Image Processing

Gordon Erlebacher

What is an image?

Rectangular region, modeled by a matrix

Each matrix element represents a pixel (pixel element)

Each pixel has a color, typically represented as a triplet (RGB)

R: red
G: green
B: blue

What is a matrix

- A matrix is:
  - a table of numbers with M rows and N colums
    - every row has the same size
    - every column as the same size
      columns do not have labels
    - almost always, all matrix elements have the same type

Creating matrices

- a = matrix(1:20)
  b = matrix(1:20,c(2,10)) # 2 rows, 10 col
  d = matrix(1:20, c(10,2)) # 10 rows, 2 col

  d[2,] # extract row 2, produce a vector
  d[,2] # extract column 2, produce a vector
  d[2:3, 1:3] # extract a smaller matrix

Functions to try on matrices

http://www.r-tutor.com/r-introduction/matrix

str()
summary()
class()

colnames()
names()
extraction [3], [3,4]
[3:4, 4:5]

?matrix
??matrix

mean()
as.matrix()
is.matrix()

as.data.frame()
is.data.frame()

What is an image?

Rectangular region, modeled by a matrix

Each matrix element represents a pixel (pixel element)

Each pixel has a color, typically represented as a triplet (RGB)

R: red
G: green
B: blue

ImageJ

- Framework to perform image processing tasks
- Manipulate color, compute statistics, enhance image, detect edges
- Many tools
- Extensible program, etc.

ImageJ

- We first use ImageJ to demonstrate various concepts related to images (on Tuesday)
- We then describe ImageJ and its use a little more in detail (on Thursday)

Low resolution image

Image size: 380x252 = 95,760 pixels

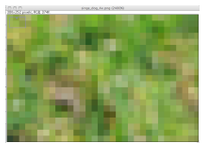
A 10 Mpix camera usually has on the order of

3,000 x 3,000 pixels
= 9 million pixels!

High resolution

Zoom in on picture

4x zoom factor



Color image in text form [380x252] pixels

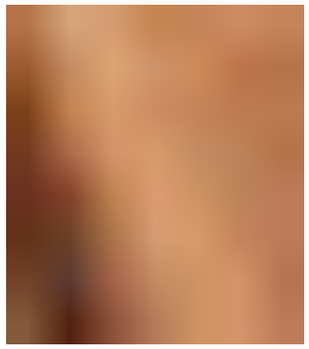
48.667 40.667 52.667 72.000 76.333 43.000 57.667 92.667 119.000121.667107.66786.000 72.667 70.667 85.000 92.333 82.000 88.667 88.333 88.000 99.000 101.66
103.66792.333 74.000 90.333 103.66797.333 98.667 101.333102.33397.000 87.333 102.00099.333 92.000 85.667 80.333 93.333 97.000 98.667 77.000 73.000 71.667
82.000 92.333 92.333 82.000 86.667 87.000 81.667 98.333 114.667122.00096.000 55.333 64.333 60.333 43.333 35.000 51.333 82.333 89.333 90.667 128.00058.667
36.000 21.000 18.333 14.667 12.333 15.333 12.667 21.667 44.000 77.333 104.667112.333109.667106.33393.667 98.667 107.667107.333105.667119.000130.333142.66
156.000153.333157.667155.333150.667161.333162.333166.333162.333165.667159.667153.333157.333163.333168.000175.000168.000167.000164.000170.000
167.000161.000168.333165.333165.333157.667163.333173.333174.000176.667194.333218.333229.000241.667247.667247.667252.667253.667251.667254.333
253.000255.000255.000255.000254.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000254.000
254.000253.667252.333250.333250.667246.000245.000246.000238.333221.333211.333215.667214.000214.000213.000211.667211.667207.667204.000204.000
204.667207.000210.667212.000212.333216.333218.333216.000212.667210.667196.333158.000127.667124.667123.667116.000128.333137.667152.667161.000
161.333159.667164.000159.333145.333105.333111.667109.00081.667 85.000 105.000120.667124.667110.66798.333 98.000 100.000103.667113.333112.333119.667
144.333150.000135.333117.000105.333111.333126.667125.333101.00098.667 117.333123.333117.000114.333118.000116.667122.333128.333140.000141.000108.333
72.333 65.667 81.667 76.667 80.333 85.333 81.000 102.667140.667148.667131.667121.667121.667109.33389.000 85.667 90.000 95.333 103.667121.000124.33
125.333133.333126.333111.000101.667103.33398.333 95.000 95.000 97.000 102.000107.000105.667103.000108.333100.00087.667 73.667 70.333 73.667 81.000 87.333
107.667117.333113.333116.333121.333112.333100.00091.667 94.333 96.333 95.667 95.667 106.667119.333123.333112.000111.667123.000125.000118.000109.333
100.333101.000120.667124.000109.66791.333 92.667 100.000105.333114.333118.333108.00097.333 102.00093.667 101.667107.333103.333101.000101.667107.333
156.667117.66787.667 78.333 92.000 113.000123.000112.00087.000 79.667 88.000 90.667 86.333 86.333 79.333 80.333 81.333 81.000 59.000 45.333 75.333 90.000
95.667 88.667 82.000 77.000 74.667 84.000 94.667 87.000 76.000 76.667 73.000 74.333 83.333 83.000 85.000 80.333 84.333 88.000 92.000 92.000 92.667 104.00
118.000117.000106.000100.00092.333 79.333 79.000 91.000 98.333 104.000109.333118.333115.66798.333 95.000 96.333 91.000 78.667 85.000 85.667 84.333
73.333 58.000 48.333 62.667 68.000 48.000 45.000 78.333 112.000116.667115.333110.667101.33388.333 91.333 84.667 81.000 94.333 92.667 86.000 99.667 112.00
115.33399.667 68.000 89.333 96.333 95.000 107.667113.333109.00089.667 82.000 94.000 93.667 87.333 86.667 85.667 98.000 106.000107.66786.333 79.333 86.000
104.000102.66787.333 83.333 95.333 95.000 88.333 99.333 121.000141.333130.66797.667 71.333 58.333 36.667 30.333 54.000 80.000 82.667 68.000 64.333 111.66
49.667 21.333 17.667 12.000 12.333 13.333 14.000 12.667 16.000 40.667 75.333 104.000109.333103.66789.667 94.667 103.667114.000120.333122.333135.333143.00
154.000153.667154.667158.333157.333164.333165.333162.667162.667164.667160.000155.333158.333160.333164.667173.333169.000165.000163.000169.000
168.000163.000164.000161.000161.000163.333168.333169.333174.333185.667203.000229.667238.000247.667249.667248.667251.667252.667252.667255.000
254.000255.000255.000254.000254.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000255.000254.000
254.000253.667252.333250.333249.667245.000244.000247.000238.667222.333210.333212.667210.000211.667211.667210.667212.667211.667208.667207.000
203.000202.667206.000210.00021

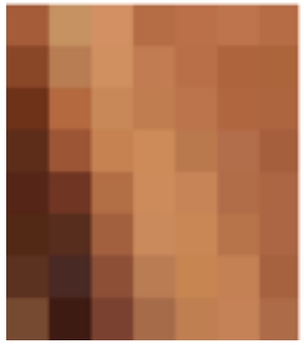
Small Selection

380x252 pixels; RGB; 374K

Matrix size 7x8

Text format below are not triplets





81.000 92.333 86.667 71.667 66.667 79.333 86.333
92.333 89.667 89.667 68.000 74.000 80.667 85.667
89.667 81.667 88.000 69.667 68.000 88.667 85.333
98.667 87.333 81.000 75.000 71.667 72.000 82.333
91.333 87.333 98.000 74.000 78.667 72.667 80.333
85.333 86.000 91.333 84.667 81.667 78.000 86.333
94.333 86.333 90.000 87.000 81.000 83.667 81.000
104.333 83.333 85.000 89.333 89.333 85.000 82.000

7 rows, 8 columns

Represent gray values

How to get text with triplets of color?

- On triplet represents (Red, Green, Blue)
- Red -- R
  Green -- G
  Blue -- B

Read text image into R

- Average color of image
- Use 7x8 image to illustrate

Read image into R

```
#file to read into R
file = "dog_tiny_7x8.txt"

# data is space-delimited
dfr = read.table(file)
class(dfr)                    # dataframe

# transform data frame into a matrix
mat = as.matrix(dfr)

# What is retrieved are numbers in the range [0-255],
# which represent a gray value (black to white)
```

Matrix versus dataframe

```
mean(mat)
[1] 83.57738
```

mean of entire set of numbers

```
colMeans(dfr)
      V1       V2       V3       V4       V5       V6       V7
92.12488 86.74988 88.70838 77.41675 76.37512 80.00012 83.66650
```

A dataframe is a collection of columns. **colMeans(..)** takes the average of each column. They are not named by default since they were not named in the file.

Mean of matrix rows
Mean of matrix columns

rmeans = rowMeans(mat)
cmeans = colMeans(mat)

rowMeans= **80.57143** 82.85729 81.57157 81.14286 83.19043 **84.76186** 86.19043 88.33314

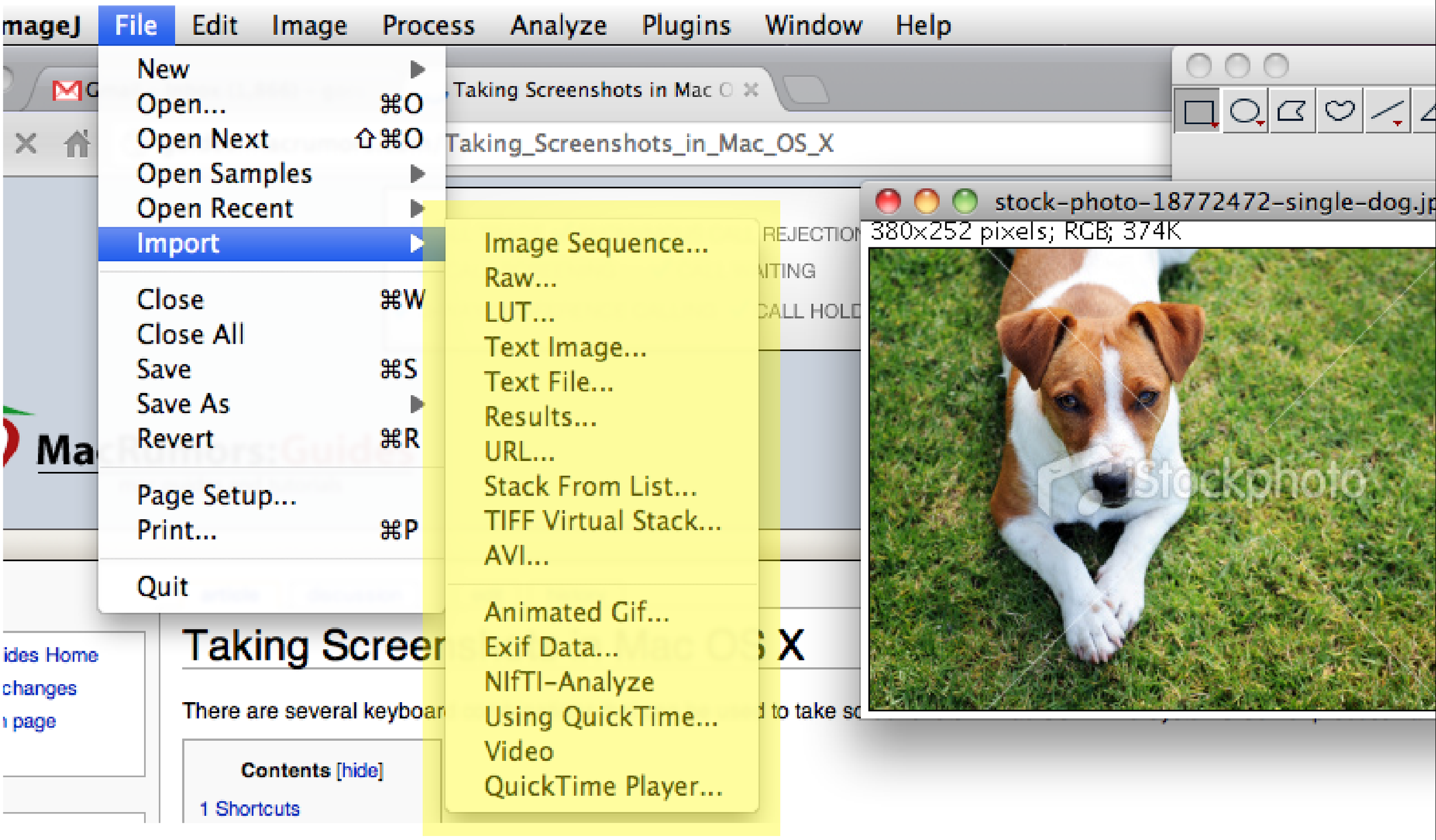
colMeans= 92.12488 **86.74988** 88.70838 77.41675 **76.37512** 80.00012 83.6665

81.000	92.333	86.667	71.667	66.667	79.333	86.333
92.333	89.667	89.667	68.000	74.000	80.667	85.667
89.667	81.667	88.000	69.667	68.000	88.667	85.333
98.667	87.333	81.000	75.000	71.667	72.000	82.333
91.333	87.333	98.000	74.000	78.667	72.667	80.333
85.333	86.000	91.333	84.667	81.667	78.000	86.333
94.333	86.333	90.000	87.000	81.000	83.667	81.000
104.333	83.333	85.000	89.333	89.333	85.000	82.000

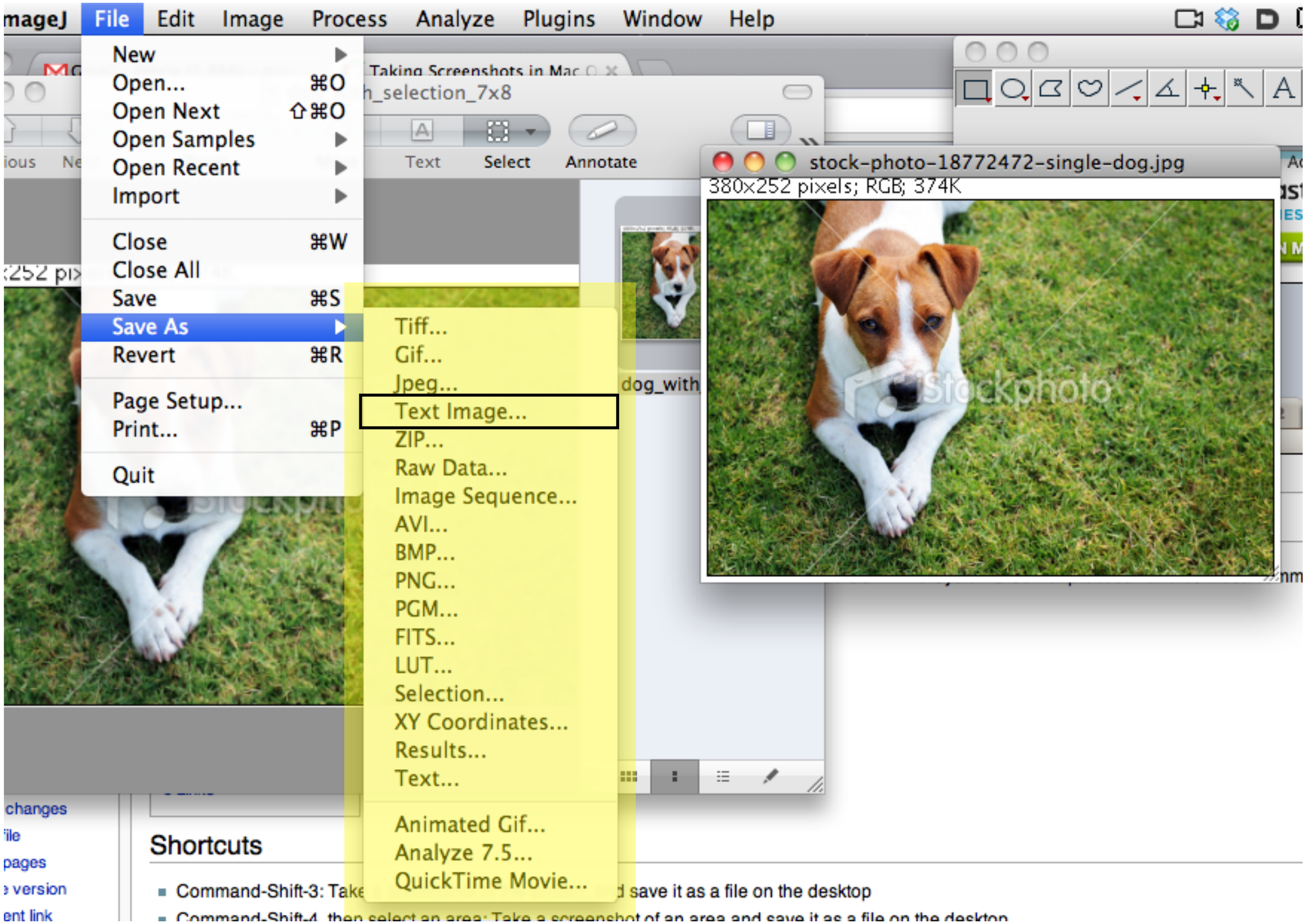
Return to original image of dog

Read image into ImageJ, print it out in text form, and plot its histogram in R

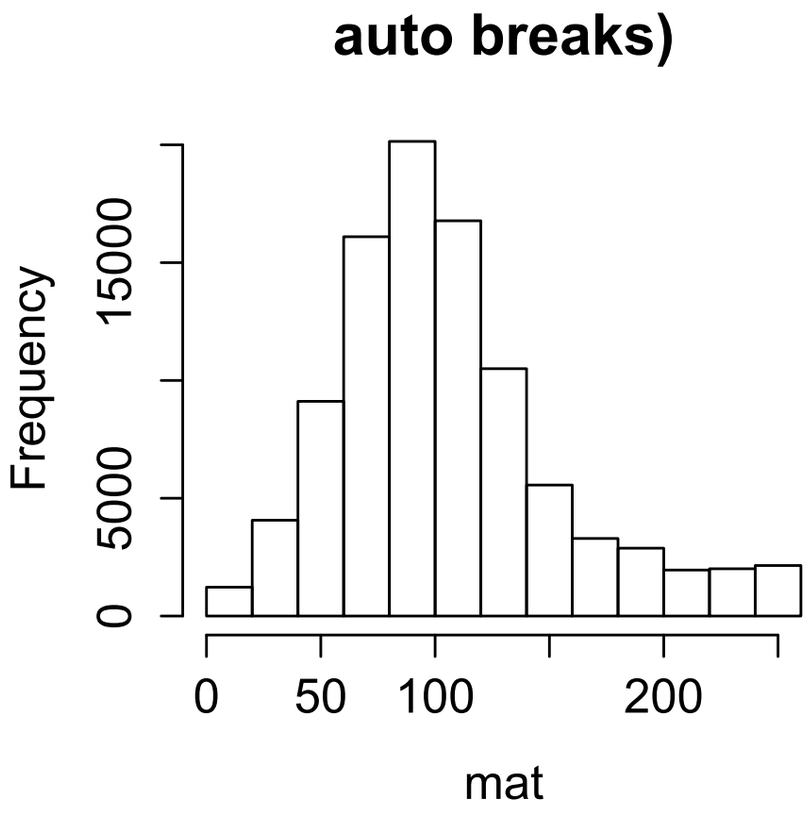
Import Formats

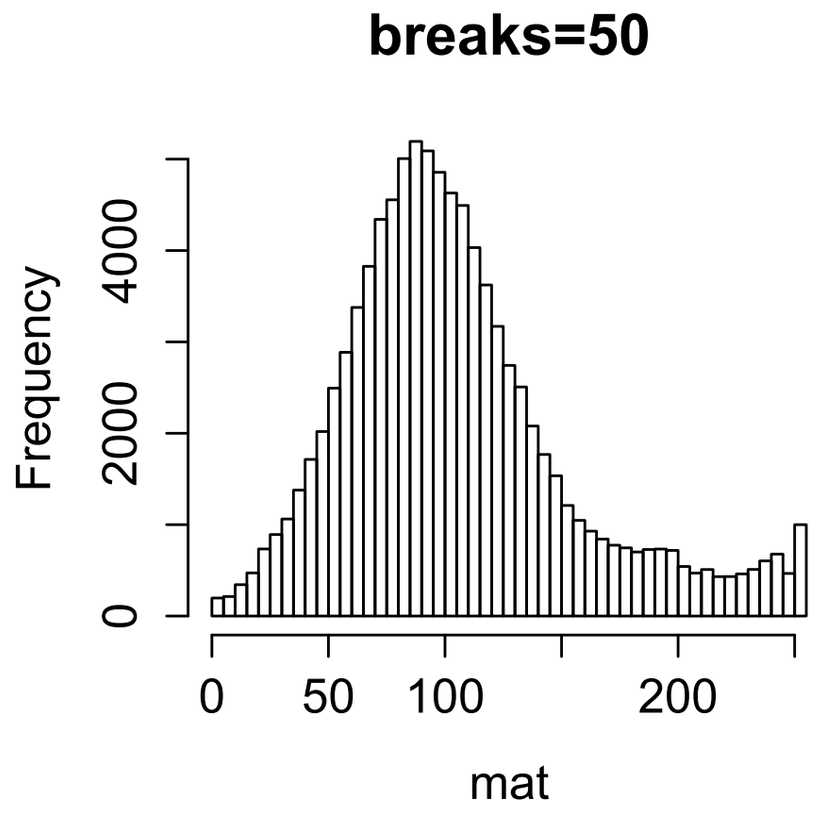


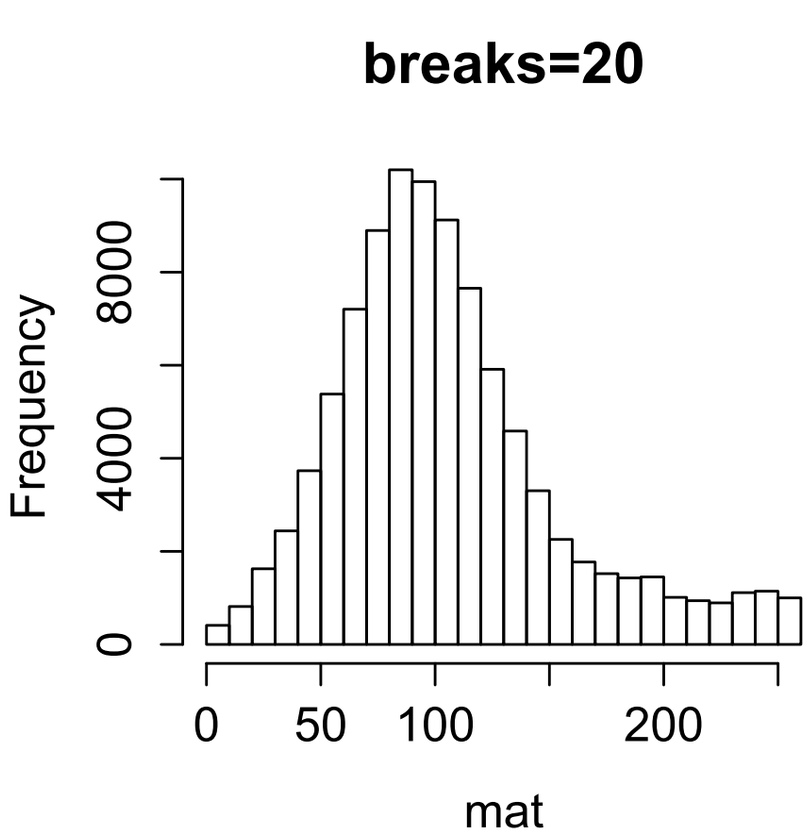
Export Formats



- Export Dog file in **Text Image** format
- The default file name is the same as the original file, except for the file extension
  - stock-photo-18772472-single-dog**.txt**
- Read file in R using read.table(), and plot its histogram








```
cat(str(mat), "\n")
num [1:252, 1:380] 48.7 73.3 81 77.3 77.7 ...
 - attr(*, "dimnames")=List of 2
  ..$ : NULL
  ..$ : chr [1:380] "V1" "V2" "V3" "V4" ...
```

Normal distribution?

R code

```
file = "stock-photo-18772472-single-dog.txt"
mat = as.matrix(read.table(file))

par(mfcol=c(2,2), ps=16)
hist(mat, main="auto breaks)")
hist(mat, main="breaks=20", breaks=20)
hist(mat, main="breaks=50", breaks=50)

cat(str(mat), "\n")
```

Checking normality

- Use shapiro.test() to find out whether a sample of random numbers taken from a population is close to normal
- ?test.shapiro()
  Performs the Shapiro-Wilk test of normality.

Effect of source(…)

- Log into R
- Type

```
>   10:20
```

- R returns

```
[1] 10 11 12 13 14 15 16 17 18 19 20
```

- We are now typing at the console

- Store the above command in the file vector.r and
- source("vector.r") # typed at the console
- R returns nothing
- But ...

```
> source("vector.r", print.eval=T)
 [1] 10 11 12 13 14 15 16 17 18 19 20
>
```

The outputs go to the screen without the need for *print(…)* or *cat(…)*

Source with echo

Only prints the *output* of R commands

```
> source("vector.r", print.eval=T)
 [1] 10 11 12 13 14 15 16 17 18 19 20
>
```

Prints the *input and output* of R commands

```
> source("vector.r", echo=T)

> 10:20
 [1] 10 11 12 13 14 15 16 17 18 19 20
```

Return to imaging

Input text file into imageJ

Original image

Image output from imageJ
in text_image format

ImageJ

Grayscale

- The image started off in RGB format when input to imageJ
- The image got converted to grayscale when exported from imageJ in text_image format

- If it is possible to output all three colors in text_image format, I have not figured it out.
- We will try to do this
- Two ways to compute grayscale (see preferences of ImageJ)

Gray

- The gray color is an RGB triplet with
  R = G = B
- By convention, the gray value is computed as
  gray = 0.2989*R + 0.5870*G + 0.1140*B
- The default approach in ImageG is to compute
  gray = mean(R, G, B)

Convert to Grayscale

- Several options
  - RGB image: transform to 8-bit image
  - Stacked RGB => RGB non-stacked => 8-bit image
  - Save grayscale images: *text image* format

Color Channels

- In RGB color mode, there are three channels:
  - Red (R)
  - Green (G)
  - Blue (B)
- Each channel takes value from 0 to 255
  - 0 is black, 255 is maximum saturation
- How to extract channels in ImageJ?

stock-photo-18772472-single-dog.jpg (green)

380x252 pixels; 8-bit; 94K

stock-photo-18772472-single-dog.jpg (red)

380x252 pixels; 8-bit; 94K

stock-photo-18772472-single-dog.jpg (blue)

380x252 pixels; 8-bit; 94K

stock-photo-18772472-single-dog.jpg

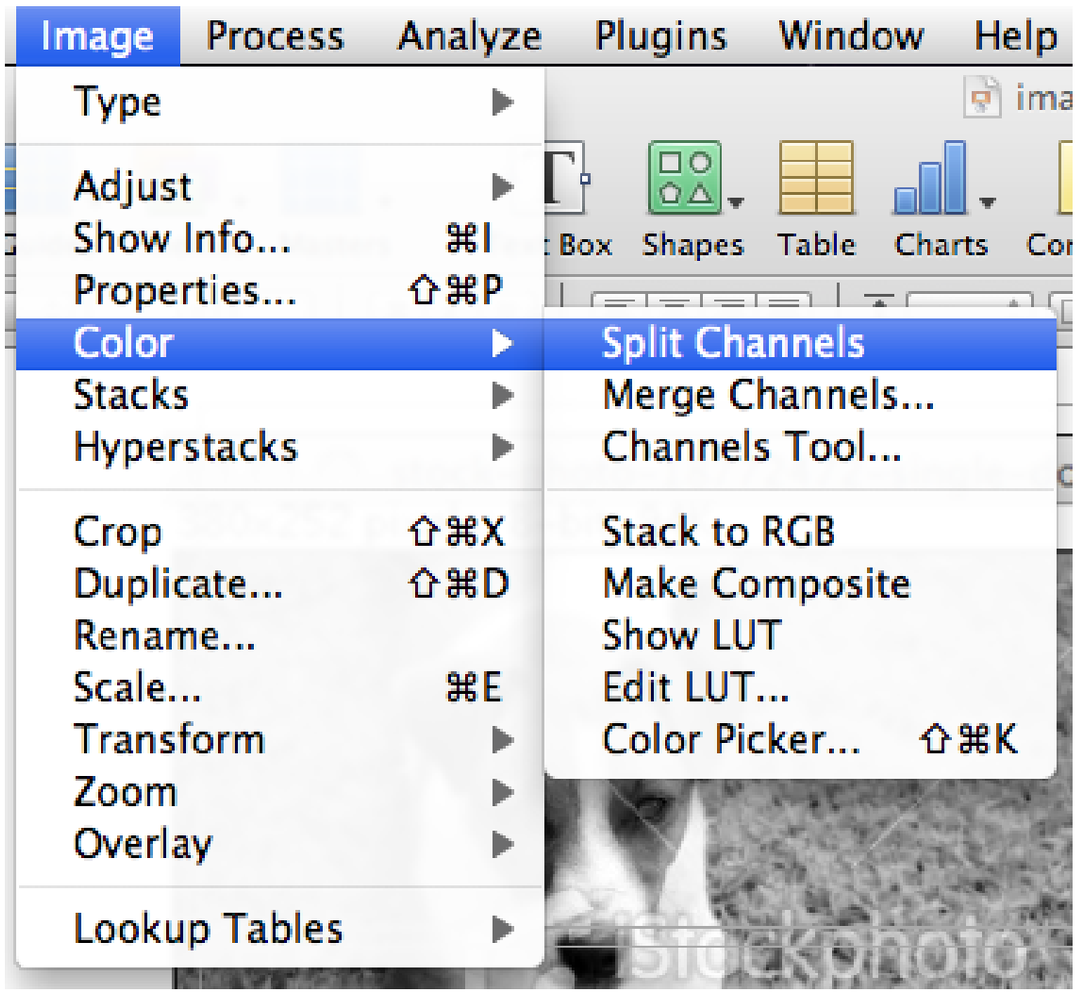
Gray

380x252 pixels; 32-bit; 374K

Channels

- RGB image => split channels
  - 3 images, one per channel, displayed as gray
- Stacked image ==> split channels
  - 3 images, one per channel, but gray

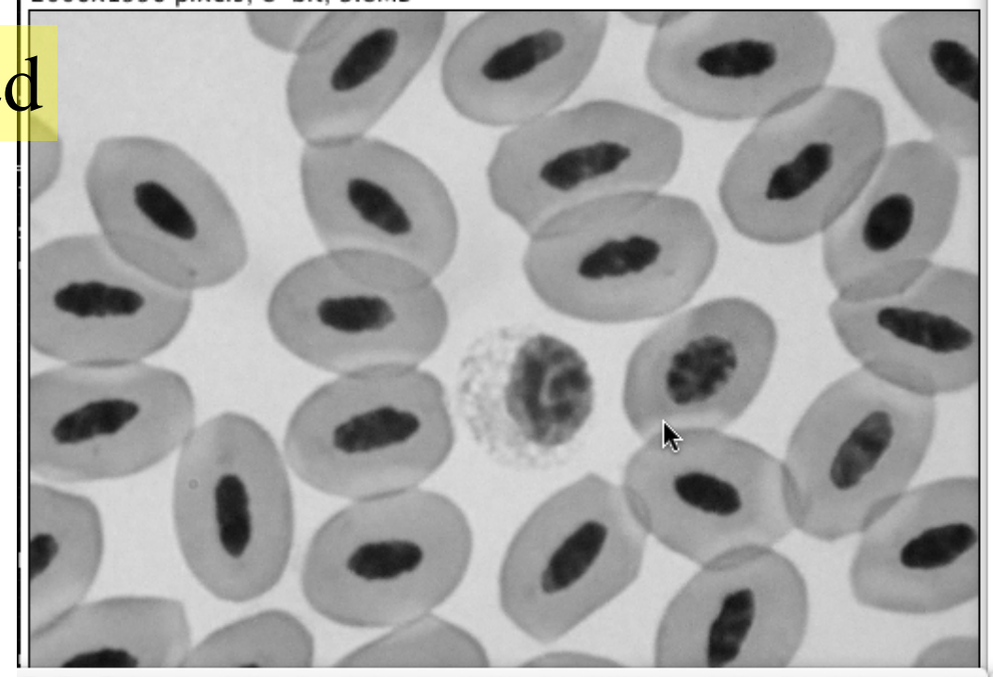
ImageJ: split colors



nonnetje1.jpg (red) (25%)

2006x1996 pixels; 8-bit; 3.8MB

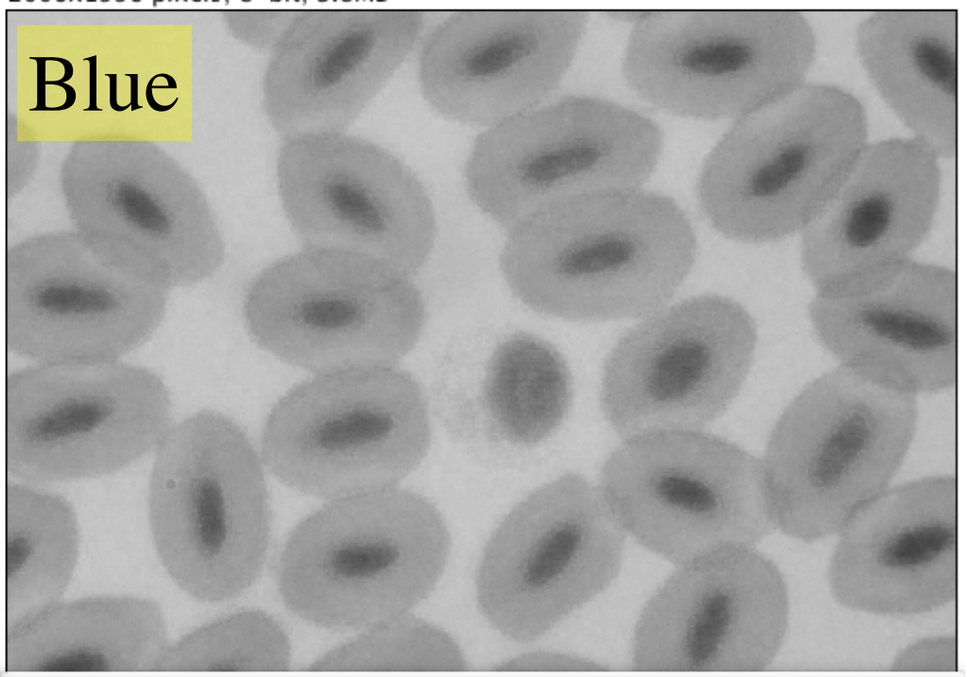
Red



nonnetje1.jpg (blue) (25%)

2006x1996 pixels; 8-bit; 3.8MB

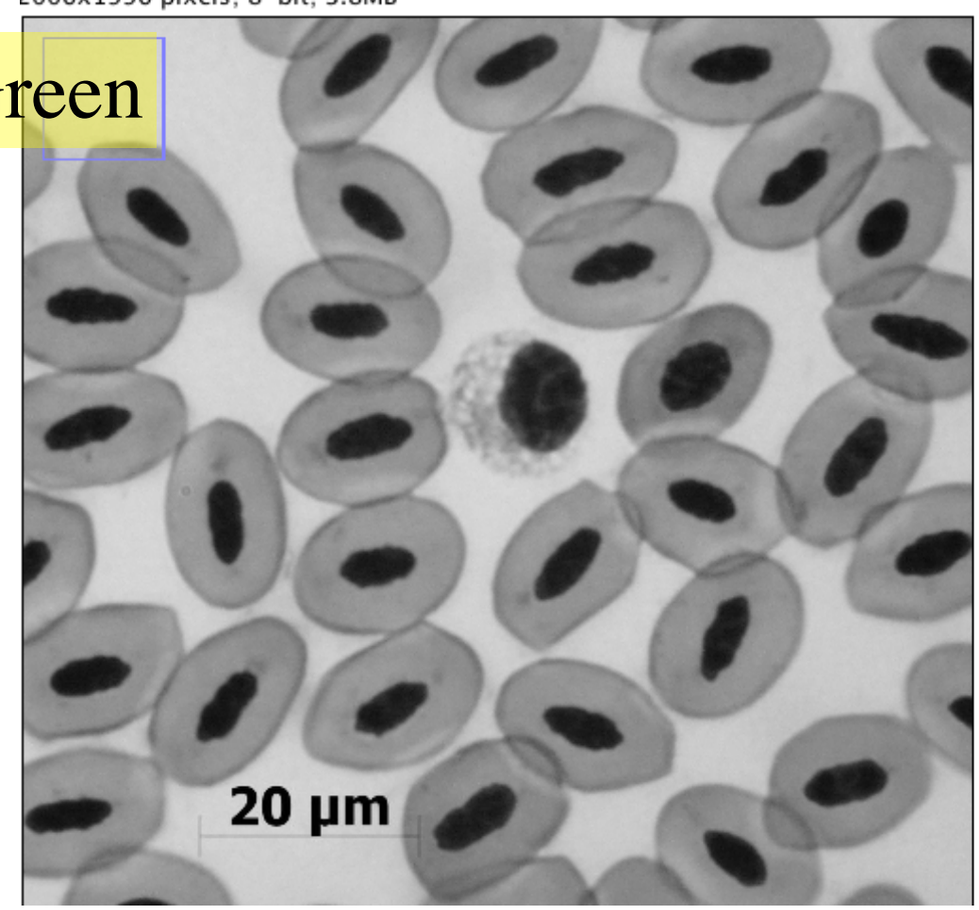
Blue



nonnetje1.jpg (green) (25%)

2006x1996 pixels; 8-bit; 3.8MB

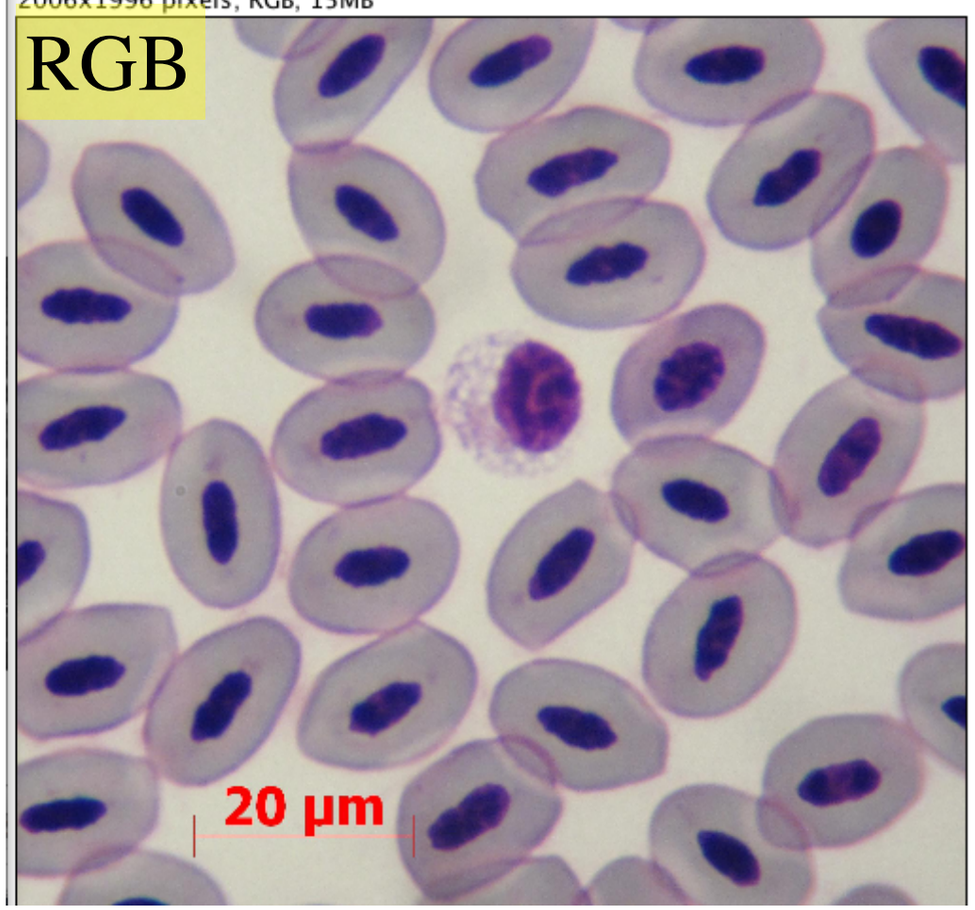
Green



nonnetje1.jpg (25%)

2006x1996 pixels; RGB; 15MB

RGB



“jpeg versus “text image”

- Jpeg stores data in an array of size [height,width,3]
  - [height,width,1] is the image formed from red component of color.
  - Same for [height,width,2] and [height,width,3] (green and blue components)
- Text Image stores a gray image (can be of two types):
  - gray = mean(red, green, blue)
  - gray = weighted average of red/green/blue (see next slide)

gray formula

- Saves the active image as a spreadsheet compatible tab-delimited text file. For calibrated images and floating-point images, the Decimal Places field in Analyze ◁ Set Measurements. . . determines the number of digits to the right of the decimal point. For RGB images, each pixel is converted to grayscale using the formula gray = (red + green + blue)/3 or the formula gray = 0.299 × red + 0.587 × green + 0.114 × blue if Weighted RGB to Grayscale Conversion is checked in Edit ◁ Options ◁ Conversions. . .

Image Processing

- Take an image
- Apply a series of transformations to the image
- Perform some analysis on the image
- Save the image to a hard drive

Image sizes

- Consider a 10 Mpixel camera
  - image resolution: 3000 x 3000 (or more)
- Each pixel has a R, G, B value
- Each color is stored in a single byte
- Size of image:
  - 3000 x 3000 * (byte*3) = 27,000,000 bytes
  - 27 Mbytes (fairly large)

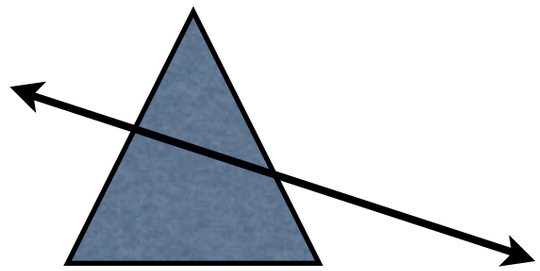
RGBA

- Sometimes, images have a 4th color channel:
  - opacity α:
  - $\alpha = 0$ (totally transparent)
    - image cannot be seen
  - $\alpha = 255$ (fully opaque)
    - image is transparent (invisible)

File Formats

- Main formats I use:
  - PNG (can store α channel)
    - compressed with no loss of information
    - best for images with sharply defined edges
    - Only works for images in RGB or RGBA formats
  - JPEG (lossy compression)
    - best suited for images with smooth color variations
  - GIF (similar to PNG, but copyrighted)

Good images for png



Triangle + arrow

The images have well-defined edges. PNG will lead to better quality images, that take less space on disk. But images are still “pixelated”

PDF images are even higher quality. No pixelation as one zooms.

Good images for jpeg

Not many sharp edges

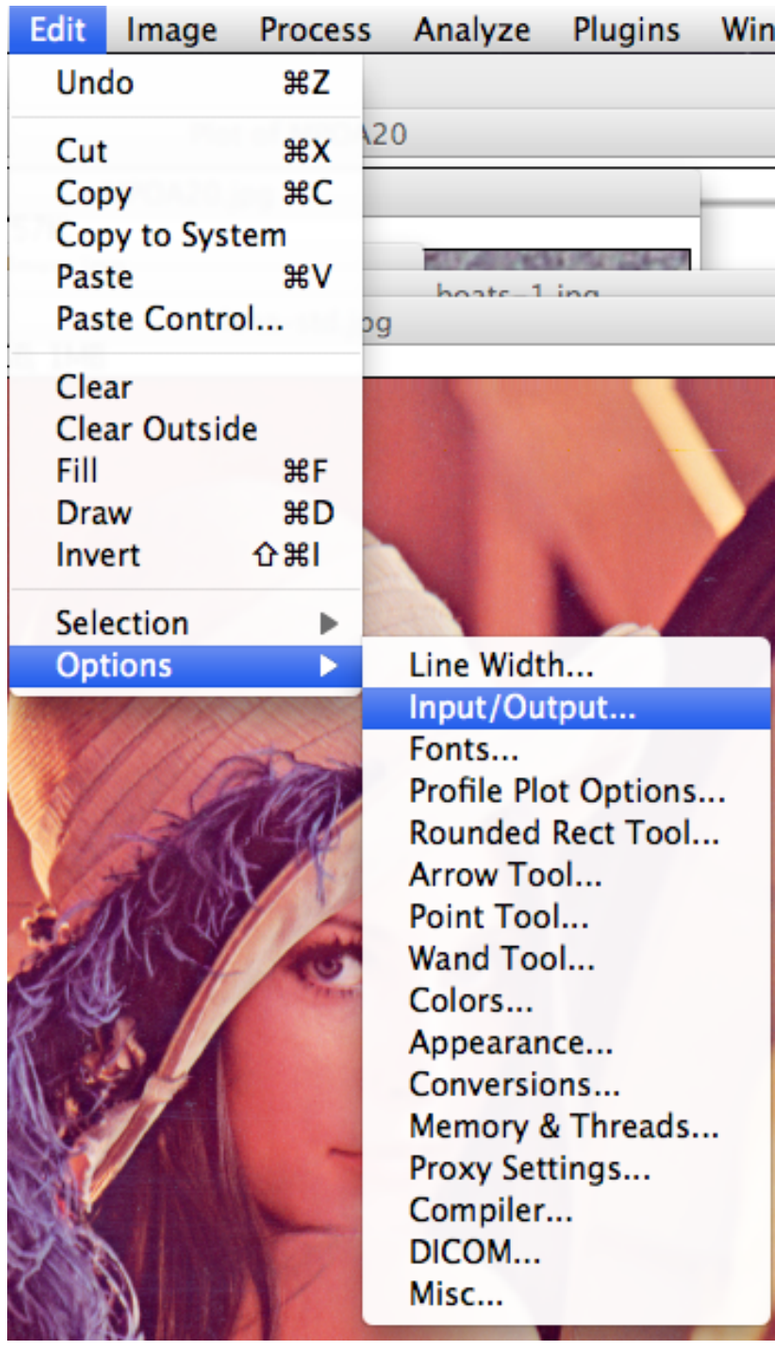
Example Image

- ducklings.tif
  - 3367 x 2223 pixels, RBG
  - 24,455,223 bytes
  - file size on the computer: 24459,590 bytes
- Read this file in ImageJ and output it in jpeg and PNG formats

ducklings.tif : 24459590 bytes
3367*2223*3 = 23464623 bytes

Low quality jpeg

- Go to ImageJ
- Find Options/Input.Output menu
- Change jpeg quality from 85% to 10%
- Save image to lena-10%.jpg



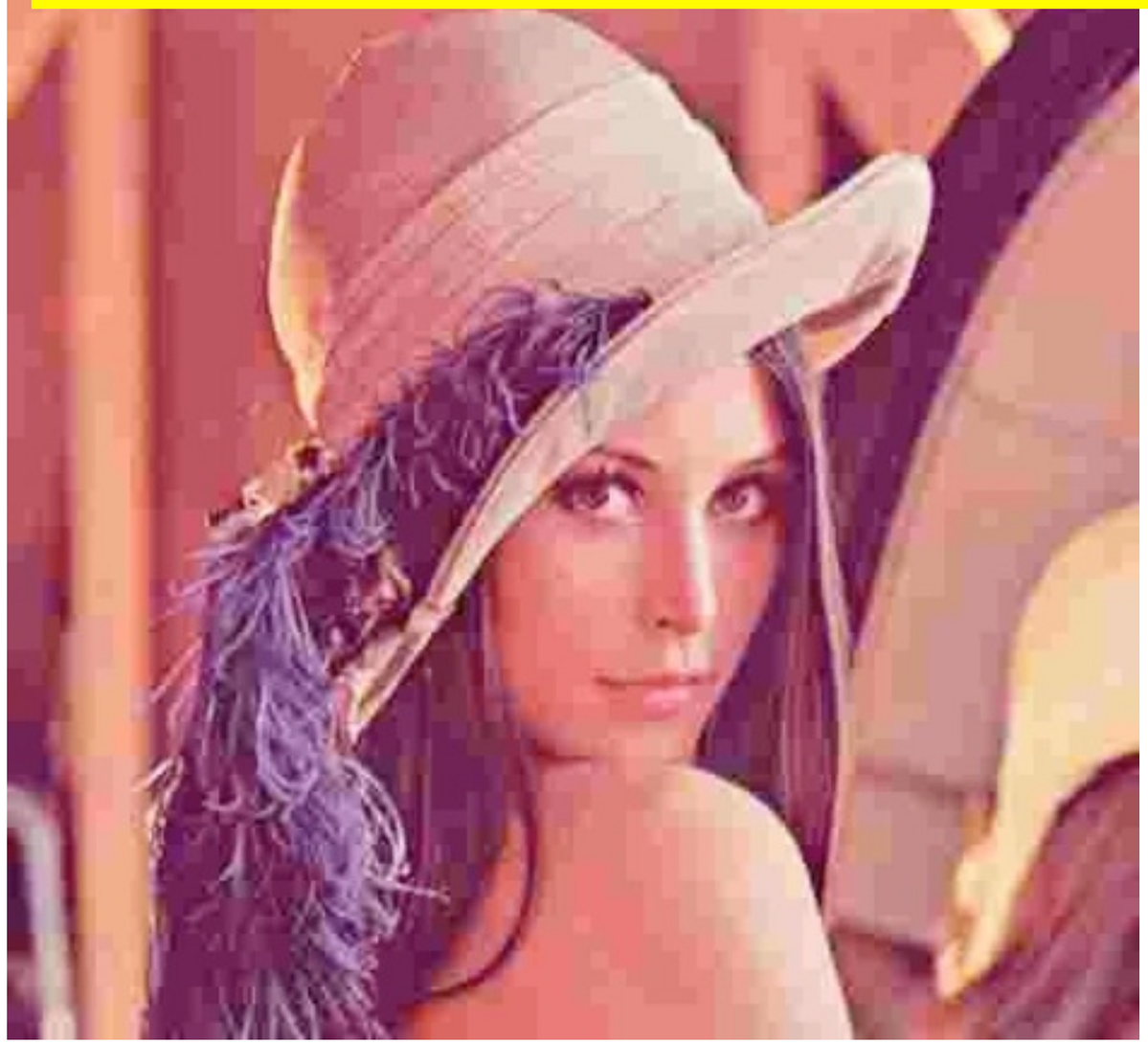
Use Lena image from available ImageJ image samples

Quality: 10%

File size: 2028267 bytes

Quality: 10% of 10%

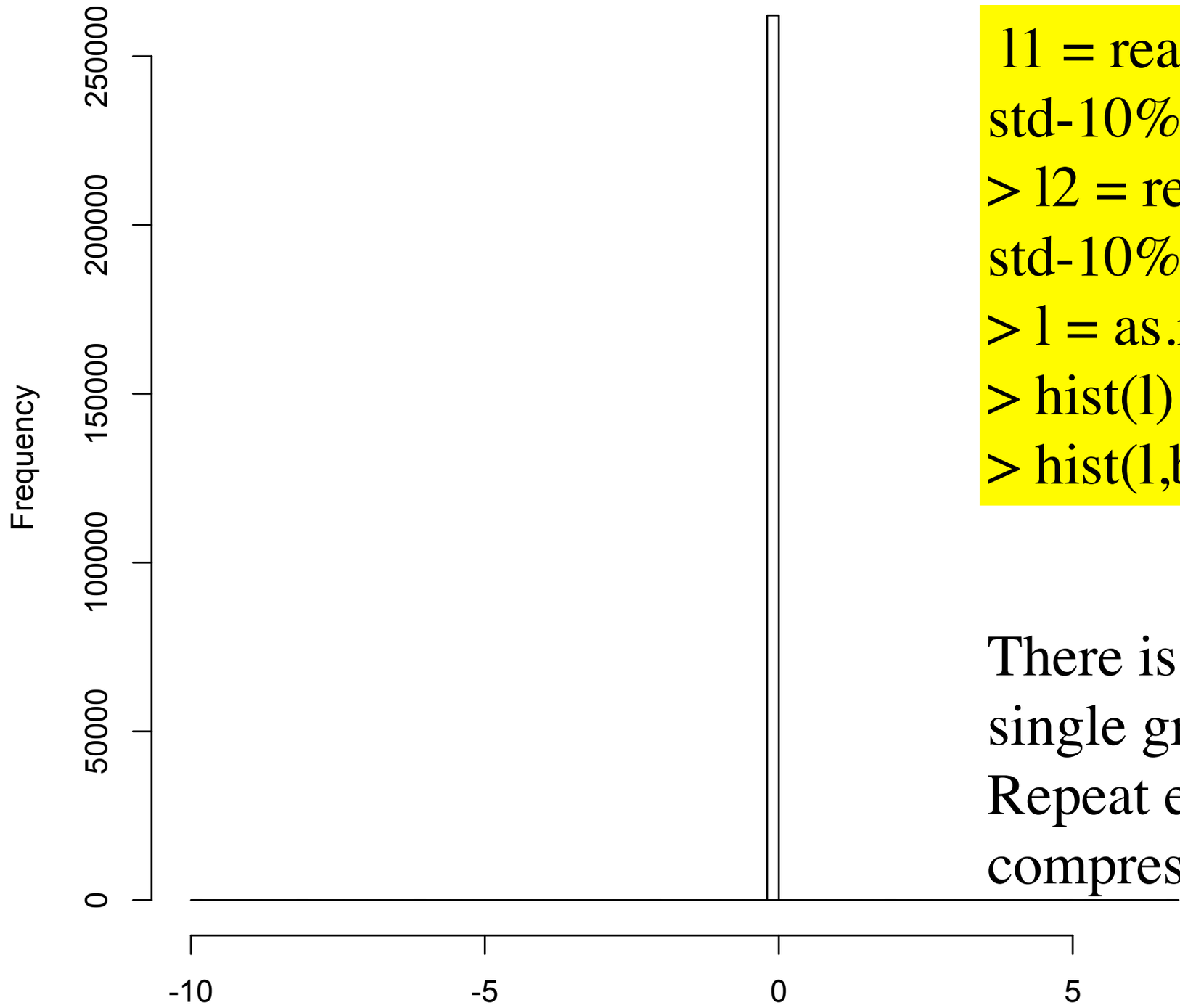
Read left picture into ImageJ and output again at jpeg 10% quality



Experiment

- Read the two files back into ImageJ
- Output both files in Text-Image… format
- Read these two text files into R
- Subtract the two files
- Plot the histogram of the difference
- Compute the mean and std of the difference
- Execute ??jpeg (find readJPEG in jpeg package)
- Use: library(jpeg) to access new functions
- Use readJPEG to read the file directly (no need for read.table)

Histogram, 100 bins




```
 l1 = read.table("lena-std-10%.txt")
> l2 = read.table("lena-std-10%10%.txt")
> l = as.matrix(l1-l2)
> hist(l)
> hist(l,breaks=100)
```

There is a difference in only a single gray value. Strange. Repeat experiment with jpeg compression at 1%

Quality: 1%

Quality: 1% of image on left after reloading into R

File size: 5730 bytes

Testing with R shows that the images are identical. But quality is seriously degraded with respect to original image.

```
library(jpeg)
hist2 = function(img1, img2) {
    l1 = readJPEG(img1)   # make sure JPEG package is installed
    l2 = readJPEG(img2)
    hist(l1-l2, breaks=100)
}
```

```
par(mfrow=c(2,2))
hist2("MPOA20-1%.jpg", "MPOA20-1%-1%.jpg")
hist2("lena-std-1%-1%.jpg", "lena-std-1%.jpg")
```

I created a function that plots the histogram
of the difference between two image files.
In both cases, I find that the files are identical.

```
> source("diff_images.r")
max(l1-l2)=  0
min(l1-l2)=  0
```

 # Proves that l1 equals l2

```
> source("diff_images.r",echo=T)

> library(jpeg)

> # img1 seems to be nxmx3 and img2 is nxmx1.
> # check this in properties in ImageJ.
>
> hist2 = function(img1, img2) {
+     l1 = readJPEG(img1)
+      .... [TRUNCATED]

> par(mfrow=c(2,2))

> hist2("MPOA20-1%.jpg", "MPOA20-1%-1%.jpg")
max(l1-l2)=  0
min(l1-l2)=  0

> hist2("lena-std-1%-1%.jpg", "lena-std-1%.jpg")
max(l1-l2)=  0
min(l1-l2)=  0
```

```
library(jpeg)

# img1 seems to be nxmx3 and img2 is nxmx1.
# check this in properties in ImageJ.

hist2 = function(img1, img2) {
        l1 = readJPEG(img1)
        l2 = readJPEG(img2)
        hist(l1-l2, breaks=100)
        cat("max(l1-l2)= ", max(l1-l2), "\n")
        cat("min(l1-l2)= ", min(l1-l2), "\n")
}

par(mfrow=c(2,2))
hist2("MPOA20-1%.jpg", "MPOA20-1%-1%.jpg")
hist2("lena-std-1%-1%.jpg", "lena-std-1%.jpg")
```

Formats

ducklings.png: 20,719,730 bytes

png: compressed tif file so reduced size.

No loss of information

ducklings.jpg: 1,285,937 bytes

Very small file

highly compressed

Loss of information

Initial file cannot be reconstructed

On images with sharp edges, and well-define formats, png files can sometimes be smaller than jpg files.

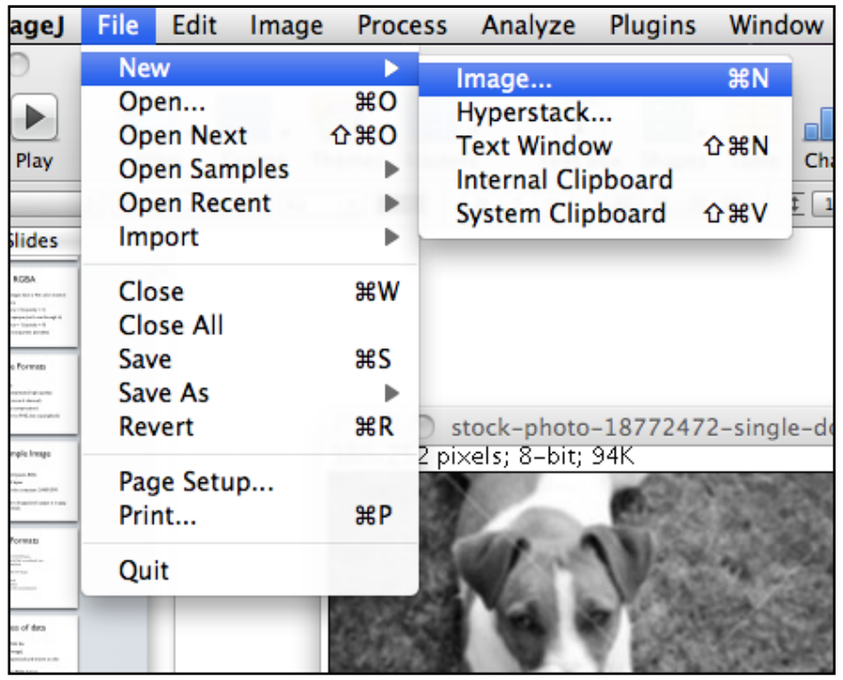
Image Processing II

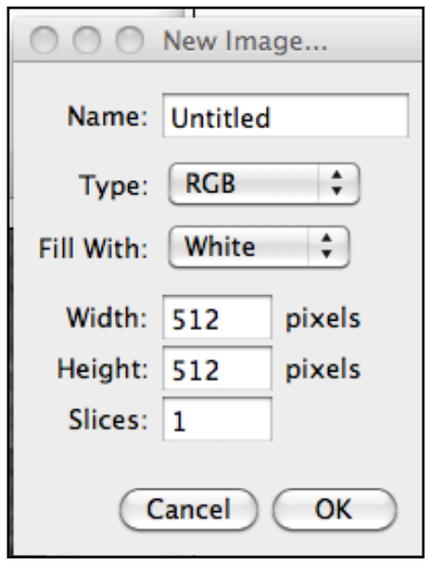
Regions

- Any rectangular region of size w x h can be viewed as an image
- ImageJ allows arbitrary selections from an image (freeform, or via dimensions)

Regions

- Use left mouse to select a region
- copy the region (cmd-C or ctrl-C)
- create a new plot
  - specify plot size
  - paste the region





Freeform regions

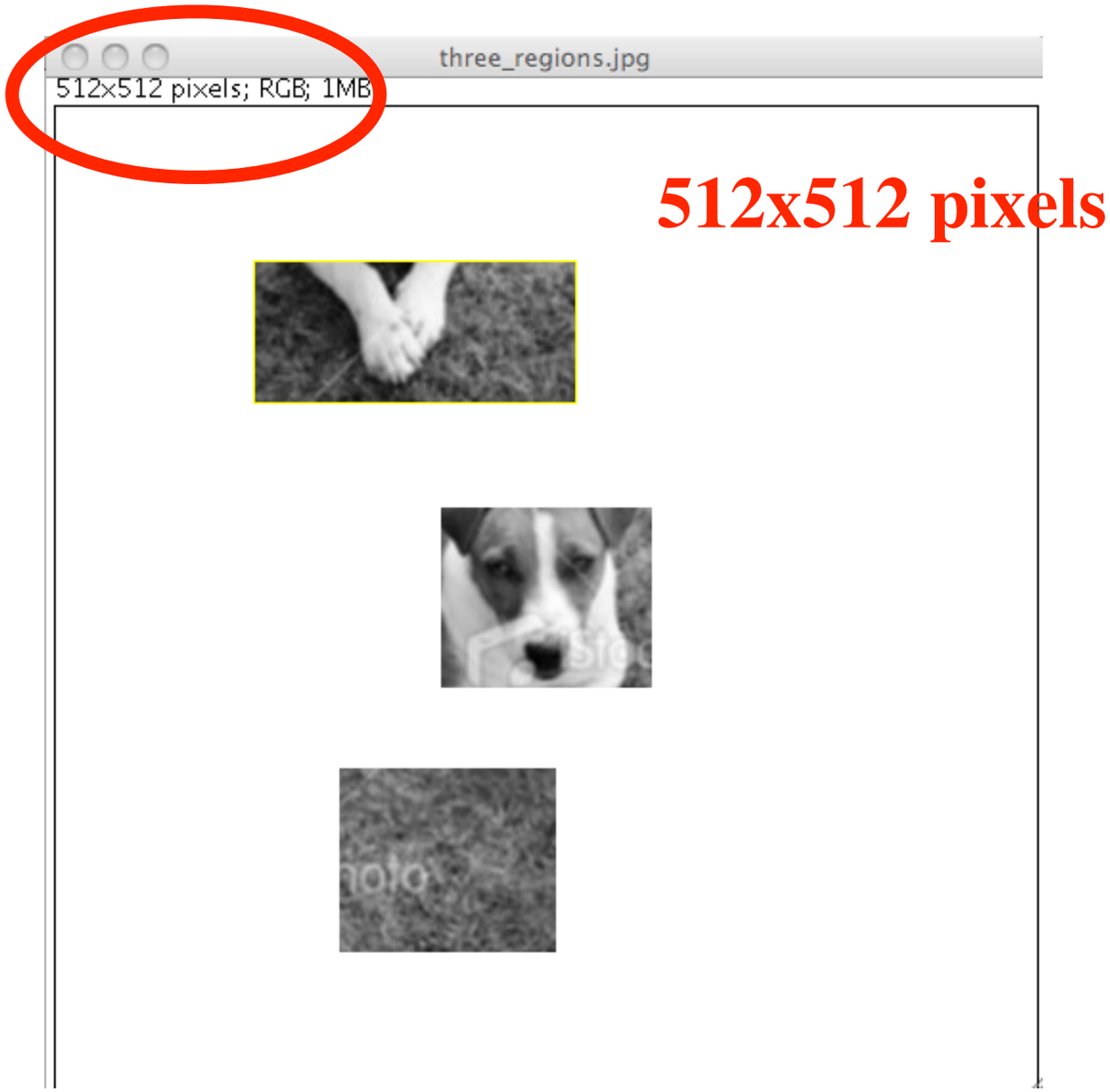


Image Analysis

We define the term image analysis in the following way:

Image analysis is the extraction of meaningful information from digital images by means of digital image processing techniques.

What information

- Edges in image
- Extract cell structures
- Regions (for segmentation)

Tools for extraction

- Smoothing filters
- Enhancing filters
- Extrusion
- Histogram manipulation
- Color, brightness, saturation manipulation
- ...

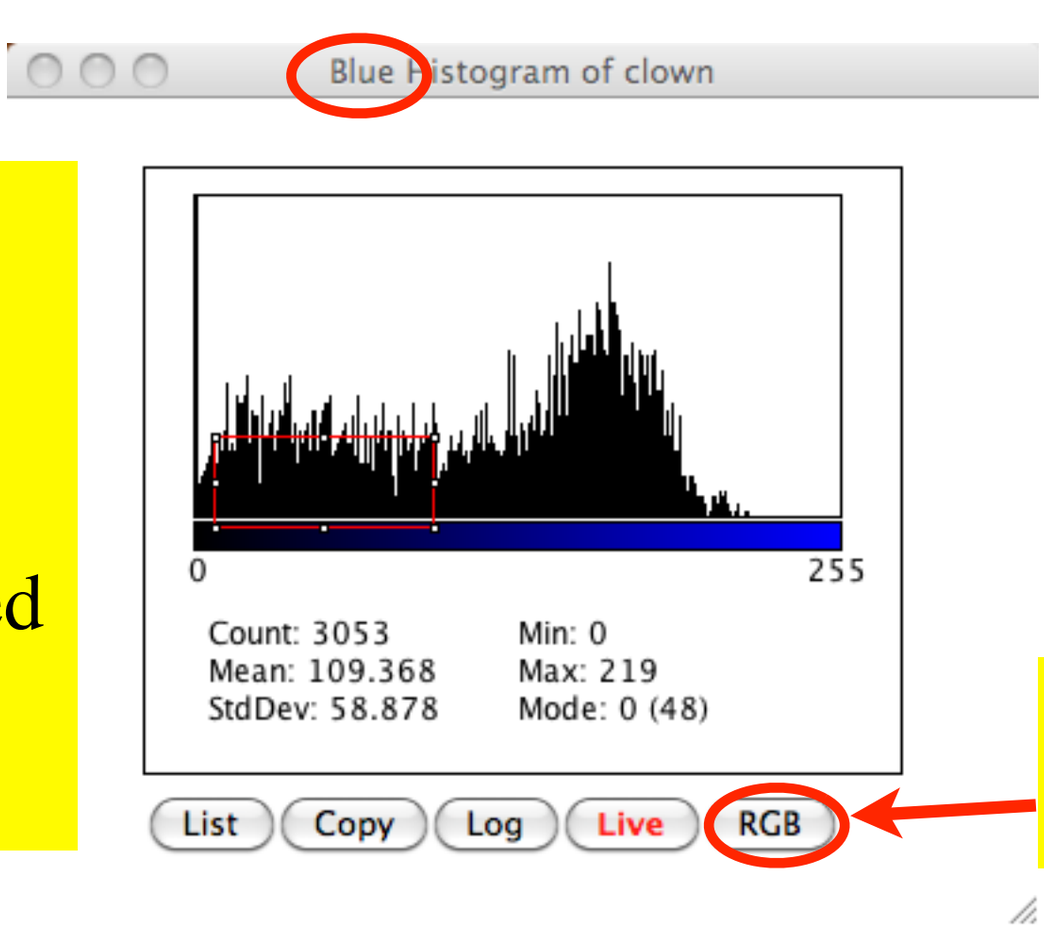
Tools in ImageJ

- Image annotation
  - add arrows, text, etc.

Color histogram

Start with color image

Menu: Analyze--> histogram

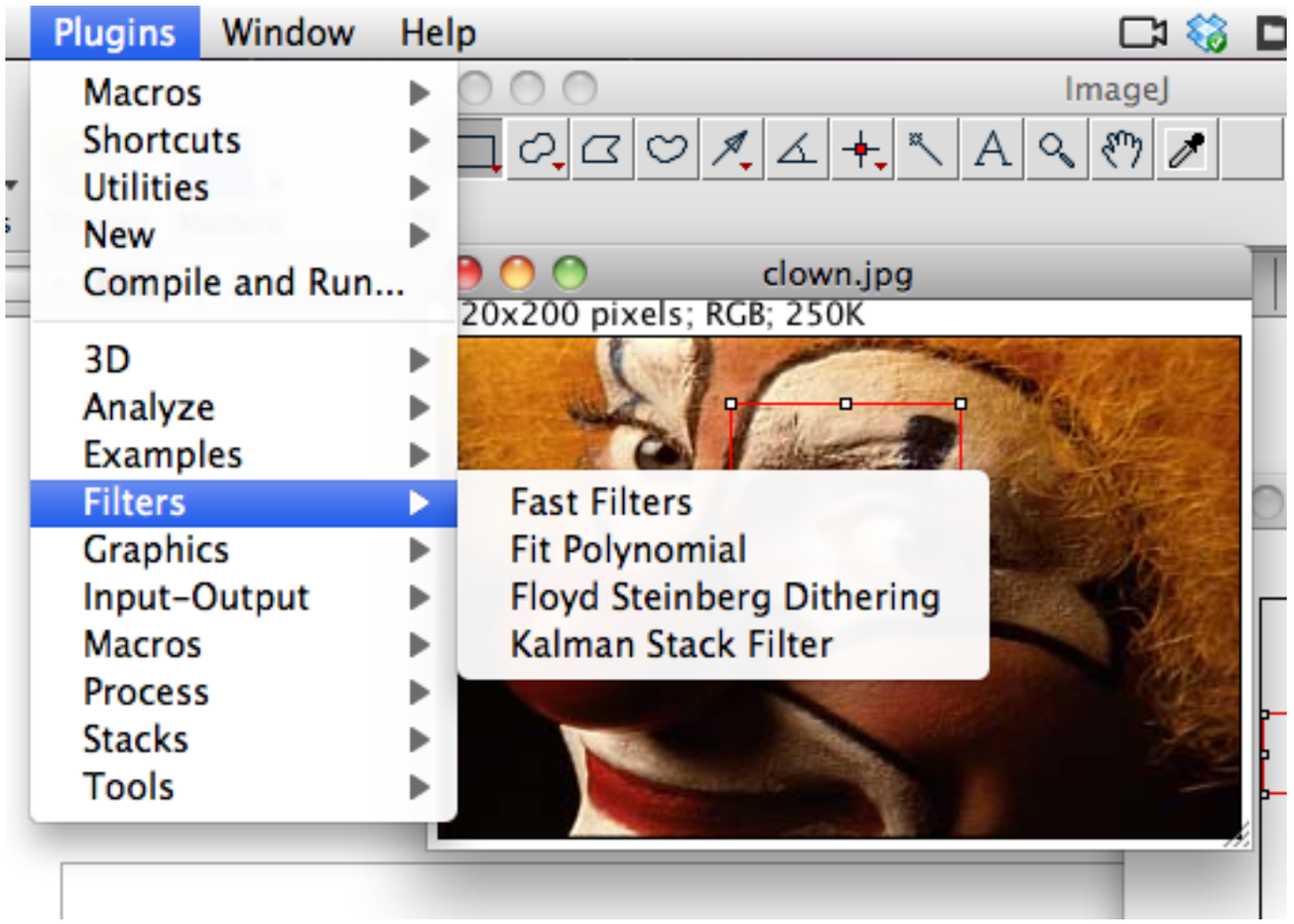


Live:
Histogram
updates in
real time
as the selected
region is
modified

Blue
Histogram

Cycle through Red,
Green, Blue histograms

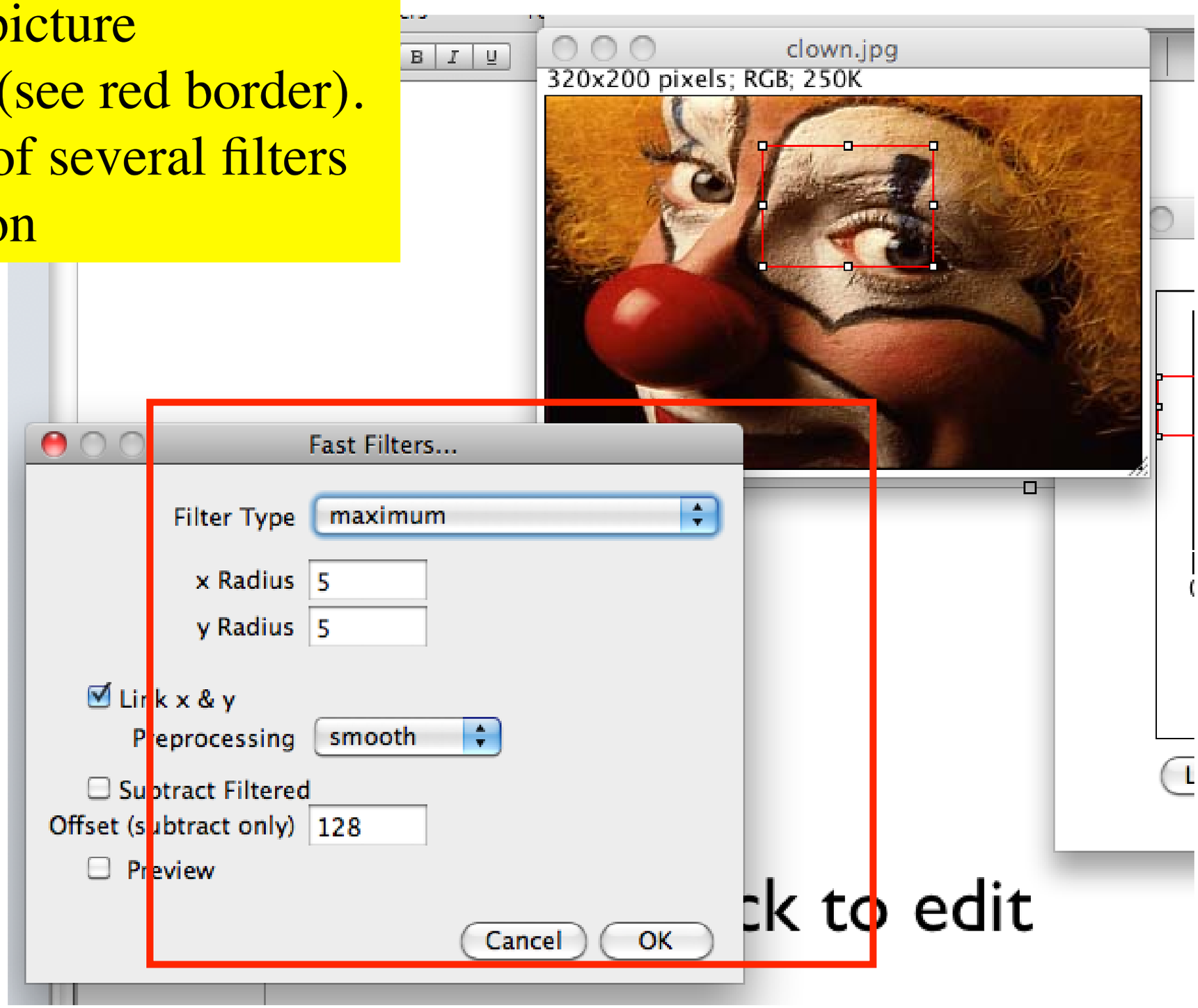
Filters



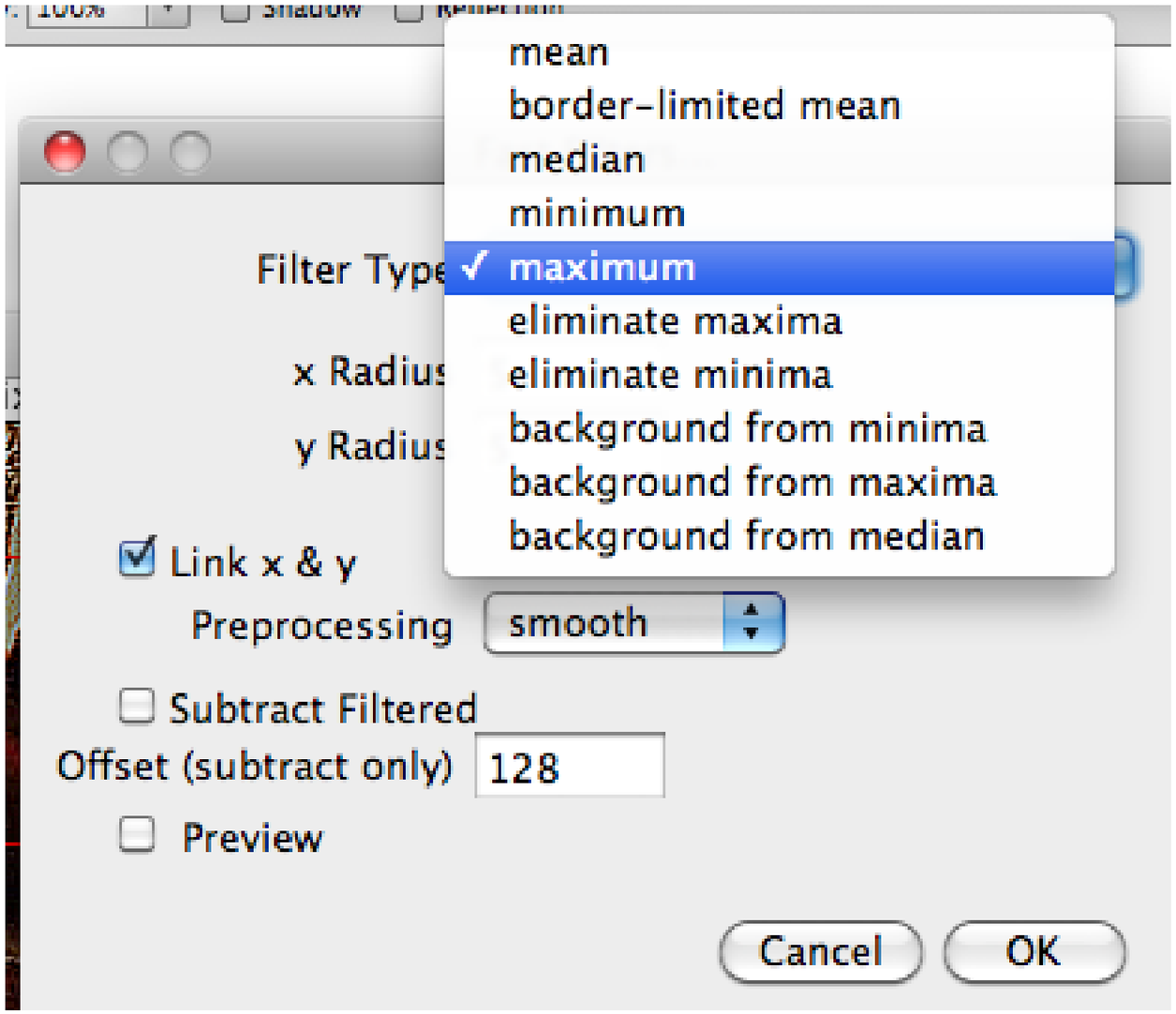
use of fast filters
See next slide

Analysis/Fast Filters

Select full picture
or a region (see red border).
Apply one of several filters
to that region



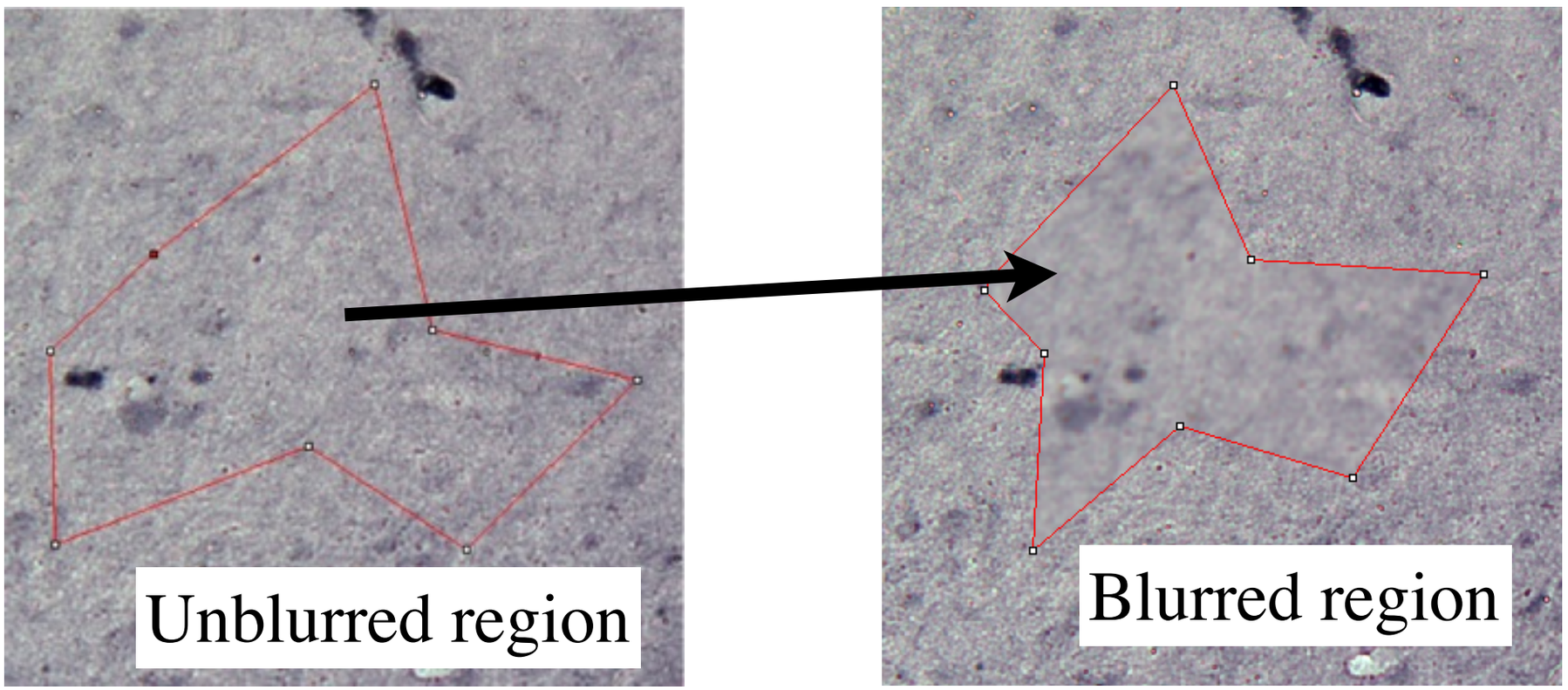
Fast Filter List



What is a Filter?

Filter is some operator that takes a region as input, and transforms its interior.

(in the image below, the polygons are different. Ignore this issue)

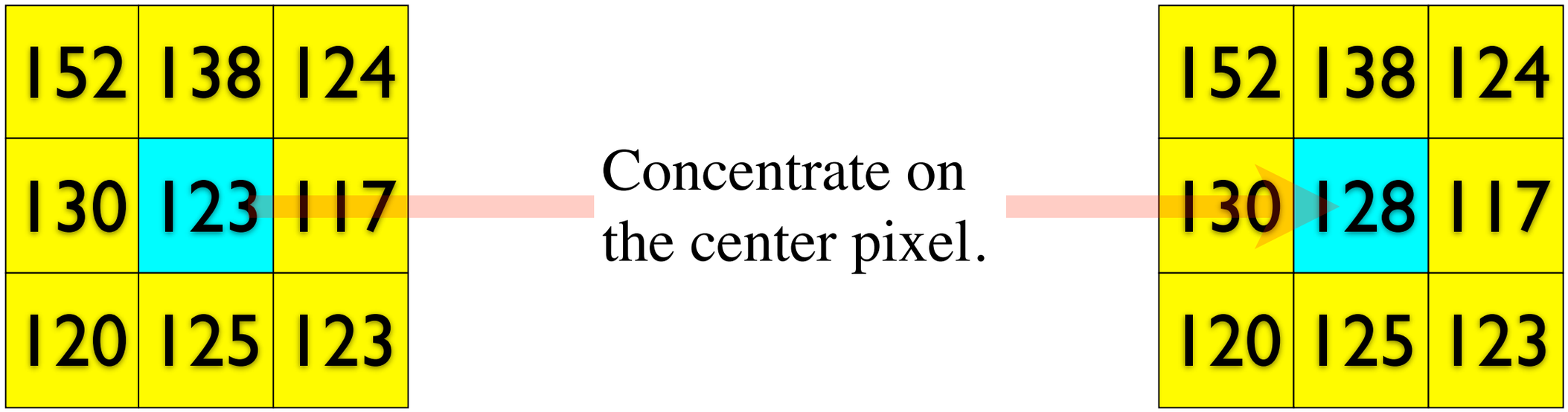


Filter as a matrix

- A filter is stored inside a computer as a matrix of dimensions 3x3 or 5x5 of numbers
- Example on next slide, using a 3x3 matrix

How do filters work?

Assume an original gray image. Each pixel is represented as an integer in the range 0 to 255.



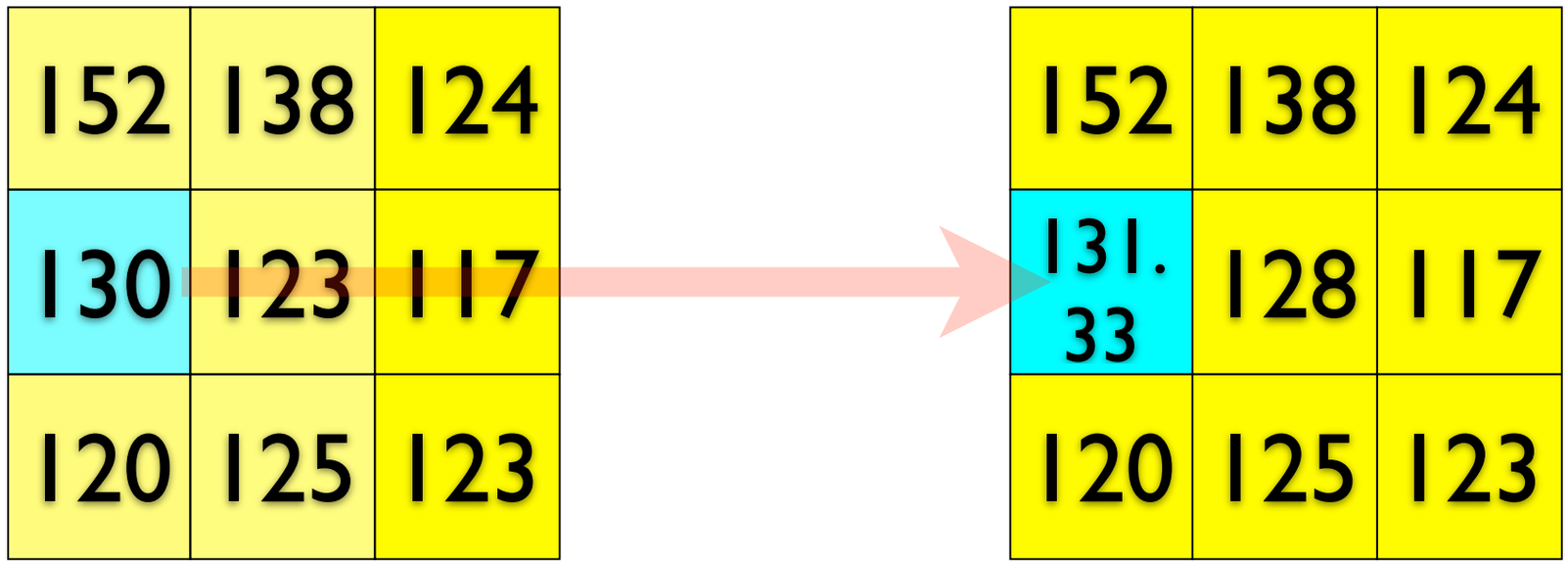
Calculate the average of all pixel gray values:

mean = (1/9)*(152+138+124+130+123+117+120+125+123)

= 128

Repeat this operation for every pixel in the image (with special treatment for the edge pixels)

Edge Pixels



mean = (1/6)*(152+138+130+123+120+125)
= 131.333

More generally

Combine all 9 interior pixels with a formula such as

$$\text{filter} = w_1*152+w_2*138+w_3*124+w_4*130+w_5*123+ w_6*117+w_7*120+w_8*125+w_9*123)$$

with the condition that

$$\text{sum}(c(w_1, w_2, w_3, w_4, w_5, w_6, w_7, w_8, w_9)) = 1$$

In the previous examples, all w's equal (1/9) so the sum is indeed 1.

Some Filters

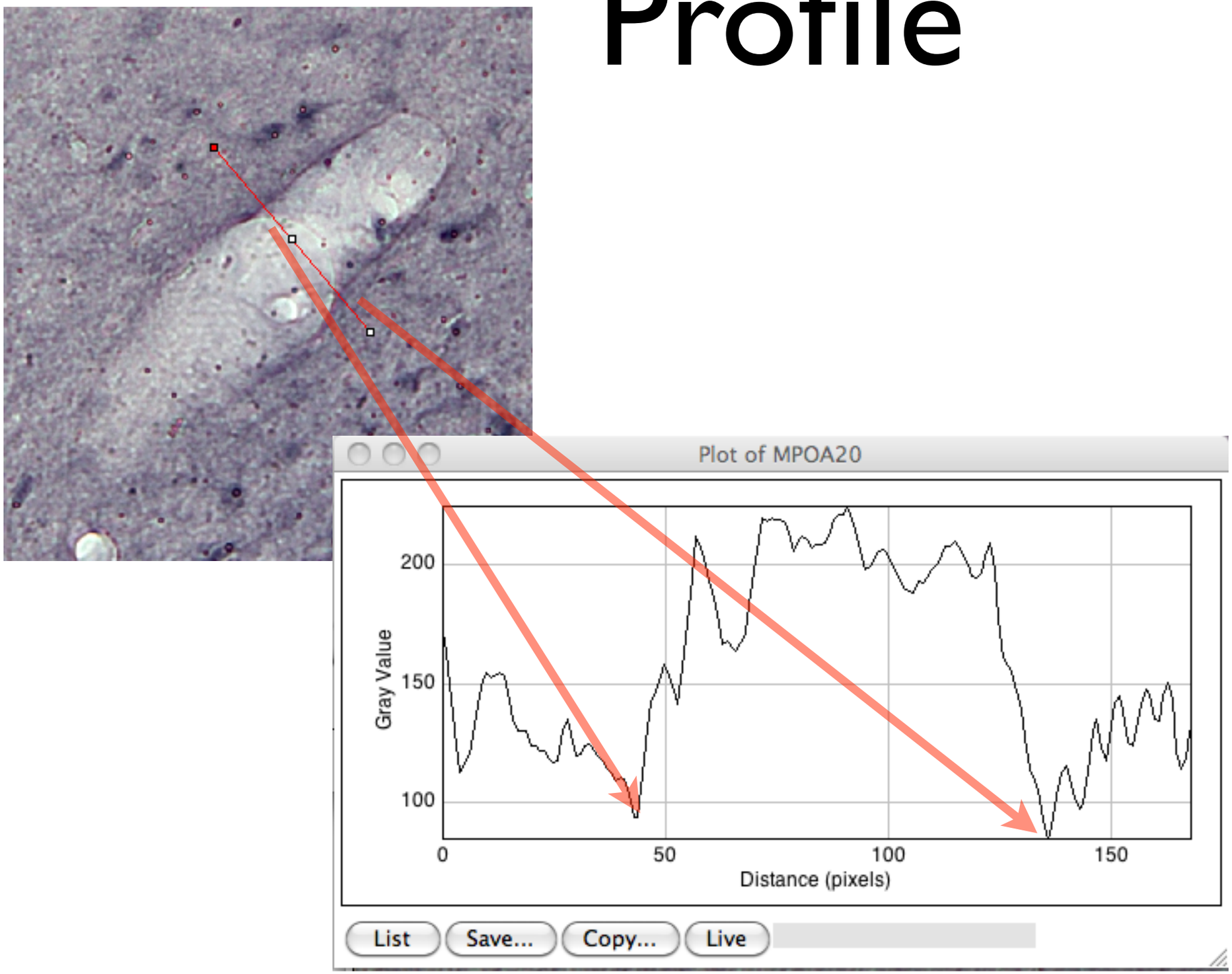
Gaussian Blur

- Used to smooth out an image
- A larger radius implies stronger smoothing

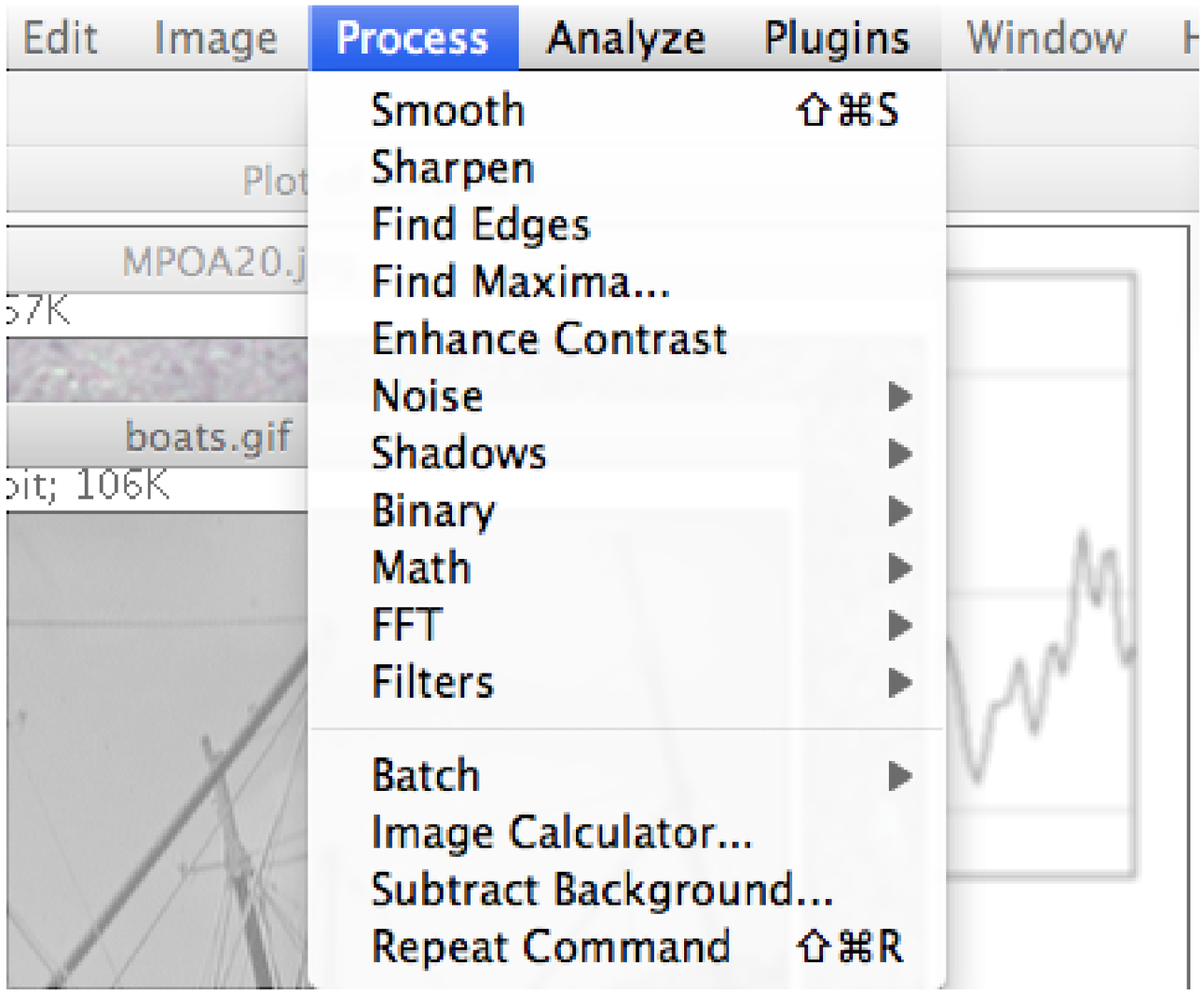
Edge Detection

- Enhance edges in an image

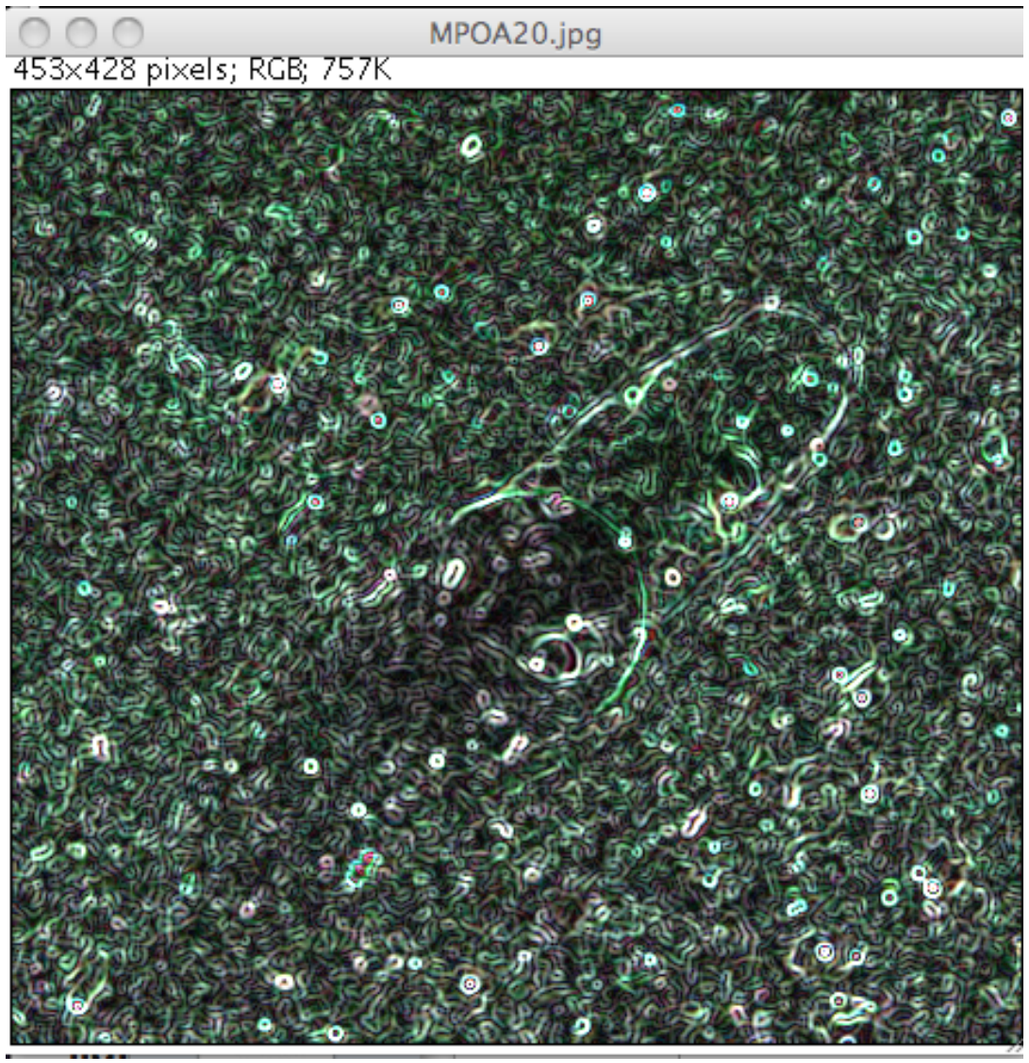
Profile


Plot of MPOA20
Gray Value
200
150
100
0
50
100
150
Distance (pixels)
List
Save...
Copy...
Live

Edge Detection

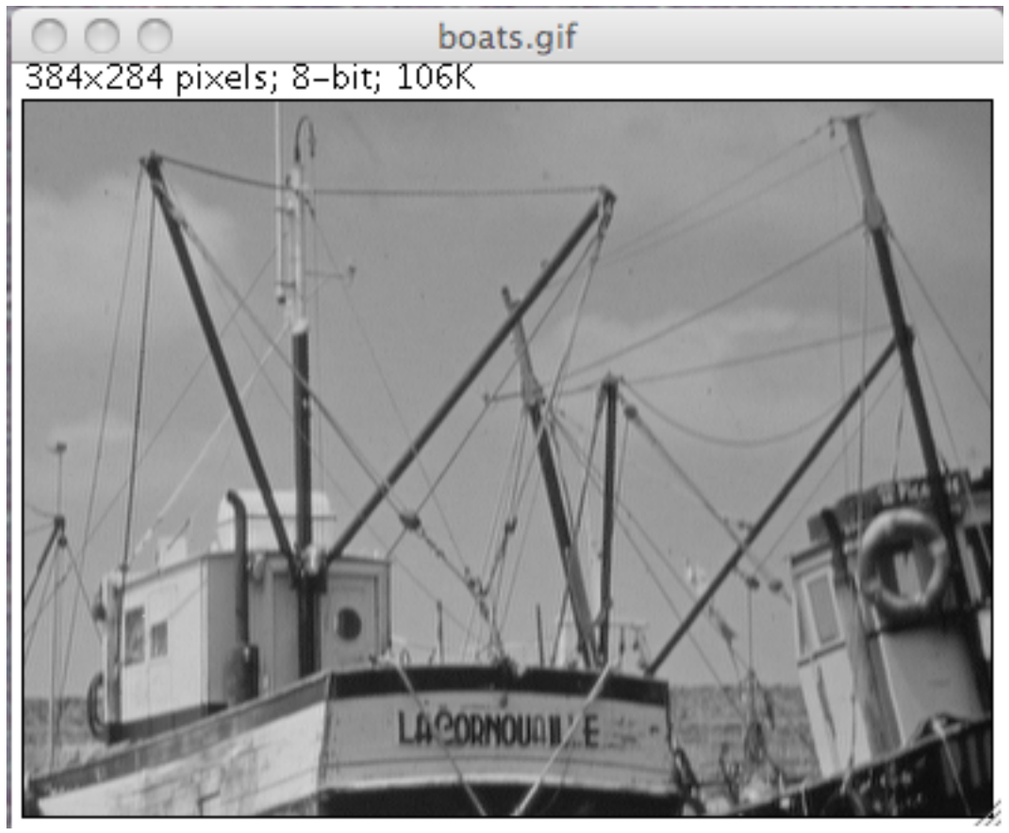


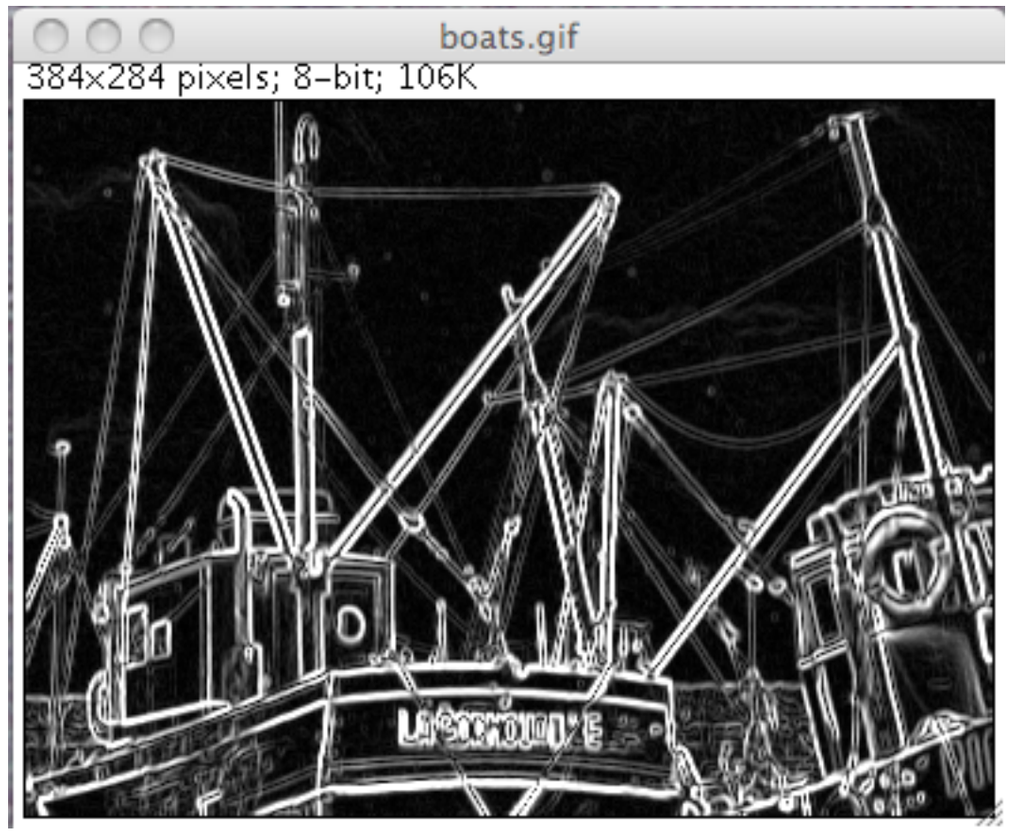
Edge Detection



Messy results. Original image was too noisy.

Edge Detection on smoother image





Color maps

- Color Lookup Tables
- Until know, each pixel had an RGB triplet
- Now consider that one stores in each pixel, a number between 0 and 255
  - construct a lookup table (LUT) that maps an integer between 0 and 255 into a color triplet

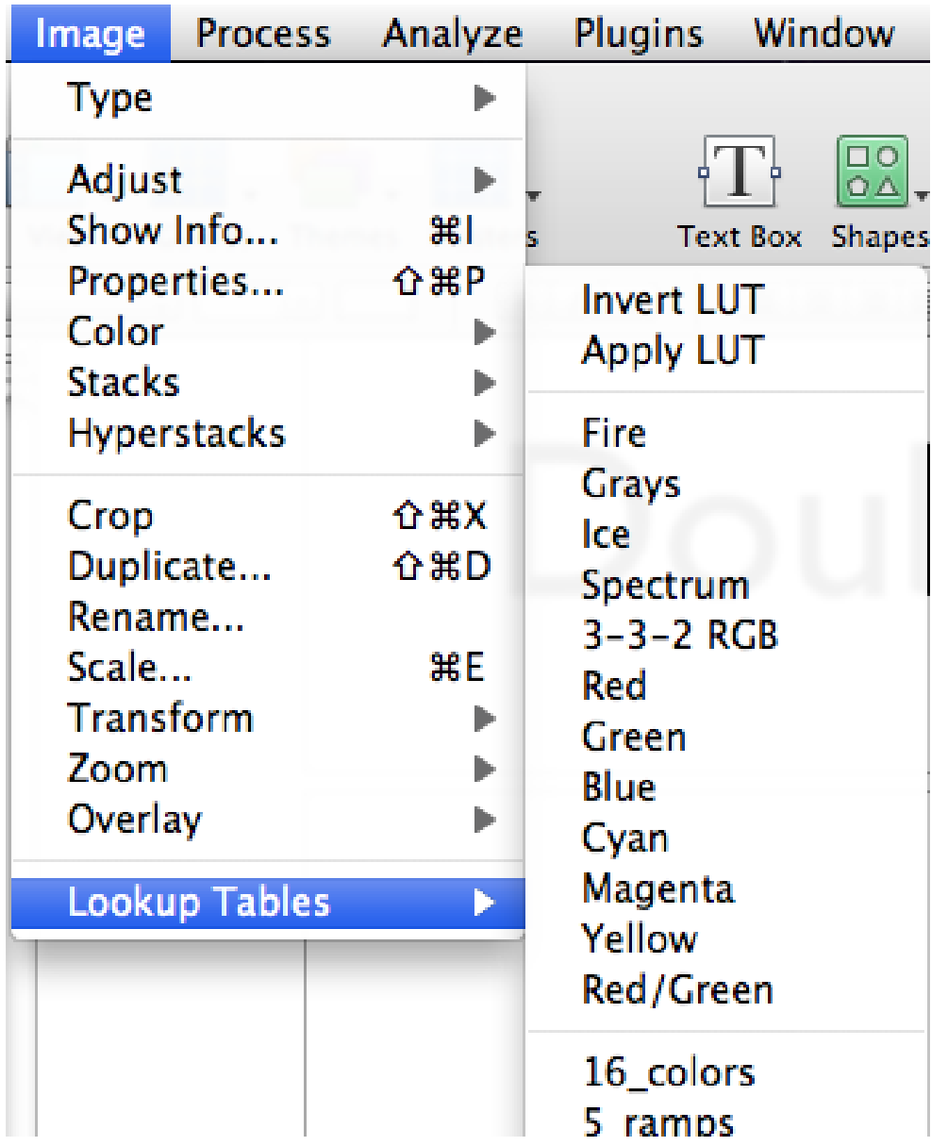


Image
Process
Analyze
Plugins
Window
Type
Adjust
Show Info... ⌘I
Properties... ⇧⌘P
Color
Stacks
Hyperstacks
Crop ⇧⌘X
Duplicate... ⇧⌘D
Rename...
Scale... ⌘E
Transform
Zoom
Overlay
Lookup Tables
Text Box
Shapes
Invert LUT
Apply LUT
Fire
Grays
Ice
Spectrum
3-3-2 RGB
Red
Green
Blue
Cyan
Magenta
Yellow
Red/Green
16_colors
5 ramps

… and more

Examples of Lookup tables

- Convert an RGB image to a color index image

Color Tables (LUT)

- manual, p. 25

Selections

- Ellipse, rectangle, polygon, etc. (p. 17, 18)
- Annotations (arrows, etc.)
  - appear in overlay (non-destructive)
  - “d” to store annotation into the image
- measurements (“m”)
  - accumulate measurements into a table

brightness/contrast

- p. 65 manual
- image->adjust (menu)

Color index converstion

- p. 64 in manual
- Start with RGB image
- image->type->8-bit-color
- play with LUTs

Window level

- Image->adjust->window level
- Color threshold
- show-info

color

- Show LUT
- color picker

stacks

- image->stacks->montage
- manual, p. 77
- reslice (orthogonal cuts)

overlays

- Experiment non-destructively
- manual, p. 89
- ROI manager

Process menu

- Many tools to process an image
- bandpass filters (using FFT)

batch

- Useful for lab

- – Files can be easily overwritten since the batch processor will silently override existing files with the same name.

– The destination Output folder should have adequate disk space to receive the created images.
– In the case of non-native formats, batch operations will be influenced by the behavior of the reader plugin or library (cf. Non–native Formats).

p. 112 in manual

Removing Background

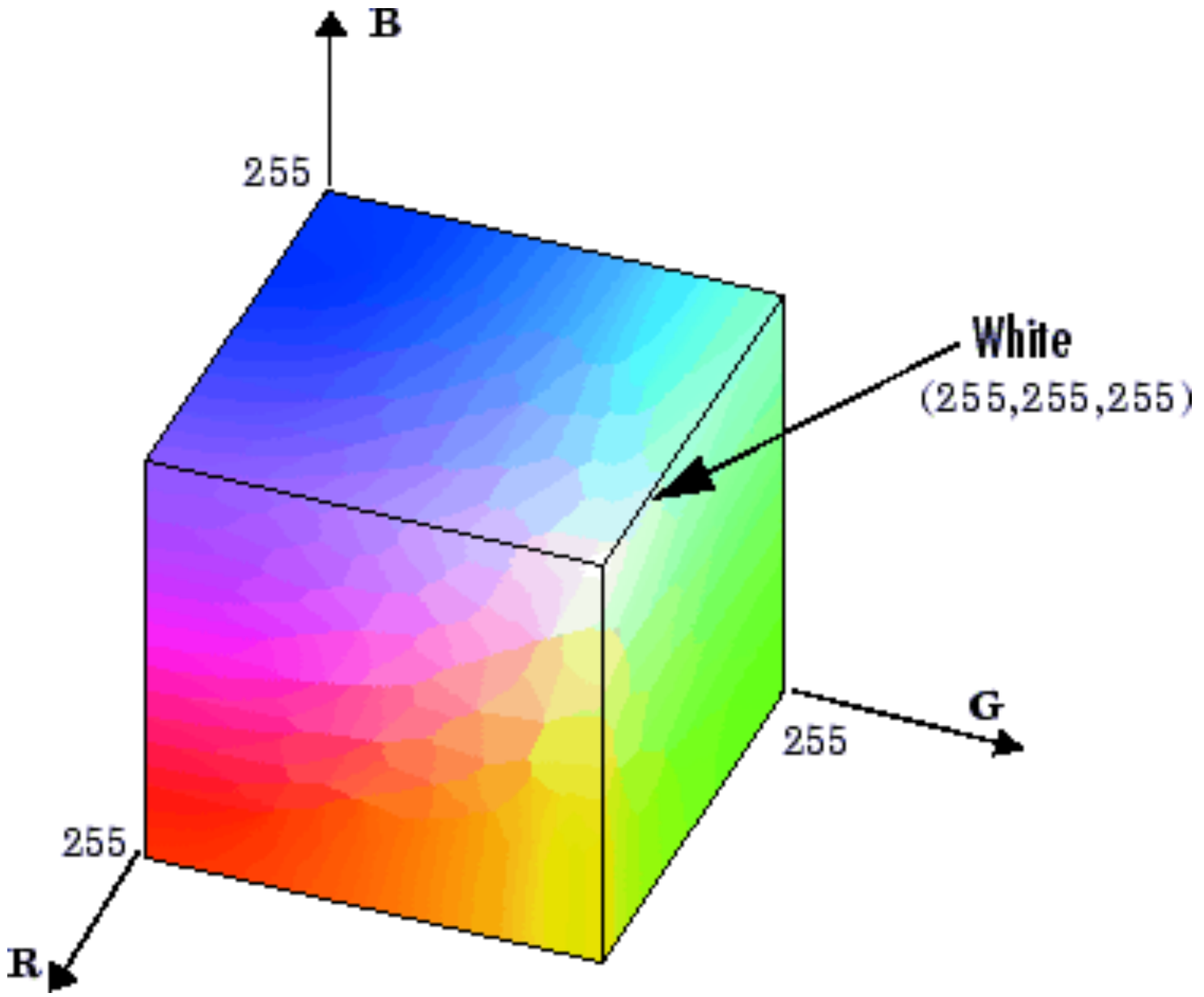
- Ultra-slick
- If background has slight shadows for varying color, it can be removed, which will enhance contrast and possibly simplify counting.
- Process->subtract background

Color characteristics

- Brightness
- Hue
- Saturation
- Red, Green, Blue Levels
- Transparency (Alpha) (used for PNG files)

Color models

Color Cube



Hue-Value-Saturation cone