Image Segmentation with Gaussian Mixture Models: A Hands-On Tutorial

Alex Hagiopol

***Abstract*—The expensive requirements of high-precision training data and large-scale computing resources for implementing modern image segmentation approaches motivate a look back at classical segmentation approaches. This article explains Gaussian Mixture Models (GMMs), shows how to compute GMMs using the Expectation Maximization (EM) algorithm, and shows how to apply these two concepts to image segmentation in a fully unsupervised and computationally tractable way. Implementation techniques for achieving stability despite limited bit precision are described, and image segmentation results obtained using these techniques are provided. The accompanying open source software[1] is a reference implementation with visualization tools.**

I. Introduction

Image segmentation is clustering or classification of image pixel data. Segmentation may be used to infer semantic meanings for image pixels or to compress the amount of data required to convey the meaning of an image as shown in Figure 1. State-of-the-art image segmentation techniques such as [1], [3], and [2] are useful in volumetric performance capture applications [4] [5], general 3D scene reconstruction applications [7] [6], and autonomous driving applications [8]. Such segmentation techniques rely on large training data sets such as [12] and [8] and the computational resources to train neural networks with millions of parameters on many thousands of examples. Furthermore, state-of-the-art segmentation approaches also require that the data be accurately labeled pixelwise to obtain accurate segmentation results as observed explicitly by [13] and observable in the figures of [1] and [3]. The need for pixelwise accurate segmentation results is especially strong in volumetric reconstruction applications such as [7] and [6] where incorrect segmentation results defeat the purpose of producing photorealistic 3D models of scene elements such as performers.

While neural networks have achieved state-of-the-art results in image segmentation, the costs of precise data labeling and network training computation motivate this study of alternative segmentation techniques. This article explores Gaussian Mixture Models (GMM) and Expectation Maximization (EM), the core components of classical image segmentation literature such as [9], [10], and [11]. The mathematics of the techniques are laid out, numerically stable software implementations are presented, and the algorithm states are illustrated with image segmentation results[2].

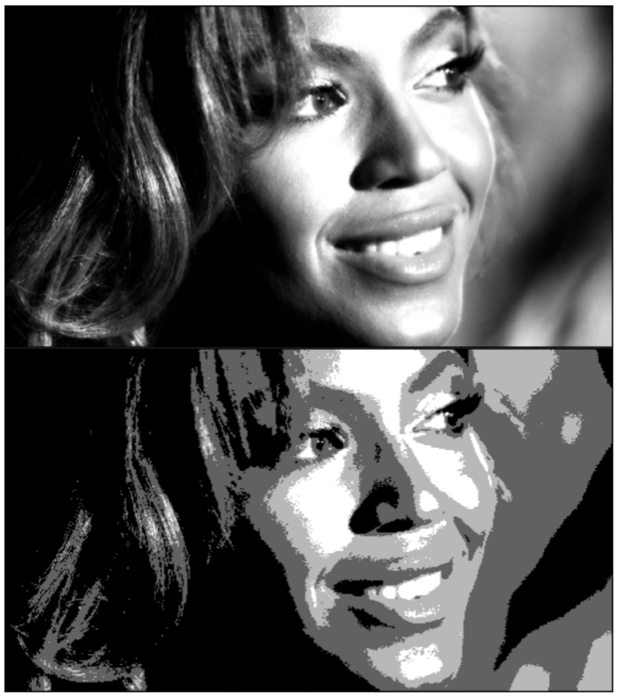

Figure 1. Example of an image segmentation. **Top:** Original grayscale image with 8-bit pixel depth. **Bottom:** The top image segmented into 4 groups and visualized with each original pixel value replaced by the mean value of its assigned group. This segmentation yields a compressed grayscale image with 2-bit pixel depth - a 4X reduction in data size at the cost of image quality.

II. Theory

The segmentation problem is cast as a problem of estimating probabilistic models representing K image segments where K is an application-dependent constant selected *a priori*[3]. K is the number of the components in the mixture of models representation. Each of K groups of pixels is represented using a Gaussian model that represents the probability of each pixel value belonging to that group. Once the state variables of each probabilistic model are estimated, predicting to which model each pixel belongs is equivalent to selecting the model with the highest likelihood for that pixel value. This mixture of Gaussian models used for predicting segment assignment is the source of the name "Gaussian Mixture Models" and is defined as

[1]Software available at github.com/alexhagiopol/gmm.

[2]For practical segmentation pipelines useful in applications such as [5], [4], [7], and [6] using GM and EMM, see [9], [10], and [11].

[3]K is determined from application parameters e.g. an image must be segmented into background and foreground components yielding K=2 or an image must be compressed to 2-bit pixel depth yielding K=4

$$P(x) = \sum_{i=1}^{k} P(C=i)P(x|C=i), \qquad (1)$$

where the Gaussian mixture distribution *P* over data points *x* has *k* Gaussian distribution components represented by *C*.

The goal of the GMM framework is to estimate the highest probability state variables - the scalar mean μ_k and scalar standard deviation σ_k in the 1D case - of each of *K* Gaussian models in the mixture as shown in Figure 2. Although images are generally represented as 2D matrices, GMM is introduced by representing an image as a container of many 1D pixel values. Thus a 1 Megapixel image is a container of 1 million 8-bit scalar values in the range[4] [0, 255]. Since each pixel is represented in 1D, estimating the parameters of GMMs is thought of as estimating model probabilities over a 1D state space.

We follow the theory set out by [14] and [15] and represent the state variables of each Gaussian model for each pixel group as unobservable hidden variables. If the problem is cast in terms of only observable variables, the number of parameters that must be estimated drastically increases. To clarify the advantage of hidden variables, [14] provides the example of diagnosing heart disease. Heart disease itself is unobservable: only risk factors (e.g. smoking, diet, and exercise) and symptoms (e.g. chest pain, shortness of breath, and nausea) are observable. The unobservable presence of the disease can therefore be inferred by reasoning about the observable variables. If one were to attempt instead to estimate the presence of the symptoms using only information about the risk factors, the number of parameters to be estimated would be much higher than the number of parameters in the system with a hidden variable. The issue with using hidden variables in representing the structure of the segmentation problem is that it is not obvious how to estimate hidden variables because their values cannot be directly observed.

To estimate the most likely values of hidden variables, the EM algorithm is applied as summarized in [14] and in Chapter 9 of [15]. It is restated here in terms applicable to image segmentation.

1) **Initialization Step** Assume an initialization for parameters mean μ_k and standard deviation σ_k of each of *K* GMM component model. Initialize a mean μ_k, a variance σ_k^2, and a weight w_k for each component *k*. A textbook implementation calls for choosing from among arbitrary initialization, initialization using randomly selected pixel intensities present from the input dataset, or initialization using another clustering algorithm such as K-Means [15]. The choice of initialization strategy is very application-specific and heavily impacts convergence behavior. In the single-image segmentation mode of the accompanying software, means μ_k are initialized to evenly distributed values between 0 and 1. The

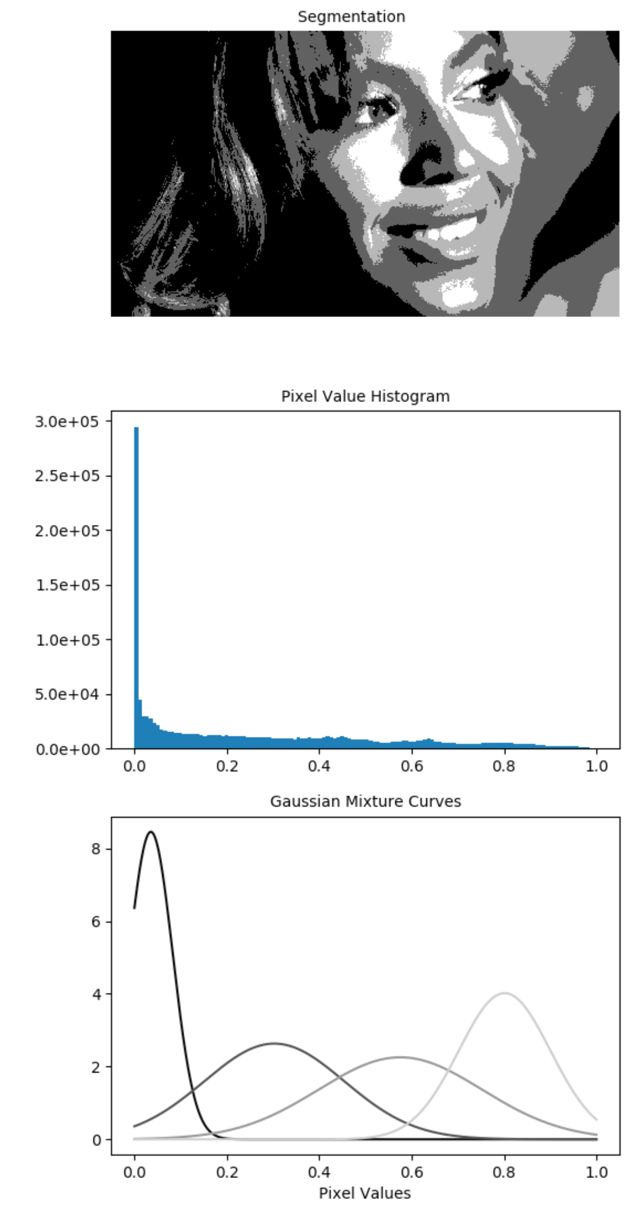

Figure 2. Example of intermediate results of Gaussian mixture models estimation after 4 iterations of the Expectation Maximization algorithm. Given the same original image from Figure 1, 4 Gaussian models (**bottom**) are fit to the frequency distribution of pixel gray values (**middle**). The segmentation image (**top**) is created by assigning to each pixel the value of the mean of the Gaussian model whose probability is highest at that pixel's value.

variances σ^2 are initialized to arbitrary small fractions such as $\frac{10}{255}$following from the maximum pixel intensity representable by an 8 bit integer.

2) **Expectation Step** For each pixel value x_n, compute the responsibility γ_{nk} i.e. the probability that the n^{th} pixel belongs to the k^{th} component divided by the sum of the probabilities that the n^{th} pixel belongs to all other components. In the following equation, *n* is an individual pixel index out of *N* total pixels, *k* is an individual component index out of *K* total components, and $\mathcal{N}$ is

[4]Although image data is often stored with 8 bit pixel depth, for numerical stability reasons, our implementation normalizes pixel values to vary between floating point values 0.0 and 1.0. See Implementation section.

the Gaussian probability density function [16].

$$\gamma_{nk} = \frac{w_k * \mathcal{N}(x_n, \sigma_k, \mu_k)}{\sum_{j=0}^{K} \Big(w_j * \mathcal{N}(x_n, \sigma_j, \mu_j) \Big)} \tag{2}$$

In the case of 1D image processing, this equation is executed once for each pixel in an image independently: unlike the graph cut based global optimization technique described in [9], the GMM technique does not consider relationships among neighboring pixels through any mechanism except the variables w_k, μ_k, and σ_k which are common among all pixels belonging to each k of K total Gaussian component models.

While the previous equation is useful for understanding the EM algorithm, a direct computer program implementation of the Expectation step as described thus far suffers from numerical instability. Due to the limits of representing an infinite real number space with finite bit precision, a legal mathematical operation (dividing by a very small number) is represented as an illegal mathematical operation (dividing by zero) in a direct computer program implementation. When the Expectation step estimates probabilities of pixel membership in a model component that are vanishingly small (e.g. probabilities for pixels that very certainly do not belong to a given model component), the probabilities are often not representable even by a 64-bit double precision floating point number. In this case, the underflow phenomenon causes these small probabilities to be approximated as zero in machine memory which leads to division by zero.

To remedy the numerical stability problem, the fundamental properties of the $\ln()$ function and a derivation of these properties called the LogSumExp [17] technique are applied. During the Expectation step, small values are converted into $\ln()$ space, manipulated mathematically in that space, then converted back into real number space before the Maximization step is performed. The reasoning for using $\ln()$ space is that in $\ln()$ space, positive values extremely close to zero are mapped from real number space to values far enough away from zero that they are readily representable by double precision floating point numbers in machine memory. Thus an implementation of the Expectation step is created such that the step is mathematically equivalent to what is described in Equation 2, but does not require the use of extremely small values in machine memory.

Since the Expectation step contains multiplication, division, and series summation operations, $\ln()$ space equivalents for these operations ($\ln(A * B)$, $\ln(\frac{A}{B})$, and $\ln(\sum_{i=0}^{N} x_i)$) must be developed to adapt the Expectation step to $\ln()$ space. First, multiplication and division operations must be expressed in $ln()$ space using the fundamental rules of logarithms:

$$\ln(A * B) = \ln(A) + \ln(B) \tag{3}$$

$$\ln(\frac{A}{B}) = \ln(A) - \ln(B) \tag{4}$$

Next, the summation of an arithmetic series must also be expressed in $ln()$ space by deriving an expression from the definition of LogSumExp presented in [17]. In these equations, x_i is an element of an arithmetic series and x_{max} is the maximum element of the arithmetic series containing x_i.

$$LSE(x_0 : x_n) = \ln \Big(\sum_{i=0}^{N} e^{x_i} \Big) \tag{5}$$

$$LSE(x_0 : x_n) = x_{max} + \ln \Big(\sum_{i=0}^{N} e^{x_i - x_{max}} \Big) \tag{6}$$

Modify both sides of Equation 6 to compute the LSE of $ln(x_0) : ln(x_n)$, and apply the rule of logarithms stating that $x = e^{\ln(x)}$:

$$\ln(\sum_{i=0}^{N} x_i) = \ln(x_{max}) + \ln \Big(\sum_{i=0}^{N} e^{\ln(x_i) - \ln(x_{max})} \Big) \tag{7}$$

Thus, the numerically stable expression for the responsibilities calculated in $ln()$ space during the Expectation step, $\ln(\gamma_{nk})$, is derived:

$$\gamma_{nk} = \frac{w_k * \mathcal{N}(x_n, \sigma_k, \mu_k)}{\sum_{j=0}^{K} \Big(w_j * \mathcal{N}(x_n, \sigma_j, \mu_j) \Big)} \tag{8}$$

$$P_{nk} = \ln(w_k) + \ln \Big(\mathcal{N}(x_n, \sigma_k, \mu_k) \Big) \tag{9}$$

$$P_{n\_max} = \text{argmax}_k (P_{n0} ... P_{nk} ... P_{nK}) \tag{10}$$

$$\begin{aligned} \ln(\gamma_{nk}) &= \ln(w_k) + \ln \Big(\mathcal{N}(x_n, \sigma_k, \mu_k) \Big) \\ &- \Bigg(P_{n\_max} + \ln \Big(\sum_{k=0}^{K} e^{P_{nk} - P_{n\_max}} \Big) \Bigg) \end{aligned} \tag{11}$$

Once $\ln(\gamma_{nk})$ is computed, it is exponentiated to yield γ_{nk} which is then used to continue the EM steps that follow.

3) **Optional: Inference Step** At this point in the algorithm, an inference may be performed in which the component k with the maximum probability for a given pixel is selected as the model that gives rise to the value of that pixel. One way to visualize this inference is to assign to each pixel x_n in the visualization the mean value μ_k of the model k with the highest responsibility γ_{nk} for that pixel as illustrated in Figure 1.
4) **Maximization Step** For each component model k, refit and update the model parameters w_k, μ_k, and σ_k to the input dataset with each pixel value x_n weighted by the previously calculated responsibility γ_{nk}. The first step in estimating the model parameters is to estimate

the number N_k of pixels most likely to belong to each model k. [5]

$$N_k = \sum_{n=1}^{N} \gamma_{nk} \tag{12}$$

With N_k computed, the rest of the model parameters follow:

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} (\gamma_{nk} * x_n) \tag{13}$$

$$\sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \left(\gamma_{nk} * (x_n - \mu_k^{new})^2 \right) \tag{14}$$

$$w_k^{new} = \frac{N_k}{N} \tag{15}$$

5) **Optional: Log Likelihood Step** At this point in the algorithm, the log likelihood of the state of the algorithm can be calculated as follows.

$$LogLikelihood = \sum_{n=1}^{N} \left(\ln \left(\sum_{k=1}^{K} \left(w_k * \mathcal{N}(x_n, \sigma_k, \mu_k) \right) \right) \right) \tag{16}$$

This calculation is used to indicate convergence of the EM algorithm and as a debugging feature that indicates an implementation error in the event that the log likelihood decreases: it is provable [14] that the Expectation Maximization algorithm can only increase its log likelihood on every iteration.

6) **Loop Step** The new model parameters computed in the maximization step can then be used in a new expectation step and inference step. Thus, an iterative algorithm emerges: repeat steps **2** through **5** until a terminating condition is reached. The terminating condition of this algorithm can be convergence of the log likelihood (i.e. an increase of the log likelihood that is below a threshold) or a fixed maximum number of Expectation Maximization iterations.

III. Implementation

Python program implementations of the mathematical definitions of each step of the EM algorithm follow.

1) **Initialization Step** The Initialization step is dependent on input such as tuning parameters and images from disk.

[5]The number of pixels belonging to each model is an integral number, and this equation can be implemented by casting the sum of floating point γ_{nk} values into to an integer sum. However this choice may yield N_k values of zero which would lead to zero division later in the procedure. To increase the numerical stability of the implementation, γ_{nk}, N_k, and all other variables should be represented as floating point numbers preferably with double precision.

```
# 1. Initialization Step:
image = # N*N matrix from user input
components = # num. components from user input
iterations = # num. iterations from user input
means = np.linspace(0, 1, components)
variances = np.float64(np.ones(components)) *
  np.float64(10 / 255)
stdevs = np.sqrt(variances)
weights = np.ones(components)
total_log_likelihoods = np.zeros(iterations)
rows, cols, chans = image.shape
```

2) **Expectation Step** The following is a direct implementation of the Expectation step presented only for clarification: while it is instructive, it should not be relied upon to produce accurate results due to numerical instability issues discussed in the *Theory* section description of the Expectation step.

```
# 2a. Expectation Step:
gammas = np.zeros((rows, cols, components))
denominator = np.zeros((rows, cols))
for k in range(components):
    denominator = denominator + weights[k] *
     sp.stats.norm.pdf(image, means[k],
     stdevs[k])
    gammas[:, :, k] = np.divide(weights[k] *
     sp.stats.norm.pdf(image, means[k],
     stdevs[k]), denominator)
for k in range(components):
    gammas[:, :, k] = np.divide(weights[k] *
     sp.stats.norm.pdf(image, means[k],
     stdevs[k]), denominator)
```

The following is an implementation of the Expectation step that lends itself to numerical stability.

```
# 2b. Numerically Stable Expectation Step:
# compute P matrix containing P_n_k values for
 every n and every k
P = np.zeros((N, M, K))
for k in range(K):
    P[:, :, k] = np.log(weights_list[k]) +
     np.log(sp.stats.norm.pdf(intensities,
     means_list[k], stdevs_list[k]))

# compute P_max matrix containing P_n_max
 values for every n
P_max = np.max(P, axis=2)

# implement expsum calculation used to
 calculate ln(gamma_n_k)
expsum = np.zeros((N, M))
for k in range(K):
    expsum += np.exp(P[:, :, k] - P_max)

# implement responsibilities (gamma_n_k)
 calculation for every n and every k
ln_responsibilities = np.zeros((N, M, K))
ln_expsum = np.log(expsum)
for k in range(K):
    ln_responsibilities[:, :, k] = P[:, :, k] -
      (P_max + ln_expsum)
responsibilities = np.exp(ln_responsibilities)
```

3) **Optional: Inference Step** The inference step can be called (1) at every iteration for debugging, (2) only after the final iteration for producing a results summary, or (3) not at all in the case of production code that simply

passes on converged model parameters as part of a larger image processing pipeline.

```
# 3. Inference Step:
rows, cols, components = gammas.shape
segmentation = np.zeros((rows, cols))
segmentation_idxs = gammas.argmax(axis=2)
for r in range(rows):
    for c in range(cols):
        segmentation[r, c] =
         means[segmentation_idxs[r, c]]
```

4) **Maximization Step** The maximization step, if preceeded by a numerically stable Expectation step, does not need modification to ensure numerical stability. Its main role is to update algorithm state variables.

```
# 4. Maximization Step:
N_k = np.sum(np.sum(gammas, axis=0), axis=0)
for k in range(components):
    means[k] = np.divide(
     np.sum(np.sum(np.multiply(
     gammas[:, :, k], image), axis=0),
     axis=0), N_k[k])
    diffs_from_mean = np.subtract(image,
     means[k])
    variances[k] = np.divide(
     np.sum(np.sum(np.multiply(
     gammas[:, :, k],
     np.square(diffs_from_mean)), axis=0),
     axis=0), N_k[k])
    stdevs[k] = np.sqrt(variances[k])
    weights[k] = np.divide(numPoints[k], (rows
     * cols))
```

5) **Optional: Log Likelihood Step** The log likelihood calculation is the same as the sum of the log of the denominator of the equation implemented in the Expectation step. The log likelihood step may be performed on every iteration to measure convergence progress or it may not be performed at all in the case of a fixed-iteration EM implementation. This step re-uses the procedures from the numerically stable implementation of the Expectation step, specifically the computation of P_{nk}, P_{n_max}, and the expsum $\sum_{k=0}^{K} e^{P_{nk}-P_{n\_max}}$.

```
# 5. Log Likelihood Step:
# ... compute numerically stable expsum, P,
 P_max as in step 2b.
expsum, P, P_max =
 compute_expsum_stable(intensities,
 weights_list, means_list, stdevs_list)
ln_inner_sum = P_max + np.log(expsum)  # inner
 sum of log likelihood equation
log_likelihood = np.sum(np.sum(ln_inner_sum,
 axis=0), axis=0)  # outer sum of log
 likelihood equation
```

6) **Loop Step** The loop step is best implemented as a *for* loop wrapped around the Expectation, Inference, Maximization, and Log Likelihood steps as shown:

```
# 6. Loop Step:
    # 1. Initialization Step ...
    for i in range(iterations):
        # 2. Expectation Step ...
        # 3. Inference Step ...
        # 4. Maximization Step ...
        # 5. Log Likelihood Step ...
```

IV. Results

Figure 3 illustrates the internal state of the Expectation Maximization algorithm for the first six iterations of the algorithm's execution on the input image from Figure 1. Along the top row of Figure 3 are visualizations of the optional inference step (see Theory section) during each iteration. In the inference visualizations, the original input image seen in Figure 1 is segmented into 4 segments hence the presence of 4 different gray levels in the visualizations. The mixture of 4 different gray levels represents the mixture of 4 Gaussian models that is computed by Expectation Maximization. The inference visualizations change on each iteration of the algorithm as the internal state of the Expectation Maximization procedure (represented by the state variables σ_k, μ_k, w_k for each of K models) changes. Along the middle row are histogram plots that show the number of occurrences for each possible pixel value in the input image. These plots do not change with the state of the algorithm because the input image does not change; they are reproduced multiple times only for comparison against the bottom row of Figure 3. The bottom row of Figure 3 shows plots of the Gaussian curves yielded by the algorithm state variables for each of the 4 component models of the mixture. The evolution of the state of the algorithm is most clearly visible in the changes in the Gaussian curves at the bottom of Figure 3.

To summarize the evolution of the algorithm after a significant number of iterations, Figure 4 shows the initial and final mixture models over 15 iterations. Of interest is the leftmost and darkest Gaussian curve that has evolved to closely represent the overwhelming number of black (i.e. 0-valued) pixels in the histograms. After just 15 iterations, this Gaussian curve represents a maximum responsibility 15 times greater than the maximum responsibility represented by its initialization! This illustrates the EM algorithm's ability to produce a mixture model that closely represents its input dataset.

To illustrate EM convergence behavior for an image segmentation application, the bottom of Figure 5 shows segmentation results for each iteration of the EM algorithm when executed using the normalized difference [18] of the two images in the top of Figure 5 as input. The images in the top of Figure 5 are representative of a common segmentation scenario: after capturing an empty scene ("background") followed by an action sequence ("background + foreground"), the goal is to separate pixels belonging to the subject alone ("foreground") in a video sequence. Image pair subtraction alone generally does not produce accurate results because of (1) the necessity of manually specifying a subtraction threshold for input image each pair and (2) non-foreground lighting and shadow effects introduced by inserting an opaque foreground object into the scene. The first segmentation result in the bottom of Figure 5 shows a poor background-foreground segmentation made with an arbitrarily chosen subtraction threshold that is refined — in a completely unsupervised way with no training! — by the EM algorithm into the final segmentation result. The final result - although by no means perfect and requiring further

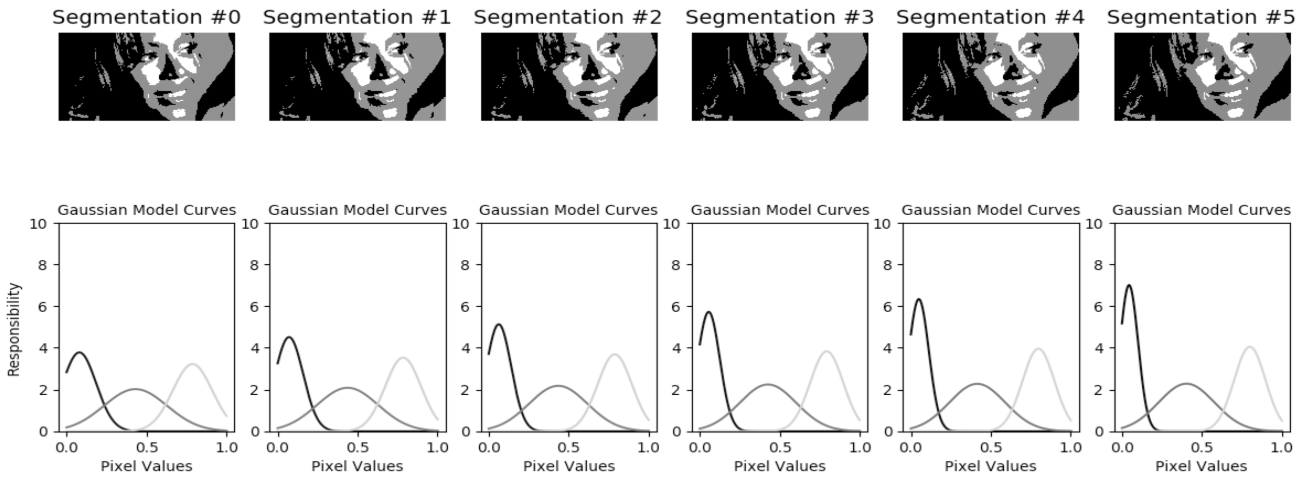

Figure 3. Full algorithm state visualization of first 6 iterations of the numerically stable expectation maximization procedure.

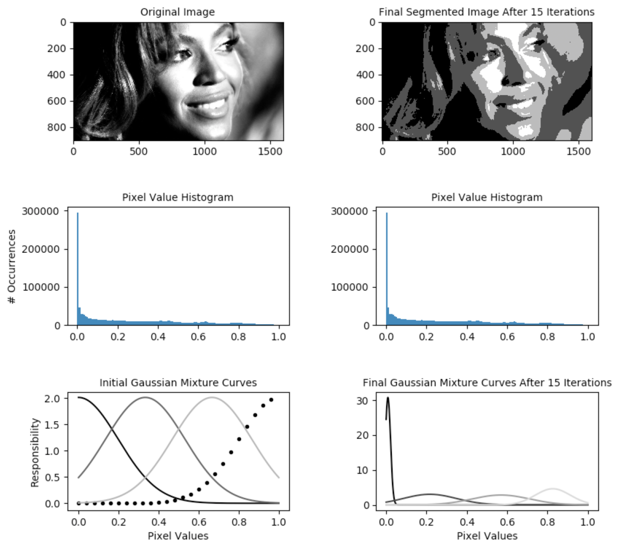

Figure 4. Summary of EM algorithm evolution over 15 iterations. Responsibility curves are colored with the mean of the Gaussian model they represent. When the mean color is too light to visualize, a dotted line is used.

processing for practical uses as described by [9] — is (1) clearly a more accurate representation of the segmentation between the foreground and background than the initialization and (2) arrived at after only 6 iterations of the EM algorithm. Figure 6 illustrates the results of the EM algorithm applied to segmentation of more image data.

The plot in Figure 7 shows the log likelihood values for each iteration of the EM algorithm executed on the dataset in Figure 5. From the plot it is inferred that convergence starts to happen approximately after iteration 5. After this point, the increase in the closeness of the Gaussian model fit may not be worth the additional computation time. Choosing the definition for convergence is an application-specific and dataset-specific design decision: canonical approaches include detect sub-threshold changes in log likelihood or specifying a set number of iterations. As a final example of convergence behavior, the EM algorithm is executed on the input data in Figure 1 to produce the final segmentation result shown in 4. The drastic changes in the Gaussian responsibility curves in the bottom third of 4 are apparent. Figure 3 shows the evolution of the GMM state over a selection of iterations. Visualizations such as those in Figures 3, 4, and 7 will aid in designing a good convergence criterion for the specific application and dataset at hand.

V. Acknowledgements

I thank Neal Grantham for his insightful guidance and patience in reviewing this work.

References

[1] L.C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. Yuille, *DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs,* arXiv:1606.00915v2, 2017.

[2] H. Zhao, J. Shi, X. Qi, X. Wang, J. Jia, *Pyramid Scene Parsing Network,* The IEEE Conference on Computer Vision and Pattern Recognition, 2017.

[3] K. He, G. Gkioxari, P. Dollar, R. Girshick, *Mask R-CNN,* arXiv:1703.06870v3, 2017.

[4] A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, Steve Sullivan, *High-Quality Streamable Free-Viewpoint Video,* ACM Transactions on Graphics, 2015.

[5] M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. Fanello, A. Kowdle, S. Escolano, C. Rhemann, D. Kim, J. Taylor, P. Kohli, V. Tankovich, and S. Izadi, *Fusion4D: Real-time Performance Capture of Challenging Scenes,* ACM Transactions on Graphics, 2016.

[6] W. Matusik, C. Buehler, R. Raskar, S. J. Gortler, L. McMillan, *Image-Based Visual Hulls,* 2000.

[7] P. Song, X. Wu, M. Y. Wang, *Volumetric Stereo and Silhouette Fusion for Image-Based Modeling,* 2010.

[8] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, *The Cityscapes Dataset for Semantic Urban Scene Understanding,* The IEEE Conference on Computer Vision and Pattern Recognition, 2016.

[9] C. Rother, V. Kolmogorov, A. Blake, *GrabCut - Interactive Foreground Extraction using Iterated Graph Cuts*, ACM Transactions on Graphics, 2004.

[10] M. Tang, L. Gorelick, O. Veksler, and Y. Boykov, *GrabCut in One Cut*, The IEEE International Conference on Computer Vision (ICCV), 2013.

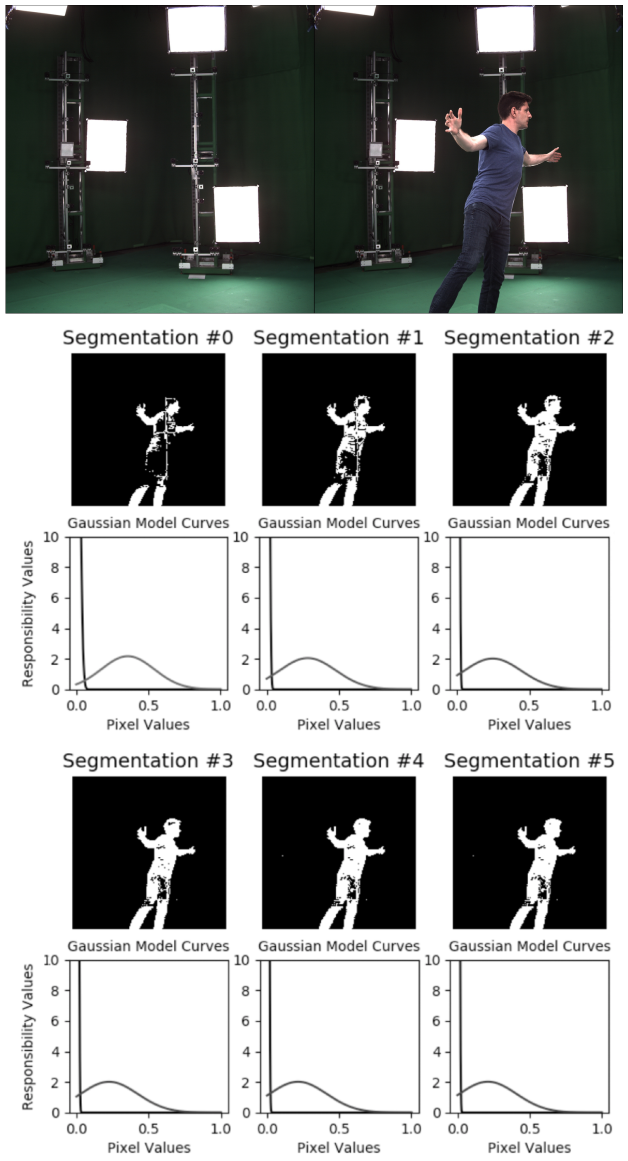

Figure 5. **Top:** Image pair with background (left) and background plus foreground (right). **Bottom:** Background-foreground segmentation results over 6 EM iterations.

[11] M. M. Cheng, V. Prisacariu, S. Zheng, P. Torr, and C. Rother, *DenseCut: Densely Connected CRFs for Realtime GrabCut*, Computer Graphics Forum - Eurographics, 2015.
[12] T.Y. Lin, M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick, P. Dollar, *Microsoft COCO: Common Objects in Context,* arXiv:1405.0312v3, 2015.
[13] Y. Aksoy, T.H. Oh, S. Paris, M. Pollefeys, W. Matusik, *Semantic Soft Segmentation,* ACM Transactions on Graphics, 2018.
[14] S. Russel and P. Norvig, *Artificial Intelligence: A Modern Approach*, Pearson Education, 2010.
[15] C. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2011.
[16] Wikipedia contributors, *Probability Density Function,* "https://en.wikipedia.org/wiki/Probability_density_function", accessed 12-Oct-2018.
[17] Wikipedia contributors, *LogSumExp,* "https://en.wikipedia.org/wiki/LogSumExp", accessed 12-Oct-2018.
[18] Wikipedia contributors, *ColorDifference,* "https://en.wikipedia.org/wiki/Color_difference", accessed 12-Oct-2018.

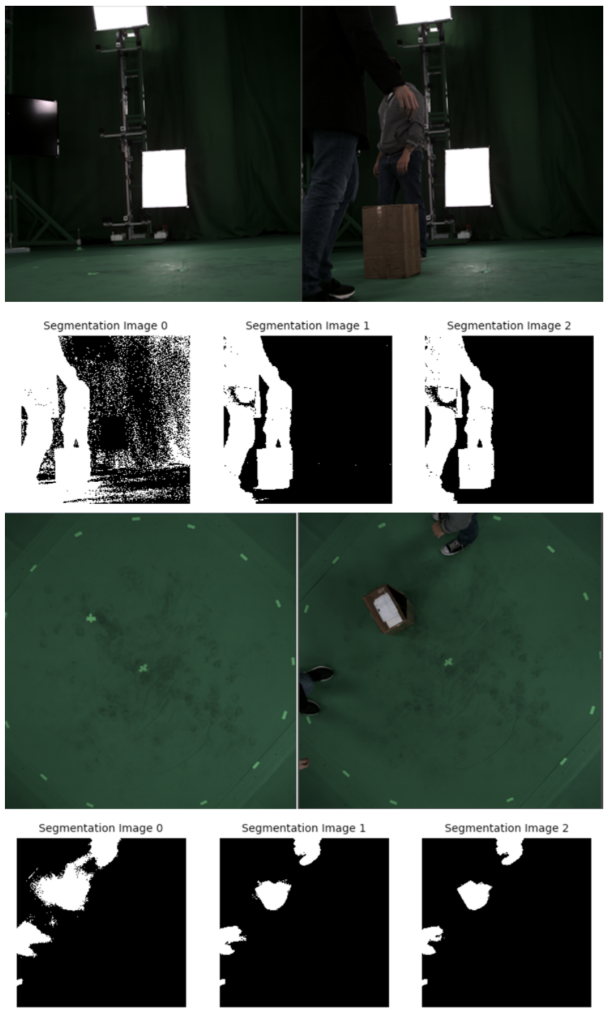

Figure 6. **Color images:** Background-foreground image pairs. **Binary images:** Corresponding segmentation results over 3 EM iterations.

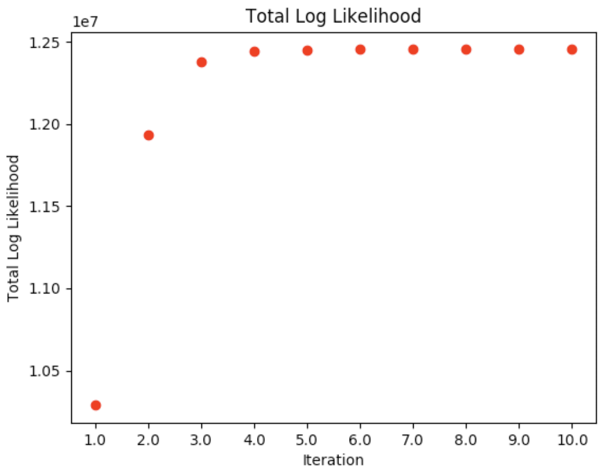

Figure 7. Log likelihood plot of algorithm state during 10 iterations of the EM algorithm executed on the dataset in Figure 5.