Lecture 19
Image processing
Lecture 19 Outline:

Image Processing (in MATLAB):

- Image Processing Toolbox
  - imread
  - imwrite
- Image Data
  - Vectorizing computations
- Example Transformations
  - Mirroring
  - Color to grayscale
  - Negative
  - Median noise filter
  - Edge detection
Images are stored as a bunch of numbers

318-by-25

0

<table>
<thead>
<tr>
<th>49</th>
<th>55</th>
<th>58</th>
<th>59</th>
<th>57</th>
<th>53</th>
</tr>
</thead>
<tbody>
<tr>
<td>60</td>
<td>67</td>
<td>71</td>
<td>72</td>
<td>72</td>
<td>70</td>
</tr>
<tr>
<td>102</td>
<td>108</td>
<td>111</td>
<td>111</td>
<td>112</td>
<td>112</td>
</tr>
<tr>
<td>157</td>
<td>167</td>
<td>169</td>
<td>167</td>
<td>165</td>
<td>164</td>
</tr>
<tr>
<td>196</td>
<td>205</td>
<td>208</td>
<td>207</td>
<td>205</td>
<td>205</td>
</tr>
<tr>
<td>199</td>
<td>208</td>
<td>212</td>
<td>214</td>
<td>213</td>
<td>216</td>
</tr>
<tr>
<td>190</td>
<td>192</td>
<td>193</td>
<td>195</td>
<td>195</td>
<td>197</td>
</tr>
<tr>
<td>174</td>
<td>169</td>
<td>165</td>
<td>163</td>
<td>162</td>
<td>161</td>
</tr>
</tbody>
</table>
Pictures as Arrays

A black and white picture can be encoded as a 2D Array, each element corresponds to a Pixel in the image.

\[ 0 \leq A(i,j) \leq 255 \]

(black) \hspace{1cm} (white)

Other possible ranges are also used, e.g. floating point between 0 and 1. Smallest possible value is black, largest is white, values in between are gray.
Pixel Indexing in MATLAB

For an image A
- $A(1,1)$ is upper left corner pixel of image
- $A(\text{end},1)$ is lower left corner pixel
- $A(1,\text{end})$ is upper right corner
- $A(\text{end},\text{end})$ is lower right corner
Color Images

Each pixel has 3 color components, so can view a color image as 3 different matrices (aka *pages*), one for each color.

MATLAB stacks them in a 3D array, eg

\[
0 \leq A(i,j,1) \leq 255 \quad \text{(red)}
\]
\[
0 \leq A(i,j,2) \leq 255 \quad \text{(green)}
\]
\[
0 \leq A(i,j,3) \leq 255 \quad \text{(blue)}
\]
Storing Images

There are a lot of file formats for images. Some common ones:

JPEG
   (Joint Photographic Experts Group)
GIF
   (Graphics Interchange Format)
PNG
   (Portable Network Graphics)
Storing Images

Why are there so many formats?

- Different Compression Algorithms
  - Lossy (JPEG, GIF)
  - Lossless (PNG)
- Need for additional attributes
  - Patient Name, Radiologist Comments, etc. (DICOM)
- Intellectual Property Rights, Royalties
Compression Idea:
Small Regions May be Similar

FIGURE 27.9
JPEG image division. JPEG transform compression starts by breaking the image into 8×8 groups, each containing 64 pixels. Three of these 8×8 groups are enlarged in this figure, showing the values of the individual pixels, a single byte value between 0 and 255.
Images can be written as a sum of a relatively small number of tables.

1000-by-2000 picture might be well approximated by weighted sums of 100 tables:

2,000,000 vs 300,000 (100 x 3000)
Operations on Images

Amount to operations on 2D and 3D Arrays.

A good place to practice “array” thinking.
Two Problems

We have:
Problem 1

Want:

LawSchoolMirror.jpg
Problem 2

Want:

LawSchoolUpDown.jpg
Solution Framework

Read LawSchool.jpg from disk and convert it into an array.

Manipulate the Array.

Convert the array to a jpg file and write it to memory.
imread

% Read in image as 3D array...
A = imread('LawSchool.jpg');

The color of the pixel at location (i,j) is given by
A(i,j,1) = red value
A(i,j,2) = green value
A(i,j,3) = blue value
The 3D Array

>> [m,n,p] = size(A)

m = 1458 rows
n = 2084 column
p = 3 layers
The Layers

1458-by-2084

A(:,:,1)

1458-by-2084

A(:,:,2)

1458-by-2084

A(:,:,3)
How can we find the Mirror image of A?
% Store left-right mirror of A in array B

\[
\begin{align*}
[nr, nc, np] &= \text{size}(A); \\
\text{for } r &= 1:nr \\
\quad &\text{for } c = 1:nc \\
\quad \quad B(r, c, \phantom{i}) &= A(r, nc-c+1, \phantom{i}); \\
\text{end} \\
\text{end}
\end{align*}
\]
% Store left-right mirror of A
% in array B

[nr, nc, np] = size(A);
for r = 1:nr
    for c = 1:nc
        for p = 1:np
            B(r, c, p) = A(r, nc-c+1, p);
        end
    end
end
A = imread('LawSchool.jpg')
[m,n,p] = size(A);
for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end
imwrite(B,'LawSchoolMirror.jpg')
A = imread('LawSchool.jpg')
[m,n,p] = size(A);
for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end
imwrite(B,'LawSchoolMirror.jpg')

A) row
B) column
C) row+panel
D) column+panel
What does each loop copy to B?

A = imread('LawSchool.jpg')
[m,n,p] = size(A);
for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end
imwrite(B,'LawSchoolMirror.jpg')

A) row
B) column
C) row+panel
D) column+panel
MATLAB Loves to Vectorize!

for j=1:n
    B(:,j,1) = A(:,n+1-j,1)
    B(:,j,2) = A(:,n+1-j,2)
    B(:,j,3) = A(:,n+1-j,3)
end

B = A(:,end:-1:1,:);
A = imread('LawSchool.jpg');
B = A(:,end:-1:1,:);
Imwrite(A,'LawSchoolMirror.jpg');
The Upside Down Image

A = imread('LawSchool.jpg');
C = _______
imwrite(C,'LawSchoolUpDown.jpg');

A) C(:,,:,end:-1:1)
B) C(:,end:-1:1,:)  
C) C(end:-1:1,:,:)
D) I don’t know
The Upside Down Image

A = imread('LawSchool.jpg');
C = _______
imwrite(C,'LawSchoolUpDown.jpg');

A) C(:, :, end:-1:1)
B) C(:, end:-1:1, :)
C) C(end:-1:1, :, :)
D) I don’t know
New Problem
Color $\rightarrow$ Black and White

Have:
New Problem
Color → Black and White

Want:
A = imread('LawSchool.jpg');
bwA = rgb2gray(A);
imwrite(bwA,'LawSchoolBW.jpg')
How Does the Conversion Work?

<table>
<thead>
<tr>
<th>r</th>
<th>g</th>
<th>b</th>
<th>gray</th>
</tr>
</thead>
<tbody>
<tr>
<td>167</td>
<td>219</td>
<td>241</td>
<td>206</td>
</tr>
<tr>
<td>66</td>
<td>35</td>
<td>15</td>
<td>42</td>
</tr>
<tr>
<td>95</td>
<td>14</td>
<td>20</td>
<td>39</td>
</tr>
<tr>
<td>163</td>
<td>212</td>
<td>242</td>
<td>201</td>
</tr>
<tr>
<td>182</td>
<td>228</td>
<td>215</td>
<td>213</td>
</tr>
<tr>
<td>225</td>
<td>244</td>
<td>222</td>
<td>236</td>
</tr>
<tr>
<td>136</td>
<td>199</td>
<td>240</td>
<td>185</td>
</tr>
</tbody>
</table>

It’s a weighted average
Weighted average of the RGB values

\[ M(i,j) = 0.3R(i,j) + 0.59G(i,j) + 0.11B(i,j) \]

for \( i = 1:m \)
  for \( j = 1:n \)
    \[ M(i,j) = 0.3R(i,j) + 0.59G(i,j) + 0.11B(i,j) \]
  end
end

scalar operation
Why a Weighted Average?

• Due to the way our visual system works
• We are most sensitive to Green, then Red then Blue
• Other approaches are close
  – Equal Weights (i.e. \((r+g+b)/3\))
  – \(\max(r, g, b)\)
Weighted average of the RGB values

\[ M = 0.3R + 0.59G + 0.11B \]
Coding Average

```matlab
bwA = uint8(zeros(m,n))
for i=1:m
    for j = 1:n
        bwA(i,j) = ( A(i,j,1) + ... + A(i,j,2) + A(i,j,3) )/3;
    end
end
imwrite(bwA, 'LawSchoolBW.jpg')
```

Type **uint8**: unsigned 8-bit integers (0,1,2,...,255)
Whoa! Why is it so bad?

- `uint8`
  - Values always between 0 and 255
  - “uint8 arithmetic clips values”
    - Negative => Zero
    - 256 and larger => 255
  - E.g.,
    - `uint8(255)+100 == 255`

Convert image to double for average to work
Work with Doubles, then convert back

\[ A = \text{double}(A); \]
\[ \text{bwA} = \text{uint8}(\text{zeros}(m,n)) \]
for i=1:m
    for j = 1:n
        \[ \text{bwA}(i,j) = (A(i,j,1) + \ldots + A(i,j,2) + A(i,j,3))/3; \]
    end
end
\[ \text{imwrite(bwA,'LawSchoolBW.jpg')} \]

Type \texttt{uint8}: unsigned 8-bit integers (0,1,2,…,255)
bwA = uint8(zeros(m,n))
for i=1:m
    for j = 1:n
        bwA(i,j) = max([A(i,j,1) ... 
                        A(i,j,2)  A(i,j,3)])
    end
end
imwrite(bwA,‘LawSchoolBW.jpg')
Vectorized Max?

• max(A, [ ], n) finds the max along dimension n
  
  – For n=1, returns \( r(j,k) = \max(A(:,j,k)) \)
  – For n=2, returns \( r(i,k) = \max(A(i,:,k)) \)
  – For n=3, returns \( r(i,j) = \max(A(i,j,:)) \)

\[ M = \max(A,[ ],3) \]
Problem: Produce a Negative
If matrix $A$ represents the image and

$$B(i,j) = 255 - A(i,j)$$

for all $i$ and $j$, then $B$ will represent the negative.
function newIm = toNegative(im)
% newIm is the negative of image im
% im, newIm are 3-d arrays; each component is uint8

[nr,nc,np]= size(im);  % dimensions of im
newIm= zeros(nr,nc,np); % initialize newIm
newIm= uint8(newIm); % Type for image color values

for r = 1:nr
    for c = 1:nc
        for p = 1:np
            newIm(r,c,p)= 255 - im(r,c,p);
        end
    end
end
Vectorized toNegative

function newIm = toNegative(im)
% newIm is the negative of image im
% im, newIm are 3-d arrays
newIm = 255-im;

This is cleaner, clearer, and less error prone. But, always keep complexity in mind.
Previous operations were all element-wise, now we’ll look at some region based operations

Filtering Noise
Edge Detection
Can We Filter Out the “Noise”?
Dirt!

Note how the “dirty pixels” look out of place
Idea

Assign “typical” neighborhood gray values to “dirty pixels”
Getting Precise

“Typical neighborhood gray values”

Could use Median or Mean

We’ll look at “Median Filtering” first…
Median Filtering

At each pixel, replace its gray value by the median of the gray values in the “neighborhood” i.e. a small region surrounding the pixel.
Using a radius 1 “Neighborhood”

Before

<table>
<thead>
<tr>
<th>7</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

After

<table>
<thead>
<tr>
<th>7</th>
<th>7</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>
How to Visit Every Pixel

\begin{align*}
\text{for } i &= 1 : m \\
& \quad \text{for } j = 1 : n \\
& \quad \quad \text{Compute new gray value for pixel } (i, j). \\
& \quad \text{end} \\
& \text{end}
\end{align*}
Replace the median of the values under the window.
Original:

\[
i = 1 \quad \text{and} \quad j = 2
\]

Filtered:

Replace the median of the values under the window.
Replace \( \Box \) with the median of the values under the window.
Original:

\[
i = 1 \\
j = n
\]

Filtered:

Replace \( \times \) with the median of the values under the window.
Original:

Filtered:

Replace \( \times \) with the median of the values under the window.
Replace $\times$ with the median of the values under the window.
Original:

\[ i = m \]
\[ j = n \]

Filtered:

Replace \( \times \) with the median of the values under the window.
What We Need…

(1) A function that computes the median value in a 2-dimensional array C:

\[ m = \text{medVal}(C) \]

(2) A function that builds the filtered image by using median values of radius r neighborhoods:

\[ B = \text{medFilter}(A,r) \]
Computing Medians

\[ x : \begin{array}{ccccccccc} 21 & 89 & 36 & 28 & 19 & 88 & 43 \end{array} \]

\[ x = \text{sort}(x) \]

\[ x : \begin{array}{ccccccccc} 19 & 21 & 28 & 36 & 43 & 88 & 89 \end{array} \]

\[ n = \text{length}(x); \quad \% \quad n = 7 \]

\[ m = \text{ceil}(n/2); \quad \% \quad m = 4 \]

\[ \text{med} = x(m); \quad \% \quad \text{med} = 36 \]

If \( n \) is even, then use:

\[ \text{med} = \frac{\left( x(m) + x(m+1) \right)}{2} \]
Median of a 2D Array

```matlab
function med = medVal(C)
[p,q] = size(C);
x = [];
for k=1:p
    x = [x C(k,:)];
end
Compute median of x and assign to med.
```
Vectorized Median

function med = medVal(C)

med = median(C(:));
Back to Filtering…

\[
\begin{array}{c}
\text{m} = 9 \\
\text{n} = 18
\end{array}
\]

\[
\text{for } \text{i}=1:\text{m} \\
\text{for } \text{j}=1:\text{n} \\
\quad \text{Compute new gray value for pixel (i,j).} \\
\text{end} \\
\text{end}
\]
Window Inside…

\[ n = 18 \]

\[ m = 9 \]

New gray value for pixel (7,4) =

\[ \text{medVal}( A(6:8,3:5) ) \]
Window Partly Outside…

New gray value for pixel (7,1) =

\[
\text{medVal}( A(6:8,1:2) )
\]
Window Partly Outside…

New gray value for pixel (9,18) =

\[ \text{medVal}( A(8:9,17:18) ) \]
function B = medFilter(A,r)
% B from A via median filtering
% with radius r neighborhoods.

[m,n] = size(A);
B = uint8(zeros(m,n));
for i=1:m
    for j=1:n
        C = pixel (i,j) neighborhood
        B(i,j) = medVal(C);
    end
end
The Pixel \((i,j)\) Neighborhood

\[
\begin{align*}
  i_{\text{Min}} &= \max(1,i-r) \\
  i_{\text{Max}} &= \min(m,i+r) \\
  j_{\text{Min}} &= \max(1,j-r) \\
  j_{\text{Max}} &= \min(n,j+r) \\
  C &= A(i_{\text{Min}}:i_{\text{Max}},j_{\text{Min}}:j_{\text{Max}})
\end{align*}
\]
B = medFilter(A)
What About Using the Mean instead of the Median?

Replace each gray value with the average gray value in the radius r neighborhood.
Mean Filter with $r = 10$
### Medians vs Means

$$A =$$

<table>
<thead>
<tr>
<th>150</th>
<th>151</th>
<th>158</th>
<th>159</th>
<th>156</th>
</tr>
</thead>
<tbody>
<tr>
<td>153</td>
<td>151</td>
<td>156</td>
<td>155</td>
<td>151</td>
</tr>
<tr>
<td>150</td>
<td>155</td>
<td>152</td>
<td>154</td>
<td>159</td>
</tr>
<tr>
<td>156</td>
<td>154</td>
<td>152</td>
<td>158</td>
<td>152</td>
</tr>
<tr>
<td>152</td>
<td>158</td>
<td>157</td>
<td>150</td>
<td>157</td>
</tr>
</tbody>
</table>

Median = 154  
Mean = 154.2
# Medians vs Means

\[ A = \]

\[
\begin{array}{cccccc}
150 & 151 & 158 & 159 & 156 \\
153 & 151 & 156 & 155 & 151 \\
150 & 155 & 0 & 154 & 159 \\
156 & 154 & 152 & 158 & 152 \\
152 & 158 & 157 & 150 & 157 \\
\end{array}
\]

*Median = 154  Mean = 148.2*
Why it Fails

The mean does not capture representative values.
And Median Filters Leave Edges (Pretty Much) Alone

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
<td>100</td>
</tr>
</tbody>
</table>

Inside the box, the 200’s stay at 200 and the 100’s stay at 100.
Finding Edges
What is an Edge?

Near an edge, grayness values change abruptly

```
200  200  200  200  200  200  200  200
200  200  200  200  200  200  200  100
200  200  200  200  200  100  100  100
200  200  200  100  100  100  100  100
200  200  100  100  100  100  100  100
200  100  100  100  100  100  100  100
```
General plan for showing the edges in an image

- Identify the “edge pixels”
- Highlight the edge pixels
  - make edge pixels white; make everything else black

```
200  200  200  200  200  200
200  200  200  200  200  100
200  200  200  200  100  100
200  200  200  100  100  100
200  200  100  100  100  100
200  100  100  100  100  100
```
General plan for showing the edges in in image

• Identify the “edge pixels”
• Highlight the edge pixels
  – make edge pixels white; make everything else black

```
200 200 200 200 200 200
200 200 200 200 200 100
200 200 200 200 100 100
200 200 200 100 100 100
200 200 100 100 100 100
200 100 100 100 100 100
```

BLACK

WHITE

BLACK
The Rate-of-Change-Array

Suppose \( A \) is an image array with integer values between 0 and 255.

\( B(i,j) \) be the maximum difference between \( A(i,j) \) and any of its eight neighbors.
The Rate-of-Change-Array

Suppose $A$ is an image array with integer values between 0 and 255.

Let $B(i,j)$ be the maximum value in

$$A(\max(1,i-1):\min(m,i+1),... \max(1,j-1):\min(n,j+1)) - A(i,j)$$

*Neighborhood of $A(i,j)$*
Rate-of-change example

Be careful! In “uint8 arithmetic”
57 - 60 is 0
function Edges(jpgIn,jpgOut,tau)
% jpgOut is the “edge diagram” of image jpgIn.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.

A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
    for j = 1:n
        B(i,j) = ????
    end
end
end
Recipe for rate-of-change \( B(i,j) \)

% The 3-by-3 subarray that includes
% \( A(i,j) \) and its 8 neighbors
Neighbors = A(i-1:i+1,j-1:j+1);

% Subtract \( A(i,j) \) from each entry
Diff = abs(double(Neighbors) - ... double(A(i,j)));

% Compute largest value in each column
colMax = max(Diff);

% Compute the max of the column max's
B(i,j) = max(colMax);
function Edges(jpgIn,jpgOut,tau)
% jpgOut is the “edge diagram” of image jpgIn.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.

A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
    for j = 1:n
        B(i,j) = ?????
    end
end
end
function Edges(jpgIn,jpgOut,tau)
% jpgOut is the “edge diagram” of image jpgIn.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.

A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
    for j = 1:n
        Neighbors = A(max(1,i-1):min(i+1,m), ...
                       max(1,j-1):min(j+1,n));
        B(i,j)=max(max(abs(double(Neighbors)– ... 
                        double(A(i,j)))));
    end
end
“Edge pixels” are now identified; display them with maximum brightness (255)

\[
A =
\begin{bmatrix}
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 1 & 1 & 90 & 90 \\
1 & 1 & 1 & 90 & 90 & 90 \\
1 & 1 & 90 & 90 & 90 & 90 \\
1 & 1 & 90 & 90 & 90 & 90
\end{bmatrix}
\]

\[B(i,j) =
\begin{bmatrix}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 89 & 89 & 89 \\
0 & 0 & 89 & 89 & 0 & 0 \\
0 & 89 & 89 & 0 & 0 & 0 \\
0 & 89 & 0 & 0 & 0 & 0 \\
0 & 89 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

\[
\text{if } B(i,j) > \tau \\
B(i,j) = 255; \\
\text{end}
\]
function Edges(jpgIn,jpgOut,tau)
% jpgOut is the “edge diagram” of image jpgIn.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.
A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
    for j = 1:n
        Neighbors = A(max(1,i-1):min(i+1,m), ...
                       max(1,j-1):min(j+1,n));
        B(i,j)=max(max(abs(double(Neighbors)- ...
                        double(A(i,j)))));
        if B(i,j) > tau
            B(i,j) = 255;
        end
    end
end
function Edges(jpgIn, jpgOut, tau)
% jpgOut is the “edge diagram” of image jpgIn.
% At each pixel, if rate-of-change > tau
% then the pixel is considered to be on an edge.
A = rgb2gray(imread(jpgIn));
[m,n] = size(A);
B = uint8(zeros(m,n));
for i = 1:m
    for j = 1:n
        Neighbors = A(max(1,i-1):min(i+1,m), ... 
                      max(1,j-1):min(j+1,n));
        B(i,j)=max(max(abs(double(Neighbors)- ... 
                        double(A(i,j))));
        if B(i,j) > tau
            B(i,j) = 255;
        end
    end
end
imwrite(B, jpgOut, ’jpg’)
Threshold = 40
Threshold = 20
Threshold = 30