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

2.1 Successful Segmentations

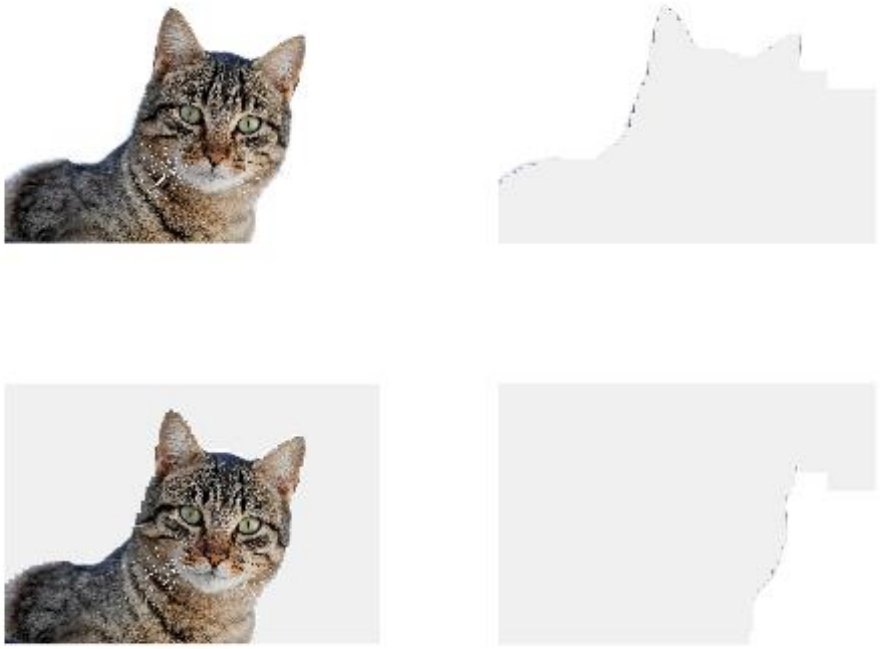

Figure 1: `cat_march.jpg`, using HAC with $k = 3$, position + color features, feature normalization, and a resize factor of 0.025.

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

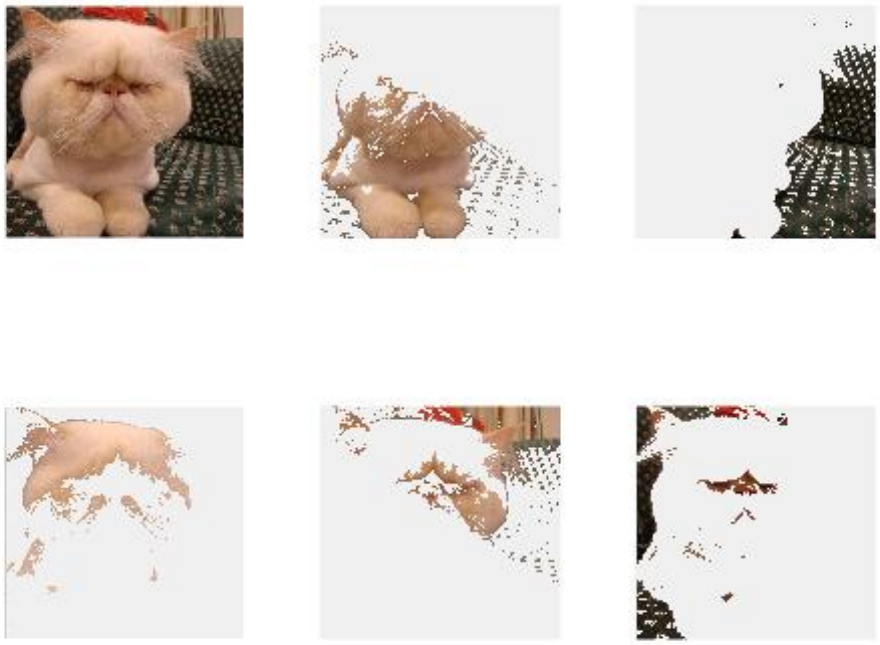

Figure 4: `cat_grumpy.jpg`, using k-means clustering with $k = 5$, position + color features, and no feature normalization.

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

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

3.1 Questions

1. What effect do each of the segmentation parameters (feature transform, feature normalization, number of clusters, clustering method, resize) have on the quality of the

final segmentation?

2. How do each of these parameters affect the speed of computing a segmentation?

   For K-Means: Increasing the number of clusters drastically increases the run-time of the segmentation.

   For HAC: Increasing the Resize/Max Pixels value slows the algorithm down to a crawl and requires much more RAM to compute the segmentation.

3. How do the properties of an image affect the difficulty of computing a good segmentation for that image?

4. Based on your quantitative experiments, how do each of the segmentation parameters affect the quality of the final foreground-background segmentation?

5. Are some images simply more difficult to segment correctly than others? If so, what are the qualities of these images that cause the segmentation algorithms to perform poorly?

6. Also feel free to point out or discuss any other interesting observations that you made.