EXPERIMENT - 1 FINAL REPORT

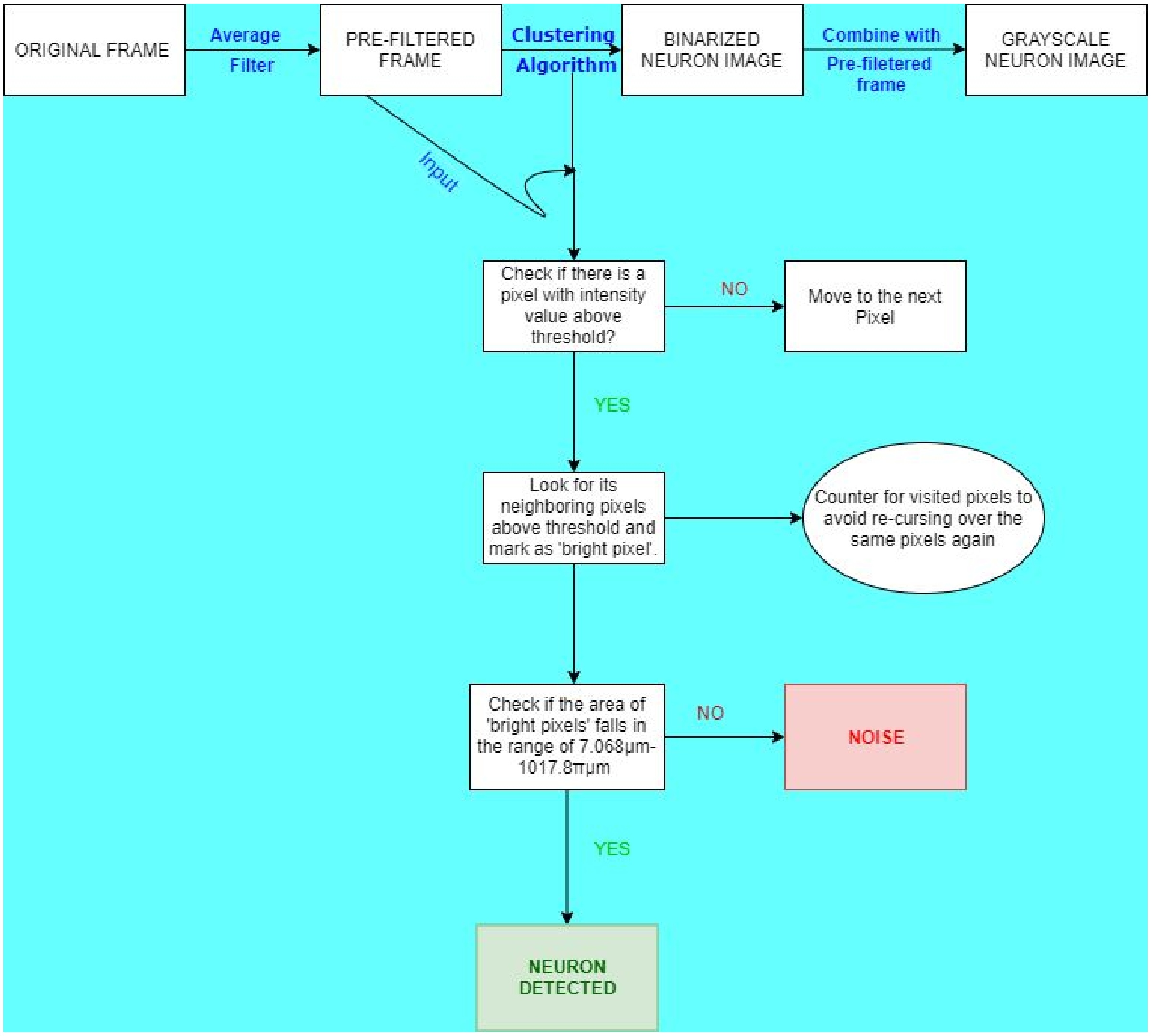
CALCIUM VIDEO DENOISING

Archana Narayanan, Malcolm Miller

VIDEO DENOISING:

For video denoising, the processing is done frame by frame. First, the average filter is applied across all frames to remove white noise. The result is then processed using our clustering algorithm for further denoising and neuron detection.

The clustering algorithm works as shown below:


ORIGINAL FRAME
Average Filter
PRE-FILTERED FRAME
Clustering Algorithm
BINARIZED NEURON IMAGE
Combine with Pre-filetered frame
GRAYSCALE NEURON IMAGE
Input
Check if there is a pixel with intensity value above threshold?
NO
Move to the next Pixel
YES
Look for its neighboring pixels above threshold and mark as 'bright pixel'.
Counter for visited pixels to avoid re-cursing over the same pixels again
Check if the area of 'bright pixels' falls in the range of 7.068μm-1017.8πμm
NO
NOISE
YES
NEURON DETECTED

The result of the average filter is taken as input to the clustering algorithm applied on each frame. The algorithm looks for a pixel value which is greater than the threshold value. This pixel is called the ‘target pixel’ since it could be a part of a potential neuron. A ‘visited’ counter is maintained to keep track of pixels visited to avoid processing the same pixels again. For each ‘target pixel’, the algorithm checks to see if any of its 8 neighbors (that have not yet been visited) are above our target threshold intensity. When a neighbor meets this condition, it is queued to be a future ‘target pixel’. We know that,

Area of pixel : $0.9\mu m * 0.9\mu m = 0.81\mu m^2$
Diameter of Firing Neuron : $3\mu m \leq$ Diameter $\leq 18\mu m$
Area of Firing Neuron : $1.52\pi\mu m \leq$ Area $\leq 182\pi\mu m \Rightarrow 7.068\mu m \leq$ Area $\leq 1017.8\pi\mu m$
Number of Pixels contained in a Firing Neuron : $8.72 \leq$ #Pixels ≤ 1256.

Thus, if the number of pixels in a connected cluster are in the range of $8.72 \leq$ #Pixels ≤ 1256, we consider them to be a neuron. Otherwise, the pixels are treated as noise. The pixels that form the neuron are given the intensity 255. Pixels identified as noise are given the intensity 0. This constitutes the binarized neuron image.

Pixels that are a part of the neuron in the Binarized Neuron Image are then mapped with their respective intensity values from the pre-filtered(average filtered) frame to produce the Grayscale Neuron Image.

The resulting grayscale Neuron Image therefore contains only clusters of pixels that constitute a neuron with its respective intensity values. This process is repeated over all the 500 frames to denoise the complete video. This way, every other pixel that is not a part of a neuron is eliminated.

The results for frame 50 and frame 100 are displayed below. A visual comparison can be made between the results of the average filter and the results obtained after the clustering algorithm. Therefore, we can conclude that our algorithm performs better than the averaging filter since the averaging filter mostly removes white noise and does not completely denoise the video.

SELECTION OF THRESHOLD INTENSITY VALUE:

The histogram of the entire video was computed to understand the distribution of brightness values in the image. Additionally, we used the data cursor to identify pixel intensities of both neurons and noise within the average filtered video. The algorithm was tested with a range of threshold values to identify the most efficient threshold value to detect all neurons. The most appropriate threshold value involved two different luminosities. Our algorithm tests for pixels that have an intensity ≥ 80. Once a valid pixel is found, its neighbors must have an intensity $\geq$

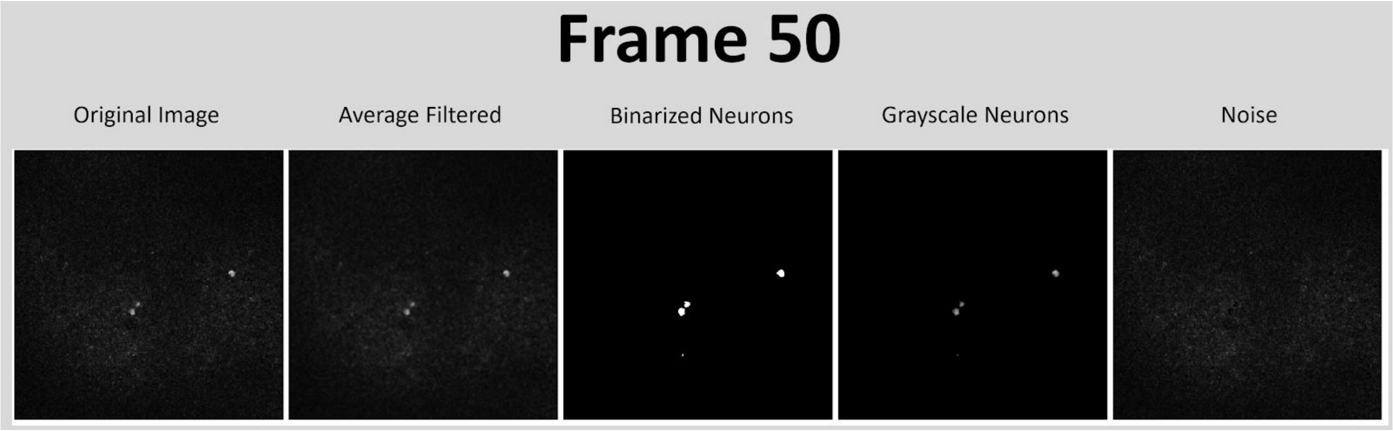
$\frac{2}{3} * 80 \approx 53.33$ in order to be considered part of a valid cluster. Thus, all neuron pixels have intensities over 53.33 .

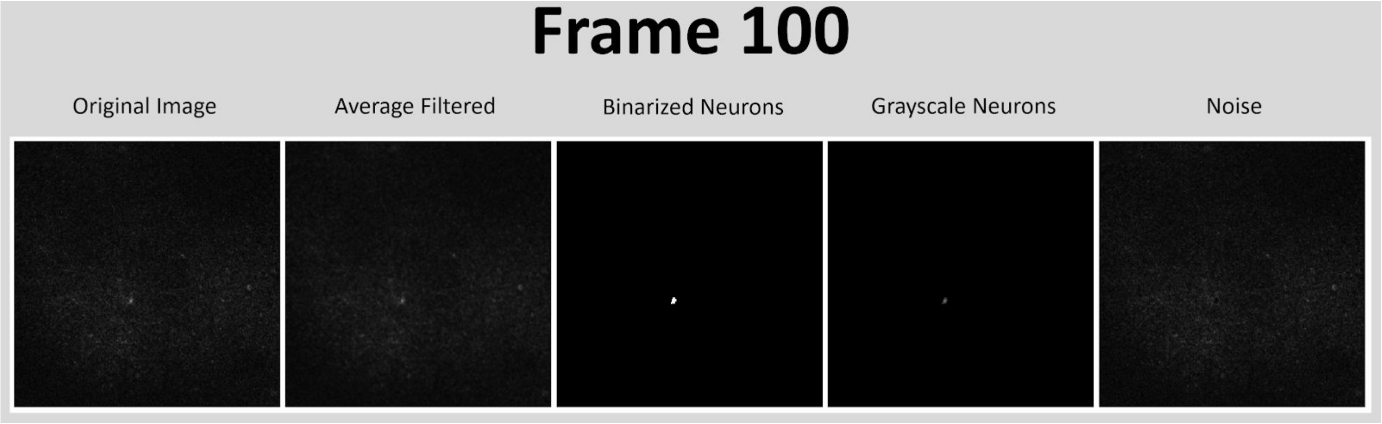
GENERALIZATION AND QUALITY:

Although this threshold was chosen specifically for this video, we believe that by tuning the threshold to fit a particular video, our algorithm can properly denoise any video of this type.

Since all the pixels that are not a part of the neuron are eliminated, the video is completely void of noise leading to quality results that are effective for further processing.

RESULTS FOR FRAME-50 AND FRAME-100:



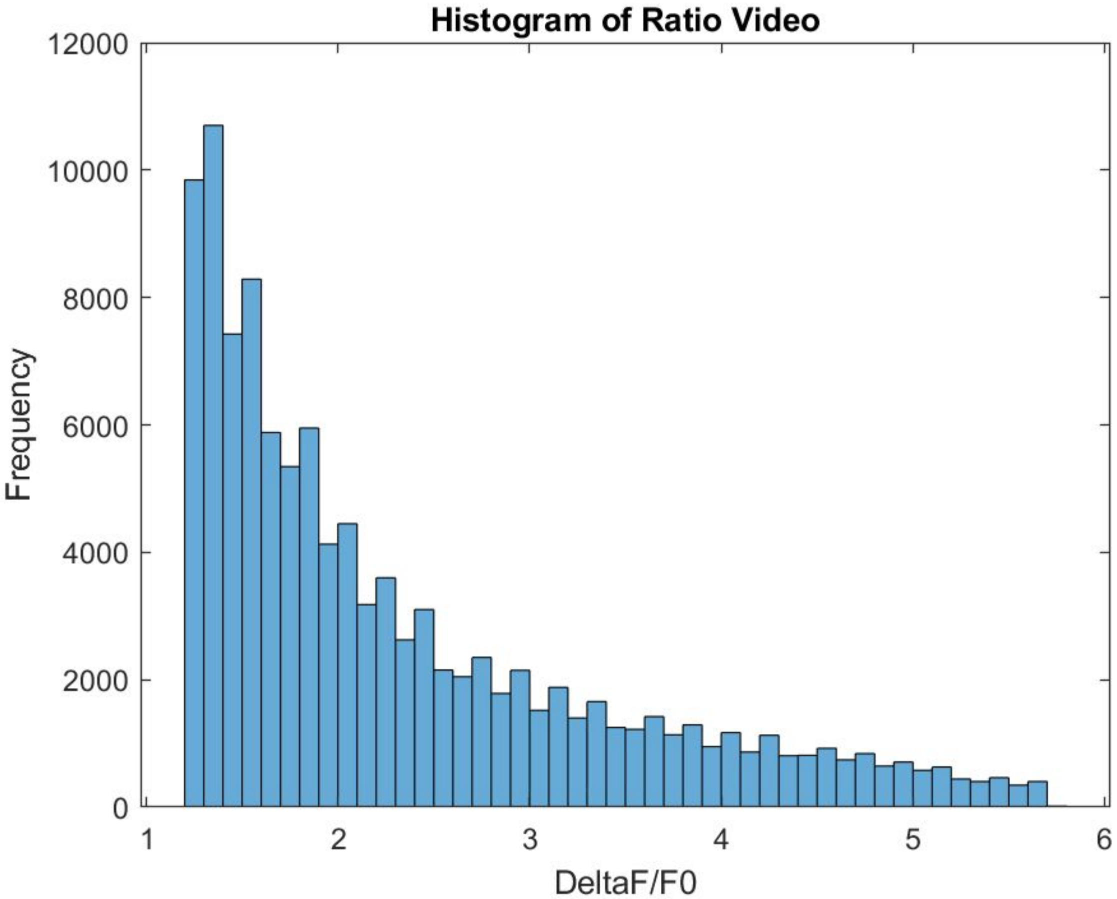


NEURON FIRING DETECTION:

Since the output of our filtering is a series of fames containing either the pixel intensity of the average filtered video (for a detected neuron) or 0 (for noise), we can identify the beginning of a firing event as the transition of a group of pixels from dark (all zeros) to a nonzero value. Since the clustering algorithm required a pixel intensity of 53.33 (out of 255), all neuron values will have intensities greater than this value. Since biologists often determine a baseline level of luminescence, F_0, such that spikes in intensity are $\geq 1.2F_0$, we will define our F_0 as follows: $1.2F_0 = 53.33 \Rightarrow F_0 \approx 44.4$

A few modifications to this algorithm that greatly sped up our execution times were preallocating space in memory for each array of frames and using uint8 data types (instead of double precision floating point values).

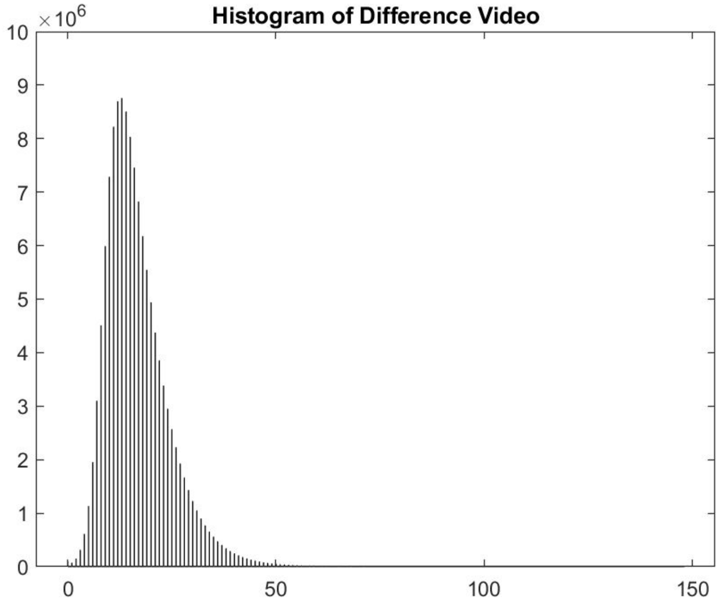
HISTOGRAM OF $\Delta F/F_0$ VALUES:



Note: this histogram only takes into account pixels of neurons (thus, all intensities had to be greater than 53.33).

NOISE CHARACTERIZATION:

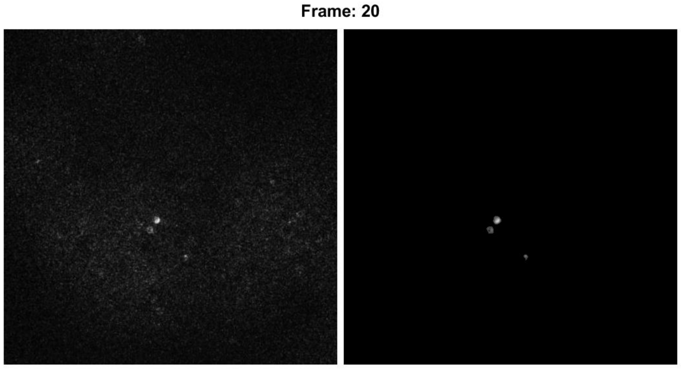
In order to characterize the noise, we took the difference between the original frames and the filtered frames. Next, we created a histogram that represents every pixel of every frame in the difference video:



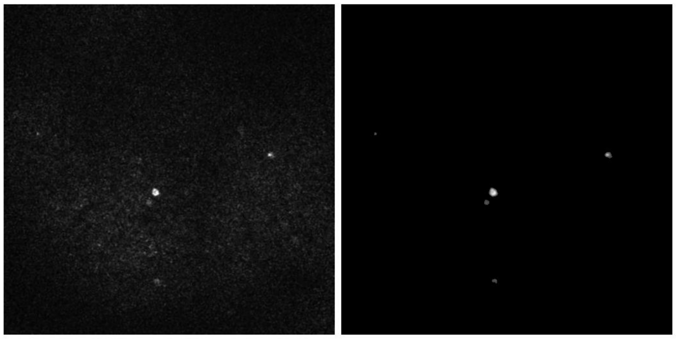
As seen above, the noise appears to have a poisson distribution. This makes sense because shot noise has a poisson distribution. Shot noise is present in many medical imaging settings, including functional multi neuron calcium imaging (fMCI) (Okada et al., 2016, p.1).

DETECTION OF FIRINGS IN FRAMES 20,30,40,50&60:

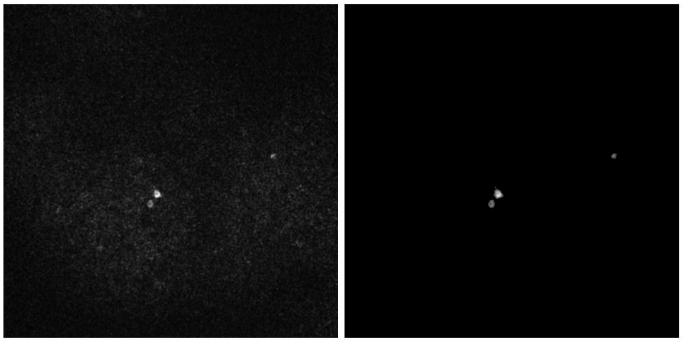
The left hand side shows the frame from the original video and the right hand side shows the filtered frame detecting firing events in neurons.



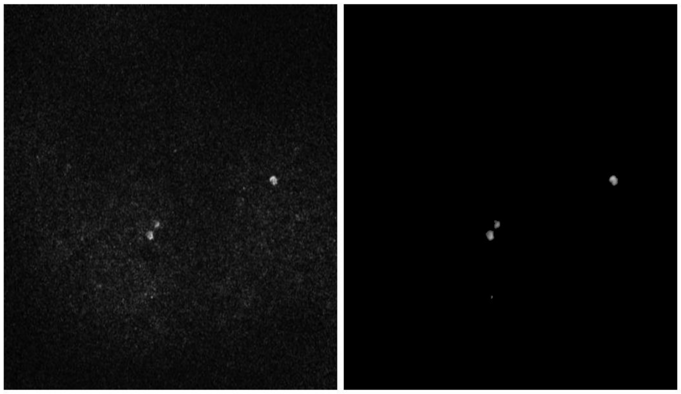
Frame: 30

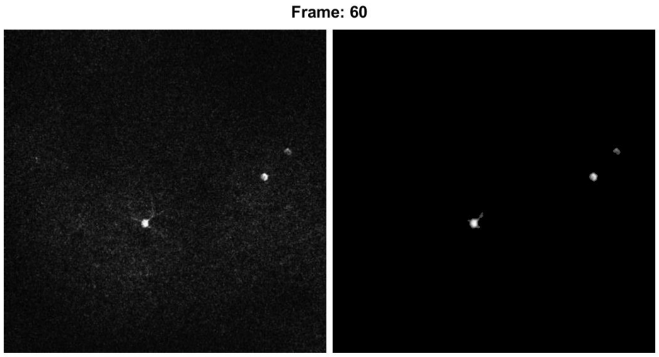


Frame: 40



Frame: 50





TOTAL NUMBER OF FIRING EVENTS DETECTED:

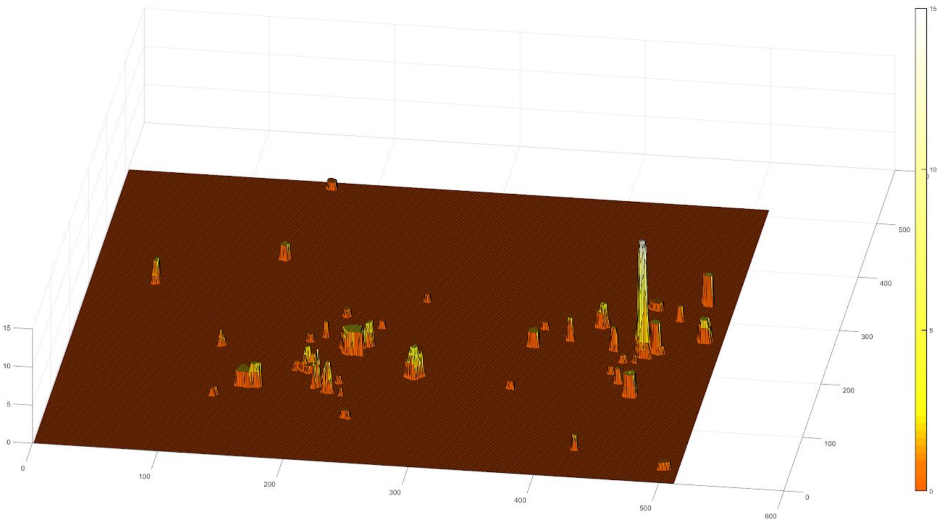
We take the maximum number of firings from each cluster from the countMatrix(which contains the number of firings of each cluster) and sum the values to obtain the result. The total number of firings detected from all the neurons across the 500 frames in the movie is **93**.

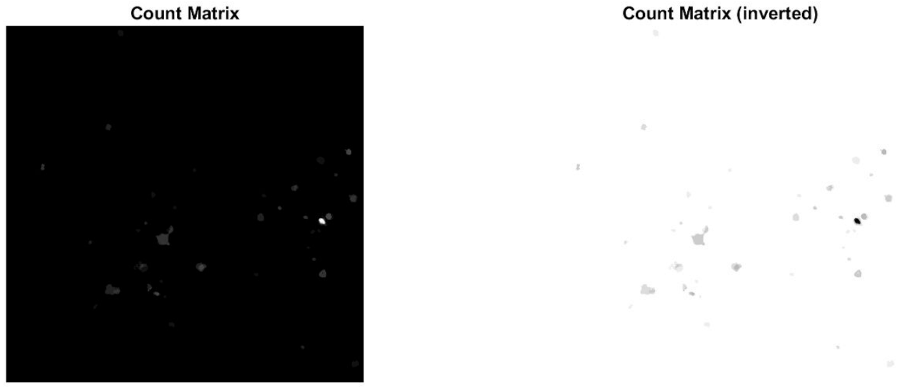
VISUALIZATION OF ENTIRE VIDEO IN ONE FRAME:

We decided to represent the entire video in one frame by showing the number of firings for each neuron. Since the location of a neuron never changes, we can superimpose the values of its various firing counts in order to represent all neurons.

One of the main challenges we faced in this approach was in determining how to spatially represent a neuron in one frame. Since the size of any given firing neuron varies throughout the video, we had to decide which pixels to associate with its firing count. Initially, we attempted to represent a neuron by the maximum size that it took throughout the video; however, we ran into trouble when two neurons overlapped. This event caused our algorithm to assign the same count to both neurons, rather than maintaining separate values. We overcame this issue by assigning different values to each cluster and checking these values in the previous frame to avoid including overlapping neurons.

The figure below represents the location of firing neurons with their firing count across all 500 frames.





CONTRIBUTIONS:

As a team, most of our work was done together. When designing the algorithms, we worked together to write pseudocode and implement our functions in MATLAB.

DETECTION OF RELATED NEURON FIRINGS:

As discussed in “How to see a Memory”, memories that are temporally close together can often be linked. Thus, we believe that a way to detect related neuron firings would be to check the difference (in frames) between two neuron firings and classify the two as related if their temporal difference falls below some predefined threshold.

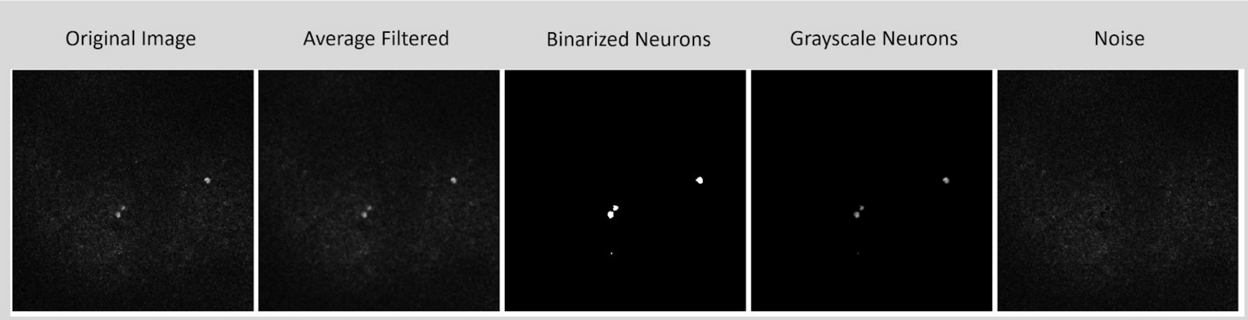
CITATIONS:

Okada, M., Ishikawa, T., & Ikegaya, Y. (2016). A Computationally Efficient Filter for Reducing Shot Noise in Low S/N Data. *PloS one*, *11*(6), e0157595. doi:10.1371/journal.pone.0157595

Shen, H. (2018). How to see a memory. *Nature*, *553*(7687), 146–148. doi: 10.1038/d41586-018-00107-4

APPENDIX A: LINK TO DENOISING VIDEO

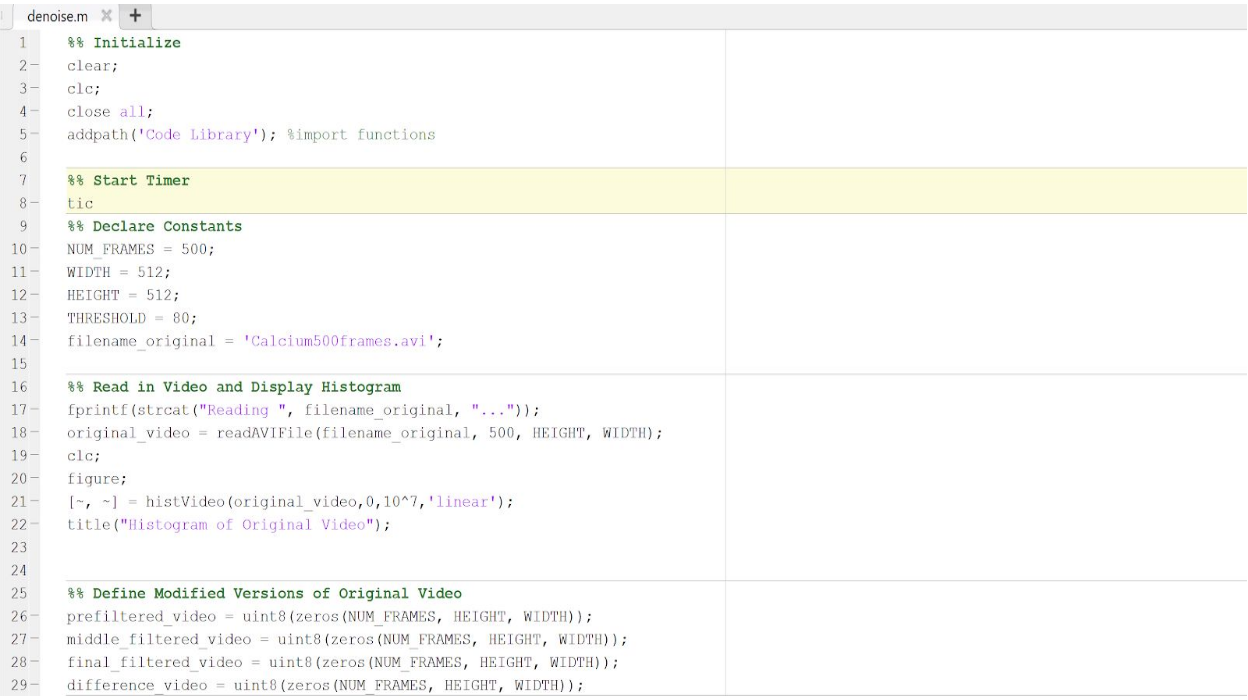
http://bit.ly/neuronDenoisingDIP



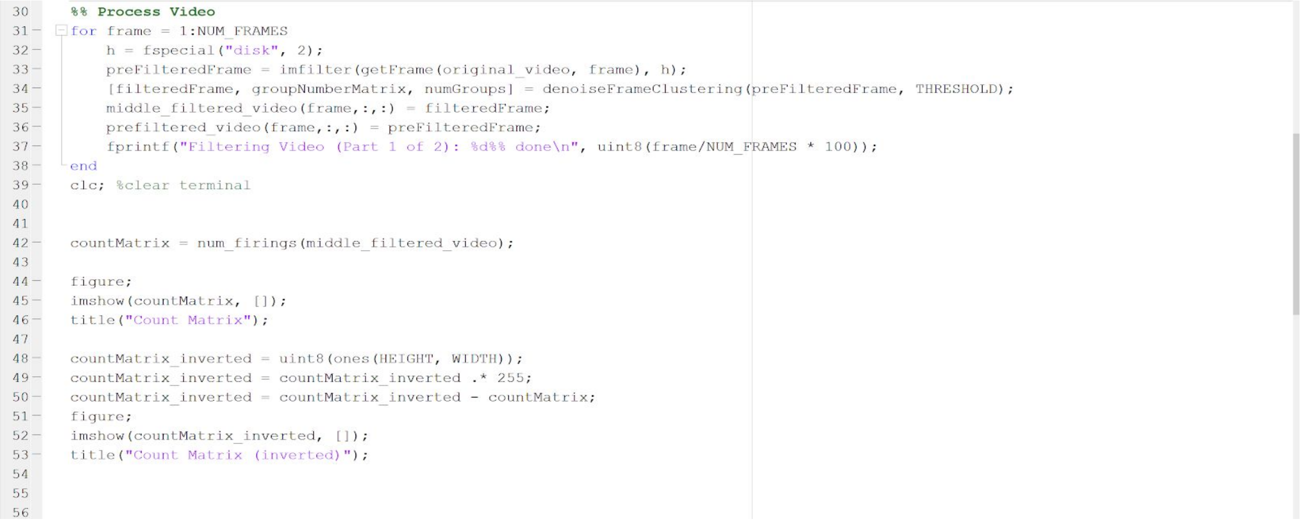
Note: since UVA doesn't include Google Photos with their accounts, be sure that you are logged into a different Google account (or no account at all) when clicking the link.

APPENDIX B: CODE

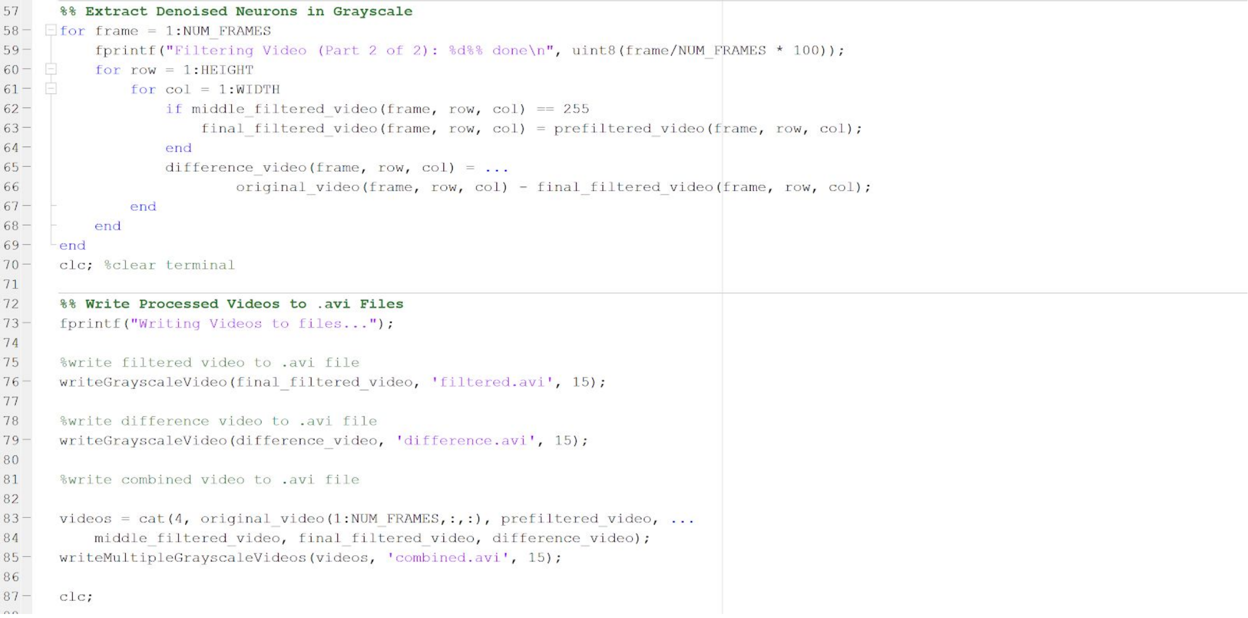
Denoise.m



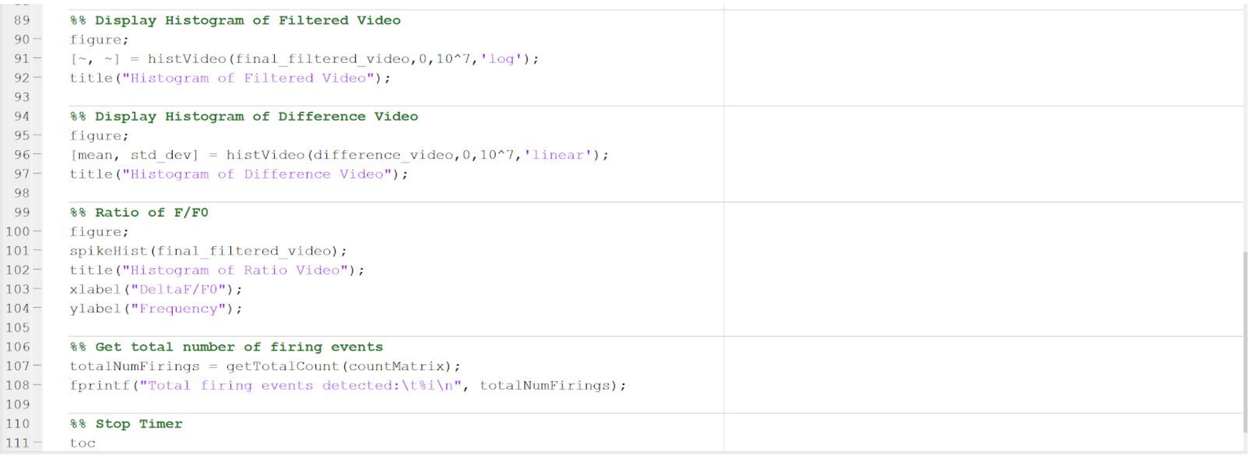
```
%% Initialize
clear;
clc;
close all;
addpath('Code Library'); %import functions

%% Start Timer
tic
%% Declare Constants
NUM_FRAMES = 500;
WIDTH = 512;
HEIGHT = 512;
THRESHOLD = 80;
filename_original = 'Calcium500frames.avi';

%% Read in Video and Display Histogram
fprintf(strcat("Reading ", filename_original, "..."));
original_video = readAVIFile(filename_original, 500, HEIGHT, WIDTH);
clc;
figure;
[~, ~] = histVideo(original_video,0,10^7,'linear');
title("Histogram of Original Video");


%% Define Modified Versions of Original Video
prefiltered_video = uint8(zeros(NUM_FRAMES, HEIGHT, WIDTH));
middle_filtered_video = uint8(zeros(NUM_FRAMES, HEIGHT, WIDTH));
final_filtered_video = uint8(zeros(NUM_FRAMES, HEIGHT, WIDTH));
difference_video = uint8(zeros(NUM_FRAMES, HEIGHT, WIDTH));
```



```
%% Process Video
for frame = 1:NUM_FRAMES
    h = fspecial("disk", 2);
    preFilteredFrame = imfilter(getFrame(original_video, frame), h);
    [filteredFrame, groupNumberMatrix, numGroups] = denoiseFrameClustering(preFilteredFrame, THRESHOLD);
    middle_filtered_video(frame,:,:) = filteredFrame;
    prefiltered_video(frame,:,:) = preFilteredFrame;
    fprintf("Filtering Video (Part 1 of 2): %d%% done\n", uint8(frame/NUM_FRAMES * 100));
end
clc; %clear terminal


countMatrix = num_firings(middle_filtered_video);

figure;
imshow(countMatrix, []);
title("Count Matrix");

countMatrix_inverted = uint8(ones(HEIGHT, WIDTH));
countMatrix_inverted = countMatrix_inverted .* 255;
countMatrix_inverted = countMatrix_inverted - countMatrix;
figure;
imshow(countMatrix_inverted, []);
title("Count Matrix (inverted)");



```

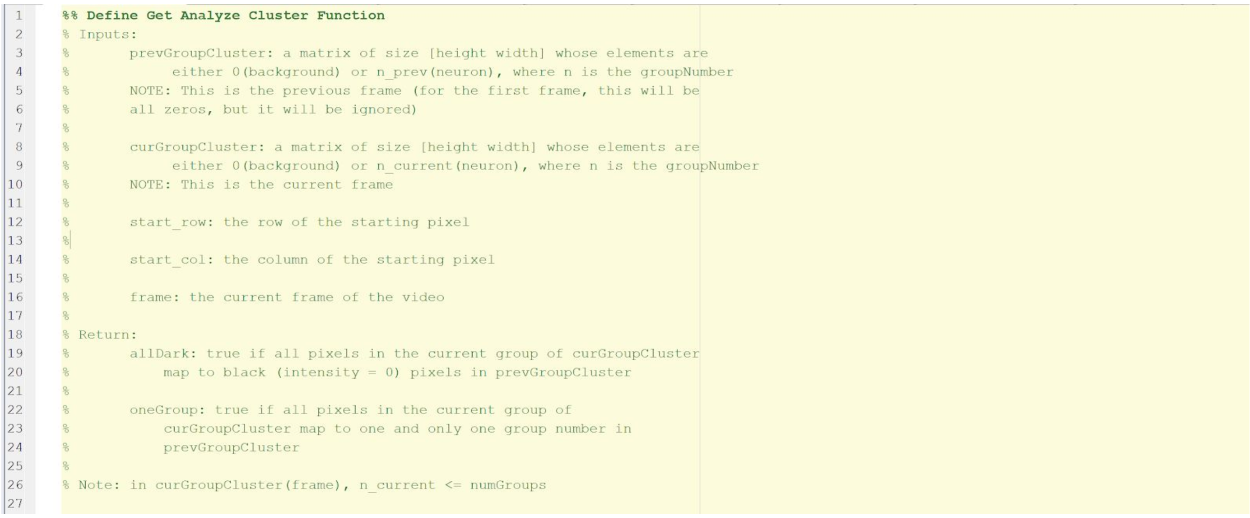


```
%% Extract Denoised Neurons in Grayscale
for frame = 1:NUM_FRAMES
    fprintf("Filtering Video (Part 2 of 2): %d%% done\n", uint8(frame/NUM_FRAMES * 100));
    for row = 1:HEIGHT
        for col = 1:WIDTH
            if middle_filtered_video(frame, row, col) == 255
                final_filtered_video(frame, row, col) = prefiltered_video(frame, row, col);
            end
            difference_video(frame, row, col) = ...
                    original_video(frame, row, col) - final_filtered_video(frame, row, col);
        end
    end
end
clc; %clear terminal

%% Write Processed Videos to .avi Files
fprintf("Writing Videos to files...");

%write filtered video to .avi file
writeGrayscaleVideo(final_filtered_video, 'filtered.avi', 15);

%write difference video to .avi file
writeGrayscaleVideo(difference_video, 'difference.avi', 15);

%write combined video to .avi file

videos = cat(4, original_video(1:NUM_FRAMES,:,:), prefiltered_video, ...
    middle_filtered_video, final_filtered_video, difference_video);
writeMultipleGrayscaleVideos(videos, 'combined.avi', 15);

clc;
```

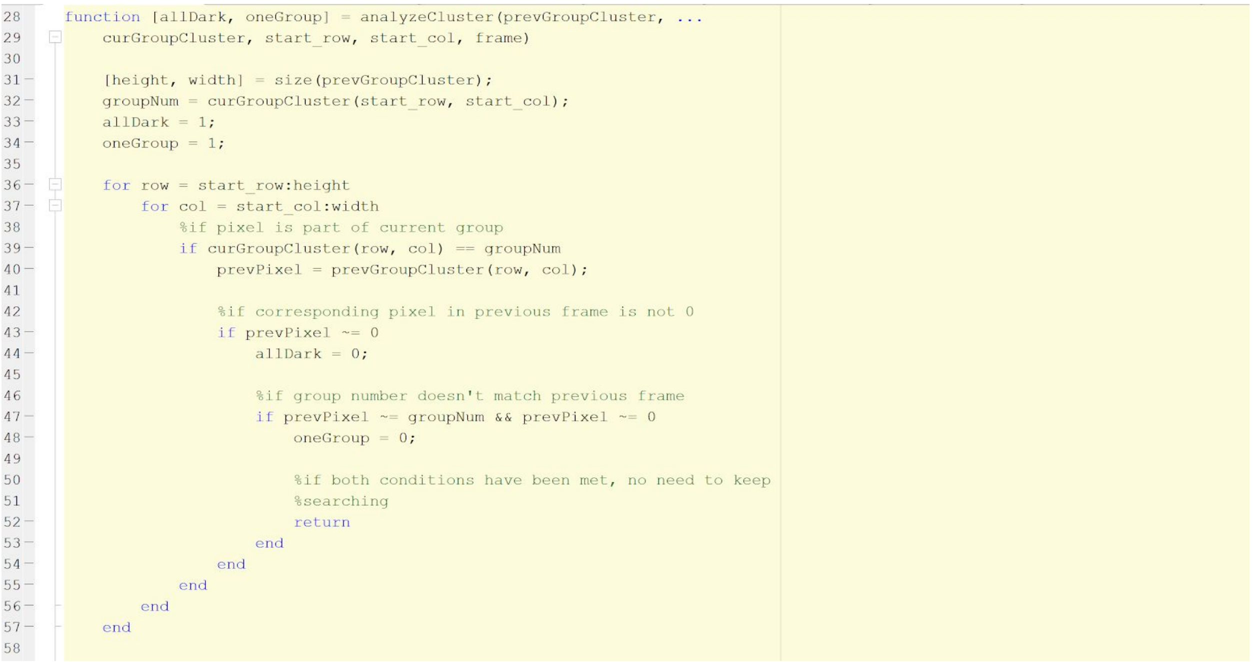


```
%% Display Histogram of Filtered Video
figure;
[~, ~] = histVideo(final_filtered_video,0,10^7,'log');
title("Histogram of Filtered Video");

%% Display Histogram of Difference Video
figure;
[mean, std_dev] = histVideo(difference_video,0,10^7,'linear');
title("Histogram of Difference Video");

%% Ratio of F/F0
figure;
spikeHist(final_filtered_video);
title("Histogram of Ratio Video");
xlabel("DeltaF/F0");
ylabel("Frequency");

%% Get total number of firing events
totalNumFirings = getTotalCount(countMatrix);
fprintf("Total firing events detected:\t%i\n", totalNumFirings);

%% Stop Timer
toc
```

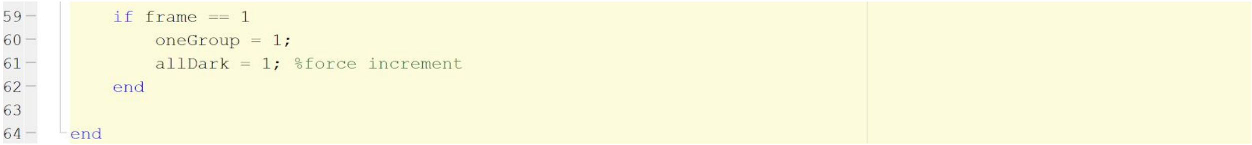
analyzeCluster.m



```
%% Define Get Analyze Cluster Function
% Inputs:
%       prevGroupCluster: a matrix of size [height width] whose elements are
%            either 0(background) or n_prev(neuron), where n is the groupNumber
%       NOTE: This is the previous frame (for the first frame, this will be
%       all zeros, but it will be ignored)
%
%       curGroupCluster: a matrix of size [height width] whose elements are
%            either 0(background) or n_current(neuron), where n is the groupNumber
%       NOTE: This is the current frame
%
%       start_row: the row of the starting pixel
%
%       start_col: the column of the starting pixel
%
%       frame: the current frame of the video
%
% Return:
%       allDark: true if all pixels in the current group of curGroupCluster
%           map to black (intensity = 0) pixels in prevGroupCluster
%
%       oneGroup: true if all pixels in the current group of
%           curGroupCluster map to one and only one group number in
%           prevGroupCluster
%
% Note: in curGroupCluster(frame), n_current <= numGroups

```

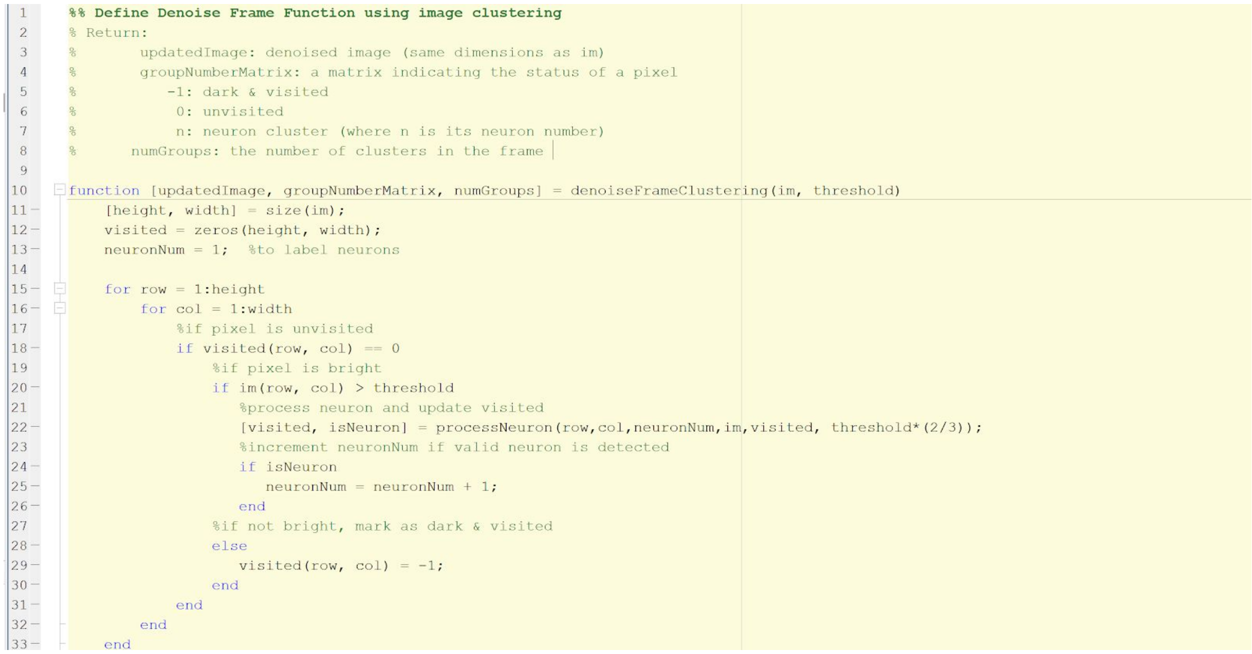


```
function [allDark, oneGroup] = analyzeCluster(prevGroupCluster, ...
    curGroupCluster, start_row, start_col, frame)

    [height, width] = size(prevGroupCluster);
    groupNum = curGroupCluster(start_row, start_col);
    allDark = 1;
    oneGroup = 1;

    for row = start_row:height
        for col = start_col:width
            %if pixel is part of current group
            if curGroupCluster(row, col) == groupNum
                prevPixel = prevGroupCluster(row, col);

                %if corresponding pixel in previous frame is not 0
                if prevPixel ~= 0
                    allDark = 0;

                    %if group number doesn't match previous frame
                    if prevPixel ~= groupNum && prevPixel ~= 0
                        oneGroup = 0;

                        %if both conditions have been met, no need to keep
                        %searching
                        return
                    end
                end
            end
        end
    end

```

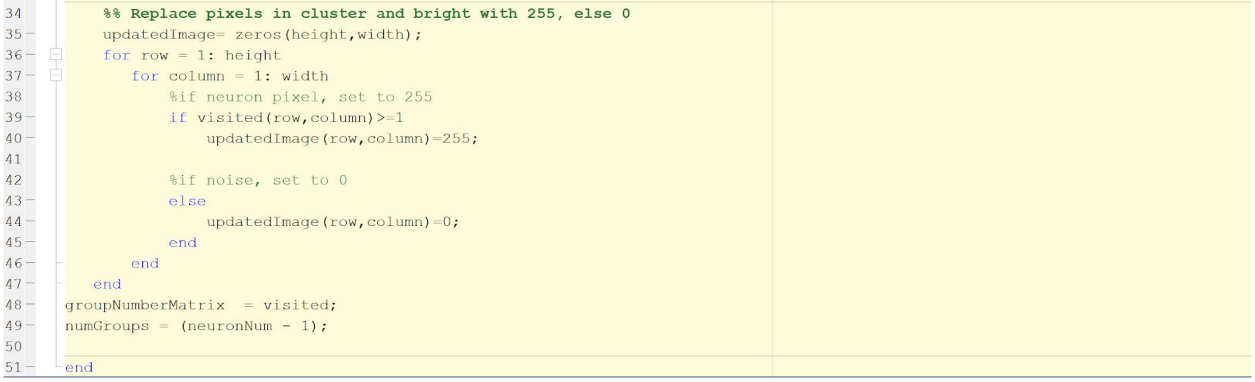


```
    if frame == 1
        oneGroup = 1;
        allDark = 1; %force increment
    end

end
```

denoiseFrameClustering.m

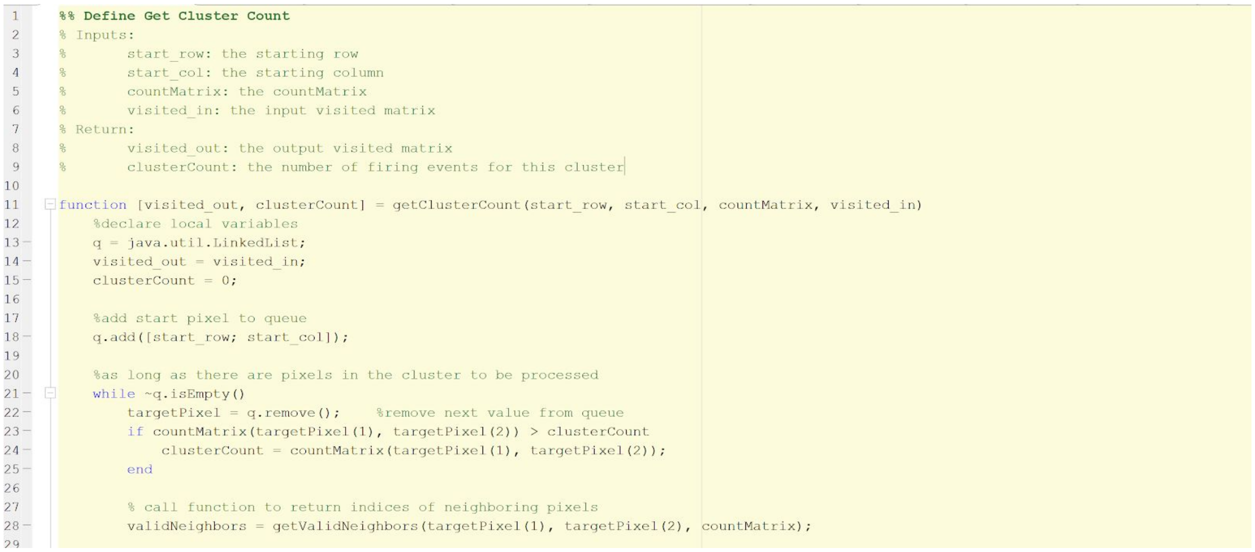


```
%% Define Denoise Frame Function using image clustering
% Return:
%       updatedImage: denoised image (same dimensions as im)
%       groupNumberMatrix: a matrix indicating the status of a pixel
%          -1: dark & visited
%           0: unvisited
%           n: neuron cluster (where n is its neuron number)
%      numGroups: the number of clusters in the frame

function [updatedImage, groupNumberMatrix, numGroups] = denoiseFrameClustering(im, threshold)
    [height, width] = size(im);
    visited = zeros(height, width);
    neuronNum = 1;  %to label neurons

    for row = 1:height
        for col = 1:width
            %if pixel is unvisited
            if visited(row, col) == 0
                %if pixel is bright
                if im(row, col) > threshold
                   %process neuron and update visited
                   [visited, isNeuron] = processNeuron(row,col,neuronNum,im,visited, threshold*(2/3));
                   %increment neuronNum if valid neuron is detected
                   if isNeuron
                      neuronNum = neuronNum + 1;
                   end
                %if not bright, mark as dark & visited
                else
                   visited(row, col) = -1;
                end
            end
        end
    end
```

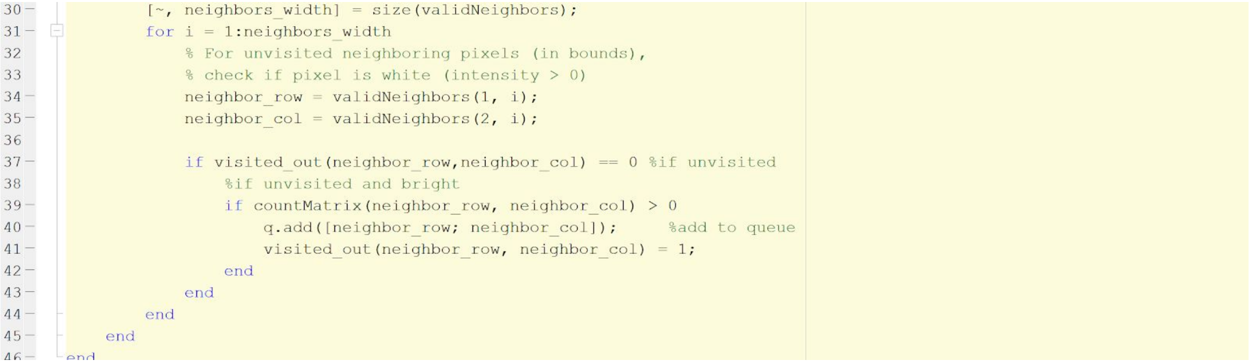


```
    %% Replace pixels in cluster and bright with 255, else 0
    updatedImage= zeros(height,width);
    for row = 1: height
        for column = 1: width
            %if neuron pixel, set to 255
            if visited(row,column)>=1
                updatedImage(row,column)=255;

            %if noise, set to 0
            else
                updatedImage(row,column)=0;
            end
        end
    end
groupNumberMatrix  = visited;
numGroups = (neuronNum - 1);

end
```

getClusterCount.m

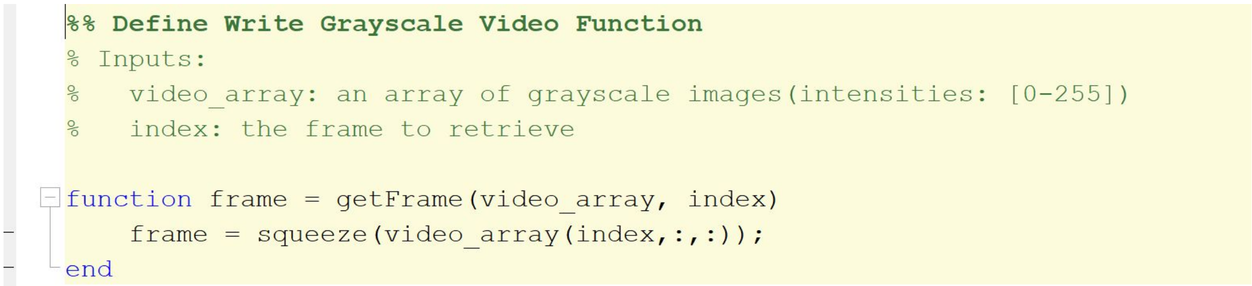


```
%% Define Get Cluster Count
% Inputs:
%       start_row: the starting row
%       start_col: the starting column
%       countMatrix: the countMatrix
%       visited_in: the input visited matrix
% Return:
%       visited_out: the output visited matrix
%       clusterCount: the number of firing events for this cluster

function [visited_out, clusterCount] = getClusterCount(start_row, start_col, countMatrix, visited_in)
    %declare local variables
    q = java.util.LinkedList;
    visited_out = visited_in;
    clusterCount = 0;

    %add start pixel to queue
    q.add([start_row; start_col]);

    %as long as there are pixels in the cluster to be processed
    while ~q.isEmpty()
        targetPixel = q.remove();    %remove next value from queue
        if countMatrix(targetPixel(1), targetPixel(2)) > clusterCount
            clusterCount = countMatrix(targetPixel(1), targetPixel(2));
        end

        % call function to return indices of neighboring pixels
        validNeighbors = getValidNeighbors(targetPixel(1), targetPixel(2), countMatrix);
```



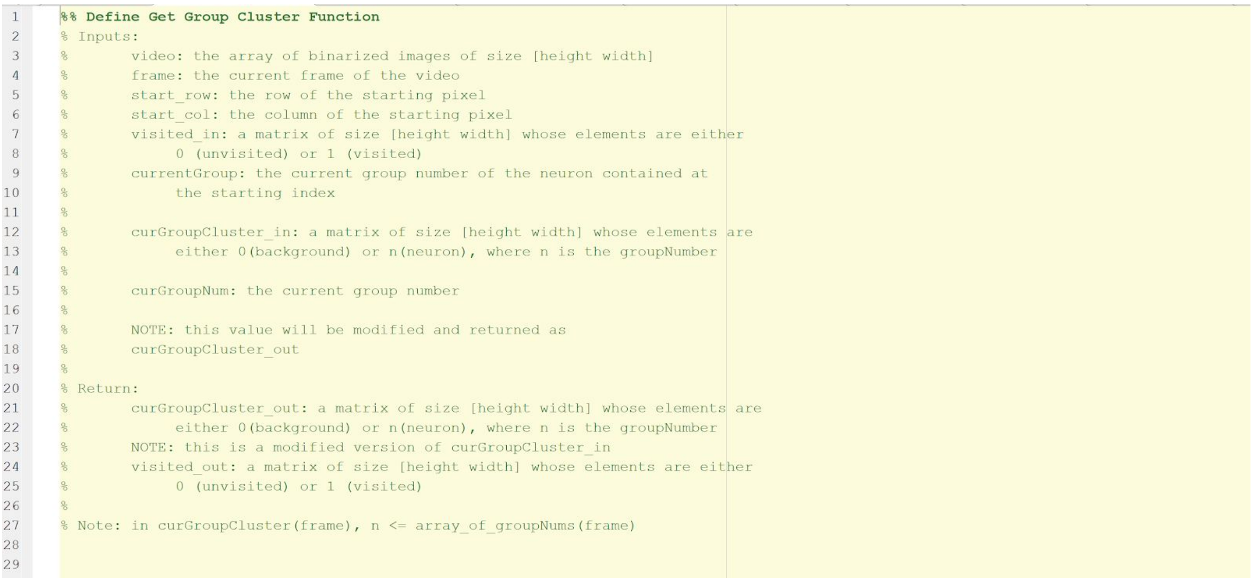
```
        [~, neighbors_width] = size(validNeighbors);
        for i = 1:neighbors_width
            % For unvisited neighboring pixels (in bounds),
            % check if pixel is white (intensity > 0)
            neighbor_row = validNeighbors(1, i);
            neighbor_col = validNeighbors(2, i);

            if visited_out(neighbor_row,neighbor_col) == 0 %if unvisited
                %if unvisited and bright
                if countMatrix(neighbor_row, neighbor_col) > 0
                    q.add([neighbor_row; neighbor_col]);     %add to queue
                    visited_out(neighbor_row, neighbor_col) = 1;
                end
            end
        end
    end
end
```

getFrame.m

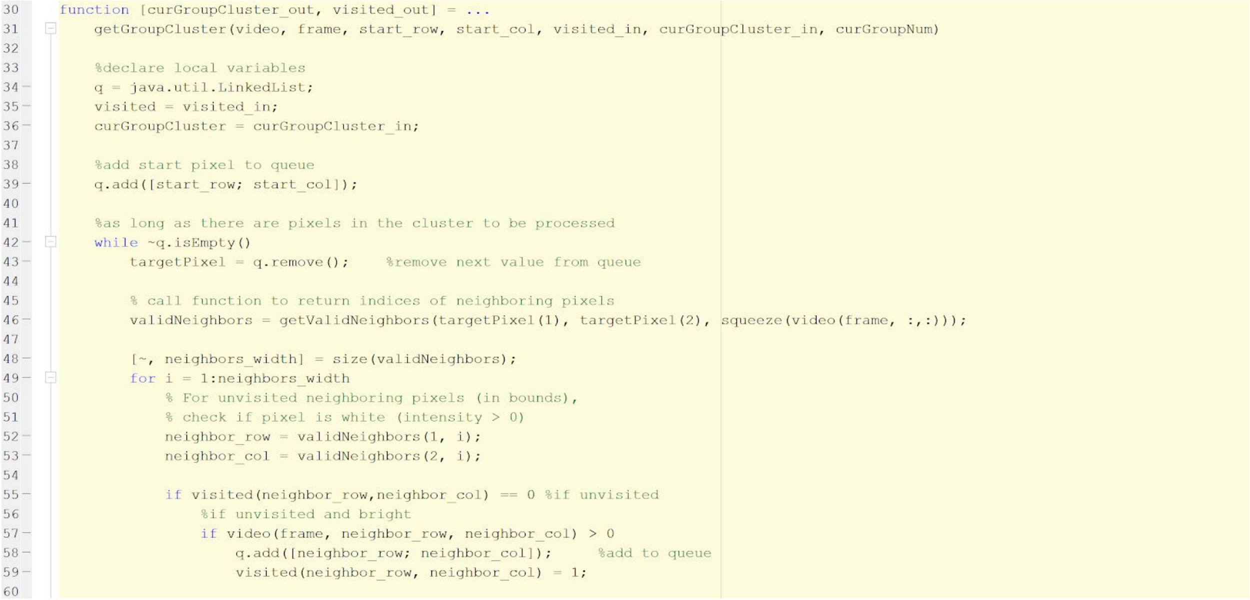


```matlab
%% Define Write Grayscale Video Function
% Inputs:
%   video_array: an array of grayscale images(intensities: [0-255])
%   index: the frame to retrieve

function frame = getFrame(video_array, index)
    frame = squeeze(video_array(index,:,:));
end
```

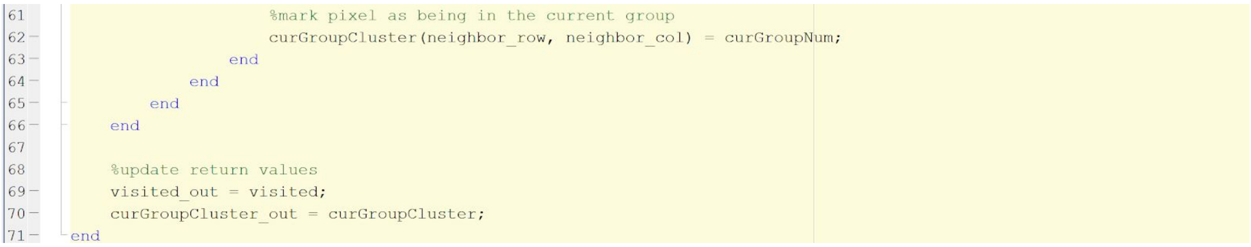
getGroupCluster.m



```matlab
%% Define Get Group Cluster Function
% Inputs:
%       video: the array of binarized images of size [height width]
%       frame: the current frame of the video
%       start_row: the row of the starting pixel
%       start_col: the column of the starting pixel
%       visited_in: a matrix of size [height width] whose elements are either
%            0 (unvisited) or 1 (visited)
%       currentGroup: the current group number of the neuron contained at
%            the starting index
%
%       curGroupCluster_in: a matrix of size [height width] whose elements are
%            either 0(background) or n(neuron), where n is the groupNumber
%
%       curGroupNum: the current group number
%
%       NOTE: this value will be modified and returned as
%       curGroupCluster_out
%
% Return:
%       curGroupCluster_out: a matrix of size [height width] whose elements are
%            either 0(background) or n(neuron), where n is the groupNumber
%       NOTE: this is a modified version of curGroupCluster_in
%       visited_out: a matrix of size [height width] whose elements are either
%            0 (unvisited) or 1 (visited)
%
% Note: in curGroupCluster(frame), n <= array_of_groupNums(frame)


```

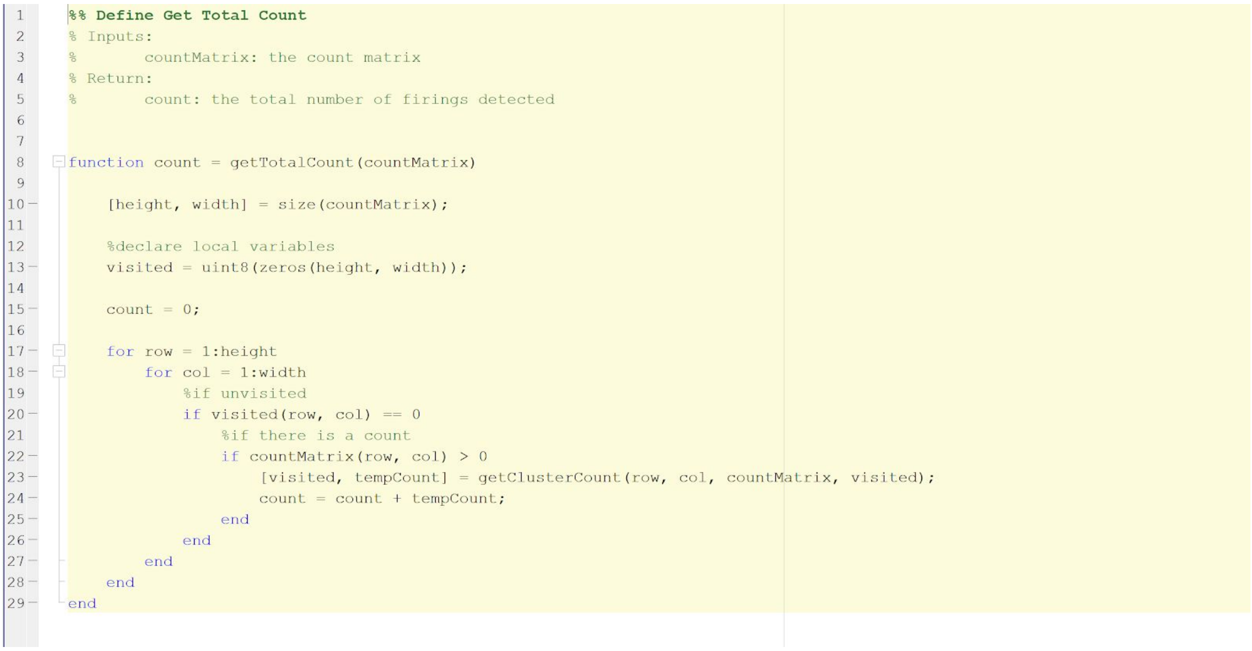


```matlab
function [curGroupCluster_out, visited_out] = ...
    getGroupCluster(video, frame, start_row, start_col, visited_in, curGroupCluster_in, curGroupNum)

    %declare local variables
    q = java.util.LinkedList;
    visited = visited_in;
    curGroupCluster = curGroupCluster_in;

    %add start pixel to queue
    q.add([start_row; start_col]);

    %as long as there are pixels in the cluster to be processed
    while ~q.isEmpty()
        targetPixel = q.remove();    %remove next value from queue

        % call function to return indices of neighboring pixels
        validNeighbors = getValidNeighbors(targetPixel(1), targetPixel(2), squeeze(video(frame, :,:)));

        [~, neighbors_width] = size(validNeighbors);
        for i = 1:neighbors_width
            % For unvisited neighboring pixels (in bounds),
            % check if pixel is white (intensity > 0)
            neighbor_row = validNeighbors(1, i);
            neighbor_col = validNeighbors(2, i);

            if visited(neighbor_row,neighbor_col) == 0 %if unvisited
                %if unvisited and bright
                if video(frame, neighbor_row, neighbor_col) > 0
                    q.add([neighbor_row; neighbor_col]);     %add to queue
                    visited(neighbor_row, neighbor_col) = 1;

```



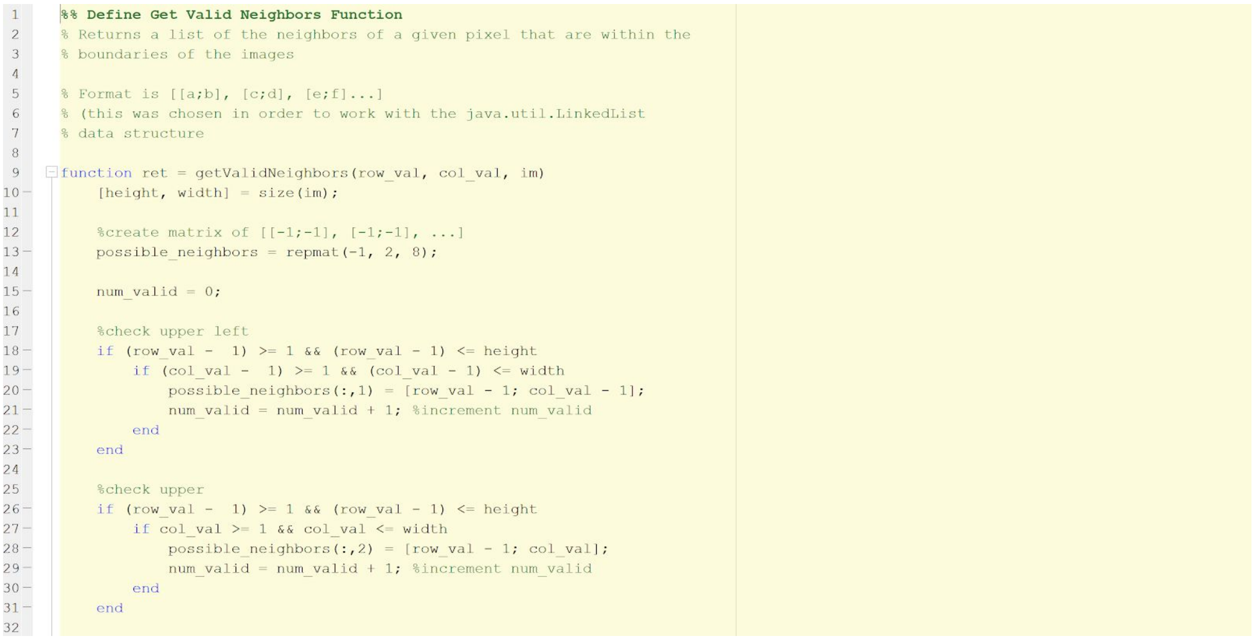
```
                    %mark pixel as being in the current group
                    curGroupCluster(neighbor_row, neighbor_col) = curGroupNum;
                end
            end
        end
    end

    %update return values
    visited_out = visited;
    curGroupCluster_out = curGroupCluster;
end
```

getTotalCount.m

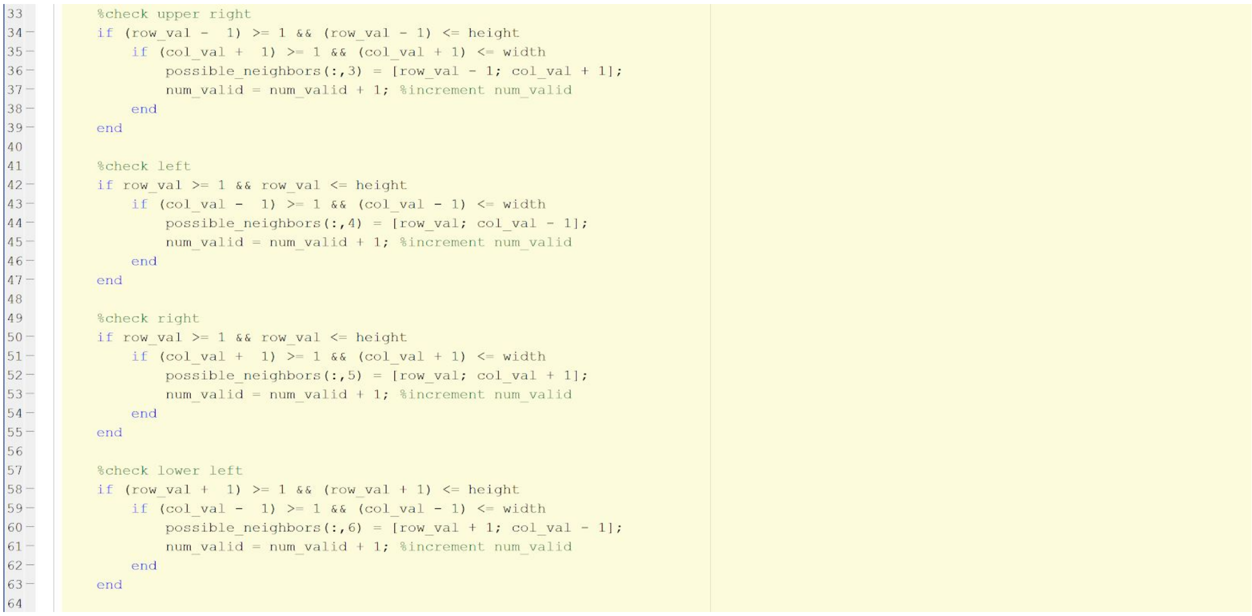


```
%% Define Get Total Count
% Inputs:
%       countMatrix: the count matrix
% Return:
%       count: the total number of firings detected


function count = getTotalCount(countMatrix)

    [height, width] = size(countMatrix);

    %declare local variables
    visited = uint8(zeros(height, width));

    count = 0;

    for row = 1:height
        for col = 1:width
            %if unvisited
            if visited(row, col) == 0
                %if there is a count
                if countMatrix(row, col) > 0
                    [visited, tempCount] = getClusterCount(row, col, countMatrix, visited);
                    count = count + tempCount;
                end
            end
        end
    end
end
```

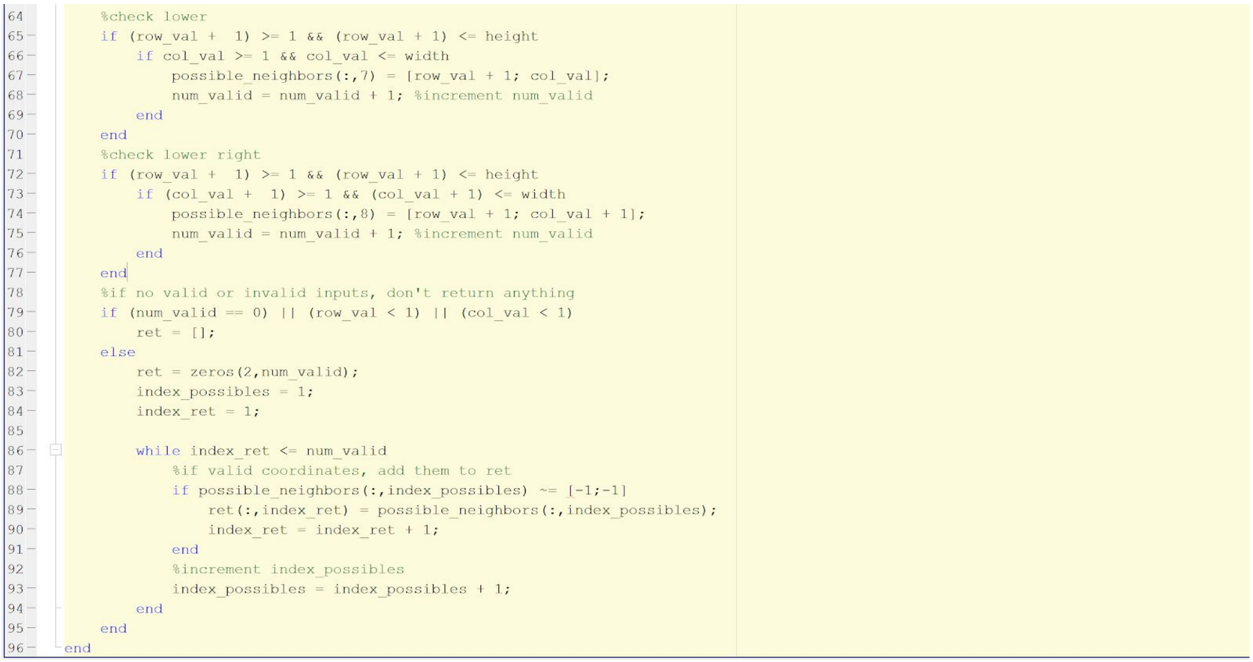
getValidNeighbors.m



```
%% Define Get Valid Neighbors Function
% Returns a list of the neighbors of a given pixel that are within the
% boundaries of the images

% Format is [[a;b], [c;d], [e;f]...]
% (this was chosen in order to work with the java.util.LinkedList
% data structure

function ret = getValidNeighbors(row_val, col_val, im)
    [height, width] = size(im);

    %create matrix of [[-1;-1], [-1;-1], ...]
    possible_neighbors = repmat(-1, 2, 8);

    num_valid = 0;

    %check upper left
    if (row_val -  1) >= 1 && (row_val - 1) <= height
        if (col_val -  1) >= 1 && (col_val - 1) <= width
            possible_neighbors(:,1) = [row_val - 1; col_val - 1];
            num_valid = num_valid + 1; %increment num_valid
        end
    end

    %check upper
    if (row_val -  1) >= 1 && (row_val - 1) <= height
        if col_val >= 1 && col_val <= width
            possible_neighbors(:,2) = [row_val - 1; col_val];
            num_valid = num_valid + 1; %increment num_valid
        end
    end

```

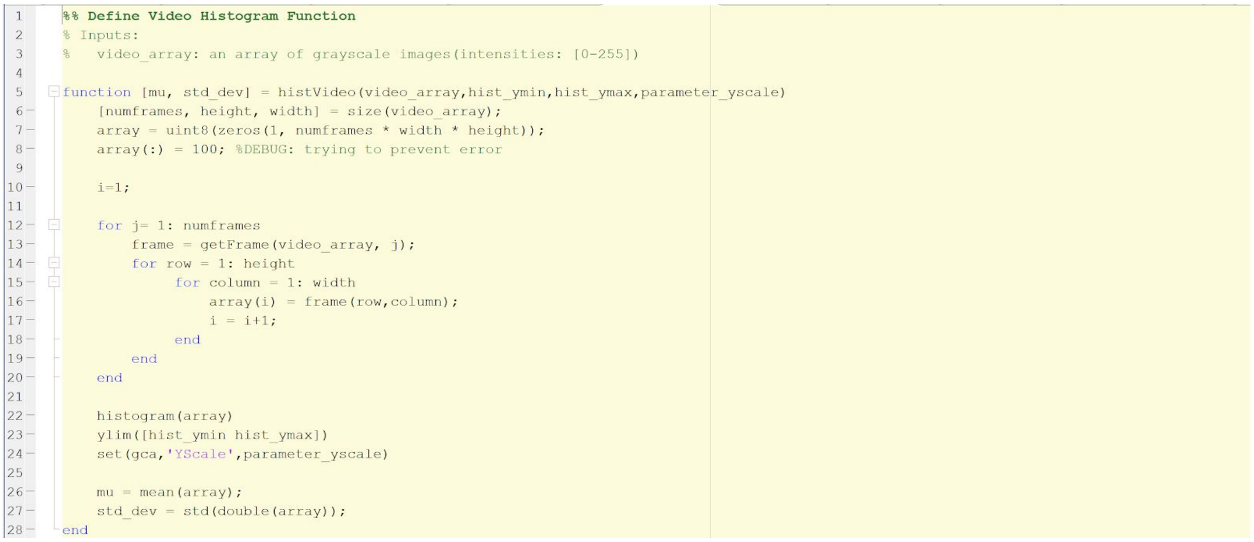


```
    %check upper right
    if (row_val -  1) >= 1 && (row_val - 1) <= height
        if (col_val +  1) >= 1 && (col_val + 1) <= width
            possible_neighbors(:,3) = [row_val - 1; col_val + 1];
            num_valid = num_valid + 1; %increment num_valid
        end
    end

    %check left
    if row_val >= 1 && row_val <= height
        if (col_val -  1) >= 1 && (col_val - 1) <= width
            possible_neighbors(:,4) = [row_val; col_val - 1];
            num_valid = num_valid + 1; %increment num_valid
        end
    end

    %check right
    if row_val >= 1 && row_val <= height
        if (col_val +  1) >= 1 && (col_val + 1) <= width
            possible_neighbors(:,5) = [row_val; col_val + 1];
            num_valid = num_valid + 1; %increment num_valid
        end
    end

    %check lower left
    if (row_val +  1) >= 1 && (row_val + 1) <= height
        if (col_val -  1) >= 1 && (col_val - 1) <= width
            possible_neighbors(:,6) = [row_val + 1; col_val - 1];
            num_valid = num_valid + 1; %increment num_valid
        end
    end

```



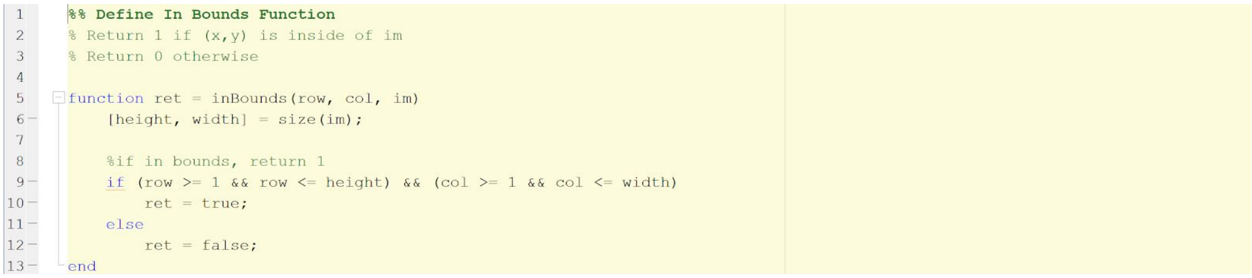
```
    %check lower
    if (row_val +  1) >= 1 && (row_val + 1) <= height
        if col_val >= 1 && col_val <= width
            possible_neighbors(:,7) = [row_val + 1; col_val];
            num_valid = num_valid + 1; %increment num_valid
        end
    end
    %check lower right
    if (row_val +  1) >= 1 && (row_val + 1) <= height
        if (col_val +  1) >= 1 && (col_val + 1) <= width
            possible_neighbors(:,8) = [row_val + 1; col_val + 1];
            num_valid = num_valid + 1; %increment num_valid
        end
    end
    %if no valid or invalid inputs, don't return anything
    if (num_valid == 0) || (row_val < 1) || (col_val < 1)
        ret = [];
    else
        ret = zeros(2,num_valid);
        index_possibles = 1;
        index_ret = 1;

        while index_ret <= num_valid
            %if valid coordinates, add them to ret
            if possible_neighbors(:,index_possibles) ~= [-1;-1]
                ret(:,index_ret) = possible_neighbors(:,index_possibles);
                index_ret = index_ret + 1;
            end
            %increment index_possibles
            index_possibles = index_possibles + 1;
        end
    end
end
```

histVideo.m



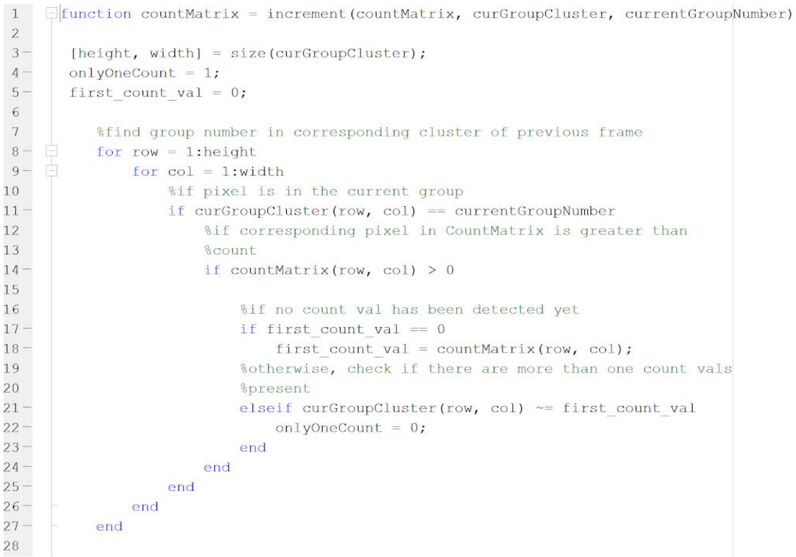
```
%% Define Video Histogram Function
% Inputs:
%   video_array: an array of grayscale images(intensities: [0-255])

function [mu, std_dev] = histVideo(video_array,hist_ymin,hist_ymax,parameter_yscale)
    [numframes, height, width] = size(video_array);
    array = uint8(zeros(1, numframes * width * height));
    array(:) = 100; %DEBUG: trying to prevent error

    i=1;

    for j= 1: numframes
        frame = getFrame(video_array, j);
        for row = 1: height
             for column = 1: width
                 array(i) = frame(row,column);
                 i = i+1;
             end
        end
    end

    histogram(array)
    ylim([hist_ymin hist_ymax])
    set(gca,'YScale',parameter_yscale)

    mu = mean(array);
    std_dev = std(double(array));
end
```

inBounds.m



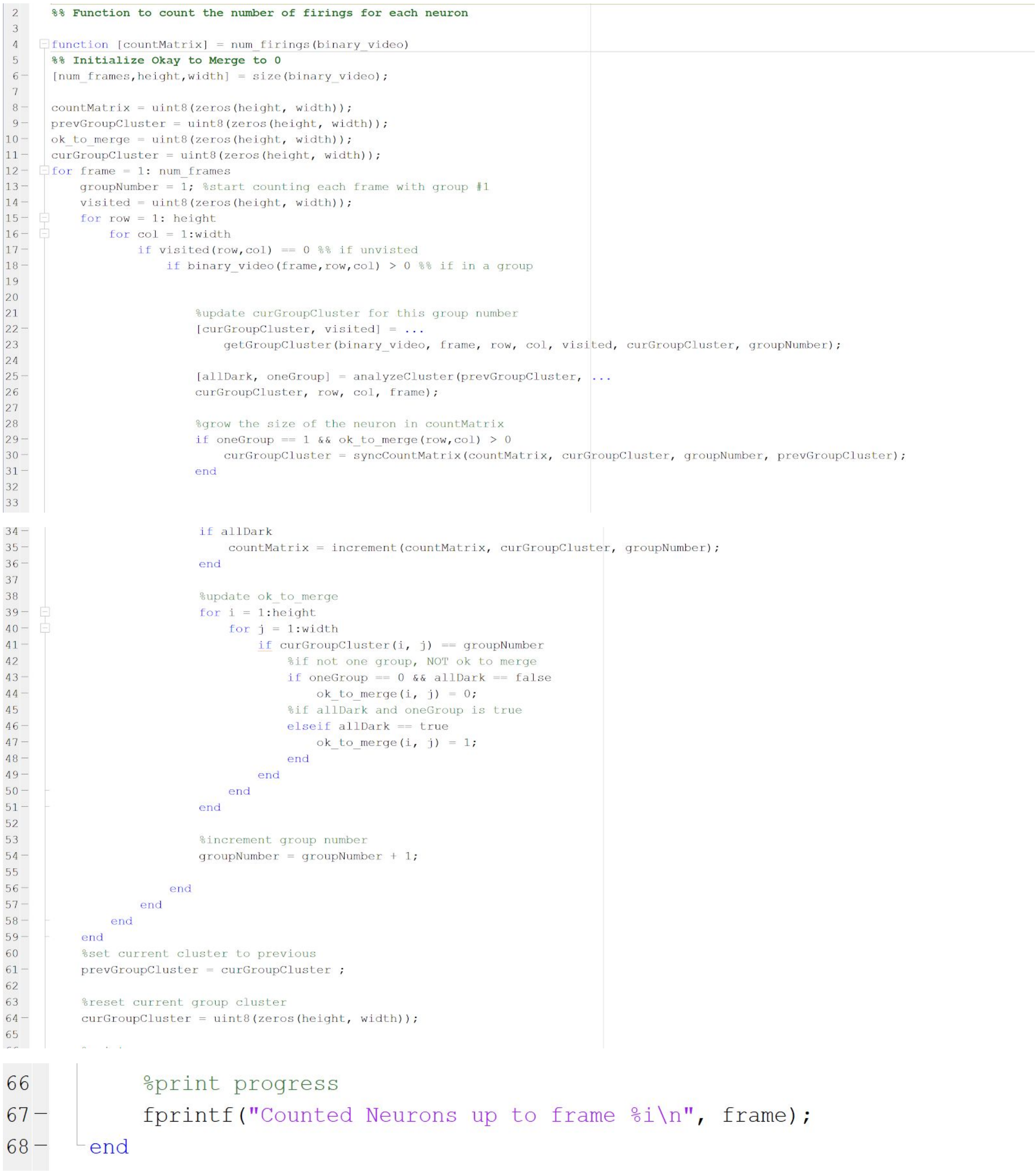
```
%% Define In Bounds Function
% Return 1 if (x,y) is inside of im
% Return 0 otherwise

function ret = inBounds(row, col, im)
    [height, width] = size(im);

    %if in bounds, return 1
    if (row >= 1 && row <= height) && (col >= 1 && col <= width)
        ret = true;
    else
        ret = false;
end
```

Increment.m



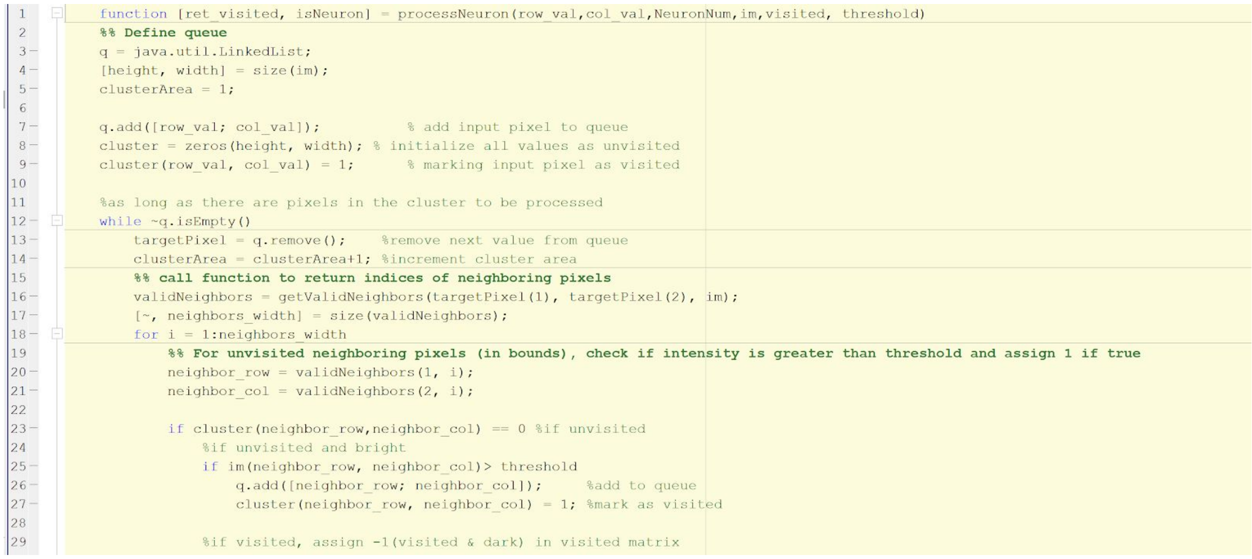
```
function countMatrix = increment(countMatrix, curGroupCluster, currentGroupNumber)

 [height, width] = size(curGroupCluster);
 onlyOneCount = 1;
 first_count_val = 0;

    %find group number in corresponding cluster of previous frame
    for row = 1:height
        for col = 1:width
            %if pixel is in the current group
            if curGroupCluster(row, col) == currentGroupNumber
                %if corresponding pixel in CountMatrix is greater than
                %count
                if countMatrix(row, col) > 0

                    %if no count val has been detected yet
                    if first_count_val == 0
                        first_count_val = countMatrix(row, col);
                    %otherwise, check if there are more than one count vals
                    %present
                    elseif curGroupCluster(row, col) ~= first_count_val
                        onlyOneCount = 0;
                    end
                end
            end
        end
    end

```

Num_firings.m

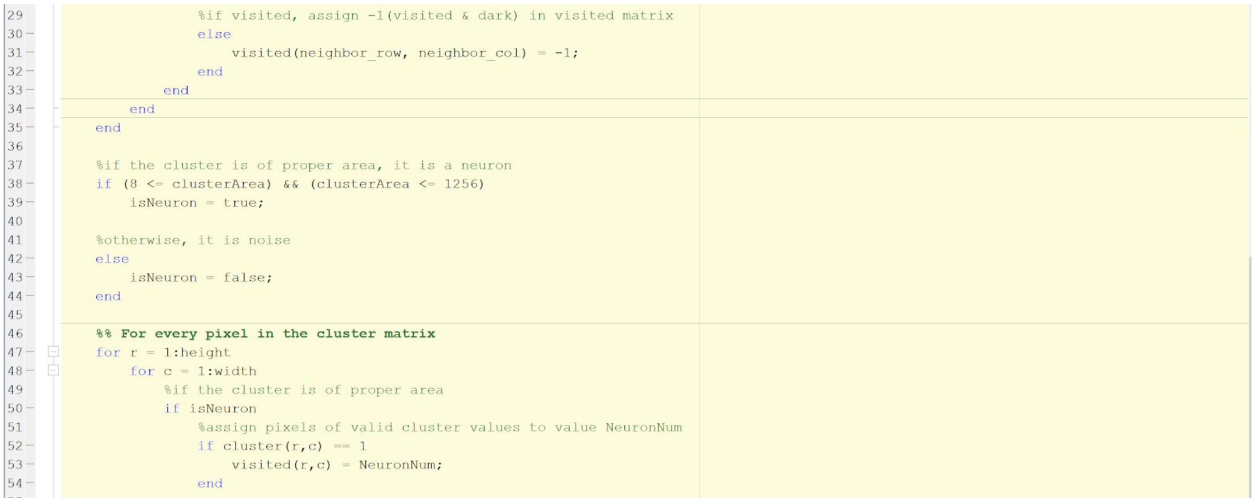


```
%% Function to count the number of firings for each neuron

function [countMatrix] = num_firings(binary_video)
%% Initialize Okay to Merge to 0
[num_frames,height,width] = size(binary_video);

countMatrix = uint8(zeros(height, width));
prevGroupCluster = uint8(zeros(height, width));
ok_to_merge = uint8(zeros(height, width));
curGroupCluster = uint8(zeros(height, width));
for frame = 1: num_frames
    groupNumber = 1; %start counting each frame with group #1
    visited = uint8(zeros(height, width));
    for row = 1: height
        for col = 1:width
            if visited(row,col) == 0 %% if unvisted
                if binary_video(frame,row,col) > 0 %% if in a group


                    %update curGroupCluster for this group number
                    [curGroupCluster, visited] = ...
                        getGroupCluster(binary_video, frame, row, col, visited, curGroupCluster, groupNumber);

                    [allDark, oneGroup] = analyzeCluster(prevGroupCluster, ...
                    curGroupCluster, row, col, frame);

                    %grow the size of the neuron in countMatrix
                    if oneGroup == 1 && ok_to_merge(row,col) > 0
                        curGroupCluster = syncCountMatrix(countMatrix, curGroupCluster, groupNumber, prevGroupCluster);
                    end


                    if allDark
                        countMatrix = increment(countMatrix, curGroupCluster, groupNumber);
                    end

                    %update ok_to_merge
                    for i = 1:height
                        for j = 1:width
                            if curGroupCluster(i, j) == groupNumber
                                %if not one group, NOT ok to merge
                                if oneGroup == 0 && allDark == false
                                    ok_to_merge(i, j) = 0;
                                %if allDark and oneGroup is true
                                elseif allDark == true
                                    ok_to_merge(i, j) = 1;
                                end
                            end
                        end
                    end

                    %increment group number
                    groupNumber = groupNumber + 1;

                end
            end
        end
    end
    %set current cluster to previous
    prevGroupCluster = curGroupCluster ;

    %reset current group cluster
    curGroupCluster = uint8(zeros(height, width));

    %print progress
    fprintf("Counted Neurons up to frame %i\n", frame);
end
```

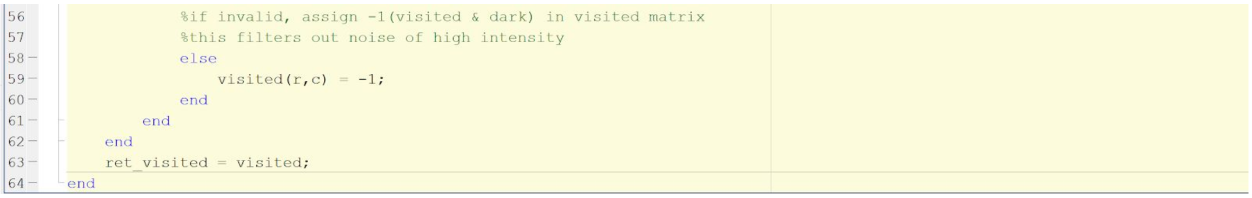
processNeuron.m



```
    function [ret_visited, isNeuron] = processNeuron(row_val,col_val,NeuronNum,im,visited, threshold)
    %% Define queue
    q = java.util.LinkedList;
    [height, width] = size(im);
    clusterArea = 1;

    q.add([row_val; col_val]);           % add input pixel to queue
    cluster = zeros(height, width); % initialize all values as unvisited
    cluster(row_val, col_val) = 1;       % marking input pixel as visited

    %as long as there are pixels in the cluster to be processed
    while ~q.isEmpty()
        targetPixel = q.remove();    %remove next value from queue
        clusterArea = clusterArea+1; %increment cluster area
        %% call function to return indices of neighboring pixels
        validNeighbors = getValidNeighbors(targetPixel(1), targetPixel(2), im);
        [~, neighbors_width] = size(validNeighbors);
        for i = 1:neighbors_width
            %% For unvisited neighboring pixels (in bounds), check if intensity is greater than threshold and assign 1 if true
            neighbor_row = validNeighbors(1, i);
            neighbor_col = validNeighbors(2, i);

            if cluster(neighbor_row,neighbor_col) == 0 %if unvisited
                %if unvisited and bright
                if im(neighbor_row, neighbor_col)> threshold
                    q.add([neighbor_row; neighbor_col]);     %add to queue
                    cluster(neighbor_row, neighbor_col) = 1; %mark as visited

                %if visited, assign -1(visited & dark) in visited matrix
```

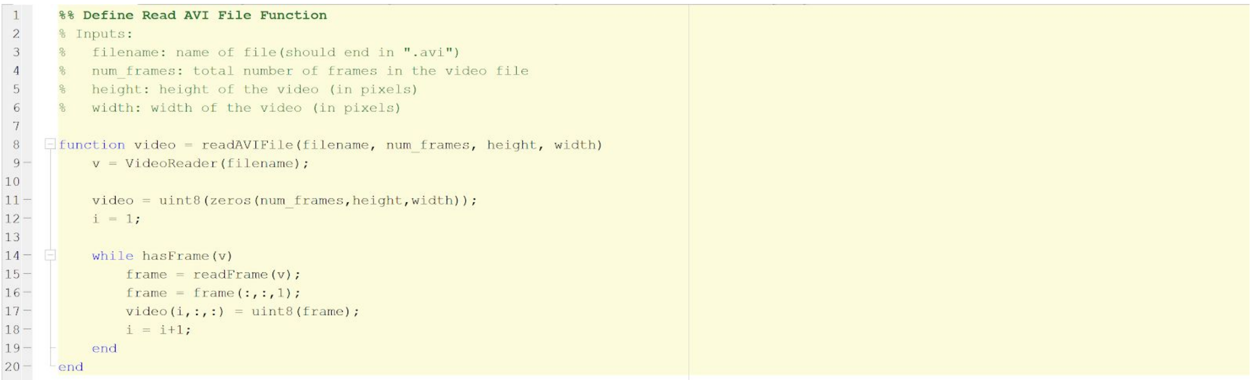


```
                %if visited, assign -1(visited & dark) in visited matrix
                else
                    visited(neighbor_row, neighbor_col) = -1;
                end
            end
        end
    end

    %if the cluster is of proper area, it is a neuron
    if (8 <= clusterArea) && (clusterArea <= 1256)
        isNeuron = true;

    %otherwise, it is noise
    else
        isNeuron = false;
    end

    %% For every pixel in the cluster matrix
    for r = 1:height
        for c = 1:width
            %if the cluster is of proper area
            if isNeuron
                %assign pixels of valid cluster values to value NeuronNum
                if cluster(r,c) == 1
                    visited(r,c) = NeuronNum;
                end
```



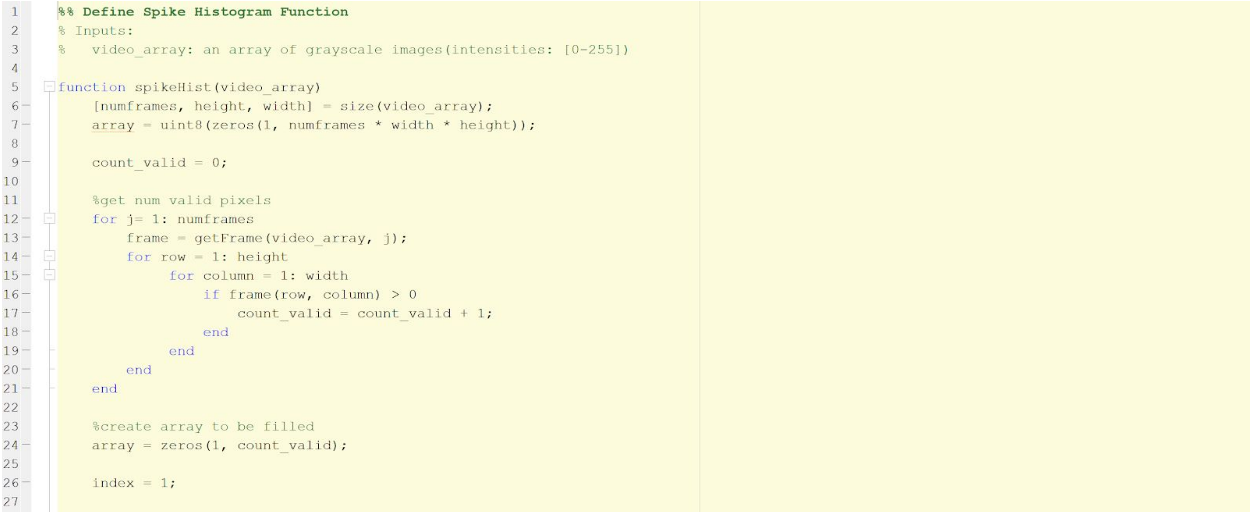
```
            %if invalid, assign -1(visited & dark) in visited matrix
            %this filters out noise of high intensity
            else
                visited(r,c) = -1;
            end
        end
    end
    ret_visited = visited;
end
```

readAVIFile.m

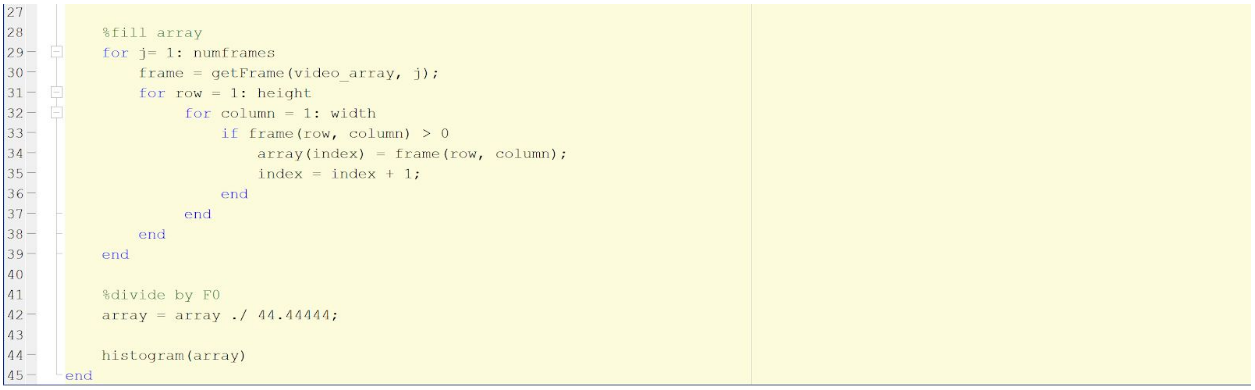


```matlab
%% Define Read AVI File Function
% Inputs:
%   filename: name of file(should end in ".avi")
%   num_frames: total number of frames in the video file
%   height: height of the video (in pixels)
%   width: width of the video (in pixels)

function video = readAVIFile(filename, num_frames, height, width)
    v = VideoReader(filename);

    video = uint8(zeros(num_frames,height,width));
    i = 1;

    while hasFrame(v)
        frame = readFrame(v);
        frame = frame(:,:,1);
        video(i,:,:) = uint8(frame);
        i = i+1;
    end
end
```

spikeHist.m

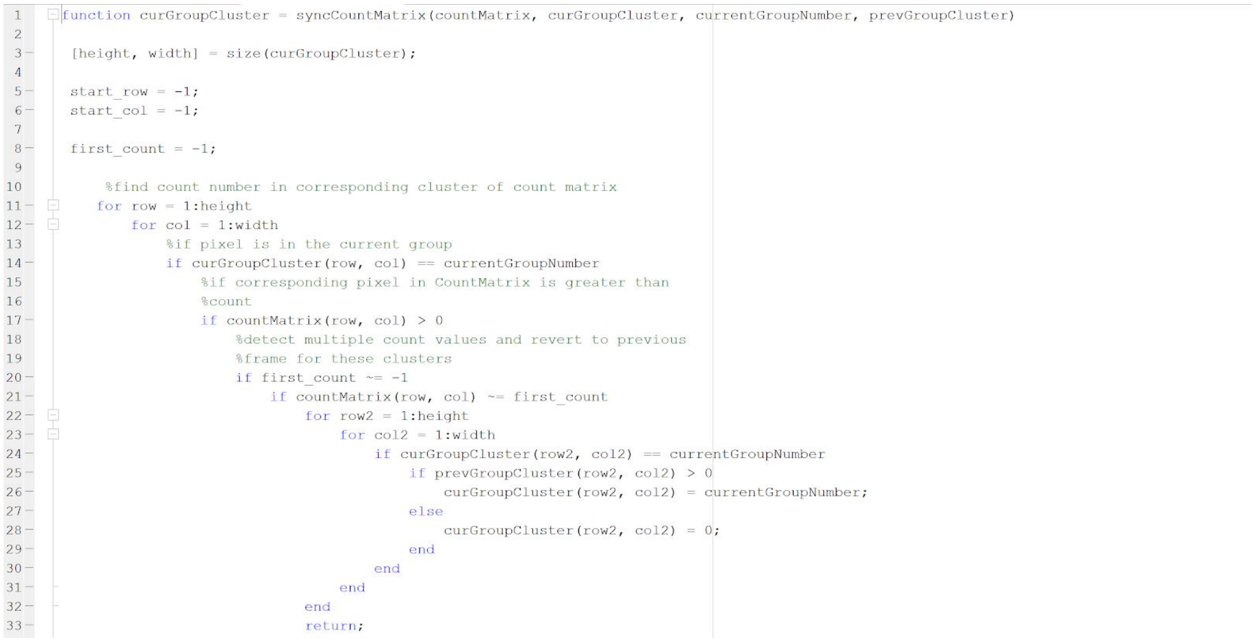


```matlab
%% Define Spike Histogram Function
% Inputs:
%   video_array: an array of grayscale images(intensities: [0-255])

function spikeHist(video_array)
    [numframes, height, width] = size(video_array);
    array = uint8(zeros(1, numframes * width * height));

    count_valid = 0;

    %get num valid pixels
    for j= 1: numframes
        frame = getFrame(video_array, j);
        for row = 1: height
             for column = 1: width
                 if frame(row, column) > 0
                     count_valid = count_valid + 1;
                 end
             end
        end
    end

    %create array to be filled
    array = zeros(1, count_valid);

    index = 1;

```

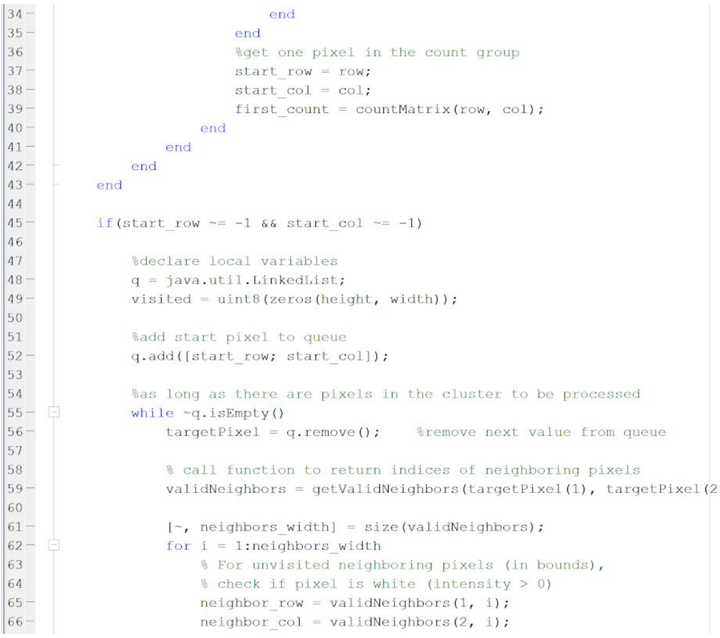


```matlab

    %fill array
    for j= 1: numframes
        frame = getFrame(video_array, j);
        for row = 1: height
             for column = 1: width
                 if frame(row, column) > 0
                     array(index) = frame(row, column);
                     index = index + 1;
                 end
             end
        end
    end

    %divide by F0
    array = array ./ 44.44444;

    histogram(array)
end
```

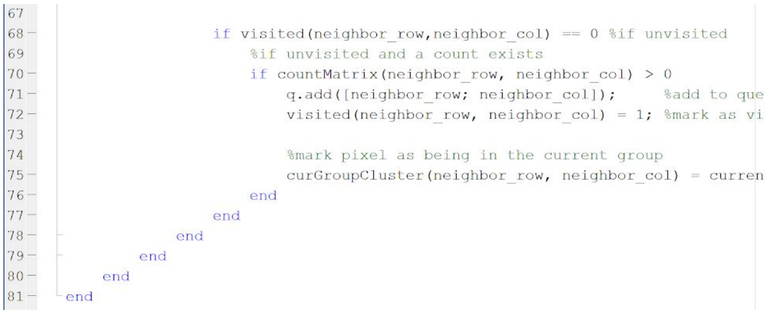
syncCountMatrix.m



```
function curGroupCluster = syncCountMatrix(countMatrix, curGroupCluster, currentGroupNumber, prevGroupCluster)

 [height, width] = size(curGroupCluster);

 start_row = -1;
 start_col = -1;

 first_count = -1;

     %find count number in corresponding cluster of count matrix
    for row = 1:height
        for col = 1:width
            %if pixel is in the current group
            if curGroupCluster(row, col) == currentGroupNumber
                %if corresponding pixel in CountMatrix is greater than
                %count
                if countMatrix(row, col) > 0
                    %detect multiple count values and revert to previous
                    %frame for these clusters
                    if first_count ~= -1
                        if countMatrix(row, col) ~= first_count
                            for row2 = 1:height
                                for col2 = 1:width
                                    if curGroupCluster(row2, col2) == currentGroupNumber
                                        if prevGroupCluster(row2, col2) > 0
                                            curGroupCluster(row2, col2) = currentGroupNumber;
                                        else
                                            curGroupCluster(row2, col2) = 0;
                                        end
                                    end
                                end
                            end
                            return;
```

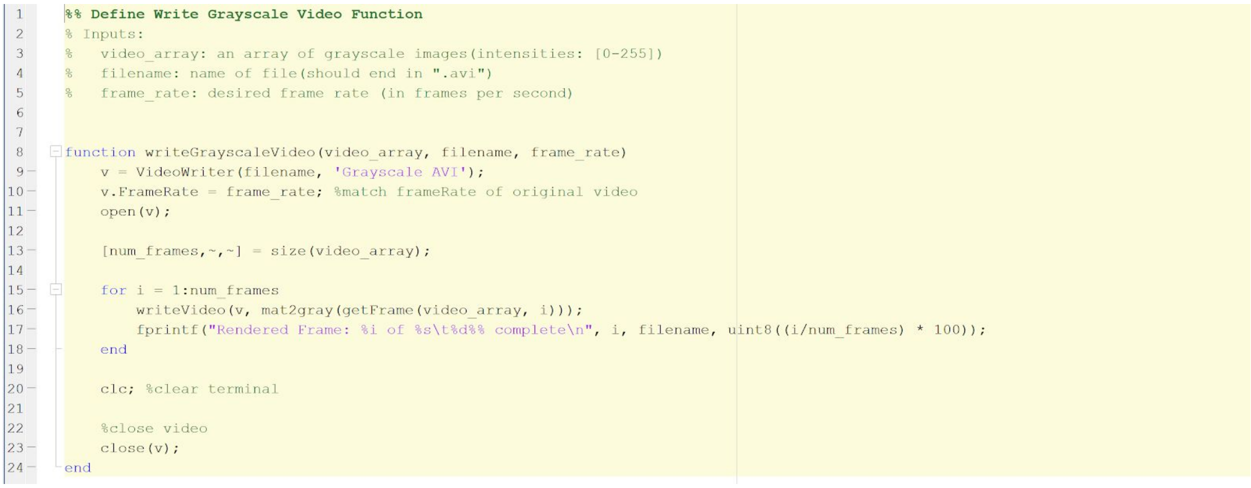


```
                        end
                    end
                    %get one pixel in the count group
                    start_row = row;
                    start_col = col;
                    first_count = countMatrix(row, col);
                end
            end
        end
    end

    if(start_row ~= -1 && start_col ~= -1)

        %declare local variables
        q = java.util.LinkedList;
        visited = uint8(zeros(height, width));

        %add start pixel to queue
        q.add([start_row; start_col]);

        %as long as there are pixels in the cluster to be processed
        while ~q.isEmpty()
            targetPixel = q.remove();    %remove next value from queue

            % call function to return indices of neighboring pixels
            validNeighbors = getValidNeighbors(targetPixel(1), targetPixel(2), countMatrix);

            [~, neighbors_width] = size(validNeighbors);
            for i = 1:neighbors_width
                % For unvisited neighboring pixels (in bounds),
                % check if pixel is white (intensity > 0)
                neighbor_row = validNeighbors(1, i);
                neighbor_col = validNeighbors(2, i);
```



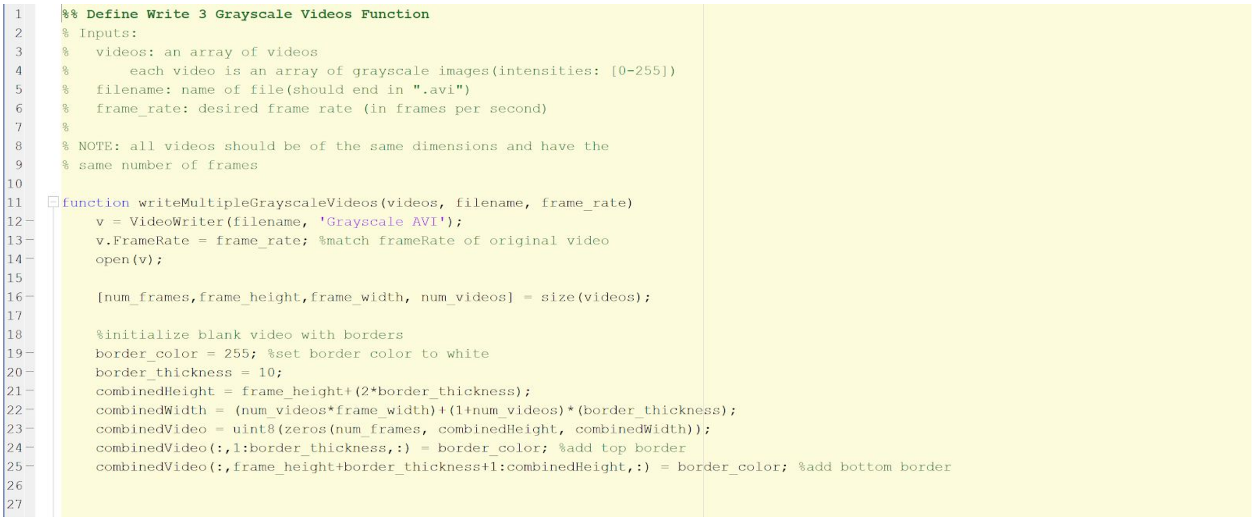
```

                if visited(neighbor_row,neighbor_col) == 0 %if unvisited
                    %if unvisited and a count exists
                    if countMatrix(neighbor_row, neighbor_col) > 0
                        q.add([neighbor_row; neighbor_col]);      %add to queue
                        visited(neighbor_row, neighbor_col) = 1; %mark as visited

                        %mark pixel as being in the current group
                        curGroupCluster(neighbor_row, neighbor_col) = currentGroupNumber;
                    end
                end
            end
        end
    end
end
```

writeGrayscaleVido.m

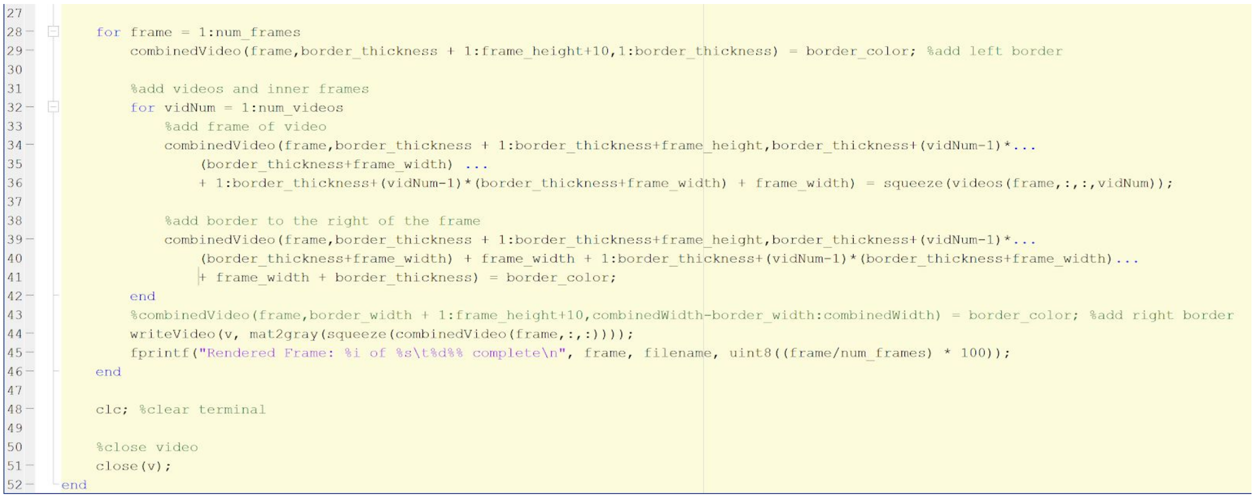


```
%% Define Write Grayscale Video Function
% Inputs:
%   video_array: an array of grayscale images(intensities: [0-255])
%   filename: name of file(should end in ".avi")
%   frame_rate: desired frame rate (in frames per second)


function writeGrayscaleVideo(video_array, filename, frame_rate)
    v = VideoWriter(filename, 'Grayscale AVI');
    v.FrameRate = frame_rate; %match frameRate of original video
    open(v);

    [num_frames,~,~] = size(video_array);

    for i = 1:num_frames
        writeVideo(v, mat2gray(getFrame(video_array, i)));
        fprintf("Rendered Frame: %i of %s\t%d%% complete\n", i, filename, uint8((i/num_frames) * 100));
    end

    clc; %clear terminal

    %close video
    close(v);
end
```

writeMultipleGrayscaleVideos.m



```
%% Define Write 3 Grayscale Videos Function
% Inputs:
%   videos: an array of videos
%       each video is an array of grayscale images(intensities: [0-255])
%   filename: name of file(should end in ".avi")
%   frame_rate: desired frame rate (in frames per second)
%
% NOTE: all videos should be of the same dimensions and have the
% same number of frames

function writeMultipleGrayscaleVideos(videos, filename, frame_rate)
    v = VideoWriter(filename, 'Grayscale AVI');
    v.FrameRate = frame_rate; %match frameRate of original video
    open(v);

    [num_frames,frame_height,frame_width, num_videos] = size(videos);

    %initialize blank video with borders
    border_color = 255; %set border color to white
    border_thickness = 10;
    combinedHeight = frame_height+(2*border_thickness);
    combinedWidth = (num_videos*frame_width)+(1+num_videos)*(border_thickness);
    combinedVideo = uint8(zeros(num_frames, combinedHeight, combinedWidth));
    combinedVideo(:,1:border_thickness,:) = border_color; %add top border
    combinedVideo(:,frame_height+border_thickness+1:combinedHeight,:) = border_color; %add bottom border


```



```

    for frame = 1:num_frames
        combinedVideo(frame,border_thickness + 1:frame_height+10,1:border_thickness) = border_color; %add left border

        %add videos and inner frames
        for vidNum = 1:num_videos
            %add frame of video
            combinedVideo(frame,border_thickness + 1:border_thickness+frame_height,border_thickness+(vidNum-1)*...
                (border_thickness+frame_width) ...
                + 1:border_thickness+(vidNum-1)*(border_thickness+frame_width) + frame_width) = squeeze(videos(frame,:,:,vidNum));

            %add border to the right of the frame
            combinedVideo(frame,border_thickness + 1:border_thickness+frame_height,border_thickness+(vidNum-1)*...
                (border_thickness+frame_width) + frame_width + 1:border_thickness+(vidNum-1)*(border_thickness+frame_width)...
                + frame_width + border_thickness) = border_color;
        end
        %combinedVideo(frame,border_width + 1:frame_height+10,combinedWidth-border_width:combinedWidth) = border_color; %add right border
        writeVideo(v, mat2gray(squeeze(combinedVideo(frame,:,:))));
        fprintf("Rendered Frame: %i of %s\t%d%% complete\n", frame, filename, uint8((frame/num_frames) * 100));
    end

    clc; %clear terminal

    %close video
    close(v);
end
```