Initialize

Read in Statistical Add-in packages:

```
Off[General::spell1];
<< MultivariateStatistics`
```

```
SetOptions[ArrayPlot, ColorFunction \rightarrow \text{"GrayTones"}, DataReversed \rightarrow \text{False},
Frame \rightarrow \text{False}, AspectRatio \rightarrow \text{Automatic}, Mesh \rightarrow \text{False},
PixelConstrained \rightarrow \text{True}, ImageSize \rightarrow \text{Small}];
SetOptions[ListPlot, ImageSize \rightarrow \text{Small}];
SetOptions[Plot, ImageSize \rightarrow \text{Small}];
SetOptions[DensityPlot, ImageSize \rightarrow \text{Small}, ColorFunction \rightarrow \text{GrayLevel}];
```

```
nbinfo = NotebookInformation[ EvaluationNotebook[] ];
dir =
("FileName" /. nbinfo /. FrontEnd\File Name[d_List, nam_, ___] \rightarrow
   ToFileName[d]);
```

```
downsample[imaged_, f_] := Take[imaged, {1, -1, f}, {1, -1, f}]
```

```
cup = ImageData[
   ];
```

```
width = Dimensions[ cup ][[1]];`
Outline

Last time: Functional explanations of neuron coding properties

Efficient coding is a task general explanation

-- 1st order statistics showing a non-uniform distribution of input intensities could be more efficiently transmitted if recoded using the density-mapping theorem. This resulted in an even distribution of the frequency of responses.

-- 2nd order spatial statistics suggested that efficiency could be further improved by taking advantage of nearby correlations in intensity to encode differences, thereby reducing the average range of variation of the responses.

-- We introduced a standard statistical technique, PCA, for efficient coding that seeks out subspaces that capture where most of the data “live”. More specifically, subspaces that can account for most of the variance.

PCA rotates the coordinates of the data space (e.g. space of a collection of images) to find where projections are decorrelated.

We looked at how to do PCA with neural networks

Today

- V1 and efficient coding
- V1 neuron receptive fields as edge detectors, computing spatial derivatives

Local image measurements that correlate well with useful surface properties

  task specific explanation-find "significant" intensity changes

  edge detection

  1st and 2nd spatial derivatives (i.e. the edge and bar detectors)

  relation to center-surround and oriented receptive fields

- Problems with edge detection

More efficient coding

Efficient coding
We’ve seen that neurons in V1 are tuned to spatial orientation and spatial frequency. One common descriptive model is to represent receptive fields by filters that are either a rotated, scaled, or phase-shifted version of a standard oriented center-surround template. This kind of representation is closely related to wavelet theory. The recoding can be thought of as a transformation of incoming image information into neural activity that represents how much contrast is contributed by each of a set of basis functions. A representative set for one spatial location is shown below. This set gets repeated across space, with the assumption that the larger, low-spatial frequency filters don’t need to get repeated as frequently as the small, high-spatial frequency filters. If the template filter is centered symmetrically about the excitatory part, it is called a cosine-phase filter. If it is centered anti-symmetrically, it is a sine-phase filter. Examples of sine-phase filters are below:

![Cosine-Phase Filters](image1.png)

**Efficient, sparse coding in V1**

Kurtosis.

Recall 1rst order statistics as shown in the histogram of intensities:
If we convolve the above image with a gabor filter:

and plot the histogram of the response, we get this:
This histogram has high “kurtosis”. Kurtosis compares the concentration of data around the peak to the tails versus the concentration in the flanks. A high kurtosis reflects the fact that this image (and almost all natural images) tend to have a large proportion of responses near the mean, together with “heavy tails”, meaning that there are more response contributions from edge-like features. This simple measure applied to natural images shows that they are not gaussian. The reason is that linear combinations of gaussian random numbers are gaussian. The filtering operation is just a linear combination, so would be gaussian, but it is not.

### Demonstration

```plaintext
In[13]:= 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)]*Grating[x, y, fx, fy, phase]
Gratin[kern[fx_, fy_, sig_, phase_] :=
    Table[GratingPatch[x, y, fx, fy, sig, phase], {x, -1, 1, .05}, {y, -1, 1, .05}];

In[16]:= fr = 1.3; theta = 2.3; sig = 0.5; phase = 0.0;
g11 = Image[kern[fr * Cos[theta], fr * Sin[theta], sig, phase]] // ImageAdjust;

In[18]:= g12 =
    ImageConvolve[kern[fr * Cos[theta], fr * Sin[theta], sig, phase]] // ImageAdjust;

GraphicsRow[{g11, g12, ImageHistogram[g11], ImageHistogram[g12]}]
```

Out[19]=

![Images showing the demonstration of edge detection](image-url)
**Olshausen & Field: Primary cortex**

The range of orientation and spatial frequency selectivity found in V1 cells is consistent with a local Fourier, or "gabor function", or wavelet decomposition of image patches, in showing both spatial frequency and orientation selectivity. We might expect something like Fourier analysis of the image to result in efficient coding because of the close relationship between Fourier rotations and Principal Components Analysis (e.g. Appendix A, Andrews, 1983). Fourier coefficients for natural images tend to be uncorrelated. However, PCA doesn’t provide an explanation for the local property of receptive fields--i.e. that V1 responses are primarily driven by contrast within a small region of visual space.

In 1996, Olshausen and Field showed that one could derive a set of basis functions that have the same characteristics as the ensemble of visual simple cells, including local in primary visual cortex by requiring two simple constraints:

1) One should be able to express the image \( I(x,y) \) as a weighted sum of the basis functions, \( \{ \phi_i(x,y) \} \), (functions of the cumulative effect of synaptic weights from retina to V1).

\[
I(x, y) = \sum_i a_i \phi_i(x, y)
\]

2) The total activity across the ensemble should, on average, be small. This latter constraint is called "sparse coding". That is, a typical input image should activate a relatively small fraction of neurons in the ensemble. \( S() \) for example could be the absolute value of the activity \( a_i \).

Constraints 1) and 2) could be encouraged by finding the set \( \{ \phi_i(x,y) \} \), that minimizes:

\[
\sum_{x,y} \left[ I(x, y) - \sum_i a_i \phi_i(x, y) \right]^2 + \sum_i S(a_i)
\]  

These two constraints led to an explanation of V1 receptive fields:


**Adaptation**

Human orientation and spatial frequency selectivity changes with adaptation (Recall the orientation and spatial frequency adaption demonstration in an earlier lecture.). Adaptation has been interpreted as an optimal change to new conditions.
Higher order redundancies & contrast normalization

- Contrast normalization

There are many higher order redundancies in natural images, and a major challenge is to characterize them, and understand how the visual system exploits these redundancies. For example, the figure below shows that the output of one spatial filter (receptive field (RF) responses) can predict the variability in a second spatial filter.

Simoncelli has shown how a non-linearity called “contrast normalization” removes this redundancy. Recall that simple and complex cells show contrast normalization—a feature not included in the above simple model. For a discussion of steady-state models of simple and complex cells see Heeger (1991) and Carandini et al. (1997).

(e.g. see, Wainwright, M. J. (1999). Visual adaptation as optimal information transmission. Vision Research, 39, 3960–3974.)
\[ R_i = \sum_{k=1}^{N_i} R_k^2 \] where \( N_i \) is a neighborhood of neuron \( i \).


**Edge Detection**

**Introduction**

The above explanations for receptive fields can be considered “task general”. But the original explanation for V1 orientation responses was that they represented edges in the image. This can be thought of as “task specific”. We now take this perceptive.

Let’s plot the intensities across and object, and see what it might mean to detect an edge.
Edge detection as differentiation

The Noise/Scale trade-off

The definition of edge detection is very tricky—exactly what do we want to detect? We would like to label "significant" intensity changes in the image. One definition of significant edges is that they are the ones with the biggest change in intensity. The biggest would correspond to step changes. In Mathematica, these can be modeled as $g(x) = \text{UnitStep}[x]$. One of the first problems we encounter is that edges in an image are typically fuzzy, either due to optical blur in the imaging device, or because the scene causes of edges are not changing abruptly. A second problem is noise, which we get to later.

Our first generative model for image data $f$ is to convolve the step with a blur function:

$$f(x) = \int g(x - x') \text{blur}(x') \, dx' = g*\text{blur}$$

where $g()$ is the signal to be detected or estimated. $g(x)$ is a step function:
Depending on the type of blur, the image intensity profile f(x) will look more or less like:

One way of locating the position of the edge in this image would be to take the first derivative of the intensity function, and then mark the edge location at the peak of the first derivative:

Alternatively, we could take the second derivative, and look for zero-crossings to mark the edge location.
So far so good. But real images rarely have a nice smooth intensity gradation at points that we subjectively would identify as a clean edge. A second, more realistic generative model for intensity data would be:

\[ f(x) = \int g(x - x') \, \text{blur}(x') \, dx' + \text{noise} \]

We'll add a fixed sample of high-frequency "noise":

\[ \text{noisyedge}[x, s_] := \text{edge}[x, s] + \]
\[ 0.01 \cos(10 \, x) + -0.02 \sin(10 \, x) + 0.03 \cos(12 \, x) + 0.04 \sin(12 \, x) + \]
\[ -0.01 \cos(13 \, x) + -0.03 \sin(13 \, x) + 0.01 \cos(14 \, x) + 0.01 \sin(14 \, x) + \]
\[ -0.04 \cos(25 \, x) + -0.02 \sin(25 \, x) + 0.02 \cos(26 \, x) + 0.03 \sin(26 \, x); \]

Now, if we take the first derivative, there are all sorts of peaks, and the biggest isn't even where the edge is:
Looking for zero-crossings looks even worse:

There are many spurious zero-crossings.

In general, the higher the frequency of the noise, the bigger the problem gets. We can see what is going on by taking the nth derivative of the sinusoidal component with frequency parameter f. Here is the 3rd derivative of a component with frequency f:

The magnitude of the output is proportional to the frequency raised to the power of the derivative. Not good.

- **A solution: pre-blur using convolution**

  A possible solution to the noise problem is to pre-filter the image with a convolution operation that blurs out the fine detail which is presumably due to the noise. And then proceed with differentiation. The problem is how to choose the degree of blur. Blur the image too much, and one can miss edges; don't blur it enough, and one gets false edges.

  This is one edge detection dilemma: *Too much blur and we miss edges, too little and we have false alarms.*
Some biologically motivated edge detection schemes

Edge detection using 2nd derivatives: Marr-Hildreth

The Marr-Hildreth edge detector. The idea is to: 1) pre-blur with a Gaussian; 2) take second derivatives of the image intensity using the Laplacian; 3) locate zero-crossings. In short,

\[
\text{Find zero-crossings of: } \nabla^2 r(x,y) = \int \nabla^2 G_\sigma(x',y') g(x-x', y-y') \, dx' \, dy' = (\nabla^2 G_\sigma)^* g
\]

\[\nabla^2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\]

is the Laplacian operator, which takes the second derivatives in x and y directions, and sums up the result.

As you saw in one of the problem sets, the Laplacian and convolution operators can be combined into the "del-squared G" operator, \(\nabla^2 G_\sigma\), where

\[
G_\sigma[x,y] = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2+y^2}{2\sigma^2}}
\]

The order of the operators doesn't matter, so one can take the Laplacian of the Gaussian first, and then convolve this del-squared G kernel with the image, or one can blur the image first, and then take the second derivatives:

\[
r(x,y) = \nabla^2 (G_\sigma * g)
\]

The operators are said to "commute".

As \(\sigma\) approaches zero, \(G_\sigma\) becomes a delta function, and the \(\nabla^2 G_\sigma\) becomes a Laplacian \(\nabla^2\), i.e. a second derivative operator. For small \(\sigma\), the detector is sensitive to noise. For large \(\sigma\), it is less sensitive to noise, but misses edges. The biological appeal of the Marr-Hildreth detector is that lateral inhibitory filters provide the \((\nabla^2 G_\sigma)\) kernel.

One could build zero-crossing detectors by ANDing the outputs of appropriately aligned center-surround filters effectively building oriented filters out of symmetric ganglion-cell (or LGN) like spatial filters (Marr and Hildreth).

*Mathematica* provides built-in functions to convolve an image with the Laplacian of a Gaussian, followed by a zero-crossing detector:
Edge detection using 1rst derivatives

But what about the oriented filters in the cortex? One interpretation consistent with the Hubel-Wiesel edge detector interpretation of sine-phase receptive fields in V1, is in terms of 1rst derivatives.

Because of the orientation selectivity of cortical cells, they have sometimes been interpreted as edge detectors. We noted earlier how a sine-phase Gabor function filter (1 cycle wide) would respond well to an edge oriented with its receptive field.
As the width of the gaussian envelope decreases, these sine-phase or odd-symmetric filters can also be viewed as 1rst order spatial derivatives. The image below represents weights (1, -1):

In a 1D horizontal operation, the filtered image response is approximated by the derivative of the image intensity, \( l \), with respect to space, \( \frac{dl(i)}{di} = l_i - l_{i-1} \), where \( l_i \) is the intensity of the \( i \)th pixel.

How can we combine oriented filters to signal an edge in 2D? The first-derivative operation takes the gradient of the image. From calculus, you learned that the gradient of a 2D function evaluated at \( (x,y) \) is a vector that points in the direction of maximum change. So taking the gradient of an image should produce a vector field where the vectors are perpendicular to the edges. The length of the gradient is a measure of the steepness of the intensity gradient.

- **The gradient of a function**

\[
\nabla g = \left( \frac{\partial g(x, y)}{\partial x}, \frac{\partial g(x, y)}{\partial y} \right)
\]

In[48]:=    
contcup = ListInterpolation[Transpose[Reverse[cup]],
{1, width}, {1, width}];
Plot the derivative in the x-direction

\[
\text{In[49]} := \text{gd1 = DensityPlot[contcup[x, y], \{x, 1, \text{width}\}, \{y, 1, \text{width}\},}
\text{Mesh \to \text{False}, \text{PlotPoints} \to 128];}
\]
\[
\text{gd2 = DensityPlot[Evaluate[D[contcup[x, y], x]], \{x, 1, \text{width}\},}
\text{\{y, 1, \text{width}\}, \text{PlotPoints} \to \text{width} / 2, \text{Mesh} \to \text{False}, \text{Frame} \to \text{False},}
\text{\text{PlotRange} \to \{-100, 100\}];}
\]

\[
\text{In[51]} := \text{GraphicsRow[\{gd1, gd2\]}
\]

\[
\text{Out[51]} =
\]

Let's take the derivatives in both the x and y directions:

\[
\text{In[52]} := \text{fxcontcup[x\_, y\_] := D[contcup[x1, y1], x1] /.(x1 \to x, y1 \to y);}\\
\text{fycontcup[x\_, y\_] := D[contcup[x1, y1], y1] /.(x1 \to x, y1 \to y);}\\
\]

Now let's put the x and y directions together and compute the squared gradient magnitude:

\[
\text{In[54]} := \text{fcontcup[x\_, y\_] := D[contcup[x1, y1], x1]^2 + D[contcup[x1, y1], y1]^2 /.(x1 \to x, y1 \to y);}\\
\]

\[
\text{In[55]} := \text{gradientedge = Table[N[fcontcup[x, y]], \{x, 1, \text{width}\}, \{y, 1, \text{width}\}];}
\]

The range of gradientedge is large, so we'll plot the Log to squeeze it down:
In[56]:= `ArrayPlot[Log[Transpose[gradientedge] + .0001], DataReversed -> True]

Out[56]=


Doesn't look too bad, but it isn't clean and some of our satisfaction is premature and the result of our visual system effectively fitting the edge representation above into the interpretation of a cup. Further, we haven't specified a blur level, or a criterion for the threshold. We haven't put a measure of confidence on the edges. We can manually adjust for threshold:

In[57]:= `Manipulate[ArrayPlot[Sign[t - Transpose[gradientedge]]], DataReversed -> True], {{t, .01}, 0, Max[gradientedge]]

Out[57]=


but the selection is subjective.

There is also useful information in the direction of the gradient vectors:
Imagine trying to link up points along an edge with the information in the left panel---You get a better idea of how much variability remains in terms of both direction and magnitude.

If we took many pictures of the same cup under different illumination conditions, one could measure how much variability (at a point) is in the magnitude vs. direction of the gradient. Chen et al. (2000) did this and showed that there is much more variability in the magnitude than the direction of the gradient. This suggests that for illumination-invariant recognition, one should rely more on orientation than contrast magnitude.

**Summing up: Combining a smoothing pre-blur with 1st derivatives**

As with the 2nd derivative zero-crossing detector, the idea is to blur the image, but instead then take the first derivates in the x and y directions, square each and add them up.

The x and y components of the gradient of the blur kernel :
\[ G[x_, y_, \sigma_x, \sigma_y_] := \frac{1}{\sqrt{2\pi}} \left( \frac{x^2}{\sigma_x^2} + \frac{y^2}{\sigma_y^2} \right) \ln \left( \sqrt{\sigma_x^2 + \sigma_y^2} \right) \]

\[ dGx[x_, y_] := D[G[x1, y1, 1, 2], x1] /. \{x1 -> x, y1 -> y\}; \]

\[ xg = DensityPlot[-dGx[x, y], \{x, -2, 2\}, \{y, -2, 2\}, Mesh -> False, Frame -> False, PlotPoints -> 64]; \]

\[ dGx[x_, y_] := D[G[x1, y1, 2, 1], y1] /. \{x1 -> x, y1 -> y\}; \]

\[ yg = DensityPlot[-dGx[x, y], \{x, -2, 2\}, \{y, -2, 2\}, Mesh -> False, Frame -> False, PlotPoints -> 64]; \]

\[ GraphicsRow[{xg, yg}] \]

--2D smoothing operator followed by a first order directional derivatives in the x and y directions.

If one takes the outputs of two such cells, one vertical and one horizontal, the sum of the squares of their outputs correspond to the squared magnitude of the gradient of the smoothed image:

\[ (r_x(x,y), r_y(x,y)) = \left( \frac{\partial G_{\sigma}(x,y)}{\partial x}, \frac{\partial G_{\sigma}(x,y)}{\partial y} \right) g(x,y) = \nabla G_{\sigma} \ast g(x,y) = \left( \frac{\partial G_{\sigma}(x,y)}{\partial x} \ast g(x,y), \frac{\partial G_{\sigma}(x,y)}{\partial y} \ast g(x,y) \right) \]

Then to get a measure of strength, compute the squared length:

\[ |\nabla G_{\sigma} \ast g(x,y)|^2 = \left| \nabla (G_{\sigma} \ast g (x,y)) \right|^2 = r_x(x,y)^2 + r_y(x,y)^2 \]

We'll encounter this idea later when we extend detecting edges in space to detecting edges in space-time in order to make motion measurements.

**Morrone & Burr edge detector--combining even and odd filters**

The Marr-Hildreth 2nd derivative operation is similar to the even-symmetric cosine-phase gabor or "bar detector". The
1st derivative gradient operator is similar to the odd-symmetric sine-phase gabor. Any reason to combine them?

![Image of gradient operators]

Sometimes the important "edge" is actually a line--i.e. a pair of edges close together. A line-drawing is an example.

The Appendix shows how one can combine both sine and cosine phase filters to detect both edges and lines. A sine and cosine phase pair are sometimes called "quadrature (phase) pairs". The summed squared outputs can be interpreted as "local contrast energy".

Quadrature pairs will also crop up later in the context of motion detection.

**Problems with interpreting V1 simple/complex cells as edge detectors**

Although one can build edge detectors from oriented filters, a simple cell cannot uniquely signal the presence of an edge for several reasons. One is that a cell’s response is a function of many different parameters. A low contrast bar at an optimal orientation will produce the same response as a bar of higher contrast at a non-optimal orientation. There is a similar trade-off with other parameters such as spatial frequency and temporal frequency. In order to make explicit the location of an edge from the responses of a population of cells, one needs to compute something like the "center-of-mass" over the population, where response rate takes the place of mass. Another problem is that edge detection has to take into account a range of spatial scales. We discussed evidence earlier that the cortical basis set does encompass a range of spatial scales, and in fact may be "self-similar" across these scales. See Koenderink (1990) for a theoretical discussion of "ideal" receptive field properties from the point of view of basis elements. One way of combining information efficiently across scales is to use a Laplacian image pyramid. Oriented information can be computed across scales using a steerable pyramid.

Konishi et al. (2003) used signal detection theory and real images to show that there is an advantage in combining information across scales when doing edge detection.

**Segmentation & Why using edge detectors to find object boundaries is hard**

The problem is to get from intensity edges to object boundaries.

**The problem of texture and background**

The above analysis assumed that an edge detection should be the solution to an image-based generative problem: Given
The above analysis assumed that an edge detection should be the solution to an image-based generative problem: 

\[ f(x) = \int g(x - x') \, \text{blur}(x') \, dx' + \text{noise} \]

We used the cup image to illustrate how scale and noise (represented by the blur and noise processes) confound estimates of \( g() \). But the cup image had a fairly uniform figure and background. Consider the more typical case, such as a patterned cup against a non-uniform background:

```plaintext
In[66]:= camo = ImageData[ColorConvert[camo, "Grayscale"]];

In[67]:= camod = downsample[camo, 2];
cupd = downsample[cup, 2];
images = {Image[cupd], Image[camod]};
Manipulate[
  Pane[
    Map[Binarize[ImageAdjust[GradientFilter[#, radius], {.7, .6}], t] &,
        images, {300, 150}], {radius, 1, 10, 1}, {t, 0, 1, .1},
    SaveDefinitions -> True]
```

Out[70]=

```plaintext
{ }```

From: http://www.flickr.com/photos/96221617@N00/280637989/

The above example illustrates the problem of misleading edges both at the boundary between the object and background, but also between texture elements in the object and its boundary. One needs to take into account texture as well as intensity in determining object boundaries (see Malik et al, 2001).

**Edge classification: Some causes of edges are more important than others: task-dependence**

We’ve seen that uncertainty due to noise and spatial scale confound reliable edge detection. But the above demonstrates another reason why edge detection is hard--local intensity gradients can have several possible meanings. Even when there is little or no noise, local measurements of contrast change say very little about the physical cause of the gradient. And a "smart" visual system takes the causes into account.

![Image of edge classification](https://example.com/image.png)

So on the one hand, it still makes sense to interpret lateral inhibitory filters and oriented cortical filters as possible components of an edge detection system, but we have to allow for considerable uncertainty in the meaning of their outputs--i.e. a local edge detector typically has a low signal-to-noise ratio for a variety of ways of defining signals, e.g., whether the causes are geometric or photometric.

For tasks such as object recognition, vision places a higher utility on surface and material edges than on other types. Surface edges are used differently from material edges. Shadow edges are variable yet potentially important for stereo. Specular edges are variable, but problematic for stereo because they are at different surface positions for the two eyes.

**Natural images & segmentation**

Where to draw the important contours?
How can one go from the imperfect output of a low-level edge detector to a clean "line-drawing" representing the true boundaries of an object? Grouping local measurements that are similar is one step. This can be at the edge or region level, i.e. grouping local edge measurements into longer lines and grouping features within a region. Grouping processes are sometimes called "intermediate-level" because they don't rely on specific knowledge about particular object classes, but just on how contours typically go, or how features are typically related (i.e. similar orientations, colors, motion direction,...). Perceptual grouping or similarity principles were studied by Gestalt psychologists in the early 1900s.

In addition, the visual system seems to be solving a generative model that is more scene-based than image-based--it cares about the type of edge, and the types of objects and arrangements likely to be encountered. This will be the focus of the next few lectures.

Even with intermediate-level grouping, and edge selection based on scene-based filtering, finding the boundaries of objects requires more specific knowledge or memory about the possible shapes of previously seen objects. The well-known dalmation dog illusion illustrates the high-level knowledge the human visual system brings to bear on the problem of segmenting an object.

To translate edge detection into useful segmentations is a non-trivial computer vision problem (see Malik et al. 2001, and Tu and Zhu (2002)).

Given the problems of edge detection in the absence of context, it seems more appealing to interpret the spatial filtering properties of V1 as efficient encoding. However, if one thinks of V1 oriented cell responses as representing tentative "edges", with a representation of uncertainty, then one can begin to understand how high-level "models" may be used to select the edgest that belong, and reject those that don't (Yuille and Kersten, 2006). How these tentative edges might be extracted from both intensity and textural transitions, and how high-level information might constrain these remains a challenging area of research.
**Interest points & saliency**

Human vision does more than seek out edges as a basis for object detection and recognition. One basic function is to direct the eyes to so-called salient points in an image. In 1954, Fred Attneave pointed out that people's eye fixations were attracted to some features of an image more than others. In "Attneave's cat", shown below, eye movements tend to go to points of high curvature. The Appendix shows one way to extend derivative operators to amplify corners.

![Attneave's cat](image)

There has been a focus in recent years to model first fixations in natural images (Itti & Koch, 2001; Torralba et al., 2006, Zhang et al., 2008). The idea is that first-fixations may be driven by fairly low-level image properties. The key idea is that eye movements go to regions that have low probability given either the current image context (based on statistics from the current image of interest), or a generic natural image context (based on statistics gathered over a large number of images seen in the past).

Subsequent fixations are more difficult to model because of they can be largely determined by the task the person is trying to accomplish.

**Appendices**

**Combining signal detection theory with edge detection**

Canny (1986).

Possible project idea: Build a Bayesian edge detector using gradient statistics measured on and off of real edges, and using methods from signal detection theory to decide whether a given...
measurement is on or off of an edge.

The left panel of the figure below shows "range data", where geometric depth from the camera is represented by graylevel, with dark meaning close, and light meaning far. The right panel shows intensity data. The data in the left can be used to define measures of "geometric ground truth", and one can devise edge detectors based on a signal detection theory analysis of how well the intensity changes on the right predicts geometrical changes on the left. In other words, what edge detector provides the best ROC performance? (See Konishi et al., 2003).

The Hessian, "Interest operators", and saliency.

The input 64x64 image: face

Computing both the first and second derivatives of image intensity can be thought of as filters to pick out regions of an
Computing both the first and second derivatives of image intensity can be thought of as filters to pick out regions of an image that have "salient", i.e. rapid, intensity changes. A natural extension is to look at all four combinations of second derivatives.

Calculating the Hessian of an image using function interpolation.

The Hessian of a function \( f(x_1, x_2, \ldots, x_n) \) is given by:

\[
H(f) = \begin{bmatrix}
\frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\
\frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2}
\end{bmatrix}
\]

For our purposes, \( \{x_1, x_2\} = \{x, y\} \), so the Hessian of an image returns a 2x2 matrix at each point \( (x, y) \) that represents the four combinations of second derivatives in the \( x \) and \( y \) directions. The determinant of each of the 2x2 matrices provides a scalar which is a measure of the "area" of each 2x2 matrix. The area can be used as a rough measure of saliency or local "interest", which takes into account rates of change in \( x \) and \( y \), for example at "corners".

For more information about interest operators in computer vision, read about the SIFT filter (Lowe, 1999).

Let \( \text{filterface} = f \), where we've blurred out face a little to reduce quantization artifacts:

```plaintext
In[77]:= kernel = N[{{1, 1, 1}, {1, 1, 1}, {1, 1, 1}}];
filterface = ListConvolve[kernel, face];

In[79]:= faceFunction = ListInterpolation[Transpose[filterface],
{(-1, 1), (-1, 1)}];

In[80]:= hessian[x_, y_] := Evaluate[D[faceFunction[x, y], {{x, y}, 2}]];
```

- Calculate and plot each of the components of the Hessian at each image point

```plaintext
In[81]:= dxxtemp = Table[hessian[x, y][[1, 2]], {x, -1, 1, 0.005}, {y, -1, 1, 0.005}];
```
The determinant of the Hessian provides a simple measure of "salience". Better models take into account how unexpected local features are relative to the background or likely backgrounds (See Torralba et al., 2006, Itti & Koch, 2001, and Zhang et al., 2008.) These models have been applied to predicting human visual eye movements.

For current computer vision work on local feature detection, see papers by Lowe in the references, and http://en.wikipedia.org/wiki/Scale-invariant_feature_transform

Also see: http://en.wikipedia.org/wiki/Interest_point_detection
Morrone & Burr: polarity sensitive & polarity insensitive

Morrone and Burr edge/bar detectors

Suppose we convolve an input signal with an even filter (e.g. Gaussian enveloped cosine-wave) to produce response Re, and then convolve the same input with an odd filter (say, a Gaussian enveloped sine-wave) to produce response Ro. The filters are orthogonal to each other, and so are the responses. Re will tend to peak at "bars" in the image whose size is near half the period of the cosine-wave. Ro will tend to peak near edges.

The local contrast "energy" is defined to be: Sqrt[Re^2 + Ro^2]. Morrone and Burr showed that the local energy peaks where the Fourier components of an image line up with zero-phase--i.e. at points where the various Fourier components are all in sine-phase. These points are edges. But it also peaks near bar features, arguably also interesting image features where the phase coherence is at 90 degrees. In addition to its neurophysiological appeal, a particularly attractive feature of this model is that if one adds up responses over multiple spatial scales, evidence accumulates for edges because the local energy peaks coincide there. They also showed how their model could be used to explain Mach bands.

Mach bands & the Morrone & Burr edge detector

In[85]:=
size = 256; Clear[y];
low = 0.2; hi = 0.8;
y[x_] := low /; x < size/3
y[x_] := (((hi-low)/(size/3)) x + (low-(hi-low)) /; x >= size/3 && x < 2*size/3
y[x_] := hi /; x > 2*size/3
Plot[y[x],{x,0,256},PlotRange->{0,1}];

In[91]:=
picture = Table[Table[y[i],{i,1,size}],{i,1,size}];
ArrayPlot[picture,Frame->False,Mesh->False,
PlotRange->{0,1},AspectRatio->Automatic];

Gabor filters

In[93]:=
sgabor[x_, y_, fx_, fy_, sig_] :=
N[Exp[(-x^2 - y^2) / (2 sig * sig)] Sin[2 Pi (fx x + fy y)]];
cgabor[x_, y_, fx_, fy_, sig_] :=
N[Exp[(-x^2 - y^2) / (2 sig * sig)] Cos[2 Pi (fx x + fy y)]];

In[95]:=
fs = 32;
sfilter = Table[sgabor[(i - fs / 2), (j - fs / 2)], 0, 1 / 8, 4],
{i, 0, fs}, {j, 0, fs}];
sfilter = Chop[sfilter];
g10 = ArrayPlot[sfilter, Mesh -> False, PlotRange -> {-1, 1}, Frame -> False];
Apply odd (sine) filter

\[
\text{fspicture} = \text{ListConvolve}[\text{sfilter}, \text{picture}];
\]

\[
\text{ArrayPlot}[\text{fspicture}, \text{Mesh} ightarrow \text{False}];
\]

Apply even (cosine) filter

\[
\text{fcpicture} = \text{ListConvolve}[\text{cfilter}, \text{picture}];
\]

\[
\text{ArrayPlot}[\text{fcpicture}, \text{Mesh} ightarrow \text{False}];
\]

Look for peaks in local contrast energy

\[
\text{ss} = \text{Sqrt}[\text{fspicture}^2 + \text{fcpicture}^2];
\]

\[
\text{ArrayPlot}[\text{ss}, \text{Mesh} ightarrow \text{False}]
\]

References


