Brain Undither

A neural network approach to undithering images.

What is Dithering?

- Dithering is best described as the intentional use of noise to approximate a higher bit-depth image.

- The first dithering algorithms were invented in the early days of binary displays, when simple thresholding of an image produced unsatisfactory renderings.

- It was found that by strategically “dotting” the image, finer shades of gray could be approximated.

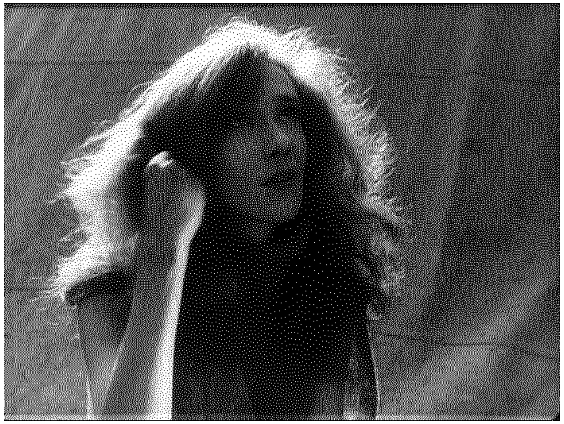
An Example...

Left: Full 8-bit greyscale image
Right: 1-bit thresholded image.

[Picture from the film *Po Zakonu (1926)*]

An Example...

Floyd-Steinberg dithered 1-bit image.



[Picture from the film *Po Zakonu (1926)*]

How does dithering work?

- In order to produce the dots seen in dithered images, it is necessary to add/remove from the total value of pixels before they are thresholded, so they have a chance of being assigned a different color.

- Floyd Steinberg dithering is a form of “Error diffusion” dithering: When a pixel is thresholded, the difference between the approximated and actual pixel is sent to the pixel neighbors, eventually causing some pixels to be thresholded to a completely different color (creating dots).

Ordered Dithering

- In Ordered Dithering, the dithering effect produces a structured pattern instead of “random” dots.

- This image uses 3 colors instead of 2.

Ordered Dithering

- Instead of diffusing the error outwards, we add values to the pixels by tiling a Bayer Matrix over the image (this is an 8x8 variant).

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

- We use this matrix because its elements can be found using only bit operations (fast):

M(i, j) = bit_reverse(bit_interleave(bitwise_xor(x, y), x)) / n ^ 2

Ordered Dithering

- Consecutive values in the matrix are fairly far apart from each other, which gives a "checkerboard" effect.

- Because the dithering is structured, it is highly compressible and also suitable for animations.

An Example

A 3 second clip from *Koyaanisqatsi (1982).* Only 8 colors (3-bits) are being used.

Undithering

- So I want to try and undo this process – go from a dithered image back to the original (or as near as possible).
- Clearly some information is unrecoverable.

- What methods are generally used?

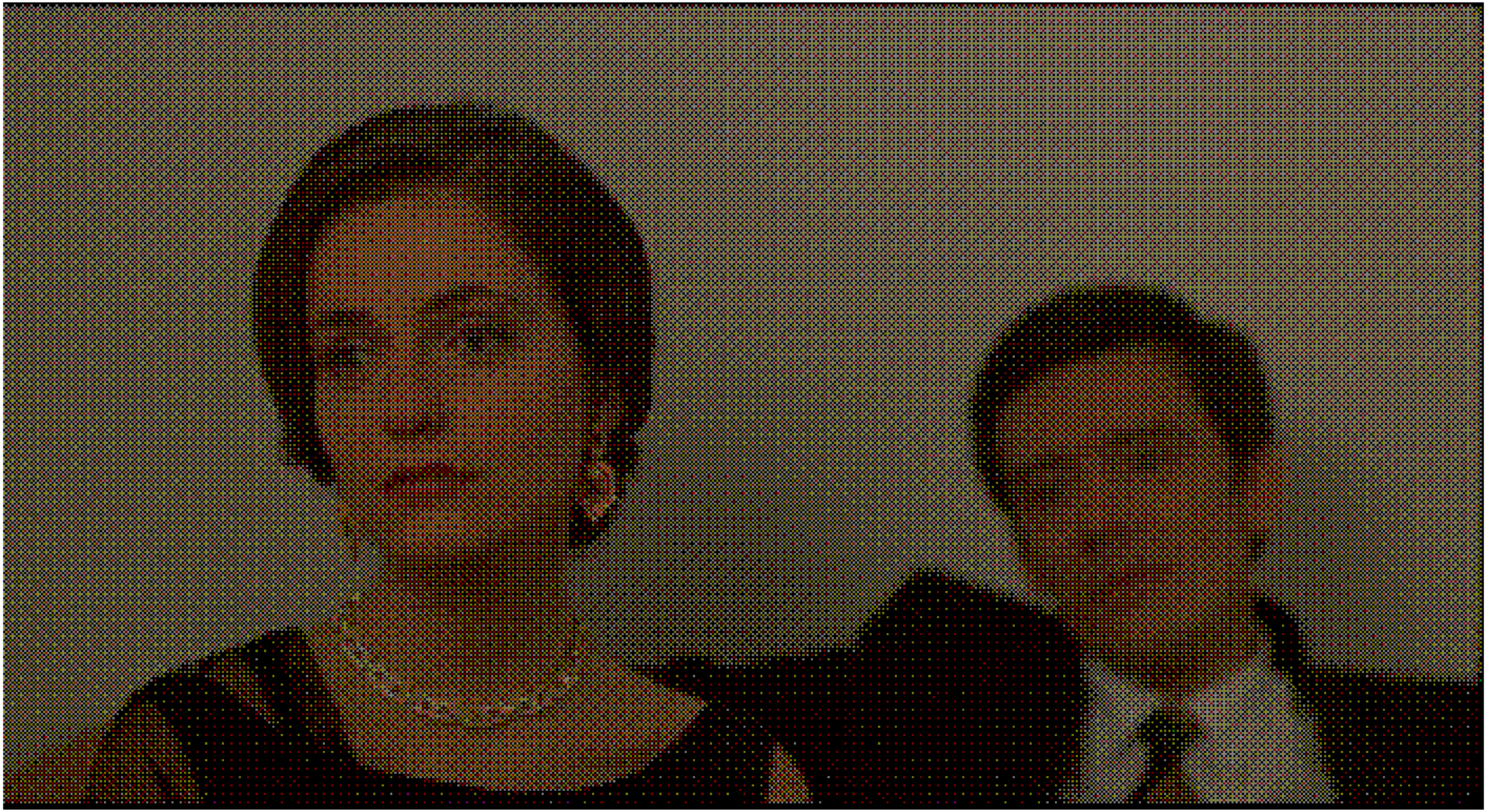
Gaussian Blur

- We can pretend that the dithering pattern is Gaussian noise, and use a Gaussian blur to smooth the image.
- It’s fast, and it’ll work for any dithering method – but the result is obviously blurred.

An Example…

[Picture from the film *Abraham's Valley (1993)*]

An Example...



[Picture from the film *Abraham's Valley (1993)*]

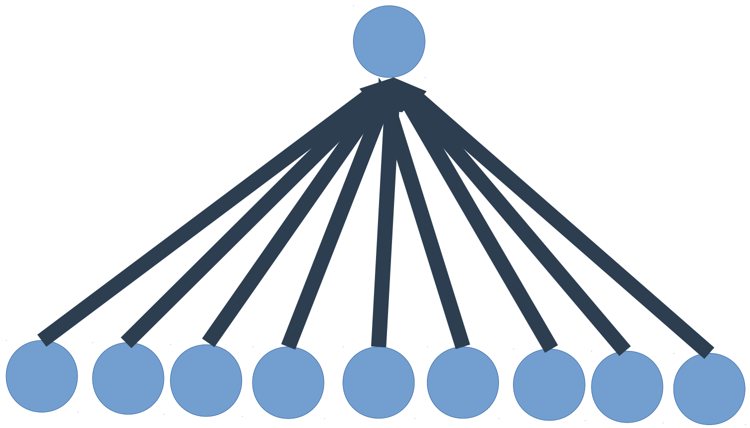
An Example…

[Picture from the film *Abraham's Valley (1993)*]

My Method

- **Instead of applying a simple blur to the image, why not pass each window to a ANN?**
- **Notice first that a convolution is equivalent to a perceptron regressor applied independently to each channel:**

1	1	1
1	1	1
1	1	1



A “Box Blur” filter

My Method

- I want to try and take a 7x7 window (147 features; 49 x 3 color channels) and predict a single color (3 outputs, 1 for each color channel)

- We take all colors at once, since we hope that there is some spectral correlation between colors that we can use.

- But our feature vector is huge!

PCA

- To reduce the size of the input into the ANN, we first apply Principal Components Analysis to our data.
- I found that 95% of the variation in each window can be described by just 49 dimensions (a third of the original 147).

Limiting Focus

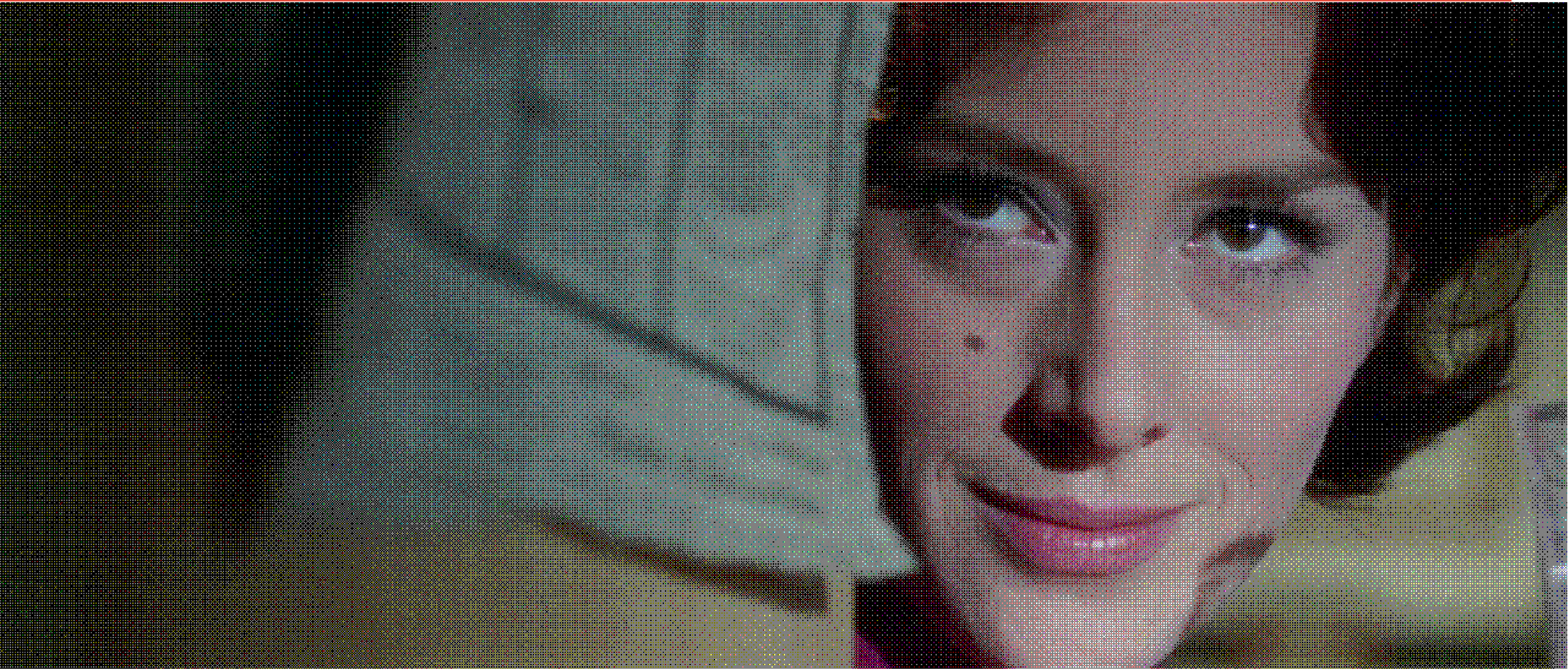
- Because I wanted to maximize the potential for the ANN to work, and because I wanted to see if a specific dithering method encoded extra information in its structure, I trained my network solely on ordered dithered images (specifically using an 8x8 Bayer Matrix and 3-bits of color).

- This allows us to do some useful pre-processing as well...

Removing the structuring element

- Why not try remove the Bayer Matrix from the image, to normalize the colors a bit and to restore some color variation?

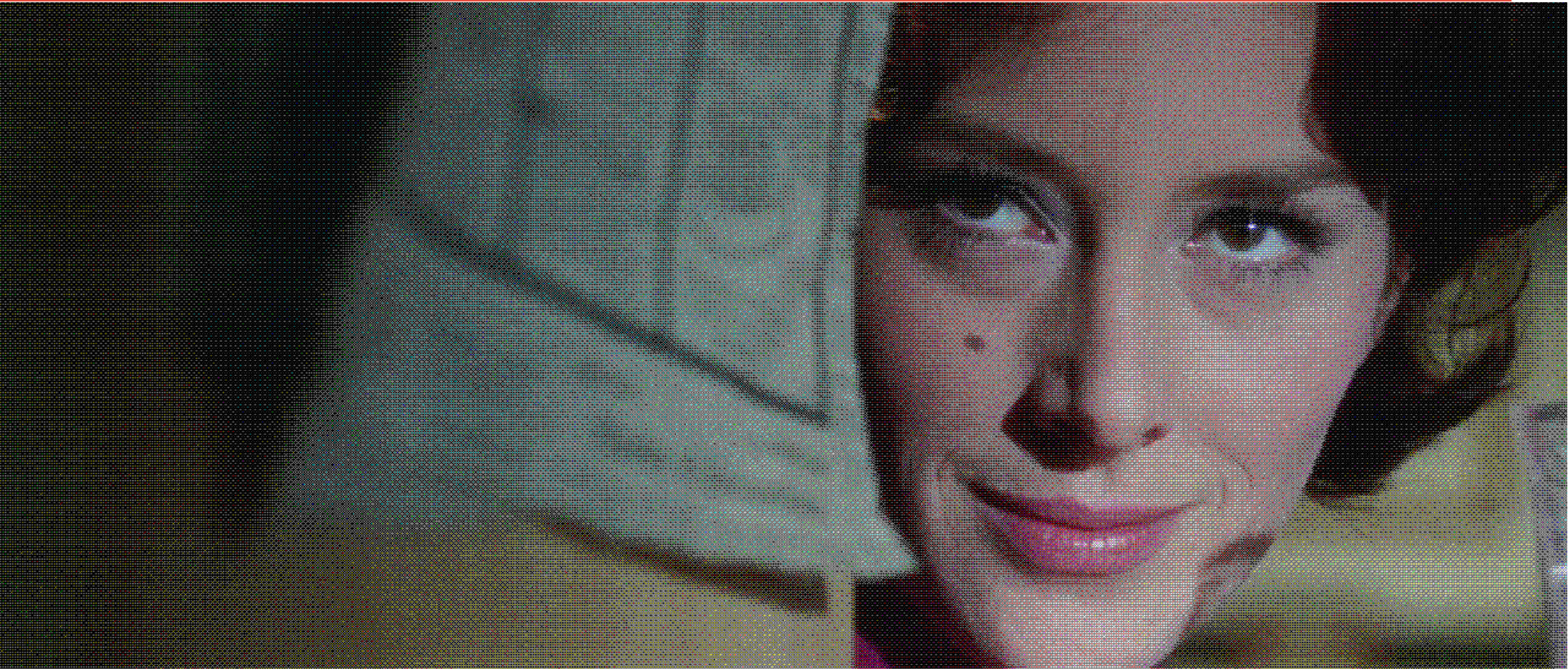
An Example...



The original 8x8, 3 bit ordered dithered image.

[Picture from the film *The Ipcress File (1965)*]

An Example...



The “unmasked” version.

[Picture from the film *The Ipcress File (1965)*]

An Example

Top Image: Original
Bottom Image: Unmasked ver

[Picture from the film *The Ipcress File (1965)*]

Unmasking Evaluation

- It turns out that this has minimal effect on the resulting image, But since it’s almost “free” we’ll keep it. :)

Training data

- The training data was a selection of stills from a variety of videos.

- The stills are of varying brightness, color, and spatial frequency (standard vs. high definition, small image vs large image, sharp vs blurred, etc.).

- Each sample contains a window from the dithered image, with the corresponding center pixel from the original image.

- I took approximately 10,000 random samples from each of 110 random images to give a total of 1,100,000 data points.

ANN structure

- My ANN has 2 hidden layers, of size 50 (7x7 + 1 bias node) and size 25.
- Initial tests comparing the use of 1 hidden layer over 2 showed that the use of 1 layer left the image too blurred.
- More layers = Better approximation
- More layers = Slower training; prediction

- 2 hidden layers is a good compromise.

Result...

- Here is an image from the trained ANN:

Original | My Method | Gaussian Blur 5x5

[Picture from the film *Abraham's Valley (1993)*]

Conclusions...

- In the process of going from 24-bits of color to 3-bits of color, it is inevitable that some color information will be unrecoverable.

- My method gives (arguably) better results than a standard Gaussian Blur but it is considerably slower and more intensive.
(Holding the neccesary data in memory for an image of 1920x816 requires 3 GB!)
- Optimizing the code and implementing the method in a lower level language will likely help correct this.
- And now...