Report
on

Image Recognition
12-Jan-2019

Submitted by

Love Prakash

DronaMaps Private Limited

Contents

1 Introduction to Image Recognition: **1**

2 Understanding Pixel Arrays: **1**

3 Graphing our images in Matplotlib: **2**

4 Thresholding: **3**

5 Thresholding function: **4**

6 Types of Loss functions: **5**

6.1 Mean Squared Error: 5

7 Thresholding Logic: **6**

8 Saving our data for training and testing: **8**

9 Basic Testing: **9**

10 Conclusion: **11**

1 Introduction to Image Recognition:

There are many applications for image recognition. One of the largest that people are most familiar with would be facial recognition, which is the art[1] of matching faces in pictures to identities. Image recognition goes much further, however. It can allow computers to translate written text on paper into digital text, it can help the field of machine vision, where robots and other devices can recognize people and objects.

Here, our goal is to begin to use machine learning, in the form of pattern recognition, to teach our program what text looks like. In this case, we'll use numbers, but this could translate to all letters of the [1]alphabet, words, faces, really anything at all. The more complex the image, the more complex the code will need to become. When it comes to letters and characters, it is relatively simplistic, however.

2 Understanding Pixel Arrays:

Now we're ready to dig into what makes an image in numbers. For this, we use PIL or Pillow, depending on what you were able to install (depending on your Python bit-version). Use whichever import you need to use, depending on your bit-version. I use PIL because I am using a 64-bit version of Python. We are then just importing numpy as np. Numpy is used for number-crunching. First we open the image using our image processor. Then we are saving the NumPy array version to iar, then outputting to console. Here is a 3-dimensional array of the data. All of the data is the image, each matrix block is a row of data, and each element within that is the pixel values in RGB-A (Red Green Blue Alpha). Here's a picture that should help:

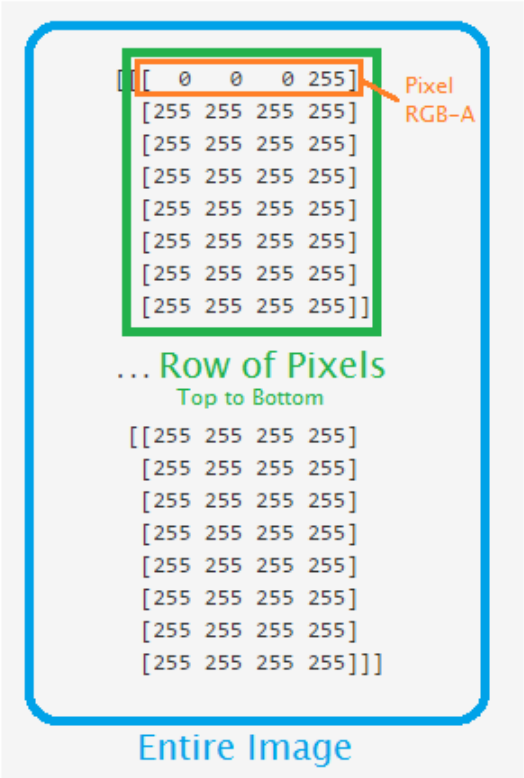

Figure 1: Entire Image

```
from PIL import Image
import numpy as np
i = Image.open('images/1DJI_0003.jpg')

iar = np.asarray(i)

print(iar)
```

3 Graphing our images in Matplotlib:

Now we're going to cover how to see the image array visually. Obviously, we start with images that we can already see, but soon we're going to start manipulating the images a bit. To see our changes, we could save to an image and then look at it, but it will be quicker and easier to display it via our program.

For this, we're going to use Matplotlib.

With some additional ending code, you can show the image:

```
from PIL import Image
import numpy as np
####
```

```
import matplotlib.pyplot as plt
###
i = Image.open('images/1DJI_0003.jpg')

iar = np.asarray(i)


plt.imshow(iar)
print(iar)
plt.show()
```

```
In [5]: i=Image.open('images/1DJI_0003.jpg')

In [6]: var=np.asarray(i)

In [7]: print(var)
[[[115  80  42]
  [119  83  49]
  [117  80  51]
  ...
  [114 148 186]
  [112 147 185]
  [110 148 187]]

 [[128  91  64]
  [127  89  66]
  [125  88  69]
  ...
  [117 149 188]
  [117 150 191]
  [117 152 192]]

 [[132  94  81]
  [128  90  79]
  [127  90  81]
  ...
  [123 155 194]
  [121 155 193]
  [119 152 193]]

 ...

 [[ 96 107 125]
  [ 96 107 125]
  [ 93 104 122]
```

Figure 2: matrix output

4 Thresholding:

The idea of thresholding is to simplify the image. Some people particularly like the visual effect as well, but we're interested in the simplifying aspect. An issue arises when we're

trying to identify characters, shapes, objects, whatever, because there is a massive list of colors. Anything complex, to be analyzed, needs to be broken down to the most basic parts. With thresholding, we can look at an image, analyze the "average" color, and turn that "average" into the threshold between white or black. Some thresholding wont go all of the way to either full black or white, there will be some gradient, but, for our basic purposes, we want to go all of the way!

Let's see an example of an image that we want to threshold.

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt

i = Image.open('images/1DJI_0003.jpg')

iar = np.asarray(i)


plt.imshow(iar)
print(iar)
plt.show()
```

5 Thresholding function:

Now what we want to do is create a function that will take the images we feed it, and threshold it. The way we're going to do this is by taking the "average" color value, and then thresholding any pixel as black if it is any darker or white if it is lighter. We define the function, and we specify that we're expecting a parameter, the imageArray. This will be that array that we've been seeing with the pixel values.

Next, we define the balanceAr as an empty list, and the newAr, for now, is the imageArray. We cannot modify the actual iar without NumPy throwing a fit, so we do this. Balance array, at the end, will be averaged, to find our threshold.

```
i=Image.open('images/1DJI_0004.jpg')
var=np.array(i)

i=Image.open('images/1DJI_0005.jpg')
var2=np.array(i)
i=Image.open('images/1DJI_0006.jpg')
var3=np.array(i)
i=Image.open('images/1DJI_0007.jpg')
var4=np.array(i)
fig = plt.figure()
ax1 = plt.subplot2grid((8,6),(0,0), rowspan=4, colspan=3)
ax2 = plt.subplot2grid((8,6),(4,0), rowspan=4, colspan=3)
ax3 = plt.subplot2grid((8,6),(0,3), rowspan=4, colspan=3)
ax4 = plt.subplot2grid((8,6),(4,3), rowspan=4, colspan=3)
```

```
ax1.imshow(var)
ax2.imshow(var2)
ax3.imshow(var3)
ax4.imshow(var4)

plt.show()
```

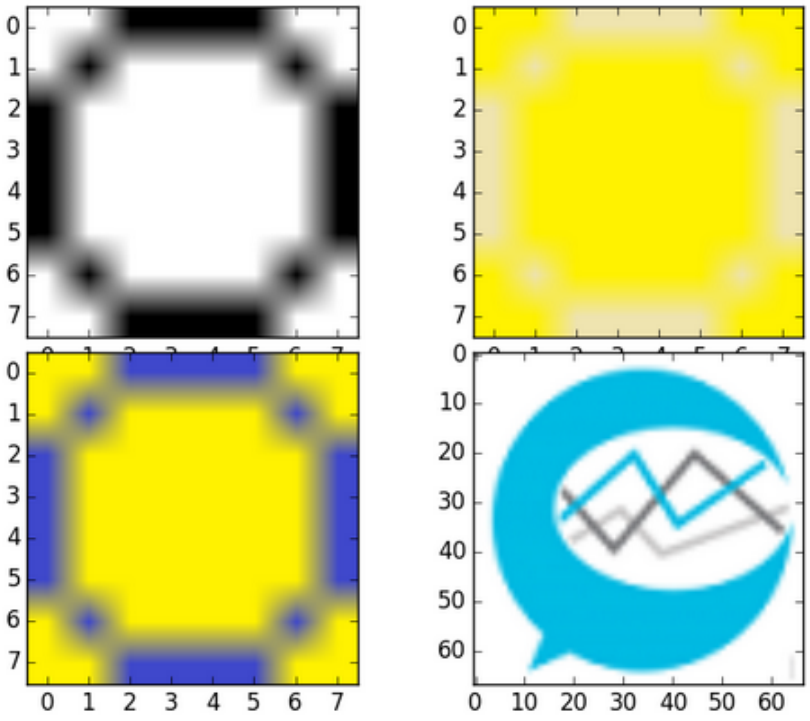

Figure 3: Example

and introduce the mathematical expressions of several commonly-used loss functions as well as the corresponding expression in DeepLearning.

6 Thresholding Logic:

What we've added for now is, from the average of the balance array, we then assess each pixel. If the pixel is brighter than the average, then it is a white. If it is darker than the average, then it is black.

Now, what we can do with this function is feed in the image array, and we're going to be returned the thresholded image array.

Remember our first image was:

```
def threshold(imageArray):
```

```
balanceAr = []
newAr = imageArray
from statistics import mean
for eachRow in imageArray:
for eachPix in eachRow:
avgNum = mean(eachPix[:3])
balanceAr.append(avgNum)

balance = mean(balanceAr)
for eachRow in newAr:
for eachPix in eachRow:
if mean(eachPix[:3]) > balance:
eachPix[0] = 255
eachPix[1] = 255
eachPix[2] = 255
eachPix[3] = 255
else:
eachPix[0] = 0
eachPix[1] = 0
eachPix[2] = 0
eachPix[3] = 255
return newAr
```

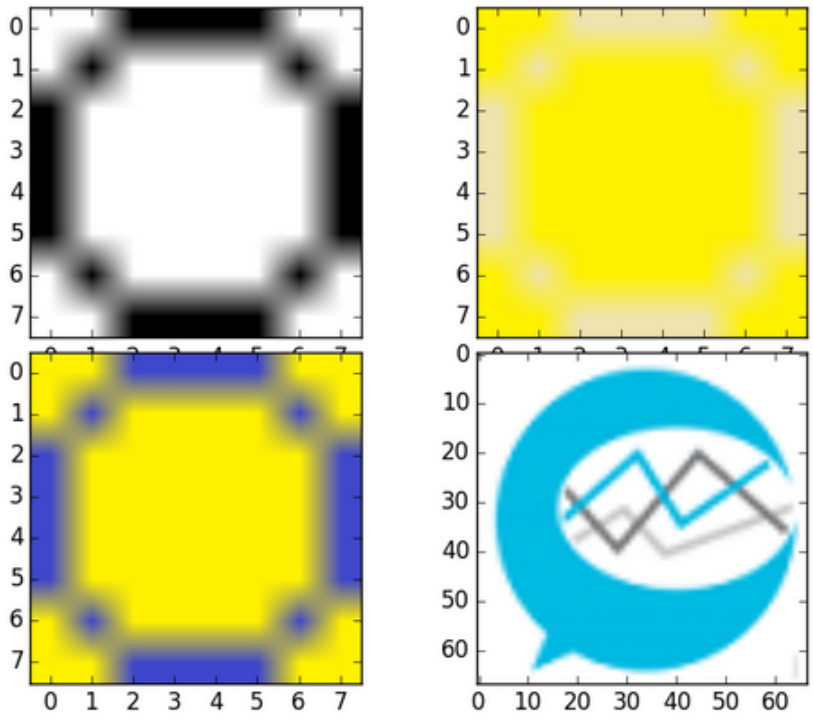

Figure 4: Remember our first image was

Now let us apply the threshold to everything

```
i=Image.open('images/1DJI_0004.jpg')
var=np.array(i)

i=Image.open('images/1DJI_0005.jpg')
var2=np.array(i)
i=Image.open('images/1DJI_0006.jpg')
var3=np.array(i)
i=Image.open('images/1DJI_0007.jpg')
var4=np.array(i)

threshold(var)
threshold(var2)
threshold(var3)
threshold(var4)

fig=plt.figure()
ax1=plt.subplot2grid((8,6),(0,0),rowspan=4, colspan=3)
ax2=plt.subplot2grid((8,6),(4,0),rowspan=4, colspan=3)
ax3=plt.subplot2grid((8,6),(0,3),rowspan=4, colspan=3)
ax4=plt.subplot2grid((8,6),(4,3),rowspan=4, colspan=3)
```

```
ax1.imshow(var)
ax2.imshow(var2)
ax3.imshow(var3)
ax4.imshow(var4)
plt.show()
```

The resulting threshold applied to our images gives us:

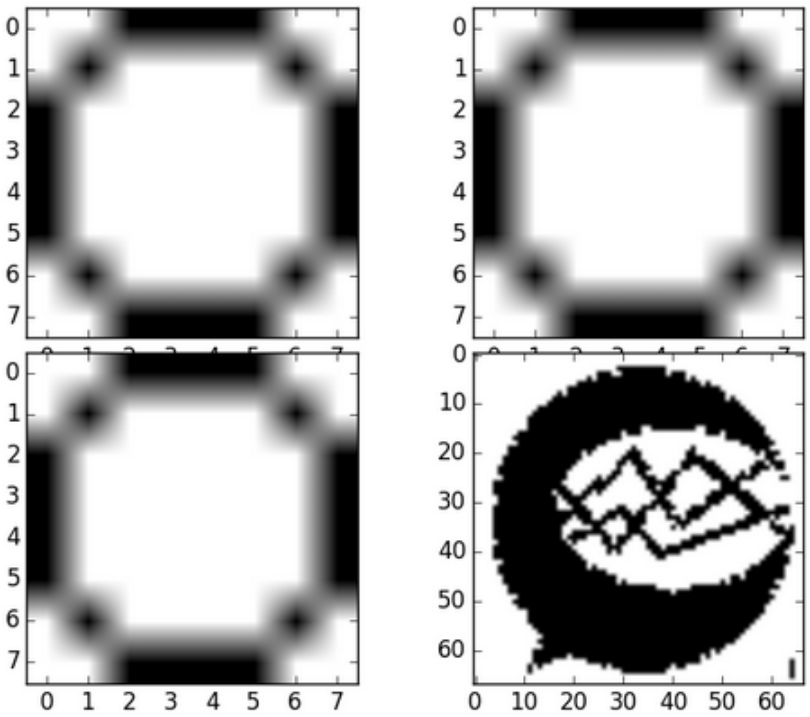

Figure 5: Threshold result

7 Saving our data for training and testing:

Next up, we're going to start using our example numbers data. We want to create image arrays out of our numbers data, saving them, so that we can reference them later for pattern recognition.

For this, we're going to create a "createExamples" function: The purpose of this function is to literally just append the image's array to the file so we can reference it later.

In this, we're just using a flat file as our database. This is fine for smaller data-sets, but you may want to look into working with databases, either SQLite or MySQL in the future.

```
def createExamples():
numberArrayExamples = open('numArEx.txt','a')
numbersWeHave = range(1,10)
```

```
for eachNum in numbersWeHave:
#print eachNum
for furtherNum in numbersWeHave:
# you could also literally add it *.1 and have it create
# an actual float, but, since in the end we are going
# to use it as a string, this way will work.
print(str(eachNum)+'.'+str(furtherNum))
imgFilePath = 'images/numbers/'+str(eachNum)+'.'+str(furtherNum)+'.png'
ei = Image.open(imgFilePath)
eiar = np.array(ei)
eiarl = str(eiar.tolist())

print(eiarl)
lineToWrite = str(eachNum)+'::'+eiarl+'\n'
numberArrayExamples.write(lineToWrite)
```

8 Basic Testing:

Now that we have our sample data, we're ready to compare it. The method that we're going to use will be very simple, yet you'll be surprised how decent it works.

We will simply go, pixel by pixel, comparing whether or not they are the same. First we're running against the test image, which is an image of my hand-drawn 2, which is not included in the training set. Running the code, we get our counters list for matches: Thus, the prediction is a 2, as it is the highest match. So that was a success. You may point out that the next closest match, a 3, is 389. To this I would point out the least matched option was a 321, so you can see that the scale is not really something like 0-500. It is really probably best though of as a range from 400-500 or so. Anything under 400 is too loose of a match to be considered a confident match.

```
from PIL import Image
import numpy as np

import time
from collections import Counter


def whatNumIsThis(filePath):

matchedAr = []
loadExamps = open('numArEx.txt','r').read()
loadExamps = loadExamps.split('\n')

i = Image.open(filePath)
iar = np.array(i)
iarl = iar.tolist()
```

```
inQuestion = str(iarl)

for eachExample in loadExamps:
try:
splitEx = eachExample.split('::')
currentNum = splitEx[0]
currentAr = splitEx[1]

eachPixEx = currentAr.split('],')
eachPixInQ = inQuestion.split('],')

x = 0

while x < len(eachPixEx):
if eachPixEx[x] == eachPixInQ[x]:
matchedAr.append(int(currentNum))

x+=1
except Exception as e:
print(str(e))

print(matchedAr)
x = Counter(matchedAr)
print(x)
print(x[0])

whatNumIsThis('images/test.png')
```

9 Conclusion:

The main objective of this project was to get awared that, while image recognition is a somewhat complex topic in layers, each problem can be broken down and solved with very simple, easily understood, code. I hope you enjoyed.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.