CAV - Codificação Audio e Video
Guião 1

Francisco Oliveira (80108), Hugo Sampaio (79961) and Ricardo Chaves (79946)

***Abstract*—This guide is about audio and video/image manipulation. Python was the chosen language.**

For audio we used the wave python library. It allows us to access directly into the header of the wav file and check or set parameters. It also allows us to extract each sample individually or in bulk. It also allows to write samples into an wav file.

For images and video OpenCV 3.[check version] was used as it allows quick access to video and image contents, since it has a deep integration with Python library NumPy. In some of the exercises (histogram for example), despite the OpenCV toolset already offering a function to calculate the histogram, our own version was implemented for the sake of the exercise.

I. Handling Files

A. WAV Audio Files

1) exercise 1: This exercise asks to copy an audio wav file sample by sample. First three values are extracted from the original wav file: the number of channels, sample width and frame rate. We then set this values into the copied wav file header. Next we access each sample of the wav file individually and write it into the copy file.

B. Image Files

1) exercise 2: Exercise 2 asks to copy an image pixel by pixel.

First an empty matrix (this will be the destination image) with the same dimensions as the source image was created, filled with zeros to allocate memory. Because every pixel is being replaced by the source image's, the initial values of this matrix could've also been 255 (white).

Then the *for loop* iterates through every pixel, reads the pixel intensity value of the source image and writes it to the corresponding destination position - the entire pixel value is copied, whether it's a triple channel image or single channel image.

2) exercise 3: This exercise asks to display an image and video on the screen.

After having a variable with a matrix data structure that represents the image, the *cv.imShow()* method can be called with the name of the window and this matrix variable as arguments. Regarding the video, it's not as simple as with the image. A *VideoCapture* of the source video must be opened, and while it is running, read a frame. Afterwards it's the same as having just an image.

A new window with the image/frame will be displayed.

II. Histograms

A histogram is simply a representation of numerical data. In the case of audio it represents how often each frequency occurred, whereas with image or video (each frame) it shows how many times each intensity value occurred. If the image has more than one channel, a histogram per channel is created.

A. Audio

1) exercise 4: With stereo audio (two channels in this case) we wanted to do an histogram of the left channel, right channel and the average of both of them (mono audio). With this in mind we went sample by sample, extracting the first two bytes as the left channel sample and the last two bytes as the right channel sample. These were then converted into integer numbers (and averaged out for the mono channel) and its occurrence was incremented in the respective dictionary structure. With the dictionaries formed for the right, left and mono channels we then proceeded to create the graphs using the matplotlib library of python.

B. Image/Video

Fig. 1. Image used in the next examples.

1) exercise 5.1 (image): Considering the example images with 8 bit colors (24 bits per pixel if it has 3 channels), the intensities values range from 0 (lowest value) to 255 (highest value), comprehending a possible interval of 256 values (2^8).

An array with 256 positions was created and every pixel iterated on. For each intensity value the corresponding array count was incremented, resulting in an array that has how many times each intensity value occurred in the image.

```
[ 3379.   553.   819.  1305.  1516.  1677.  1940.  1913.  2160.  1890.
  2147.  2226.  2454.  2497.  2809.  3483.  3495.  3743.  3610.  3385.
  3065.  3178.  2526.  3201.  3002.  3038.  2831.  3351.  3694.  3875.
  3985.  3610.  4229.  4557.  5088.  3792.  5618.  5053.  6559.  4744.
  6855.  6413.  6508.  8983.  8238.  7767.  7857.  8627.  7195.  6700.
  7441.  6510.  6705.  7130.  6370.  5967.  6809.  7420.  7398.  7046.
  8165.  7837.  7491.  6921.  5762.  4531.  5055.  4416.  4460.  3993.
  4693.  4130.  6018.  5719.  6193.  6040.  7868. 10105.  8138.  9660.
 10565. 13059.  9865. 11803. 11708. 11256.  6701.  9341.  9324.  7174.
  9440.  9145. 11854. 10953. 10646. 14028. 13320.  8814. 13418. 18414.
 12087.  9934. 13056. 13234. 14009. 10033. 12752. 11121. 14123. 11327.
 11782. 10173. 12517. 11703. 12265. 10572.  9529.  7805.  8484.  8312.
  9022.  7598.  7702.  8644.  7865.  7985.  6867.  7396.  6988.  7885.
  6708.  7298.  6086.  5941.  6008.  6092.  5370.  5347.  5373.  5294.
  5211.  5172.  4437.  4601.  4595.  4892.  3678.  4052.  4420.  4023.
  3861.  3431.  3571.  3739.  3775.  3434.  3488.  3621.  3916.  3531.
  3159.  3148.  3069.  3407.  3087.  2512.  2489.  2707.  2565.  2518.
  2149.  2255.  1942.  2339.  1896.  1681.  1638.  1945.  1714.  1526.
  1618.  1381.  1597.  1473.  1169.  1258.  1230.  1255.  1145.  1142.
   982.  1117.   852.  1166.   817.   662.  1088.  1052.   784.   792.
   875.   923.   615.   792.   808.   719.   762.   818.   738.   611.
   750.   645.   640.   614.   496.   566.   502.   466.   373.   474.
   391.   368.   428.   361.   342.   389.   363.   393.   338.   364.
   327.   321.   334.   286.   255.   312.   252.   250.   227.   231.
   259.   231.   244.   233.   202.   251.   203.   207.   193.   233.
   201.   209.   210.   201.   229.  5571.]
```

Fig. 2. Array representation of the histogram

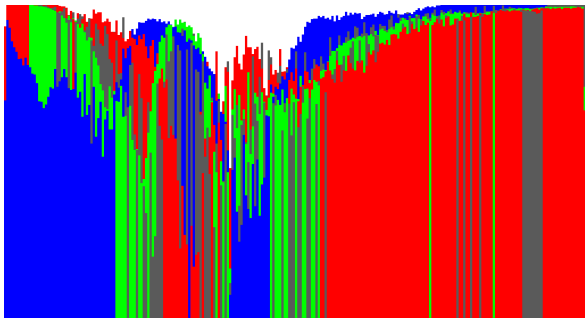

Fig. 3. Visual representation of the histogram (one per channel + gray)

2) exercise 5.2 (video): When considering a video as a sequence of frames (images), each exercise can actually be reduced to just an image exercise. In this case, the procedure described above was used for each frame, resulting in a live visualization of the video's histogram.

III. Bit Reduction

Bit reduction, or quantization, is the process of constraining an set of values to a smaller set of values.

A. Audio

1) exercise 6: Since each sample has a left channel and right channel component the quantization will have to be individually done for each channel, and then joined again as just one sample We reduced the number of bits of each sample using uniform scalar quantization. For this process we used the following equation:

$$new\_sample\_value = floor(\frac{sample\_value}{2 \times (ob - nb)})$$

ob = original bits per sample per channel
nb = new bits per sample per channel

(1)

B. Image/Video

1) exercise 7: We are able to reduce the number of bits use to represent each pixel on each channel using uniform scalar quantization. See Equation (2) below.

$$new\_pixel\_value = floor(\frac{pixel\_value}{2^{ob}} \times 2^{nb}) \times (256/2^{nb}) ob = original$$

nb = new image bits per pixel

(2)

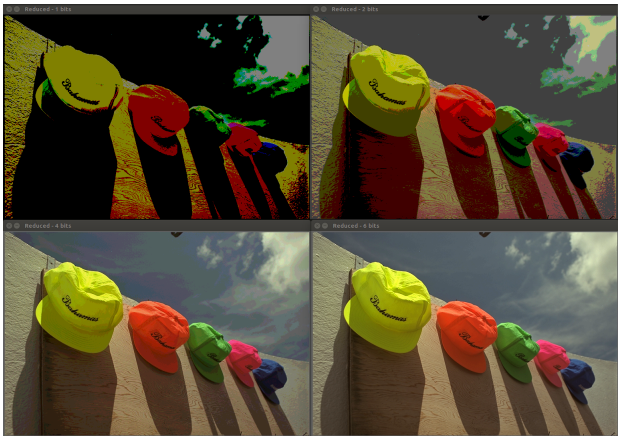

Fig. 4. Different bit reductions applied to Figure 1. From left to right: 1 bit, 2 bits, 4 bits and 6 bits.

IV. Signal-to-noise ratio (SNR)

The signal-to-noise ration is a measure that compares the power of a signal with meaningful information to the power of a signal with unwanted noise.

$$SNR = \frac{P_{signal}}{P_{noise}} \quad (3)$$

A. Audio

1) exercise 8: To calculate the SNR first we read sample by sample both the desired audio file and its original. We find the difference between the two samples to check if it's bigger than the current maximum per sample absolute error. If it is then this difference becomes the new maximum per sample absolute error. We also obtain the sum of the squares of each sample. Using these sums we then apply the following equation to obtain the SNR:

$$SNR = 10 * \log_{10}(sumOriginal - sumCopy) \quad (4)$$

B. Image/Video

1) exercise 9: Before SNR can be calculated, the absolute difference matrix between both images is calculated and squared. On the resulting matrix, Equation 5 is applied.

$$\text{MSE} = \frac{\sum_{i}^{num\_channels} (\sum all\ pixels\ for\ channel\ i)}{number\ of\ pixels \times number\ of\ channels}$$

$$\text{PSNR} = 10.0 \times log_{10}(255^2/MSE)$$

PSNR = peak signal-to-noise ratio
MSE = mean square error

(5)

where the mean square error (mse) is the summation, for each channel, of the sum of every pixel value divided by the number of pixels multiplied by the number of channels.

Using this method the SNR was obtained by comparing an original (our signal) with a second version of the same image to which salt and peeper noise was intentionally added (our noise), as seen in Figure 5.

Fig. 5. PSNR of Figure 1 with salt and pepper noise: 41.95

A second example was also used to obtain the SNR values. This time the resulting image from exercise 7 was used as the noisy image, and the signal to noise ratio to the original version was computed. The result can be seen in Figure 6.

Fig. 6. PSNR of Figure 1 with bit reduction: 27.75

V. Entropy

In the information theory field, entropy is an important measurement. It quantifies the amount of uncertainty involved in the value of a variable. The Shannon entropy, in bits per symbol is given by

$$\sum p_i log_2(p_i), \tag{6}$$

where p is the probability of occurrence of the *i-th* possible value.

A. Audio

1) exercise 10: To build the finite context model we first go sample by sample and extract the value for the left, right and mono channel. For each channel we have a dictionary that is its respective finite context model, with the key of the dictionary being the context. Each entry of the dictionary is also a dictionary with the keys being the value and their content being the number of occurrences.

With the finite context models built we then proceed to calculate the entropy for each channel. We do this first calculating the entropy for each line in the finite context model. We first calculate the probability of occurrence of each value and then for the entropy we use the following equation:

$$entropy = \sum probVal \times log_2(\frac{1}{probVal})$$

probVal = probability of a value occurring given a context

(7)

To calculate the entropy of each channel we then simply calculate the weighted mean of all the values of entropy calculated before, with the weight being the fraction between number of occurrences with a given context and the total number of samples in the audio file.

B. Image/Video

1) exercise 11: While calculating entropy, images were first converted to gray-scale as it's a more representative analysis than considering each channel separate.

To calculate entropy in an image it is required to obtain the probability of each intensity value to appear on the image given a certain context. In this exercise the context is simply the occurrence of a certain intensity value on the previous pixel of the image (the previous pixel to a pixel is considered to be the pixel to it's left). To obtain these probabilities the image must be scanned and the absolute frequency of each intensity value is counted for each context. The relative frequency in a context will correspond to the probability of a value given that context.

With these probabilities computed it is possible to apply equation 6 to obtain the entropy value for each first degree context. From the average of these values it will result the entropy value of the image in bits per symbol.

As reference, Figure 1 has an entropy value of 7.092