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:	WRIGHT	Student ID:	19779085
Other name(s):	JONATHAN		
Unit name:	Machine Perception	Unit ID:	COMP3007
Lecturer / unit coordinator:	Senjian	Tutor:	Sam Bray
Date of submission:	25/10/2020	Which assignment?	2 (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: JONATHAN WRIGHT Date of signature: 25/10/2020

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

Machine Perception Assignment 2

Jonathan Wright - 19779085
Curtin University
Computer Science

***Abstract*—Within this Assignment I investigated extracting building numbers using various Machine Perception techniques and used machine learning to interpret those images and make sense of the data.**

I. Supervised Learning - Digit Reading

A. Chosen Classifier

For this assignment I chose to use a **Support Vector Machine (SVM)** because in general they perform much faster for this type of data (low features and low dataset). According to the spec we are more concerned with speed vs accuracy so a drop in roughly 5 - 7% vs using a KNN Classifier is acceptable, hence I chose a SVM.

B. Chosen Feature Set

I chose to use Hog Feature detection as the OpenCV docs recommended it for OCR, I also considered using SIFT, however, it detects digits with a 93% accuracy so I felt no need to switch.

II. Digit Detection & Segmentation

For Digit Detection and Segmentation I have a series of steps, I will list them in order below.

A. Step 1 - Gaussian Blur

Before I begin trying to extract the images I attempt to blur out as much useless information I can i.e the background, I found through trial and error optimal values that work for most images to remove almost all useless detected features.

B. Step 2 - Canny Edge Detection

After blurring the image I use Canny Edge Detection to attempt to detect the edges of the plate or numbers, again through trial and error I found optimal values that work for most images to remove useless edges.

C. Step 3 - Image Morphological Transformations

To make certain features more detectable and remove some I used image morphs specifically I first closed the image alot to make the numbers appear bigger and I also dilated on y more than x to extend them so I could detect them better. Again trial and error was involved to find optimal values for dilating and closing.

D. Step 4 - Find All Contours In Detected Edges

Using OpenCV's findContours() method all edges were converted to contours, this will return all the edges areas and x ys.

E. Step 5 - Filtering

Various filters were used in this step, firstly a area filter is used i.e if the area is too small I disregard it as its likely not a number. Then I filter by how tall the numbers are or how wide, and I also filter by height width ratio and width height ratio. Then after filtering by those I group the contours into pairs, if pairs are found I select these over any other contours. Finally the contours are returned and this works for all validation images and only 4 training images failed testing.

III. Performance Of Program

A. Big O Notation Of Program

The algorithm I have written I believe is at most O(n^3) where n is the number of contours detected. This is not what I would call optimal but for this assignment I did not try to optimize the O notation of program as it completes very quickly already.

B. How Fast The Program Truly Runs (seconds)

Using Python's inbuilt Time.time() function the program completed training in roughly 0.23 seconds. It completed validation in roughly 0.16 seconds. It completed testing in 0.16 seconds (where testing included all validation + training images). **Due to this I can conclude that the program meets the minimum requirement of completing in under one minute.**

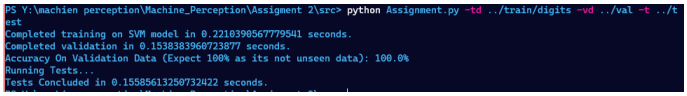

Fig. 1. Speed Of Program

IV. Inspiration or References

My SVM Model was made following the OpenCV Documentation, other then that I went into this assignment with no inspiration.

```
import argparse
import os
import cv2 as cv
from Training import trainDigits, validation, test
from Utils import makeFolder

'''
    Purpose: Entry point for program,
    -td trains digits
    -vd validates digits
    -t runs test.
    -d enables debugging
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("-td", '--train_digits',
                        help="This will train digits based off data in
                           ↪ directory (NOTE: it expects a certain
                           ↪ format as its hardcoded labels.).")
    parser.add_argument("-vd", '--validate_digits',
                        help="This will begin the validation phase in
                           ↪ given directory.")
    parser.add_argument(
        "-t", '--test', help="This will begin testing in given
           ↪ directory.", type=str)
    parser.add_argument(
        "-d", '--debug', help="Enable Debug Statements", default=False,
           ↪  type=bool)
    parser.add_argument("-o", "--output", default="output/")
    args = parser.parse_args()

    if args.output == "output/":
        makeFolder("output")
    makeFolder("{}/validation".format(args.output))

    if args.train_digits:
        trainDigits(args.train_digits)
    if args.validate_digits:
        try:
            validation(args.validate_digits, args.output)
        except Exception as e:
            print(e)
    if args.test:
        test(args.test, args.output, args.debug)

'''
    Purpose: Call main.
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
if __name__ == "__main__":
    main()
```

```
import os
import re
import time
import cv2 as cv
import numpy as np
from Utils import extractDigits

SZ = 40  # height of 40 px for each image.
bin_n = 16
affine_flags = cv.WARP_INVERSE_MAP | cv.INTER_LINEAR
# Chosen learning model: SVM (its fast.) Source: https://github.com/
    ↪ OSSpk/Handwritten-Digits-Classification-Using-KNN-
    ↪ Multiclass_Perceptron-SVM
# https://docs.opencv.org/master/dd/d3b/tutorial_py_svm_opencv.html
# 16 bins

'''
    Purpose: Deskew img for hog. (Credit to OpenCV docs)
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
def deskew(img):
    m = cv.moments(img)
    if abs(m['mu02']) < 1e-2:
        return img.copy()
    skew = m['mu11']/m['mu02']
    M = np.float32([[1, skew, -0.5*SZ*skew], [0, 1, 0]])
    img = cv.warpAffine(img, M, (SZ, SZ), flags=affine_flags)
    return img

'''
    Purpose: Run hog against img. (Credit to OpenCV docs)
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
def hog(img):
    gx = cv.Sobel(img, cv.CV_32F, 1, 0)
    gy = cv.Sobel(img, cv.CV_32F, 0, 1)
    mag, ang = cv.cartToPolar(gx, gy)
    bins = np.int32(bin_n*ang/(2*np.pi))    # quantizing binvalues in
        ↪ (0...16)
    bin_cells = bins[:10, :10], bins[10:, :10], bins[:10, 10:], bins
        ↪ [10:, 10:]
    mag_cells = mag[:10, :10], mag[10:, :10], mag[:10, 10:], mag[10:,
        ↪ 10:]
    hists = [np.bincount(b.ravel(), m.ravel(), bin_n)
             for b, m in zip(bin_cells, mag_cells)]
    hist = np.hstack(hists)     # hist is a 64 bit vector
    return hist

'''
    Purpose: Train using SVM model with a HOG Feature set.
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
```

```
'''
def trainDigits(trainDir):
    start = time.time()
    digits = {}
    for dirpath, dirnames, filenames in os.walk(trainDir):
        if len(filenames) > 0:
            digits[dirpath[-1]] = []  # [0 - 9] = ... ROWS
            for image in filenames:
                image = cv.imread(dirpath + "/" + image, 0)
                digits[dirpath[-1]].append(hog(deskew(image)))
    labels = []
    train_data = []
    for key, val in digits.items():
        for x in val:
            labels.append(int(key))
            train_data.append(x)
    labels = np.array(labels).reshape(-1, 1)
    train_data = np.float32(train_data).reshape(-1, 64)
    svm = cv.ml.SVM_create()
    svm.setKernel(cv.ml.SVM_LINEAR)
    svm.setType(cv.ml.SVM_C_SVC)
    svm.setC(2.67)
    svm.setGamma(5.383)
    svm.train(train_data, cv.ml.ROW_SAMPLE, labels)
    svm.save('digits.dat')
    end = time.time()
    print("Completed training on SVM model in {} seconds.".format(end -
      ↪  start))

'''
    Purpose: Run validation set (times + outputs)
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
def validation(validationDir, output):  # sort out the output name
  ↪ eventually.
    if not os.path.isfile("./digits.dat"):
        raise Exception("SVM Model needs to be trained.")
    svm = cv.ml.SVM_load('./digits.dat')
    for dirpath, dirnames, filenames in os.walk(validationDir):
        break
    valImages = [dirpath + '/' + filenames[i] for i in range(len(
      ↪ filenames))]
    actual = ["48", "35", "94", "302", "71", "26"]
    correct = 0
    start = time.time()
    for i in range(len(valImages)):
        digits = extractDigits(valImages[i])
        z = re.search("(\d+)(\.(.+))\Z", valImages[i])[1]
        cv.imwrite(
            '{}/validation/DetectedArea{}.jpg'.format(output, z),
              ↪ digits[1])
        with open("{}/validation/BoundingBox{}.txt".format(output, z),
          ↪ 'w') as f:
```

```
            f.write("{} x, {} y, {} w, {} h".format(str(digits[2][0]),
                ↪ str(
                    digits[2][1]), str(digits[2][2]), str(digits[2][3])))
        reading = ""
        for target in digits[0]:
            gray = cv.resize(cv.cvtColor(target, cv.COLOR_BGR2GRAY),
                ↪ (28, 40))
            prediction = int(svm.predict(np.float32(
                hog(deskew(gray))).reshape(1, -1))[1][0][0])
            reading += str(prediction)
        if reading == actual[i]:
            correct += 1
        with open("{}/validation/House{}.txt".format(output, z), 'w')
            ↪ as f:
            f.write("Building {}".format(reading))
    end = time.time()
    accuracy = str((float(correct) / float(len(actual))) * 100.0)+"%"
    print("Completed validation in {} seconds.\nAccuracy On Validation
        ↪ Data (Expect 100% as its not unseen data): {}".format(
        end - start, accuracy))

'''
    Purpose: Run test suite (Times + Outputsa)
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
def test(testDir, outputDir, debug):  # 28 x 40
    start = time.time()
    print('Running Tests...')
    if not os.path.isfile("./digits.dat"):
        raise Exception("SVM Model needs to be trained.")
    svm = cv.ml.SVM_load('./digits.dat')
    for dirpath, dirnames, filenames in os.walk(testDir):
        break
    images = [dirpath + '/' + filenames[i] for i in range(len(filenames
        ↪ ))]
    for image in images:
        digits = extractDigits(image)
        if len(digits) > 0:
            z = re.search("(\d+)(\.(.+))\Z", image)[1]
            cv.imwrite('{}/DetectedArea{}.jpg'.format(outputDir, z),
                ↪ digits[1])
            with open("{}/BoundingBox{}.txt".format(outputDir, z), 'w')
                ↪  as f:
                f.write("{},{},{},{}".format(str(digits[2][0]), str(
                    digits[2][1]), str(digits[2][2]), str(digits[2][3])
                        ↪ ))
            if debug:
                test = cv.rectangle(cv.imread(image), (digits[2][0],
                    ↪ digits[2][1]), (
                    digits[2][0]+digits[2][2], digits[2][1]+digits
                        ↪ [2][3]), (255, 0, 0))
                cv.imshow('test', test)
                cv.waitKey()
                cv.destroyAllWindows()
```

```
            reading = ""
            for target in digits[0]:
                gray = cv.resize(cv.cvtColor(
                    target, cv.COLOR_BGR2GRAY), (28, 40))
                prediction = int(svm.predict(np.float32(
                    hog(deskew(gray))).reshape(1, -1))[1][0][0])
                reading += str(prediction)
            with open("{}/House{}.txt".format(outputDir, z), 'w') as f:
                f.write("Building {}".format(reading))
    print('Tests Concluded in {} seconds.'.format(time.time() - start))
```

```
import cv2 as cv
import numpy as np
import os

'''
    Purpose: Make a folder, dont care if it fails.
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
def makeFolder(name):
    try:
        os.makedirs(name)
    except Exception as e:
        pass

'''
    Purpose: Performs canny and various blurs + filters to extract and
    ↪ return digits.
    Date: 24/10/2020
    Author: Jonathan Wright 19779085
'''
def extractDigits(path):
    img = cv.imread(path, cv.IMREAD_COLOR)
    gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    gray = cv.GaussianBlur(gray, (15, 17), 0)
    edges = cv.Canny(gray, 115, 160)
    edges = cv.morphologyEx(edges, cv.MORPH_CLOSE, np.ones((5, 5), np.
    ↪ uint8))
    edges = cv.dilate(edges, np.ones((2, 2), np.uint8))
    image, contours, hierarchy = cv.findContours(
        edges, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
    contours = [cnt for cnt in contours if cv.contourArea(cnt) > 150]
    acceptable = []

    for cnt in contours:
        if len(contours) > 1:
            x, y, w, h = cv.boundingRect(cnt)
            widthHeightRatio = w / h
            heightWidthRatio = h / w
            if w < (img.shape[1] * 0.95):
                if widthHeightRatio <= 1.5:
                    if heightWidthRatio <= 4.8:
                        acceptable.append(cnt)
        else:
            acceptable.append(cnt)
    if len(acceptable) == 1:  # Assume it is a plate and try looking
    ↪ inside
        x, y, w, h = cv.boundingRect(acceptable[0])
        target = img[y:y+h, x:x+w]
        gray = cv.cvtColor(target, cv.COLOR_BGR2GRAY)
        gray = cv.GaussianBlur(gray, (17, 15), 0)
        edges = cv.Canny(gray, 120, 175)
        edges = cv.morphologyEx(edges, cv.MORPH_CLOSE,
                                np.ones((5, 5), np.uint8))
```

```
edges = cv.dilate(edges, np.ones((2, 2), np.uint8))

image, contours, hierarchy = cv.findContours(
    edges, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

contours = [cnt for cnt in contours if cv.contourArea(cnt) >
   ↪ 100]
acceptable = []
for cnt in contours:
    if len(contours) > 1:
        x2, y2, w2, h2 = cv.boundingRect(cnt)
        widthHeightRatio = w2 / h2
        heightWidthRatio = h2 / w2
        if w2 < (w * 0.95):
            if h2 < (h * 0.95):
                if widthHeightRatio <= 1.5:
                    if heightWidthRatio <= 3.5:
                        acceptable.append(cnt)
    else:
        acceptable.append(cnt)

detected = []
for cnt in acceptable:
    x2, y2, w2, h2 = cv.boundingRect(cnt)
    yPos = y+y2
    xPos = x+x2
    detected.append(
        (
            (xPos, yPos),
            (w2, h2)
        )
    )
detected.sort(key=lambda x: x[0][0])
targets = []
total_width = 0
total_height = 0
if len(detected) == 0:
    return []
previous_x = detected[0][0][0]
highest_y = detected[0][0][1]
for cnt in detected:
    targets.append(
        (img[cnt[0][1]:cnt[0][1]+cnt[1][1], cnt[0]
             [0]:cnt[0][0]+cnt[1][0]])  # the target
    )
    total_width += cnt[1][0]
    total_height = cnt[1][1] if cnt[1][1] > total_height else
       ↪ total_height
    highest_y = cnt[0][1] if cnt[0][1] < highest_y else
       ↪ highest_y
plate = img[
    (cnt[0][1]-10):(cnt[0][1]-10)+total_height+10,
    (cnt[0][0]-10):(cnt[0][0]-10)+total_width+10
]
xStart = detected[0][0][0]
```

```
    xEnd = detected[-1][0][0]
    widthh = abs(xStart - xEnd) + detected[-1][1][0]
    yPlate = highest_y
    plate = img[yPlate:yPlate+total_height, xStart:xStart + widthh]
    # 0 list of sorted numbers.
    detected = [targets, plate, [xStart, yPlate, widthh,
       ↪ total_height]]
    return detected

if len(acceptable) == 0:  # just find a decent plate i guess
    for cnt in contours:
        x, y, w, h = cv.boundingRect(cnt)
        if w < (img.shape[1] * 0.95):
            if h < (img.shape[1] * 0.95):
                acceptable.append(cnt)
if len(acceptable) > 1:  # find pairs
    pairs = dict()
    for i in range(len(acceptable)):
        for k in range(i+1, len(acceptable)):
            if k < len(acceptable):
                x, y, w, h = cv.boundingRect(acceptable[i])
                x2, y2, w2, h2 = cv.boundingRect(acceptable[k])
                if abs(w - w2) <= 30 and abs(h - h2) <= 15 and abs(
                   ↪ y-y2) <= 15 and abs(x - x2) <= (image.shape[1]
                   ↪  * 0.30):
                    pairs[i] = acceptable[i]
                    pairs[k] = acceptable[k]
    if len(pairs) > 1:
        acceptable = list(pairs.values())

not_inside = []  # Check no numbers on same x, as i assume vertical
   ↪  signs will not occur, bad assumption but im doin git
for i in range(len(acceptable)):
    ni = False
    for k in range(i+1, len(acceptable)):
        if k < len(acceptable):
            x, y, w, h = cv.boundingRect(acceptable[i])
            x2, y2, w2, h2 = cv.boundingRect(acceptable[k])
            if x > x2 and x < x2 + w2:
                # keep the bigger one.
                a = w * h
                a2 = w2 * h2
                print(a)
                print(a2)
                if a < a2:
                    ni = True
    if ni == False:
        not_inside.append(acceptable[i])
acceptable = not_inside

contours = []
for cnt in acceptable:
    x, y, w, h = cv.boundingRect(cnt)
    contours.append(
        (
```

```
                (x, y),
                (w, h)
            )
        )

    contours.sort(key=lambda cnt: cnt[0][0])
    targets = []
    total_width = 0
    total_height = 0
    previous_x = contours[0][0][0]
    highest_y = contours[0][0][1]
    for cnt in contours:
        targets.append(
            (img[cnt[0][1]:cnt[0][1]+cnt[1][1], cnt[0]
                 [0]:cnt[0][0]+cnt[1][0]])  # the target
        )
        total_width += cnt[1][0] + abs(previous_x - cnt[0][0])
        total_height = cnt[1][1] if cnt[1][1] > total_height else
           ↪ total_height
        highest_y = cnt[0][1] if cnt[0][1] < highest_y else highest_y
        plate = img[
            (cnt[0][1]-10):(cnt[0][1]-10)+total_height+10,
            (cnt[0][0]-10):(cnt[0][0]-10)+total_width+10
        ]
        xStart = contours[0][0][0]
        xEnd = contours[-1][0][0] + contours[-1][1][0]
        widthh = abs(xEnd - xStart)
        yPlate = highest_y
        plate = img[yPlate:yPlate+total_height, xStart:xStart+widthh]

    # 0 list of sorted numbers.
    detected = [targets, plate, [xStart, yPlate, widthh, total_height]]
    return detected
```