ARITHMETIC AND LOGIC OPERATIONS
FUNDAMENTALS AND APPLICATIONS

• Arithmetic operations in images perform on a pixel-by-pixel basis.
• Given a 2D array, X, and Y,
• Z obtains by calculating:
  \[ Z = X \oplus Y \]
• Where \( \oplus \) is a binary arithmetic (\(+, -, \times, /\)) operator.
Addition Operator

- To blend the pixel contents from two images

\[ X + Y = Z \]
Addition Operator

• Or to add a constant value to pixel values of an image.

X + 50 = Z
Noise Addition Operator

- Adding random amounts to each pixel value is a common way to simulate additive noise.

\[ X + \text{noise}(0, 0.01) = Z \]
Adding two images must be careful with overflow values.

Two ways to deal with the overflow issue:

- normalization

\[ g = L_{max} \left( \frac{Z - Z_{min}}{Z_{max} - Z_{min}} \right) \]

- truncation.
Addition Operator

Example:

\[
X = \begin{bmatrix}
200 & 100 & 100 \\
0 & 10 & 50 \\
50 & 250 & 120
\end{bmatrix}
\]

\[
Y = \begin{bmatrix}
100 & 220 & 230 \\
45 & 95 & 120 \\
205 & 100 & 0
\end{bmatrix}
\]

\[
W = \text{uint16}(X) + \text{uint16}(Y);
\]

\[
Z_a = 255 \times (W-45)/(350-45);
\]

\[
Z_b = X + Y; \ %\text{imadd}(X,Y);
\]

\[
Z_a = \begin{bmatrix}
213 & 230 & 238 \\
0 & 50 & 105 \\
175 & 255 & 63
\end{bmatrix}
\]

\[
Z_b = \begin{bmatrix}
255 & 255 & 255 \\
45 & 105 & 170 \\
255 & 255 & 120
\end{bmatrix}
\]
Subtraction Operator

• Used to detect differences between two images.

• Such differences may be due to several factors
  – Such as artificial addition to or removal of relevant contents from the image (e.g., using an image manipulation program)
  – relative object motion between two frames of a video sequence, and many others.
Subtraction Operator

- Subtracting a constant value from an image causes a decrease in its overall brightness, a process sometimes referred to as subtractive image offset.

\[ X - 50 = Z \]
Subtraction Operator

Image subtraction can also be used to obtain the negative of an image.

\[ Z = L_{max} - X \]
Subtraction Operator

• Subtracting one image from another or a constant from an image, you must be careful with underflow.

• There are two ways of dealing with this underflow issue:
  – **absolute difference** (which will always result in positive values proportional to the difference between the two original images without indicating, however, which pixel was brighter or darker) and
  – **truncating** the result, so that negative intermediate values become zero.
Subtraction Operator

Example:

\[
X = \begin{bmatrix}
200 & 100 & 100 \\
0 & 10 & 50 \\
50 & 250 & 120 \\
\end{bmatrix}
\]

\[
Y = \begin{bmatrix}
100 & 220 & 230 \\
45 & 95 & 120 \\
205 & 100 & 0 \\
\end{bmatrix}
\]

\[
Z_a = X - Y; \quad \% \text{ims subtract}(X, Y)
\]

\[
Z_b = Y - X; \quad \% \text{ims subtract}(X,Y)
\]

\[
Z_c = |X - Y|; \quad \% \text{im abs diff}(X,Y)
\]

\[
Z_a = \begin{bmatrix}
100 & 0 & 0 \\
0 & 0 & 0 \\
0 & 150 & 120 \\
\end{bmatrix}
\]

\[
Z_b = \begin{bmatrix}
0 & 120 & 130 \\
45 & 85 & 70 \\
155 & 0 & 0 \\
\end{bmatrix}
\]

\[
Z_c = \begin{bmatrix}
100 & 120 & 130 \\
45 & 85 & 70 \\
155 & 150 & 120 \\
\end{bmatrix}
\]
Multiplication and Division Operators

- Multiplication and division by a scalar are often used to perform brightness adjustments on an image.
- *Multiplicative image scaling*—makes each pixel value brighter (or darker) by multiplying its original value by a scalar factor:
  - if the value of the scalar multiplication factor is greater than one, the result is a brighter image;
  - if it is greater than zero and less than one, it results in a darker image.
- Multiplicative image scaling usually produces better subjective results than the additive image offset process described previously.
Multiplication and Division Operators

X  X*0.5  X/0.5
Combining Arithmetic Operations

• To combine several arithmetic operations applied to one or more images may compound the problems of overflow and underflow discussed previously.

• To achieve more accurate results without having to explicitly handle truncations and round-offs, the IPT offers a built-in function to perform a linear combination of two or more images: imlincomb.

• imlincomb computes each element of the output individually, in double-precision floating point.

• If the output is an integer array, imlincomb truncates elements that exceed the range of the integer type and rounds off fractional values.
**Combining Arithmetic Operations**

Example:

\[
X = \begin{bmatrix}
200 & 100 & 100 \\
0 & 10 & 50 \\
50 & 250 & 120 \\
\end{bmatrix} \quad Y = \begin{bmatrix}
100 & 220 & 230 \\
45 & 95 & 120 \\
205 & 100 & 0 \\
\end{bmatrix} \quad Z = \begin{bmatrix}
200 & 160 & 130 \\
145 & 195 & 120 \\
105 & 240 & 150 \\
\end{bmatrix}
\]

\[
Sa = (X + (Y + Z))/3; \quad \% \text{ imdivide(imadd(X,imadd(Y,Z)),3)}
\]

\( a = \text{uint16}(X) + \text{uint16}(Y) \)

\( b = a + \text{uint16}(Z) \)

\( Sb = \text{uint8}(b/3) \)

\( Sc = \text{imlincomb}(1/3,X,1/3,Y,1/3,Z,'\text{uint8}') \)

\[
S_a = \begin{bmatrix}
85 & 85 & 85 \\
63 & 85 & 85 \\
85 & 85 & 85 \\
\end{bmatrix} \quad S_b = \begin{bmatrix}
167 & 160 & 153 \\
63 & 100 & 97 \\
120 & 197 & 90 \\
\end{bmatrix} \quad S_c = \begin{bmatrix}
167 & 160 & 153 \\
63 & 100 & 97 \\
120 & 197 & 90 \\
\end{bmatrix}
\]
LOGIC OPERATIONS

• They are performed in a bit-wise for each pixel value.

• NOT operator requires only one argument.
LOGIC OPERATIONS

- AND, XOR, and OR operators require two or more operands.

![Diagram showing logic operations]

- $X \text{ AND } Y$
- $X \text{ OR } Y$
- $X \text{ XOR } Y$
- $(\neg X) \text{ AND } Y$
LOGIC OPERATIONS with Grayscale

$X$

$Y$

$Z = \text{bitand}(X,Y)$

$Z = \text{bitlor}(X,Y)$

$Z = \text{bitxor}(X,Y)$

$Z = \text{bitcmp}(X,Y)$