**OCR ของข้อมูลที่เขียนด้วยมือโดยใช้ kNN**

เป้าหมาย

ในบทนี้

* เราจะใช้ความรู้ของเราเกี่ยวกับ kNN เพื่อสร้างแอ็พพลิเคชันพื้นฐาน OCR
* เราจะลองใช้ข้อมูล Digits และ Alphabets ที่มาพร้อมกับ OpenCV

OCR ของตัวเลขที่เขียนด้วยมือ

เป้าหมายของเราคือการสร้างแอปพลิเคชันที่สามารถอ่านตัวเลขที่เขียนด้วยลายมือ สำหรับข้อมูลนี้เราจำเป็นต้องมีข้อมูล train\_data และ test\_data OpenCV มาพร้อมกับภาพdigits.png (ในโฟลเดอร์opencv / samples / python2 / data / ) ซึ่งมีตัวเลขที่เขียนด้วยลายมือ 5000 (500 ตัวต่อหนึ่งหลัก) แต่ละหลักเป็นภาพ 20x20 ขั้นตอนแรกของเราคือการแบ่งภาพนี้เป็น 5000 หลักที่ต่างกัน สำหรับแต่ละหลักเราจะแบนลงในแถวเดียวด้วย 400 พิกเซล นั่นคือชุดคุณลักษณะของเราเช่นค่าความเข้มของพิกเซลทั้งหมด เป็นชุดคุณลักษณะที่ง่ายที่สุดที่เราสามารถสร้างได้ เราใช้ตัวอย่างแรกของแต่ละหมายเลข 250 เป็น train\_data และอีก 250 ตัวอย่างเป็น test\_data ลองมาเตรียมตัวก่อน

**import** **numpy** **as** **np**

**import** **cv2**

**from** **matplotlib** **import** pyplot **as** plt

img = cv2.imread('digits.png')

gray = cv2.cvtColor(img,cv2.COLOR\_BGR2GRAY)

*# Now we split the image to 5000 cells, each 20x20 size*

cells = [np.hsplit(row,100) **for** row **in** np.vsplit(gray,50)]

*# Make it into a Numpy array. It size will be (50,100,20,20)*

x = np.array(cells)

*# Now we prepare train\_data and test\_data.*

train = x[:,:50].reshape(-1,400).astype(np.float32) *# Size = (2500,400)*

test = x[:,50:100].reshape(-1,400).astype(np.float32) *# Size = (2500,400)*

*# Create labels for train and test data*

k = np.arange(10)

train\_labels = np.repeat(k,250)[:,np.newaxis]

test\_labels = train\_labels.copy()

*# Initiate kNN, train the data, then test it with test data for k=1*

knn = cv2.KNearest()

knn.train(train,train\_labels)

ret,result,neighbours,dist = knn.find\_nearest(test,k=5)

*# Now we check the accuracy of classification*

*# For that, compare the result with test\_labels and check which are wrong*

matches = result==test\_labels

correct = np.count\_nonzero(matches)

accuracy = correct\*100.0/result.size

**print** accuracy

แอ็พพลิเคชัน OCR พื้นฐานของเราพร้อมแล้ว ตัวอย่างนี้ทำให้ฉันมีความถูกต้อง 91% หนึ่งทางเลือกในการปรับปรุงความถูกต้องคือการเพิ่มข้อมูลสำหรับการฝึกอบรมโดยเฉพาะอย่างยิ่งคนที่ไม่ถูกต้อง ดังนั้นแทนที่จะหาข้อมูลการฝึกอบรมนี้ทุกครั้งที่เริ่มใช้แอพพลิเคชั่นผมควรจะเก็บข้อมูลไว้เพื่อให้ในครั้งต่อไปผมอ่านข้อมูลนี้จากแฟ้มและเริ่มการจำแนก คุณสามารถทำได้ด้วยความช่วยเหลือของฟังก์ชัน Numpy บางอย่างเช่น np.savetxt, np.savez, np.load เป็นต้นโปรดตรวจสอบเอกสารของรายละเอียดเพิ่มเติม

*# save the data*

np.savez('knn\_data.npz',train=train, train\_labels=train\_labels)

*# Now load the data*

**with** np.load('knn\_data.npz') **as** data:

**print** data.files

train = data['train']

train\_labels = data['train\_labels']

ในระบบของฉันจะใช้เวลาประมาณ 4.4 MB ของหน่วยความจำ เนื่องจากเราใช้ค่าความเข้ม (uint8 data) เป็นคุณสมบัติจึงควรแปลงข้อมูลเป็น np.uint8 ก่อนแล้วจึงบันทึก ใช้เวลาเพียง 1.1 MB ในกรณีนี้ ขณะโหลดแล้วคุณสามารถแปลงกลับเป็น float32 ได้

OCR ของตัวอักษรภาษาอังกฤษ

ถัดไปเราจะทำเช่นเดียวกันสำหรับตัวอักษรภาษาอังกฤษ แต่มีการเปลี่ยนแปลงเล็กน้อยในข้อมูลและชุดคุณลักษณะ ที่นี่แทนภาพ, OpenCV มาพร้อมกับแฟ้มข้อมูล, letter-recognition.dataในOpenCV / ตัวอย่าง / CPP /โฟลเดอร์ หากเปิดดูคุณจะเห็นเส้น 20000 เส้นซึ่งอาจเห็นได้ชัดตั้งแต่แรกเห็นเหมือนขยะ จริงๆแล้วในแต่ละแถวคอลัมน์แรกเป็นตัวอักษรที่เป็นป้ายกำกับของเรา ตัวเลขถัดไป 16 ตัวต่อไปนี้เป็นคุณลักษณะที่แตกต่างกัน คุณสมบัติเหล่านี้จะได้รับจาก[UCI เครื่องเรียนรู้](http://archive.ics.uci.edu/ml/)Repository คุณสามารถดูรายละเอียดของคุณสมบัติเหล่านี้ได้ใน[หน้า](http://archive.ics.uci.edu/ml/datasets/Letter+Recognition)นี้

มีตัวอย่างจำนวน 20000 ชุดดังนั้นเราจึงใช้ข้อมูล 10000 ครั้งแรกเป็นตัวอย่างการฝึกอบรมและเหลือ 10000 เป็นตัวอย่างทดสอบ เราควรเปลี่ยนตัวอักษรเป็นอักขระ ASCII เพราะเราไม่สามารถใช้งานตัวอักษรได้โดยตรง

**import** **cv2**

**import** **numpy** **as** **np**

**import** **matplotlib.pyplot** **as** **plt**

*# Load the data, converters convert the letter to a number*

data= np.loadtxt('letter-recognition.data', dtype= 'float32', delimiter = ',',

converters= {0: **lambda** ch: ord(ch)-ord('A')})

*# split the data to two, 10000 each for train and test*

train, test = np.vsplit(data,2)

*# split trainData and testData to features and responses*

responses, trainData = np.hsplit(train,[1])

labels, testData = np.hsplit(test,[1])

*# Initiate the kNN, classify, measure accuracy.*

knn = cv2.KNearest()

knn.train(trainData, responses)

ret, result, neighbours, dist = knn.find\_nearest(testData, k=5)

correct = np.count\_nonzero(result == labels)

accuracy = correct\*100.0/10000

**print** accurac

มันทำให้ฉันถูกต้อง 93.22% อีกครั้งถ้าคุณต้องการเพิ่มความถูกต้องคุณสามารถเพิ่มข้อมูลข้อผิดพลาดซ้ำในแต่ละระดับได้