Applications: Compression and Edge Detection

Patrick J. Van Fleet

Center for Applied Mathematics
University of St. Thomas
St. Paul, MN USA

PREP - Wavelet Workshop, 2006
Outline

Today’s Schedule

Image Compression
- The Compression Algorithm
- Modifying the Transform
- Quantizing the Transform and Encoding

Edge Detection
- The Basic Edge Detection Algorithm
- An Example of Edge Detection
- Refining the Quantization Process

In the Classroom
- Teaching Ideas
- Computer Usage

Thursday, 7 June, 2006
Lecture 5
Applications: Compression and Edge Detection
Today’s Schedule

9:00-10:15 ⇒ Lecture Five: Applications: Image Compression and Edge Detection

10:15-10:30 Coffee Break (OSS 235)

10:30-11:45 Computer Session One: Applications: Image Compression and Edge Detection

12:00-1:00 Lunch (Cafeteria)

1:15-2:30 Lecture Six: Daubechies Filters

2:30-2:45 Coffee Break (OSS 235)

2:45-4:00 Computer Session Two: Daubechies Filters and Transform Modules

4:30-9:00 Excursion to the Science Museum

5:30-6:30 Dinner (Cafeteria)
Once the students get the Haar 2D modules running, we move straight to applications.

Two applications that lend themselves well to this transform are naive image compression and edge detection in images.

The Haar transform is not the best transform for image compression – we’ll see that later. The reason it is good now is that it is easily adaptable to map integers to integers or at least $\frac{1}{2}\mathbb{Z}$.
Once the students get the Haar 2D modules running, we move straight to applications.

Two applications that lend themselves well to this transform are **naive image compression** and **edge detection** in images.

The Haar transform is not the best transform for image compression - we’ll see that later. The reason it is good now is that it is easily adaptable to map integers to integers or at least $\frac{1}{2}\mathbb{Z}$.
Once the students get the Haar 2D modules running, we move straight to applications.

Two applications that lend themselves well to this transform are **naive image compression** and **edge detection** in images.

The Haar transform is not the best transform for image compression - we’ll see that later. The reason it is good now is that it is easily adaptable to map integers to integers or at least $\frac{1}{2}\mathbb{Z}$. 
Once the students get the Haar 2D modules running, we move straight to applications.

Two applications that lend themselves well to this transform are naive image compression and edge detection in images.

The Haar transform is not the best transform for image compression - we’ll see that later. The reason it is good now is that it is easily adaptable to map integers to integers or at least $\frac{1}{2}\mathbb{Z}$. 

Thursday, 7 June, 2006

Applications: Compression and Edge Detection
The Compression Algorithm

Image Compression

The Compression Algorithm

- The image compression algorithm is very straightforward.
- For image $A$,
  1. Haar transform $A$ to obtain $B$.
  2. Quantize $B$ to obtain $\tilde{B}$
  3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
  4. Transmit $C$.
- For lossless compression, omit the second step.
The image compression algorithm is very straightforward.

For image $A$,

1. Haar transform $A$ to obtain $B$.
2. Quantize $B$ to obtain $\tilde{B}$
3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
4. Transmit $C$.

For lossless compression, omit the second step.
The image compression algorithm is very straightforward.

For image $A$,

1. Haar transform $A$ to obtain $B$.
2. Quantize $B$ to obtain $\tilde{B}$
3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
4. Transmit $C$.

For lossless compression, omit the second step.
The Compression Algorithm

Image Compression

The Compression Algorithm

- The image compression algorithm is very straightforward.
- For image $A$,
  1. Haar transform $A$ to obtain $B$.
  2. Quantize $B$ to obtain $\tilde{B}$
  3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
  4. Transmit $C$.
- For lossless compression, omit the second step.
Image Compression

The Compression Algorithm

- The image compression algorithm is very straightforward.
- For image $A$,
  1. Haar transform $A$ to obtain $B$.
  2. Quantize $B$ to obtain $\tilde{B}$
  3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
  4. Transmit $C$.
- For lossless compression, omit the second step.
The image compression algorithm is very straightforward.

For image $A$,

1. Haar transform $A$ to obtain $B$.
2. Quantize $B$ to obtain $\tilde{B}$
3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
4. Transmit $C$.

For lossless compression, omit the second step.
The image compression algorithm is very straightforward.

For image $A$,

1. Haar transform $A$ to obtain $B$.
2. Quantize $B$ to obtain $\tilde{B}$
3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
4. Transmit $C$.

For lossless compression, omit the second step.
The Compression Algorithm

Image Compression

The Compression Algorithm

- The image compression algorithm is very straightforward.
- For image $A$,
  1. Haar transform $A$ to obtain $B$.
  2. Quantize $B$ to obtain $\tilde{B}$
  3. Huffman encode $\tilde{B}$ to get encoded bitstream $C$
  4. Transmit $C$.
- For lossless compression, omit the second step.
Image Compression

The Compression Algorithm

▶ To uncompress, we simply undo the steps of the previous algorithm.
▶ For encoded bitstream $C$,
   1. Receive $C$.
   2. Unencode to get $\tilde{B}$
   3. Take the inverse Haar wavelet to get $\tilde{A}$

▶ Note that if we have performed lossy compression, then we will never recover $A$. 
The Compression Algorithm

Image Compression

The Compression Algorithm

- To uncompress, we simply undo the steps of the previous algorithm.
- For encoded bitstream $C$,
  1. Receive $C$.
  2. Unencode to get $\tilde{B}$.
  3. Take the inverse Haar wavelet to get $\tilde{A}$.
- Note that if we have performed lossy compression, then we will never recover $A$.
The Compression Algorithm

Image Compression

The Compression Algorithm

To uncompress, we simply undo the steps of the previous algorithm.

For encoded bitstream $C$,

1. Receive $C$.
2. Unencode to get $\tilde{B}$
3. Take the inverse Haar wavelet to get $\tilde{A}$

Note that if we have performed lossy compression, then we will never recover $A$. 
Image Compression

The Compression Algorithm

► To uncompress, we simply undo the steps of the previous algorithm.

► For encoded bitstream $C$,
  1. Receive $C$.
  2. Unencode to get $\tilde{B}$
  3. Take the inverse Haar wavelet to get $\tilde{A}$

► Note that if we have performed lossy compression, then we will never recover $A$. 
Image Compression

The Compression Algorithm

- To uncompress, we simply undo the steps of the previous algorithm.
- For encoded bitstream $C$,
  1. Receive $C$.
  2. Unencode to get $\tilde{B}$
  3. Take the inverse Haar wavelet to get $\tilde{A}$
- Note that if we have performed lossy compression, then we will never recover $A$. 

Thursday, 7 June, 2006
Lecture 5
Applications: Compression and Edge Detection
Image Compression
The Compression Algorithm

- To uncompressed, we simply undo the steps of the previous algorithm.
- For encoded bitstream $C$,
  1. Receive $C$.
  2. Unencode to get $\tilde{B}$
  3. Take the inverse Haar wavelet to get $\tilde{A}$
- Note that if we have performed lossy compression, then we will never recover $A$.
Image Compression

The Compression Algorithm

To uncompress, we simply undo the steps of the previous algorithm.

For encoded bitstream $C$,

1. Receive $C$.
2. Unencode to get $\tilde{B}$
3. Take the inverse Haar wavelet to get $\tilde{A}$

Note that if we have performed lossy compression, then we will never recover $A$. 
Image Compression

Modifying the Transform

- The first step in compression is to compute \( i \) iterations of the Haar wavelet transform.
- We just did the work necessary to orthogonalize it, and for this application, we are going to modify the transform so it maps integers to integers.
- The reason we do this is it is much easier for the Huffman coder to encode a list of integers versus machine roundoff of irrational numbers that are returned by the orthogonal transform.
The first step in compression is to compute $i$ iterations of the Haar wavelet transform.

We just did the work necessary to orthogonalize it, and for this application, we are going to modify the transform so it maps integers to integers.

The reason we do this is it is much easier for the Huffman coder to encode a list of integers versus machine roundoff of irrational numbers that are returned by the orthogonal transform.
Image Compression

Modifying the Transform

- The first step in compression is to compute $i$ iterations of the Haar wavelet transform.

- We just did the work necessary to orthogonalize it, and for this application, we are going to modify the transform so it maps integers to integers.

- The reason we do this is it is much easier for the Huffman coder to encode a list of integers versus machine roundoff of irrational numbers that are returned by the orthogonal transform.
Image Compression

Modifying the Transform

The first step in compression is to compute $i$ iterations of the Haar wavelet transform.

We just did the work necessary to orthogonalize it, and for this application, we are going to modify the transform so it maps integers to integers.

The reason we do this is it is much easier for the Huffman coder to encode a list of integers versus machine roundoff of irrational numbers that are returned by the orthogonal transform.
Image Compression

Modifying the Transform
At least we can adapt the Haar transform - for other filters, we have to either round the quantized values (automatically lossy), or go to great lengths to figure out how to have the transform map integers to integers.
Image Compression
Modifying the Transform

It is easy to adapt the Haar transform. Instead of using the orthogonal filter pair

\[ h = \left( \frac{\sqrt{2}}{2}, \frac{\sqrt{2}}{2} \right), \quad g = \left( \frac{\sqrt{2}}{2}, -\frac{\sqrt{2}}{2} \right) \]

we instead use

\[ h = (1, 1), \quad g = \left( \frac{1}{2}, -\frac{1}{2} \right) \]
It is good to ask students what effect these alterations have on the lowpass/highpass nature of the new filters.

You may want to open the Mathematica notebook HaarImageCompression.nb to follow along.
Image Compression
Modifying the Transform

- It is good to ask students what effect these alterations have on the lowpass/highpass nature of the new filters.
- You may want to open the Mathematica notebook `HaarImageCompression.nb` to follow along.
Image Compression

Modifying the Transform

We will use the following image as a running example.
Image Compression
Modifying the Transform

We apply 3 iterations of the modified Haar wavelet transform to obtain
Image Compression
Modifying the Transform

Here is a plot of the cumulative energy of the wavelet transform - we’ll use it to quantize the transform.

As you can see, the energy of this transform is highly concentrated!
As a matter of fact, if we choose to retain 99.99% of the energy of the transform, it turns out we only need to keep the largest (in absolute value) 4617 values.

The image is $200 \times 200$ so we will be converting about 88% of the values to 0.
As a matter of fact, if we choose to retain 99.99% of the energy of the transform, it turns out we only need to keep the largest (in absolute value) 4617 values.

The image is $200 \times 200$ so we will be converting about 88% of the values to 0.
Here is a histogram of the pixel distribution of $A$
Image Compression

Quantizing the Transform

We will retain 99.99% of the energy. So we convert the smallest 35383 (in absolute value) values of the wavelet transform to 0.
Image Compression

Quantizing the Transform

- Note there is very little error in the lowpass portion of the transform.
- Now we are ready to perform Huffman coding.
Image Compression

Quantizing the Transform

- Note there is very little error in the lowpass portion of the transform.
- Now we are ready to perform Huffman coding.
Image Compression
Quantizing the Transform

▶ The original image is $200 \times 200 = 40000$ pixels for a bitstream length of $40000 \cdot 8 = 320000$.

▶ The bitstream length of the Huffman-encoded data is 84066.

▶ That’s about 2.1 bits per pixel (bpp). That’s about a 75% savings!

▶ The entropy of the wavelet transform is 1.61749.
Image Compression
Quantizing the Transform

▶ The original image is $200 \times 200 = 40000$ pixels for a bitstream length of $40000 \cdot 8 = 320000$.

▶ The bitstream length of the Huffman-encoded data is 84066.

▶ That’s about 2.1 bits per pixel (bpp). That’s about a 75% savings!

▶ The entropy of the wavelet transform is 1.61749
The original image is $200 \times 200 = 40000$ pixels for a bitstream length of $40000 \cdot 8 = 320000$.

The bitstream length of the Huffman-encoded data is 84066.

That's about 2.1 bits per pixel (bpp). That's about a 75% savings!

The entropy of the wavelet transform is 1.61749.
Image Compression
Quantizing the Transform

- The original image is $200 \times 200 = 40000$ pixels for a bitstream length of $40000 \cdot 8 = 320000$.
- The bitstream length of the Huffman-encoded data is 84066.
- That’s about 2.1 bits per pixel (bpp). That’s about a 75% savings!
- The entropy of the wavelet transform is 1.61749
Image Compression

Quantizing the Transform

Here is the inverse wavelet transform:

Compressed Image
Image Compression

Quantizing the Transform

and here is the original image:

Original Image
Image Compression
Quantizing the Transform

- If you’re interested, the PSNR is 37.263.
- This value would be useful if we had another compression scheme with which to compare.
- That’s the basic of image compression!
- We can make better filters, smarter quantizers, and more efficient encoders (i.e. JPG2000).
- For the computer session, you’ll be asked to compress a color image.
Image Compression
Quantizing the Transform

- If you’re interested, the PSNR is 37.263.
- This value would be useful if we had another compression scheme with which to compare.
- That’s the basic of image compression!
- We can make better filters, smarter quantizers, and more efficient encoders (i.e. JPG2000).
- For the computer session, you’ll be asked to compress a color image.
Image Compression
Quantizing the Transform

- If you’re interested, the PSNR is 37.263.
- This value would be useful if we had another compression scheme with which to compare.
- That’s the basic of image compression!
- We can make better filters, smarter quantizers, and more efficient encoders (i.e. JPG2000).
- For the computer session, you’ll be asked to compress a color image.
Image Compression

Quantizing the Transform

► If you’re interested, the PSNR is 37.263.
► This value would be useful if we had another compression scheme with which to compare.
► That’s the basic of image compression!
► We can make better filters, smarter quantizers, and more efficient encoders (i.e. JPG2000).
► For the computer session, you’ll be asked to compress a color image.
Image Compression
Quantizing the Transform

▶ If you’re interested, the PSNR is 37.263.
▶ This value would be useful if we had another compression scheme with which to compare.
▶ That’s the basic of image compression!
▶ We can make better filters, smarter quantizers, and more efficient encoders (i.e. JPG2000).
▶ For the computer session, you’ll be asked to compress a color image.
Edge Detection

The Basic Edge Detection Algorithm

- When we say we will edge detect an image, that means we are looking for areas in the images where large changes occur - i.e., boundaries of regions.

- Wavelet transforms are well-suited for this task since the 2D wavelet transform returns differences between an approximation and the original in three separate directions.

- In this part of the lecture, we will present a naive algorithm for performing edge detection - in the computer session, you will be asked to write software for a more refined process.
The Basic Edge Detection Algorithm

Edge Detection
The Basic Edge Detection Algorithm

- When we say we will edge detect an image, that means we are looking for areas in the images where large changes occur - i.e., boundaries of regions.

- Wavelet transforms are well-suited for this task since the 2D wavelet transform returns differences between an approximation and the original in three separate directions.

- In this part of the lecture, we will present a naive algorithm for performing edge detection - in the computer session, you will be asked to write software for a more refined process.
Edge Detection

The Basic Edge Detection Algorithm

- When we say we will edge detect an image, that means we are looking for areas in the images where large changes occur - i.e., boundaries of regions.

- Wavelet transforms are well-suited for this task since the 2D wavelet transform returns differences between an approximation and the original in three separate directions.

- In this part of the lecture, we will present a naive algorithm for performing edge detection - in the computer session, you will be asked to write software for a more refined process.
The Basic Edge Detection Algorithm

Edge Detection

The Basic Edge Detection Algorithm

The algorithm is very simple:

2. Set every value in the blur portion of the transform to 0.
3. Compute $i$ iterations of the inverse transform.

An obvious addition would be to quantize the highpass portions of the transform before inverting.

How would this quantizer compare to image compression?
The Basic Edge Detection Algorithm

Edge Detection
The Basic Edge Detection Algorithm

- The algorithm is very simple:
  2. Set every value in the blur portion of the transform to 0.
  3. Compute $i$ iterations of the inverse transform.

- An obvious addition would be to quantize the highpass portions of the transform before inverting.

- How would this quantizer compare to image compression?
The Basic Edge Detection Algorithm

Edge Detection

The algorithm is very simple:

2. Set every value in the blur portion of the transform to 0.
3. Compute $i$ iterations of the inverse transform.

An obvious addition would be to quantize the highpass portions of the transform before inverting.

How would this quantizer compare to image compression?
The Basic Edge Detection Algorithm

Edge Detection

The Basic Edge Detection Algorithm

The algorithm is very simple:

2. Set every value in the blur portion of the transform to 0.
3. Compute $i$ iterations of the inverse transform.

An obvious addition would be to quantize the highpass portions of the transform before inverting.

How would this quantizer compare to image compression?
The Basic Edge Detection Algorithm

Edge Detection
The Basic Edge Detection Algorithm

- The algorithm is very simple:
  2. Set every value in the blur portion of the transform to 0.
  3. Compute $i$ iterations of the inverse transform.

- An obvious addition would be to quantize the highpass portions of the transform before inverting.

- How would this quantizer compare to image compression?
The Basic Edge Detection Algorithm

Edge Detection

The Basic Edge Detection Algorithm

The algorithm is very simple:

2. Set every value in the blur portion of the transform to 0.
3. Compute $i$ iterations of the inverse transform.

An obvious addition would be to quantize the highpass portions of the transform before inverting.

How would this quantizer compare to image compression?
You may want to work along with the example. It is contained in the Mathematica notebook

HaarEdgeDetection.nb
Edge Detection

An Example of Edge Detection

We will use the image below for our running example.
Edge Detection

An Example of Edge Detection

We begin by computing 2 iterations of the Haar wavelet transform:
An Example of Edge Detection

We convert the blur portion of the transform to a zero matrix:
Edge Detection

An Example of Edge Detection

We invert the modified transform to display the detected edges:
Edge Detection
An Example of Edge Detection

What do you suppose would happen if we used 1 iteration? 3 iterations?

It is quite possible for the Haar transform to “miss” an edge (think about what it does to the simple vector $v = (1, 1, 200, 200)$).

In the Computer Session that follows, you will be asked to think about radically changing the edge detection algorithm and it involves adding redundancy back into the Haar transform.
Edge Detection

What do you suppose would happen if we used 1 iteration? 3 iterations?

It is quite possible for the Haar transform to “miss” an edge (think about what it does to the simple vector \( \mathbf{v} = (1, 1, 200, 200) \)).

In the Computer Session that follows, you will be asked to think about radically changing the edge detection algorithm and it involves adding redundancy back into the Haar transform.
Edge Detection
An Example of Edge Detection

▶ What do you suppose would happen if we used 1 iteration? 3 iterations?

▶ It is quite possible for the Haar transform to “miss” an edge (think about what it does to the simple vector \( \mathbf{v} = (1, 1, 200, 200) \)).

▶ In the Computer Session that follows, you will be asked to think about radically changing the edge detection algorithm and it involves adding redundancy back into the Haar transform.
Edge Detection

An Example of Edge Detection

In their text *Digital Image Processing Using Matlab*, (Prentice Hall, 2004), Gonzalez, Woods, and Eddins describe a simple method for quantizing the elements in the highpass portions of the wavelet transform before inverting.

We will describe it and leave it as an exercise for the Computer Session.
Refining the Quantization Process

Edge Detection
An Example of Edge Detection

- In their text *Digital Image Processing Using Matlab*, (Prentice Hall, 2004), Gonzalez, Woods, and Eddins describe a simple method for quantizing the elements in the highpass portions of the wavelet transform before inverting.

- We will describe it and leave it as an exercise for the Computer Session.

Thursday, 7 June, 2006
Lecture 5
Applications: Compression and Edge Detection
Refining the Quantization Process

Edge Detection
An Example of Edge Detection

Let $S$ be the values in a highpass portion(s) in the wavelet transform.

1. Choose a tolerance $\alpha > 0$ - a stopping criteria.
2. Set $\tau_1$ to be the mean of the min and max of $S$.
3. Divide the values of $S$ into two sets $S_1$ and $S_2$. The values in $S_1$ are smaller than $\tau_1$ and the values in $S_2$ are $\geq \tau_1$.
4. Compute the means $\overline{s_1}$ for $S_1$ and $\overline{s_2}$ for $S_2$ and take $\tau_2$ to be the mean of $\overline{s_1}$ and $\overline{s_2}$.
5. Repeat steps 3 and 4 until $|\tau_{n+1} - \tau_n| < \alpha$. 
Edge Detection
An Example of Edge Detection

Let $S$ be the values in a highpass portion(s) in the wavelet transform.

1. Choose a tolerance $\alpha > 0$ - a stopping criteria.
2. Set $\tau_1$ to be the mean of the min and max of $S$.
3. Divide the values of $S$ into two sets $S_1$ and $S_2$. The values in $S_1$ are smaller than $\tau_1$ and the values in $S_2$ are $\geq \tau_1$.
4. Compute the means $\overline{s}_1$ for $S_1$ and $\overline{s}_2$ for $S_2$ and take $\tau_2$ to be the mean of $\overline{s}_1$ and $\overline{s}_2$.
5. Repeat steps 3 and 4 until $|\tau_{n+1} - \tau_n| < \alpha$. 

University of St. Thomas
Refining the Quantization Process

Edge Detection
An Example of Edge Detection

Let $S$ be the values in a highpass portion(s) in the wavelet transform.

1. Choose a tolerance $\alpha > 0$ - a stopping criteria.
2. Set $\tau_1$ to be the mean of the min and max of $S$.
3. Divide the values of $S$ into two sets $S_1$ and $S_2$. The values in $S_1$ are smaller than $\tau_1$ and the values in $S_2$ are $\geq \tau_1$.
4. Compute the means $\bar{s}_1$ for $S_1$ and $\bar{s}_2$ for $S_2$ and take $\tau_2$ to be the mean of $\bar{s}_1$ and $\bar{s}_2$.
5. Repeat steps 3 and 4 until $|\tau_{n+1} - \tau_n| < \alpha$. 
Refining the Quantization Process

**Edge Detection**

An Example of Edge Detection

Let $S$ be the values in a highpass portion(s) in the wavelet transform.

1. Choose a tolerance $\alpha > 0$ - a stopping criteria.
2. Set $\tau_1$ to be the mean of the min and max of $S$.
3. Divide the values of $S$ into two sets $S_1$ and $S_2$. The values in $S_1$ are smaller than $\tau_1$ and the values in $S_2$ are $\geq \tau_1$.
4. Compute the means $\overline{s_1}$ for $S_1$ and $\overline{s_2}$ for $S_2$ and take $\tau_2$ to be the mean of $\overline{s_1}$ and $\overline{s_2}$.
5. Repeat steps 3 and 4 until $|\tau_{n+1} - \tau_n| < \alpha$. 
Refining the Quantization Process

Edge Detection
An Example of Edge Detection

Let $S$ be the values in a highpass portion(s) in the wavelet transform.

1. Choose a tolerance $\alpha > 0$ - a stopping criteria.
2. Set $\tau_1$ to be the mean of the min and max of $S$.
3. Divide the values of $S$ into two sets $S_1$ and $S_2$. The values in $S_1$ are smaller than $\tau_1$ and the values in $S_2$ are $\geq \tau_1$.
4. Compute the means $s_1$ for $S_1$ and $s_2$ for $S_2$ and take $\tau_2$ to be the mean of $s_1$ and $s_2$.
5. Repeat steps 3 and 4 until $|\tau_{n+1} - \tau_n| < \alpha$. 

Thursday, 7 June, 2006
Lecture 5
Applications: Compression and Edge Detection
Refining the Quantization Process

Edge Detection
An Example of Edge Detection

Once we have $\tau$, we quantize $S$ as follows. If $s_{ij} \in S$, set

$$\tilde{s}_{ij} = \begin{cases} s_{ij}, & s_{ij} > \tau \\ 0, & s_{ij} \leq \tau \end{cases}$$
In the Classroom

Teaching Ideas

► The students like this part of the course - the real applications.

► In image compression, I usually give them labs and exercises where they are expected to fiddle around with the parameters and achieve a certain $\textit{bpp}$.

► They are also asked to think about other quantizing schemes - there have been some pretty exotic quantizing functions and some students have actually went out and did the internet search and found the quantizing function used by JPG2000!
In the Classroom

Teaching Ideas

- The students like this part of the course - the real applications.

- In image compression, I usually give them labs and exercises where they are expected to fiddle around with the parameters and achieve a certain $bpp$.

- They are also asked to think about other quantizing schemes - there have been some pretty exotic quantizing functions and some students have actually went out and did the internet search and found the quantizing function used by JPG2000!
The students like this part of the course - the real applications.

In image compression, I usually give them labs and exercises where they are expected to fiddle around with the parameters and achieve a certain \textit{bpp}.

They are also asked to think about other quantizing schemes - there have been some pretty exotic quantizing functions and some students have actually went out and did the internet search and found the quantizing function used by JPG2000!
I usually leave edge detection as an exam problem and/or I suggest more refined methods of quantizing and ask students to consider it for a semester-ending project.
In the Classroom

Computer Usage

- As you might imagine, we do lots of coding in this chapter.
- The students work in pairs - there tends to be LOTS of Mathematica mistakes here and since my package isn’t adequate in error-handling, lots of frustration. I encourage them to keep cells small - at least until they know portions of the code work and then they can lump them all together.
- I usually offer a sizable amount of extra credit for a student who wants to tackle the problem of writing a nice Huffman encoder - it’s not an easy programming problem!
As you might imagine, we do lots of coding in this chapter.

The students work in pairs - there tends to be LOTS of Mathematica mistakes here and since my package isn’t adequate in error-handling, lots of frustration. I encourage them to keep cells small - at least until they know portions of the code work and then they can lump them all together.

I usually offer a sizable amount of extra credit for a student who wants to tackle the problem of writing a nice Huffman encoder - it’s not an easy programming problem!
In the Classroom

Computer Usage

- As you might imagine, we do lots of coding in this chapter.
- The students work in pairs - there tends to be LOTS of Mathematica mistakes here and since my package isn’t adequate in error-handling, lots of frustration. I encourage them to keep cells small - at least until they know portions of the code work and then they can lump them all together.
- I usually offer a sizable amount of extra credit for a student who wants to tackle the problem of writing a nice Huffman encoder - it’s not an easy programming problem!
In the Classroom

Computer Usage

▶ As you might imagine, we do lots of coding in this chapter.
▶ The students work in pairs - there tends to be LOTS of Mathematica mistakes here and since my package isn’t adequate in error-handling, lots of frustration. I encourage them to keep cells small - at least until they know portions of the code work and then they can lump them all together.
▶ I usually offer a sizable amount of extra credit for a student who wants to tackle the problem of writing a nice Huffman encoder - it’s not an easy programming problem!
Today’s Schedule

9:00-10:15 **Lecture Five**: Applications: Image Compression and Edge Detection

10:15-10:30 ⇒ **Coffee Break** (OSS 235)

10:30-11:45 **Computer Session One**: Applications: Image Compression and Edge Detection

12:00-1:00 **Lunch** (Cafeteria)

1:15-2:30 **Lecture Six**: Daubechies Filters

2:30-2:45 **Coffee Break** (OSS 235)

2:45-4:00 **Computer Session Two**: Daubechies Filters and Transform Modules

4:30-9:00 **Excursion to the Science Museum**

5:30-6:30 **Dinner** (Cafeteria)