Document version: 1.1 (2015-11-15)

Curtin University – Department of Computing

Assignment Cover Sheet / Declaration of Originality

Complete this form if/as directed by your unit coordinator, lecturer or the assignment specification.

Last name:	Mahmood	Student ID:	18747612
Other name(s):	Qaiser		
Unit name:	Machine Perception	Unit ID:	COMP3007
Lecturer / unit coordinator:	Senjian An	Tutor:	Sam
Date of submission:	11/10/2019	Which assignment?	(Leave blank if the unit has only one assignment.)

I declare that:

- The above information is complete and accurate.
- The work I am submitting is *entirely my own*, except where clearly indicated otherwise and correctly referenced.
- I have taken (and will continue to take) all reasonable steps to ensure my work is *not accessible* to any other students who may gain unfair advantage from it.
- I have *not previously submitted* this work for any other unit, whether at Curtin University or elsewhere, or for prior attempts at this unit, except where clearly indicated otherwise.

I understand that:

- Plagiarism and collusion are dishonest, and unfair to all other students.
- Detection of plagiarism and collusion may be done manually or by using tools (such as Turnitin).
- If I plagiarise or collude, I risk failing the unit with a grade of ANN ("Result Annulled due to Academic Misconduct"), which will remain permanently on my academic record. I also risk termination from my course and other penalties.
- Even with correct referencing, my submission will only be marked according to what I have done myself, specifically for this assessment. I cannot re-use the work of others, or my own previously submitted work, in order to fulfil the assessment requirements.
- It is my responsibility to ensure that my submission is complete, correct and not corrupted.

Signature: Qaiser Mahmood Date of signature: 11/10/2019

(By submitting this form, you indicate that you agree with all the above text.)

MACHINE PERCEPTION ASSIGNMENT

Qaiser Mahmood (18747612)

Contents

ASSIGNMENT SUMMARY

In this machine perception assignment, I was given two tasks. Both of the tasks were concerned with detection and recognition of building numbers of Bentley campus of Curtin University. In task 1 there was only one building number in the picture whereas in task 2 there were more than one building numbers with directional signage in the picture.

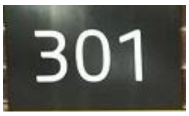
Initially I attempted to complete both of the tasks with template matching technique. For that I took one template and tried to find the matching area in the image. Because there were a lot of variations of size and lighting, I had to modify my template by changing the brightness, angle and size. Then I had to check the properties like black and white colour fractions, number of expected digits in the matched template to identify the correct template. This technique was a little bit computationally expensive but it did not generalize very well.

Then I did some research and thinking and decided to use connected component analyses. The connected component technique gave me good results in detecting the digits and directional signage in the pictures. Then I used histogram of oriented gradients (HOG) and support vector machine (SVM) to classify the digits. The detail of each step is given in the relevant parts of this report.

TASKS COMPLETED

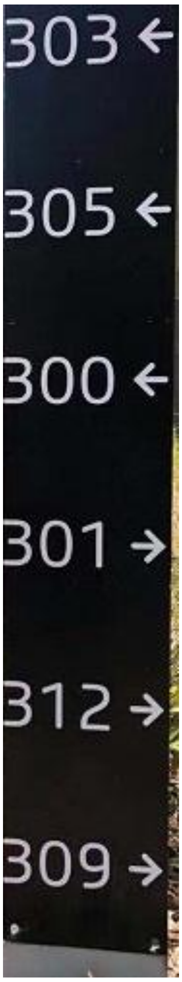
I completed both of the assignment tasks with fairly good accuracy within reasonable computation time. I used validation images provided with the assignment to test the detection and recognition accuracy. Below images show the detection and recognition accuracy for Task 1 and 2 respectively.

Task 1:

Detected regions in each image					
					
Recognition of digits in each image					
Building 202	Building 314	Building 301	Building 109	Building 206	Building 312

As you can see I got **100 %** accuracy in both detection and digit recognition in Task 1.

Task 2:

Detected regions in each image					
					
Recognition of digits in each image					
Bold entries are incorrect ones	Building 309 to the right Building 312 to the right Building 301 to the right Building 300 to the left Building 305 to the left **Building 503 to the left**	Building 101 to the right Building 109 to the right Building 201 to the right Building 208 to the left Building 209 to the left Building 210 to the left	Building 207 to the right Building 314 to the right Building 312 to the right **Building 19 to the left**	**Building 10 to the right** **Building 09 to the right** Building 216 to the right Building 500 to the left Building 212 to the left **Building 406**	**Building 01 to the right** **Building 99 to the right** Building 204 to the right Building 311 to the left Building 302 to the left

The detection accuracy = (5 correct / 6 total) * 100 = **83.33 %**
The recognition accuracy = (20 correct / 27 total) * 100 = **74.07 %**

TASK 1: Building signage detection and recognition

Description:

In this task, I was assigned to develop a program that reads in colour images from a specified directory. The provided images were from the Bentley campus of Curtin University.

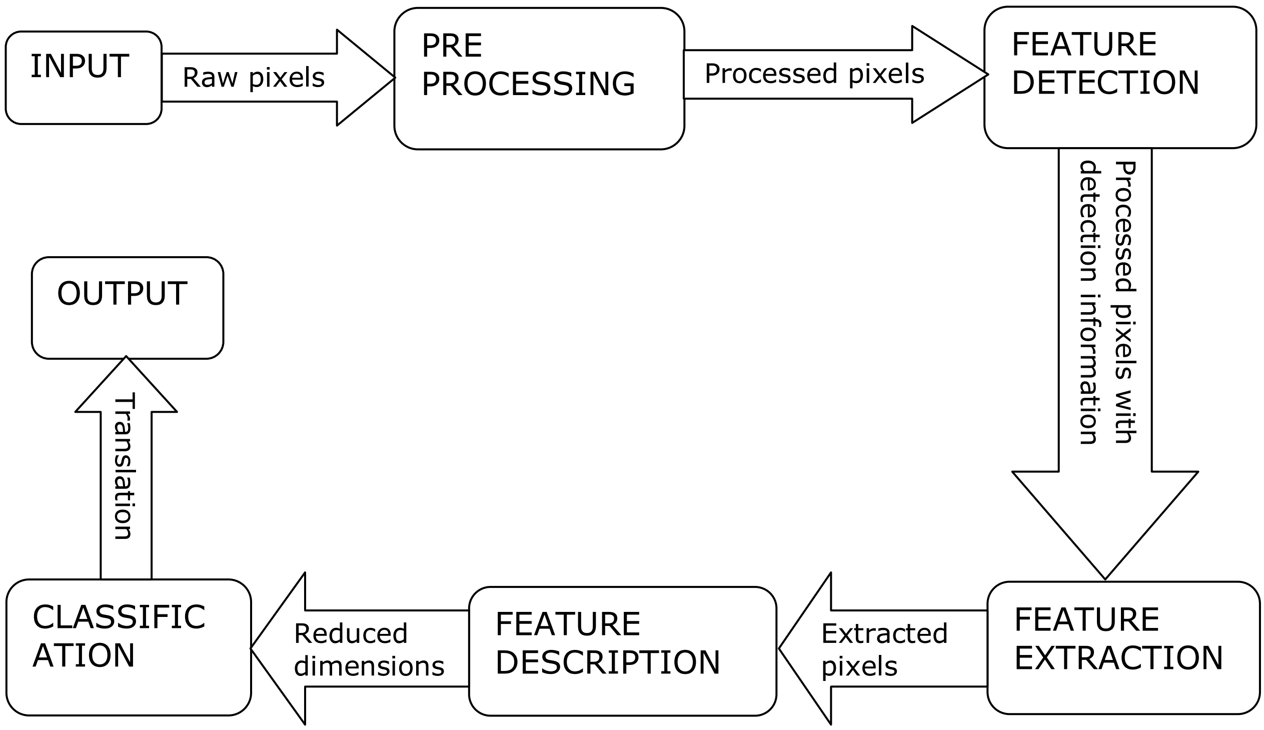
For each image, the program should be able to DETECT, EXTRACTE and CLASSIFY the digits.

The program should output the extracted part of the image and building number in the form of image and text files.

Approach:

Machine perception pipeline:

I followed the following machine perception pipeline to complete this task.



The detail (actions performed) of each step is described below.

Input:

Read the image files from specific directory using opencv.

Pre processing:

Opencv stores the image in BGR format. I applied following pre processing steps.

- Changed the color space from BGR to HSV which is better color space to separate information based on colors.
- Separated each channel of the HSV format.
- Using the V channel reduced the shadows from the image by applying a threshold of 85. All the values that were below 85 were changed to white.
- Applied median filter and erosion to remove little noise from the background. The number of iterations of erosion step was 3.

Feature detection:

The distinctive feature of the area that contains building numbers is the shape that is rectangle, size color of the rectangle. I performed following actions to detect the features from the image

- Using connected components, all the components were highlighted.
- Connected components were filtered based on horizontal thickness. I used **clean_labels** function that I developed. A Thresholding value of 15 was used.
- Applied clean_labels function on **transpose** of the image to filter connected components based on vertical thickness. The same Thresholding value of 15 was used.
- After filtering of small and unwanted shaped connected components applied Thresholding to change the image to binary.

Feature extraction:

I performed following actions to extract the features from the image

- I drew the filtered connected components from previous step on black background of the same size as the original image and applied the modified form of connected component detection to get more information like height, width of connected components.
- Calculated the area of each connected component.
- Removed the connected components that were within the range of 6000 to 22000 area and had the aspect ratio between 0.5 to 0.7

Feature description:

The extracted features had high dimensions. I performed following actions to reduce the dimensionality of the extracted features.

- I used contour detection to detect the individual numbers in the extracted region of the image.
- Based on contour area I extracted the bigger contours
- Sorted the big contours (based on y coordinate of the bounding rectangle of the contour) to keep track of the order of numbers. I used **contour_order** function that I developed.
- I extracted the ordered contours of digits on black background
- Resized the extracted digit image to the same size as my training images of digits
- Applied **HOG** to reduce the dimensionality

Classification:

I used support vector machine to classify the digits. I was provided with the images of digits and left and right arrows with different angles and orientations. The steps taken to train the model are given below:

- The most of the training images were 28 X 40 pixels but some of the images were 29 X 40 pixels. I resized all the images to 28 X 40.
- The images were pre processed with Thresholding.
- HOG was applied to reduce the dimensionality from 1120 to 64.
- I trained support vector machine model provided by opencv on HOG features. I tried various hyper parameters and chose the one that gave me best results.
- Saved the trained model in the same directory as the program to use it later without doing training again.
- Feed the feature descriptor to the trained model to predict the result.

Output:

The output was saved at two steps as per the requirements specified in the assignment specification.

- After feature extraction the region of interest of the image was saved in the specified folder.
- The output of the classifier was stored as text file in the specified directory.

TASK 2: Directional signage detection and recognition

Description:

In this task, I was assigned to develop a program that reads in colour images from a specified directory. The provided images were from the Bentley campus of Curtin University.

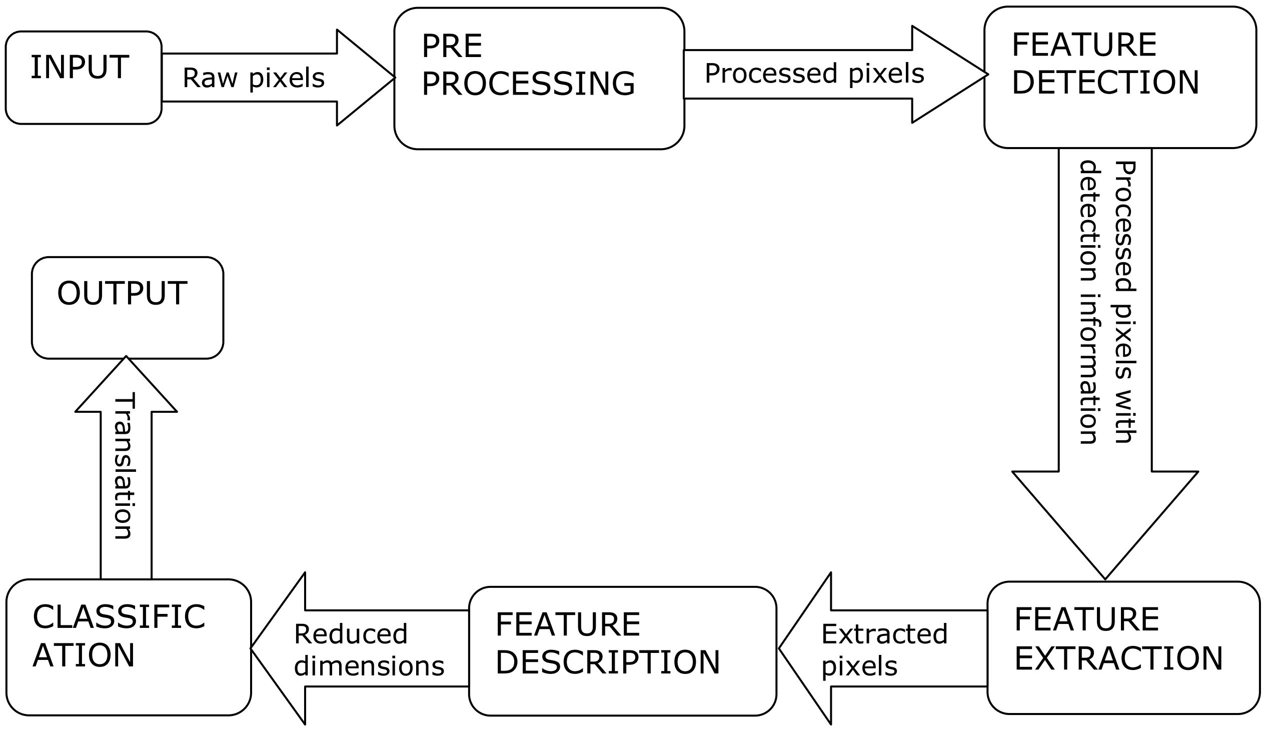
For each image, the program should be able to DETECT, EXTRACTE and CLASSIFY the digits as well as the directional sign in the image.

The program should output the extracted part of the image and building number in the form of image and text files.

Approach:

Machine perception pipeline:

I used the same machine perception pipeline as described in Task 1. However there were few changes in applying those steps.



The detail (actions performed) of each step is described below.

Input:

Read the image files from specific directory using opencv.

Pre processing:

Opencv stores the image in BGR format. I applied following pre processing steps.

- Changed the color space from BGR to HSV which is better color space to separate information based on colors.
- Separated each channel of the HSV format.
- Using the V channel reduced the shadows from the image by applying a threshold of 100. All the values that were below 100 were changed to white.
- Applied erosion to remove little noise from the background. The number of iterations of erosion step was 29. I did not apply median filter in this task as it was giving me poor results due to a lot of noise in the background of the images.

Feature detection:

The distinctive feature of the area that contains building numbers is the shape that is rectangle, size color of the rectangle. I performed following actions to detect the features from the image

- Using connected components, all the components were highlighted.
- Connected components were filtered based on vertical thickness. A threshold value of 200 was used. I used the same **clean_labels** function that was used in task 1 but this time it was applied on the transpose of image only. No filtering based on horizontal thickness was done.
- After filtering of small and unwanted shaped connected components applied Thresholding to change the image to binary.

Feature extraction:

I performed following actions to extract the features from the image

- I drew the filtered connected components from previous step on black background of the same size as the original image and applied the modified form of connected component detection to get more information like height, width of connected components.
- Calculated the area of each connected component.
- Removed the connected components that were greater than the area of 10000 and had the aspect ratio between 3 to 7
- The extracted part of the image was again processed in the function **cut_plate** that I developed to cut the big area of image into smaller parts such that each part contain only building number and directional sign.
- I used contour detection and aspect ratio of digits to cut the big number plate in smaller number plates.

Feature description:

This part was mostly the same as in task 1 but the parameters were different because this number plate contained an extra symbol for directional signage. I performed following actions to reduce the dimensionality of the extracted features.

- I used contour detection to detect the individual numbers in the extracted region of the image.
- Based on contour area I extracted the bigger contours
- Sorted the big contours (based on y coordinate of the bounding rectangle of the contour) to keep track of the order of numbers. I used **contour_order** function that I developed.
- I extracted the ordered contours of digits on black background
- Resized the extracted digit image to the same size as my training images of digits
- Applied **HOG** to reduce the dimensionality

Classification:

I used the support vector machine that was trained in task 1 to classify the digits. The steps taken to predict from the model are given below:

- Feed the feature descriptor to the trained model to predict the result.
- The prediction step was performed in loop to predict all the building numbers that were on the extracted region of interest.

Output:

The output was saved at two steps as per the requirements specified in the assignment specification.

- After feature extraction the region of interest of the image was saved in the specified folder.
- The output of the classifier was stored as text file in the specified directory.

Assignment Retrospective

What went well during the assignment?

It was very good learning experience. The assignment gave me the opportunity to apply the concepts learned during the lectures and tutorials.

What went wrong during the assignment?

Mostly this assignment went very well at good pace. No major issues / surprises happened. Initially I just got stuck with pattern matching technique. Although that gave me good opportunity to master that technique and I was able to get some reasonable accuracy. But that technique was not computationally efficient and failed to generalize well.

What could be done differently to improve?

Right now if my algorithm failed to detect the region of interest using connected component method it just give up. We can have multiple attempts to detect the region of interest with different techniques.

For example, if we cannot find the building number and directional signage with connected components then we can use template matching on that particular image to extract the desired part of image.

TASK 1 SOURCE CODE

TASK1 SOURCE CODE

```
import cv2
import os
import time
import numpy as np
from pathlib import Path


# This function draws the connected components on black background
def draw_label(lbls):
    # Map component labels to hue val
    label_hue = np.uint8(179*lbls/np.max(lbls))
    blank_ch = 255*np.ones_like(label_hue)
    labeled_img = cv2.merge([label_hue, blank_ch, blank_ch])

    # cvt to BGR for display
    labeled_img = cv2.cvtColor(labeled_img, cv2.COLOR_HSV2BGR)

    # set bg label to black
    labeled_img[label_hue == 0] = 0
    return labeled_img


# This function removes the connected components that have less horizontal thickness than a specified threshold
def clean_labels(lbls, count_threshold):
    unique_labels = np.unique(lbls)  # To calculate number of connected components in the parameter lbls
    for i in range(1, len(unique_labels)):
        current_label_indices = np.where(lbls == unique_labels[i])  # Grab the indices of one component
        current_label_row_indices = current_label_indices[0]  # Number of indices in one row
        current_label_col_indices = current_label_indices[1]  # Number of indices in one column

        # Unique indices of one row. This is basically indices of indices.
        unique_indices, indices_index, indices_count = np.unique(current_label_row_indices, return_index=True, return_counts=True)
        less_count_indices = indices_index[np.where(indices_count <= count_threshold)]  # Indices that are below threshold
        less_count_row_indices = current_label_row_indices[less_count_indices]  # Row indices that are below threshold
        less_count_row_indices_count = indices_count[np.where(indices_count <= count_threshold)]  # Number of row indices that are below threshold

        # This loop changes all the indices of one component that are below threshold to 0
        for j in range(len(less_count_row_indices)):
            row_index = np.where(current_label_row_indices == less_count_row_indices[j])[0][0]
            row_value = current_label_row_indices[row_index]
            col_value = current_label_col_indices[row_index]
            lbls[row_value][col_value:col_value+less_count_row_indices_count[j]] = 0
    return lbls


# This function is used to detect and localize the building signage
def locate_building_signage(im):
```

```
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)  # Change color space from BGR to HSV
    h, s, v = cv2.split(hsv)  # Split channels of the image
    h1 = h * 0
    s1 = s * 0
    v1 = v * 0

    v_indices = np.where(v <= 85)  # Indices of the V channel that are below 85
    v[v_indices] = v[v_indices] * 0 + 255  # Make them white. This will reduce the shadows in the image
    s[v_indices] = s[v_indices] * 0 + 255  # Make them white. This will reduce the shadows in the image

    # Make a copy of the indices that are below threshold of 85
    h1[v_indices] = h[v_indices]
    s1[v_indices] = s[v_indices]
    v1[v_indices] = v[v_indices]

    img2 = cv2.merge((h1, s1, v1))  # This image will have shadows reduced
    img2 = cv2.medianBlur(img2, 11)  # Remove the background noise

    img2 = cv2.erode(img2, (5, 5), iterations=3)  # Remove the background noise. In opencv erode works on background
    img2 = cv2.cvtColor(img2, cv2.COLOR_HSV2BGR)  # Change color from HSV to BGR. In opencv cannot convert from HSV to Gray
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)  # Change color from BGR to Gray

    ret, labels = cv2.connectedComponents(img2_gray)  # Find connected components
    labels = clean_labels(labels, 15)  # Remove the smaller and unwanted shaped connected components
    labels = labels.T  # Transpose of the image
    labels = clean_labels(labels, 15)  # Remove the smaller and unwanted shaped connected components
    labels = labels.T  # Change the image back to its original orientation
    lbl_img = draw_label(labels)  # Draw the final connected components on black background
    lbl_img_gray = cv2.cvtColor(lbl_img, cv2.COLOR_BGR2GRAY)  # Change color from BGR to Gray
    ret, thresh = cv2.threshold(lbl_img_gray, 20, 255, cv2.THRESH_BINARY)  # Change the image to Binary using a threshold of 20

    nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(thresh)  # Find the connected components with more information
    roi_list = list()
    for i in range(1, nlabels):  # background is at the 0 index. That's why, loop starts from 1
        top_left = (stats[i][0], stats[i][1])  # Top left corner x and y
        width, height = stats[i][2], stats[i][3]
        bot_right = (stats[i][0] + width, stats[i][1] + height)  # Bottom right corner x and y
        area = height*width
        if 0.7 >= height/width >= 0.5 and 22000 >= area >= 6000:  # Filter based on area and aspect ratio
            roi_list.append((top_left, bot_right))
    return roi_list


# This function changes the file name to numeric value which is used as the target value during the training of SVM
def get_label(fname):
    lbl_list = ['Zero', 'One', 'Two', 'Three', 'Four', 'Five', 'Six', 'Seven', 'Eight', 'Nine', 'Left', 'Right']
    for i in range(len(lbl_list)):
```

```
        if lbl_list[i] in fname:
            return i


# This function reads the digits and convert them into data set required for the training of SVM
def load_digits_data(p):
    digits_data = list()
    digits_target = list()

    list_of_folders = os.listdir(p)
    for folder in list_of_folders:
        folder = p.joinpath(folder)
        list_of_files = os.listdir(folder)
        for digit_file in list_of_files:
            digit_img = cv2.imread(str(folder.joinpath(digit_file)), cv2.IMREAD_GRAYSCALE)  # Read the image in Gray scale
            _, thresh = cv2.threshold(digit_img, 127, 255, cv2.THRESH_BINARY)  # Convert to binary
            hog_img = hog(thresh)  # Find the HOG
            img_label = get_label(digit_file)  # Convert the file name to numeric value
            digits_data.append(hog_img)
            digits_target.append(img_label)
    return np.array(digits_data, dtype='float32'), np.array(digits_target)  # Final data set


# This function finds the HOG
def hog(im):
    number_of_bins = 16
    gradient_x = cv2.Sobel(im, cv2.CV_32F, 1, 0)  # Horizontal gradient
    gradient_y = cv2.Sobel(im, cv2.CV_32F, 0, 1)  # Vertical gradient
    magnitude, angle = cv2.cartToPolar(gradient_x, gradient_y)  # Combine two gradients
    bins = np.int32(number_of_bins * angle / (2 * np.pi))  # Make 16 bins from 0 to 360 degree angle

    bin_cells = bins[:10, :10], bins[10:, :10], bins[:10, 10:], bins[10:, 10:]  # Image is divided into 4 squares
    magnitude_cells = magnitude[:10, :10], magnitude[10:, :10], magnitude[:10, 10:], magnitude[10:, 10:]  # Image is divided into 4 squares

    list_of_histogram = list()

    # This loop calculates the histogram of each big square of the image
    for bin_cell, magnitude_cell in zip(bin_cells, magnitude_cells):
        bin_count = np.bincount(bin_cell.ravel(), magnitude_cell.ravel(), number_of_bins)
        list_of_histogram.append(bin_count)

    hist = np.hstack(list_of_histogram)  # Change the histogram into 16 X 4 = 64 dimension vector
    return hist


def train_svm():
    training_digits_folder = Path.cwd().joinpath('../Digits/augmented')
```

```
    data, targets = load_digits_data(training_digits_folder)
    svm = cv2.ml.SVM_create()  # Create the model
    svm.setKernel(cv2.ml.SVM_INTER)  # Choose the filter
    svm.train(data, cv2.ml.ROW_SAMPLE, targets)  # Train the SVM using each Row of the data set as one sample
    svm.save('trained_svm.dat')  # Save the trained model in current directory


# This function recognizes the digits
def read_plate(plate):
    svm = cv2.ml.SVM_load('trained_svm.dat')  # Load the pre trained model

    plate_gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)  # Change color from BGR to Gray
    thresh = cv2.inRange(plate_gray, 150, 255)  # Change the image to Binary using a threshold of 150

    _, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  # Find contours of the digits
    hierarchy = hierarchy[0]

    big_contours = list()
    for cnt in zip(contours, hierarchy):
        if cnt[1][3] < 0 and cv2.contourArea(cnt[0]) > 100:  # outer contours
            big_contours.append(cnt[0])
    big_contours.sort(key=lambda c: contour_order(c, plate.shape[1]))  # Sort contours based on same y from left to right
    plate_str = ''

    # This loop cuts the each digit from the number plate, resizes it to the required dimensions and then feed it to SVM for translation
    for i in range(3):
        mask = np.zeros_like(plate_gray)  # Black background of same dimensions as the number plate
        cv2.drawContours(mask, big_contours, i, 255, -1)  # Draw the digit on black background
        out = np.zeros_like(plate_gray)  # Another black background of same dimensions as the number plate
        out[mask == 255] = plate_gray[mask == 255]  # Shift the mask to second black background
        (y, x) = np.where(mask == 255)  # Find the x and y coordinate of each white pixel
        (topy, topx) = (np.min(y), np.min(x))  # Find the minimum of both x and y. This is the top left corner of digit
        (bottomy, bottomx) = (np.max(y), np.max(x))  # Find the maximum of both x and y. This is the bottom right corner of digit
        out = out[topy-3:bottomy+3, topx-3:bottomx+3]  # Add padding of 3 pixels to the sides of the digit
        _, thresh = cv2.threshold(out, 127, 255, cv2.THRESH_BINARY)  # Change it to binary
        dgt_resized = cv2.resize(thresh, (28, 40))  # Resize the digit to required dimensions
        hog_im = hog(dgt_resized)  # Find the descriptor of less dimensions using histogram of oriented gradients
        dgt_resized = np.array(hog_im, dtype='float32').reshape(-1, 64)
        result = svm.predict(dgt_resized)[1].ravel()  # Translate the digit
        plate_str += str(int(result[0]))
    return plate_str


# Function arranges the contours from left to right that are approximately at same y value.
def contour_order(cnt, ncols):
    height_tolerance = 10
    x, y, _, _ = cv2.boundingRect(cnt)  # Find x and y coordinates of the bounding box of the contour
```

```
    same_height = (y // height_tolerance) * height_tolerance
    return same_height * ncols + x


if __name__ == '__main__':
    # Uncoment the below line if you want to train the new model
    # train_svm()
    t1 = time.time()
    program_folder = Path.cwd()  # Current directory of the program
    output_folder = program_folder.joinpath('output/task1')

    # For each run empty the task1 folder first in output directory
    for f in os.listdir(str(output_folder)):
        os.remove(str(output_folder.joinpath(f)))

    # Validation or Test images path
    images_folder = program_folder.joinpath('../test/task1')
    files = os.listdir(str(images_folder))

    list_of_plates = list()
    for f in files:
        print(f)
        img = cv2.imread(str(images_folder.joinpath(f)))  # Read the image
        img_copy = np.copy(img)
        list_of_roi = locate_building_signage(img)  # Find the coordinates of the region of interest that contains the building numbers
        if len(list_of_roi) != 0:
            for roi in list_of_roi:
                (x1, y1), (x2, y2) = roi[0], roi[1]
                num_plate = img_copy[y1: y2, x1: x2]  # Crop the part of the image that contains building number
                fn = 'DetectedArea' + f[-6:]  # Last 6 characters of the current image file name
                fn2 = 'Building' + f[-6:-4] + '.txt'  # Part of the file name that has number
                cv2.imwrite(str(output_folder.joinpath(fn)), num_plate)  # Write the image to task2 folder
                res = read_plate(num_plate)  # Translate each number plate
                text_file = open(str(output_folder.joinpath(fn2)), 'w')
                text_file.write('Building ' + res)  # Write the translated text file to task2 folder
                text_file.close()
    t2 = time.time()
    print('Processing Time: ', round((t2 - t1) * 100), 'ms')  # Total time for processing in milli seconds
```

TASK 2 SOURCE CODE

TASK2 SOURCE CODE

```python
import cv2
import os
import time
import numpy as np
from pathlib import Path
from task1 import hog
from task1 import contour_order
from task1 import draw_label
from task1 import clean_labels


# This function is used to detect and localize the building and directional signage
def locate_building_signage(im):
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)  # Change color space from BGR to HSV
    h, s, v = cv2.split(hsv)  # Split channels of the image
    h1 = h * 0
    s1 = s * 0
    v1 = v * 0

    v_indices = np.where(v <= 100)  # Indices of the V channel that are below 100
    v[v_indices] = v[v_indices] * 0 + 255  # Make them white. This will reduce the shadows in the image
    s[v_indices] = s[v_indices] * 0 + 255  # Make them white. This will reduce the shadows in the image

    # Make a copy of the indices that are below threshold of 100
    h1[v_indices] = h[v_indices]
    s1[v_indices] = s[v_indices]
    v1[v_indices] = v[v_indices]

    img2 = cv2.merge((h1, s1, v1))  # This image will have shadows reduced
    img2 = cv2.erode(img2, (5, 5), iterations=29)  # Remove the background noise. In opencv erode works on background
    img2 = cv2.cvtColor(img2, cv2.COLOR_HSV2BGR)  # Change color from HSV to BGR. In opencv cannot convert from HSV to Gray
    img2_gray = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)  # Change color from BGR to Gray
    ret, labels = cv2.connectedComponents(img2_gray)  # Find connected components
    labels = labels.T  # Transpose of the image
    labels = clean_labels(labels, 200)  # Remove the smaller and unwanted shaped connected components
    labels = labels.T  # Change the image back to its original orientation
    lbl_img = draw_label(labels)  # Draw the final connected components on black background
    lbl_img_gray = cv2.cvtColor(lbl_img, cv2.COLOR_BGR2GRAY)  # Change color from BGR to Gray
    ret, thresh = cv2.threshold(lbl_img_gray, 20, 255, cv2.THRESH_BINARY)  # Change the image to Binary using a threshold of 20

    nlabels, labels, stats, centroids = cv2.connectedComponentsWithStats(thresh)  # Find the connected components with more information
    roi_list = list()
    for i in range(1, nlabels):  # background is at the 0 index. That's why, loop starts from 1
        top_left = (stats[i][0], stats[i][1])  # Top left corner x and y
        width, height = stats[i][2], stats[i][3]
        bot_right = (stats[i][0] + width, stats[i][1] + height)  # Bottom right corner x and y
        area = height*width
```

```
        if 7.0 >= height/width >= 3.0 and area >= 10000:  # Filter based on area and aspect ratio
            roi_list.append((top_left, bot_right))
    return roi_list


# This function cut the bigger plate of numbers and returns a list of plates that contain each set of numbers
def cut_plate(plate):
    width = plate.shape[:2][1]
    plate_gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)  # Change color from BGR to Gray
    thresh = cv2.inRange(plate_gray, 90, 255)  # Change the image to Binary using a threshold of 90
    _, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  # Find contours to detect the digits
    hierarchy = hierarchy[0]
    list_of_plate = list()
    list_of_digit = list()
    for cmp in zip(contours, hierarchy):
        area = cv2.contourArea(cmp[0])
        if cmp[1][3] < 0 and 500 > area > 20:  # outer contours
            x, y, w, h = cv2.boundingRect(cmp[0])
            aspect_ratio = float(w)/h
            if 0.75 >= aspect_ratio >= 0.3 or 1.5 >= aspect_ratio >= 0.9:  # Filter contours based on aspect ration of digits and directional arrow
                list_of_digit.append((x, y, w, h))

    # This loop removes all the digits that are at approximately (+- 50) same y
    while len(list_of_digit) > 0:
        list_of_digit_at_same_height = list()
        dgt_y = list_of_digit[0][1]
        for dgt in list_of_digit:
            if (dgt_y - 50) <= dgt[1] <= (dgt_y + 50):  # Filter digits that are approximately at same y
                list_of_digit_at_same_height.append(dgt)

        # Crop the area of the bigger plate that contain the digits of approximately same y
        if len(list_of_digit_at_same_height) > 0:
            arr = np.array(list_of_digit_at_same_height)
            plate_x = np.min(arr[:, 0])
            plate_y = np.min(arr[:, 1])
            plate_height = np.max(arr[:, 3])
            p = plate[plate_y: plate_y + plate_height, plate_x: plate_x + width]  # Part of the image that have digits of same y
            list_of_plate.append(p)
        for dgt in list_of_digit_at_same_height:  # Remove the same y digits from the original list of digits
            list_of_digit.remove(dgt)
    return list_of_plate


# This function recognizes the digits
def read_plate(plate):
    svm = cv2.ml.SVM_load('trained_svm.dat')  # Load the pre trained model

```

```python
    plate_gray = cv2.cvtColor(plate, cv2.COLOR_BGR2GRAY)  # Change color from BGR to Gray
    thresh = cv2.inRange(plate_gray, 90, 255)  # Change the image to Binary using a threshold of 90
    _, contours, hierarchy = cv2.findContours(thresh, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)  # Find contours of the digits
    hierarchy = hierarchy[0]

    big_contours = list()
    for cnt in zip(contours, hierarchy):
        if cnt[1][3] < 0 and 500 > cv2.contourArea(cnt[0]) > 20:  # outer contours
            _, _, w, h = cv2.boundingRect(cnt[0])
            aspect_ratio = float(w)/h
            if 0.75 >= aspect_ratio >= 0.3 or 1.5 >= aspect_ratio >= 0.9:  # Filter contours based on aspect ration of digits and directional arrow
                big_contours.append(cnt[0])

    big_contours.sort(key=lambda c: contour_order(c, plate.shape[1]))  # Sort contours based on same y from left to right

    plate_str = ''

    # This loop cuts the each digit from the number plate, resizes it to the required dimensions and then feed it to SVM for translation
    for i in range(len(big_contours)):
        mask = np.zeros_like(plate_gray)  # Black background of same dimensions as the number plate
        cv2.drawContours(mask, big_contours, i, 255, -1)  # Draw the digit on black background
        out = np.zeros_like(plate_gray)  # Another black background of same dimensions as the number plate
        out[mask == 255] = plate_gray[mask == 255]  # Shift the mask to second black background

        (y, x) = np.where(mask == 255)  # Find the x and y coordinate of each white pixel
        (topy, topx) = (np.min(y), np.min(x))  # Find the minimum of both x and y. This is the top left corner of digit
        (bottomy, bottomx) = (np.max(y), np.max(x))  # Find the maximum of both x and y. This is the bottom right corner of digit
        out = out[topy:bottomy+3, topx:bottomx+3]  # Add padding of 3 pixels to the sides of the digit
        _, thresh = cv2.threshold(out, 90, 255, cv2.THRESH_BINARY)  # Change it to binary
        dgt_resized = cv2.resize(thresh, (28, 40))  # Resize the digit to required dimensions

        hog_im = hog(dgt_resized)  # Find the descriptor of less dimensions using histogram of oriented gradients
        dgt_resized = np.array(hog_im, dtype='float32').reshape(-1, 64)
        result = svm.predict(dgt_resized)[1].ravel()  # Translate the digit

        # Decoding of left and right directional arrow
        if result == 10:
            plate_str += ' to the left'
        elif result == 11:
            plate_str += ' to the right'
        else:
            plate_str += str(int(result[0]))
    return plate_str


if __name__ == '__main__':
    t1 = time.time()
```

```
    program_folder = Path.cwd()  # Current directory of the program
    output_folder = program_folder.joinpath('output/task2')

    # For each run empty the task2 folder first in output directory
    for f in os.listdir(str(output_folder)):
        os.remove(str(output_folder.joinpath(f)))

    # Validation or Test images path
    images_folder = program_folder.joinpath('../test/task2')
    files = os.listdir(str(images_folder))

    for f in files:
        print(f)
        img = cv2.imread(str(images_folder.joinpath(f)))  # Read the image
        img_copy = np.copy(img)
        list_of_roi = locate_building_signage(img)  # Find the coordinates of the region of interest that contains the building numbers
        if len(list_of_roi) != 0:
            for roi in list_of_roi:
                (x1, y1), (x2, y2) = roi[0], roi[1]
                num_plate = img_copy[y1: y2, x1: x2]  # Crop the part of the image that contains building numbers

                fn = 'DetectedArea' + f[-6:]  # Last 6 characters of the current image file name
                fn2 = 'BuildingList' + f[-6:-4] + '.txt'  # Part of the file name that has number
                cv2.imwrite(str(output_folder.joinpath(fn)), num_plate)  # Write the image to task2 folder

                list_of_plates = cut_plate(num_plate)  # Cut the bigger number plate into small number plates
                text_file = open(str(output_folder.joinpath(fn2)), 'a')
                for j in range(len(list_of_plates)):
                    res = read_plate(list_of_plates[j])  # Translate each number plate
                    text_file.write('Building ' + res + '\n')  # Write the translated text file to task2 folder
                text_file.close()
    t2 = time.time()
    print('Processing Time: ', round((t2 - t1) * 100), 'ms')  # Total time for processing in milli seconds
```