



MACHINE PERCEPTION ASSIGNMENT 2

Shrenik Jobanputra

19128014

Contents

1.0	Printout of Source Code:.....	2
2.0	Summary	7
3.0	Method	8
3.1	Pipeline for the detection of the digits	8
3.2	Method Discussion.....	9
4.0	References	10

1.0 Printout of Source Code:

```
1  # Shrenik Jobanputra
2  # 19128814
3
4  import numpy as np
5  import cv2
6  import os
7
8
9  #The directory of where the digits are located
10 trainFeaturesList = []
11 trainFeaturesLabel = []
12 testFeaturesList = []
13 testFeaturesLabel = []
14
15 for digit in range(0,10):
16     label = digit
17     training_directory = "/home/student/Documents/Machine_Perception/Assignment_2_2020/Digits-202052/" + str(label) + "/"
18     for Index, filename in enumerate(os.listdir(training_directory)):
19
20         if filename.endswith(".jpg"):
21             training_digit_image = cv2.imread(training_directory + filename)
22             training_digit = cv2.cvtColor(training_digit_image, cv2.COLOR_BGR2GRAY)
23             winSize = (20,20)
24             blockSize = (10,10)
25             blockStride = (5,5)
26             cellSize = (10,10)
27             nbins = 20
28             derivAperture = 1
29             winSigma = -1.
30             histogramNormType = 0
31             L2hysThreshold = 0.2
32             gammaCorrection = 1
33             nlevels = 64
34             useSignedGradients = True
35
36             hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,histogramNormType,L2hysThreshold,gammaCorrection,nlevels, useSignedGradients)
37
38             descriptor = hog.compute(training_digit)
39
40             # first half are training data and the other half is testing
41             if index + 1 < len(os.listdir(training_directory))/2:
42                 trainFeaturesList.append(descriptor)
43                 trainFeaturesLabel.append(label)
44             else:
45                 testFeaturesList.append(descriptor)
46                 testFeaturesLabel.append(label)
47
48 # store features array into a numpy array
49 train = np.array(trainFeaturesList, np.float32) # USE ONLY FLOAT 32
50 test = np.array(testFeaturesList, np.float32)
51
52 trainLabels = np.array(trainFeaturesLabel, np.float32)
53 testLabels = np.array(testFeaturesLabel, np.float32)
54
55 #train = train.reshape((57,243,1))
56 #trainLabels = trainLabels.reshape((57,1))
57
58 print(train.shape)
59 print(trainLabels.shape)
60 #train using KNN
61
62 knn = cv2.ml.KNearest_create()
63 knn.train(train, cv2.ml.KNN_SAMPLE_T, trainLabels)
64 ret, result, neighbours, dist = knn.findNearest(test, k = 3)
65 #print(result)
66
67 # Now we check the accuracy of the classification
68 # For that, compare the result with the testLabels and check which are wrong
69
70 matches = result-testLabels
71 correct = np.count_nonzero(matches)
72 print(correct)
73 print(result.size)
74 accuracy = correct*100.0/result.size
75 print("accuracy " + str(accuracy))
76
77 #----- READING IMAGES STARTS -----
78 #
79 #
80 #
81 #-----
82
83 # The directory of where the images are located
84 img_dir = "/home/student/Documents/Machine_Perception/Assignment_2_2020/va1_updated/"
85
86 for index, filename in enumerate(os.listdir(img_dir)):
87     if filename.endswith(".jpg") and index != 2 and index != 3:
88         img = cv2.imread(img_dir + "va10" + str(index + 1) + ".jpg")
89         cv2.imshow("Original Image", img)
90         cv2.waitKey(0)
91
92         #CHANGE TO GRAYSCALE IMAGE
93         imggray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
94
95         #APPLY THE GAUSSIAN BLUR
96         blur = cv2.GaussianBlur( imggray, (5,5), 0 ) #img, ksize, sigmaX
97         modeThresh = [] # will be used to find correct Threshold Value
98         mode = 0 # will be used to find correct Threshold Value
99         indexNode = 0 # will be used to find the VALUE( 0 - 255 )
100
101
102         #----- find the perfect Threshold Value -----
103
104         rows, cols = blur.shape
105
106         for i in range (256):
107             modeThresh.append(0)
108
109         for i in range(rows):
110             for j in range(cols):
```

```

111         #print(blur[i,j])
112         if blur[i,j] < 120:
113             modeThresh[blur[i,j]] = modeThresh[blur[i,j]] + 1
114
115     for i in range(256):
116         if modeThresh[i] > mode:
117             mode = modeThresh[i]
118             indexMode = i
119     print(indexMode)
120     #-----
121
122     #APPLY IMAGE THRESHOLDING
123     thresh = cv2.threshold(blur,indexMode + 9 ,255, cv2.THRESH_BINARY)[1][1] gives the 2nd element
124
125     #edged = cv2.Canny(blur, 50 , 200, 255)
126     #cv2.imshow("edged",edged)
127     #cv2.waitKey(0)
128
129     # find the contours
130     #in2, cnts, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
131
132     #cv2.drawContours(blur, cnts, -1, (0,255,0),3)
133
134     cv2.imshow("Thresholded Image", thresh)
135     cv2.waitKey(0)
136
137     kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
138     thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
139
140     cv2.imshow("Thresholded Image with Morphology", thresh)
141     cv2.waitKey(0)
142
143
144     rows, cols = thresh.shape
145     print(rows, cols)
146
147     yT = 0 # will be used for cropping( Y top value )
148     yB = rows - 1 # will be used for cropping( Y bottom value )
149     xL = 0 # will be used for cropping(X left Value)
150     xR = cols - 1 # will be used for cropping(X right value)
151     reachedBlackT = 0 # used to see if it reached many black pixels
152     reachedBlackB = 0 # used to see if it reached many black pixels
153     reachedBlackL = 0 # used to see if it reached many black pixels
154     reachedBlackR = 0 # used to see if it reached many black pixels.
155     checkModeX = 0
156     checkModeY = 0
157     modeX = 0 # to get mode black pixels in x direction
158     modeY = 0 # to get mode black pixels in y direction
159     blackInRow = [] # array
160     blackInCol = [] # array
161
162     for i in range(cols):
163         blackInRow.append(0)
164     for i in range(rows):
165         blackInCol.append(0)
166
167     #----- find the modeX ( Actually the Average ) -----
168
169     for i in range(rows):
170         x = -1 # will be used to see how many black pixels there are in a row
171         for j in range(cols):
172             if thresh[i,j] == 0:
173                 x = x + 1 # so we disregard this row
174             if x > 0 :
175                 blackInRow[x] = blackInRow[x] + 1 # number of lines with the specific number of black
176                 #print("blackInRow:" + str(blackInRow[x]) + " x:" + str(x))
177
178     colsTemp = cols
179
180     for i in range(cols):
181         if blackInRow[i] != 0:
182             modeX = modeX + i
183         if blackInRow[i] == 0:
184             colsTemp = colsTemp - 1
185
186     modeX = modeX / colsTemp
187
188     # ----- find the modeY( Actually the average ) -----
189
190     for j in range(cols):
191         x = -1 # will be used to see how many black pixels there are in a col
192         for i in range(rows):
193             if thresh[i,j] == 0:
194                 x = x + 1 # so we disregard this col
195             if x > 0:
196                 blackInCol[x] = blackInCol[x] + 1
197
198     rowsTemp = rows
199
200     for i in range(rows):
201         if blackInCol[i] != 0:
202             modeY = modeY + i
203         if blackInCol[i] == 0:
204             rowsTemp = rowsTemp - 1
205
206     modeY = modeY / rowsTemp
207
208
209     # ----- Finding the Top Y Value-----
210
211     for i in range(rows):
212         x = -1 # will be used to see how many black pixels there are in a row
213         for j in range(cols):
214             if thresh[i,j] == 0:
215                 x = x + 1 # so we disregard this row
216             if x >= modeX and reachedBlackT == 0:
217                 reachedBlackT = 1 # we have reached the abundance of black so we stop
218                 #print(x)

```

```

219         if x < modeX and reachedBlackI == 0: # need to use while loop until it reaches abundance of black pixels
220             yf = yf + 1
221             #print(yf)
222
223     # ----- Finding the Bottom Y value -----
224     for i in range(rows):
225         x = -1 # will be used to see how many black pixels there are in a row
226         for j in range(cols):
227             if thresh[rows - 1 - i,j] == 0:
228                 x = x + 1 # so we disregard this col
229         if x > modeX and reachedBlackO == 0:
230             reachedBlackO = 1 # we have reached the abundance of black so we stop
231             #print(x)
232         if x < modeX and reachedBlackB == 0: # need to use while loop until it reaches abundance of black pixels
233             yb = yb - 1
234             #print(yb)
235
236     # ----- Finding the Left X Value -----
237     for j in range(cols):
238         x = -1 # will be used to see how many black pixels there are in a col
239         for i in range(rows):
240             if thresh[i,j] == 0:
241                 x = x + 1 # so we disregard this col
242         if x > modeY and reachedBlackL == 0:
243             reachedBlackL = 1 # we have reached the abundance of black so we stop
244             #print(x)
245         if x < modeY and reachedBlackI == 0: # need to use while loop until it reaches abundance of black pixels
246             xl = xl + 1
247             #print(xl)
248
249     # ----- Finding the Right X value -----
250     for j in range(cols):
251         x = -1 # will be used to see how many black pixels there are in a col
252         for i in range(rows):
253             if thresh[i,cols - 1 - j] == 0:
254                 x = x + 1 # so we disregard this col
255         if x > modeY and reachedBlackR == 0:
256             reachedBlackR = 1 # we have reached the abundance of black so we stop
257             #print(x)
258         if x < modeY and reachedBlackR == 0: # need to use while loop until it reaches abundance of black pixels
259             xr = xr - 1
260             #print(xr)
261
262     imgCropped = img[yf : yb, xl : xr]
263     cv2.imshow("Image cropped", imgCropped)
264     cv2.waitKey(0)
265
266     # these values will be used to store into text file for actual coordinates
267     yfActual = yf
268     xlActual = xl
269
270     cv2.destroyAllWindows()
271
272     # ----- This gives where the BLACK AREA IS!! -----
273
274     # ----- WE REPEAT THIS AGAIN! -----
275     #
276     # NOTE: BLACK means WHITE down here!!
277     # this is why there is 255 instead of 0
278     # TOO LAZY to change code and comments
279     #
280     # ----- REPEATING -----
281
282     #CHANGE TO GRAYSCALE IMAGE
283     imgGrayCropped = cv2.cvtColor(imgCropped, cv2.COLOR_BGR2GRAY)
284
285     #APPLY THE GAUSSIAN BLUR
286     blur = cv2.GaussianBlur(imgGrayCropped, (5,5), 0) #img, ksize, sigmaX
287
288     #APPLY IMAGE THRESHOLDING
289     thresh = cv2.threshold(blur,180,255, cv2.THRESH_BINARY)[1]#1 gives the 2nd element
290
291     cv2.imshow("Thresholded Image", thresh)
292     cv2.waitKey(0)
293
294     kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
295     thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)
296
297     cv2.imshow("Thresholded Image with Morphology", thresh)
298     cv2.waitKey(0)
299
300
301     rows, cols = thresh.shape
302     print(rows, cols)
303
304     yf = 0 # will be used for cropping( Y top value )
305     yb = rows - 1 # will be used for cropping( Y bottom value )
306     xl = 0 # will be used for cropping(X left Value)
307     xr = cols - 1 # will be used for cropping(X right value)
308     reachedBlackI = 0 # used to see if it reached many black pixels
309     reachedBlackO = 0 # used to see if it reached many black pixels
310     reachedBlackL = 0 # used to see if it reached many black pixels
311     reachedBlackR = 0 # used to see if it reached many black pixels
312     aveX = 0 # to get mode white pixels in x direction
313     aveY = 0 # to get mode white pixels in y direction
314     blackInRow = [] # array
315     blackInCol = [] # array
316
317     for i in range(cols):
318         blackInRow.append(0)
319     for i in range(rows):
320         blackInCol.append(0)
321
322     # ----- Find the modeX -----
323
324     for i in range(rows):
325         x = -1 # will be used to see how many black pixels there are in a row
326         for j in range(cols):

```

```

327         if thresh[i,j] == 255:
328             x = x + 1 # so we disregard this row
329         if x > 0:
330             blackInRow[x] = blackInRow[x] + 1
331
332     colsTemp = cols
333
334     for i in range(cols):
335         if blackInRow[i] != 0:
336             aveX = aveX + i
337         if blackInRow[i] == 0:
338             colsTemp = colsTemp
339
340     aveX = aveX / colsTemp
341
342     #----- find the modeY -----
343
344     for j in range(cols):
345         x = -1 # will be used to see how many black pixels there are in a col
346         for i in range(rows):
347             if thresh[i,j] == 255:
348                 x = x + 1 # so we disregard this col
349         if x > 0:
350             blackInCol[x] = blackInCol[x] + 1
351
352     rowsTemp = rows
353
354     for i in range(rows):
355         if blackInCol[i] != 0:
356             aveY = aveY + i
357         if blackInCol[i] == 0:
358             rowsTemp = rowsTemp
359     aveY = aveY / rowsTemp
360
361
362
363     #----- Finding the Top Y Value-----
364     for i in range(rows):
365         x = -1 # will be used to see how many black pixels there are in a row
366         for j in range(cols):
367             if thresh[i,j] == 255:
368                 x = x + 1 # so we disregard this row
369         if x >= aveX and reachedBlackI == 0:
370             reachedBlackI = 1 # we have reached the abundance of black so we stop
371         #print(x)
372         if x < aveX and reachedBlackI == 0: # need to use while loop until it reaches abundance of black pixels
373             yT = yT + 1
374         #print(yT)
375
376     #----- Finding the Bottom Y value -----
377     for i in range(rows):
378         x = -1 # will be used to see how many black pixels there are in a row
379         for j in range(cols):
380             if thresh[i,j] == 255:
381                 x = x + 1 # so we disregard this row
382         if x >= aveX and reachedBlackB == 0:
383             reachedBlackB = 1 # we have reached the abundance of black so we stop
384         #print(x)
385         if x < aveX and reachedBlackB == 0: # need to use while loop until it reaches abundance of black pixels
386             yB = yB + 1
387         #print(yB)
388
389     #----- Finding the Left X Value-----
390     for j in range(cols):
391         x = -1 # will be used to see how many black pixels there are in a col
392         for i in range(rows):
393             if thresh[i,j] == 255:
394                 x = x + 1 # so we disregard this col
395         if x >= aveY and reachedBlackL == 0:
396             reachedBlackL = 1 # we have reached the abundance of black so we stop
397         #print(x)
398         if x < aveY and reachedBlackL == 0: # need to use while loop until it reaches abundance of black pixels
399             xL = xL + 1
400         #print(xL)
401
402     #----- Finding the Right X value -----
403     for j in range(cols):
404         x = -1 # will be used to see how many black pixels there are in a col
405         for i in range(rows):
406             if thresh[i,cols - 1 - j] == 255:
407                 x = x + 1 # so we disregard this col
408         if x >= aveY and reachedBlackR == 0:
409             reachedBlackR = 1 # we have reached the abundance of black so we stop
410         #print(x)
411         if x < aveY and reachedBlackR == 0: # need to use while loop until it reaches abundance of black pixels
412             xR = xR + 1
413         #print(xR)
414
415     img_dir_save = "/home/student/Documents/Machine_Perception/Assignment_2_2020/"
416
417
418     imgCroppedCropped = imgCropped[yT : yB, xL : xR]
419     y = yfActual + yf
420     x = xfActual + xL
421     w = xR - xL
422     h = yB - yf
423
424     cv2.imshow("image cropped cropped", imgCroppedCropped)
425     cv2.imwrite(img_dir_save + "DetectedArea0" + str(index + 1) + ".jpg", imgCroppedCropped)
426
427     file = open(img_dir_save + "Boundingbox0" + str(index + 1) + ".txt", "w")
428     content = "(" + str(x) + "," + str(y) + "," + str(w) + "," + str(h) + ")"
429     file.write(content)
430     file.close()
431
432     #np.savetxt(img_dir_save + "Boundingbox05.txt", "(" + str(x) + "," + str(y) + "," + str(w) + "," + str(h) + ")", fmt="%d")
433     cv2.waitKey(0)
434     cv2.destroyAllWindows()
435

```

```

435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543

```

```

# ---- NOW WE SHOULD HAVE THE AREA OF WHERE THE DIGITS ARE -----
#----- WE REPEAT THIS AGAIN! -----
#
# NOTE: BLACK means WHITE down here!!
# THIS is why there is 255 instead of 0
# TOO LAZY to change code and comments
#
# ----- REPEATING -----
#
#CHANGE TO GRAYSCALE IMAGE
imgCroppedCropped = cv2.cvtColor(imgCroppedCropped, cv2.COLOR_BGR2GRAY)

#APPLY THE GAUSSIAN BLUR
blur = cv2.GaussianBlur( imgCroppedCropped, (3,3), 0 ) #img, ksize, sigmaX

#APPLY IMAGE THRESHOLDING
thresh = cv2.threshold(blur,150,255, cv2.THRESH_BINARY)[1][1] gives the 2nd element
cv2.imshow("Thresholded Image", thresh)
cv2.waitKey(0)

kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (5,5))
thresh = cv2.morphologyEx(thresh, cv2.MORPH_OPEN, kernel)

cv2.imshow("Thresholded Image with Morphology", thresh)
cv2.waitKey(0)

rows, cols = thresh.shape
print(rows, cols)

y1 = 0 # will be used for cropping( Y top value )
y0 = rows - 1 # will be used for cropping( Y bottom value )
x1 = 0 # will be used for cropping(X Left Value)
x0 = cols - 1 # will be used for cropping(X right value)
reachedBlackL = 0 # used to see if it reached many black pixels
reachedBlackR = 0 # used to see if it reached many black pixels
reachedBlackL = 0 # used to see if it reached many black pixels
reachedBlackR = 0 # used to see if it reached many black pixels
modeX = 0 # to get mode white pixels in x direction
modeY = 0 # to get mode white pixels in y direction
checkModeX = 0
checkModeY = 0
blackInRowCropped = [] # array
blackInColCropped = [] # array

for i in range(cols):
    blackInRowCropped.append(0)
for i in range(rows):
    blackInColCropped.append(0)

#----- find the modeY -----
for j in range(cols):
    x = -1 # will be used to see how many black pixels there are in a col
    for i in range(rows):
        if thresh[i,j] == 0:
            x = x + 1 # so we disregard this col
        if x > 0:
            blackInColCropped[x] = blackInColCropped[x] + 1

for i in range(rows):
    if blackInColCropped[i] > checkModeY:
        checkModeY = blackInColCropped[i]
    modeY = i
#print("MODEY" + str(modeY))

# ----- Finding the Left X Value -----
for j in range(cols):
    x = -1 # will be used to see how many black pixels there are in a col
    for i in range(rows):
        if thresh[i,j] == 0:
            x = x + 1 # so we disregard this col
        if x >= rows - 2 and reachedBlackL == 0:
            reachedBlackL = 1 # we have reached the abundance of black so we stop
        # print("x1" + str(x))
        if x < rows - 2 and reachedBlackL == 0: # need to use while loop until it reaches abundance of black pixels
            x1 = x1 + 1

# ----- Finding the Right X value -----
for j in range(cols):
    x = -1 # will be used to see how many black pixels there are in a col
    for i in range(rows):
        if thresh[i,cols - 1 - j] == 0:
            x = x + 1 # so we disregard this col
        if x >= rows - 2 and reachedBlackR == 0:
            reachedBlackR = 1 # we have reached the abundance of black so we stop
        # print(x)
        if x < rows - 2 and reachedBlackR == 0: # need to use while loop until it reaches abundance of black pixels
            xR = xR - 1
        # print(xR)

imgCroppedCroppedCroppedL = imgCroppedCropped[0 : rows-1, 0 : x1]
cv2.imshow("Image cropped cropped cropped Left", imgCroppedCroppedCroppedL)
cv2.waitKey(0)

imgCroppedCroppedCroppedR = imgCroppedCropped[0 : rows-1, xR : cols - 1 ]
cv2.imshow("Image cropped cropped cropped Right", imgCroppedCroppedCroppedR)
cv2.waitKey(0)

#imgCroppedCroppedCroppedL = cv2.resize(imgCroppedCroppedCroppedL, (40,20))
#imgCroppedCroppedCroppedR = cv2.resize(imgCroppedCroppedCroppedR, (40,20))

cv2.imwrite(img_dir_save + "ExtractedDigitLeft" + str(index + 1) + ".jpg", imgCroppedCroppedCroppedL)
cv2.imwrite(img_dir_save + "ExtractedDigitRight" + str(index + 1) + ".jpg", imgCroppedCroppedCroppedR)

```

```

544 #----- CLASSIFYING THE IMAGE NOW!!! -----
545
546 #----- Left Digit-----
547 testingFeatureList = []
548
549 imgCroppedCroppedCroppedL = cv2.resize(imgCroppedCroppedCroppedL, (40,20))
550 imgDigitL = cv2.cvtColor(imgCroppedCroppedCroppedL, cv2.COLOR_BGR2GRAY)
551 winSize = (20,20)
552 blockSize = (10,10)
553 blockStride = (5,5)
554 cellSize = (10,10)
555 nbins = 20
556 derivAperture = 1
557 winSigma = -1.
558 histogramNormType = 0
559 L2HysThreshold = 0.2
560 gammaCorrection = 1
561 nlevels = 64
562 useSignedGradients = True
563
564 hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,histogramNormType,L2HysThreshold,gammaCorrection,nlevels, useSignedGradients)
565
566 descriptor = hog.compute(imgDigitL)
567 testingFeatureList.append(descriptor)
568
569 # store features array into a numpy array
570 testingFeature = np.array(testingFeatureList, np.float32) # USE ONLY FLOAT 32
571
572 #test using KNN
573
574 ret,resultL,neighbours,dist = knn.FindNearest(testingFeature, k = 1)
575
576 # ----- Right Digit -----
577
578 testingFeatureList = []
579
580 imgCroppedCroppedCroppedR = cv2.resize(imgCroppedCroppedCroppedR, (40,20))
581 imgDigitR = cv2.cvtColor(imgCroppedCroppedCroppedR, cv2.COLOR_BGR2GRAY)
582 winSize = (20,20)
583 blockSize = (10,10)
584 blockStride = (5,5)
585 cellSize = (10,10)
586 nbins = 20
587 derivAperture = 1
588 winSigma = -1.
589 histogramNormType = 0
590 L2HysThreshold = 0.2
591 gammaCorrection = 1
592 nlevels = 64
593 useSignedGradients = True
594
595 hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,histogramNormType,L2HysThreshold,gammaCorrection,nlevels, useSignedGradients)
596
597 descriptor = hog.compute(imgDigitR)
598 testingFeatureList.append(descriptor)
599
600 # store features array into a numpy array
601 testingFeature = np.array(testingFeatureList, np.float32) # USE ONLY FLOAT 32
602
603 #test using KNN
604
605 ret,resultR,neighbours,dist = knn.FindNearest(testingFeature, k = 1)
606
607 #-----print the result!-----
608
609 #print(str(int(resultL)) + str(int(resultR)) )
610
611 file = open(img_dir.save + "House0" + str(index + 1) + ".txt", "w")
612 content = "Building " + str(int(resultL)) + str(int(resultR))
613 file.write(content)
614 file.close()
615 cv2.destroyAllWindows()
616

```

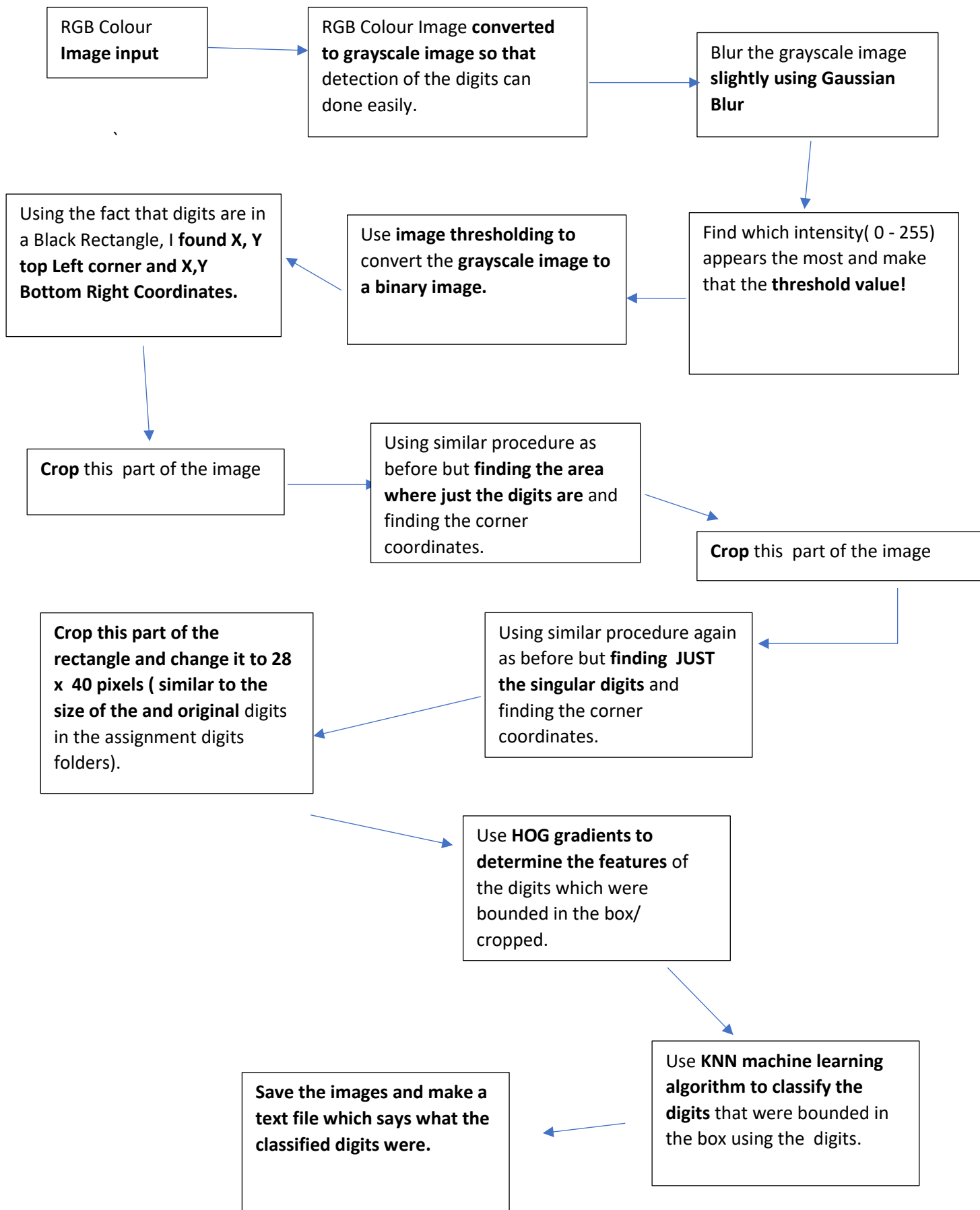
2.0 Summary

The Task has been attempted and has been completed, except for the fact that the house numbers extracted were not accurate. The digits on the photo can be detected and is Cropped out and is saved as a “.jpg” . The classification was also completed, so DetectedAreaX.jpg, ExtractedDigitsX.jpg, BoundingBoxX.txt and HouseX.txt were saved into the directory

To achieve the detection of the digits, several methods were incorporated such as Gaussian Blur, image Thresholding, morphology, Histogram of Oriented Gradients(HOG) and K – Nearest Neighbour algorithm(KNN).

3.0 Method

3.1 Pipeline for the detection of the digits



3.2 Method Discussion

The pipeline shown in section 3.1 will be explained further here.

My methods involved finding the abundance of black pixels and also of white pixels. To get from the original image to a cropped image (not the detected Area), I ran a “for loop” through each row in the thresholded image to find how many black pixels are in a row.

I repeated the same procedure with columns too. What this does is finds how many black pixels there were in each row and each column. Now the row and column where there is the MOST AMOUNT of black pixels becomes the border of the image(the black rectangle where the digits are).

The image with the black rectangle and digit was

cropped and once again, the “for loop” was run. But in this case, it was to find the AVERAGE Number of white pixels in each row and column. Using this, we can find the borders of where the area of the detected digits stay.

Figure 1 and Figure 2 both represent successful image of the DetectedArea!

Figure 3 however does not show a proper result, and this I believe was due to incorrect value chosen for thresholding.

After finding the DetectetArea, a functon was run to find an abundance of black pixels down a column. Whenever the black abundance is close to the actual size of down the column – which is the middle split between the two digits. So, whenever the black Abundance is decided, we split the image into left and right portions.

Image Classification was through HOG and KNN. Using HOG, I obtained a feature vector for the the digits and passed it into KNN training classifier. Using which, the accuracy was 88%. After, for each Test Image(after the cropping and singling the digits), the HOG vector is once again found and this is passed into KNN findNearest function with $K = 1$. This results in a number being outputted which corresponds to the digit.

The performance of the Detected Area detection was very good for most images.

The overlap of the detected area with the ground truth contained at least 50% of the detected area and contain at least 50% of the ground truth area which was a success.

After, the extraction of the individual digits was also very good for most images and there was very little margin of



FIGURE 2 : image of building 71 with Detected and Extracted Area



FIGURE 1 : image of building 26 with Detected and Extracted Area

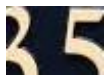


FIGURE 3 : image of building 35 with Detected and Extracted Area.

Although, this one is not proper.

useless bordering pixels(so lack of black and the border was touching the digit), which is good.

However, the digit recognition using the classifier was very poor. For the validation images, it got 2 digits correct, which thus has more room for improvement.

4.0 References

- Medium. 2020. *Optical Character Recognizer Using Knn And Opencv ! Part2..* [online] Available at: <<https://medium.com/@sudhirjain01dec/optical-character-recognizer-using-knn-and-opencv-part2-57637649079c>> [Accessed 30 October 2020].
- Medium. 2020. *Scanned Numbers Recognition Using K-Nearest Neighbor (K-NN).* [online] Available at: <<https://towardsdatascience.com/scanned-digits-recognition-using-k-nearest-neighbor-k-nn-d1a1528f0dea>> [Accessed 30 October 2020].