Aerial Image Labeling and Semantic Segmentation

Fang, Zi Hang	Liao, Zhi Yuan	Stransky, Cristian
Department of Bioengineering	Department of Electrical and Computer Engineering	Department of Electrical and Computer Engineering
Northeastern University	Northeastern University	Northeastern University
fang.zi@husky.neu.edu	liao.z@husky.neu.edu	stransky.c@husky.neu.edu

I. Introduction:

The standard practice of image classification and semantic segmentation for aerial imagery is through the use of deep neural networks. Semantic segmentation modeling based on convolutional neural networks (CNN) [4][8] at a pixel level may achieve an average pixel accuracy of 96.08% [3][6], but requires significant training resources and is highly time-consuming. Our project attempts to discover a viable alternative to neural networks through the use of approximate nearest neighbor (ANN) and windowed Hough transform.

To discover this viable method, we used MATLAB to create and process our algorithms. We used a database of aerial imagery that contains the classifications for only buildings within an image, so all algorithms were based on accurately classifying buildings from an overhead view. We created a simple 10-layer CNN first to generate results as a baseline to compare with the other methods. We then designed algorithms using approximate nearest neighbor and windowed Hough transform to compete with CNN. All code is of our own design. All results were based on the accuracy and processing time across 20 randomly chosen images from our database of 18,000 image pairs, compared with the expected results from the corresponding ground truth image. It should be noted that CNN will include the processing time for training, and not simply inference, as we want to show the contrast between using methods that require training and methods that don't.

II. Methods:

Convolutional Neural Network (CNN)

For our project, we wanted to create a simple CNN to generate results that can be used as a baseline for the other algorithms. As mentioned previously, using CNNs for semantic segmentation of features within an image, especially the mostly rectangular features of buildings, will produce very satisfactory classification identification with over a 90% accuracy rating, so even using a simple CNN should yield results comparable to our other methods. We did not have to generate classification labels for our aerial imagery, as they were provided with the images as a separate pair (1 training image and ground truth image determining buildings), so our goal was strictly creating a CNN with satisfactory results to prove the hypothesis that CNN is very effective for classification of aerial imagery.

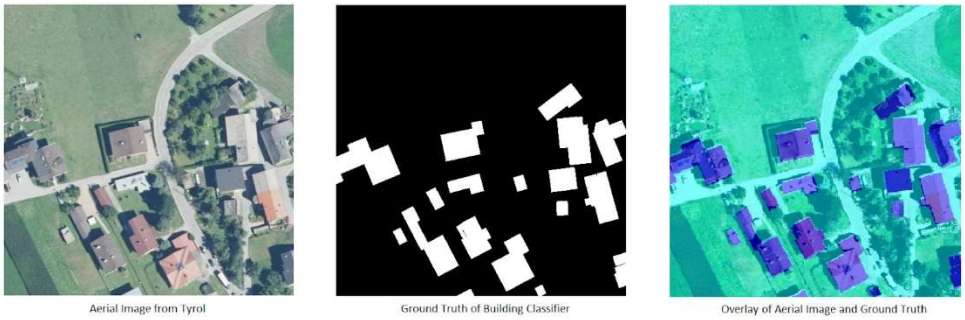

Figure 1. Aerial Image and Ground Truth Training Images

We used Matlab to create a self-made and self-trained 10 layer CNN that produced results within an average accuracy of 87%. Below is the specific make-up of our CNN:

1. **Image Input Layer** (500 x 500 x 3): Converts images fed into the network and applies data normalization.
2. **Convolution Layer** (3 x 3; Filters: 32; Padding: 1): Slides filters across the image and computes the dot product of the input and weights in the filters.
3. **ReLU Layer:** A threshold layer that turns any value less than 0 to 0.
4. **Max Pooling Layer** (2 x 2; Stride 3 x 3): Down-samples the image by dividing the input into rectangular pooling regions based on the maximum of each region.
5. **Convolution Layer** (3 x 3; Filters: 32; Padding: 1)
6. **ReLU Layer**
7. **Transposed Convolution Layer** (4 x 4; Filters: 32): Applies filters across the image, but transposed, which allows for the image to upscale to a larger size.
8. **Convolution Layer** (1 x 1; Filters: 2)
9. **Softmax Layer**: Calculates prob. distrib. for each pixel from a range of 0 to 1.
10. **Classification Layer** (with Class Weights applied): Provides category labels.

    Other hyperparameters:
    Number of filters for each convolution layer (except layer 8): 32
    Mini-batch size for the gradient of the loss function and the weights: 16
    Number of Epochs: 10
    Number of classes: 2 (“buildings” & “terrain”)

It should be noted that layers 1 - 6 are used to apply filters and down-sample our images in the CNN to determine our network, while layers 7 - 10 are used to upscale and apply filters to allow our CNN to apply labels to each pixel within the image. The pixel windows for each layer were chosen based off of optimal CNN settings. Class weights were applied to the last classification layer to allow for much more accurate results:

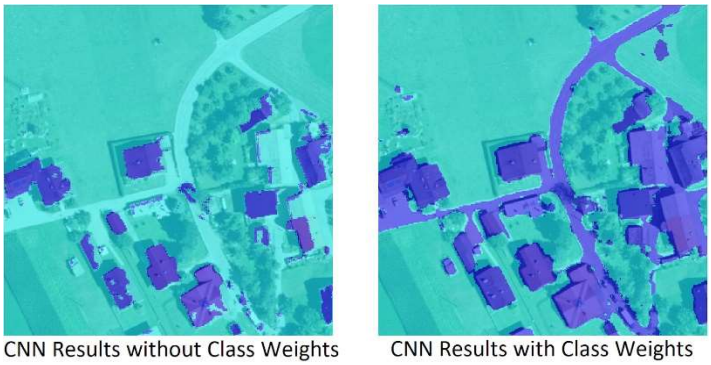

Figure 2. CNN Results with and without Class Weights

The class weights were determined by finding the total difference in the number of pixels between the 2 classes to generate class weights of [buildings: 17.1131, terrain: 1.0621].

For our training, we used 36000 image pairs made of 500 x 500 pixels in color (3 channel RGB) of aerial images and ground truths with object classification (7200 images total) all comprising of images from Tyrol, Austria. Tyrol was chosen because it is a mountainous region with vast green plains that have a distinct contrast to the local buildings, which allowed for our CNN to be more critical of selecting the building classifier because of the greater amount of terrain to buildings which allowed for more accurate results (this is further demonstrated by the fact that our class weights are 17 to 1 for buildings and terrain respectively).

Approximate Nearest Neighbors (ANN)

The heart of the algorithm consists of three main parts. It begins with random assign segments on a training image (B) with segments labeled in a ground truth image (A). For each segment Ai in A, there exists a patch Bi centered at Bi's mean. The features extracted in ANN are the average RGB values of each pixel. The maximum number of patches in either A or B are defined by sqrt(rows*cols). And the number of segments in the training image cannot be less than the number of buildings in the ground truth image. After this step, propagation and search are performed iteratively to improve the initial matches.

In the propagation step, ANN defines a patch distance function D(Ai, Bj) that accounts for adjacent patches. However, the distance function computes one-to-one matching between the structural elements and cannot accurately deal with the overlap of superpixels that requires a one-to-many mapping [9]. Another limitation is that it mixes two different pieces of information: superpixel similarities and the cost of removing or adding superpixels. The distance function D is hence defined as [7]:

$$D(\mathbf{A_i}, \mathbf{B_j}) = \frac{\sum_{i' \in \mathcal{I}_i^A} \sum_{j' \in \mathcal{I}_j^B} w(A_{i'}, B_{j'}) d(F_{i'}^A, F_{j'}^B)}{\sum_{i' \in \mathcal{I}_i^A} \sum_{j' \in \mathcal{I}_j^B} w(A_{i'}, B_{j'})}$$

Where ***Ii*** denotes the set of segments in patch i, ***w*** is a spatial distance between the centers of segments Ai and Bj. The distance ***d*** is the Euclidean distance between features FiA and FjB.

The test candidate for the ANN of Ai is chosen based on the adjacent segment of each Ai′. This candidate is chosen by comparing the angle between Ai′ and Bj' with the most similar symmetric orientation.

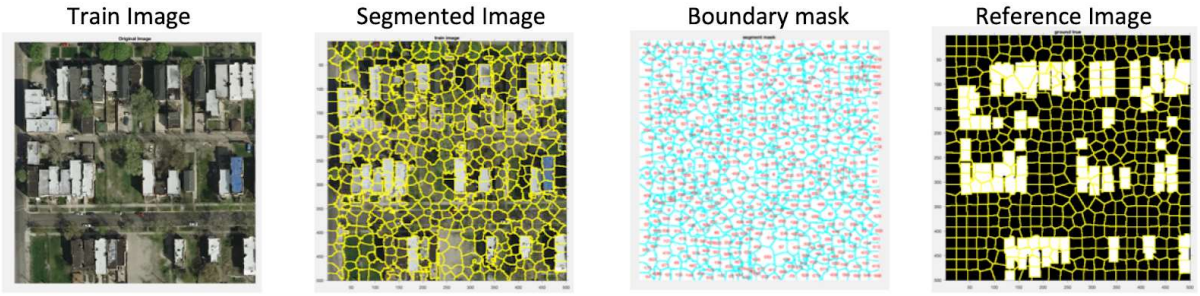

Figure 3. ANN image output

In the final search step, it consists of sampling number of segments based on the image size around Bj. The processing order is from top-to-bottom, and then left-to-right. Due to the irregularity of segments, it samples the first 3 segments in the top-left region in the first interaction. Then repeats the sampling in the lower-right region. This step helps ANN to escape from possible local minima [10].

```
1: For each pixel xi ∈ A do
2:        Random initialization of the map M (xi) = xj ∈ B
3: For each iteration do
4:        For each pixel xi ∈ A do
5:           Propagation from neighbors xi ± [1, 0] and xi ± [0, 1]
6:      Random search around the best match M (xi)
7: Return M
```

Figure 4. ANN pseudo code

Rectangle Detection based on Clustering Results

We also considered a completely unsupervised classification method that performed rectangle detection on each cluster in the image. Since buildings usually appear the shape of rectangles, this additional step was supposed to better outline the region of interests for buildings and thus improve the labeling accuracy.

Hough transform was originally designed for detecting linear structures in images. As suggested by Rosenfeld and Weiss [12], Hough transform can further be used to uniquely determine the existence of convex polygons. The fundamental of Hough transform relies on a very simple representation of straight lines that relates cartesian coordinates to polar coordinates:

$$\rho = xcos\theta + ysin\theta$$

In this equation, ρrepresents the normal distance between the origin and the line, and θrepresents the normal angle of the straight line. When Hough transform is applied on a straight line $\rho_0 = xcos\theta_0 + ysin\theta_0$ in the cartesian domain, we shall expect a local maximum in the Hough space

at exactly (θ_0, ρ_0) simply because this is where the line can most accurately be described. Similarly, if we apply Hough transform to a rectangle centered at the origin of the coordinate system, the output in Hough space should have four peaks, each representing one side of the rectangle.

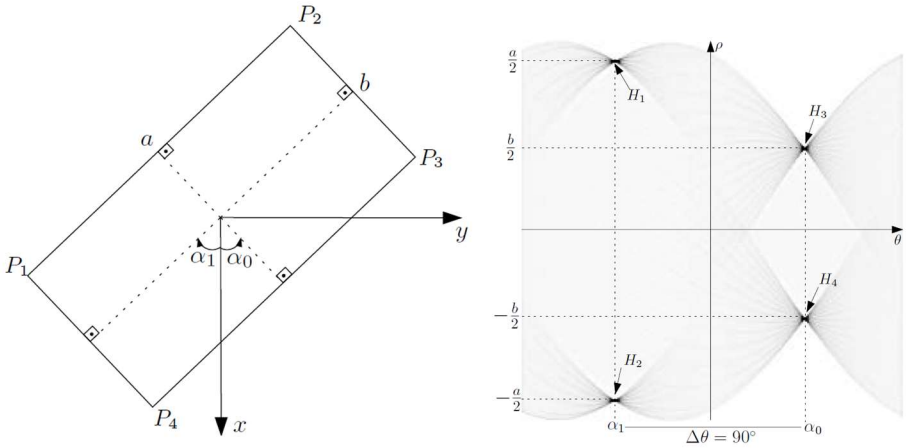

Figure 5. A rectangle centered at the origin can be converted into Hough space, which will be represented by four local maxima [13].

It is worth noting that local maxima in the Hough space can sometimes reflect geometric properties of the object. For rectangles, Schramm et al. suggested three specific relations as: 1) peaks that are vertically aligned are also symmetric with respect to θ axis; 2) vertical distances between each pair of peaks represent the side of the rectangles; 3) the two pairs of peaks are always separated by 90 degrees [13]. In Schramm et al.'s paper, these three relations were used to check whether a given set of local maxima satisfied such conditions, and therefore determine the presence of rectangle in image. For this project, we also considered using these three relations as the guideline for rectangle detection. However, as will be discussed below, under conditions where not all local maxima are clearly present, these geometric properties can also help locating the missing local maxima and reconstruct the rectangles.

For this method, we mainly focused on exploring the potential of rectangle detection based on Hough transform. While different types of unsupervised clustering algorithms would yield varying results and would directly alter the performance, we decided to go with the one that worked best with aerial images.

III. Experimentation

Convolutional Neural Network (CNN)

The CNN's training duration and hyperparameters were mostly determined by the hardware available. We used an Nvidia GTX 1080 (8GB VRAM GDDR5) as our main processor for the training of our CNN, which allowed us to use 32 filters per convolution layer, 16 mini-batch size, and 10 epochs for an average processing time of 30 minutes. In theory, increasing the number of filters and increasing the mini-batch size would lead to more accurate results, but in practice, increasing those sizes stopped the training because of a lack of necessary memory to

process. These parameters were mostly chosen through experimentation by determining which parameters would allow us the most accurate results while not exceeding 8GB of memory.

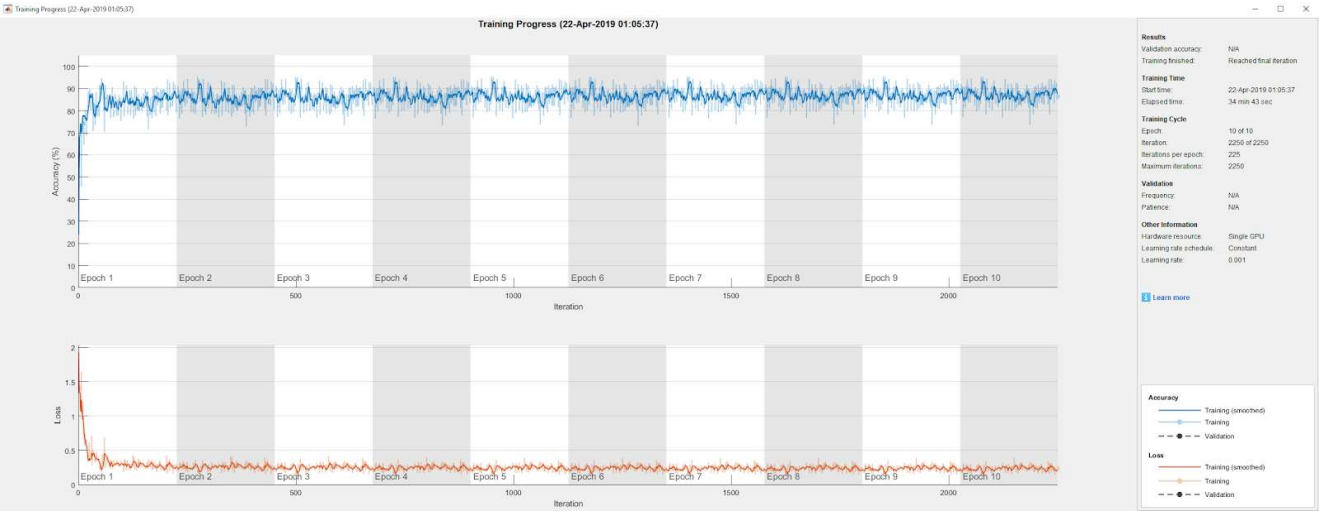

Figure 6. CNN in-progress Training results

Our CNN seemed to classify undesirable results such as roads being classified as buildings. Applying the class weights to the last classification layer seemed to be the main reason, as our results without the class weights didn't misclassify the roads, but without the class weights, some buildings were not being classified either. Our CNN's accuracy could be improved by adding more layers to the CNN (even adding layers for edging and rectangles similar to Windowed Hough Transform), having significantly more images to train with, having more memory available in our hardware to allow for greater parameters, using AdaBoosting and/or using bagging. For our project, we sought to create a CNN that could generate results to comparable to the 2 other algorithms, and even with our simple 10 layer CNN, we obtained results that were more accurate than the other algorithms.

Approximate Nearest Neighbors (ANN)

The ANN algorithm used a randomly chosen set of 20 images out of 18000 training set. There are two options implemented on how to label each segment. One option is to use the major of the features on each super-pixel. The other option is to use the mean from the segmented ground truth image.

On average, It took 10.6 seconds for initialization, 98.44 seconds for feature extraction, 2.2 seconds to set up the algorithm, and 834 seconds to process the 20 selected training images. It took 210 repeated interactions to complete the ANN algorithm.

At the evaluation step, we used a created MATLAB function "imbinarize" to find the Otsu threshold. On average,78.64% of pixels were classified correctly according to the ground truth. However, the algorithm does not guarantee that every segment in the training image set will be matched to a segment in the ground truth image set. As a result, an average of 43.29% of the pixels was not classified at all. This can likely be improved by fine turning the radius and features in the propagation and search step.

Rectangle Detection based on Clustering Results

Multiple attempts were made to classify aerial images into different clusters. Although K-means and a couple of its derived versions are commonly used for clustering in many different applications, we noticed that these algorithms were not the optimal solution. Since pixel intensity of different buildings in the same image could vary while all being greater than the background, we applied multilevel image thresholds using Otsu's method, which returned result that was close to the one of K-means with much less computation time. In addition to the thresholding, we also performed an octagon-based morphological erosion onto the processed image to further separate the objects in the image. As shown in the image on the right, features like roads and cars were highly suppressed, whereas buildings and other objects with flat surfaces remained relatively intact and were fully separated.

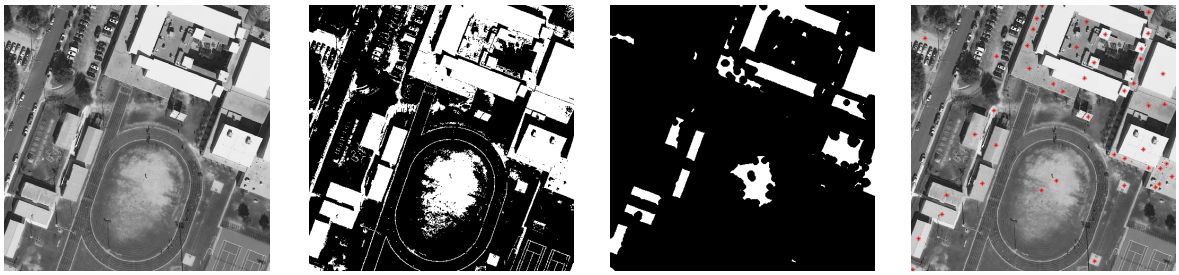

Figure 7. Original aerial image (left), thresholding (middle left), morphological erosion (middle right), and calculated center for each cluster (right)

To determine the center of each cluster for the following process, we used a MATLAB function called *regionprops* which returned properties like center location, major and minor axis length for each of the 8-connected component in the binary image. With its help, we located the center of all possible buildings in the image and mapped them on top of the original image. It is inevitable that many continuous regions were also picked up by the algorithm and marked as 8-connected clusters. This would not be an issue because later we would run a series of algorithms to check whether the labeled cluster was actually a building.

The next step of this method was to crop a series of images based on the clusters. Given the center location and the size of the clusters, we can easily define the window for each of these cropped images. We also set a threshold for the size of window that would be generated. If the major axis length of the original cluster was smaller than a certain threshold, we simply marked that cluster as noise and moved on. For each of these windows, we performed Sobel edge detection algorithm and Hough transform. Results for each of these intermediate steps are shown below:

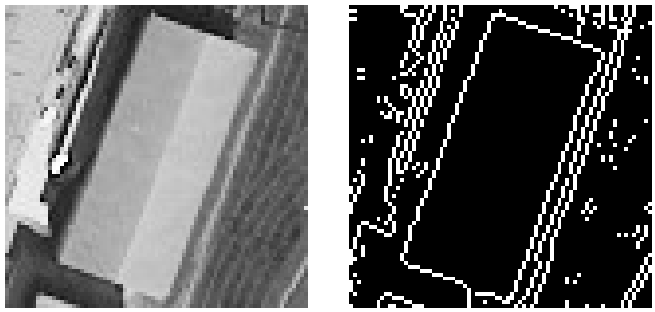

Figure 8. Cropped image (left) and edge detection (right)

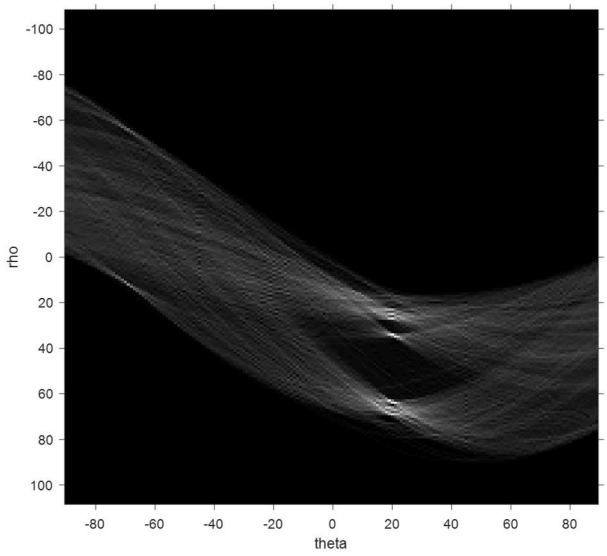

Figure 9. Hough transform of the edge detection image

Comparing to the sample image shown earlier, the Hough transform of a real image had a far worse quality. An unexpected complication also arose due to the fact that MATLAB processed its images based on matrix, meaning the origin of the image would be at the top left corner instead of being at the center of the image. While the geometric properties of a rectangle still hold true in this Hough space, the corresponding math become more complicated, and is the main focus of this method.

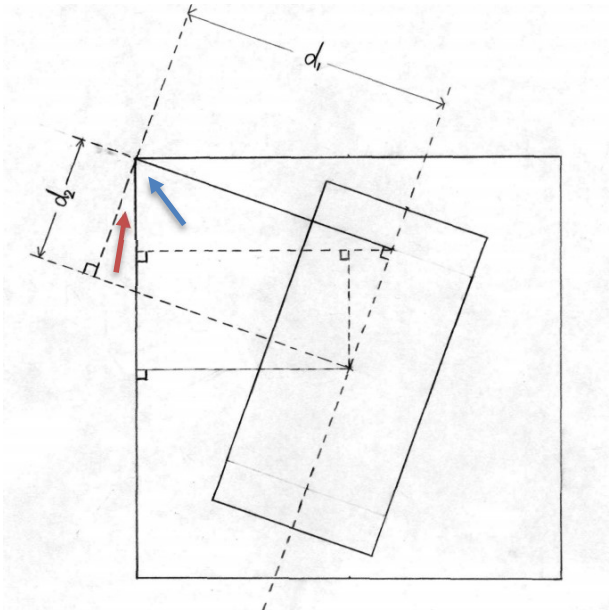

Figure 10. A hand sketched geometric diagram of a rectangular building inside a square window

To give an overview of the Hough space, the bright dots in the figure are the local maxima of the Hough transform, which represent the straight lines in the original image. For each of the local maximum, the θ axis is determined by the normal angle of the line, and the ρ axis is determined by the distance from the top left corner to the line. As shown on the right half of the image where a couple of local maxima are visible, the upper ones have ρ values around 20~30, and the lower ones have ρ values around 60~70. Since they are all positive, these two groups of lines must be on the same side of the origin, meaning these two groups of local maxima are the longer sides of the rectangle. It makes even more sense as the longer sides also make the local maxima on the right brighter than the local maxima of the shorter sides, which are located on the left side of the Hough space. With this in mind, we came up with an updated version of the specific relations based on the geometric diagram shown above. For the right half of the Hough transform, normal angle of the lines can be determined as the θ value, which is indicated as red arrow in the diagram. A distance from the origin to the middle line of the rectangle d_1 was also calculated by averaging the ρ values of the local maxima. It was also given that the window size, denoted as D, would be determined by the cluster size and was 77 for this specific one.

$$(\frac{D}{2} - d_1 sin\theta) * tan\theta + \frac{D}{2} = d_1 cos\theta$$

While the size of D was fixed for each window, values of θ and ρ could vary for each local maximum. However, only a specific set of θ and ρ can satisfy the equation above. With a carefully adjusted threshold, we can filter out most of the local maxima in the right side of the Hough space. For the left side of the Hough space, the local maxima would be the shorter sides of the rectangles. We denote d_2 as the averaged distance from the origin to the two sides, and their angle to the origin is indicated as blue arrow in the geometric diagram. A similar math can be used to check whether the available local maxima satisfy the conditions:

$$\left(\frac{D}{2} - \frac{d_2}{sin\theta}\right) * tan\theta = \frac{D}{2}$$

With the help of the two equations above, we can narrow down the candidate of local maxima that represent the true rectangle. A couple of additional steps were applied before we finalized our selection, which included finding the pair of local maxima that had the maximum Hough transform value and checking whether the two pairs of local maxima were separated by 90 degrees.

Unfortunately, as shown in the figure below. This algorithm only worked for some regions in the image, and it had trouble locating some rectangles (buildings) with a different angle of rotation. By checking the intermediate outputs from the checking equation, we noticed that some local maxima that appeared to be valid options were not selected by the algorithm. This is possibly due to the fact that the implementation of the checking equations did not fully cover all possible scenarios. However, it is convincing to see how accurate and robust this algorithm could be based on the result shown on the left. With a complete version of this rectangle detection algorithm, we shall expect a significant improvement over the classification accuracy. For the method comparison, we simply use the K-means method to just give an idea of how accuracy may vary among three different types of classification methods.

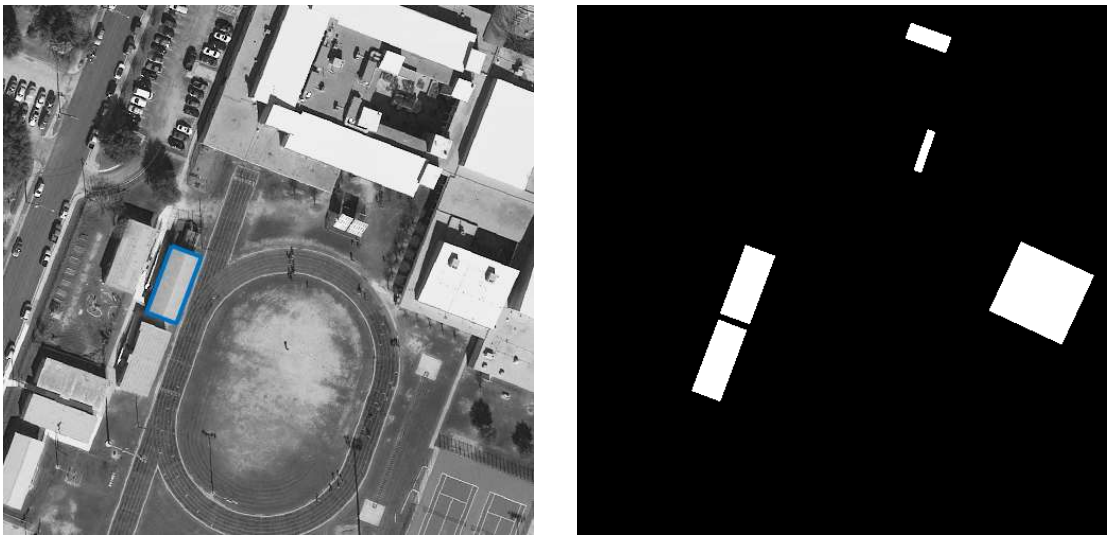

Figure 11. Found rectangle (left) and generated label maps (right)

IV. Datasets

We obtain our training data from Inria Aerial Image Labeling Dataset [14]. We divided the data set into smaller pieces of around 36,000 training/ground truth and test set. This preprocess helps us get around the limitation of processing an amount of memory size which MATLAB cannot handle.

V. Simulation results

Method	Accuracy (%)	Resource (MB)	Performance (sec)
CNN	87.11	4403	2083
ANN	78.64	1013	834
K-Means	66.16	934	636

Table 1. Comparison among three different classification approaches.

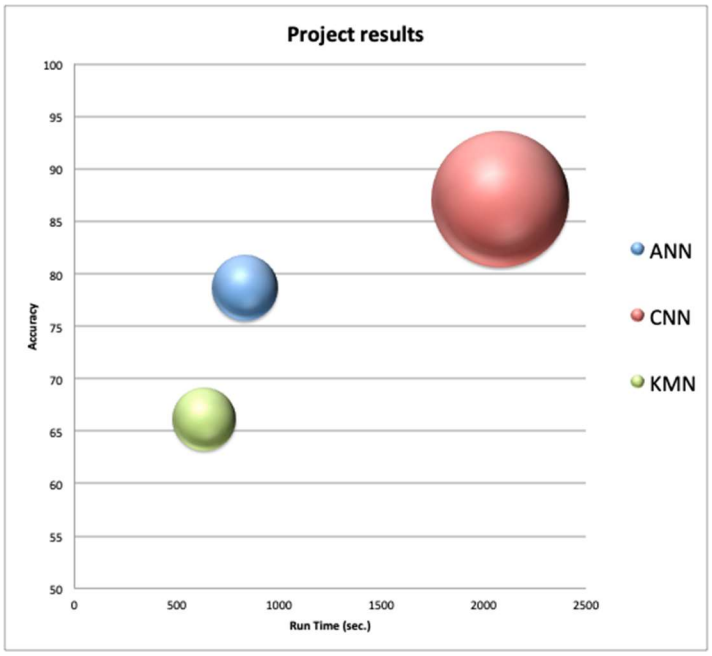

Figure 12. Visualization on simulation results.

VI. Conclusion:

With expected results, Convolutional Neural Network (CNN) provides the highest accuracy in image labeling and semantic segmentation compare to Approximate Nearest Neighbors (ANN) and Windowed Hough Transform (WHT). However, the main issue observed with using this method is that its high accuracy requires training on a very large data set with powerful computation resources. This requirement becomes prohibitive when there isn't access to a cloud machine or a very powerful computer. It's much simpler and easy to interpret aerial imagery with an ANN or WHT approach. With some improvements, these algorithms have the potential to achieve similar results compared to CNN.

References:

[1] Mnih, Volodymyr. “Machine Learning for Aerial Image Labeling” https://www.cs.toronto.edu/~vmnih/docs/Mnih_Volodymyr_PhD_Thesis.pdf

[2] Ankit, Utkarsh. “Semantic Segmentation of Aerial images Using Deep Learning” https://towardsdatascience.com/semantic-segmentation-of-aerial-images-using-deep-learning-90fdf4ad780

[3] Ronneberger, O., Fischer, P., & Brox, T. (2015). U-Net: Convolutional Networks for Biomedical Image Segmentation.

[4] Jordan, Jeremy. “An overview of image segmentation” https://www.jeremyjordan.me/semantic-segmentation/

[5] Jianbo Shi, J., & Malik. (2000). Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 22*(8), 888-905.

[6] Lin, K., Gong, L., Huang, Y., Liu, C., & Pan, J. (2019). Deep Learning-Based Segmentation and Quantification of Cucumber Powdery Mildew Using Convolutional Neural Network. *Frontiers in Plant Science, 10*. doi:10.3389/fpls.2019.00155

[7] Giraud, R., Ta, V.T., Bugeau, A., Coup, P., and Papadakis, N. (2019). SuperPatchMatch: an Algorithm for Robust Correspondences using Superpixel Patches https://arxiv.org/pdf/1903.07169.pdf

[8] Bi, Kim, Ahn, Kumar, Feng, & Fulham. (2019). Step-wise integration of deep class-specific learning for dermoscopic image segmentation. *Pattern Recognition, 85*, 78-89.

[9] R. Sawhney, F. Li, and H. I. Christensen, “GASP: Geometric association with surface patches,” in Proc. 3DV, 2014, pp. 107–114.

[10] C. Barnes, E. Shechtman, A. Finkelstein, and D. Goldman, “PatchMatch: A randomized correspondence algorithm for structural image editing,” ACM Trans. Graph., vol. 28, no. 3, 2009.

[11] Inria Aerial Image Labeling Dataset https://project.inria.fr/aerialimagelabeling/

[12] A. Rosenfeld and I. Weiss. A convex polygon is determined by its hough transform. *Pattern Recognition Letters*, 16(3):305 – 306, March 1995.

[13] Jung, C., & Schramm, R. (2004). Rectangle detection based on a windowed Hough transform. Proceedings. 17th Brazilian Symposium on Computer Graphics and Image Processing, 113-120.

Team Responsibilities:

All members: Proposal
Zi Hang Fang: Windowed Hough Transform (presentation, report)
Zhi Yuan Liao: Approximate Nearest Neighbor (presentation, report)
Cristian Stransky: Convolution Neural Network (presentation, report)

Even though we were responsible for our separate methods, all members helped the others in their respective methods when help was needed.

Appendix:
Convolutional Neural Network (CNN)

```
training = false; % SET THIS TO "true" TO TRAIN THE NETWORK
if ~training
    load('cnn_parameters.mat');
end

%%%%%%%%%%%%% CHANGE FOLDER CONTAINING THE PROJECT HERE %%%%%%%%%%%%
localFolder =
'C:\Users\Bazooka\Desktop\EECE5644_Machine_Learn_Pattern_Recogntn\project';
%localFolder =
'/home/bazooka/Documents/EECE5644_Machine_Learn_Pattern_Recogntn/project';
imageFolder = fullfile(localFolder, 'trainImagesSelect');
resultsFolder = fullfile(localFolder, 'trainResultsSelect');
imds = imageDatastore(imageFolder, 'LabelSource', 'foldernames',
'IncludeSubfolders',true);

if training
    buildingRGB = 255;
    terrainRGB = 0;
    classNames = ["buildings", "terrain"];
    pixelLabelID = [buildingRGB terrainRGB];
    pxds = pixelLabelDatastore(resultsFolder,classNames,pixelLabelID);

    I = read(imds);

    numFilters = 32;
    filterSize = 3;
    numClasses = 2;
    layers = [
        imageInputLayer([500 500 3])
        convolution2dLayer(filterSize,numFilters,'Padding',1)
        reluLayer()
        maxPooling2dLayer(2,'Stride',2)
        convolution2dLayer(filterSize,numFilters,'Padding',1)
        reluLayer()
        transposedConv2dLayer(4,numFilters,'Stride',2,'Cropping',1);
        convolution2dLayer(1,numClasses);
        softmaxLayer()
        pixelClassificationLayer()
        ];

    opts = trainingOptions('sgdm', ...
        'InitialLearnRate',0.001, ...
        'MaxEpochs',10, ...
        'MiniBatchSize',16, ...
        'Plots','training-progress'); ...

    trainingData = pixelLabelImageDatastore(imds,pxds);

    tbl = countEachLabel(trainingData);
    totalNumberOfPixels = sum(tbl.PixelCount);
    frequency = tbl.PixelCount / totalNumberOfPixels;
```

```
    classWeights = 1./frequency;
    layers(end) =
pixelClassificationLayer('Classes',tbl.Name,'ClassWeights',classWeights);

    net = trainNetwork(trainingData,layers,opts);
end

testImage = readimage(imds,2);
imshow(testImage);
C = semanticseg(testImage,net);
B = labeloverlay(testImage,C);
imshow(B)
```

Approximate Nearest Neighbors (ANN)

```
close all; clear; clc;

% for reproducibility
rng('default');

cd('../trainImages');
train_img_files = dir('*.tif');
cd('../trainResults');
train_gt_files = dir('*.tif');
cd('../ANN')

% Cropped AerialImageDataset contains 18,001 images. Randomly choose N
N = 20;
radius = 20;
train_image = cell(N, 1);
ground_true = cell(N, 1);
X = randperm(length(train_img_files), N);
train_img_files = train_img_files(X);
train_gt_files = train_gt_files(X);
n = zeros(N, 1);


% patch-based approximate nearest neighbor (ANN) search methods are used to
% find correspondences.
ANN = cell(N, N);

% Requested 5000x5000x100 (18.6GB) array exceeds maximum array size
preference.
% Creation of arrays greater than this limit may take a long time and cause
% MATLAB to become unresponsive.
% Data = zeros(5000, 5000, N);
Data = zeros(500, 500, N);
tic;

%% ANN initialization step.
disp('Step 1: ANN initialization step. ')
for i = 1:N
   img =
imread(strcat('C:\Users\Bazooka\Desktop\EECE5644_Machine_Learn_Pattern_Recogn
tn\project\trainImages\', train_img_files(i).name));
```

```
    gt = imread(strcat('../trainResults/', train_gt_files(i).name));
    [l, h, w] = size(img);
    [L, nLables] = superpixels(img, floor(sqrt(l*h)));

    % ANN initialization step, For each superpixel Ai in A, we assign a
    % random superpixel B(i) in B.
    L = patMat_init(L);

    % image too large for 5000x5000
    Data(:, :, i) = L;
    n(i) = nLables;
    train_image{i} = img;
    ground_true{i} = gt;
    fprintf('%d ', i)
end
disp('%n Done! ')


%% ANN propagation step.
disp('Step 2: ANN propagation step.')
labels = cell(N, 1);

% assign features to a variable
features = cell(N, 1);
for i = 1:N
    labels{i} = label_img(ground_true{i}, Data(:, :, i), n(i));
    for j = 1:n(i)
        features{i} = [features{i} ; m_tri(j, Data(:, :, i), ...
            double(train_image{i}))];
    end
    fprintf('%d ', i)
end

disp('%n Done! ')

%% ANN random search step.
for i = 1:N
    i_A = train_image{i};
    L_A = Data(:, :, i);
    n_A = n(i);
    f_A = features{i};
    for j = (i+1):N
        fprintf('\nperformaning ANN on images %d and %d:\n', i, j)
        i_B = train_image{j};
        L_B = Data(:, :, j);
        n_B = n(j);
        f_B = features{j};

        % rearrange if the the image don't have enought superpixel
        if (n_B < n_A)
            temp=i_A; i_A=i_B; i_B=temp;
            temp=L_A; L_A=L_B; L_B=temp;
            temp=n_A; n_A=n_B; n_B=temp;
            temp=f_A; f_A=f_B; f_B=temp;
        end
```

```
        % Assign pathc match result to approximate nearest neighor matrix
        ANN{i, j} = patMat(i_A, L_A, n_A, f_A, i_B, L_B, n_B, f_B, radius);
    end
end

% show total time spent on running step 1 to 3
fprintf('Total time spent: %f\n', toc)

%% Evaluate solution

% Save ANN results
save('output','ANN','Data','features','ground_true','train_image',...
    'labels','n');

Loss = 0;
for l = 1:length(n)
    IMG_ = train_image{l};
    GT_ = logical(ground_true{l});

    threshold = imbinarize(rgb2gray(IMG_), graythresh(IMG_));
    Loss = Loss+mean(mean(abs(GT_-threshold)));
end

Loss = Loss/N;
fprintf('Loss of accuarcy during binary transformation = %f\n', Loss)

% find correctness
n_correct = ones(length(n)) .* Inf;
for i = 1:length(n)
    labels_A = labels{i};
    n_A = n(i);
    for j = (i+1):length(n)

        ANN_AB = ANN{i, j};
        n_B = n(j);
        try
            labels_B = labels{j};

            labels_B_est = ones(n_B, 1) .* inf;

            for k = 1:length(labels_A)
                labels_B_est(ANN_AB(k)) = labels_A(k);
            end

            l_compare = [labels_B, labels_B_est];
            l_compare = l_compare(labels_B_est ~= Inf, :);
            count_l = l_compare(:, 1) == l_compare(:, 2);
            n_correct(i, j) = length(count_l(count_l == 1))/length(count_l);
        catch
        end
    end
end
```

```
final_result = mean(mean(n_correct(n_correct ~= Inf)));

fprintf('Percentage correctness = %f\n', final_result*100)

function Output = sp_weight(c_Ai, c_Ai_, c_Bj, c_Bj_, m, n, nA, radius)

    ws_Ai_ = exp(-((c_Ai(2)-c_Ai_(2))^2+(c_Ai(1)-c_Ai_(1))^2)/(2*radius^2));
    ws_Bj_ = exp(-((c_Bj(2)-c_Bj_(2))^2+(c_Bj(1)-c_Bj_(1))^2)/(2*radius^2));

    Output = exp(-(c_Bj_ - c_Ai_+ c_Ai - c_Bj)*(c_Bj_-c_Ai_+c_Ai-
c_Bj)'/(0.5 ...
        *sqrt(m*n/nA))) * ws_Ai_ * ws_Bj_;
end

function Output = sp_dist(Ai, Bj, c_Ai, c_Bj,c_A, c_B, radius, nA, ...
    F_A, F_B, row, col)
    % find super patch distance base on give radius
    num_ij = 0;
    den_ij = 0;

    for k = Ai
        nc_Ak = c_A(k, :);
        nF_Ak = F_A(k);
        for l = Bj
            nc_Bl = c_B(l, :);
            nF_Bl = F_B(l);

            weight_ij = sp_weight(c_Ai, nc_Ak, c_Bj, nc_Bl, row, col, nA,
radius);

            num_ij = num_ij + (weight_ij * norm(nF_Bl - nF_Ak));
            den_ij = den_ij + weight_ij;
        end
    end

    Output = num_ij / den_ij;
end

function Output = sp_angle(c_A, c_B)
    % find super patch angle base on given centers barycenters
    x = c_B(1) - c_A(1);
    y = c_B(2) - c_A(2);

    angle = atan2(y, x);

    if (c_B(2) < c_A(2))
        Output = -angle;
    else
        Output = 2*pi - angle;
    end
end

function Output = SPM_iter(i, j, ANN, adj_A, c_A, spm_A, n_A, f_A, ...
    adj_B, c_B, spm_B, n_B, f_B, radius, row, col)
```

```
    pooling = 0;
    thresholdAngle = inf;
    Ai = find(spm_A(i, :) == 1);
    ANNi = find(spm_B(ANN(i), :) == 1);

    % distance between Ai and its current ANN
    center_Ai = c_A(i, :);
    center_ANNi = c_B(ANN(i), :);

    % sp_dist(i, ANN(i)) = min_dist;
    min_dist = sp_dist(Ai, ANNi, center_Ai, center_ANNi, c_A, c_B,
radius, ...
        n_A, f_A, f_B, row, col);

    find_adj_Ai = find(adj_A(i, :) == 1);
    find_adj_Ai = find_adj_Ai(find_adj_Ai ~= i);

    % Dual interation
    if (mod(j, 2) == 0)
        for i_ = find_adj_Ai
            anglei_i = sp_angle(c_A(i, :), c_A(i_, :));
            if (0.25*pi <= anglei_i && anglei_i < 1.25*pi)
                find_adj_Bi = find(adj_B(i_, :) == 1);
                find_adj_Bi = find_adj_Bi(find_adj_Bi ~= i_);

                for k = find_adj_Bi
                    % add overflower checker
                    if k>length(c_B)
                        break;
                    end
                    anglei_k = sp_angle(c_B(i_, :), c_B(k, :));
                    absAngle = abs(anglei_k - anglei_i);
                    if (absAngle < thresholdAngle)
                        pooling = k;
                        thresholdAngle = absAngle;
                    end
                end
            end

            % varify the distance
            try
                superpatch_Bk = find(spm_B(pooling, :) == 1);
                center_Bk = c_B(pooling, :);
                comp_dist = sp_dist(Ai, superpatch_Bk, center_Ai,
center_Bk, ...
                    c_A, c_B, radius, n_A, f_A, f_B, row, col);
                % save the minimun distance super patch to ANN
                if (comp_dist < min_dist)
                    ANN(i) = pooling;
                    min_dist = comp_dist;
                end
            catch
            end
        end
```

```
    % Single interation
    else
        for i_ = find_adj_Ai
            anglei_i = sp_angle(c_A(i, :), c_A(i_, :));
            if (anglei_i < 0.25*pi || (1.25*pi <= anglei_i && anglei_i <=
2*pi))
                find_adj_Bi = find(adj_B(i_, :) == 1);
                find_adj_Bi = find_adj_Bi(find_adj_Bi ~= i_);

                for k = find_adj_Bi
                    % add overflower checker
                    if k>length(c_B)
                        break;
                    end
                    anglei_k = sp_angle(c_B(i_, :), c_B(k, :));
                    absAngle = abs(anglei_k - anglei_i);
                    if (absAngle < thresholdAngle)
                        pooling = k;
                        thresholdAngle = absAngle;
                    end
                end
            end

            % varify the distance
            try
                superpatch_Bk = find(spm_B(pooling, :) == 1);
                center_Bk = c_B(pooling, :);
                comp_dist = sp_dist(Ai, superpatch_Bk, center_Ai,
center_Bk, ...
                    c_A, c_B, radius, n_A, L_A, f_A, f_B, row, col);
                % save the minimun distance super patch to ANN
                if (comp_dist < min_dist)
                    ANN(i) = pooling;
                    min_dist = comp_dist;
                end
            catch
            end
        end
    end

    % find the number of superpixels to test from image B
    testset = randperm(n_B, 8);
    for j = testset
       if j ~= ANN(i)
          superpatch_Bj = find(spm_B(j,:) == 1);
          comp_dist =  sp_dist(Ai, superpatch_Bj, center_Ai, c_B(j, :), ...
              c_A, c_B, radius, n_A, f_A, f_B, row, col);
          dist_m(i, j) = comp_dist;
          if (comp_dist < min_dist)
              ANN(i) = j;
              min_dist = comp_dist;
          end
       end
    end
```

```
    Output = ANN;
end

function Output = patMat_init(X)
    Output = zeros(size(X));
    Y = randperm(max(max(X)));

    for i = 1:length(Y)
        Output(find(X == i)) = Y(i);
    end
end

function ANN = patMat(i_A, L_A, n_A, f_A, i_B, L_B, n_B, f_B, radius)

    [row, col] = size(L_A);
    % find adjance patch and label it with 1
    adj_A = find_adj_m(L_A);
    adj_B = find_adj_m(L_B);

    % list of barycenter of each patch
    sA = regionprops(L_A, 'centroid');
    c_A= cat(1, sA.Centroid);
    sB = regionprops(L_B, 'centroid');
    c_B = cat(1, sB.Centroid);

    % find patch match within radius equal to 1
    spm_A = gen_spm(c_A, radius);
    spm_B = gen_spm(c_B, radius);

    % patch match Inizializes as ANN(i) = i
    ANN = (1:1:n_A)';

    for i = 1:n_A-1

        for j = 1:4
            ANN = patMat_iter(i, j, ANN, adj_A, c_A, spm_A, n_A, f_A, ...
                adj_B, c_B, spm_B, n_B, f_B, radius, row, col);
        end
        %fprintf('%d ', i)
    end

    disp('%n Done! ')
end

function Output = m_tri(idx, Data, image)
    % Calculate the mean of the tristimulus values of Data..

    red_value = (Data == idx) .* image(:, :, 1);
    green_value = (Data == idx) .* image(:, :, 2);
    blue_value = (Data == idx) .* image(:, :, 3);

    RED = mean(mean(red_value(find(red_value ~= 0))));
    GREEN = mean(mean(green_value(find(green_value ~= 0))));
    BLUE = mean(mean(blue_value(find(blue_value ~= 0))));
```

```
    Output = [RED, GREEN, BLUE];
end

function Output = gen_spm(center, radius)
    % find the super patches fom the given radius
    Output = zeros(length(center));

    for i = 1:length(center(:, 1))
        temp1 = center(i, :);

        for j = 1:length(center(:, 1))
            temp2 = center(j, :);
            if (temp2(1)-temp1(1))^2+(temp2(2)-temp1(2))^2<=radius^2
                Output(i, j) = 1;
            end
        end
    end
end

function Output = find_adj_m(X)
    [row, col]=size(X);
    Output=zeros(row, col);

    for i=1:row
        for j=1:col
            if abs(i-j)==1
                Output(i,j)=1;
            end
        end
    end
end

function Output = label_img(gt, Data, numIdx)
    % label Data using the barycenter of superpatch of gt

    Y = regionprops(Data, 'centroid');
    barycenter = round(cat(1, Y.Centroid));
    Output = zeros(numIdx, 1);

    for i = 1:numIdx
       Output(i) = gt(barycenter(i, 2), barycenter(i, 1));
    end
end
```

Windowed Hough Transform

```
%% Look for Centers of Potential Clusters
clearvars; close all; clc;
I = rgb2gray(imread('E:\Download\Spring 2019\Machine
Learning\EECE5644_ANN\images\austin21_cropped_37.tif'));
level = multithresh(I);
seg_I = imquantize(I,level);
seg_I = seg_I - 1;
```

```
SE = strel('octagon',6);
J = imerode(seg_I,SE);
J = J > 0;
stats = regionprops(J,I,{'Centroid','MajorAxisLength','MinorAxisLength'});

figure(2)
imshow(I)
hold on
for k = 1:numel(stats)
    plot(stats(k).Centroid(1),stats(k).Centroid(2),'r*')
end
hold off

% Some Thresholds
diagonal_threshold = 7;
radiusMargin1 = 5;
radiusMargin2 = 1;
horizontalAlignment1 = 5;
horizontalAlignment2 = 10;
verticalTolerance1 = 0.3;
verticalTolerance2 = 0.1;
theta90_threshold = 5;
%{
%% Edge Detection with Fuzzy Logic
sharpened_I = imsharpen(I,'radius',2,'Amount',1);
tempI = im2double(sharpened_I);
Gx = [-1 1];
Gy = Gx';
Ix = conv2(tempI,Gx,'same');
Iy = conv2(tempI,Gy,'same');
edgeFIS = mamfis('Name','edgeDetection');
edgeFIS = addInput(edgeFIS,[-1 1],'Name','Ix');
edgeFIS = addInput(edgeFIS,[-1 1],'Name','Iy');
sx = 0.1;
sy = 0.1;
edgeFIS = addMF(edgeFIS,'Ix','gaussmf',[sx 0],'Name','zero');
edgeFIS = addMF(edgeFIS,'Iy','gaussmf',[sy 0],'Name','zero');
edgeFIS = addOutput(edgeFIS,[0 1],'Name','Iout');
wa = 0.1;
wb = 1;
wc = 1;
ba = 0;
bb = 0;
bc = 0.7;
edgeFIS = addMF(edgeFIS,'Iout','trimf',[wa wb wc],'Name','white');
edgeFIS = addMF(edgeFIS,'Iout','trimf',[ba bb bc],'Name','black');
r1 = "If Ix is zero and Iy is zero then Iout is white";
r2 = "If Ix is not zero or Iy is not zero then Iout is black";
edgeFIS = addRule(edgeFIS,[r1 r2]);
Ieval = zeros(size(I));
for ii = 1:size(I,1)
    Ieval(ii,:) = evalfis(edgeFIS,[(Ix(ii,:));(Iy(ii,:))]');
end
%}

%% Edge Detection with Sobel
```

```
sharpened_I = imsharpen(I,'radius',2,'Amount',1);
[~,threshold] = edge(sharpened_I,'sobel');
fudgeFactor = 0.5;
bw = edge(sharpened_I,'sobel',threshold * fudgeFactor);

%% Mark Label
finalBW = poly2mask(0,0,500,500);
for i = 8
    diagonal = (sqrt(stats(i).MajorAxisLength^2 +
stats(i).MinorAxisLength^2));
    if diagonal >= diagonal_threshold
        center = (stats(i).Centroid);
        outerR = diagonal;
        innerR = (stats(i).MinorAxisLength);
        outerR = outerR/2 + radiusMargin1;
        innerR = innerR/2 - radiusMargin2;
        %thisWindow = bw(center(2)-outerR:center(2)+outerR, center(1)-
outerR:center(1)+outerR);
        thisWindow = imcrop(bw,[center(1)-outerR center(2)-outerR
diagonal+2*radiusMargin1-1 diagonal+2*radiusMargin1-1]);
        [xgrid, ygrid] = meshgrid(1:size(thisWindow,2),1:size(thisWindow,1));
        ringMask = ((xgrid-size(thisWindow,2)/2).^2 + (ygrid-
size(thisWindow,1)/2).^2) <= outerR.^2 & ((xgrid-size(thisWindow,2)/2).^2 +
(ygrid-size(thisWindow,1)/2).^2) >= innerR^2;
        thisRing = thisWindow .* ringMask;
        [H,T,R] = hough(thisRing,'RhoResolution',1,'Theta',-90:89.5);

        % Split H and T in halves along the theta axis and find local maxima
separately
        left_H = H(:,1:90);
        right_H = H(:,91:180);
        left_T = T(:,1:90);
        right_T = T(:,91:180);

        % find local maxima in hough transform
        left_P = houghpeaks(left_H,10,'threshold',ceil(0.4*max(left_H(:))));
        right_P =
houghpeaks(right_H,10,'threshold',ceil(0.4*max(right_H(:))));
        realPeaks1 = [];
        maxIntensity = 0;
        for j = 1:size(right_P,1)
            for k = 1:size(right_P,1)
                lm_A = [right_T(right_P(j,2)) R(right_P(j,1))];
                lm_B = [right_T(right_P(k,2)) R(right_P(k,1))];
                result = check_vertical_alignment(lm_A, lm_B,
size(thisWindow,1), horizontalAlignment1, verticalTolerance1);
                avgIntensity1 = (right_H(right_P(j,1),right_P(j,2)) +
right_H(right_P(k,1),right_P(k,2))) / 2;
                if result && (avgIntensity1 > maxIntensity)
                    realPeaks1 = [lm_A;lm_B];
                    maxIntensity = avgIntensity1;
                end
            end
        end

        realPeaks2 = [];
```

```
        maxIntensity2 = 0;
        for m = 1:size(left_P,1)
            for n = 1:size(left_P,1)
                lm_C = [left_T(left_P(m,2)) R(left_P(m,1))];
                lm_D = [left_T(left_P(n,2)) R(left_P(n,1))];
                result = check_vertical_alignment2(lm_C, lm_D,
size(thisWindow,1), horizontalAlignment2, verticalTolerance2);
                avgIntensity2 = (left_H(left_P(m,1),left_P(m,2)) +
left_H(left_P(n,1),left_P(n,2))) / 2;
                if result && (avgIntensity2 > maxIntensity2)
                    realPeaks2 = [lm_C;lm_D];
                    maxIntensity2 = avgIntensity2;
                end
            end
        end
        if ~isempty(realPeaks1) && ~isempty(realPeaks2)
            if abs(abs((realPeaks1(1,1)+realPeaks1(2,1))/2 -
(realPeaks2(1,1)+realPeaks2(2,1))/2) - 90) <= theta90_threshold
                coords =
drawRectangleonImageAtAngle(I,center',abs(realPeaks1(1,2)-
realPeaks1(2,2)),abs(realPeaks2(1,2)-realPeaks2(2,2)),-
abs((realPeaks1(1,1)+realPeaks1(2,1))/2));
                finalBW = poly2mask(coords(1,:),coords(2,:),500,500) +
finalBW;
            end
        end
    end
end
imshow(finalBW)

figure(2)
%imshow(imadjust(rescale(H)),'XData',T,'YData',R,'InitialMagnification','fit'
);
H = right_H;
T = right_T;
P = right_P;
imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
axis on, axis normal; hold on;
%P = houghpeaks(H(:,end/2:end),10,'threshold',ceil(0.4*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');
%plot(T(peaks(:,2)),R(peaks(:,1)),'s','color','white');

%%
figure(3)
%imshow(imadjust(rescale(H)),'XData',T,'YData',R,'InitialMagnification','fit'
);
H = left_H;
T = left_T;
P = left_P;
imshow(H,[],'XData',T,'YData',R,'InitialMagnification','fit');
axis on, axis normal; hold on;
%P = houghpeaks(H(:,end/2:end),10,'threshold',ceil(0.4*max(H(:))));
x = T(P(:,2)); y = R(P(:,1));
plot(x,y,'s','color','white');
%plot(T(peaks(:,2)),R(peaks(:,1)),'s','color','white');
```

```
function decision = check_vertical_alignment(pointA, pointB, windowSize,
threshold, tolerance)

    horzDifference = abs(pointA(1)-pointB(1));
    horzCheck = horzDifference <= threshold;
    vertDistance = abs((pointA(2)+pointB(2))/2);
    if horzCheck
        thisTheta = abs((pointA(1)+pointB(1))/2);
        temp = (vertDistance*cosd(thisTheta) - windowSize/2) /
tand(thisTheta);
        geoCheck = abs(temp - (windowSize/2 - vertDistance*sind(thisTheta)))
/ temp <= tolerance;
        if geoCheck && (vertDistance > windowSize/2) && (abs(pointA(2)-
pointB(2)) > 7)
            decision = 1;
        else
            decision = 0;
        end
    else
        decision = 0;
    end
end

function decision = check_vertical_alignment2(pointA, pointB, windowSize,
threshold, tolerance)

    horzDifference = abs(pointA(1)-pointB(1));
    horzCheck = horzDifference <= threshold;
    vertDistance = abs((pointA(2)+pointB(2))/2);
    if horzCheck
        thisTheta = abs((pointA(1)+pointB(1))/2);
        temp = (windowSize/2 - vertDistance/sind(thisTheta)) *
tand(thisTheta);
        geoCheck = abs(temp - windowSize/2) / temp <= tolerance;
        if geoCheck && (vertDistance < windowSize/2) && (abs(pointA(2)-
pointB(2)) > 7)
            decision = 1;
        else
            decision = 0;
        end
    else
        decision = 0;
    end
end
```

K-means

```
close all; clear all; clc

% for reproducibility
rng('default');

cd('C:\Users\Bazooka\Desktop\EECE5644_Machine_Learn_Pattern_Recogntn\project\
trainImages\');
train_img_files = dir('*.tif');
```

```
cd('C:\Users\Bazooka\Desktop\EECE5644_Machine_Learn_Pattern_Recogntn\project\
trainResults\');
train_gt_files = dir('*.tif');
cd('/..')

% Cropped AerialImageDataset contains 18,001 images. Randomly choose N
N = 20;
radius = 20;
train_image = cell(N, 1);
ground_true = cell(N, 1);
X = randperm(length(train_img_files), N);
train_img_files = train_img_files(X);
train_gt_files = train_gt_files(X);
n = zeros(N, 1);
K = 8;

tic;
accuracy = [];
for i = 1:N
   img =
im2double(imread(strcat('C:\Users\Bazooka\Desktop\EECE5644_Machine_Learn_Patt
ern_Recogntn\project\trainImages\', train_img_files(i).name)));
   gt =
im2double(imread(strcat('C:\Users\Bazooka\Desktop\EECE5644_Machine_Learn_Patt
ern_Recogntn\project\trainResults\', train_gt_files(i).name)));
   new_I = reshape(img,size(img,1)*size(img,2),3);

    c_center = new_I( ceil(rand(K,1)*size(new_I,1)) ,:);
    % Distances and Labels
    DAL = zeros(size(new_I,1),K+2);
    % K-means Iteration
    KMI = 8;
    for n = 1:KMI
       for i = 1:size(new_I,1)
           for j = 1:K
               DAL(i,j) = norm(new_I(i,:) - c_center(j,:));
           end
           % 1:K are Distance from Cluster Centers 1:K
           [Distance, labels] = min(DAL(i,1:K));
           % K+1 is Cluster Label
           DAL(i,K+1) = labels;
           % K+2 is Minimum Distance
           DAL(i,K+2) = Distance;
       end
       for i = 1:K
           % Cluster K Points
           A = (DAL(:,K+1) == i);
           % New Cluster Centers
           c_center(i,:) = mean(new_I(A,:));
           % If CENTS(i,:) Is Nan Then Replace It With Random Point
           if sum(isnan(c_center(:))) ~= 0
               % Find Nan Centers
               NC = find(isnan(c_center(:,1)) == 1);
               for Ind = 1:size(NC,1)
               c_center(NC(Ind),:) = new_I(randi(size(new_I,1)),:);
               end
```

```
            end
        end
    end

    X = zeros(size(new_I));

    for i = 1:K
        idx = find(DAL(:,K+1) == i);
        X(idx,:) = repmat(c_center(i,:),size(idx,1),1);
    end

    %seg_I = reshape(X,size(I,1),size(I,2),3);
    seg_I = round(rgb2gray(reshape(X,size(img,1),size(img,2),3)));

    match=0;
    [row, col, hig] = size(img);
    for i=1:row
        for j=1:col
            if gt(i,j) == seg_I(i,j)
                match=match+1;
            end
        end
    end
    match=match/(row*col);
    accuracy = [accuracy;match];
    fprintf('correctness = %f\n', match*100)
end
toc
```