CS 166 Computational Cameras **June 05, 2024**

Probabilistic Image Segmentation for Color Tasks

Zachary Huang

Note: All of the code and input images can be found at https://github.com/zack466/cs166-final

Introduction

Alterring or enhancing the colors in an image is a common task in image processing. One can modify the global color statistics of an image to accomplish color correction tasks like adjusting brightness, contrast, and white balance. However, in many cases, it is beneficial to work in a perceptual colorspace such as $l\alpha\beta$ because it separates the color information into one luminance channel and two chromaticity channels, and human vision is generally more sensitive to luminance than to color. To recolor an image, we can keep many high-frequency details by keeping the luminance channel constant while modifying the two color channels.

In addition, it is generally desired to modify different parts of an image in different ways. However, manually segmenting an image pixel-by-pixel is extremely tedious, while naive segmentation is not great at resolving fine details and can leave obvious seams in an image. Thus, it is generally useful to consider probabilistic segmentation, in which every pixel is assigned a probability of being in a certain segment. One can segment an image probabilistically, modify individual segments, and then re-composite the image to produce a smooth result.

One of the applications of segmentation is color transfer – mapping the general color characteristics of a source image to a target image. This can be applied to accomplish tasks such as (color-based) style transfer, flash/no-flash denoising, deblurring, image colorization, and more. This is because we can transfer the colors from a reference image (even if it is noisy or blurry) to a target image that is missing certain desired color characteristics (such as if it is a photo taken with flash or a greyscale image). In addition, it is possible that segments can be manually modified to adjust image colors in a user-guided, content-aware manner.

In this report, the model of interest is that of Gaussian Mixture Models (GMMs). Essentially, GMMs model data as a linear combination of (possibly high-dimensional) gaussian distributions, each with a distinct mean and variance. One can think of this model as performing a similar task to k-means clustering, but with smooth, probabilistic boundaries instead of hard transitions between separate components. The idea is that we can learn a mixture of gaussians to model the color distributions of an image's pixels in the $l\alpha\beta$ colorspace. We can take the gaussians learned from one image and map them onto the gaussians of a separate image. This map can then be used to transfer colors probabilistically and smoothly.

One way that Gaussian Mixture Models are learned is through expectation-maximization (EM). Similarly to K-means, the parameters of a GMM are initialized randomly, and the pixel probabilities and model parameters are repeatedly re-estimated until they converge to some local minimum. Some modifications to this algorithm are provided in [1], and they are the main focus of this paper.

Methods

A naive way of estimating the GMM for an image is directly plugging individual pixels into the model with no spatial information. Since each pixel is 3-dimensional, we will denote this as the "3d" algorithm. Note that the specific equations used can be found in the original paper [1]. We also initialize the GMM parameters with the means found using K-means clustering, as this seemed to produce slightly better results than random initialization. Roughly, the code used to compute this looks like:

```
function gmm_3d(pixels, N; ϵ=0.005)
  g = gmm(pixels, N)  # the GMM model
```

```
  # initializes means and standard deviations using k-means clustering
  K = kmeans(pixels, N)
  g.means = K.centers
  g.stds = K.stddevs

  while true
    new_g = estep(g)  # re-estimate pixel probabilities
    new_g = mstep(new_g)  # re-estimate means/variances

    diff = difference(g, new_g)  # how much the GMM state changed

    g = new_g  # update current GMM state

    if diff < ϵ
      break  # if we have converged on a solution
    end
  end

  return g
end
```

However, this algorithm introduces certain issues. Since there is no spatial information, segments can easily contain pixels from all over the original image, even if they are spatially separated by a large distance. In addition, the total number of gaussian components remains constant throughout the entire algorithm, though this may not be optimal. One could consider augmenting each pixel with its spatial coordinates to produce a 5-dimensional GMM model, but [1] claims that this will lead to many local minima, likely due to the "curse of dimensionality".

As a result, [1] introduces two modifications: 1) a spatial smoothing step (essentially a bilateral filter) that incorporates spatial information into the EM algorithm, and 2) a refining step that merges gaussians with similar means. Roughly, the code used looks like:

```
function gmm_3d_spatial(pixels, N; ϵ=0.005)
  g = gmm(pixels, N)  # the GMM model

  # initializes means and standard deviations using k-means clustering
  K = kmeans(pixels, N)
  g.means = K.centers
  g.stds = K.stddevs

  while true
    new_g = estep_spatial(g)  # re-estimate pixel probabilities
    new_g = spatial_smoothing(new_g)  # bilaterally filter each component
    new_g = mstep(new_g)  # re-estimate means/variances
    new_g = merge(new_g)  # merge gaussians with approximately equal means

    diff = difference(g, new_g)  # how much the GMM state changed

    g = new_g  # update current GMM state

    if diff < ϵ
      break  # if we have converged on a solution
    end
  end

  return g
end
```

The end result is a set of $N' \leq N$ gaussians that overall capture the color characteristics of an image.

Then, two basic rules can be used to map between gaussians from different images: 1) mapping each gaussian to the corresponding gaussian with the most similar mean luminance (global), and 2) mapping each gaussian to the corresponding gaussian that spatially overlaps with it the most (spatial). Both of these rules were tested to perform color transfers. The equation used to perform color transfers between GMMs can be found in [1].

Results

The EM algorithms were run on a small set of test images, which include 3 of my own images and 6 images taken from the Kodak Lossless True Color Image Suite.

Effect of Initial Number of Gaussians

First, we evaluted the effect of the number of gaussians components on segmentation (with spatial information), with k-means clustering as a control.

Figure 1: Sunset - ground truth

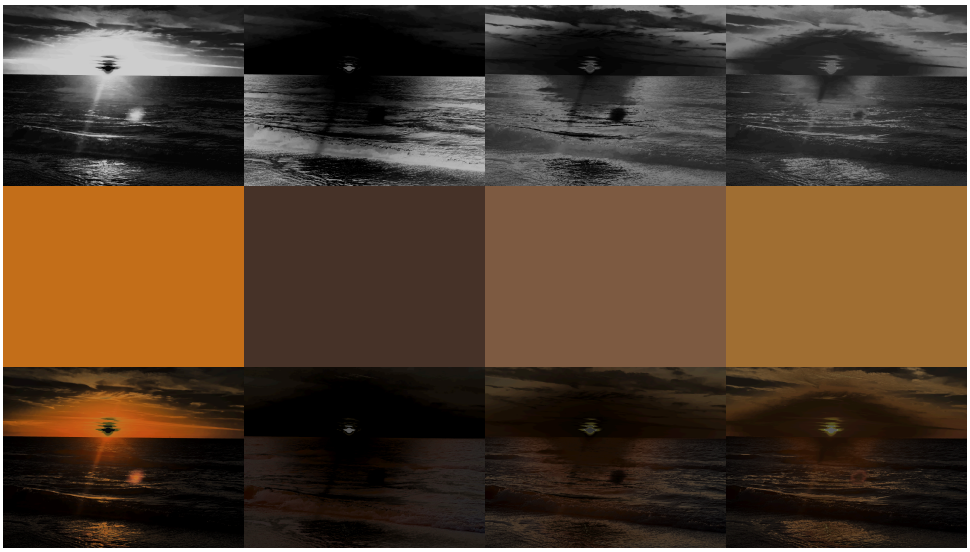


Figure 2: Sunset - 4-component spatial segmentation. Top row: the gaussian components. Middle row: the mean color of each component. Bottom row: the region of the original image each component captures.

Figure 3: Sunset - 4-component reconstruction. This roughly illustrates the image information captured by the model.

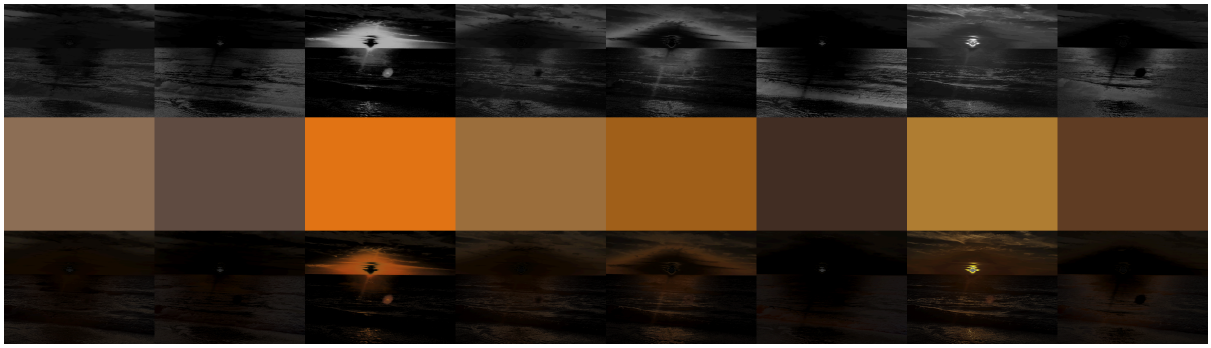


Figure 4: Sunset - 8-component spatial segmentation. Top row: the gaussian components. Middle row: the mean color of each component. Bottom row: the region of the original image each component captures.

Figure 5: Sunset - 8-component reconstruction. This roughly illustrates the image information captured by the model.

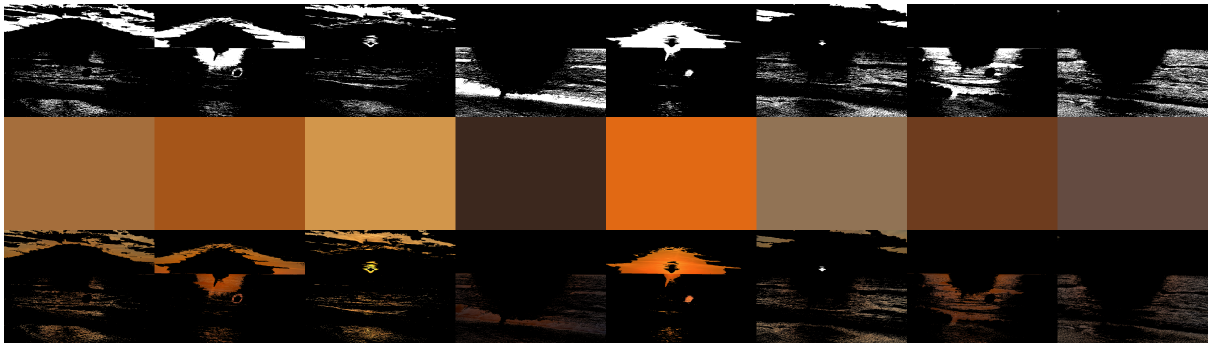


Figure 6: Sunset - 8-component k-means segmentation. Top row: the gaussian components. Middle row: the mean color of each component. Bottom row: the region of the original image each component captures.

Figure 7: Sunset - 8-component k-means reconstruction. This roughly illustrates the image information captured by the model.

We can see that even a few number of gaussian components can capture a large amount of detail due to the probabilistic aspect. Even just 4 components are able to capture the general color palette associated with an image. In comparison, the k-means clusters can capture a similar amount of information but lose many high-level details. We see that the k-means clusters are very jagged, while the GMM components are much smoother.

Effect of Spatial Filtering

Next, we checked if the spatial smoothing actually improved segmentation compared to the naive method.

Figure 8: Kodim03 - 8-component 3d segmentation. Top row: the gaussian components. Middle row: the mean color of each component. Bottom row: the region of the original image each component captures.

Figure 9: Kodim03 - 8-component 3d reconstruction. This roughly illustrates the image information captured by the model.

Figure 10: Kodim03 - 8-component spatial segmentation. Top row: the gaussian components. Middle row: the mean color of each component. Bottom row: the region of the original image each component captures.

Figure 11: Kodim03 - 8-component spatial reconstruction. This roughly illustrates the image information captured by the model.

In general, we noticed that the 3d and spatial GMM algorithms did not differ much when there were a small number of components. However, when the number of components increased, the spatial GMM algorithm definitely helped to separate out uniquely-colored segments, while the 3d algorithm tended to converge to a low-contrast, global representation.

Automatic Color Transfer (Spatial and Global)

We then performed automatic color transfer between test images using the two gaussian mapping rules explained previously. A sample of the results are shown below:

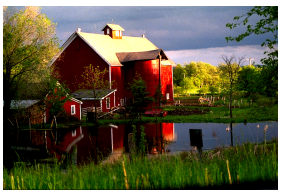
Target	Source	Result (global mapping)	Result (spatial mapping)
			
			
			
			

Table 1: The result of transferring colors from an image of macaws onto various other photos from the set of Kodak images.

The results are quite interesting. We see that the global mapping tends to produce oversaturated, unnatural images since the entire image's image statistics are used during the transfer. Meanwhile, the spatial mapping seems to work quite well – the red color of the right macaw is successfully transferred to the door in the 2nd row and the barn in the 4th row, while the yellow macaw seems to add a yellow tint to the sky in the 1st row and the leftmost cap in the 3rd row. Even though the images used are completely unrelated, we can see that the algorithm is able to transfer color characteristics from one image to another in a reasonable manner.

Manual Color Transfer

Another use case that is enabled by the GMM is manual color transfer. We can separate an image into gaussian components, manually alter the mean colors of certain components and then re-composite the components to produce a re-colored version the original image.

Figure 12: Simple drawing - ground truth

Figure 13: Simple drawing - separated into 3 components

Figure 14: Simple drawing - manually recolored

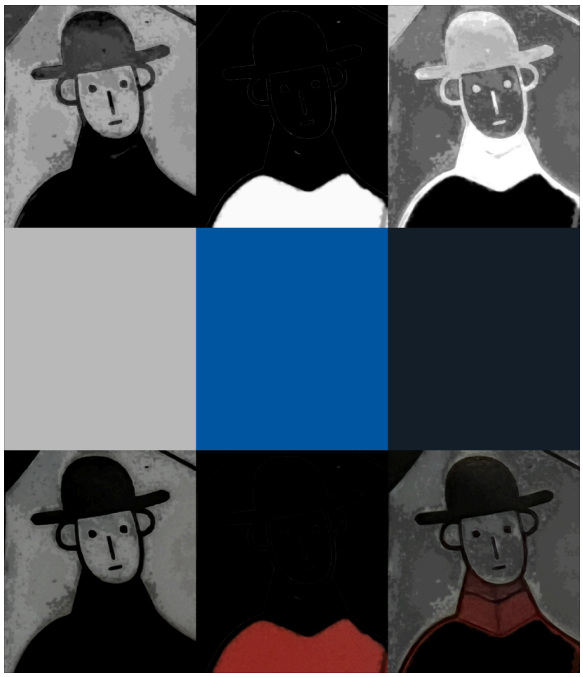


Figure 15: Simple drawing - the 2nd and 3rd components were modified to be more blue

Some more examples are shown:

Figure 16: A recolored sunset. Left: ground truth. Right: after modifying three components (out of six) to be more blue.

Figure 17: My roommate Brady. Left: ground truth. Right: after modifying four components (out of eight) to be more green.

As we can see, the manual color modification works surprisingly well. There is some level of color bleed, but the non-recolored parts of the image tend to stay about the same color, and the recoloring blends into the original image quite well.

Image Colorization

The algorithms were also applied to image colorization, but the results were quite lackluster. All of the test images were greyscaled, and recolored using a random colored test image. However, this simply resulted in coloring the entirety of the greyscale image with a random hue (and possibly modifying its brightness/ contrast). Even manually, it was difficult to add colors to greyscale components to generate an image with color. This is likely because the way that greyscale images were clustered by the GMM is different from how

color images are clustered. As a result, the gaussians did not "align", and so colorization was not able to successfully occur. In the time I worked on this project, I was not able to reproduce the image colorization results from [1].

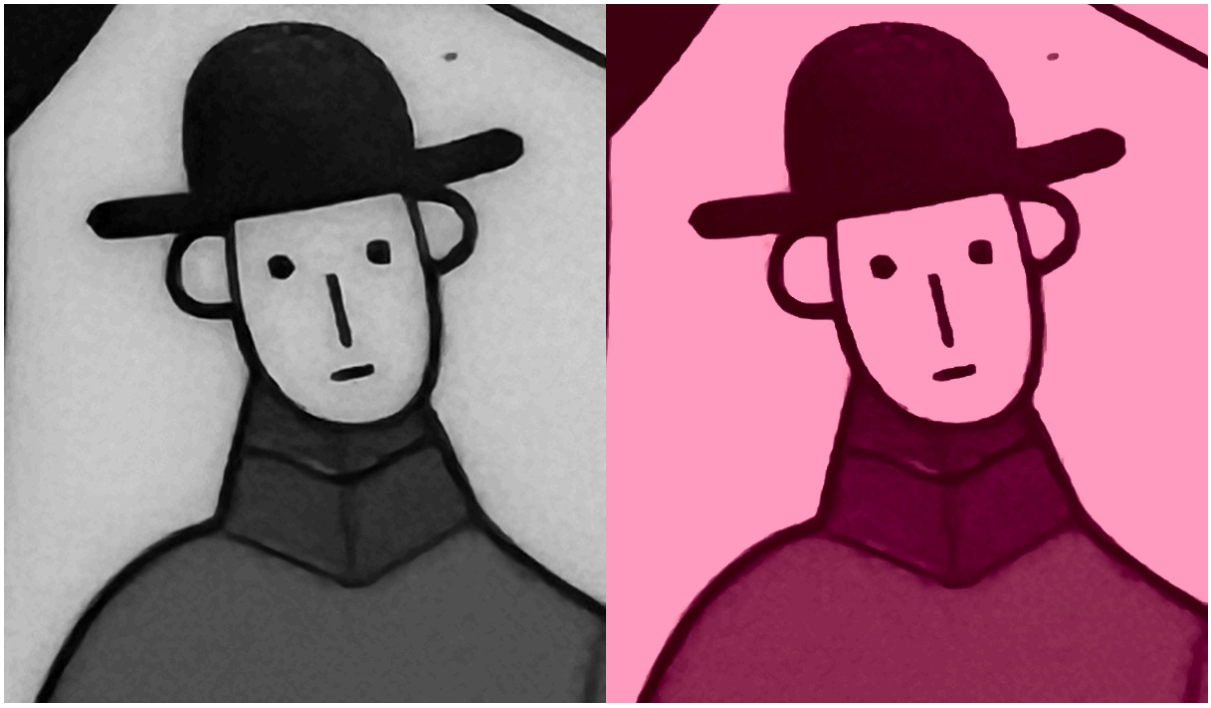


Figure 18: A failed attempt at manually coloring the simple drawing by modifying the means of the greyscale gaussian components.

Table 3: Some failed attempts are automatically recoloring kodim19.png. From left to right: the ground truth, the greyscale input, and two recoloring attempts.

Flash/No-flash pairs

I also tried applying this color transfer algorithm to combining flash/no-flash pairs (traditionally done with bilateral filtering).

Figure 19: The result of transferring colors from the no-flash (noisy) image to the flash image. Left: flash image, middle: no-flash image, right: result of color transfer.

The algorithm worked decently well – the result had much less noise than the no-flash image, but it was able to transfer much of the color characteristics to the no-flash image.

Performance

Generally, each GMM would converge within a dozen iterations if a small number of components is used $N \approx 5$, while it would converge anywhere from 20 to 80 iterations for $N \approx 15$. The spatial EM algorithm could take as long as 5 minutes to converge with $N \approx 20$, while it might take less than 10 seconds with $N \approx 5$. Most of the experiments were run on my M1 Macbook, while some bulk computations were run on the Undergraduate Computer Science Club's (UGCS) server.

Conclusion

While the algorithm worked quite well in some cases, it still has some issues. The GMM can still struggle with spatial connectivity and may get stuck in a local minima in which some segments are spread out through an entire image. While spatial information is incorporated in the algorithm, it does not constrain the components in a strong manner, and some components end up carrying a lot of high-frequency detail that is spread across the entire image. In addition, there are many parameters that can be used, ranging from the number of iterations to use, the epsilon value used to determine when the algorithm has converged, the spatial and color smoothing factors used by the bilateral filter, the initialization procedure, etc. Unfortunately, I did not have enough time to tune all of these parameters, and many of them were arbitrarily chosen during testing.

While the color transfer worked quite well, the image colorization did not. I think that the GMM algorithm fundamentally clusters pixels differently with and without color, and so there is no reason to expect colors to be transferred correctly onto the greyscale image. It is also possible that my implementation had errors which prevented the colorization from working correctly. I would imagine that the algorithm could be modified to weigh luminance higher, which could help segment images similarly regardless of color. There are many avenues in terms of this algorithm which could be researched in the future.

Bibliography

[1] Yu-Wing Tai, Jiaya Jia, and Chi-Keung Tang, "Local Color Transfer via Probabilistic Segmentation by Expectation-Maximization," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, San Diego, CA, USA: IEEE, 2005, pp. 747–754. doi: 10.1109/CVPR.2005.215.