Brain Undither

Liam Pulles
School of Computer Science
University of the Witwatersrand
Johannesburg, Gauteng
Student No.: 855442
Email: liam.pulles@students.wits.ac.za

***Abstract*—We explore whether a multilayer ANN can remove dithering effects on 8x8 Ordered Dithered images with 3-bit uniform color using a sliding window. We compare the method to another common undithering method (Gaussian Blur) to see whether ANNs are an effective approach for recovering the original image.**

I. Introduction

Dithering is a class of methods for representing a high bit-depth images with a limited color palette by including intentional noise or "dots". The methods arose in the early days of binary displays, where simple thresholding of images didn't produce a visually pleasing rendition of a greyscale image (Figure 2). Scientists found that "dotting" the image in strategic amounts, depending on the lightness/darkness of the shade of color to be represented, produced a fair approximation of the shade when the image was observed (Figure 1 and 3).

The central method behind all dithering algorithms is to add or remove values from pixels in a regular or semi-regular way, such that the changed pixels have a chance of being thresholded to a different palette color and thus giving the "dotted" effect.

II. Ordered Dithering

The method of dithering we use is called ordered dithering (Figure 4). Ordered dithering works by first constructing a matrix of carefully mixed values ranging from 1 to the size of the matrix, the elements mixed in such a way that consecutive values are on average as far away from each other as possible. The result of this is approximately a "checkerboard" pattern. The matrix is normalized so that all the values are in the range 0 to 1 (Figure 6). This is known as a Bayer matrix [1]. We will use an 8x8 Bayer Matrix for our images.

Next we need to calculate the color spread of the palette we are using, which we denote by r. Presuming the palette contains N^3 evenly distributed colors, and that we represent each color coordinate by 8 bits, r is generally chosen as

$$r \approx \frac{256}{N}$$

We will use 3 bits for our color palette, giving us $r \approx \frac{256}{2} = 128$. In order for the color spread term to make sense, this also requires that the colour palette contain uniformly distributed colors.

Figure 1. Greyscale 8-bit image.

Figure 2. Thresholded 1-bit image.

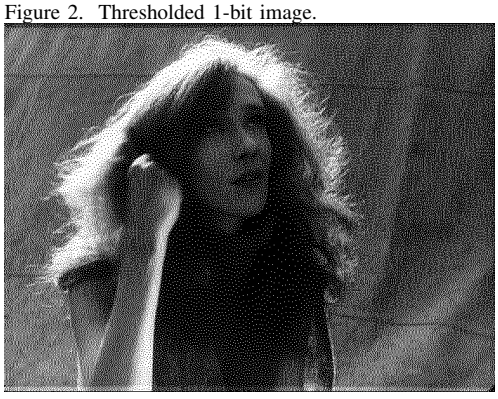

Figure 3. Floyd-Steinberg dithered 1-bit image.
Image comes from the film *Po Zakonu (1926)*

Figure 4. Ordered dithered 3-bit image (2-bits used).

Figure 5. Gaussian blurred 8-bit image.
Image comes from the film *Po Zakonu (1926)*

$$\frac{1}{64} \times \begin{bmatrix} 0 & 48 & 12 & 60 & 3 & 51 & 15 & 63 \\ 32 & 16 & 44 & 28 & 35 & 19 & 47 & 31 \\ 8 & 56 & 4 & 52 & 11 & 59 & 7 & 55 \\ 40 & 24 & 36 & 20 & 43 & 27 & 39 & 23 \\ 2 & 50 & 14 & 62 & 1 & 49 & 13 & 61 \\ 34 & 18 & 46 & 30 & 33 & 17 & 45 & 29 \\ 10 & 58 & 6 & 54 & 9 & 57 & 5 & 53 \\ 42 & 26 & 38 & 22 & 41 & 25 & 37 & 21 \end{bmatrix}$$

Figure 6. An 8x8 Bayer Matrix.

Then for the dithering process, we can choose a new color at each pixel with the following formula:

$$c' = nearest\_palette\_color(c + r(M(x \mod n, y \mod n) - \frac{1}{2}))$$

where $nearest\_palette\_color(x)$ is a function that finds the nearest color in the given palette to x, the nearest being the one with lowest Euclidean distance to x in the RGB colorspace.

Any position of a Bayer Matrix of given size can be found using only bit operations [4], which makes the process very fast. Since we are also using a fixed palette for all images, the method is also suitable for animation. Finally, as the Bayer Matrix is highly regular, the resulting images is also highly compressible.

The reason we chose this method was because of it's "dot spread" and uniform color palette. The fact that the dithering artifacts created by the uniform dithering algorithm are so spread out over the image, means that a sliding window that moves over the image will always have some dithering within it. This is important for us as it means our ANN can possibly be trained to identify certain patterns, and it can be generally guaranteed some form of pattern within each window. The fact that it uses a single, uniform color palette and a semi-uniform distribution of these colors gives us hope that our PCA will give a good reduction of the features in the image.

1/256 x

1	4	6	4	1
4	16	24	16	4
6	24	36	24	6
4	16	24	16	4
1	4	6	4	1

Figure 7. A 5x5 Gaussian convolution filter.

III. Gaussian Blur

Arguably the most common method for undithering an image is the Gaussian Blur.

A Gaussian blur is a convolution filter, whose weights come directly from a 2 dimensional normal Gaussian distribution (Figure 7).

It's chief advantages are that is a simple (and thus quick) one pass filter, and it can operate on an image produced from any dithering method. It's chief disadvantage is that the resulting image is, obviously, blurred (Figure 5). This is still quite a fair approximation of the original image however, which makes sense given that the Gaussian Blur filter's main use is to smooth Gaussian noise, which is similar to the dithering effect.

Our goal then is to see if our undithering method can produce a sharper result while still retrieving the original colors of the image and smoothing out dithering effects.

IV. Our method

Our method uses data similar to a 7x7 convolution filter, in that we take a 7x7 region of the image and through a chain of processes arrive at a new color for the center pixel. By taking a sliding window over the whole image, we can do this for every pixel.

We first add $M - \frac{1}{2}$ back onto the image to normalize the colors a bit (Figure).

For each 7x7 window, we first apply a linear transform derived from a Principal Components Analysis of the image to reduce the feature size from 7x7x3 (since we look at each RGB color channel in the window) to 7x7 while retaining approximately 95% of the variance. Our method is thus not a traditional convolution, as we consider all the color channels

Figure 8. Top: Original dithered image. Bottom: Ünmaskedd̈ithered image. Note how the outliers in the bottom image have been reduced, and shadows are more visible along the brow. Image comes from the film *The Ipcress File (1965)*

of the image together (instead of independently), and we apply a linear transform to the window.

We then use a ANN regressor with two hidden layers to predict the RGB color value (3 outputs) from the transformed 7x7 input vector. Our hidden layers are of size 50 (7x7 plus bias node) and 25 (half of 7x7 plus bias node). Initial testing found that increasing the number of hidden layers from one to two resulted in a sharper image, which motivated our design choice here. We then use the predicted RGB pixels from all windows to directly derive a new image.

V. Data

The training data comes from a collection of approximately 500 stills from video files. The images were dithered using ImageMagick with option "-dither 08x8,3" (without quotes) giving the ordered dithering specification given earlier. The stills range in size from about 640x320 to 1920x1080 - meaning a wide variety in local spatial frequencies in the images and thus a wide variety of samples.

We took approximately 10000 random windows from the dithered images with corresponding target pixels from from each of 110 randomly selected image pairs (dithered and original), giving a total of 1100000 samples

The data was split into 60% training data, 20% testing data and 20% verification data in the original ANN specification, and split internally in the altered ANN specification.

VI. Original ANN specification

As already mentioned, the layer structure of the ANN is 50 (49 plus bias node), 50, 25, 3. The sigmoid function is used as an activation function for the hidden layers, with alpha parameter 1. The network was trained using Stochastic Gradient Descent with 1100000 samples from a variety of images, with learning rate 0.1. After each epoch, the network created a test image, and I used this to decide when the network had converged.

Figure 9. Output after training with standard SGD for 77 epochs.

VII. Training issues

Multiple attempts at training the ANN using Stochastic Gradient Descent produced disappointing results, the chief problem being that the image was too sharp and monotone (Figure 9).

This likely arose from the error function not having a distinct global minimum, as the thresholding process used in dithering makes some of the information unrecoverable and the variety of dimension mixings possible mean there are a lot of possible local minima and saddle points. The fact that we took all the color channels together could mean that it was difficult for the network to decouple the channels separately from the PCA transformed input, and the result was that it trained to predict all the outputs as approximately the same values (giving a monotone image).

Observation of training the original ANN network showed that the vast proportion of weight changes was occurring in the first layer (7 orders of magnitude more than the other layers). The result of this likely led to a very sharp result in the first layer's sigmoid functions, almost approximating a min/max function. This seems to correspond to the sharp color changes we see in the output.

VIII. Altered ANN specification

Because Stochastic Gradient Descent was taking too long to converge and was not able to deal with the problem properly, and because I had time constraints on finishing the project and still wanted to see how the undithering function compared with Gaussian Blur, I decided to train the weights using a higher level library (The MLPRegressor() class from the Python Scipy sklearn library [3]). This library provided an optimized SGD denoted 'adam' [2]. I then took the weights it generated for use in my original function - the code that processes images is still all my own.

Figure 10. Original 24-bit image.

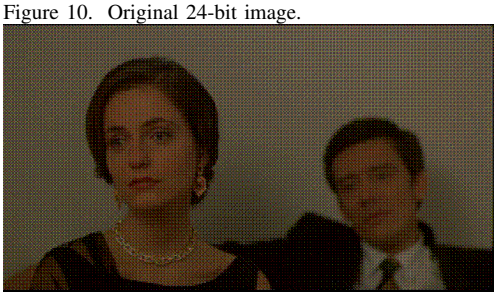

Figure 11. Ordered dithered 3-bit image.

Figure 12. 24-bit Gaussian blurred image.

Figure 13. 24-bit image from our method.
Image comes from the film *Vale Abraão (1993)*

IX. Results

The results of the trained (altered) network can be seen in (Figure 13).

Fortunately, training with the altered ANN managed to produce a much better results. Our result here is visibly sharper than the result using the Gaussian blur (Figure 12), however it also retained some of the dithering artifacts.

X. General Issues

My algorithm is still considerably slower than a Gaussian Blur from a common image processing toolkit, and uses significantly more memory (An image of 1920x816 will use 3GB of system memory to process). However, it is reasonable to assume that implementing it in a lower level library with optimization in mind will yield better results in this respect.

To give an idea of performance, it took approximately two hours to process 4500 640x360 color images with 4 threads, meaning a single image took approximately $((120 \times 60) \div 4500)4 = 6.4s$, which for an image of this size would be considered slow.

The high memory usage and performance issues can be reasonably attributed to the large amount of data extracted from the image (We need to store no. pixels x 7 x 7 in memory when we load the image), the large matrix calculation needed to transform the data with the principal components, and the heavy burden of pushing the data through an ANN with 4 layers.

XI. Conclusion

Although circumstances forced the use of a higher order library to train the weights, we did at least see that it delivered a good result. Whether you choose a Gaussian Blur or our method to undither an image, however, will depend on personal preference.

References

[1] Bryce E Bayer. "An optimum method for two-level rendition of continuous-tone pictures". In: *IEEE Int. Conf. on Communications*. Vol. 26. 1973, pp. 11–15.

[2] Diederik Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[3] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[4] Joel Yliluoma. *Joel Yliluoma's arbitrary-palette positional dithering algorithm*. 2014. URL: http://bisqwit.iki.fi/story/howto/dither/jy/ (visited on 05/22/2017).