BLUR DETECTION GEOSPATIAL

JASON FETZER, Professor Chen

Detecting Location of a Blur after reading a sequence of Images.

REQUIREMENTS

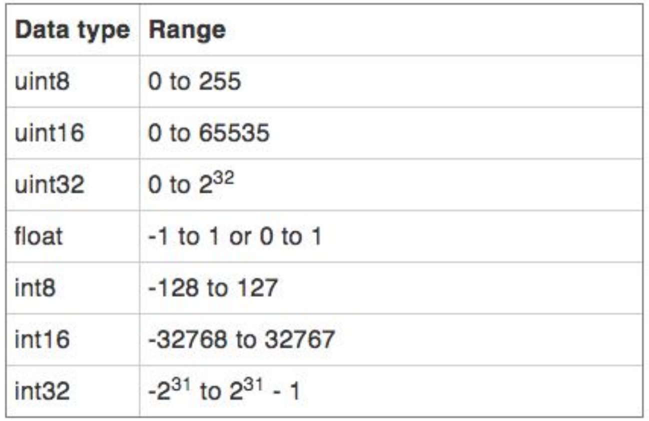
Setup:

Python, Jupyter Notebook, preferred IDE or Editor (Rodeo, Spyder, etc.),

and **A TON OF PATIENCE!**

Packages:

- Cv2
- Numpy
- (ImUtils), Contours
- glob
- matplotlib
- os



Data type	Range
uint8	0 to 255
uint16	0 to 65535
uint32	0 to 2^{32}
float	-1 to 1 or 0 to 1
int8	-128 to 127
int16	-32768 to 32767
int32	-2^{31} to $2^{31} - 1$

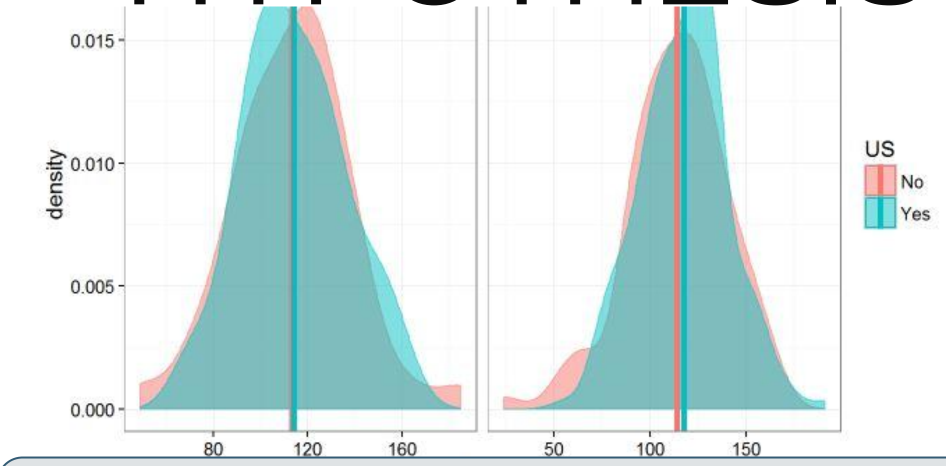
HYPOTHESIS

1. To detect a blur in a **sequence** of images, the blur (essentially) would be more stable in a sequence of images. i.e., this is the area that would change *the least* in numeric representation. (stable is subjective as behind the image would change as well.) This is also only one way of an approach.

2. Therefore, after reading a sequence of images into an array, the matrix representation would produce varying values at each [i][j] over a sequence of time.

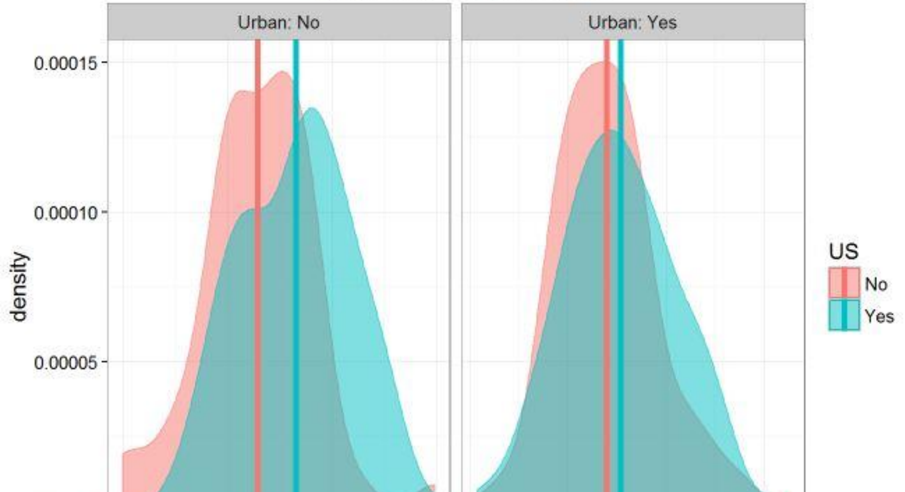
3. Instead of using a minmaxloc() bif from cv2 (or similar i.e. light/dark) where a max/min value is found in one image, it would be beneficial to pass separate filter detectors over each matrix, and then finding [all the areas] where there was least change/variance should lead to the blur.

4. To do this, there are a few filters which can be used.



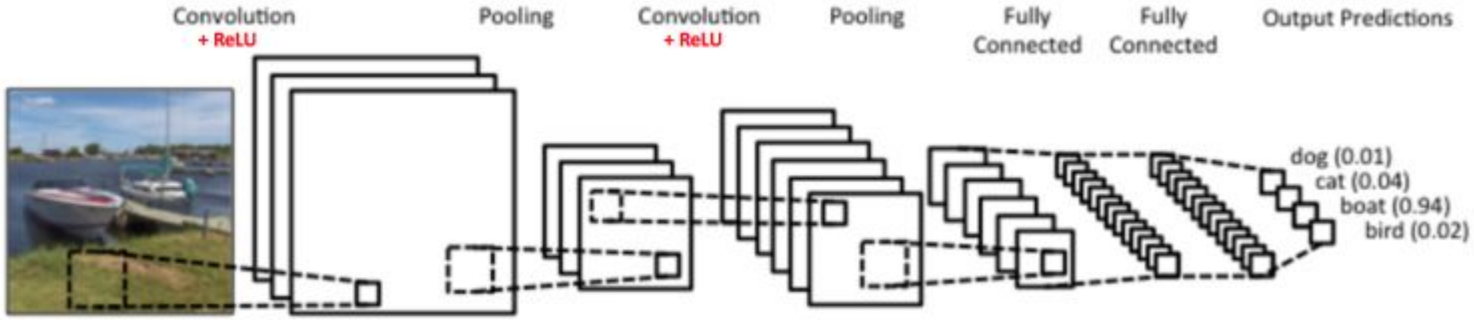
Data

See what each filter does! No Problem!



Traditional CNN

Take an image and “classify” the image. I.e. is it a cat or a dog? MNIST dataset, etc.



Our Case:

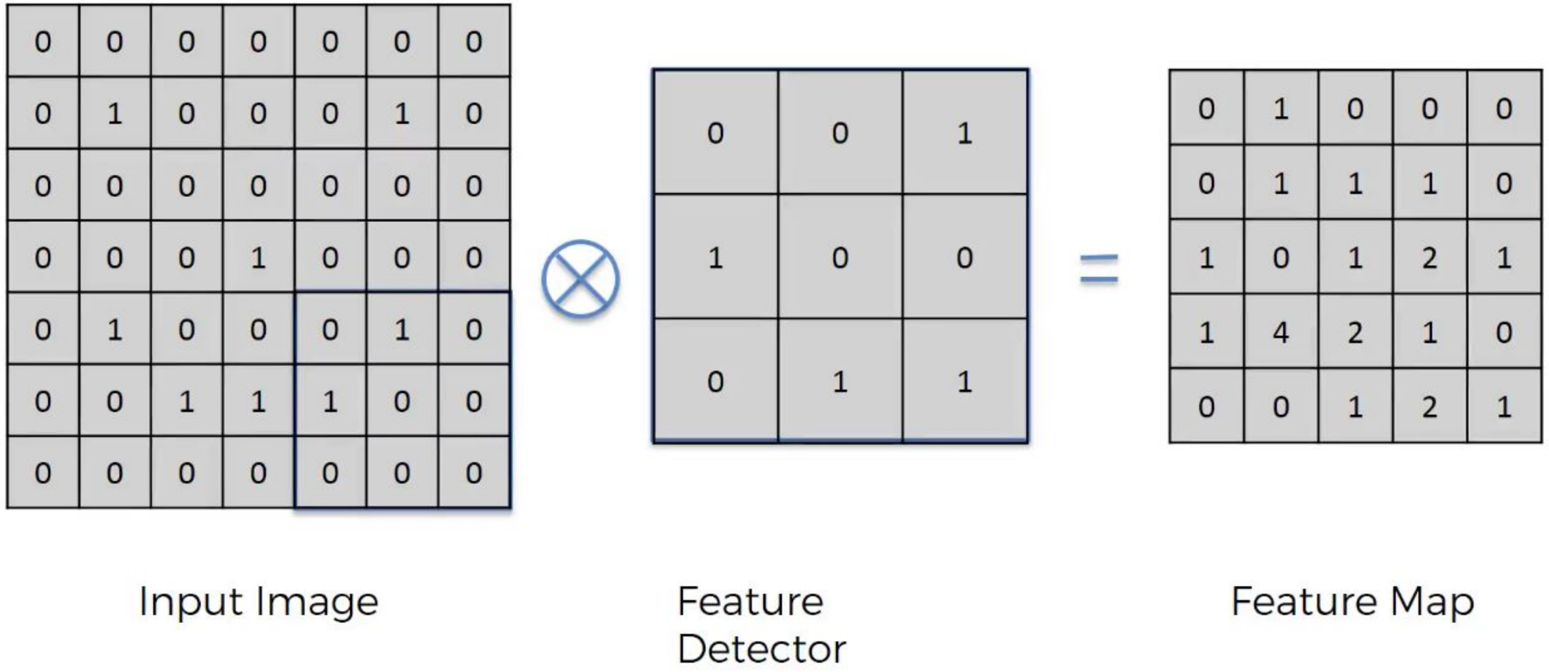
What Part Of the Image is most similar over time, and what is good approach?

Filters!

crisp
0 -1 0
-1 5 -1
0 -1 0

blur
1 1 1
1 1 1
1 1 1

edge
0 1 0
1 -4 0
0 1 0



Input Image

0	0	0	0	0	0	0
0	1	0	0	0	1	0
0	0	0	0	0	0	0
0	0	0	1	0	0	0
0	1	0	0	0	1	0
0	0	1	1	1	0	0
0	0	0	0	0	0	0

⊗

Feature Detector

0	0	1
1	0	0
0	1	1

=

Feature Map

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Wrong Intuition in Hypothesis! -- try to process just **one image.** Process of Least Variance.

An easier way to do this:

1. Resize all the images in the array for greater precision representation of the images numerically.
2. Blur all the images (will give you a finer detail in the long run).
3. Take the **“Average Image”** of all the images seen.
4. Grayscale/Binary the image for crisp lines.
5. Run the edge detection filter over the average image.
6. The filter SHOULD outline the smeared region of the lens.
7. SIMPLE!

PROCESS of LEAST VARIANCE

1. Read a sequence of 100 images as an array into the .exe program.
2. Pass all values to the numpy array, and resize the array to a testable value. (here is 500*500,3).
3. Blur each of the images to return a new sequence.
4. Take the average image of all the blurred images.
5. Find a threshold for the image.
6. Then, pass the edge detection algo over **ONE image.**

****We should be able to draw a shape around the blur, as it would be the area of the feature map with the least change from our list.***

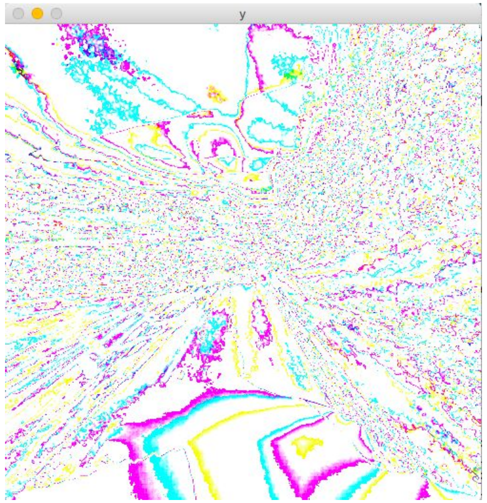
Results

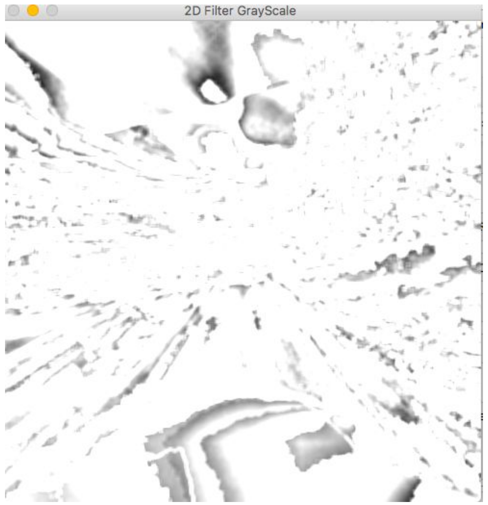
5 cam lense image folders were used, in varying length sequences from each independent folder.

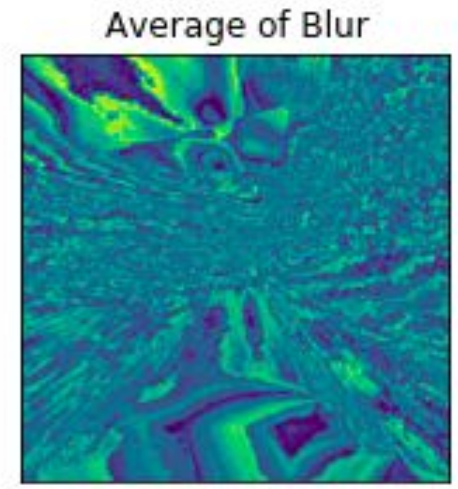
Hey, is that a blur I see?

Data Exploration

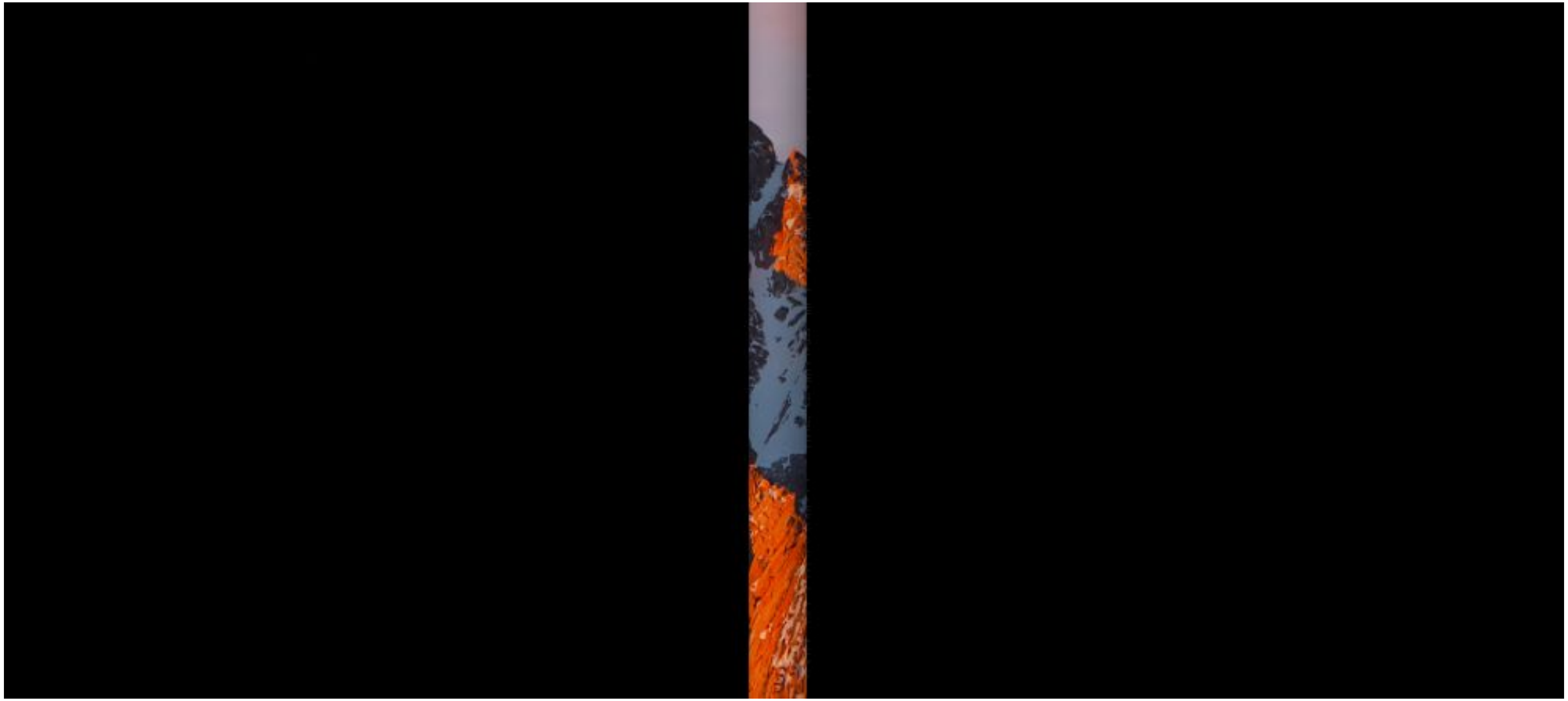
Initial Average Image, Simple GrayScale, Gaussian Blurred Average







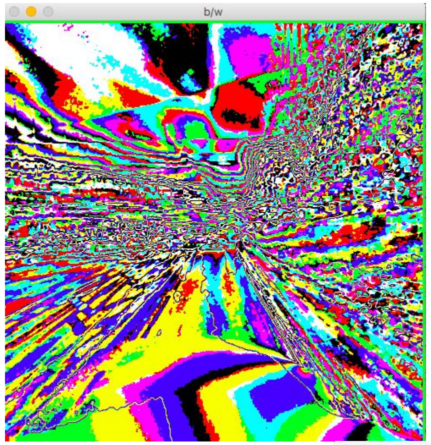
Tuning The Pics WHAT THE?!?!?

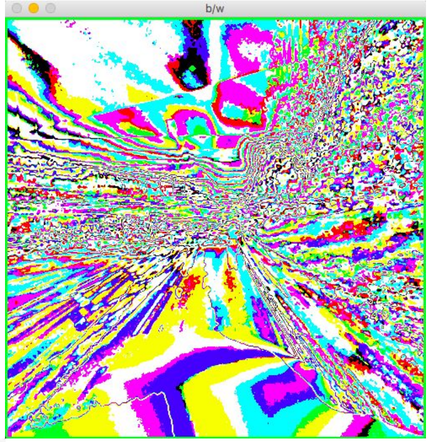


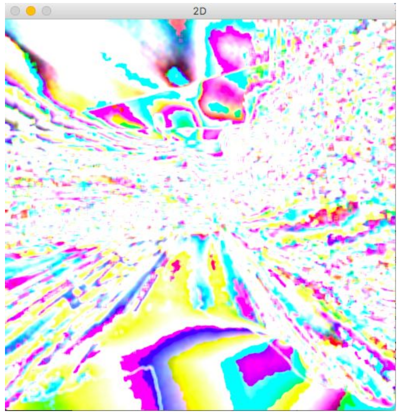
Amplified Color (making progress!)

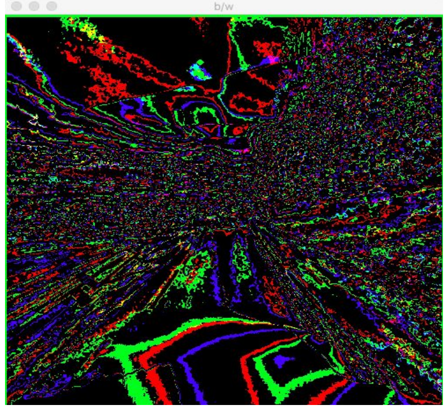
Spotting Blurred (stationary)

REGIONS

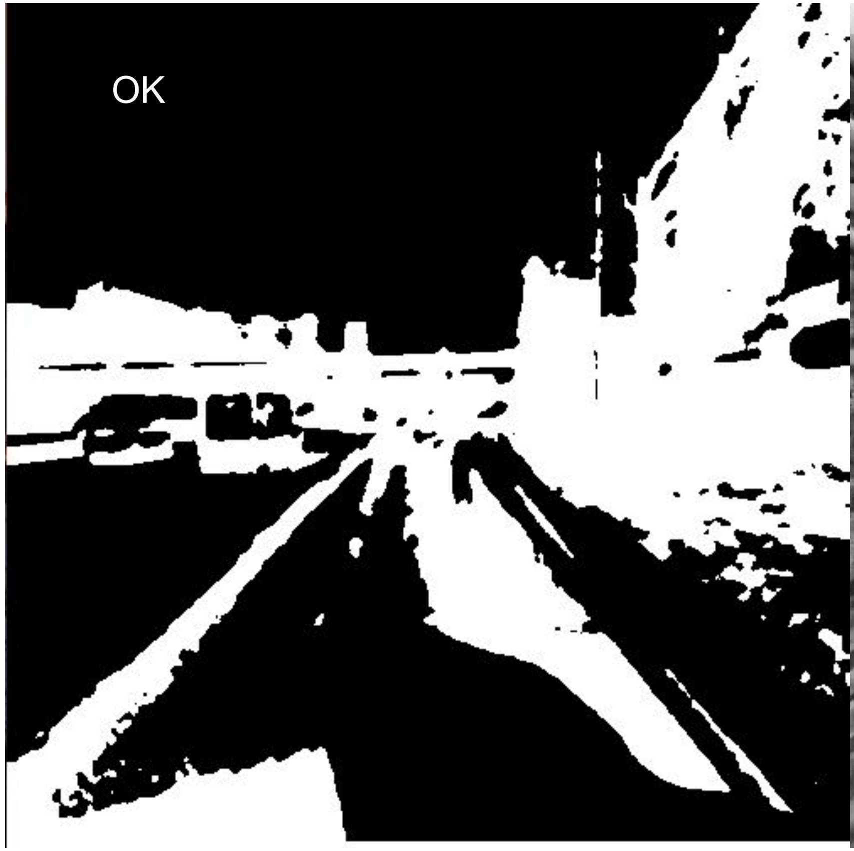




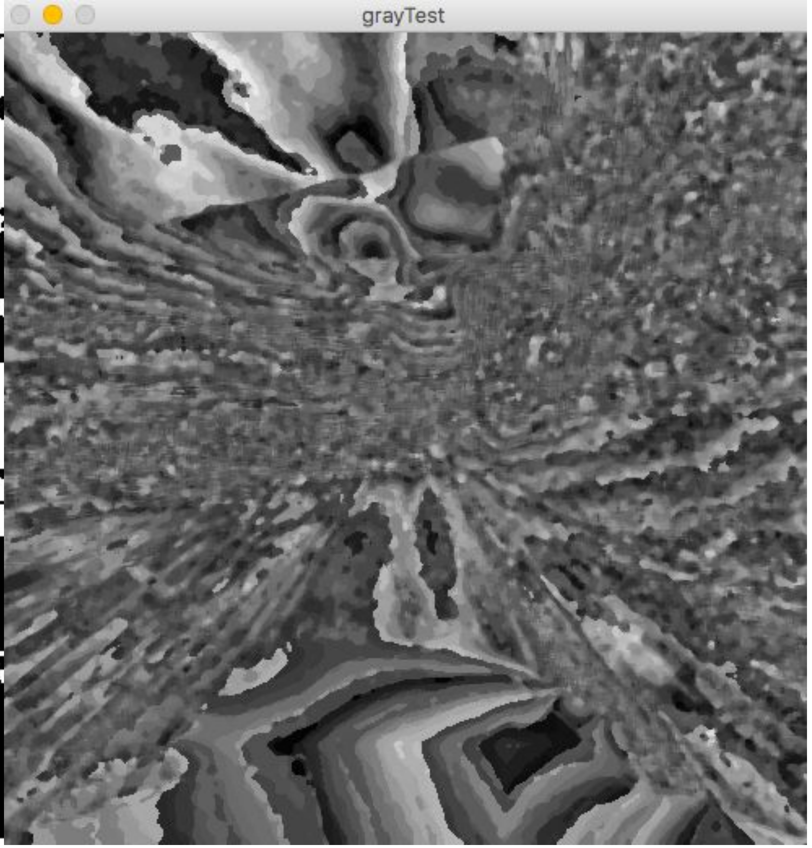




Find a good 0-1 Conversion

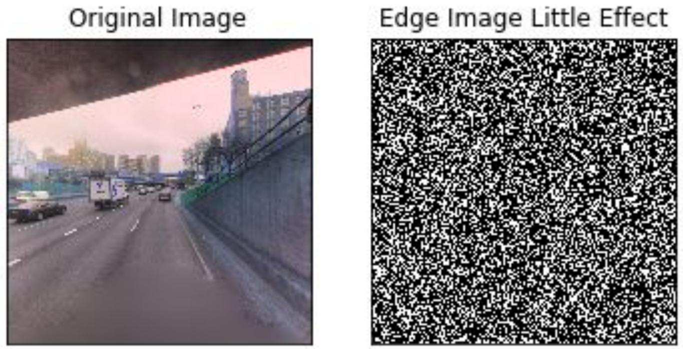


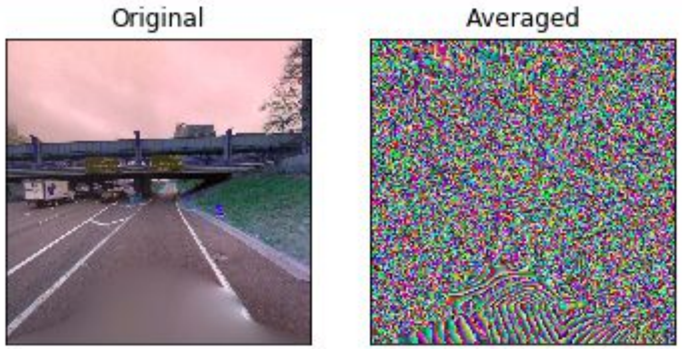
OK

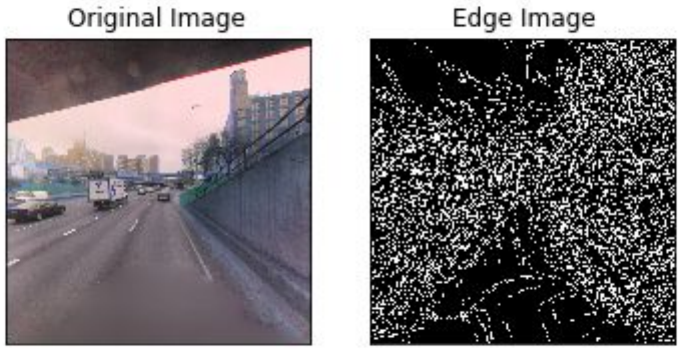


Better Detail

Getting to a Conclusion (a better threshold.)



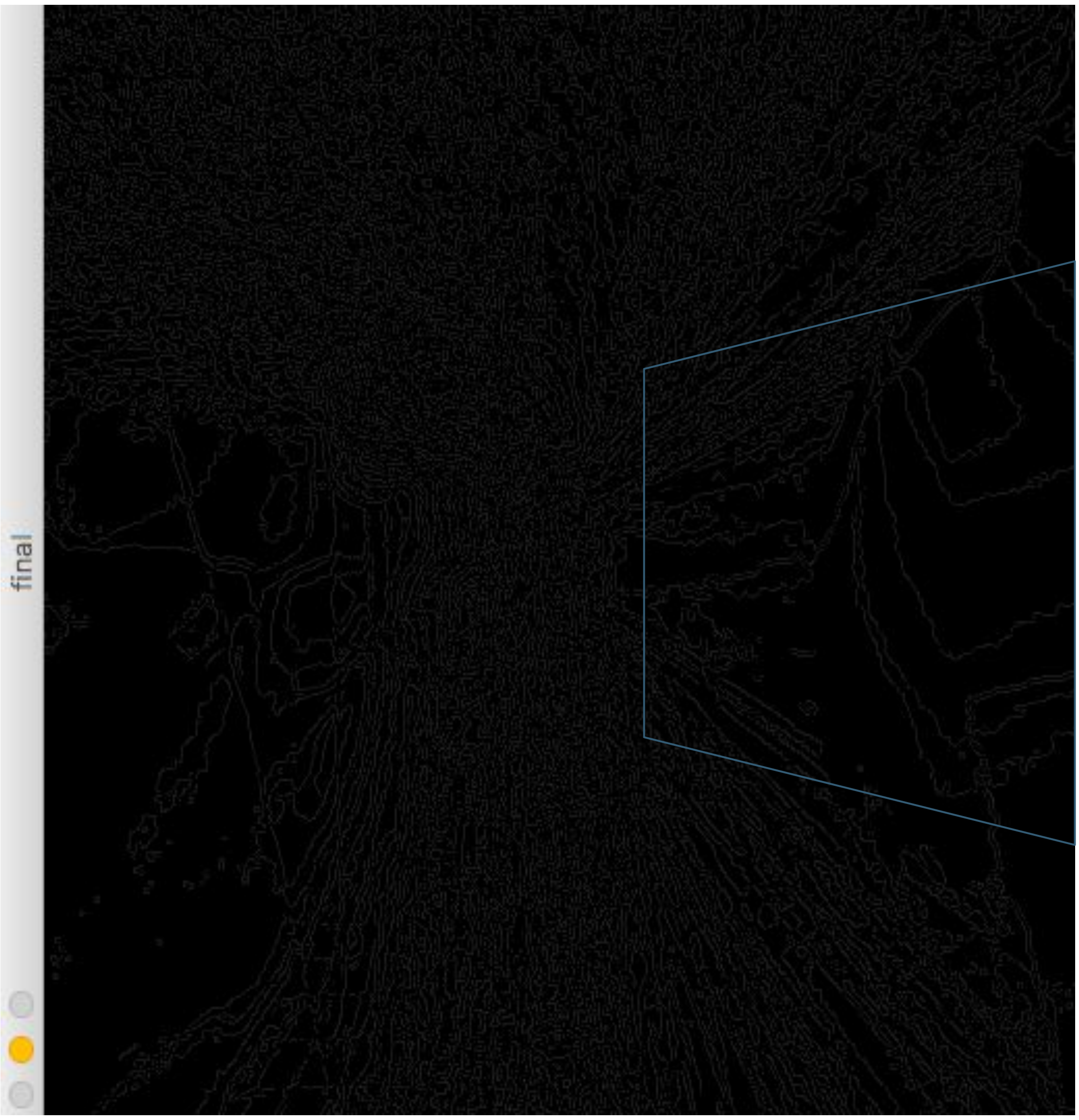




Edged Out

Running Edges, Least Change Inverse

Ramer-Douglas-Peucker Algo



What did I learn?

1. **Computer Vision is difficult!**
2. Some cases are subjective, and you really need a lot of data to test and tune. Especially if you want more than binary results (i.e. blurry/not blurry)
3. I was able to detect a region, but not effectively implement the methods consistently over different cams, or draw the polygon correctly.
4. TA’s are invaluable.

References

Gu, Ramamoorthi, Belhumeur, Nayar. “Removing Image Artifacts Due to Dirty Camera Lenses and Thin Occluders.”

Rosebrock, Adrian. “pyimagesearch.com”. A series of blog posts on computer vision, imutils, and packages in python for Image processing.

OpenCV documentation. “docs.opencv.org”

Eremenko, Kirill. “superdatascience.com.” Machine Learning A-Z, Convolutional Neural Networks.

Class TA’s -- Geospatial Vision and Visualization, were extremely helpful assisting in this project.