence of normalization makes the accuracy equal or better in all cases besides just color. 0.85/0.85 for color/position, 0.92/0.85 for color/edges, 0.91/0.86 for color/gradients, 0.92/0.86 for color/posi-

tion/edges, and 0.90/0.85 for all of the transforms together.

- *Number of Clusters* -
- *Clustering Method* -
- *Resize* -

In addition to the just the parameter-based effects, we also see that some images are simply more difficult to segment correctly than others. Inherently, images with a large and clear separation between the foreground and background are the easiest to separate (such as `cat_march` or `Cat_Bed`, as they are both subjects on a white background). Images with more complicated backgrounds are harder to segment (such as `kitten16`), and images in which the foreground and background are visually similar or blurred (such as `stripey-kitty`) also cause problems in the segmentation.

3.3 Other Observations

Another interesting thing we noticed in the behavior of these algorithms was in running the single-link Hierarchical Clustering algorithm. We noticed that, regardless of the feature transform, this algorithm would ultimately end up clustering nearly the entire image into one cluster. At a certain point, the foreground cluster would start "absorbing" the small background clusters, as the single-link distance would be the smallest between these clusters. This would be especially prevalent on some images such as `black_kitten.jpg`, where we would need a $k = 300$ clusters for the foreground and background to be classified correctly.