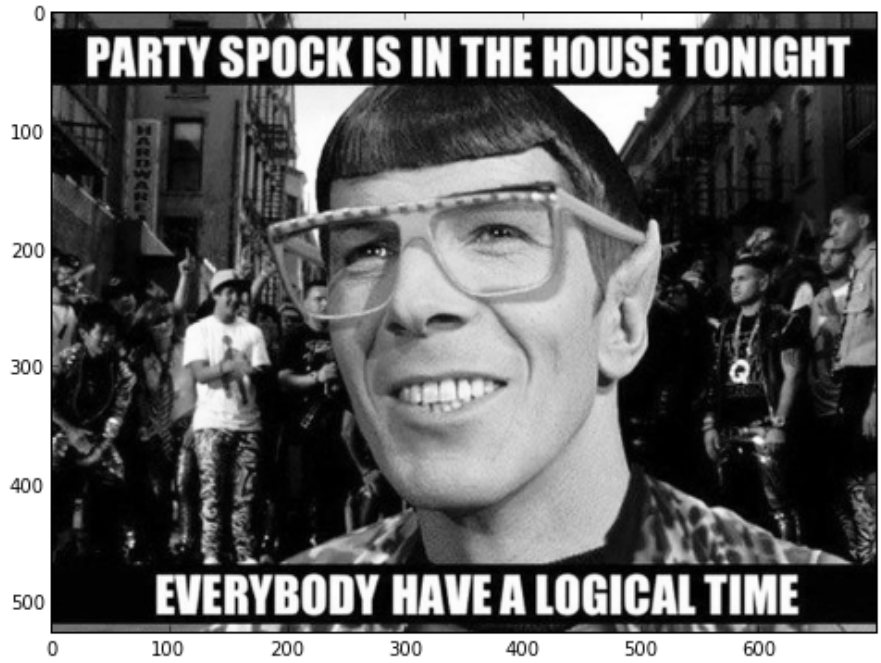
# Assignment 2: Gaussian Mixture Models

## Author: Ruffin White

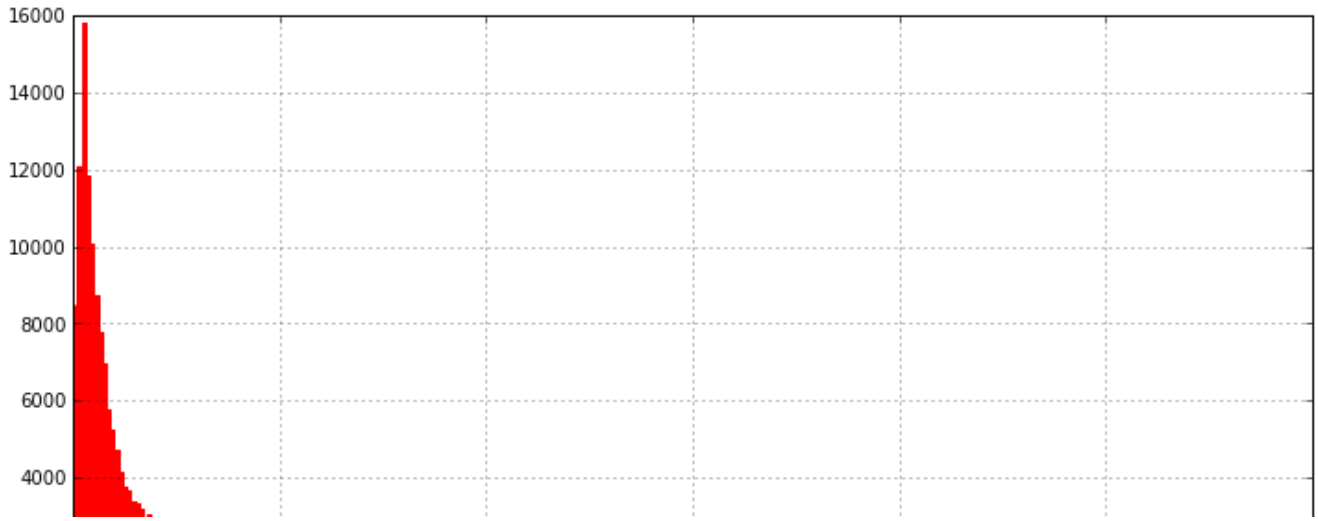
Setup environment

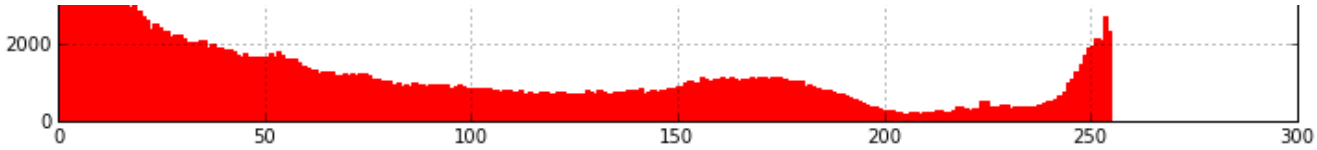
Load in the image

```
Format:    PNG
Size:      (700, 526)
Mean:      76.6811352526
Varence:   6054.15055816
```




Histogram plot of intensity values





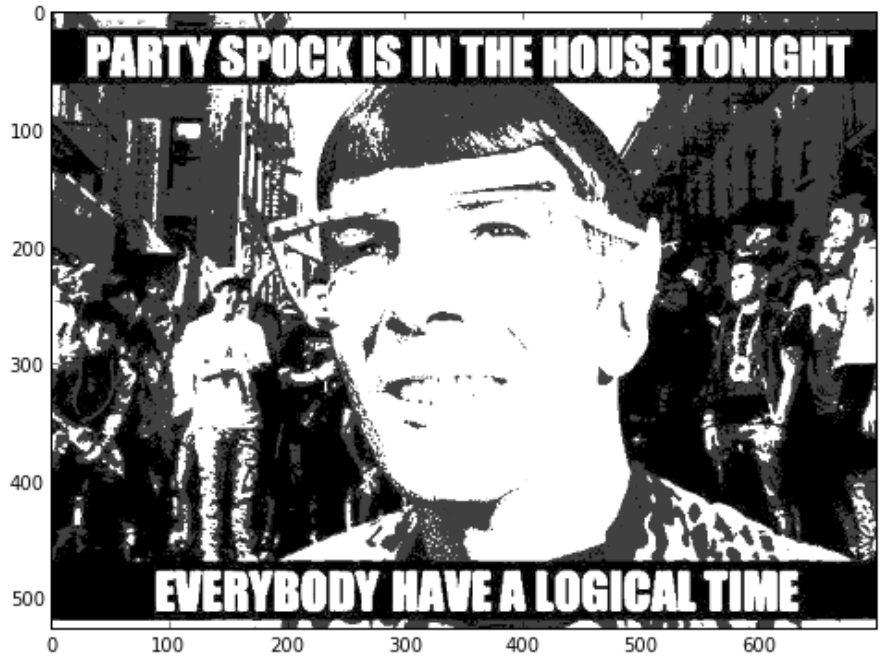

As can be seen from the histogram above, a large portion of intensity values are predominantly black, while lighter values of around 50 or greater are thinly spread across the upper range

# 2) Implement: Gaussian Mixture Models and EM

## Image Segmentation

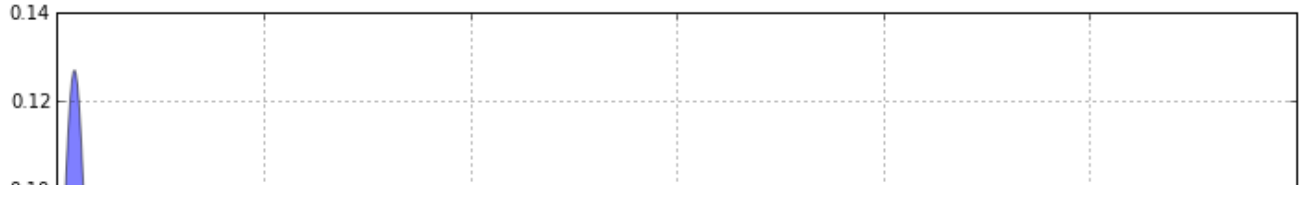
#### 3 Clusters

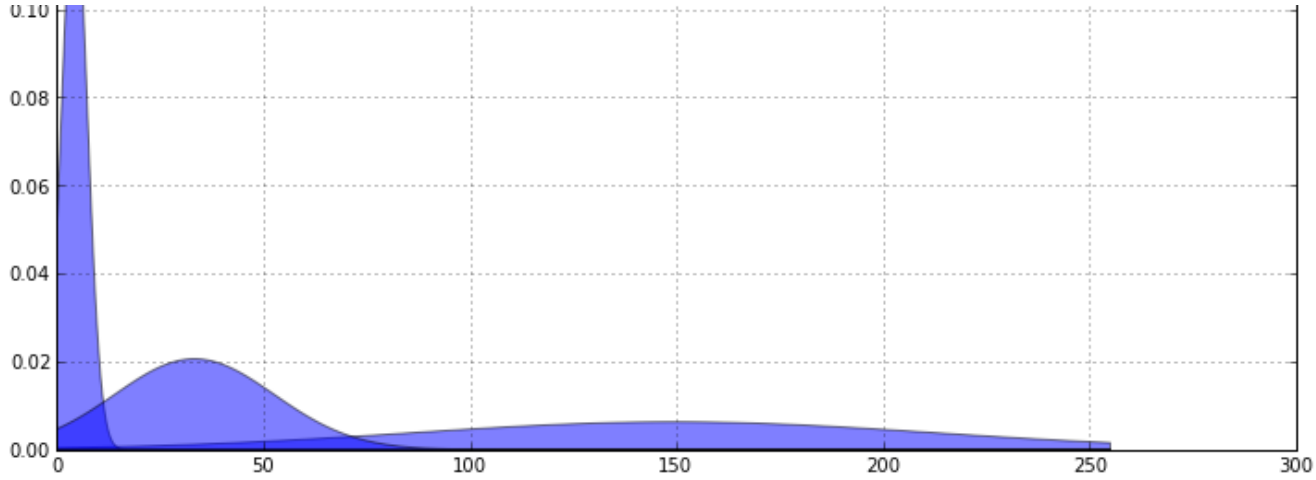
```
Randomization:  [136, 8, 32]
Starting Variance:  [ 6054.15055816  6054.15055816  6054.15055816]
Starting Weights:  [ 0.33333333  0.33333333  0.33333333]
```




```
means [ 149.33417389    4.27212175   33.3123327 ]
variance [ 4014.7618655      9.88485853   374.36545999]
weights [ 0.43523136  0.24543695  0.31933169]
score 8.11666513674
```

Gaussian Mixture Model View

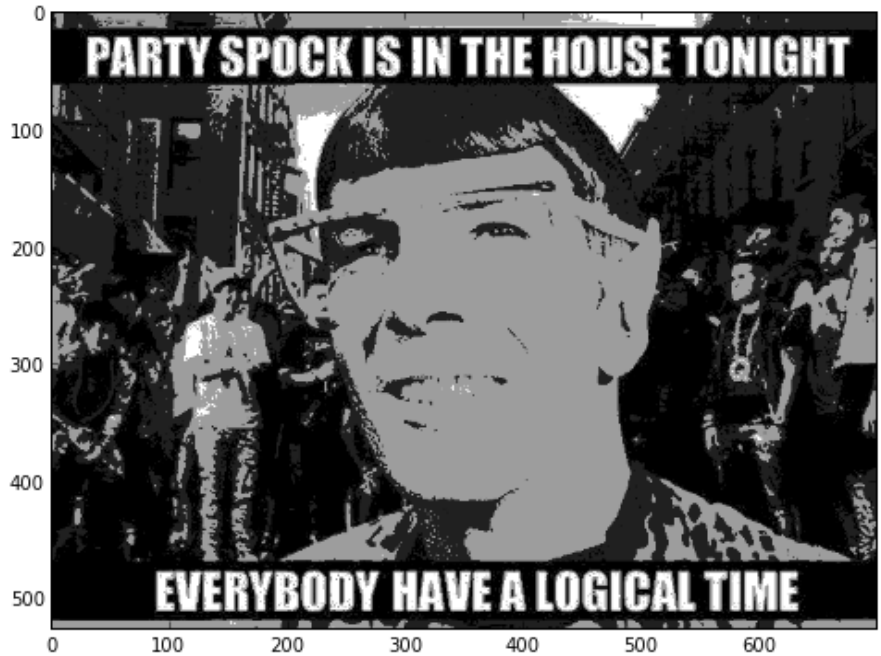





Here, given only three components to work with, we can see that GMM tries its best and converging to solution that appears to roughly mimic the overall distribution of intensity values of the image. The Gaussian far to the left, with a mean of about four, and the small variance of about nine tries to mimic the spike of the large amount of dark values clustered around the lower spectrum, while the wider Gaussian on the right with a mean of roughly 150 and a variance of 4000 tries to accommodate for the thin distribution along the upper spectrum.

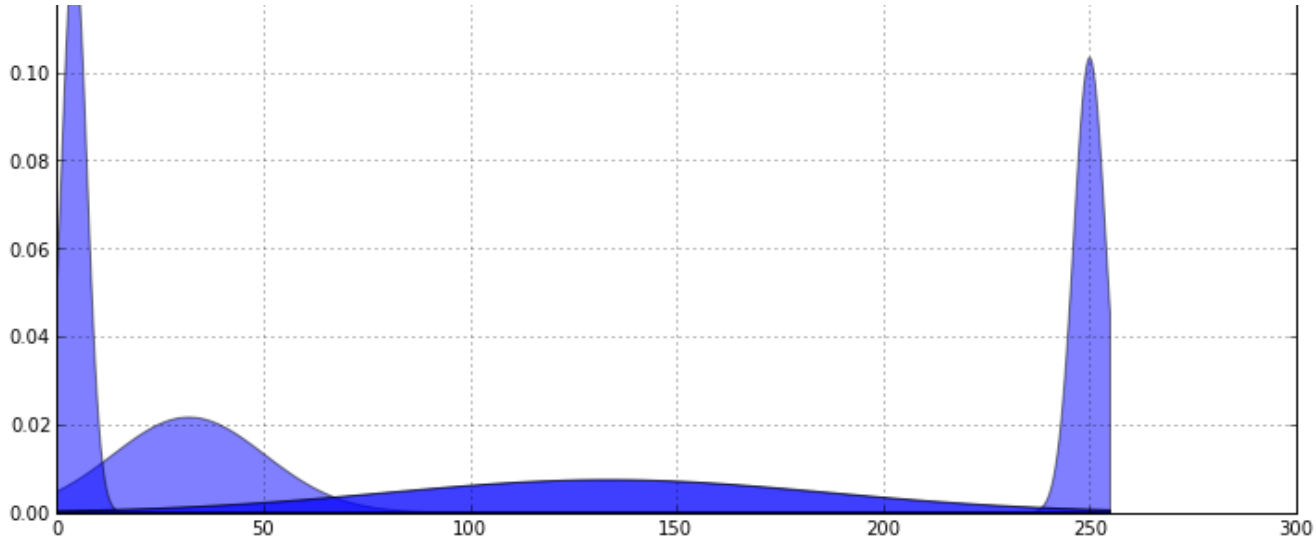
**5 Clusters**

```
Randomization:  [108, 123, 27, 184, 68]
Starting Variance:  [ 6054.15055816  6054.15055816  6054.15055816]
Starting Weights:  [ 0.2  0.2  0.2  0.2  0.2]
```




```
means [  31.96687461  250.03963554    4.22392276  133.65142113  133.11923549]
variance [  339.36632949    14.89389242     9.61995224  2882.18938047  2883.47312995]
weights [ 0.31251675  0.05570939  0.24394348  0.1941305   0.19369987]
score 8.13852635977
```

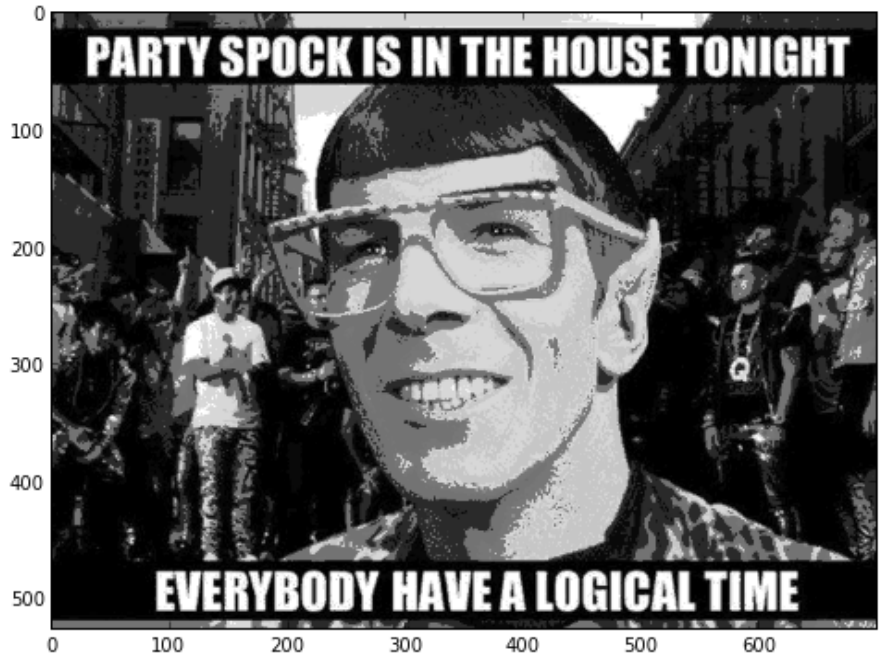
Gaussian Mixture Model View




With the addition of two more components, we are now able to mimic the small spike on the upper spectrum, while a fifth Gaussian is redundantly placed along the center to compensate for the wide and flat segment of the center spectrum. Note how the score has slightly improved due to the better coverage of the spectrum.

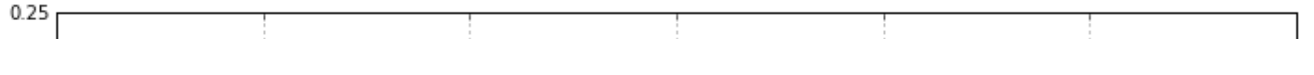
**5 Clusters**

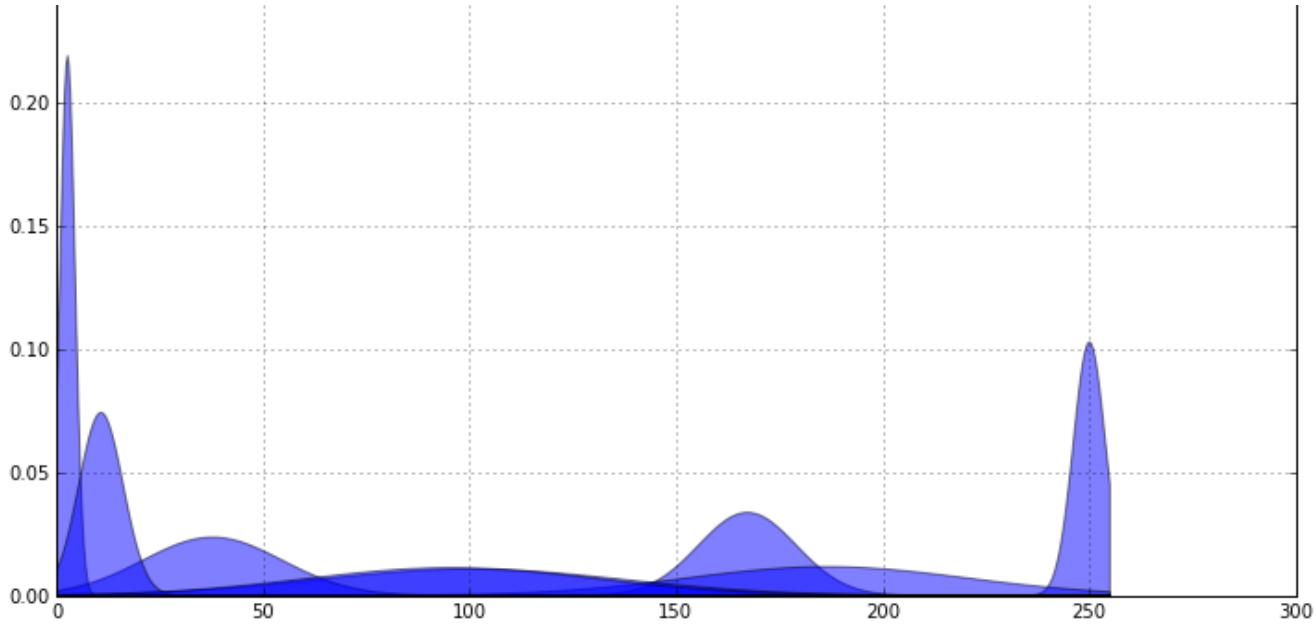
```
Randomization:  [222, 148, 160, 54, 184, 62, 45, 245]
Starting Variance:  [ 6054.15055816  6054.15055816  6054.15055816]
Starting Weights:  [ 0.125  0.125  0.125  0.125  0.125  0.125  0.125  0.125]
```




```
means [ 250.02881739  167.23641531  186.50145713  100.20105156    2.66060198
   37.76343057   10.75188487   95.60258022]
variance [   15.04948925   140.02360522  1157.60243646  1380.90956295     3.32003494
   283.86411056    28.87060654  1242.62698287]
weights [ 0.0563236   0.04916161  0.11574242  0.11052193  0.16798963  0.22618406
  0.1588016   0.11527515]
score 8.20682901821
```

Gaussian Mixture Model View






Using a component, we can now mimic the wide base of the lower spectrum spike using two separate Gaussians, along with the small dimple in the middle of the spectrum. Again, the widest of the Gaussians overlap to form the flatter regions using larger variances, while score does improve with the addidtion of the number of components.

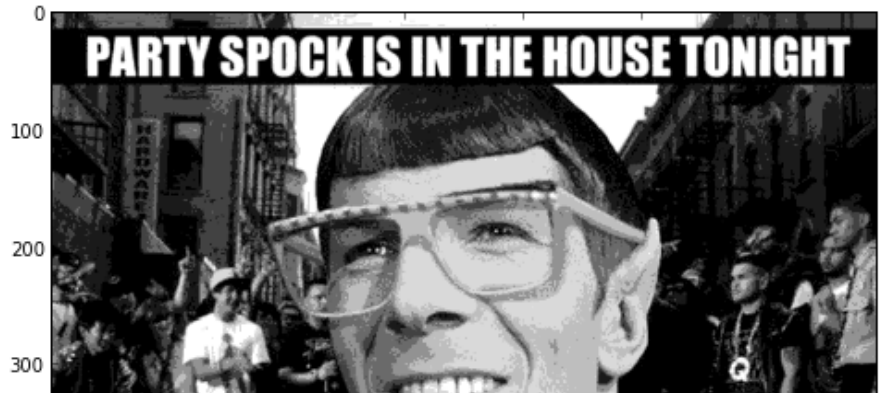
### Design Choices

Among some of my design choices, I decided upon following the same train and fit structure for many machine learning algorithms. By first initializing a created GMM class object with the desired starting means, variances, and weights, we can then give the class an image to compress. This will then 'fit' a Gaussian mixture model to the data provided through an iterative process of expectation maximization. Some other handy features inspired by other machine algorithms include a convergence threshold for score using likelihood, and iterative limits to prevent infinite loops for slow convergence.

## 3) Experiment: Another initialization

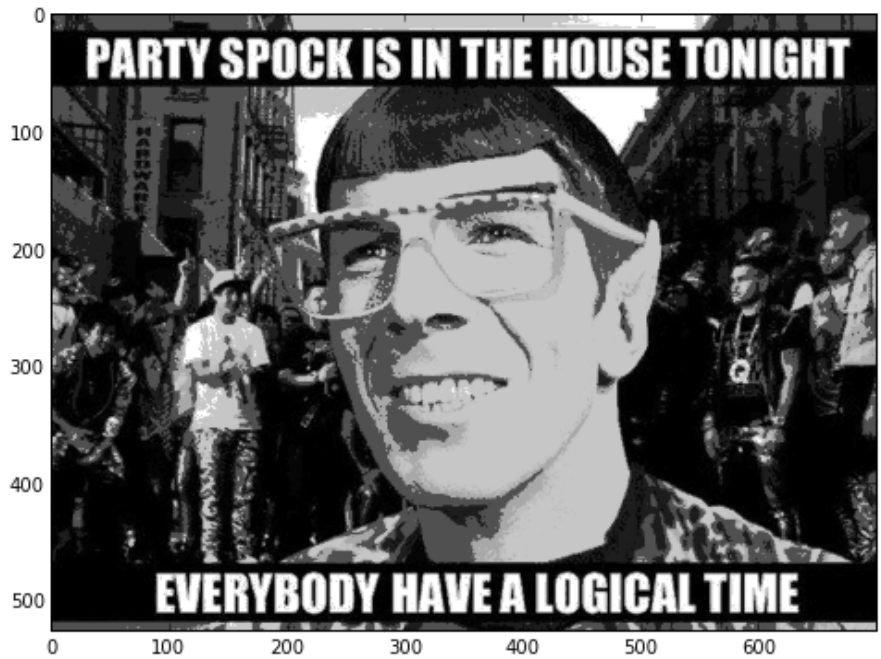
#### Using K-means as a better initialization algorithm

An example image using just k-means with a cluster of 3 on the original image




```
means [  27.97787767  184.68708484   87.01675524  243.17602049  121.40138557
    5.77097319   55.35899028  155.18142066]
```

The K-means algorithm is a simple classifier using similar recursive processes for convergence using a set of components. Although K-means may be a less graceful classifier, especially when dealing with borderline cases and encapsulating entire group in one cluster, K-means is computationally simpler in using Euclidean distances in state space and can also be quite faster with respect to achieving convergence. One method for improving Gaussian mixture model training could be to implement a better initialization method, rather than using random points from the dataset, we could first converge on a set of values using K-means given the number of components, and then use those as a seed for the the components in the Gaussian mixture model, thus helping the expectation maximization start closer to a converging state.




```
Score:  8.2192291327
```

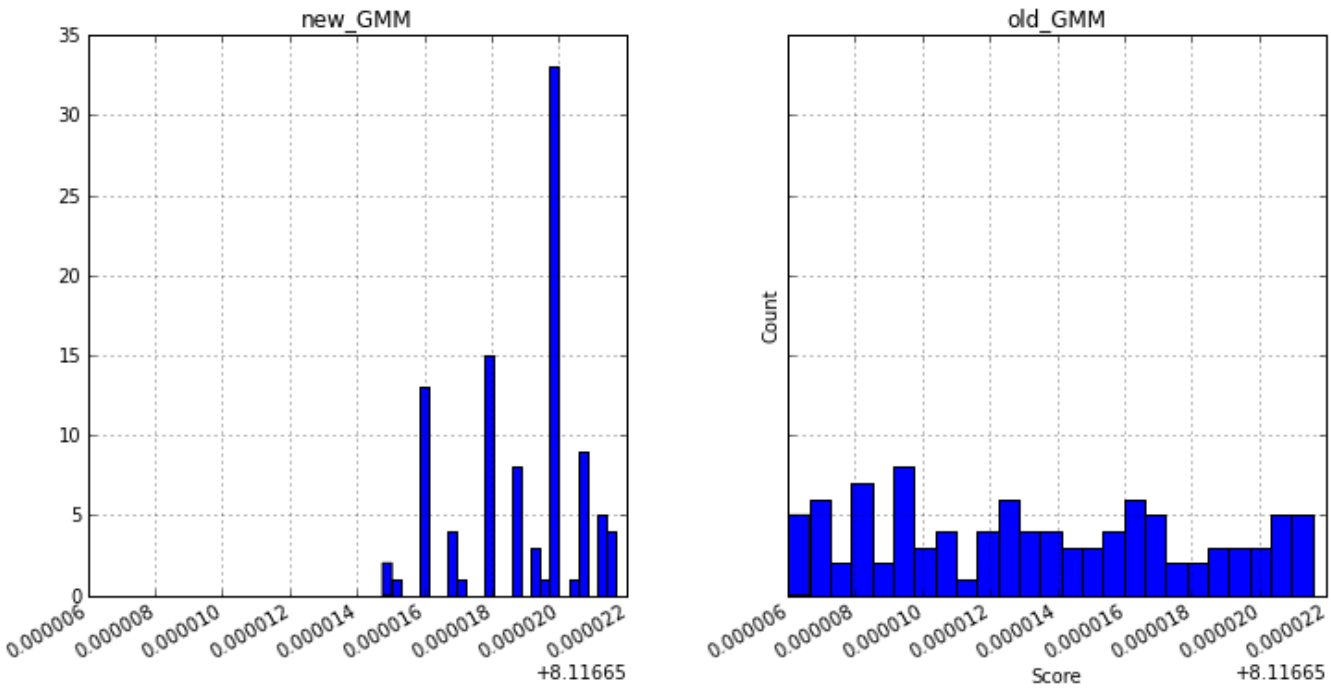
Above, we can see the result of using the component values derived from the K-means execution previously as the seeded values for the initialized component values in the Gaussian mixture model. As compared to the GMM using 8 random samples previously, this image depicts more smoother and flatter textures that may appear less realistic and more cartoonish, but perhaps more aesthetically pleasing thanks to the Headstart provided by K-means.

```
new_GMM_score:  8.11666795718
Trial:  93
old_GMM_score:  8.11666566415
new_GMM_score:  8.11666795718
Trial:  94
old_GMM_score:  8.11666868016
new_GMM_score:  8.11666983486
Trial:  95
old_GMM_score:  8.11666284344
new_GMM_score:  8.1166668349
Trial:  96
old_GMM_score:  8.11666696633
new_GMM_score:  8.11666983486
```

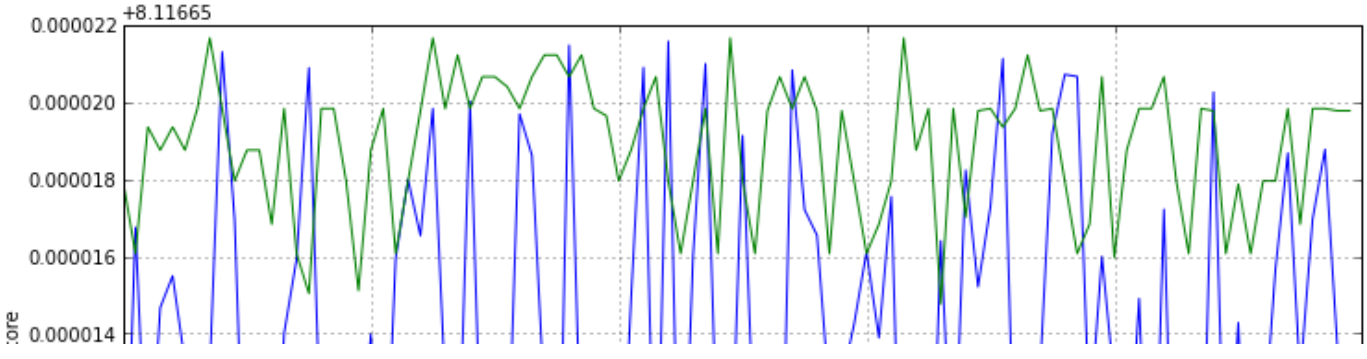
```
Trial:  97
old_GMM_score:  8.11666877505
new_GMM_score:  8.11666983486
Trial:  98
old_GMM_score:  8.11666362866
new_GMM_score:  8.11666977732
Trial:  99
old_GMM_score:  8.11665677796
```

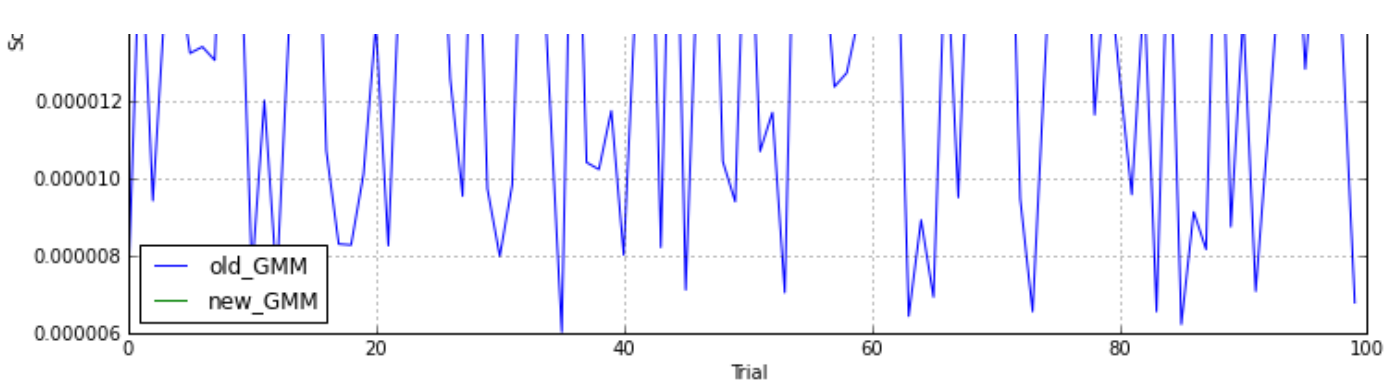
Out[4]:
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 100 entries, 0 to 99
Data columns (total 2 columns):
old_GMM    100  non-null values
new_GMM    100  non-null values
dtypes: float64(2)
```

We can now compare the improvement using k-means as an initialization method analytically by comparing scores between the two over multiple trials. Above the data depicting the scores for Gaussian mixture models using three components both with and without a k-means headstart.




Above are two separate histograms sharing the same X and Y axis depicting the new and old Gaussian mixture model methods. We can see that the new method using K-means scores both higher and more frequently than compared to the random sample method.






Out[6]:

| | old_GMM | new_GMM |
|---|---|---|
| **count** | 100.000000 | 100.000000 |
| **mean** | 8.116663 | 8.116669 |
| **std** | 0.000005 | 0.000002 |
| **min** | 8.116656 | 8.116665 |
| **25%** | 8.116659 | 8.116668 |
| **50%** | 8.116663 | 8.116670 |
| **75%** | 8.116667 | 8.116670 |
| **max** | 8.116672 | 8.116672 |

The same data within the histogram now simply depicted within overlapping line chart showing the progression of scores versus trials. Our new GMM does on average scores better as compared to the older GMM

## 4) Research: Baysian Information Criterion and Mode Selection

### a) How many parameters does a Gaussian Mixture Model have as implemented with 3 components?

A Gaussian Mixture Model keeps track of three terms per component, including a mean, of variance, and weight.
Thus given $3$ components, there will be $3 \times 3$ for a total of $9$ parameters

### b) For a log likelihood of L = -200 and N = 50 and a model with 3 components what is the BIC?

$$BIC = -2 \ln f(y|\hat{\theta}_k) + k \ln(n)$$
$$BIC = -2 \ln L + k \ln(n)$$
$$BIC = -2 \times (-200) + (3 \times 3) \times \ln(50)$$
$$BIC = 400 + 9 \times \ln(50)$$
$$BIC \approx 435.2082070$$

### c) For a log likelihood of L = -200 and N = 50 and a model with 100 components what is the BIC?

$$BIC = -2 \ln f(y|\hat{\theta}_k) + k \ln(n)$$
$$BIC = -2 \ln L + k \ln(n)$$
$$BIC = -2 \times (-200) + (3 \times 100) \times \ln(50)$$
$$BIC = 400 + 300 \times \ln(50)$$
$$BIC \approx 1573.606902$$

### d) For a log likelihood of L = -2000 and N = 50 and a model with 3 components what is the BIC?

$$BIC = -2 \ln f(y|\hat{\theta}_k) + k \ln(n)$$
$$BIC = -2 \ln L + k \ln(n)$$
$$BIC = -2 \times (-2000) + (3 \times 3) \times \ln(50)$$
$$BIC = 4000 + 9 \times \ln(50)$$
$$BIC \approx 4035.208207$$

### e) Explain how the BIC behaves with changing k in relation to the likelihood?

BIC has a goognes-of-fit term $-2 \ln L$
And a penalty term $+k \ln(n)$
Increasing the complexity by elevating the number of components or parameters will thereby be penalizing and increase the BIC. Increasing the likelihood or probability closer to $1$ will thereby decrease the log likelihood to approach $-0$, thus rewarding the BIC to become a smaller value. Decrease the likelihood or probability closer to $0$ will thereby decrease the log likelihood to approach $-\infty$. The double negative with the $-2$ scaler will make this positive, thus making the BIC to become larger.

### f) Why is the BIC score useful for finding k?

In reality, changing the complexity can have an effect of likelihood, for example adding more parameters may improve coverage, and thus improve the strength in the prediction of the model. BIC however takes both factors of complexly of the and strength into account when determining the qualities of a model. This is helpful for when you would like to search for an optimal model but do not know the correct number of parameters to start from. BIC gives you away to compare models and their specific trade-offs of complexly and strength to determine the optimal configuration with the possible realm of numerous configuration.