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:

```
# Shronik Jobanputra
# 19128814

import numpy as np
import cv2
import os


#The directory of where the digits are located
trainFeaturesList = []
trainFeaturesLabel = []
testFeaturesList = []
testFeaturesLabel = []

for digit in range(0,10):
    label = digit
    training_directory = "/home/student/Documents/Machine_Perception/Assignment_2_2020/Digits-2020S2/" + str(label) + "/"
    for index, filename in enumerate(os.listdir(training_directory)):

        if filename.endswith(".jpg"):
            training_digit_image = cv2.imread(training_directory + filename)
            training_digit = cv2.cvtColor(training_digit_image, cv2.COLOR_BGR2GRAY)
            winSize = (20,20)
            blockSize = (10,10)
            blockStride = (5,5)
            cellSize = (10,10)
            nbins = 20
            derivAperture = 1
            winSigma = -1.
            histogramNormType = 0
            L2HysThreshold = 0.2
            gammaCorrection = 1
            nlevels = 64
            useSignedGradients = True

            hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,histogramNormType,L2HysThreshold,gammaCorrection,nlevels, useSignedGradients)

            descriptor = hog.compute(training_digit)

            # first half are training data and the other half is testing
            if index + 1 < len(os.listdir(training_directory))/2:
                trainFeaturesList.append(descriptor)
                trainFeaturesLabel.append(label)
            else:
                testFeaturesList.append(descriptor)
                testFeaturesLabel.append(label)

# store features array into a numpy array
train = np.array(trainFeaturesList, np.float32) # USE ONLY FLOAT 32
test = np.array(testFeaturesList, np.float32)

trainLabels = np.array(trainFeaturesLabel, np.float32)
testLabels = np.array(testFeaturesLabel, np.float32)

#train = train.reshape((57,243,1))
#trainLabels = trainLabels.reshape((57,1))

print(train.shape)
print(trainLabels.shape)
#train using KNN

knn = cv2.ml.KNearest_create()
knn.train(train,  cv2.ml.ROW_SAMPLE, trainLabels)
ret, result, neighbours, dist = knn.findNearest(test, k = 3)
#print(result)

# Now we check the accuray of the classification
# For that, compare the result with the testLabels and check which are wrong

matches = result==testLabels
correct = np.count_nonzero(matches)
print(correct)
print(result.size)
accuracy = correct*100.0/result.size
print("accuracy " + str(accuracy))

#----------------- READING IMAGES STARTS --------------------------------
#
#
#
#-------------------------------------------------------------------------

# The directory of where the images are located
img_dir = "/home/student/Documents/Machine_Perception/Assignment_2_2020/val_updated/"

for index, filename in enumerate(os.listdir(img_dir)):
    if filename.endswith(".jpg") and index != 2 and index != 3:
        img = cv2.imread(img_dir + "val0" + str(index + 1) + ".jpg")
        cv2.imshow("Original Image", img)
        cv2.waitKey(0)

        #CHANGE TO GRAYSCALE IMAGE
        imgray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        #APPLY THE GAUSSIAN BLUR
        blur = cv2.GaussianBlur( imgray, (5,5), 0 ) #img, ksize, sigmaX
        modeThresh = [] # will be used to find correct Threshold Value
        mode = 0 # will be used to find correct Threshold Value
        indexMode = 0 # will be used to find the VALUE( 0 - 255 )


        #------------- find the perfect Threshold Value ---------------------------

        rows, cols = blur.shape

        for i in range (256):
            modeThresh.append(0)

        for i in range(rows):
            for j in range(cols):
```

```
            #print(blur[i,j])
            if blur[i,j] < 120:
                modeThresh[blur[i,j]] = modeThresh[blur[i,j]] + 1

    for i in range(256):
        if modeThresh[i] > mode:
            mode = modeThresh[i]
            indexMode = i
    print(indexMode)
    #--------------------------------------------------------------

    #APPLY IMAGE THRESHOLDING
    thresh = cv2.threshold(blur,indexMode + 9 ,255, cv2.THRESH_BINARY)[1]#[1] gives the 2nd element

    #edged = cv2.Canny(blur, 50 , 200, 255)
    #cv2.imshow("edged",edged)
    #cv2.waitKey(0)

    # find the contours
    #im2, cnts, hierarchy = cv2.findContours(edged, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    #cv2.drawContours(blur, cnts, -1, (0,255,0),3)

    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)

    yT = 0 # will be used for cropping( Y top value )
    yB = rows - 1 # will be used for cropping ( Y bottom  value )
    xL = 0 # will be used for cropping(X left Value)
    xR = cols - 1 # will be used for cropping(X right value)
    reachedBlackT = 0 # used to see if it reached many black pixels
    reachedBlackB = 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
    checkModeX = 0
    checkModeY = 0
    modeX = 0 # to get mode black pixels in x direction
    modeY = 0 # to get mode black pixels in y direction
    blackInRow = [] # array
    blackInCol = [] # array

    for i in range(cols):
        blackInRow.append(0)
    for i in range(rows):
        blackInCol.append(0)

    #-------------- find the modeX ( Actually the Average ) ------------------------------

    for i in range(rows):
        x = -1 # will be used to see how many black pixels there are in a row
        for j in range(cols):
            if thresh[i,j] == 0:
                x = x + 1 # so we disregard this row
        if x > 0 :
            blackInRow[x] = blackInRow[x] + 1 # number of lines with the specific number of black
        #print("blackInrow:" +str(blackInRow[x]) + " X:" + str(x))

    colsTemp = cols

    for i in range(cols):
        if blackInRow[i] != 0:
            modeX = modeX + i
        if blackInRow[i] == 0:
            colsTemp = colsTemp - 1

    modeX = modeX / colsTemp

    #-------------- find the modeY( Actually the average ) ------------------------------

    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:
            blackInCol[x] = blackInCol[x] + 1

    rowsTemp = rows

    for i in range(rows):
        if blackInCol[i] != 0:
            modeY = modeY + i
        if blackInCol[i] == 0:
            rowsTemp = rowsTemp - 1

    modeY = modeY / rowsTemp



    #-----------------Finding the Top Y Value----------------------
    for i in range(rows):
        x = -1 # will be used to see how many black pixels there are in a row
        for j in range(cols):
            if thresh[i,j] == 0:
                x = x + 1 # so we disregard this row
        if x >= modeX and reachedBlackT == 0:
            reachedBlackT = 1 # we have reached the abundance of black so we stop
        #print(x)
```

```
            if x < modeX and reachedBlackT == 0: # need to use while loop until it reaches abundunce of black pixels
                yT = yT + 1
                #print(yt)

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

        # ----------------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 >= modeY and reachedBlackL == 0:
                reachedBlackL = 1 # we have reached the abundance of black so we stop
            #print(x)
            if x < modeY and reachedBlackL == 0: # need to use while loop until it reaches abundunce of black pixels
                xL = xL + 1
                #print(xL)

        # ------------ 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 >= modeY and reachedBlackR == 0:
                reachedBlackR = 1 # we have reached the abundance of black so we stop
            #print(x)
            if x < modeY and reachedBlackR == 0: # need to use while loop until it reaches abundunce of black pixels
                xR = xR - 1
                #print(xR)

        imgCropped = img[yT : yB, xL : xR]
        cv2.imshow("Image cropped", imgCropped)
        cv2.waitKey(0)

        # These values will be used to store into text file for actual coordinates
        yTActual = yT
        xLActual = xL

        cv2.destroyAllWindows()

        #---------------------- This gives where the BLACK AREA IS!!! ------------------

        # -------- 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
        imgrayCropped = cv2.cvtColor(imgCropped, cv2.COLOR_BGR2GRAY)

        #APPLY THE GAUSSIAN BLUR
        blur = cv2.GaussianBlur( imgrayCropped, (5,5), 0 ) #img, ksize, sigmaX

        #APPLY IMAGE THRESHOLDING
        thresh = cv2.threshold(blur,180,255, 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)

        yT = 0 # will be used for cropping( Y top value )
        yB = rows - 1 # will be used for cropping ( Y bottom  value )
        xL = 0 # will be used for cropping(X left Value)
        xR = cols - 1 # will be used for cropping(X right value)
        reachedBlackT = 0 # used to see if it reached many black pixels
        reachedBlackB = 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
        aveX = 0 # to get mode white pixels in x direction
        aveY = 0 # to get mode white pixels in y direction
        blackInRow = [] # array
        blackInCol = [] # array

        for i in range(cols):
            blackInRow.append(0)
        for i in range(rows):
            blackInCol.append(0)

        # ------------ find the modeX ------------------------------

        for i in range(rows):
            x = -1 # will be used to see how many black pixels there are in a row
            for j in range(cols):
```

```
            if thresh[i,j] == 255:
                x = x + 1 # so we disregard this row
        if x > 0:
            blackInRow[x] = blackInRow[x] + 1

    colsTemp = cols

    for i in range(cols):
        if blackInRow[i] != 0:
            aveX = aveX + i
        if blackInRow[i] == 0:
            colsTemp = colsTemp

    aveX = aveX / colsTemp

    #------------ 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] == 255:
                x = x + 1 # so we disregard this col
        if x > 0:
            blackInCol[x] = blackInCol[x] + 1

    rowsTemp = rows

    for i in range(rows):
        if blackInCol[i] != 0:
            aveY = aveY + i
        if blackInCol[i] == 0:
            rowsTemp = rowsTemp
    aveY = aveY / rowsTemp



    # ----------------Finding the Top Y Value----------------------
    for i in range(rows):
        x = -1 # will be used to see how many black pixels there are in a row
        for j in range(cols):
            if thresh[i,j] == 255:
                x = x + 1 # so we disregard this row
        if x >= aveX and reachedBlackI == 0:
            reachedBlackT = 1 # we have reached the abundance of black so we stop
        #print(x)
        if x < aveX and reachedBlackT == 0: # need to use while loop until it reaches abundunce of black pixels
            yT = yT + 1
            #print(yT)

    #             Finding the Bottom Y value ----------------------
    for i in range(rows):
        x = -1 # will be used to see how many black pixels there are in a row
        for j in range(cols):
            if thresh[rows - 1 - i,j] == 255:
                x = x + 1 # so we disregard this row
        if x >= aveX and reachedBlackB == 0:
            reachedBlackB = 1 # we have reached the abundance of black so we stop
        #print(x)
        if x < aveX and reachedBlackB == 0: # need to use while loop until it reaches abundunce of black pixels
            yB = yB - 1
            #print(yB)

    # ---------------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] == 255:
                x = x + 1 # so we disregard this col
        if x >= aveY and reachedBlackL == 0:
            reachedBlackL = 1 # we have reached the abundance of black so we stop
        #print(x)
        if x < aveY and reachedBlackL == 0: # need to use while loop until it reaches abundunce of black pixels
            xL = xL + 1
            #print(xL)

    # ------------ 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] == 255:
                x = x + 1 # so we disregard this col
        if x >= aveY and reachedBlackR == 0:
            reachedBlackR = 1 # we have reached the abundance of black so we stop
        #print(x)
        if x < aveY and reachedBlackR == 0: # need to use while loop until it reaches abundunce of black pixels
            xR = xR - 1
            #print(xR)

    img_dir_save = "/home/student/Documents/Machine_Perception/Assignment_2_2020/"


    imgCroppedCropped = imgCropped[yT : yB, xL : xR]
    y = y1Actual + yT
    x = x1Actual + xL
    w = xR - xL
    h = yB - yT

    cv2.imshow("image cropped cropped", imgCroppedCropped)
    cv2.imwrite(img_dir_save + "DetectedArea0" + str(index + 1) + ".jpg", imgCroppedCropped)

    file = open(img_dir_save + "BoundingBox0"  + str(index + 1) + ".txt" , "w+")
    content =  "(" + str(x) + "," + str(y) + "," + str(w) + ","+ str(h) + ")"
    file.write(content)
    file.close()

    #np.savetxt(img_dir_save + "BoundingBox05.txt", "(" + str(x) + "," + str(y) + "," + str(w) + ","+ str(h) + ")", fmt="%s")
    cv2.waitKey(0)
    cv2.destroyAllWindows()
```

```
# ----  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
imgrayCroppedCropped = cv2.cvtColor(imgCroppedCropped, cv2.COLOR_BGR2GRAY)

#APPLY THE GAUSSIAN BLUR
blur = cv2.GaussianBlur( imgrayCroppedCropped, (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)

yT = 0 # will be used for cropping( Y top value )
yB = rows - 1 # will be used for cropping ( Y bottom  value )
xL = 0 # will be used for cropping(X Left Value)
xR = cols - 1 # will be used for cropping(X right value)
reachedBlackT = 0 # used to see if it reached many black pixels
reachedBlackB = 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("lolol" + str(x))
    if x < rows - 2 and reachedBlackL == 0: # need to use while loop until it reaches abundance of black pixels
        xL = xL + 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 : xL]
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, (40,28))
#imgCroppedCroppedCroppedR = cv2.resize(imgCroppedCroppedCroppedR, (40,28))

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

```
#------- CLASSIFYING THE IMAGE NOW!!! -----------------------

#------------------ Left Digit----------------------
testingFeatureList = []

imgCroppedCroppedCroppedL = cv2.resize(imgCroppedCroppedCroppedL, (40,28))
imgDigitL = cv2.cvtColor(imgCroppedCroppedCroppedL, cv2.COLOR_BGR2GRAY)
winSize = (20,20)
blockSize = (10,10)
blockStride = (5,5)
cellSize = (10,10)
nbins = 20
derivAperture = 1
winSigma = -1.
histogramNormType = 0
L2HysThreshold = 0.2
gammaCorrection = 1
nlevels = 64
useSignedGradients = True

hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,histogramNormType,L2HysThreshold,gammaCorrection,nlevels, useSignedGradients)

descriptor = hog.compute(imgDigitL)
testingFeatureList.append(descriptor)

        # store features array into a numpy array
testingFeature = np.array(testingFeatureList, np.float32) # USE ONLY FLOAT 32

#test using KNN

ret,resultL,neighbours,dist = knn.findNearest(testingFeature, k = 1)

# ---------------------- Right Digit -------------------------

testingFeatureList = []

imgCroppedCroppedCroppedR = cv2.resize(imgCroppedCroppedCroppedR, (40,28))
imgDigitL = cv2.cvtColor(imgCroppedCroppedCroppedR, cv2.COLOR_BGR2GRAY)
winSize = (20,20)
blockSize = (10,10)
blockStride = (5,5)
cellSize = (10,10)
nbins = 20
derivAperture = 1
winSigma = -1.
histogramNormType = 0
L2HysThreshold = 0.2
gammaCorrection = 1
nlevels = 64
useSignedGradients = True

hog = cv2.HOGDescriptor(winSize,blockSize,blockStride,cellSize,nbins,derivAperture,winSigma,histogramNormType,L2HysThreshold,gammaCorrection,nlevels, useSignedGradients)
```

```
descriptor = hog.compute(imgDigitL)
testingFeatureList.append(descriptor)

        # store features array into a numpy array
testingFeature = np.array(testingFeatureList, np.float32) # USE ONLY FLOAT 32

#test using KNN

ret,resultR,neighbours,dist = knn.findNearest(testingFeature, k = 1)

#---------------print the result!-------------------------

#print(str(int(resultL)) + str(int(resultR)) )

file = open(img_dir_save + "House0" + str(index + 1) + ".txt", "w+")
content = "Building " + str(int(resultL)) + str(int(resultR))
file.write(content)
file.close()
cv2.destroyAllWindows()
```

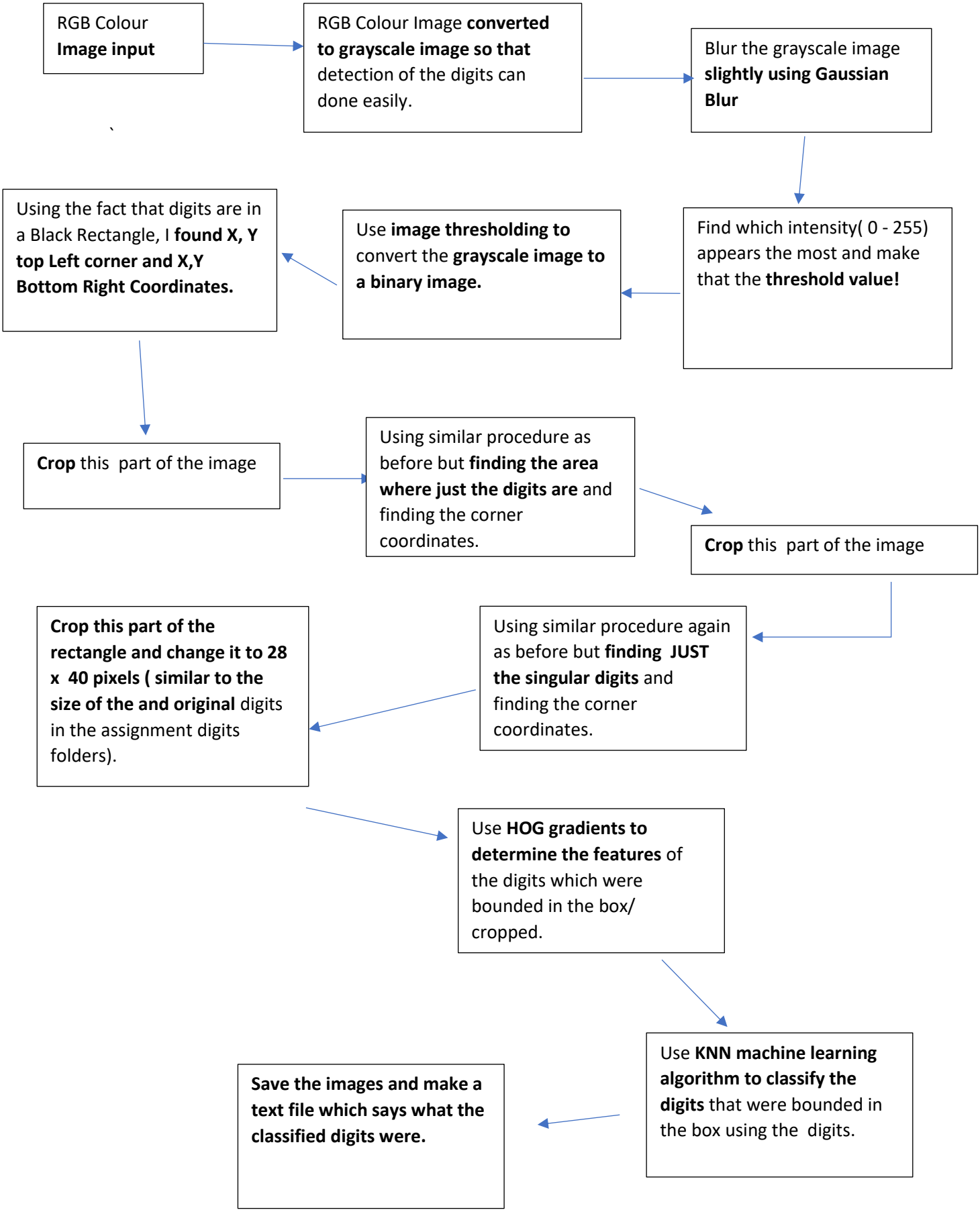
2.0 Summary

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).

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

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

The image with the black rectanlge 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 funciton 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 outputed 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.

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

Although, this one is not proper.

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

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].