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