

# Computational Vision

U. Minn. Psy 5036  
Daniel Kersten  
Lecture 11: Image Coding

## Initialize

### ■ Read in Statistical Add-in packages:

```
In[20]:= Off[General::"spell1"];  
SetOptions[ArrayPlot, ColorFunction -> "GrayTones", DataReversed -> False,  
Frame -> False, AspectRatio -> Automatic, Mesh -> False,  
PixelConstrained -> True, ImageSize -> Small];
```

---

## Outline

### Last time

Image operations

Point non-linearities, and simple statistics

Using function interpolation for image morphing, symbolic differentiation.

### Today

Natural image statistics and efficient coding

Understanding first-order statistics in terms of efficient coding of natural images

---

## Efficient coding: 1st order statistics & point operations

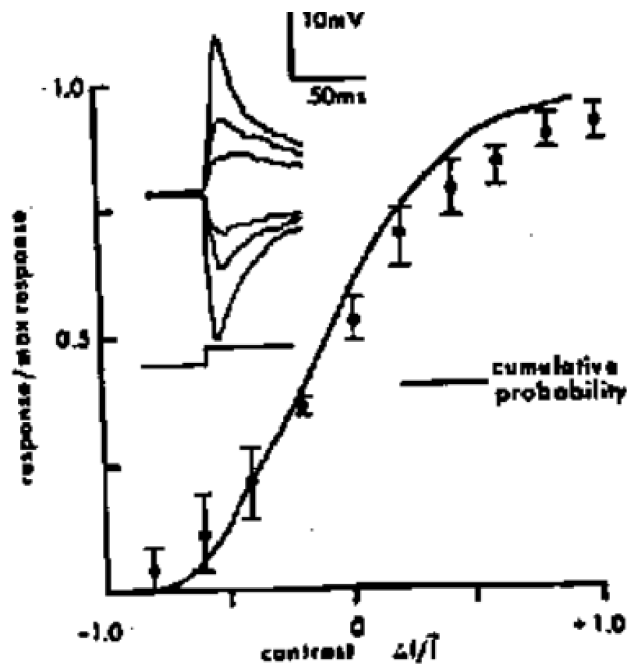
In 1981, Simon Laughlin published a paper in which he showed that the contrast response function of LMC interneurons ("large monopolar cells") of the fly's ommatidium had a sigmoidal non-linearity as shown below. This kind of non-linearity wasn't new, and is common, especially in sensory response functions. It is a de facto standard point non-linearity

in generic neural network models.

But why the sigmoidal shape near the sensory input? The mechanistic explanation for this kind of sigmoidal shape was and has been that small signals tend to get suppressed (e.g. a "soft" threshold), and signals get saturated because of biophysical limitations at the high end.

Laughlin came up with a different kind of answer, based on a functional argument that went along the following lines: the fly lives in a visual world in which big contrasts (local intensity relative to a global mean) (negative or positive) are less common than contrasts near zero, so neurons should devote more of their resolving capacity to the middle contrasts, i.e. those near zero. This kind of argument is based on information theory. In this lecture, we'll develop some basic tools of information theory to understand his model and others like it.

In image processing jargon, the fly's visual neuron is doing "histogram equalization". Let's see what that means.



**Fig. 2.** The contrast-response function of light adapted fly LMC's compared to the cumulative probability function for natural contrasts ( $50^\circ$  interval). Response is normalized with the maximum amplitude to light off as 0.0, and the maximum amplitude to an increment as 1.0. Data points averaged from 6 cells; range bars show total scatter. Inset

## Image calibration

In the next lectures we are going to study statistical regularities found in natural images. Efficient coding takes advantage of regularities to represent images under some specified optimality constraint, for example with fewer "bits".

Precise models of efficient coding depend on carefully calibrated image datasets. Depending on the kind of compression (e.g. "lossy"), images like jpeg images have statistical properties that can deviate from the original uncompressed image. There exist several carefully calibrated image data sets, such as the "van Hateren" database: <http://www.kyb.tuebingen.mpg.de/?id=227>. But the regularities and the means to exploit them efficiently can be illustrated with arbitrary images that haven't been carefully photometrically calibrated.

## Read image file

In[22]:=

```
graniteg = ImageData [
```



```
];
```

To manipulate pixel graylevel values, we will do a weighted average of the RGB channels to produce a grayscale image, quantized to 256 levels, called **granite**:

In[23]:=

```
mg = Max [graniteg];
granite =
  Round [255 * Map [
    
$$\frac{0.30 \#[[1]] + 0.59 \#[[2]] + 0.11 \#[[3]]}{mg}$$

    &, graniteg, {2}]]];
(*ArrayPlot is one way to display*)
ArrayPlot [granite];
```

Calculate the max, mean and standard deviation. The standard deviation gives us a measure of contrastiness, without normalization by the mean (see previous lecture's definitions of contrast).

By displaying using Image[], *Mathematica* provides the option of getting information about the image (e.g. histogram), editing the image, and filtering it. (Click on the image, then “i”, or “more”.)

In[26]:=

```
{width = Dimensions [granite] [[1]], mgranite = Max [Flatten [granite]],
 N [Mean [Flatten [granite]]], N [StandardDeviation [Flatten [granite]]]}

(*Image[] expects 0 to map to black, and 1 to white *)
Image [granite / N [mgranite], Magnification -> .5]
```

Out[26]=

```
{256, 255, 128.657, 46.4504}
```

Out[27]=



## Histogram

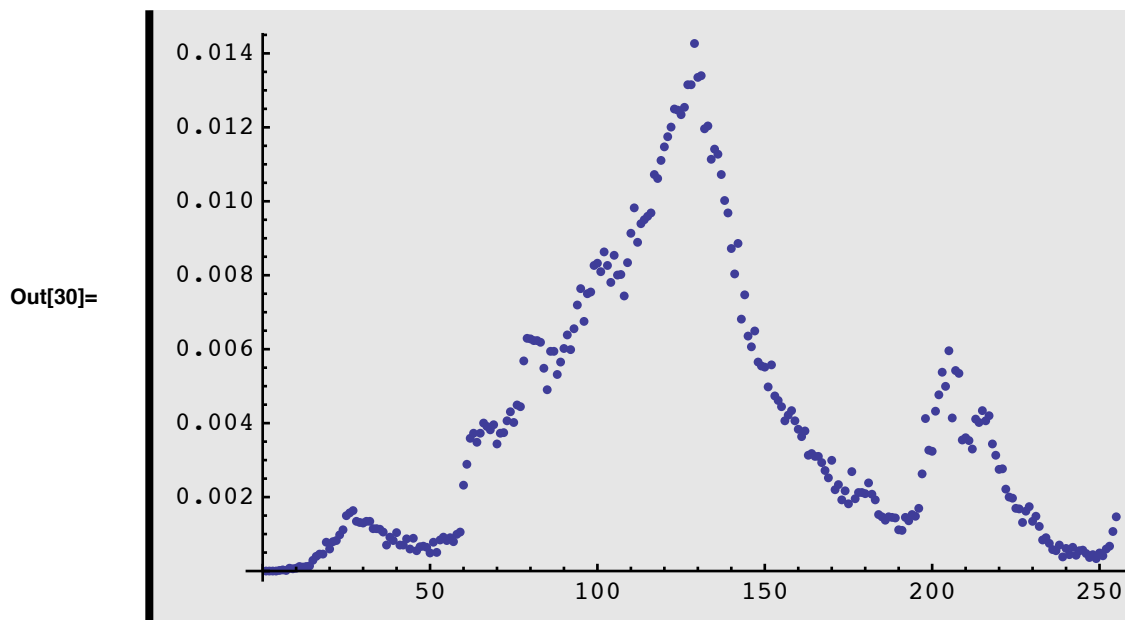
Let's calculate some first-order statistics. First-order means that we are looking at the frequency of occurrence of pixel intensity values, i.e. the histogram. In the next lecture we'll look at second-order statistics that describe how pixel intensities at one location are related to other ones.

- Define a histogram function that accepts an image with graylevels [lowlimt,highlimit], and outputs a histogram normalized to 1:

```
In[28]:= histogram[image_, binsize_, lowlimit_, highlimit_] := Module[{histx},
  histx = BinCounts[Flatten[image], {lowlimit, highlimit, binsize}];
  Return[N[histx / Plus@@histx]];
];
```

The histogram is normalized so gives us an estimate of  $p_i$ , the probability of the  $i$ th intensity, e.g. the probability of a pixel's value being say graylevel 103.

```
In[29]:= histogramimage = histogram[granite, 1, 0, 255];
ListPlot[histogramimage, PlotStyle -> PointSize[0.01]]
```



You can also use built-in histogram functions:

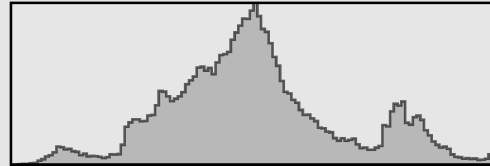
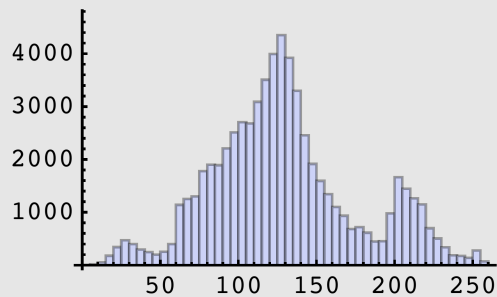
In[31]:=

```
GraphicsRow[ { Histogram[Flatten[granite], ImageSize -> Small],
```

```
ImageHistogram[ granite ] ] ] ]
```



Out[31]=



**What is the frequency of occurrence of graylevel 103?**

**Confirm that the histogram values add up to 1**

## Entropy: a measure of information

Entropy provides a scalar measure of the degree of disorder in an image. "Disorder" isn't necessarily bad because it is intuitively related to the degree of novelty or surprise in a pattern. We want a measure of information that reflects degree of surprise in a formal objective sense. Surprise in a subjective sense is trickier, because high objective entropy can also mean boring.

A highly ordered or regular pattern is by definition somewhat predictable--there is a "pattern there". You might be able to predict one part of the pattern from another. Think of pixels making up a gradual ramp in intensity. The pattern is said to have redundant information. A pattern can also be predictable because certain elements are just more likely to occur than others. For example, some gray values in the above histogram are more frequent than other gray values. Or think of pixels making up a uniform gray square.

A highly probable event doesn't convey as much information as a low probability event. "I will go to sleep tonight" vs. "I will parachute tonight". The second statement conveys more information than the first, because the a priori probability is almost 1, so changes little with the new information.

An event for us will be a pixel taking on a particular graylevel value, say  $i$ , where we have a space of 256 possible events.

### ■ Communication and information transmission

We also think of an "event" in this context as a member (a symbol) of an "alphabet" used to send information about a signal. So to send information about a picture, I send a series of symbols (each one corresponding to  $i$ , where  $i = 0, 1, \dots, 255$ ) from the alphabet  $\{0, 1, \dots, 255\}$ .

Intuitively the degree of informativeness is inversely related to the probability of the event. Any monotonic function (such as a logarithm) of the reciprocal of probability would preserve a measure of informativeness.

In a classic piece of work done at Bell Labs in the late 1940s, Claude Shannon defined the information of an event  $i$  as:

$$-\text{Log}_2 p_i = \text{Log}_2 1 / p_i$$

This formula satisfies at least one requirement for a measure of information, that it should be monotonically related to the degree of surprise.

The base 2 corresponds to a unit of information called a "bit", which thanks to computers is now part of common parlance.

```
In[33]:= Log[2, 1 / histogramimage[[127]]]
Log[2, 1 / histogramimage[[5]]]
```

```
Out[33]= 6.24857
```

```
Out[34]= 15.9984
```

But we have a whole collection of symbols making up the alphabet, so it is also useful to have a measure of the average information for the set of symbols given their respective probabilities. Some alphabets may be better than others at efficiently sending information about signals.

Shannon went on to define *entropy* as the average (expectation) information over all  $N$  events,

So by the definition of expectation (or average):

$$entropy = -\sum_{i=1}^N p_i \text{Log}_2 p_i \quad (1)$$

If a pixel can take on any graylevel with equal probability, like the noise images you have generated, the entropy is high (and in fact maximum). On the other hand, if pixels can take on only one value, entropy is low (and in fact 0). (Entropy for a continuous random variable is different-- the entropy doesn't have a natural lower bound, and the minimum can be negative)

Information and entropy are measured in bits for Log base 2. Again, recall that an event for us is a particular graylevel.

Entropy reflects the amount of information conveyed on average by a set of symbols. A 256x256 gaussian white noise image may look boring and unsurprising to you, but a true sample is actually completely novel--you've never seen it before, and you'll never see it again (or at least the odds are vanishingly low).

Sometimes probabilities are 0, so we set  $0 \text{Log}(0) = 0$ , and then the following function calculates entropy for a list of probabilities:

```
In[35]:= entropy[probdist_] := Plus@@ (If[# == 0, 0.0, -# Log[2, #]] & /@probdist)
```

**Plus@@list** is short-hand for **Apply[Plus, list]**. They both return the sum of the elements in the **list**.

**If[#==0,0,-# Log[2,#]]&** is short-hand for a function to be applied to elements of our list. **#** is a placeholder for the variable that gets plugged in. We could have defined a function: **information[p\_]:=If[p==0,0,-p Log[2,p]]**;

**/@** is short-hand for **Map[]**. I.e. we could have done: **Apply[Plus, Map[information, probdist]]**, or even longer, we could have written a loop.

We can use our entropy function to verify that the entropy is biggest when the probability distribution is uniform, i.e. when  $p_i=1/N$ . So the maximum entropy for our graylevel pictures would be: **entropy[Table[1/256, {256}]] = 8** bits.

---

### Verify this and calculate: `entropy[Table[1/256, {256}]]`

---

Let's calculate the "first-order" entropy of our grayalpine image:

```
In[36]:= entropy[histogramimage]
```

```
Out[36]:= 7.41789
```

*Mathematica's* built-in function `Entropy[]` calculates entropy from a list, effectively calculating the histogram for you:

```
In[37]:= N[Entropy[2, Flatten[granite]]]
```

```
Out[37]:= 7.42209
```

### Find out why the two entropy definitions don't give exactly the same answer

---

Use *Mathematica's* function `Entropy[2, <list>]` to compare the entropy of `{a,b,c,d}` with `{a,a,c,d}`, `{a,a,a,d}`, etc.. Without doing another calculation, what is the entropy of `{2,2,15,4}`?

---

## Histogram equalization

Suppose we have a set of symbols (e.g. intensity levels called `{g}`) used to represent an incoming signal. And that some levels are much more common than others.

Can we remap these symbols to a new set in which the new symbols (a new set of levels `{f}`) are equally probable? I.e. so that entropy is maximum?

### ■ Theory

In the subsequent analysis, we will initially treat images as having continuous, rather than discrete intensity values. (When we calculate entropy, we'll first bin the intensities into 256 bins). Consider for the moment, a large set of images `{g(x,y)}` with continuous intensity values. Suppose that the distribution of intensities was truly gaussian with a particular mean graylevel and standard deviation:

$$p_g(g) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(g-\mu)^2}.$$

If we map  $f = \phi(g)$ , we'd get a new set of images `{f(x,y)}`. What would the histogram look like over this set? The answer comes from the density mapping theorem. (See the Probability Overview in Lecture 5) Let's call the new density of  $f$ :  $p_f(f)$ . Assume  $\phi()$  is monotonic. The fundamental idea is that probability mass should be conserved,

$$p_f(f)\Delta f \approx p_g(g)\Delta g. \tag{2}$$

What if we want  $p_f(f) = \epsilon$ , a constant over some domain?

$\text{cdf} = p_g(g) dg$ , then we integrate both sides to get: (3)

$$f(g) \propto \int_{-\infty}^g p_g(g') dg' \quad (4)$$

Note that  $f$  is exactly proportional to the function given by the cumulative distribution.

In other words, if we draw a gaussian number,  $g$ , and run it through the point-wise non-linearity given by the formula for the cumulative gaussian,

$$f(g) = \frac{1}{2} \left( 1 + \text{Erf} \left[ \frac{g-\mu}{\sqrt{2} \sigma} \right] \right) \quad (5)$$

$f(g)$  will be uniformly distributed. Thus the function  $f$  gives us the mapping rule to convert a non-uniform histogram to a uniform one, i.e. to do "histogram equalization" for an image.

(Note that if we use the inverse  $f^{-1}$ , we have the means to generate gaussian random variables from uniformly distributed ones. See the Exercises section of the ProbabiltyOverview.nb)

### ■ Synthetic images: Gaussian white noise (discretized & clipped)

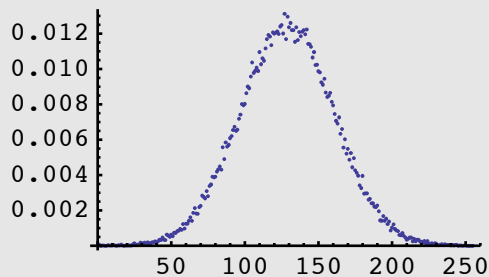
Recall that the "white" in "white noise" means that the pixel intensities are independent of each other. We'll make this notion precise in the next lecture when we learn about 2nd order statistics in images. For our purposes here, it just means we randomly draw samples for each pixel independent of what any of the previous draws were.

Because a display graylevel is in the range  $[0,255]$ , we clip the extreme samples using **Which[]** in the sampling function **randomguess**:

In[38]:=

```
sigma = 32;
ndist = NormalDistribution[128, sigma];
randomgauss := Which[(r = Random[ndist]) > 255, 1, r < 0, 0, True, r];
gaussimage = Table[randomgauss, {i, 1, 256}, {j, 1, 256}];
histogaussimage = histogram[gaussimage, 1, 0, 255];
ListPlot[histogaussimage, PlotStyle -> PointSize[0.01], ImageSize -> Small]
entropy[histogaussimage]
```

Out[43]=



Out[44]=

7.04385



## How does the entropy change if the standard deviation is smaller, say 8, instead of 32?

### ■ Non-linearity

The cumulative gaussian is the probability of an intensity being lower than x. The cumulative gives us the form for the non-linearity (i.e.  $f=y$ ):

```
In[45]:= 
$$y[x_, \mu_, \sigma_] := 256 * \frac{1}{2} \left( 1 + \text{Erf} \left[ \frac{x - \mu}{\sqrt{2} \sigma} \right] \right);$$

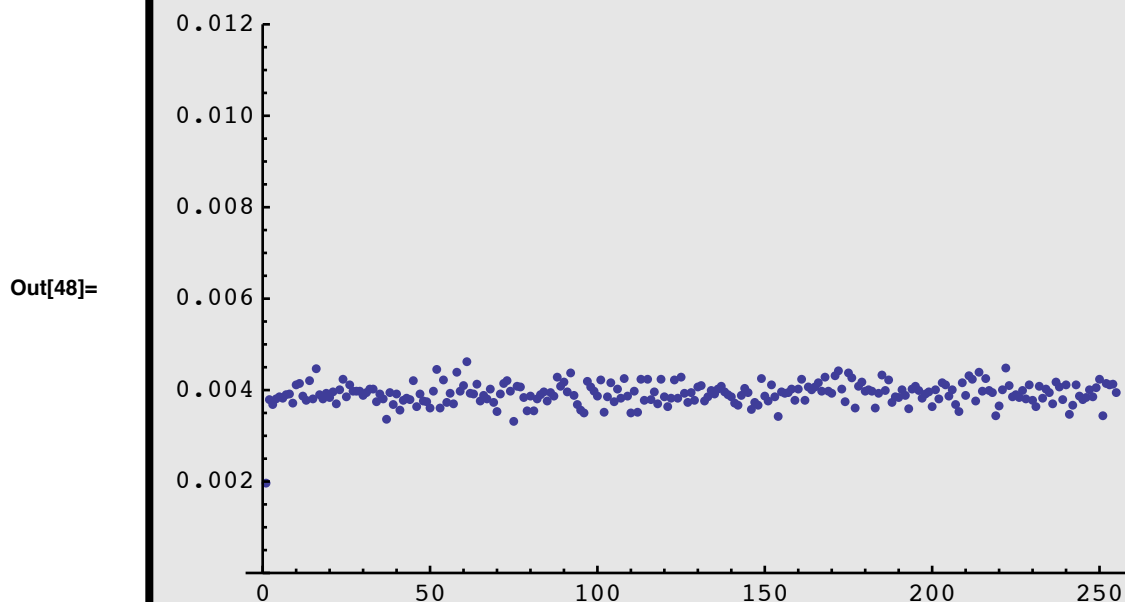
```

(Recall that *Mathematica* has a built-in add-on function for the cumulative gaussian, so one could also use that.)

If the standard deviation exactly matches that of the gaussian white noise parameters, we can make **newgaussimage** by running the old pixel values through the (sigmoidal point) non-linearity specified by **y[]** and produce a new image with a uniform white noise spectrum:

```
In[46]:= newgaussimage = Round[y[gaussimage, 128, 32]];
```

```
In[47]:= histogaussimage = histogram[newgaussimage, 1, 0, 255];
ListPlot[histogaussimage, PlotStyle -> PointSize[0.01],
PlotRange -> {0, 0.012}, ImageSize -> Medium]
entropy[histogaussimage]
```



```
Out[49]= 7.99118
```

...now the entropy is near the maximum of  $\text{Log}[2,256] = 8$  bits.

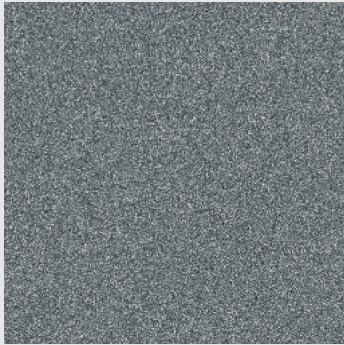
**Plot up gaussimage and newgaussimage. Which one appears "contrastier"? Which one has the greater**

standard deviation?

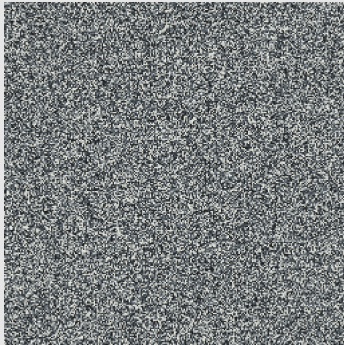
In[72]:=

```
ArrayPlot[gaussimage]
ArrayPlot[newgaussimage]
```

Out[72]=



Out[73]=



```
Image[gaussimage] // ImageAdjust;
Image[newgaussimage] // ImageAdjust;
```

### ■ Equalize a natural image: "granite"

We'll first add a tiny bit of Gaussian noise to granite. This is a bit of a fudge, but arguably more realistic, and it gives us a larger intensity vocabulary.

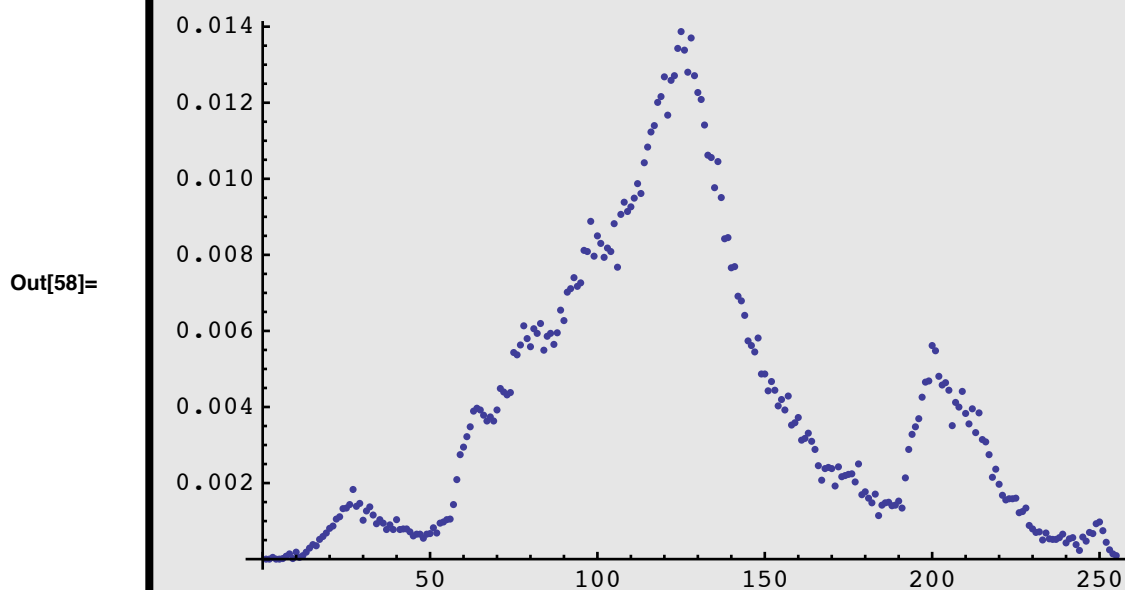
In[52]:=

```
Clear[randomgauss];
noise = Table[Random[NormalDistribution[0, 2]], {i, 1, width},
             {j, 1, width}];
```

```
In[54]:= granite = granite + noise;
granite = 255.0 * granite / Max[granite];
Max[granite]
```

```
Out[56]:= 255.
```

```
In[57]:= histogramgranite = histogram[granite, 1, 0, 255];
ListPlot[histogramgranite, PlotStyle -> PointSize[0.008]]
```



```
In[59]:= entropy[histogramgranite]
```

```
Out[59]:= 7.40978
```

Calculate the cumulative distribution from the histogram of granite:

```
In[60]:= cumulhistogramgranite =
256.0 * FoldList[Plus, histogramgranite[[1]], histogramgranite];
g1 = ListPlot[cumulhistogramgranite];
```

Use **ListInterpolation** to fit a continuous function to cumulative distribution, and plot up data with function fit.

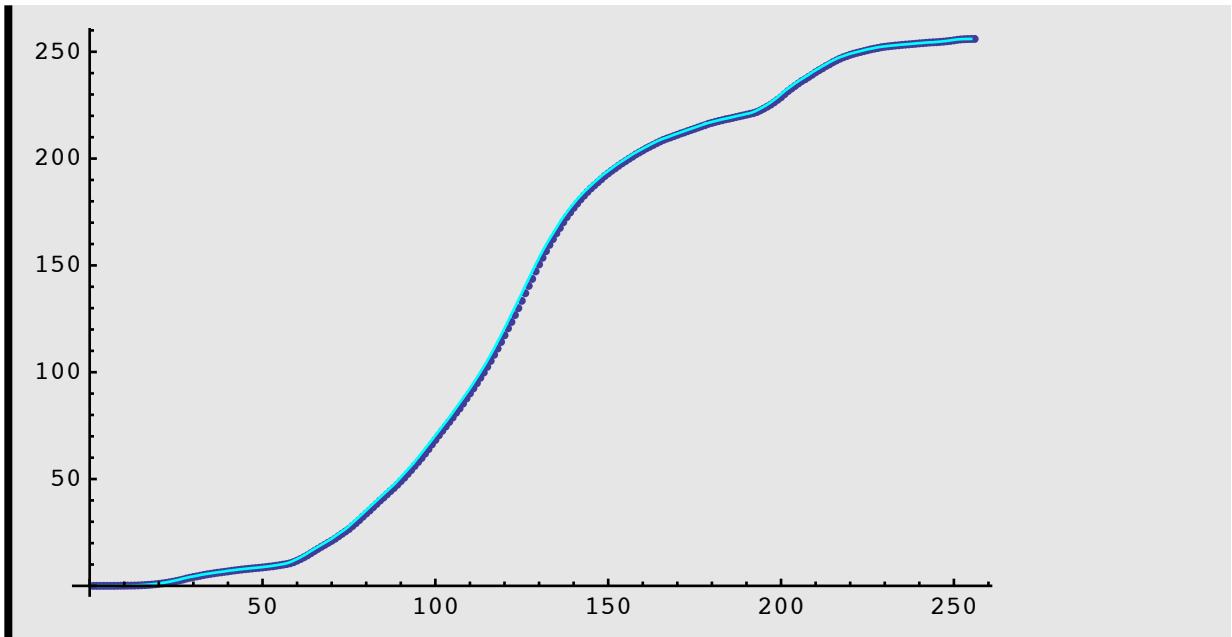
```
In[62]:= fcumulhistogramgranite = ListInterpolation[cumulhistogramgranite, {{0, 255}}];
```

```
In[63]:= g2 = Plot[fcumulhistogramgranite[x], {x, 0, 255}, PlotStyle -> Hue[0.5]];
```

In[64]:=

**Show[g1, g2]**

Out[64]=



Use cumulative function to re-map intensities:

In[76]:=

```
equalgranite = Round[Map[fcumulhistogranite, granite, {2}]];
ArrayPlot[equalgranite, Mesh → False, PlotRange → {0, 255},
ImageSize → Medium]
```

InterpolatingFunction::dmval :

Input value {-0.211948} lies outside the range of data in  
the interpolating function. Extrapolation will be used. >>

Out[77]=



As we'd expect, the histogram equalization increases apparent contrast as compared with the original image:

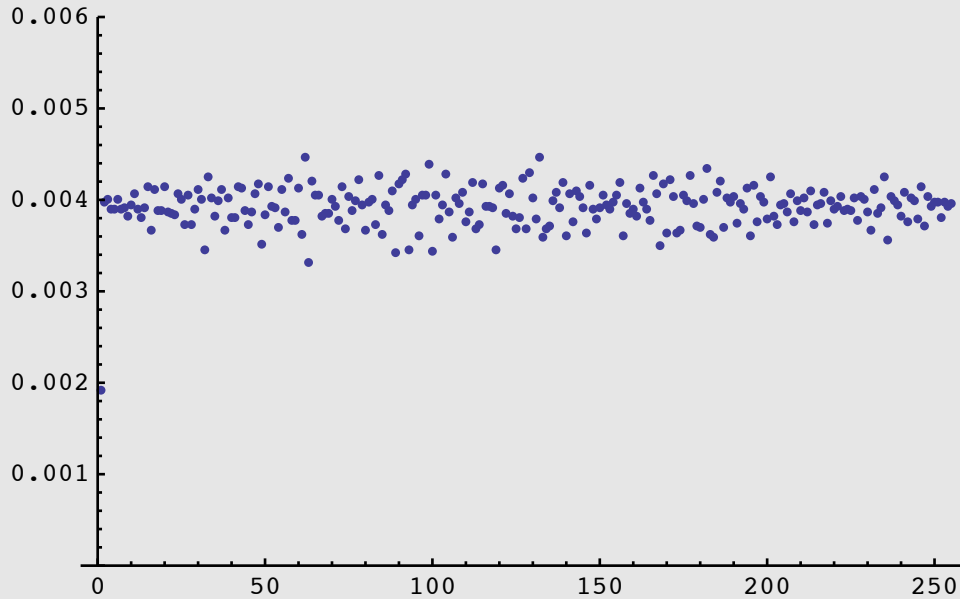
**Compare above image with original: ArrayPlot[granite,Mesh→False,PlotRange→{0,255}];**

Check final histogram and entropy:

In[78]:=

```
histoequalgranite = histogram[equalgranite, 1, 0, 255];
ListPlot[histoequalgranite, PlotStyle -> PointSize[0.01`],
PlotRange -> {0, 0.006}]
```

Out[79]=



In[69]:=

```
entropy[histogranite]
entropy[histoequalgranite]
```

Out[69]=

7.40978

Out[70]=

7.99162

(If we had kept our 256 gray-level intensity "vocabulary", the non-linearity leaves out lots of gray-levels in the output. Contrast would still be enhanced, but the mapping can actually reduce entropy. A one-to-one re-mapping of labels shouldn't affect entropy at all.)

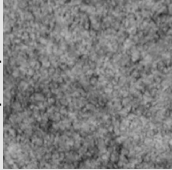
### ■ Does histogram equalization improve image quality?

There used to be a fair amount of research investigating how various forms of histogram equalization might be used to improve the visibility of important information in degraded images, such as medical radiograms. The short answer, is that histogram equalization does little to help.

## Histogram equalization by the fly's eye

Let's return to the sigmoidal non-linearity of photoreceptors found in the fly (and other animals). Laughlin argued that a sigmoid non-linearity is a good design feature to efficiently code the range of light typically found in an image (Laughlin,

1981; Richards, 1981; see figure below). The idea is that if you made a plot of the distribution of light intensities in a typical scene about some mean level, you might find something like the gaussian distribution shown below. The granite picture intensity histogram wasn't gaussian at all, but it was roughly more peaked in the middle range. Another image, actually of granite, is closer to gaussian :

```
Histogram[Flatten[ImageData[], ImageSize -> Small];
```

Even if the intensity values are continuous, the effective resolution of intensity may be limited by a constant amount of noise, reducing the effective response discriminability (like quantization). If you wish to divide up the response range efficiently, it would make sense to increase the input resolution over the range where you are likely to encounter the more frequently occurring intensities. Then the output values representing the extreme regions of the input become more probable relative to the output values coding the middle range of the input. Such a strategy could be implemented with a sigmoidal point mapping function as shown in the figure. In fact, the optimal non-linearity would correspond to the cumulative probability distribution that we developed above. Or in other words, the fly's light transducer should be performing "histogram equalization".

Laughlin was able to show a good correspondence between theory and actual photoreceptor non-linearity in the fly.

The coding using a non-linear point transformation improves the cell's information transmission capacity.

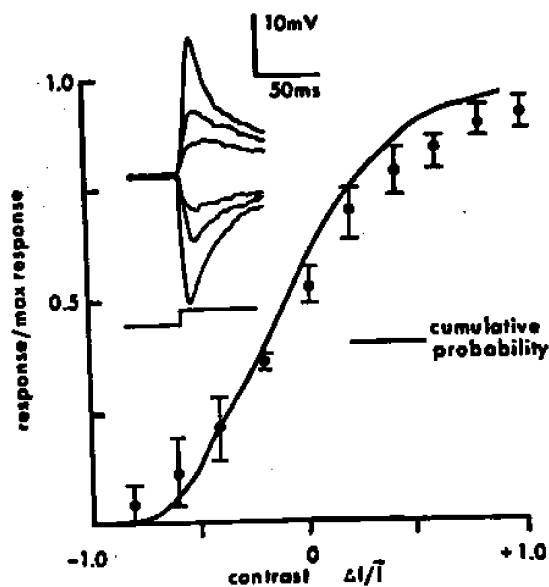


Fig. 2. The contrast-response function of light adapted fly LMC's compared to the cumulative probability function for natural contrasts (50° interval). Response is normalised with the maximum amplitude to light off as 0.0, and the maximum amplitude to an increment as 1.0. Data points averaged from 6 cells; range bars show total scatter. Inset

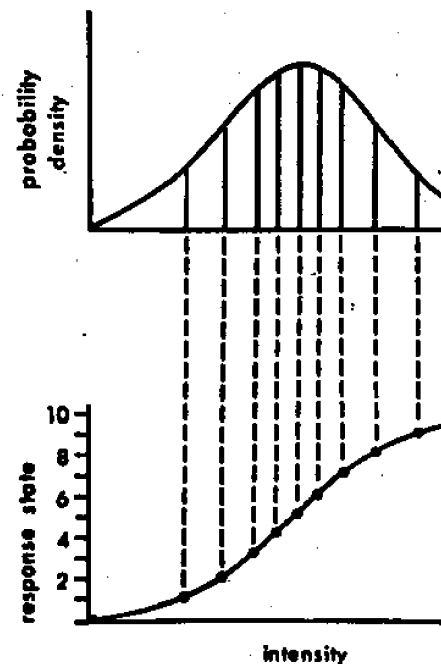


Fig. 1. The coding strategy for maximizing

Are there ways in which the retina could take advantage of other statistical regularities between the pixels in natural images to efficiently encode image information? Does lateral inhibition have an explanation in terms of efficient coding?

In the next lecture, we will see how information can be encoded even more efficiently by taking advantage of the regulari-

ties across space and time. To get a glimpse of where we are heading, let's calculate a simple statistic that measures a relationship between nearby pixel values.

## Visual coding by neural populations in V1: Are natural images special?

When Hubel and Wiesel began their seminal recordings of the responses of neurons in visual cortex to images, they made a very important observation: It was really hard to get a neuron to fire. They needed to show the neurons edges of just the right orientation and location (See: youtube).

Why are cortical V1 neurons usually so quiet?

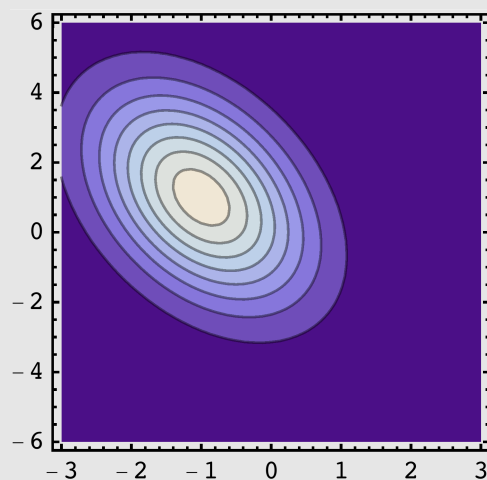
It may not seem obvious at first, but we can get a clue by calculating the histograms of the responses to linear difference filters to natural image input.

## Second order properties: Are images Gaussian?

If images were Gaussian, the joint distributions of pixel intensities could be well fit by:  $e^{-\frac{1}{2}(x-\mu)\Sigma^{-1}(x-\mu)}$ , where  $x$  is the array of pixel intensities, and  $\Sigma$  is the covariance matrix.

Suppose our the images in our collection have only 2 pixels. Here's one possible joint distribution, where the correlation is  $\rho$ :

```
 $\rho = -0.4;$ 
ContourPlot[PDF[MultinormalDistribution[{-1, 1}, {{1,  $\rho$  2}, { $\rho$  2, 4}}],
  {x, y}], {x, -3, 3}, {y, -6, 6}, PlotRange -> All, PlotPoints -> 25,
  ImageSize -> Small]
```

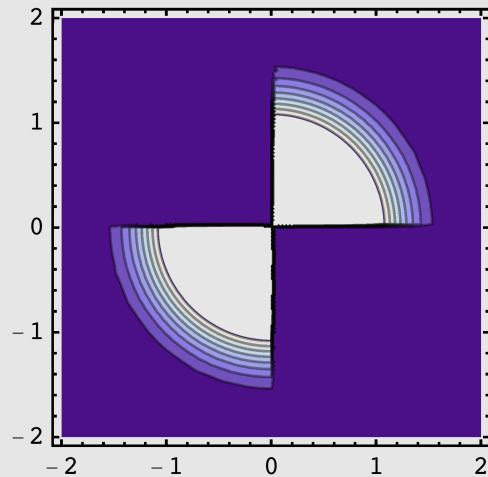


We've seen an example of a single image whose intensity histogram is not gaussian. But natural images are high-dimensional, and it is clear that nearby pixels are not independent. This raises the possibility that even if the first-order statistics, as represented in a histogram, were gaussian (as Laughlin observed in his particular images), the higher-order structure is



not gaussian. Here's an artificial example assuming another collection of 2-pixel images. The joint probability density of  $x$  and  $y$  is:

```
g3[x2_, y2_] := If[(x2 > 0.0 && y2 >= 0.0) || (x2 < 0.0 && y2 < 0.0),  
  (1 / (Pi)) * Exp[-.5 * (x2^2 + y2^2)^2], 0.0];  
ContourPlot[g3[x1, x2], {x1, -2, 2}, {x2, -2, 2}, ImageSize -> Small]
```



But if you calculate the "marginals", i.e. integrate out one of the variables, say  $y_2$ , the density is Gaussian. You can visualize this by imagining adding up all the values along columns in the above plot.

```
g3marginal = Table[NIntegrate[g3[x1, x2], {x1, -2, 2}], {x2, -2, 2, .1}];  
ListPlot[g3marginal];
```

### ■ Quick check on gaussianity for a natural image

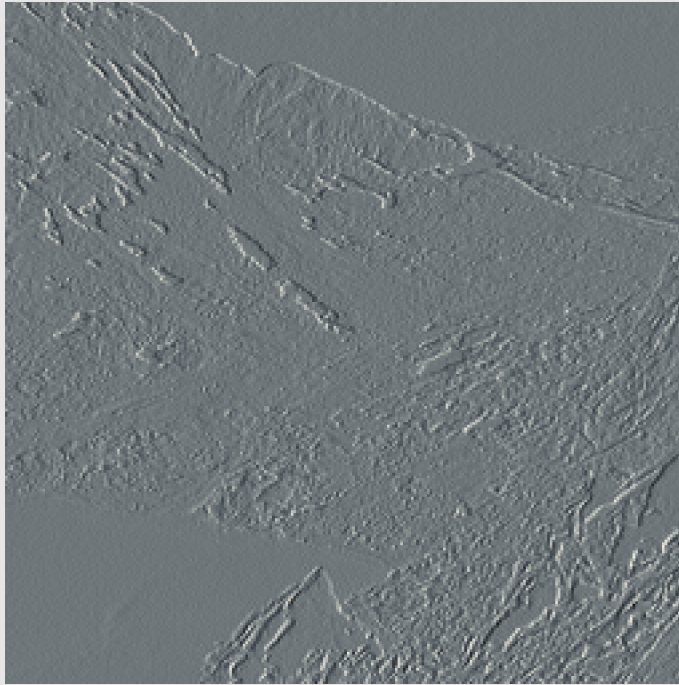
Here is a simple check on "gaussianity" you can apply to any natural image. It is a mathematical fact that the weighted sum (or difference) of two independently drawn Gaussian random variables is also a Gaussian. Thus if we compute a new image that is a linear combination of pixel intensities in a gaussian image (e.g. as in any convolution), the responses should have a gaussian histogram. Let's take a look at the histogram of a simple difference operator on the grayalpine image. Here is a discrete approximation of a short vertically oriented "edge detector", or first derivative:

```
In[81]:= kern = {{1, -1}}
```

```
Out[81]= {{1, -1}}
```

In[82]:=

```
diffgranite = ListConvolve[kern, granite];
ArrayPlot[diffgranite, ImageSize -> Medium]
```

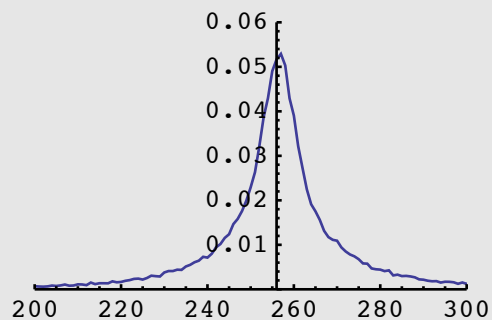


Out[83]=

In[84]:=

```
empiricalhistg = ListPlot[histogram[diffgranite, 1, -256, 255],
PlotRange -> {{200, 300}, {0, 0.06}}, Joined -> True, AxesOrigin -> {256, 0},
ImageSize -> Small]
```

Out[84]=



This histogram is strikingly regular (compared with the intensity histogram), with a fairly sharp peak and long tails. The histogram is said to have large "kurtosis".

You can try to fit this with a Gaussian, but you will discover that the tails of the distribution are too extended for a Gaussian.

It turns out that most natural images when filtered with a spatial filter whose area sums to zero (i.e. has as much positive as negative weight, as in the center-surround filters we've studied) produces a neural image with high kurtosis.

An interesting neural interpretation is that such a neural code produces sparse responses, i.e. there are a few units with

really big responses, but most responses huddle near the mean (which is zero). David Mumford has called this the "blue sky effect". But there is more to the story which you can read about in: Simoncelli, E. P., & Olshausen, B. A. (2001), and Simoncelli (1999, 2003).

The take-home message is that: *most natural images are non-Gaussian*.

### ■ Can the above empirical distribution of differences be fit by a Gaussian, a Laplacian?

In[85]:=

```
PDF[LaplaceDistribution[ $\nu$ ,  $\beta$ ], x]
```

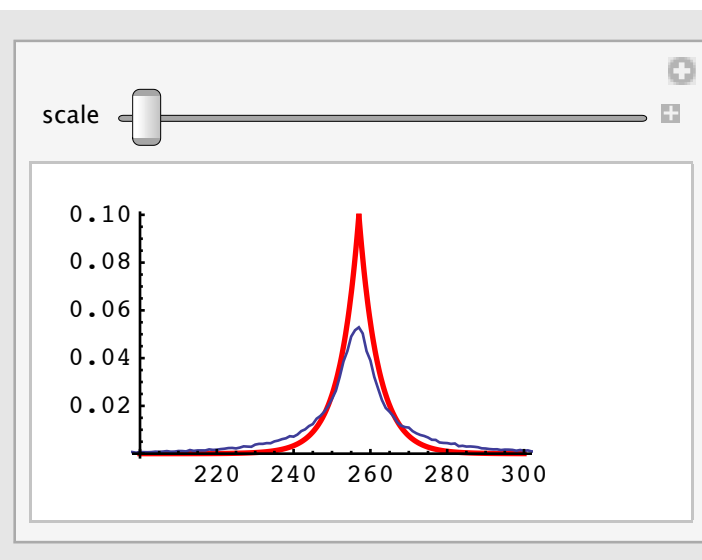
Out[85]=

$$\begin{cases} \frac{e^{-\frac{x-\nu}{\beta}}}{2\beta} & x \geq \nu \\ \frac{e^{\frac{-x+\nu}{\beta}}}{2\beta} & \text{True} \end{cases}$$

In[86]:=

```
Manipulate[
  theoreticalhistg = Plot[PDF[LaplaceDistribution[257, scale], x],
    {x, 200, 300}, PlotStyle -> {Red, Thick}, PlotRange -> Full];
  Show[{theoreticalhistg, empiricalhistg}, ImageSize -> Small],
  {scale, 5, 25}]
```

Out[86]=



If not, try the **generalized laplace distribution**, with a p value between 0.5 and 0.8 (Simoncelli, 1999):

```
In[87]:= pdf[x_, s_, p_] := Exp[-Abs[x / s]^p] / (Gamma[1 / p] * 2 * s / p)
pdf[x, s, p]
```

```
Out[88]= 
$$\frac{e^{-\text{Abs}\left[\frac{x}{s}\right]^p}}{2 s \text{Gamma}\left[\frac{1}{p}\right]}$$

```

**Plot the histogram for a center-surround filtered natural image**

---

```
In[89]:= kern = {{0, -1, 0}, {-1, 4, -1}, {0, -1, 0}};
diffgranite = ListConvolve[kern, granite];
```

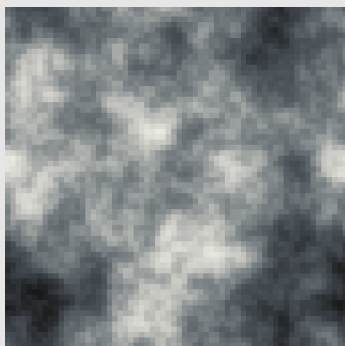
**Plot a difference histogram for a random gaussian fractal image**

---

The next lecture shows how to introduce simple correlations in an image using a fractal model.

■ **Initialize gaussianfractalimage by executing the following cell**

```
ArrayPlot[gaussianfractalimage, Mesh → False, Frame → False]
```



Even tho' the above image isn't white gaussian noise, the pixels are a reasonable sample from a gaussian. If indeed true, then a linear combination of its pixels should still be gaussian. Check this out using one of the above difference kernels.

---

## Next time

Efficient coding: 2nd order statistics and area-based operations

Lateral inhibition does "predictive coding"

Cortical wavelet-like basis sets: sparse, distributed representation that "decorrelates"

Color Trichromacy: decorrelation by principal components analysis

## Appendices

### Breaking an image into a series of subimages

- One can use `ImagePartition`, as illustrated in the *Mathematica* documentation:

```
ImagePartition[, 16] // Grid
```

- One can also use more basic functions as illustrated below.
- The input 64x64 image: face

Get dimensions of face

```
width = Dimensions[face][[1]]; hsize = width/2;
height = Dimensions[face][[2]];
(* Short[face,1 check out the first few lines*)
```

Scale so image ranges from 0 to 255.

```
 $\alpha = 255 / (\text{Max}[\text{face}] - \text{Min}[\text{face}]); \beta = -\alpha \text{Min}[\text{face}];$   
face256 =  $\alpha$  face +  $\beta$ ;
```

Scale face so that the average value is zero, and the r.m.s. contrast is 1:

```
tempm = Mean[Flatten[face]];  
tempsd = StandardDeviation[Flatten[face]];  
face = (face - tempm) / tempsd;
```

```
nregions = 4;  
swidth = width / nregions;
```

```
subface = Table[Take[face256, {i * swidth + 1, i * swidth + swidth},  
    {j * swidth + 1, j * swidth + swidth}], {i, 0, nregions - 1},  
    {j, 0, nregions - 1}];  
Dimensions[subface]
```

```
{Image[subface[[1, 1]]] // ImageAdjust,  
Image[subface[[1, 2]]] // ImageAdjust,  
Image[subface[[2, 1]]] // ImageAdjust,  
Image[subface[[2, 2]]] // ImageAdjust}
```



## ■ Using Raster[ ]

```
Show[Graphics[Raster[0.5 face]], AspectRatio -> 1, ImageSize -> Small]
```



## ■ Exporting images

```
Export["granite64x64.tif", granite, "TIFF"];
```

## ■ Find the positions of the Max and Min of a list:

```
argmax[x_] := Position[x, Max[x]] [[1, 1]];
argmin[x_] := Position[x, Min[x]] [[1, 1]];
```

## References

- Atick, J. J., & Redlich, A. N. (1990). Towards a theory of early visual processing. *Neural Computation*, 2(3), 308-320.
- Atick, J. J., & Redlich, A. N. (1992). What does the retina know about natural scenes? *Neural Computation*, 4(2), 196-210.
- Buchsbaum, G., & Gottschalk, A. (1983). Trichromacy, Opponent Colour Coding and Optimum Information Transmission in the Retina. *Proceedings of the Royal Society, London B*, 220, 89-113.
- Gopen, George D. & Swan, Judith A. The Science of Scientific Writing. *American Scientist*, 78, 550-558.
- Hacker, Diana. *A Pocket Style Manual 2nd Edition*. Bedford Books. Scientific American/St. Martin's College Publishing Group.
- Kersten, D. J. (1987). Predictability and Redundancy of Natural Images, *Journal of the Optical Society of America A*, 4, 2395-2400.
- Laughlin, S. B. (1981). A simple coding procedure enhances a neuron's information capacity, *Z. Naturforsch.*

Laughlin, S. B., de Ruyter van Steveninck, R. R., & Anderson, J. C. (1998). The metabolic cost of neural information. *Nat Neurosci*, 1(1), 36-41.

Linsker, R. (1990). Perceptual neural organization: some approaches based on network models and information theory. *Annual Review of Neuroscience*, 13, 257-281.

Olshausen, Bruno A., Field, David J. (2000) Vision and the coding of natural images. *American Scientist*, 88, 238-245.

Simoncelli, E. P., & Olshausen, B. A. (2001). Natural image statistics and neural representation. *Annu Rev Neurosci*, 24, 1193-1216.

Simoncelli, E. (1999, July, 1999). *Modeling the Joint Statistics of Images in the Wavelet Domain*. Paper presented at the Proc. SPIE 44th Annual Meeting, Denver, CO.

Simoncelli, E. P. (2003). Vision and the statistics of the visual environment. *Curr Opin Neurobiol*, 13(2), 144-149.

Srinivasan, M. V., Laughlin, S. B., & Dubs, A. (1982). Predictive coding: A fresh view of inhibition in the retina. *Proceedings of the Royal Society London B*, 216, 427-459.

van Hateren, J. H., & van der Schaaf, A. (1998). Independent component filters of natural images compared with simple cells in primary visual cortex. *Proc R Soc Lond B Biol Sci*, 265(1394), 359-366.

<http://www.cis.upenn.edu/~eero/ABSTRACTS/simoncelli90-abstract.html>

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

© 2008, 2010, 2013 Daniel Kersten, Computational Vision Lab, Department of Psychology, University of Minnesota.  
kersten.org