CSC 391 Final Project: Object Detection & Image Segmentation

Instructor: V. Paúl Pauca
Benjamin Liu, Tony Xu, Eric Yang

1. Introduction

In this project, we work on palm tree segmentation. Previously we did palm tree detection which gives a binary output of whether an extracted feature is a palm or not a palm. We would like to improve with more complicated outputs, such as segmentation. Also, the scale of the training/testing images we did previously were so small that they contain only a palm leaf or a single palm tree. In this project, we would like to detect and reveal the palm trees with highlights in an image with a large scale.

2. Methodology

2.1 Datasets

In the palm tree datasets, we have some images of canopies among which palm trees can be identified at various scales. For this final project, our team took 30 satellite images as our raw data. All 30 satellite images were taken at an even height which shows much more canopy with palm trees spread out but still identifiable.

Figure 1. Sample Images From the Raw Dataset

Since we found that smaller patches have clearer and more accurate feature patterns in the past two projects, we implemented our code to crop images into 100×100-sized patches. As a result, we got 11400 patches from 6 images from the satellite images as our training set. Then, we manually separated all patches into two groups of palm (labeled as '1') and non-palm (labeled as '0') patterns.

```
path = '/Users/xjq/PycharmProjects/391 Final/raw_data/'
count = 1

for filename in os.listdir(path):
    img = cv2.imread(path + filename)
    # print(path + filename)
    height, width = img.shape[:2]
    start_row, start_col = int(0), int(0)
    delta_height = int(100)
    delta_width = int(100)
    height_range = int(height/100)
    width_range = int(width/100)
    for i in range(0, height_range):
        for j in range(0, width_range):
            start_row, start_col = i * delta_height, j * delta_width
            end_row, end_col = (i + 1) * delta_height, (j + 1) * delta_width
            cropped = img[start_row:end_row, start_col:end_col]
            imgPath = '/Users/xjq/PycharmProjects/391 Final/output_data/%s.jpg' % count
            cv2.imwrite(imgPath, cropped)
            count = count+1
```

Figure 2. Python Code for Cropping Raw Image into 100x100-Sized Patches

After the separation work, we got 800 patches labeled as palms and the rest as non-palms. We eventually saved 800 palm features and 880 non-palm feature as our training datasets. The rest images in the raw datasets will be used to test our image segmentation algorithm.

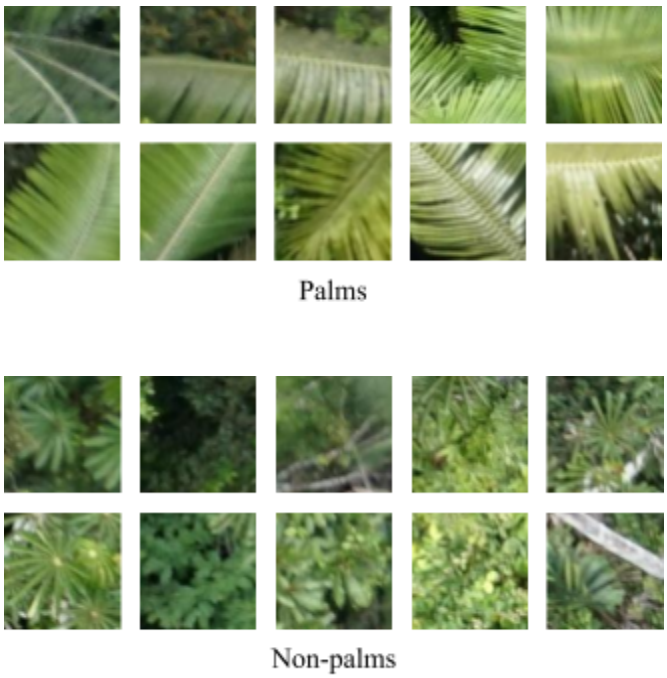

Figure 3. Data images of palms and non-palms

2.2 Histogram of Oriented Gradients (HOG)

Histogram of Oriented Gradients is a feature descriptor used to do object detection. When we implement the HOG descriptor algorithm, we first divide the image into small connected regions called cells, and for each cell compute a histogram of gradient directions or edge orientations for the pixels within the cell. Then, we discretize each cell into angular bins according to the gradient orientation. Each cell's pixel contributes weighted gradient to its corresponding angular bin. Groups of adjacent cells are considered as spatial regions called blocks. The grouping of cells into a block is the basis for grouping and normalization of histograms. We take these blocks and contrast normalizes their overall responses before passing to the next stage. Normalization introduces better invariance to illumination, shadowing, and edge contrast. Finally, the normalized group of histograms represents the block histogram. The set of these block histograms represents the descriptor.

The computation of the HOG descriptor requires parameters including masks to compute derivatives and gradients, the geometry of splitting an image into cells and grouping cells into a block, block overlapping, and normalization parameter.

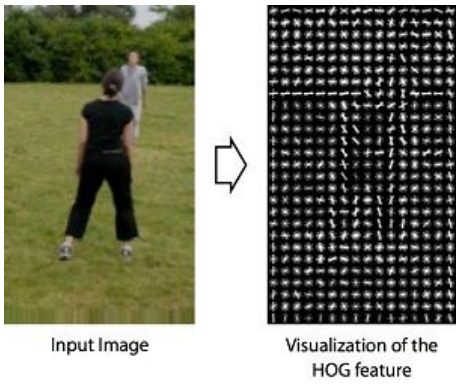

Figure 4. Example of Input Image and Corresponding Histogram of Oriented Gradients

In this project, we chose HOG as our feature descriptor for palm and non-palm images. We will train our model by extracting HOG features from palm and non-palm datasets.

2.3 Support Vector Machines

Support vector machines are a set of supervised learning methods used for classification, regression and outliers detection. The objective of the support vector machine algorithm is to find a hyperplane in N-dimensional space (N = the number of features) that distinctly classifies the data points.

SVM algorithms use a set of mathematical functions that are defined as the kernel. One of the advantages of Support Vector Machine is its versatility since we can implement different Kernel functions for the decision function, including linear, rbf, polynomial, sigmoid, and etc. The function of the kernel is to take data as input and transform it into the required form.

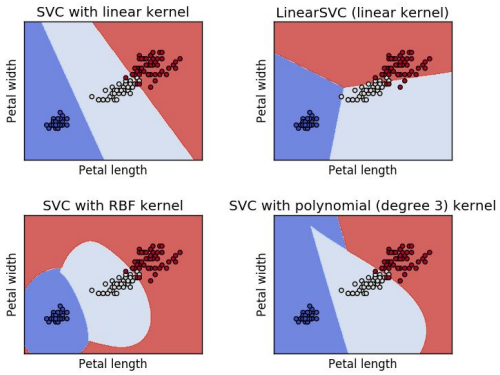

Figure 5. SVC with Various Kernels

We used SVC to classify our HOG features of palm and non-palm images. Since we labeled palm as '1' and non-palm as '0', we append these labels into a matrix as value y, while the matrix contains HOG features as value X.

3. Experimental Results

3.1 Accuracy of Prediction

Before applying the model on the raw data, we use K-fold (with K = 10) cross-validation method to test the model accuracy with our currently labeled data. We divide the whole dataset into 10 segments and use 1 as the testing set and the other 9 for training. Since we have a dataset

that contains 800 palm features and 880 non-palm features, the prediction accuracy for non-palm features is significantly higher than the prediction accuracy for palm features. Our resulting accuracy for predicting palm is 86%, and for predicting non-palm is 87%. The low accuracy for predicting palm is not promising for our segmentation and thus it results in some mismatches that will be described in Section 3.3.

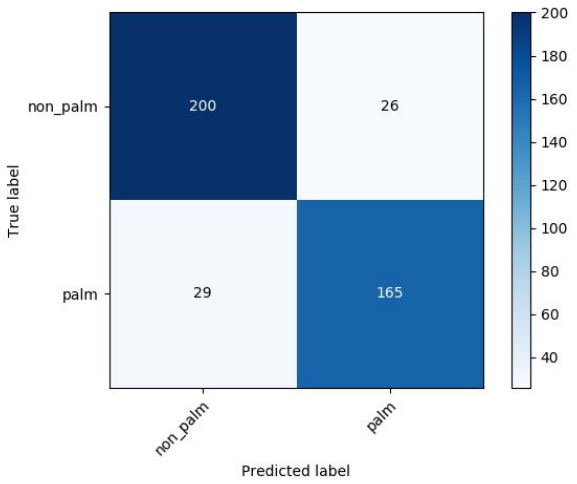

Figure 6. Confusion Matrix of Prediction

3.2 Image segmentation

After we trained our SVM model to do the prediction of palm and non-palm features, we take the SVM to do prediction on the entire canopy images.

```
for filename in os.listdir(path):
    img = cv2.imread(path + filename)
    print(path + filename)
    height, width = img.shape[:2]
    start_row, start_col = int(0), int(0)
    delta_height = int(100)
    delta_width = int(100)
    height_range = int(height/100)
    width_range = int(width/100)
    for i in range(0, height_range):
        for j in range(0, width_range):
            start_row, start_col = i * delta_height, j * delta_width
            end_row, end_col = (i + 1) * delta_height, (j + 1) * delta_width

            extract = img[start_row:end_row, start_col:end_col]
            # extract = np.array(extract)
            fd = hog(extract, orientations=8, pixels_per_cell=(16, 16),
                     cells_per_block=(1, 1), multichannel=True)
            plot_bk.append(fd)
```

Figure 7. Code for Cropping Test Images

We used a similar for-loop as what we did to get training set in the first part to crop the new test images into (100×100) patches. Then, we extract their HOG features and append them into a matrix. Using SVM, we are able to predict and separate new patches and label them as either ‘1’ or ‘0’.

After we got a matrix that contains the ‘1’ and ‘0’ values we got from the SVM prediction, we run through all the patches on the images one more time and apply the predicted non-palm (100×100) patches with a grayscale filter. As a result, the features predicted as palm will be highlighted in color.

Figure 8. Relatively Successful Palm Segmentation Examples

3.3 Mismatches

After we implemented our image segmentation algorithm on our image data, we got some relatively successful palm segmentation. However, due to the small data size, our accuracy for palm detection is not perfect. From the mismatches, we concluded some difficulties we have on identifying palm trees.

a. Non-Palms Predicted as Palms

Shown in Figure 9, though we do successfully predicted most of the palm tree on the image, there are still large areas that misclassified non-palm features are a palm tree. Both plants in Figure 9 have patterns similar to that of the palm tree. Also, since some bigger palms have thicker stalks, white tree branches are often misclassified.

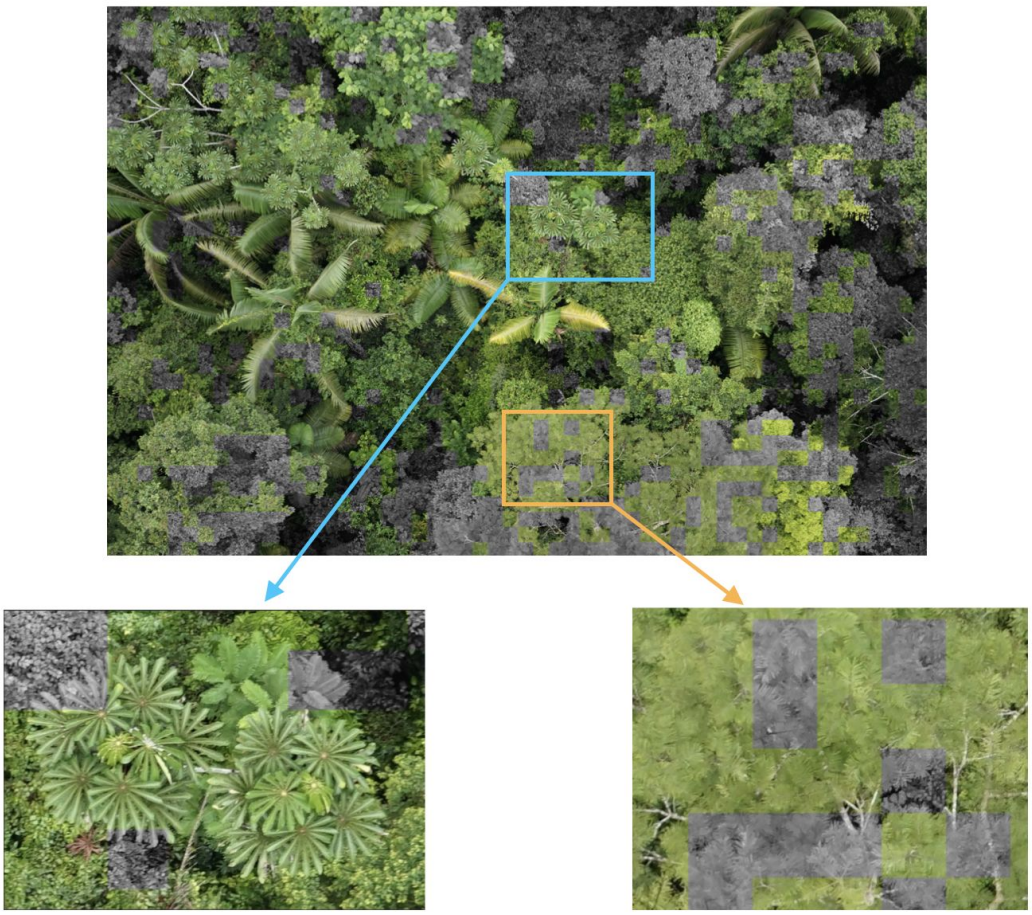

Figure 9. Confusing Non-Palm Species Mismatched

b. Neither Palms nor Non-Palms

Some patches that have neither palms nor non-palms in them, such as buildings, agricultural land, and dark areas, are sometimes classified as palms. This is due to their similar HOG features which have strong magnitudes for one direction. Shown on the left side below, Palm shows clear directions in its HOG, while for non-palm images, the magnitudes of gradients are about the same for every direction. Since buildings and agriculture lands exhibit more edges, their HOG features also contain strong magnitudes for one direction.

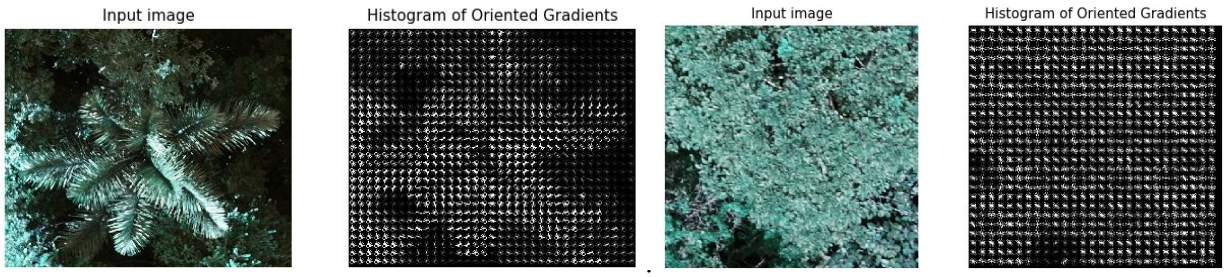

Figure 10. Left: HOG of Palm; Right: HOG of Non-Palm

Figure 11. Example of a Building Mismatched as Palm

4. Future Work

In this project, we designed an image segmentation algorithm to highlight potential palm trees from satellite images. The SVM prediction computes fast while having an acceptable accuracy. We concluded several perspectives to improve our image segmentation performance.

For the datasets, we can definitely increase the number of palm features in our training set, which will increase our model's complexity and potentially lead to higher accuracy in predicting palms. Moreover, our SVM model is a binary classifier that only predicts a feature to be either palm or not palm. Consequently, some plants have features that similar to that of the palm tree cannot be accurately classified. We can improve by constructing a multiclass classifier that gives labels to non-palm plants that often occurs. We also think that building a neural network that considers one feature for each different layer can improve the prediction accuracy.