**Edge Detection:**

**Part1: Gray Edge Gradients:**

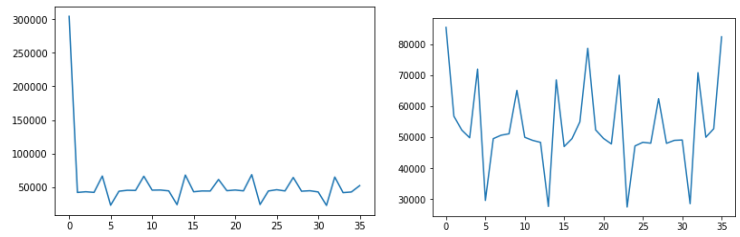
I calculated the edges from the grayscale converted images using Sobel Derivatives. The magnitude can be calculated by applying L1 norm (sum of absolute values of dx and dy) or L2 (sum of squared values of dx and dy). Direction can be calculated using np.arctan2(dx,dy) which gives directions in radians. Alternately, opencv's cartToPolarFunctions takes in both derivates and gives magnitude and direction.

I applied gaussian blur by applying 5*5 gaussian mask before computing sobel derivatives. This helped in removing the noise so that the gradients won't capture the noise as edges.

The following are the examples of gray edge maps that I obtained through my code.

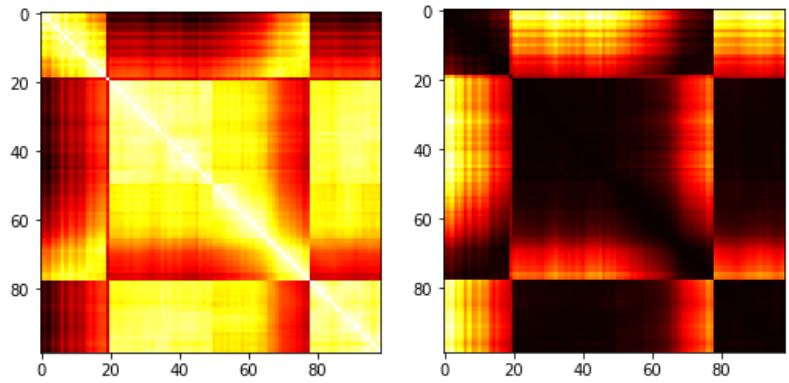
**Gray Edges – Histograms:**

I created the histograms by dividing the directions space into 36 bins. As explained earlier I obtained the edge orientations in degrees and just had to divide by 10 to the bin values. I counted the number of edges in each and used it histogram y-axis.




These are histograms generated for the corresponding edge maps presented above in first section.

I went ahead and calculated the Histogram Intersection (1st below) and Chi-Squared Norm (2nd below) for all the images in the sequence and computed the correlation map.




**Part 2: Color Gradients:**

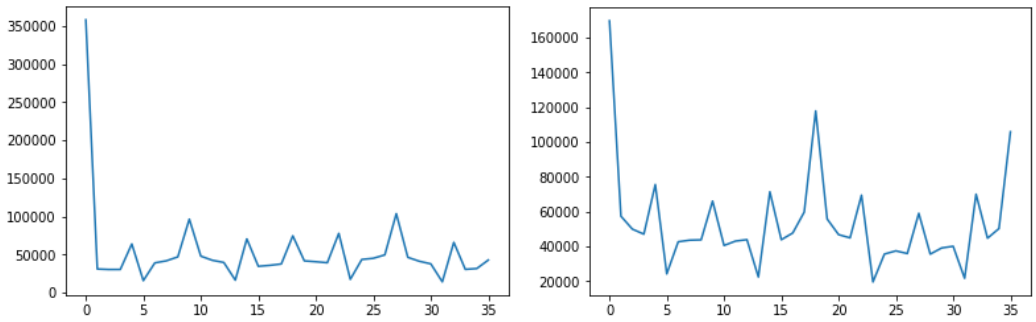
For Color Gradients, I computed the sobel derivatives similar to gray edges but using features of all three-color channels. I used numpy array splitting to carve out the R, G, B values and computed sobel derivatives on top of those, which was run-time efficient. Magnitude is computed by $m = |R_x| + |G_x| + |B_x| + |R_y| + |B_y| + |B_y|$ and direction by $np.arctan2(R_y+G_y+B_y, R_x+G_x+B_x)$ .

I applied the same gaussian blur by applying 5*5 gaussian mask before computing sobel derivatives. This helped in removing the noise so that the gradients won't capture the noise as edges.

The following are the examples of color edge maps that I obtained through my code.

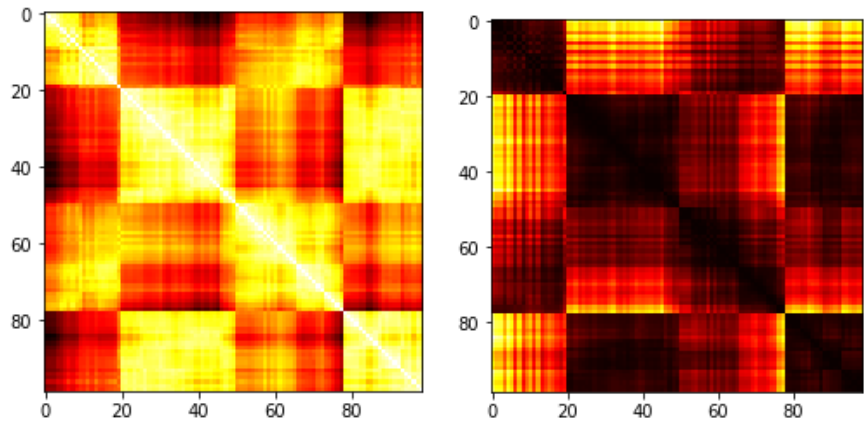
**Color Gradients - Histograms:**

I calculated the color edge histograms similar to gray edge histograms by counting the number of edges and combining the values of all color gradients.




These are histograms generated for the corresponding edge maps presented above in first section.

I went ahead and calculated the Histogram Intersection (1st below) and Chi-Squared Norm (2nd below) for all the images in the sequence and computed the correlation map.



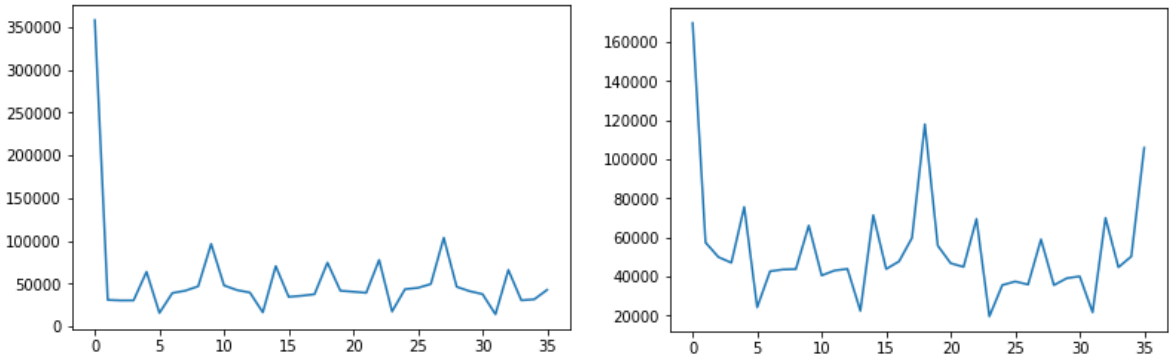

**Part3: Color Gradients with Eigen Values and Eigen Vectors:**

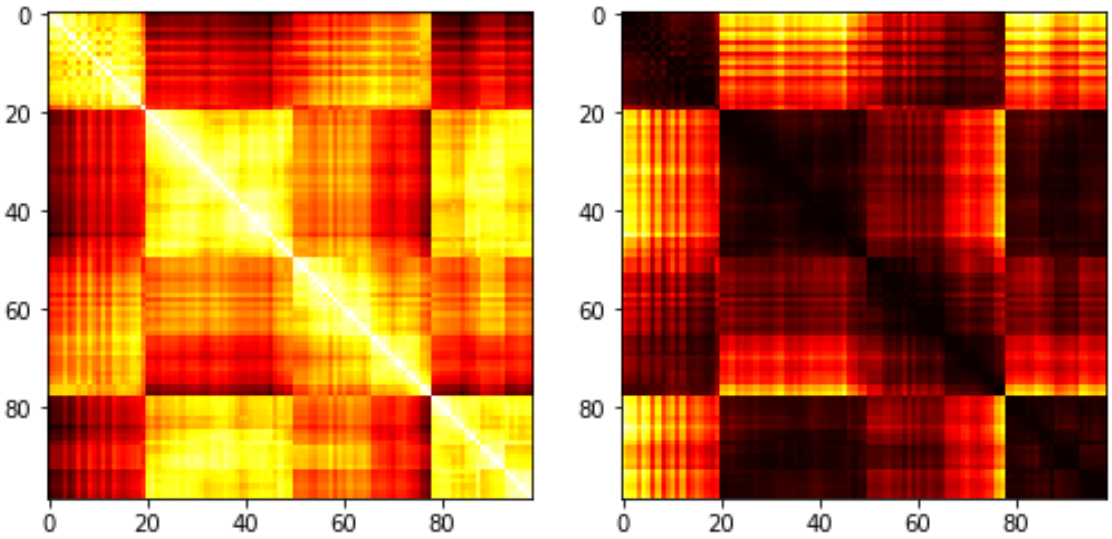
I calculated the Eigen Values and Eigen Vector calculation from (Jacobian) Structure matrix and solving for prominent eigen values and vectors. I used numpy arrays throughout to speed up the computation.

```
a = np.square(sobelRx) + np.square(sobelGx) + np.square(sobelBx)
b = np.multiply(sobelRx,sobelRy) + np.multiply(sobelGx,sobelGy) + np.multiply(sobelBx,sobelBy)
c = np.square(sobelRy) + np.square(sobelGy) + np.square(sobelBy)

#Eigen Values of the structure matrix
lambda1 = ((a+c) + np.sqrt(np.square(a+c)-4*(np.multiply(a,c)-np.square(b)))) / 2
#lambda2 = ((a+c) - np.sqrt(np.square(a+c)-4*(np.multiply(a,c)-np.square(b)))) / 2
x = np.divide((-1*b),(a-lambda1))
e_component1 = np.divide(x,np.sqrt(np.square(x)+1))
e_component2 = np.divide(1,np.sqrt(np.square(x)+1))
eigenVector = np.array([e_component1,e_component2])
magnitude = np.sqrt(lambda1)
direction = np.arctan2(e_component2,e_component1) * 360 / np.pi #radian to degrees

magnitude_8u = np.uint8(magnitude)
cv2.imwrite("eigen_edges/EigenEdges_"+img_name,magnitude_8u)
```

These are the edge maps obtained from Eigen Values as magnitude. It is similar to obtained by color edges and is better at not capturing noise as edges.

**Histograms and scores of Eigen based Gradients and Orientation:**

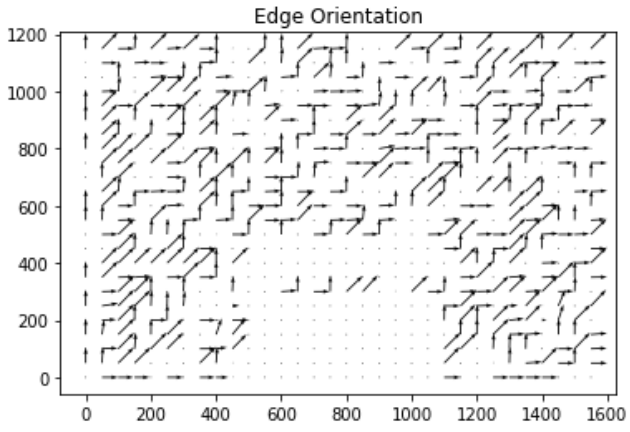



The values obtained for histograms are similar to those obtained by color derivatives in previous section.

**Quiver Plots:**

Orientation of edges gives the direction with respect to each pixel in the image. Quiver plots are a way to visualize these gradients. I created quiver plots for gradients computed based on 2 different images presented above. The arrows are drawn are on **even and well-spaced intervals** to avoid cluttering within the image.




This clearly shows the edges captured from the corresponding image. I rotate the image here for presentation since the quiver has origin in bottom left corner.

The next quiver plot is for the other image in consideration.