# Computational Vision
# U. Minn. Psy 5036
# Daniel Kersten
# Lecture 10: Image processing

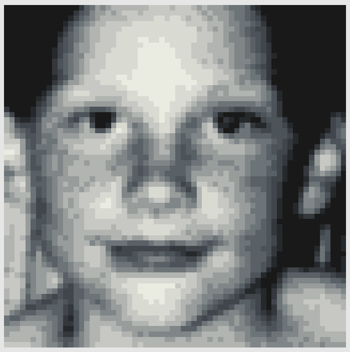
## Initialize

■ **Read in Add-in packages:**

```
Off[General::"spell1"];
SetOptions[ArrayPlot, ColorFunction → "GrayTones", DataReversed → True,
  Frame → False, AspectRatio → Automatic, Mesh → False,
  PixelConstrained → True, ImageSize → Small];
```

■ **The input 64x64 image: face**

```
width = Dimensions[face]〚1〛; size = width;
hsize = width/2;
hfwidth = hsize;
height = Dimensions[face]〚2〛;
face;
gface = ArrayPlot[face]
```



## Outline

### Last time: Multiresolution, spatial fiter models of early visual processing

### Single-channel spatial filtering

One assumes a convolution filter of a fixed shape (e.g. the weights of a DOG filter) that gets applied across the whole image to predict an output "neural image".

### Multiple spatial frequency channels

One assumes a set of convolution filters, each set has a fixed shape (e.g. the weights of gabor filters describe a fixed template) that get applied across the whole image to predict a set of output "neural images", where each neural image represents information at a particular spatial scale and orientation. The collection of filters, identical except for position, is called a channel.

**Global vs. local:**

Sinusoidal basis functions are global filters. Global filtering is not a good model because they give up spatial localization, at the expense of spatial frequency information (the filters would have to extend across the whole space implying large receptive fields).

**Psychophysical experiments.**

-> Multi-resolution, but local filters. Orientation selective, spatial frequency selective, position-dependent.

-> A multi-resolution (spatial frequency), multiple channel (distinguishes spatial frequency and orientation filtering dimensions) models. Provides a standard account of the spatial filtering properties of neurons in the primary visual cortex -- the so-called simple cells in  V1.

### Multiresolution analysis with local filters

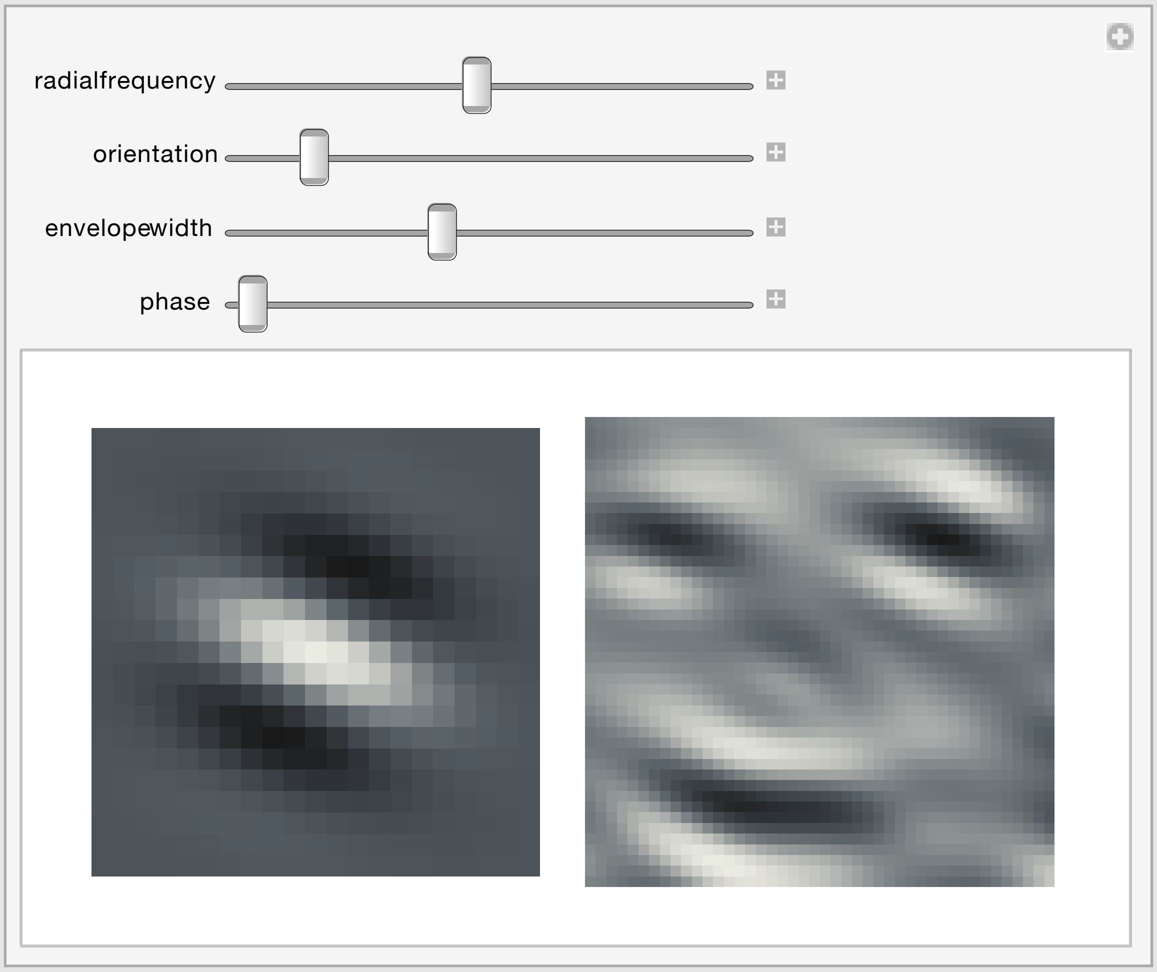
Instead of sinusoidal basis functions, we can filter with localized "gabor function" filters:

```
Grating[x_,y_,fx_,fy_,phase_] := Cos[(2.0 Pi (fx x + fy y) + phase)];
GratingPatch[x_,y_,fx_,fy_,sig_,phase_] := Exp[-((x)^2 + (y)^2)/(2*sig^2)]*Gratin
kern[fx_, fy_, sig_,phase_] :=
  Table[GratingPatch[x, y, fx, fy, sig,phase], {x, -1, 1, .1}, {y, -1, 1, .1}];
```

```
SetDelayed::write: Tag List in
    {{1., 0.882497, 0.606531, 0.324652, 0.135335, 0.0439369, 0.011109, 0.00218749,
        0.000335463, ≪34≫, 1.14688×10^-24, 1.92875×10^-22, 2.52616×10^-20,
        2.57676×10^-18, 2.04697×10^-16, 1.26642×10^-14, 6.10194×10^-13, ≪14≫},
      {≪1≫}, ≪47≫, {≪1≫}, ≪14≫}[≪1≫] is Protected ≫
```

The following demo illustrates a single neural image for various choices of spatial frequency and orientation. In addition, the envelope width manipulates "bandwidth"--i.e. more cycles under the gabor means narrow spatial frequency tuning and thus narrow bandwidth. The phase slider moves one from "bar" to "edge" type receptive fields.

```
Manipulate[
 GraphicsRow[{
   ArrayPlot[kern[fr * Cos[theta], fr * Sin[theta], sig, phase]],
   ArrayPlot[ListConvolve[kern[fr * Cos[theta], fr * Sin[theta],
      sig, phase], face]]}], {{fr, 1, "radial frequency"}, .1, 2},
 {{theta, .4, "orientation"}, 0, Pi},
 {{sig, .4, "envelope width"}, .001, 1}, {{phase, 0, "phase"}, .0, Pi / 2}]
```



## Self-similarity

When the filters have the same shape except for a change of scale ($x \to \alpha x$), they are called *self-similar*.

The self - similar idea is important to vision because of the importance of scale sensitive and scale invariant processing. Further, the self - similar aspect of these neural models bore a close resemblance to the emerging mathematical field of wavelet analysis. The emphases are different-- over - completeness may be important and vision does the projections in parallel (the serial algorithmic component of wavelet computation is integral to the mathematical interest).

### ■ Human statistical efficiency for detecting gabor patches

Burgess, Wagner, Jennings and Barlow (1981) combined the SKE observer and spatial frequency analysis of human vision to find out how efficiently humans detected patterns. They showed in a 1981 Science article that narrowly windowed sinusoids were detected with high efficiency (>70%) when added to static visual noise. Further, these targets were detected more efficiently than disks of light.

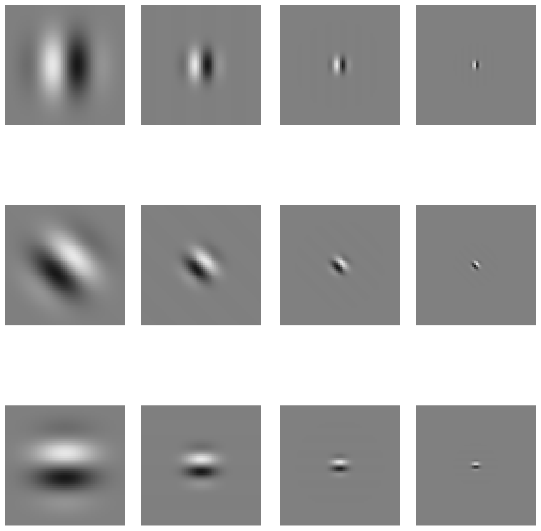
You basically have all the tools to replicate the experiment of Burgess et al. You can compute d' for the ideal observer for signal-known-exactly patterns. And you can generate Gaussian-windowed sinusoids and add them to gaussian white noise. If you measure the percent correct, and convert that to d' for the human observer, you can calculate the absolute efficiency for human detection--and contribute to answer the question of what the eye sees best.

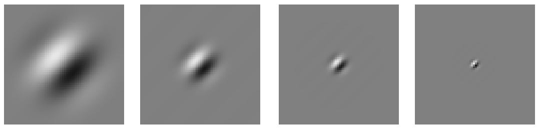
Watson, Barlow & Robson (1983) found that that a 7 c/deg grating drifting at 4 Hz, (with a narrow gaussian envelope in space and time) was detected more efficiently than other patterns. Further, the quantum efficiency was very low (<0.05%).

### ■ Bottom-line: image coding in terms of scale and orientation:

#### A model for human spatial image representation

At each spatial location, project the image onto a collection of basis vectors (i.e. compute the dot product) that span a range of *spatial scales* and *orientations*:





In general, these neural models of basis functions may be over-complete, and non-orthogonal. And there may be a range of phases. Above we show only the "sine-phase" or "edge-detectors" of Hubel and Wiesel.

## Linearity?

Not in general. There are kinds of non-linearities that have been introduced to refine, or model other types of neural populations: 1) rectification, and the combination of on-center, off-center responses 2) adding squared outputs of sine-phase and cosine-phase filter to produce “contrast energy” filters; 3) Contrast normalization, where the output of an otherwise linear neuron is normalized by the “energy” ouputs of ones nearby in space.

We’ll discuss some of these in later lectures

## What is a multiresolution scale/orientation representation good for?

What is the computational significance of a wavelet-like decomposition?

Efficient coding?

-> savings in neurons, or metabolic requirements?

-> representations for efficient learning, subsequent coding?

-> analysis of natural image statistics

Analysis of what vision needs to recognize objects, etc..

-> Edge detection?

-> Edge detection at different spatial scales. Combining over spatial scale

## Today

### ■ Upcoming dates:

Mid-term: Oct. 21th. Study-guide available later this week.

Final project outlines due: Nov. 13th.

- **Final projects**
- **Image manipulations**

## Final projects

### Format

Should be written like a scientific paper.

Might require most of the code to be put in appendices.

Can use modules you find elsewhere, but preserve copyrights, and reference

*Your "audience" will be your class peers.*

## Image processing tools

### *Mathematica* has a library of built-in functions for doing image manipulations

See the **Basic Image Manipulation**, **Image Processing & Analysis**, and the **Image Filtering & Neighborhood Processing** guides.

Here are some examples:

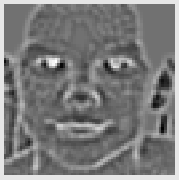
```
Image[Reverse[face]]
```

We've seen a  simple blur using BoxMatrix:

```
ImageConvolve[ , BoxMatrix[2] / 9] // ImageAdjust ;
```

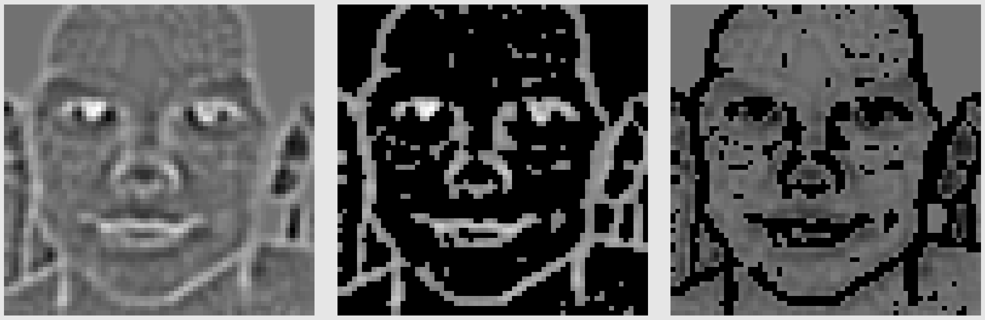
And the built-in Laplacian of a Gaussian filter which can be used to model center-surround, lateral inhibitory filtering:

```
ifiltered = LaplacianGaussianFilter[ , 2] // ImageAdjust
```



We can quickly generate the “neural image” for on- and off-center responses:

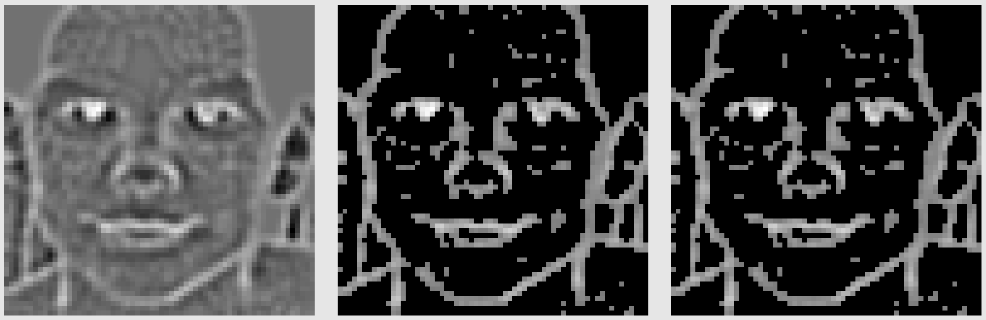
```
GraphicsRow[{ifiltered, ImageMultiply[Binarize[ifiltered ], ifiltered],
  ImageMultiply[ColorNegate[Binarize[ifiltered ]], ifiltered]}]
```



The above used a default threshold to divide above mean from below mean responses. You could use:

`threshold = Mean[Mean[ImageData[ ]]]` as the second argument of Binarize[].

```
GraphicsRow[{ifiltered, ImageMultiply[Binarize[ifiltered ], ifiltered],
  ImageMultiply[Binarize[ifiltered ], ifiltered]}]
```



## Point operations

In addition to the image specific functions, it is also useful to understand how to do image processing using more generic functions.

### ■ Various contrast definitions

For *simple stimuli*, contrast can be defined as:

(Imax - Imin)/(Imax + Imin): Called "Michelsen contrast". Particularly appropriate for gratings, or stimuli with primary luminance peak and valleys in the image.

$\Delta I/I_{\text{background}}$: Often used in psychophysics of small points/disks against a larger background, or temporal increments/decrements ($\Delta I$) relative to a base level ($I_{\text{background}}$).

$\Delta I/I_{\text{mean}}$: Gives same number as Michelsen for gratings, when $\Delta I$ corresponds with the amplitude of the grating.

For *complex stimuli*, we could represent contrast in terms of standard statistical measures on luminance. Let the mean luminance: $I_{\text{mean}} = \sum_{x,y} I(x,y)/N$, where N is the number of pixels. The variance of the luminance is: $\frac{\sum_{x,y} (I(x,y)-I_{\text{mean}})^2}{N}$. Then the standard deviation could be used to provide an overall measure of contrast:

$$\sqrt{\frac{\sum_{x,y} (I(x,y)-I_{\text{mean}})^2}{N}}$$

(The formula for the unbiased variance estimate could also used, where N is replaced by N-1).

However, the above standard deviation definition depends on the units (e.g. candelas/meter^2). Further, it is useful to have contrast definitions that come closer to capturing the perceptual aspects of contrast in an image. We begin by defining contrast at a point--called local contrast.

The visual system is relatively insensitive to the mean luminance, suggesting that a useful measure of local contrast is luminance divided by the mean, whose ratio is dimensionless. In addition, the subjective, qualitative difference between "bright" and "dark" suggests positive and negative contrast, respectively. Thus we define local contrast at a point (x,y) as:

$$c(x,y) = \frac{I(x,y) - I_{\text{mean}}}{I_{\text{mean}}}$$

The mean of c(x,y) is zero by definition, with positive and negative values corresponding to bright and dark relative to the mean. (Note that local contrast can be defined as function of time too, c(x,y,t).) The definition of local contrast at a point raises the question: over what range should $I_{\text{mean}}$ be calculated? This question is relevant to the problems of local adaption to light level, and to tone mapping. The default for us will be to take the mean over the whole image.

Given c(x,y), we now calculate a summary measure of contrast. The variance of c(x,y) is the same as the variance of I(x,y)/$I_{\text{mean}}$ and is given by: $\frac{\sum_{x,y} c^2(x,y)}{N}$. The root mean square (r.m.s.) of set of measurements $m_i^2$ is by definition: $\sqrt{\frac{\sum_i m_i^2}{N}}$ . Thus one can define r.m.s. contrast as the square root of the variance of c:

$$\sqrt{\frac{\sum_{x,y} c^2(x,y)}{N}}$$

This definition of r.m.s. contrast provides us with a useful summary measure of overall "contrastiness" for a complex image.

By analogy with physics (substituting space for time) the square of r.m.s contrast is sometimes called "*contrast power"*. And then "*contrast energy*" is defined as: *contrast power x area*. In psychophysics, "area" is often measured in squared degrees of visual angle. Contrast energy is a useful measure when one is concerned about how the size (or duration) of an image patch affects its visibility. If x and y are measured in degrees of visual angle, then, $\int c^2(x, y)$ dxdy is the contrast energy, which in the discrete case can be estimated as:

$$\sum_{x,y} c^2(x, y) \text{ x } \left(\frac{\text{area}}{N}\right) = \sum_{x,y} c^2(x, y)\, \Delta x \Delta y,$$ where Δx and Δy are the dimensions of a pixel in degrees.

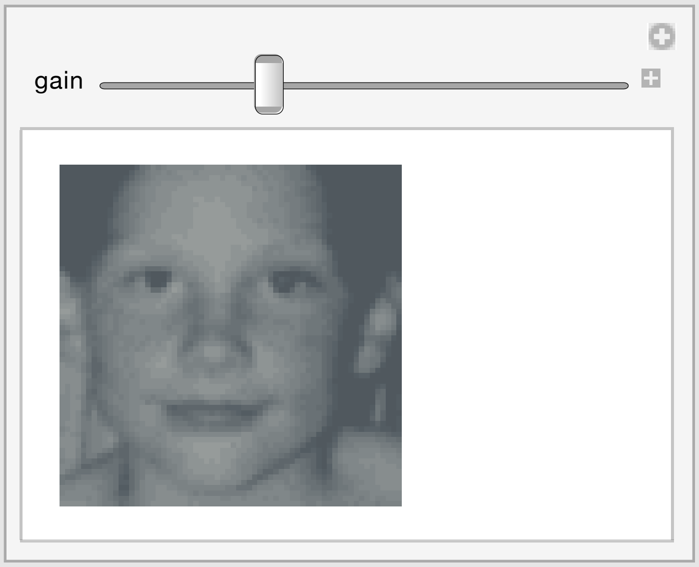
If we extend the above contrast definitions to time, in terms of general appearance, the apparent contrast of an image doesn't increase with time, so contrast energy (contrast power x duration) is irrelevant. However, over short durations (less than 100 msec or so), human vision approximately integrates contrast power, so contrast energy is a better predictor of threshold. But for longer durations, contrast power is the better predictor.

Side note on the relation of contrast energy of an image to its fourier representation: Parseval's theorem says that the contrast energy of an image is equal to the integral of the square of the amplitude spectrum (called the power spectrum) over spatial frequency. Contrast energy doesn't depend on the phase relationships.

## ■ Contrast manipulations

Adjusting contrast (gain=1 leaves image unchanged, gain=0 reduces it to a uniform field):

```
μ = Mean[Flatten[face]]; gain = 0.045;
Manipulate[
ArrayPlot[gain (face - μ) + μ, Mesh → False, Frame → False,
   PlotRange → {Min[face], Max[face]}],
  {{gain, 0.045}, 0, 1}]
```



## ■ Psychophysics and contrast

When measuring human sensitivity, it is important to carefully measure and calibrate the image stimuli. Because standard 8-bit computer displays resolve 256 graylevels, it can be useful to convert the stimuli into a range going from 0 to 255.

Scale so values are represented as graylevels between 0 and 255:

```
α = 255 / (Max[face] - Min[face]);
β = -α Min[face];
```

```
face256 = α face + β;
```

**Exercise: Normalize face so that it has a mean level of zero, and an r.m.s. contrast of 1. Use ListPlot and Flatten[] to show a scatter plot of the values before and after the scaling.**

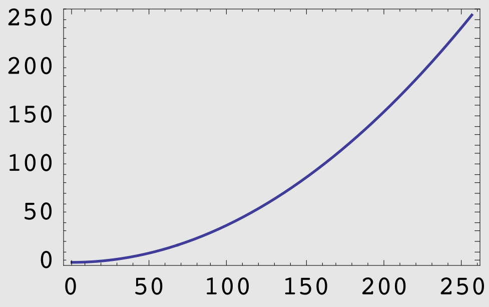
**Exercise: Given that human contrast sensitivity for a sinewave grating can be as high as 500, could you get a good measure of it using a typical computer graphics screen?**

**Exercise: Produce a negative image by reversing the contrast**

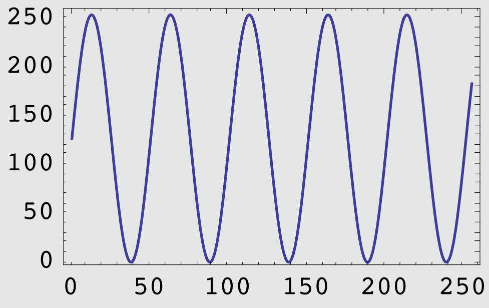
## ■ Gamma correction

Computer operating systems allow one to adjust for various non-linearities between displays. A typical CRT has a non-linear relationship between measured screen intensity and the voltage supplied. A fairly standard way of summarizing the non-linear relationship is in terms of "gamma": *intensity = a x voltage^gamma*. Let's assume that we are using intensity units that range from 0 to 255 and voltage units also going from 0 to 255.  intensity = 255^(1-gamma) x voltage^gamma:

```
output[input_, gamma_] := 255^(1-gamma) input^gamma;
Plot[output[x, 2], {x, 0, 255}, Frame → True, ImageSize → Small]
```

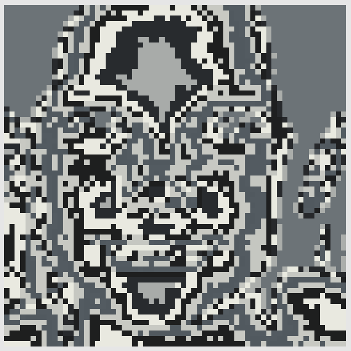


```
(output2[input_] := 127. Sin[input/8] + 127;);
Plot[output2[x], {x, 0, 255}, Frame → True, ImageSize → Small]
```



You can do a many-to-one mapping:

```
ArrayPlot[output2[face256], PlotRange → {0, 255}]
```



What does the fact that you can still see a recognizable form in the above picture tell you about human vision?

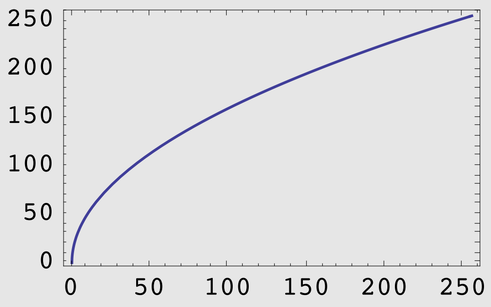
The computer's display card has a look-up-table (LUT) that can be loaded with the inverse gamma function to linearize the display.

```
Solve[output == 255^(1 - gamma) input^gamma, input]
```

```
Solve::ifun :
  Inverse functions are being used by Solve, so some solutions may not be found;
      use Reduce for complete solution information. ≫
```

$$\left\{\left\{\text{input} \to \left(255^{-1+\text{gamma}}\,\text{output}\right)^{\frac{1}{\text{gamma}}}\right\}\right\}$$

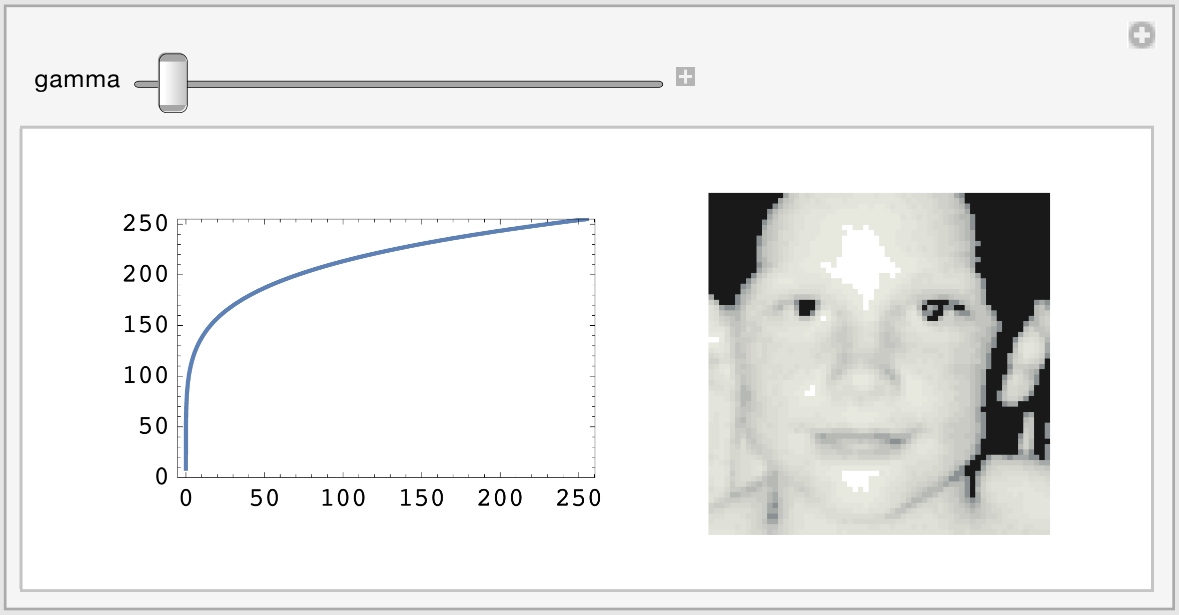
```
inverse[output_, gamma_] := (255^(-1+gamma) output)^(1/gamma);
Plot[inverse[x, 2], {x, 0, 255}, Frame → True, ImageSize → Small]
```



## Using gamma to do point operations

You can also use the gamma transformation to do non-linear point operations on an image:

```
Manipulate[
GraphicsRow[
  {Plot[output[x, gamma], {x, 0, 255}, Frame → True, ImageSize → Small,
    PlotRange → {0, 255}], ArrayPlot[output[face256, gamma],
    PlotRange → {0, 255}]}],
 {gamma, 0.1, 4}]
```

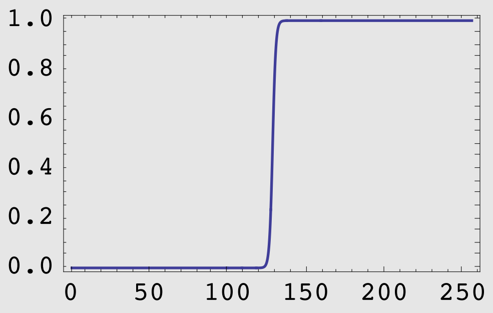


## ■ Sigmoidal contrast manipulation

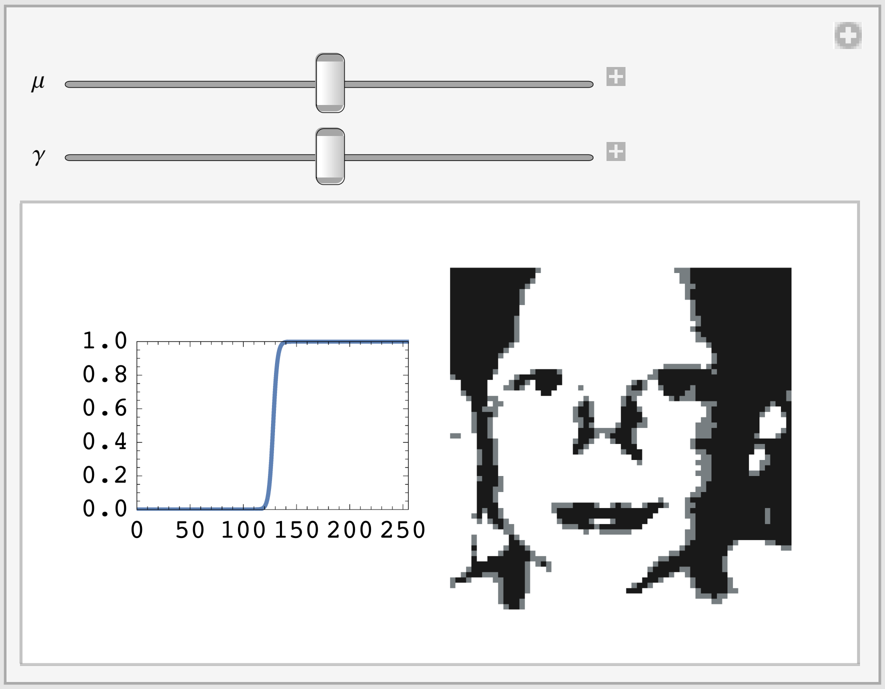
Here is a gain function (called the "logistic function") that manipulates contrast smoothly--a "soft" threshold:

`squash[x_, μ_, γ_] := N` $\left[\frac{1}{1+e^{-\gamma(x-\mu)}}\right]$;

```
Plot[squash[x, 128, 1], {x, 0, 255}, Frame → True, ImageSize → Small]
```



```
gain = 0.045`; μ0 = Mean[Flatten[face256]];
Manipulate[
 GraphicsRow[
  {Plot[squash[x, μ, γ], {x, 0, 255}, Frame → True,
    PlotRange → {{0, 255}, {0, 1}}],
   ArrayPlot[squash[(face256 - μ0) + μ0, μ, γ],
    PlotRange → {Min[face], Max[face]}]}], {{μ, 128}, 0, 255},
 {{γ, .5}, 0, 1}]
```



As $\gamma$ grows, the sigmoid approaches a hard-threshold. Images that are forced to have only two values are called "Mooney images".

We can make Mooney images more directly using a function that takes an image and sets pixels bigger than $\tau$ to 255, and if less than (or equal to) $\tau$, to 0:

```
Mooney[image_, τ_] := Map[If[# > τ, 255, 0] &, image, {2}];
Image[Mooney[face256, 32]];
```

See also: Binarize[], ColorQuantize[]. And the famous Dalmation dog in perception books.

(See Moore and Engel, 2001, and Hegdé J & Kersten D. (2007) for applications of Mooney images to studying vision.

**Exercise: Write a function that quantizes an image to a set of gray levels specified by a set of thresholds: $\tau_1, \tau_2, \tau_3, ..., \tau_{N-1}$. Set N=3. (Try using Which[]).**

---

## Simple statistics

First-order, i.e. don't take into account relations between pixels

### Mean, variance, r.m.s. contrast

```
μ = Mean[Flatten[face]];
σ = Sqrt[Variance[Flatten[face]]];
```

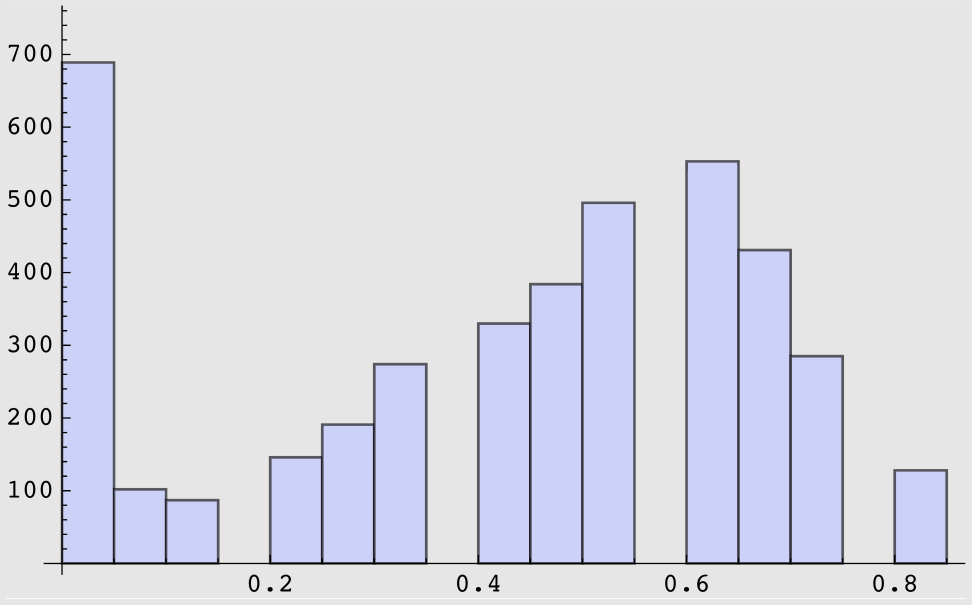
r.m.s. contrast can be calculated as:

```
Sqrt[Variance[Flatten[face]]] / Mean[Flatten[face]]
```

```
0.600059
```

### Histograms

```
Histogram[Flatten[face]]
```



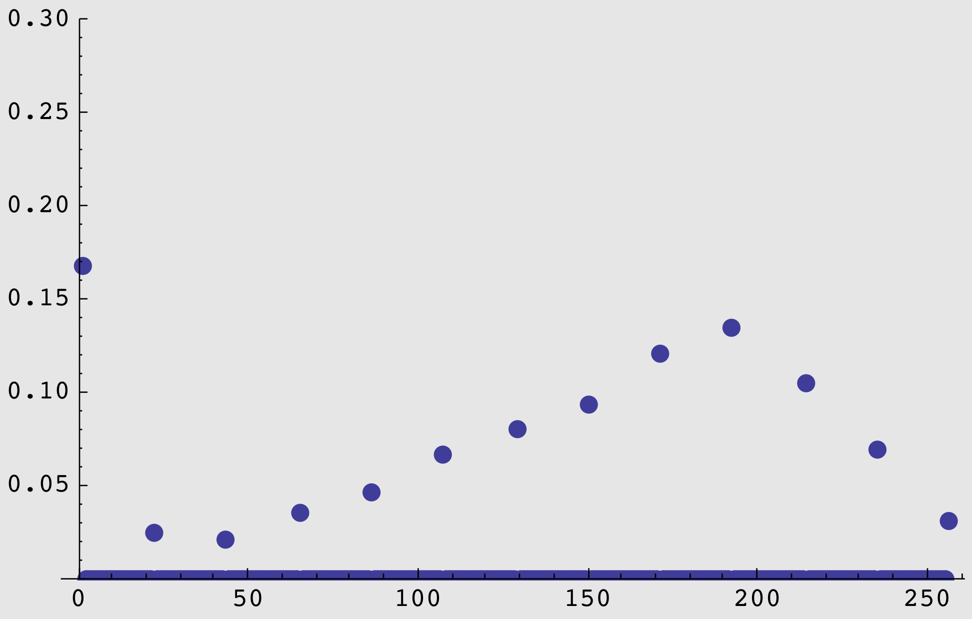
You can tell that the image is quantized at a coarse level (less than 4 bits).

Alternatively, you could calculate the histogram with more basic functions. To do the pattern match below, the floating point numbers are first converted to integers using **Round[]**:

```
domain = Range[0, 255];
Freq = Map[Count[Round[Flatten[face256]], #] &, domain];
```

If we normalize the histogram so that the sum is one, then we have a probability:

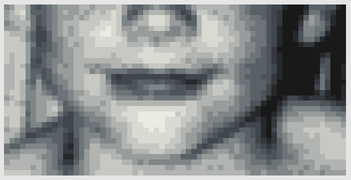
```
ListPlot[Freq / Plus @@ Freq, PlotStyle → PointSize[0.02], PlotRange → {0, .3}]
```



## Getting regions of images

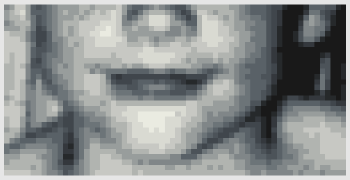
Three ways to pull

```
ArrayPlot[Take[face256, {1, 32}, {1, 64}]]
```



Or  to get rows 1 to 32 and columns 1 to 64 you can use:

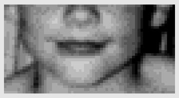
```
ArrayPlot[face256[[1 ;; 32, 1 ;; 64]]]
```



We can also use the built-in image function ImageTake[] to get rows 32 through 64:

```
ImageTake[
```

```
, {32, 64}]
```



**Use Manipulate[] and ImageTake[] to make an interactive selection tool for picking out rectangular regions of arbitrary size**

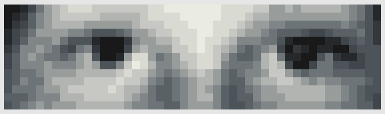
## ■ Getting coordinates by hand

```
ArrayPlot[face256, PixelConstrained → {1, 1}]
```

Click on the image above to select it. Now bring up the Drawing Tools, under the Graphics menu. Now use the “cross hairs” tool to select the first the lower left point{x0,y0}, and then

the upper right point {x1,y1} as the corners of the rectangular patch that you want. Copy and paste in the argument of the Round[] cell below. Here are coordinates for diagonal points for the eyes.

```
Round[{{6, 36.4}, {52.4, 48.4}}];
Reverse[Transpose[%]];
ArrayPlot[Take[face256, %[[1]], %[[2]]]]
```



### ■ And...finally

Click out the output of Image[]. This will bring up a whole set of image processing tools

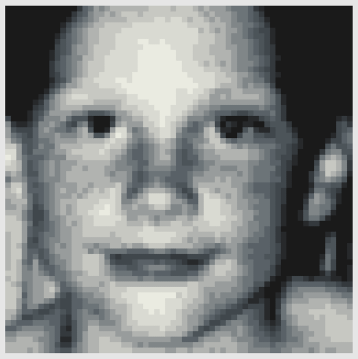
```
Image[
```

```
]
```

## Geometric image manipulations using function interpolation

Compare the plots with InterpolationOrder → 0 and InterpolationOrder → 1.

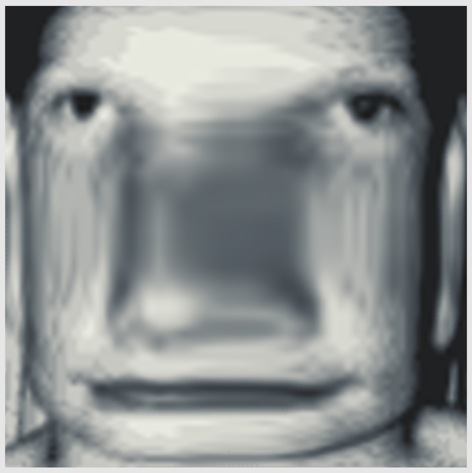
```
faceFunction = ListInterpolation[Transpose[face], {{-1, 1}, {-1, 1}},
   InterpolationOrder → 0];
DensityPlot[faceFunction[x, y], {x, -1, 1}, {y, -1, 1}, PlotPoints → 256,
 Mesh → False, AspectRatio → Automatic, Frame → None,
 ColorFunction → "GrayTones"]
```



## Morphing

```
faceFunction = ListInterpolation[Transpose[face], {{-1, 1}, {-1, 1}}];
```

```
DensityPlot[faceFunction[Sign[x] x^2, Sign[y] y^2], {x, -1, 1},
 {y, -1, 1}, PlotPoints → 100, Mesh → False, AspectRatio → Automatic,
 Frame → None, ColorFunction → "GrayTones", ImageSize → Small]
```



**FINAL project idea: Use the log polar model of the retinotopy of primary visual cortex to show how the**

**foveal representation gets expanded on cortex.**

## More filtering: Calculating the spatial gradient of an image using function interpolation

As we will see later, both first and second spatial derivatives have been used to model edge detection/amplification by neural processes in primary visual cortex. Here we show how to use symbolic derivatives to find the gradient.

The gradient of an image intensity function f, ∇f, has a maximum value in the direction of greatest change.

$$| \nabla f | = \left| \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y} \right) \right| = \sqrt{\left( \frac{\partial f}{\partial x} \right)^2, \left( \frac{\partial f}{\partial y} \right)^2} \tag{1}$$

Let filterface =f, where we've blurred out face a little to reduce quantization artifacts:

```
kernel = {{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}; (*Alternatively,
you could use BoxFilter[]*)
filterface = ListConvolve[kernel, face];
```

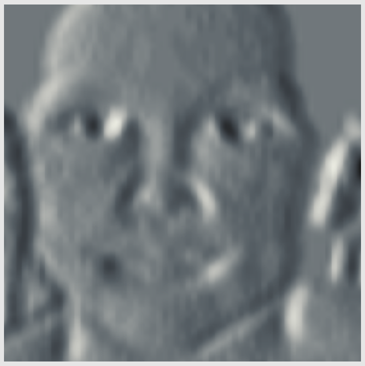
```
faceFunction = ListInterpolation[Transpose[filterface],
   {{-1, 1}, {-1, 1}}];
```

```
nx[x_, y_] := Evaluate[D[faceFunction[x, y], x]];
ny[x_, y_] := Evaluate[D[faceFunction[x, y], y]];

ImageGradient[x_, y_] :=
  Evaluate[Sqrt[D[nx[x, y], x]^2 + D[ny[x, y], y]^2] ];
```

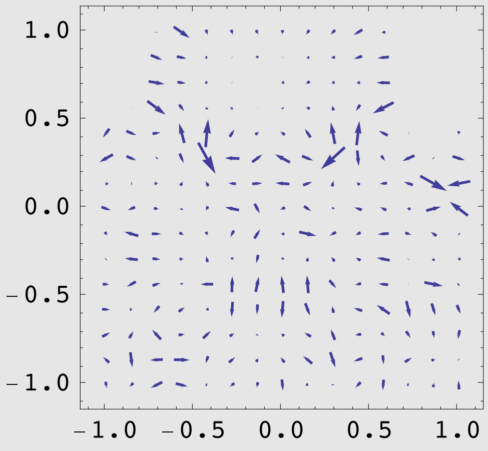
Plot the rate of change in the x-direction:

```
temp = Table[nx[x, y], {x, -1, 1, .005}, {y, -1, 1, .005}];
ArrayPlot[Transpose[temp]]
```



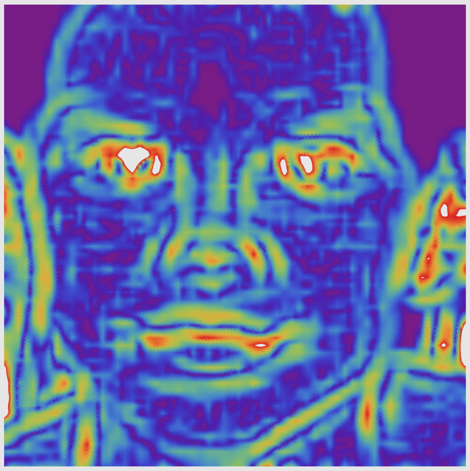
You can use VectorPlot to visualize the gradient field:

```
VectorPlot[{nx[x, y], ny[x, y]}, {x, -1, 1}, {y, -1, 1},
 ImageSize → Small]
```



Plot the magnitude of the gradient to highlight regions of the image where contrast is changing the most rapidly:

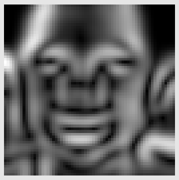
```
DensityPlot[ImageGradient[x, y], {x, -1, 1}, {y, -1, 1},
 PlotPoints → width, Mesh → False, Frame → False, ColorFunction → "Rainbow",
 ImageSize → Small]
```



### ■ Discrete spatial derivative filtering with built-in image functions

Use can also use image specific filters. **GradientFilter[**image,r**]** gives an image corresponding to the magnitude of the gradient of image, using discrete derivatives of a Gaussian of pixel radius r.

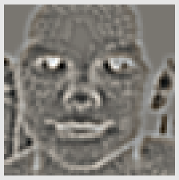
```
GradientFilter[ , 6] // ImageAdjust
```



The $\nabla^2 G$ operator in the previous lecture first convolves the image with a Gaussian blur kernel, and then takes the Laplacian $\nabla^2$.

```
LaplacianGaussianFilter[
```

```
, 2] // ImageAdjust
```



## Manipulating color images

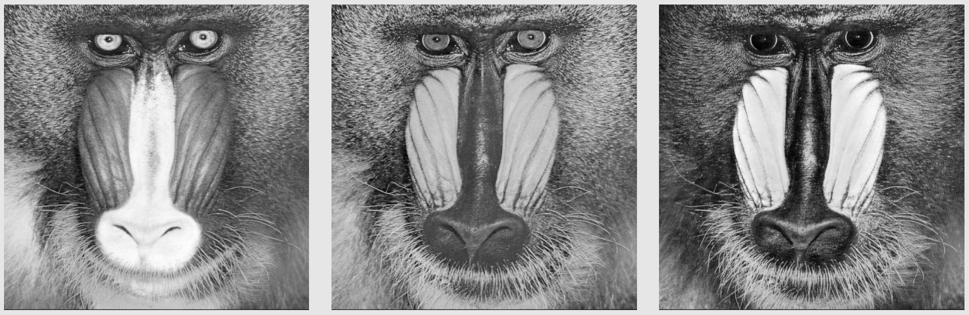
```
image = ExampleData[{"TestImage", "Mandrill"}]
```

```
RGBvalues = ImageData[image];
Dimensions[RGBvalues]
```

```
{512, 512, 3}
```

```
reds = Map[#[[1]] &, N[RGBvalues], {2}];
greens = Map[#[[2]] &, N[RGBvalues], {2}];
blues = Map[#[[3]] &, N[RGBvalues], {2}];
```

```
GraphicsRow[{Image[reds], Image[greens], Image[blues]}]
```



You can also use the function:

```
ColorSeparate[
```

```
];
```

- **A weighted average of RGB values to produce a luminance image:**

```
grayscale = Map[(0.3 #[[1]] + 0.59 #[[2]] + 0.11 #[[3]])/255 &, N[RGBvalues], {2}];
Image[grayscale] // ImageAdjust ;
```

Note the weights above are arbitrary, and the chosen values will depend on the color calibration.

- **Putting the R, G, B images back together:**

```
r = reds;
g = greens;
b = blues;
temp2 = Partition[Transpose[{Flatten[r], Flatten[g], Flatten[b]}],
   Dimensions[r][[2]]];
Image[temp2]
```

## Next time

Efficient coding

## References

Adelson, E. H., Simoncelli, E., & Hingorani, R. (1987). *Orthogonal Pyramid Transforms for Image Coding*. Paper presented at the Proc. SPIE - Visual Communication & Image Proc. II, Cambridge, MA.

Barlow, H. B., & Olshausen, B. A. (2004). Convergent evidence for the visual analysis of optic flow through anisotropic attenuation of high spatial frequencies. *J Vis, 4*(6), 415-426.

Daugman, J. G. (1988). An information-theoretic view of analog representation in striate cortex, *Computational Neuroscience*. Cambridge, Massachusetts: M.I.T. Press.

Engel, S. A., Glover, G. H., & Wandell, B. A. (1997). Retinotopic organization in human visual cortex and the spatial precision of functional MRI. *Cereb Cortex, 7*(2), 181-192.

Gold, J. M., Murray, R. F., Bennett, P. J., & Sekuler, A. B. (2000). Deriving behavioural receptive fields for visually completed contours. *Curr Biol, 10*(11), 663-666.

Hegdé, J., & Kersten, D. (2010). A Link between Visual Disambiguation and Visual Memory. *Journal of Neuroscience*, *30*(45), 15124–15133. doi:10.1523/JNEUROSCI.4415-09.2010

Konishi, S. M., Yuille, A. L., Coughlan, J. M., & Zhu, S. C. (2003). Statistical edge detection: Learning and evaluating edge cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 25*(1), 57-74.

Olman, C. A., & Kersten, D. (2004). Classification objects, ideal observers & generative models. *Cognitive Science, 28*, 227-239.

Moore, C., & Engel, S. A. (2001). Neural response to perception of volume in the lateral occipital complex. *Neuron, 29*(1), 277-286.

Schwartz, E. L. (1980). A quantitative model of the functional architecture of human striate cortex with application to visual illusion and cortical texture analysis. *Biol Cybern, 37*(2), 63-76.

http://library.wolfram.com/howtos/images/#histograms