Deep learning (BMEGT52AT07)

Coloring grayscale images

Color Rampage team

Varjas István Péter & Vitanov George

LD2J77 FVX4K2

varjas.pista@gmail.com vitanovg0@gmail.com

2018/19. ősz

1 Task description

Our task is to colorise grayscale images. We deciced to focus on grayscale portrait pictures, because most of the old pictures taken before the popularisation of color photograpy were portraits. Our main goal is to achive accurate colorisation of the human face.

To accomplish our task, we used LAB color space images. In the next section, you can read about the properties of such images, but if you are already familiar with them, just skip to the "Database" section.

2 LAB color space

The Lab color space consist of 3 elements:

1. **L** – Lightness (Intensity).
2. **a** – color component ranging from Green to Magenta.
3. **b** – color component ranging from Blue to Yellow.

In RGB color space the color information is separated into three channels but the same three channels also encode brightness information. On the other hand, in Lab color space, the L channel is independent of color information and encodes brightness only. The other two channels encode color. [1] [2]

This results in the following properties

- Perceptually uniform color space which approximates how we perceive color.
- Independent of device (capturing or displaying).
- Used extensively in Adobe Photoshop.
- Is related to the RGB color space by a complex transformation equation.

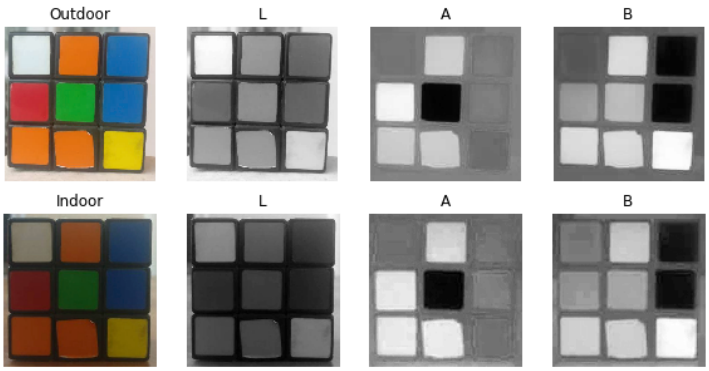

Figure 1: **The Lightness (L), and color components (A, B) in LAB Color space.**

Lets examine 2 images in LAB color space separated into 3 channels:

- The change in illumination has mostly affected the L component.
- The A and B components which contain the color information did not undergo massive changes.
- The respective values of Green, Orange and Red (which are the extremes of the A Component) has not changed in the B Component and similarly the respective values of Blue and Yellow (which are the extremes of the B Component) has not changed in the A component.

3 Database

We decided on using one particular database, but besides this, we found a good collection of portrait databases: [5]

3.1 Main parameters

Our intput database:

- IMDB database [3], with cropped images of faces [4]
- JPEG format files
- Variable sizes and aspect ratios
- Mostly small images ~10-100 kb
- Containing grayscale and faulty images

3.2 Faulty images

Our database contained faulty images:

Figure 2: Obviously faulty images, stripes on: (most edges; top)

It can be seen on the above figure that a faulty image contains vertical or horisontal stripes. The above two image is easily recognisible as fautly, but we found less obvius cases too:


Figure 3: Slightly faulty images, stripes on: (bottom; right side)

A faulty image has stripes along at least one of its edges. In one stripe (a line of pixels) all pixels have the same colour (value). This means, if we calculate the deviation of the pixel values in one stripe, we should get 0. If the image is faulty, it has at least one side, which is fully striped. This means, we can examine an image, by summing the deviation of the pixelvalues in each line of pixels on each side of an image. If on any side, the sum of the deviations results in 0, that side is striped, so the image is faulty.

We decided to examine only the outer 5 pixels on each side of an image, because this proved sufficent to eliminate all faulty image. Our code for this task is:

```
def faulty(args): # Function to determine if an image is faulty
   dev = np.zeros(4)
   val = np.zeros(4)
   err = 5                    # we check on the first 5 pixels if stripes exist
   errp = args.shape[1]-err # and on the last 5 pixels
   for i in range(args.shape[1]):
      for j in range(3):
         val[0] = np.std(args[errp:,i,j]) # this is the i-th column pixels from
         val[1] = np.std(args[:err,i,j])  # 123:i to 128:i and the j.th
         val[2] = np.std(args[i,errp:,j]) # value (L,A,B)
         val[3] = np.std(args[i,:err,j])
      chopnb(val) # Activation limit because std gives small numbers (e-15)
      dev += val  # for a list even if it contains the same elements
# if the 0-err (0-5) and the errp-lastpixel (123-128) pixelrange does not
# contain stripes then:
   if (dev[0] != 0 and dev[1] != 0 and dev[2] != 0 and dev[3] != 0):
      return 0
   else:
   return 1
```

3.3 Grayscale images

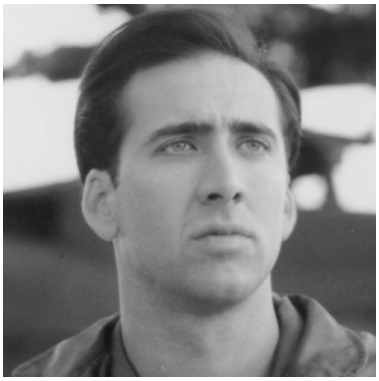


Figure 4: Grayscale images

Our database contained grayscale images too. We examined the images in LAB color space, because unlike in RGB, in LAB they can be easily recognisible as greyscale. Lets examine a 5x5 pixel grayscale image of a flamingo:

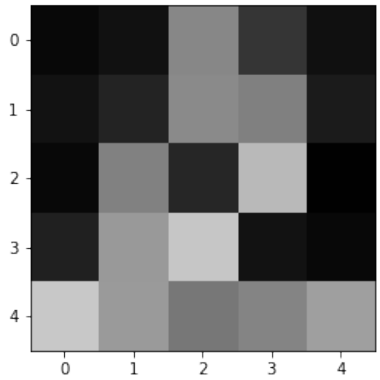

Figure 5: Grayscale flamingo 5x5 pix

$$\begin{pmatrix} 8 & 17 & 135 & 53 & 16 \\ 18 & 35 & 138 & 128 & 27 \\ 8 & 129 & 38 & 185 & 1 \\ 32 & 153 & 198 & 18 & 8 \\ 200 & 154 & 119 & 132 & 159 \end{pmatrix} \begin{pmatrix} 8 & 17 & 135 & 53 & 16 \\ 18 & 35 & 138 & 128 & 27 \\ 8 & 129 & 38 & 185 & 1 \\ 32 & 153 & 198 & 18 & 8 \\ 200 & 154 & 119 & 132 & 159 \end{pmatrix} \begin{pmatrix} 8 & 17 & 135 & 53 & 16 \\ 18 & 35 & 138 & 128 & 27 \\ 8 & 129 & 38 & 185 & 1 \\ 32 & 153 & 198 & 18 & 8 \\ 200 & 154 & 119 & 132 & 159 \end{pmatrix}$$

Figure 6: R G B channels respectively

$$\begin{pmatrix} 2 & 5 & 56 & 22 & 4 \\ 5 & 13 & 57 & 53 & 9 \\ 2 & 53 & 15 & 75 & 0 \\ 12 & 63 & 79 & 5 & 2 \\ 80 & 63 & 50 & 55 & 65 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Figure 7: L A B channels respectively

On the above figures, we can see, that working in the LAB color space gives us the opportunity, to identify a grayscale image by its A and B matrices. An image is therefore grayscale if we sum the absolute values of its pixels in channel A and B respectively, and get zero as a cumulative sum.

First we define a function for iterating and abs[] function on arrays. Then we can sum up the individual array elements with numpy.sum()

```
def abbs(expr): #defining Abs function for arrays
   return [abs(i) for i in expr]

#check if image is grayscale, if not add it to image array
if (np.sum(abbs(img[:,:,1]))+np.sum(abbs(img[:,:,2]))!= 0):
   img_array.append(img)
```

4 Data processing

4.1 Crop function

As mentioned before our database contained images in various aspect ratios. When using PIL.Image.resize() function, we encountered a problem, that this funtcion did not keep the original aspect ratio of the pictures, but rather compressed/elongated them. To overcome this problem, we created a crop funtion. The output of our function is a 1:1 aspect ratio image, with the excess sides or excess stripes on the top and bottom simmetrically removed.

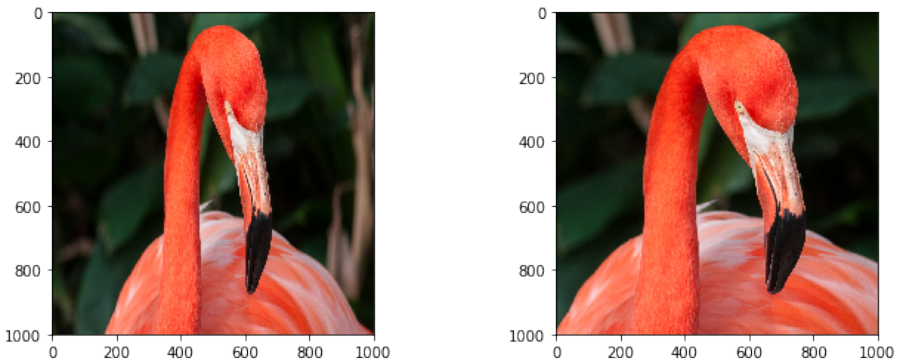

Figure 8: Resized without crop, Resized with crop

Our function to crop a picture to a square shape, with cropping the edges, and giving the max middle square back:

```
def cropmid(img):
   if(img.size[0]!= img.size[1]): # check if img needs cropping first
      xfrom = int((img.size[0]-min(img.size))/2) # we crop from here
      xto = xfrom+min(img.size)                  # til here
      yfrom = int((img.size[1]-min(img.size))/2)
      yto = yfrom+min(img.size)

      return img.crop((xfrom,yfrom,xto,yto))
   else:
      return img
```

4.2 File format

Our task does not require complex data structures, so after some experiments we decided to use .csv file format to store our data. A .csv file generally stores 2d arrays, which is perfect for storing our fix-sized images. It was also an important criteria, that the file should be easily appendable and could be partially read if nessesarry. A .csv is also fast to generate and easy to use. The figure below shows how we store pixel values in .csv format.

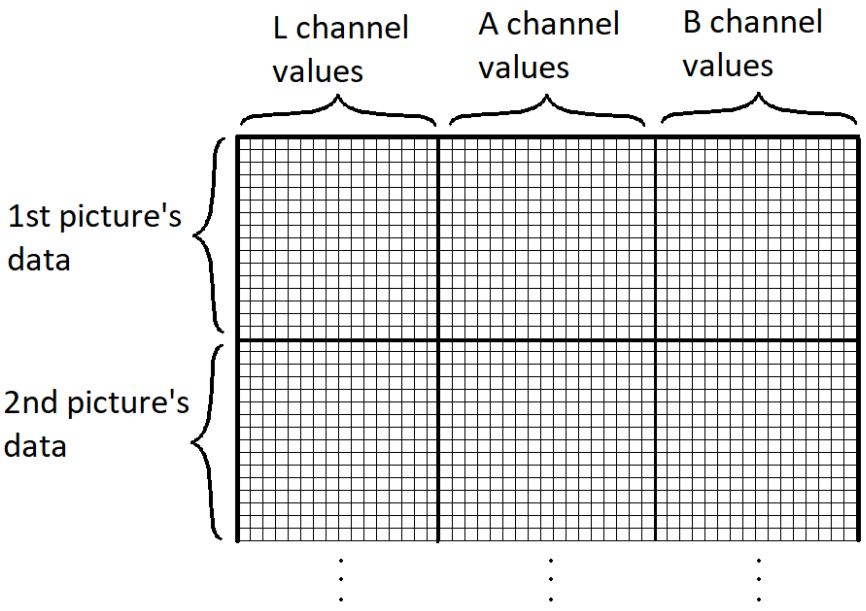

Figure 9: Data storing

4.3 Backup

Our code works by sorting a predetermined number of good images (~16 000), then saveing their LAB pixel values:

```
with open('D:/EVTNG/Dataset/csv_v3/labpics.csv', 'w+',newline='') as out:
```

```
writer = csv.writer(out, delimiter = ',')
for ind in range(len(pix)):
writer.writerows(np.round(np.concatenate((pix[ind][:,:,0], \
pix[ind][:,:,1], pix[ind][:,:,2]), axis = 1)).astype(np.int))
```

We can read the values back with pandas relatively fast:

```
import pandas as pd
df = pd.read_csv('D:/EVTNG/Dataset/csv_v3/labpics.csv', header=None, dtype =
    np.float32)
data = df.as_matrix()
```

4.4 Creating learning data

We used normalization on the input as well as on the output, because the range of our data (each channel's value range) is predetermined. By choosing this method we can easily expand our learning data if nessesarry. (otherwise the recalculation of the deviation and average for the whole dataset would be nessessarry)

4.5 Performance measurements

We measured the performance of different data processing and saving techniques on 10000 images. Data saveed by the python function pickle, will be referred to as .p file format.

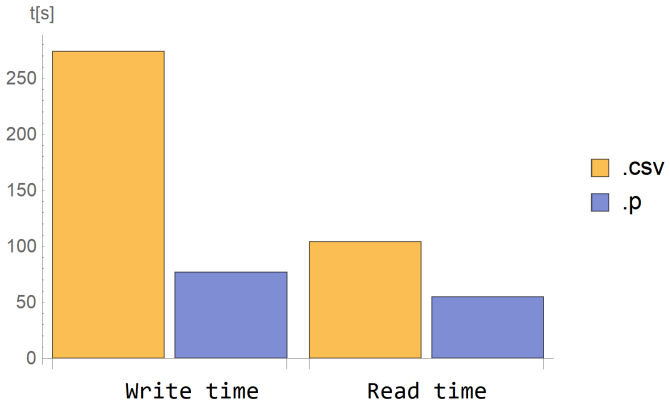

Figure 10: Time to process 10 000 images, using .csv and .p file types

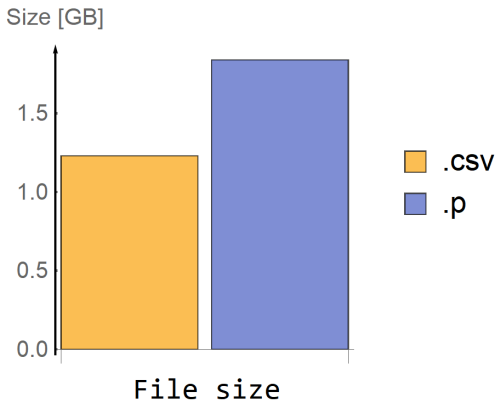

Figure 11: Data size: 10 000 images, using .csv and .p file types

The above numbers are measured means of data processing times and data sizes. We usually did each measurement with the same data 5 times. Although we achieved far better times with pickleing the data (.p) we decided to use (.csv) fileformat, because of the reduced filesize and the useability of a .csv file. (A .csv file is generally interpretable for any program, while pickeling our data gives us a file only interpretable by python.)

Collection of image colorization Deep Learning solutions:

- Colorful Image Colorization
- Deep Colorization
- Image-to-Image Translation with Conditional Adversarial Networks
- Image Colorization with Generative Adversarial Networks
- Image Colorization Using a Deep Convolutional Neural Network
- A learning-based approach for automatic image and video colorization
- Convolutional Neural Network based Image Colorization using OpenCV

References

[1] https://www.learnopencv.com/color-spaces-in-opencv-cpp-python/

[2] https://en.wikipedia.org/wiki/CIELAB_color_space

[3] https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/

[4] https://data.vision.ee.ethz.ch/cvl/rrothe/imdb-wiki/static/imdb_crop.tar

[5] http://www.face-rec.org/databases/