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

http://vision.psych.umn.edu/www/kersten-lab/courses/Psy5036/SyllabusF2000.html)

Initialize

- **Read in Statistical Add-in packages:**

```
Off[General::spell1];
<< Statistics`DescriptiveStatistics`
<< Statistics`DataManipulation`
<< Graphics`Graphics`
```

- **The input 64x64 image: face**

```
width = Dimensions[face][[1]]; hsize = width/2;
height = Dimensions[face][[2]];
Short[face,12]; (* check out the first few lines*)
```

Outline

Last time

Single-channel spatial filtering

Multiple channel filters

Psychophysical experiments.

->Multi-resolution, and wavelet bases

->A model of the spatial filtering properties of neurons in the primary visual cortex

Today

- **Upcoming dates:**

Mid-term in two weeks: Oct. 19th. Study-guide available by Tuesday next week.

3rd Assignment due week from next Tuesday. Will involve variations on today's exercises.

Final project outlines due: Nov. 14th.

- **Final projects**

- **Continue with multiresolution discussion**

- **Image manipulations**

Final projects

Proposal due Nov. 14th.

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

Will post final notebooks on the class web site.

Your "audience" will be your class peers.

Possible types of projects

Perceptual demonstrations

- **Motion illusions**
  e.g. stereograms, autostereograms, lightness illusions
  with interactive parameter variation

Visual psychophysics (quantitative measurements)

- **What does the eye see best?**

- **Data analysis/report of data collected elsewhere (by you)**

Computational models

- **Machine vision: but should have discussion/comparison of relevance to human vision.**

- **Orthogonal wavelet decomposition in Mathematica**

- **Neural network models**
  e.g. adaptive receptive field development, visual attention,

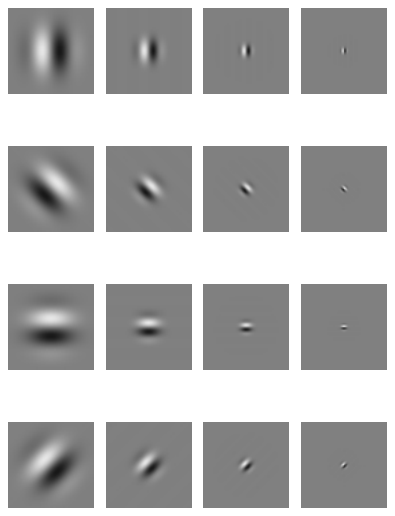
- **Models of human/biological vision**
  e.g. Principal components analysis and color trichromacy

- **Statistical analyses of images**
  e.g. Bayesian edge detector, correlational analyses, ...

Multiresolution cont'd

Bottom-line: image coding in terms of scale and orientation: Amodel 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.

The self-similar idea is important to vision because of the need for some kind of scale-invariance. 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).

Neural image? Or neural image representation?

We can view the response activities of a family of receptive fields of neurons as representing a filtered neural image of the input image. Although useful, this view can be misleading when we start to think about function, for "who is looking at the image"?

Alternatively, thinking in terms of basis functions gives us another perspective. We can view the response activities of a family of receptive fields as a representation of the input image. If linear, an activity is the result of a projection of an image on to a basis function (receptive field weights). Given such a representation we can begin to ask questions like:

1. Is the neural basis set complete? Can any image be represented?

2. A closely related question is: Is any information lost? I.e. we do the inverse transformation, can the original input be reconstructed?

3. Maybe the neural basis set is "over-complete"?

4. Are the neural basis functions orthogonal? Are they normal?

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?

-> 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

Image processing: Simple point manipuations

Point operations

■ Contrast

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

$\Delta I/I_{\text{background}}$: Used in psychophysics of small points/disks against a background.

$\Delta I/I_{\text{mean}}$: Used in psychophysics for simple stimuli. Gives same number as Michelsen for gratings.

c(x,y) = $(I(x, y) - I_{\text{mean}})/I_{\text{mean}}$: contrast at a point (x,y). Could be as function of time too, c(x,y,t).

$\sum_{x,y} \sqrt{(I(x, y) - I_{\text{mean}})^2} / I_{\text{mean}}$: r.m.s. contrast, a good summary measure for complex image. "contrast power", $\sum_{x,y} c^2$(x,y), is the square of r.m.s. contrast. "Contrast energy" is *power x area x duration*. In psychophysics, area is measured in degrees of visual angle.

■ Contrast manipulations

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

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

■ Psychophysics and contrast

When measuring human sensitivity, it is important to carefully measure and calibrate the image stimuli. Because standard computer displays can at best resolve 256 graylevels, it is 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 n r.m.s. contrast of 1. Use ListPlot and Flatten[] to show a scatter plot of the values before and after the scaling.

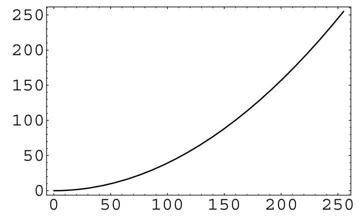
■ Exercise: Given that human contrast sensitivity for a sinewave grating can be as high as 500, could you get a good measure of it using a typical computer graphics screen?

■ 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];
```



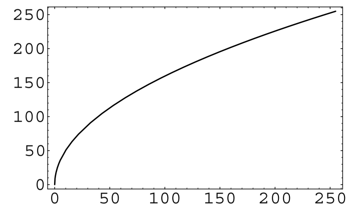
The computer's display card has a look-up-table (LUT) that can be loaded with the inverse gamma function to linearize the display.

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

```
Solve::ifun : Inverse functions are being
   used by Solve, so some solutions may not be found.
```

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

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

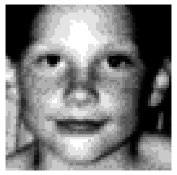


■ Using gamma to do point operations

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

```
ListDensityPlot[face256, Mesh → False, Frame → False, PlotRange → {0, 255}];
```

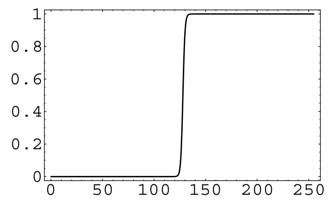
```
ListDensityPlot[output[face256, 1.3],
  Mesh → False, Frame → False, PlotRange → {0, 255}];
```



■ Sigmoidal contrast manipulation

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

```
squash[x_,μ_,γ_] := N[1/(1 + Exp[-γ*(x-μ)])];
Plot[squash[x,128,1],{x,0,255},Frame→True];
```



```
gain = 0.045; μ = Mean[Flatten[face256]];
ListDensityPlot[squash[(face256 - μ) + μ, 128, 1],
  Mesh → False, Frame → False, PlotRange → {Min[face], Max[face]}];
```

■ Hard-thresholds: "Mooney faces"

Here is a function that takes an image and sets pixels bigger than τ to 255, and if less than (or equal to) τ, to 0:

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

A hard-thresold is used to produce "Mooney faces":

```
ListDensityPlot[Mooney[face256, 200], Mesh → False,
  Frame → False, PlotRange → {Min[face], Max[face]}];
```

■ 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

Requires add-in package (above) to have been loaded.

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

■ Mean, variance, r.m.s. contrast

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

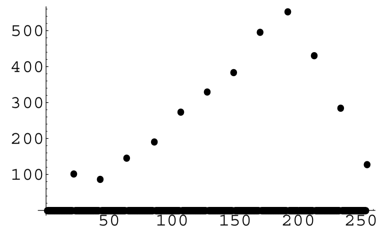
r.m.s. contrast can be calculated as:

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

```
0.600059
```

■ Histograms

```
histoface = BinCounts[Flatten[face256], {0, 255}];
ListPlot[histoface, PlotStyle → PointSize[0.02]];
```



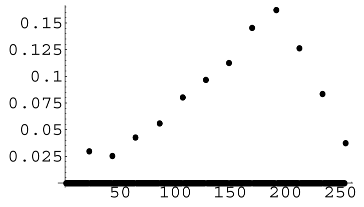
You can tell that the image is quantized at a coarse level (less than 4 bits).

Alternatively, you could calculate the histogram with built-in 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];
```

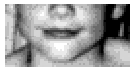
If we normalize the histogram so that the sum is one, then we have a probability:

```
ListPlot[histoface / Apply[Plus, histoface], PlotStyle → PointSize[0.02]];
```

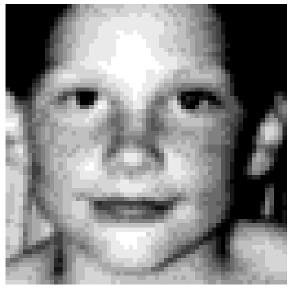


Getting regions of images

```
ListDensityPlot[Take[face256, {1, 32}, {1, 64}], Mesh → False, Frame → False,
  PlotRange → {Min[face256], Max[face256]}, AspectRatio → Automatic];
```



```
ListDensityPlot[Take[face256, {1, 64}, {1, 64}], Mesh → False, Frame → False,
  PlotRange → {Min[face256], Max[face256]}, AspectRatio → Automatic];
```



By selecting the image above, and holding down the option key on the Mac (or control key in Windows), you can use the mouse click to select coordinates. Select the {x0,y0}, and {x1,y1} as the corners of the rectangular patch that you want. Do Save, and then do Paste in a cell below. Here are the coordinates for diagonal points on the left eye:

```
{{11.0143, 39.8432}, {24.1569, 46.2025}}
```

```
{{11.0143, 39.8432}, {24.1569, 46.2025}}
```

```
Round[{{11.0143, 39.8432}, {24.1569, 46.2025}}]
```

```
{{11, 40}, {24, 46}}
```

Origin is in the lower-left corner (the starting point in matrix is the upper right).

■ Exercise: Use ListDensityPlot and Take[] to plot a sub-picture

Image processing using Fourier transforms

Introduction

Spatial filtering by convolution can often be done much more quickly in the spatial frequency domain. Suppose we want to filter an image of a face. Let our image to be filtered be **face**. The idea is to generate a filter list, **filter**, and take its fourier transform, **filterft = Fourier[filter]**. Then we multiply the **filterft** by the fourier transform of face, call it **faceft.** Our filtered output is the inverse transform of the product:

InverseFourier[filterft faceft]

Although *Mathematica* can handle vectors of arbitrary dimension, it will use a special algorithm called the Fast Fourier Transform (FFT) if the image vector has dimensions which are powers of 2 (e.g. 8, 16, 32, 64, etc..).

Complex number notation allows us to handle phase in a very elegant way. So first a review.

Complex numbers

Complex numbers can be written in terms of real and imaginary parts:

```
z = 3 + 4 I;
Re[z]
Im[z]
```

```
3
```

```
4
```

A complex number can be thought of as a 2D vector whose x-value is the real part, and the y-value is the imaginary part.

Alternatively, complex numbers can be written in polar notation where the complex number has a length r = **Abs[z]** and an angle θ = **Arg[z]** that it makes with the real axis.

$z = r (\cos\theta + i \sin\theta)$

```
Abs[z]
Arg[z]
```

```
5
```

$\text{ArcTan}\left[\frac{4}{3}\right]$

Using Euler's formula, $e^{i\theta} = \cos\theta + i \sin\theta$,

 we can put the length and angles back together.

```
z = Abs[z] Exp[I Arg[z]]
```

$5\, e^{i\, \text{ArcTan}\left[\frac{4}{3}\right]}$

```
N[%]
```

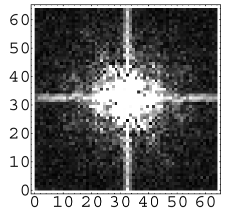
```
3. + 4. i
```

Fourier decomposition of image of a face

■ **Find the amplitude spectrum (spectrum) and the phase spectrum (phase) for the face picture. Chop[] sets small values to zero. Note that most of the energy is near zero.**

```
faceft = Fourier[face];
facespectrum = Chop[Abs[faceft]];
facephase = Chop[Arg[faceft]];
```

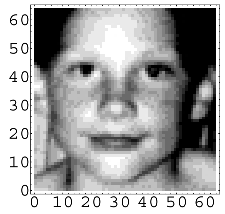
```
ListDensityPlot[RotateLeft[facespectrum,{hsize,hsize}], Mesh->False];
```



■ **Put amplitude spectrum and phase back together**

```
facerestored =
Chop[InverseFourier[facespectrum Exp[I facephase]]];
```

```
ListDensityPlot[facerestored, Mesh->False];
```



You can play with changes to the amplitude or phase spectrum to see what happens. For example, what happens if you multiply the amplitude spectrum by an array whose values are proportional to the negative of the sum of the squares of the horizontal and vertical spatial frequencies? (Answer: this is the spatial frequency equivalent to applying a Laplacian operator in the space domain).

■ **Shift operators, RotateLeft, RotateRight - useful functions for aligning filter representations**

```
shift[mat_,size_] :=
Transpose[RotateRight[Transpose[RotateRight[mat,size]],size]]
```

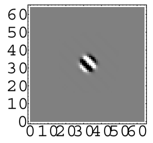
or better yet,

```
shift[mat_, size_] := RotateLeft[mat, {size, size}];
```

■ **Oriented gabor filter.**

```
sgabor[x_,y_, fx_, fy_,sig_] :=
    N[Exp[(-x^2 - y^2)/(2 sig*sig)] Sin[
        2 Pi (fx x + fy y)]];
filter = Table[sgabor[i/32,j/32,4,4,1/16],
    {i,-hsize,hsize-1},{j,-hsize,hsize-1}];
```

```
ListDensityPlot[filter,Mesh->False,
    PlotRange->{-.1,.1}];
```

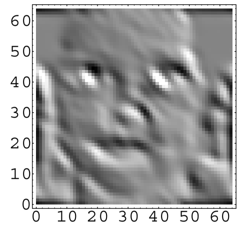


"Neural images": Apply an oriented filter to the face image

Below we show the results of applying a 45 degree oriented simple cell (a Hubel and Wiesel "edge detector") to the face image. This picture can be thought of as a "neural image" corresponding to the outputs of that sub-class of simple cells whose filtering properties can be modeled with this particular spatial frequency and orientation.

filter is generated using the sgabor filter above.

```
filterft = Fourier[shift[filter,hsize]];
ListDensityPlot[fface = Chop[
    t = InverseFourier[filterft faceft]],
    Mesh->False];
```



The neural image represents the output of a channel with the given spatial frequency and orientation.

Importance of the phase spectrum in visual recognition

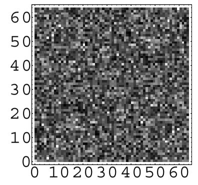
In the early days of pattern vision research, it was not often appreciated that the phase spectrum carries the significant information for recognition. This exercise demonstrates this by making an images with 1) random phase and face amplitude spectrum; 2) a random amplitude spectrum and face's phase spectrum.

■ **Make some uniformly distributed white noise. Then extract the amplitude spectrum and phase**

```
ft = Table[N[Pi (2 Random[Real]-1)],{i,1,size},{j,1,size}];
ft = Fourier[ft];
randomphase = Chop[Arg[ft]];
randomspectrum = Chop[Abs[ft]];
```

The amplitude spectrum is flat on average.

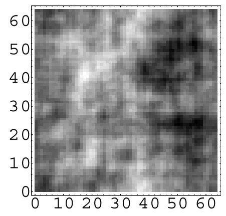
```
ListDensityPlot[
  RotateRight[randomspectrum, {hfwidth, hfwidth}], Mesh → False];
```



■ **What does face "restored" face look like with his original amplitude spectrum but with random phases?**

```
facewithrandomphase =
Chop[InverseFourier[facespectrum Exp[I randomphase]]];
```

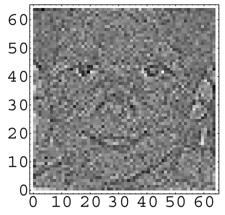
```
ListDensityPlot[facewithrandomphase, Mesh->False];
```



■ **What does face look like if we substitute a noise spectrum (randomspectrum) for his amplitude spectrum?**

```
facewithrandomspectrum =
Chop[InverseFourier[randomspectrum Exp[I facephase]]];
```

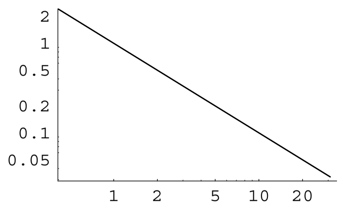
```
ListDensityPlot[facewithrandomspectrum,
    Mesh->False];
```



Random Fractals

Random fractals are a crude but good statistical models for the amplitude spectra certain classes of natural images. Random fractals can be characterized by the fractal dimension D (3<D<4) and amplitude spectrum, 1/(f_x^2 + f_y^2)^(4-D). The amplitude spectrum is a straight line when plotted against frequency in log-log coordinates. The condition If[] is used to include a fudge term (1/(2)^(q)) to prevent blow up near zero in the Block[] routine later.

```
LogLogPlot[
  If[(i ≠ 0 || j ≠ 0), 1 / (i * i + 0 * 0) ^ (q), 1 / (2) ^ (q)], {i, 0, hfwidth - 1}];
```



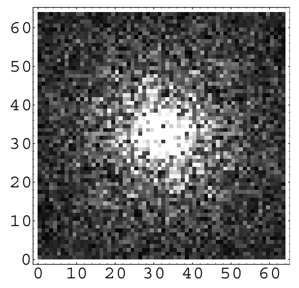
■ **Here is a function to make a low-pass filter with fractal dimension D. (D, here should be between 3 and 4). Note that we first make the filter centered in the middle, and then adjust it so that it is symmetric with respect to the four corners.**

```
fwidth = 2*hsize; hfwidth = fwidth/2;
fractalfilter[D_] :=
Block[ {q,i,j,mat},
    q = 4 - D;
    mat = Table[If[(i != 0 || j!= 0),
        1/(i*i + j*j)^(q), 1/(2)^(q)],
    {i,-hfwidth,hfwidth-1},{j,-hfwidth,hfwidth-1}];
    mat = RotateRight[mat,{hfwidth,hfwidth}];
    Return[mat];
    ];
```

```
ListDensityPlot[
  RotateLeft[fractalfilter[3.5], {hfwidth, hfwidth}], Mesh → False];
```

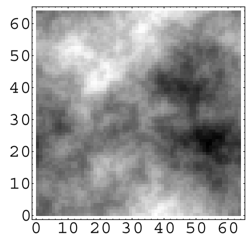
Here is the amplitude spectrum plot for a random fractal image:

```
ffilt = fractalfilter[3.5] randomspectrum;
ListDensityPlot[RotateRight[ffilt,{hfwidth,hfwidth}], Mesh->False];
```



■ **Here is a random fractal image, with D = 3.2**

```
ListDensityPlot[Chop[
InverseFourier[
fractalfilter[3.2] randomspectrum Exp[I randomphase]]],
Mesh->False];
```



Exercises: Some general properties of the spectra of filters and images

Verify the following general properties with filter lists generated from cgabor and sgabor filters.

■ **Exercise 1: What is the imaginary part of the fourier transform of a real even symmetric (e.g. cosine-phase filter) image? What is the real part of an odd-symmetric image (e.g. sine-phase filter)?**

■ **Exercise 2 - Similarity theorem: If the spatial scale of a filter or image is doubled (i.e. 2 x -> x), what happens to the scale of the filter's amplitude spectrum? What happens to the amplitude of the filter's amplitude spectrum?**

Verify this with the filter from cgabor. Below, F[] stands for fourier transform, l for an image, and L for the fourier transform of l.

$$F[l(ax, by)] = \frac{1}{ab} L(\frac{f_x}{a}, \frac{f_y}{b})$$

■ **Exercise 3 - Parseval's theorem: What is the relationship between the length of an image vector, and the length of its transform?**

■ **Exercise 4 - Shift Theorem**

Verify that if you shift cgabor by 90 degrees to get sgabor, you can calculate the fourier transform of sgabor by multiplying the fourier transform of cgabor by an appropriate exponential. L() below is the fourier transform of cgabor.

$$F[l(x-a, y-b)] = L(f_x, f_y)\mathrm{e}^{-2\pi i(f_x a + f_y b)}$$

Next time

Efficient coding

Science writing

Appendices

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.

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

Watson, A. B. (1987). Efficiency of a model human image code. *Journal of the Optical Society of America, A, 4*(12), 2401-2417.

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

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