<https://github.com/kaltinril/streamGauge>

## Limitations and Assumptions

The banding to make the mask will separate anything with that is different enough in the spatio-temporal domain into its own band, so you could have multiple bands that represent water. This means each unique setup might need to run the code and account and see how the images are banded to determine which bands should be considered water.

The feature labels are highly specific to the camera setup, each setup should train the ANN on training data from that new camera location to ensure accuracy

Some types of water are commonly misclassified, mainly reflections of sharp features (such as poles).

Data is not normalized, and there is no accounting for lighting so lighting conditions can heavily impact results.

## Image Subtractor

**Purpose:** Find the temporal differences between a set of images in a directory

**Methods:**

* **average\_then\_subtract\_images**

|  |
| --- |
| Find the temporal differences between a set of images in a directory  Steps: 1. Go through all images in the image\_directory 2 at a time.  2. Blur both images average\_width and average\_height (reduce small differences in the images)  3. Subtract each pair of images color values to produce the absolute value difference.  4. Add all these pair values together and divide by the total number of pairs (Average)  5. Return an image of the final result  Link:  https://docs.opencv.org/master/d4/d13/tutorial\_py\_filtering.html  :param image\_directory: Source directory that contains the images to be blurred, subtracted, and averaged.  :param average\_width: Width of blur region  :param average\_height: Height of blur region  :param save\_blur: Boolean: Should the image be saved?  :return: Combined image output of all image pairs blurred, subtracted, and averaged. |

* **create\_output\_directory**

|  |
| --- |
| Create a new folder/directory if it does not already exist  :param image\_directory: Directory to put the output folder in  :return: Return the image\_directory combined with the output folder |

* **load\_and\_blur\_image**

|  |
| --- |
| Load an image in, run the OpenCV BLUR method on that image to perform low pass filtering.  :param input\_image: Image to blur  :param pairs: Debug print information  :param average\_width: Size to blur together Horizontally  :param average\_height: Size to blur together Vertically  :return: The blurred image |

* **load\_arguments**

|  |
| --- |
| Load all arguments that were passed in on the command line, and set parameters used in other locations  :param argv: The arguments from the command line  :return: The set values or defaults for: input\_directory, output\_filename, average\_width, average\_height, save\_blur |

* **print\_help**

|  |
| --- |
| Print out command line usage and arguments  :param script\_name: Name of the script, used just to make the help print more specific to this file |

* **resize\_image\_to\_mask**

|  |
| --- |
| If an image size does not match the mask, resize the image to match the mask.  :param image: Image to run through Gabor Filter activations  :param mask: The mask that is used to determine which bands each ROI belongs to  :return: The resized image |

* **save\_blurred\_image**

|  |
| --- |
| Given an output directory and filename, save the passed in imager to that filename in that directory  :param output\_directory: Where the output filename should be saved  :param filename: What the output filename should be  :param image: The output image to save |

## Mask Generator

Take a temporal subtracted aggregate image in and produce a mask image from it

* **build\_mask**

|  |
| --- |
| 1. Take in the source all\_averaged.png file  2. run kmeans with the K-value clusters  3. produce the bands based on the highest k-value in that area  4. Create a bitwise\_and image to remove contradicting sections  5. Save the final mask  :param source\_filename: Image to use as the input to the kmeans mask generation  :param k\_value: Number of clusters the kmeans should use  :param output\_filename: The output final mask  :return: The output final mask |

* **convert\_banded\_to\_unique\_colors**

|  |
| --- |
| Take the banded image, the banded with original color values, and the original kmeans mask  Produce a final image with the banded colors, except when there is overlap between bands  in which case make the color BLACK so we can exclude it during the Gabor filter generation.  :param band\_img: Banded image with values like 20, 40, 80  :param band\_img\_orig: Banded image with values like 0, 1, 2, 3  :param mask: Kmeans mask with values like 0, 1, 2, 3  :return: Banded mask with cut out BLACK area's that are overlapping from the other bands |

* **create\_banding\_gray**

|  |
| --- |
| Take an input mask image and create a image with bands of the max value for that row in grayscale  :param image: The source image to run the banding on  :param band\_size: The size to check horizontally for similar color values  :return:  img = The banded image values 20, 40, 80, etc  img2 = The banded image values, but using the original color values from kmeans |

* **create\_banding\_color**

|  |
| --- |
| Take an input mask image and create a image with bands of the max value for that row in Color  :param image: The source image to run the banding on  :param band\_size: The size to check horizontally for similar color values  :return:  img = The banded image values 20, 40, 80, etc  img2 = The banded image values, but using the original color values from kmeans |

* **extra\_debug\_image\_analysis**

|  |
| --- |
| Create an image that can be used to compare and contrast how the bands will look in the final mask  :param banded\_image: Banded mask image without kimage combination or bitwise\_and  :param k\_image: KMeans mask  :param output\_filename: Output file to save the comparison to |

* **load\_arguments**

|  |
| --- |
| Load all arguments that were passed in on the command line, and set parameters used in other locations  :param argv: The arguments from the command line  :return: The set values or defaults for: source\_filename, output\_filename, k\_value |

* **overlay\_image**

|  |
| --- |
| Take two images and put 1 "ontop" of the other by making the top one slightly transparent  :param overlay: Image to put ontop  :param alpha: Transparency value  :param background: Image to put behind  :return: Combined image |

* **print\_help**

|  |
| --- |
| Print out command line usage and arguments  :param script\_name: Name of the script, used just to make the help print more specific to this file |

* **try\_k\_means**

|  |
| --- |
| Take an input image, and run the OpenCV kmeans implementation against it to reduce the image into K\_VALUE colors.  :param img\_color: The color image to run the kmeans on  :param k\_value: The number of centroids/categories  :return: The Clustered result image |

### Hog Generator

* **data2np**

|  |
| --- |
| Take data, plot it, and return a numpy array of the plotted graph image  :param graph\_data: the data to graph  :return: data converted to ndarrays |

* **build\_filters**

|  |
| --- |
| Builds the filters using the values for ksize and # of orientations  :param orientations: The number of directional vectors to generate kernals for  :param ksize: The size of the Region to use for generating the vectors  :return: Gabor filters to use on the ROI |

* **create\_color\_histogram**

|  |
| --- |
| Calculates the Color histogram for an image and returns a 3 element Python array of arrays of size bins  The size of each array element is [(blues),(greens),(reds)]. Where (color) length = bins count.  :param image: any size image to perform the histogram on  :param bins: (default: 8) - the number of separate slots/groups/bins. 8 means 0-7 is bin 1, 8-15 is bin 2, etc  :return: A Python List of length 3, where each element contains an array of bin values for that color channel. |

* **display\_histogram**

|  |
| --- |
| This is used mainly for debugging and visualization of the Hisogram and Gabor information on a Region from an image.  Displays an image in a plot on the screen  :param bins: # of bins or "orientations" used for the histogram  :param color\_image: The image that the histogram is being run on  :param combined\_image: The combined max value image of all activated gabor kernals  :param img: Greyscale image of the color\_image  :param roi: Argmax pixel values Region from the N kernal activations (This is the bin each pixels direction is)  :param roi\_size: size of the Region of Interest  :param x: Where in the image should we snag the starting ROI X  :param y: Where in the image should we snag the starting ROI Y |

* **load\_hogs\_csv**

|  |
| --- |
| Retrieves the data from all files in a folder, and returns the data and filenames  :param directory: A string that represents the path of the folder containing the hog files  :return: An ndarray containing the all the instances of the hog data, the coordinates, bands |

* **process\_threaded**

|  |
| --- |
| Run the Gabor kernal filters on the ROI (which is the img), to generate the output pixel directions for each kernal  :param img: ROI - Region Of Interest to run the filters on.  :param filters: The filters that were generated by the build\_filters method  :return: combined image max for each filter, and the individual results for each kernal orientation |

* **resize\_image\_to\_mask**

|  |
| --- |
| If an image size does not match the mask, resize the image to match the mask.  :param image: Image to run through Gabor Filter activations  :param mask: The mask that is used to determine which bands each ROI belongs to  :return: The resized image |

* **run\_gabor**

|  |
| --- |
| Takes a source input image, runs the gabor filters on it, associates the location with a banded mask region,  and saves it  :param color\_image: The image to extract features from  :param filters: The return results from build\_filters  :param mask: Banded image with different color values for each band to use as the classifications (Y)  :param image\_filename: Name of the color\_image file  :param orientations: Number of directions/orientations to split 360 into  :param mode: training/validation:  training = generate and save ROI Gabor extracted histograms  validation = Get Gabor histograms and return them for later use by an ANN to validate  :return: Complete set of all Histograms for the input color\_image |

* **run\_gabor\_on\_directory**
* **save\_hogs**

|  |
| --- |
| Saves the HOG generated from the single ROI to the file  :param hog\_info: the data from the HOG to be saved  :param region\_coords: The X,Y position of the upper left of the ROI  :param band: This is the "Y" value or prediction/category/classification  :param output\_file: already opened file to save to  :return: nothing |

## Neural Network

* **view\_predict**

|  |
| --- |
| Overlay the predicted bands onto the original image for comparison and manual human evaluation  :param base\_image: The image that was used to validate  :param pixel\_prediction: The band predictions per ROI |

* **predict**

|  |
| --- |
| Load the trained ANN, run the Gabor histogram generation on the validation image, return the predicted band  values at each ROI area  :param ann\_loc: Location of the Trained NN file  :param color\_img: The image to use to validate with  :param combined\_filename: The filename of the color\_img  :param mask: The mask that was used for the training, not really used but passed into Gabor  :return: The predicted band values |

* **train**

|  |
| --- |
| Load all Gabor filter feature data from the files in the data\_loc folder,  Train a Artificial Neural Network with that data  Save the trained model to a file ann\_1.pk1  :param data\_loc: Folder where the csv files are located |