Chapter 3: Intensity Transforms

Andre L. C. Barczak

Computer Science
Massey University, Albany

Contents

Intensity transformations

Introduction

- Pixel by pixel
- General expression:

$$g(x, y) = T[i(x, y)] \tag{1}$$

- $T[i(x, y)]$ can be any function
- linear or non-linear etc

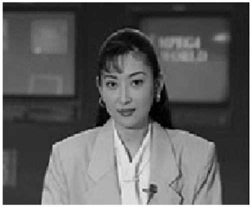
Example

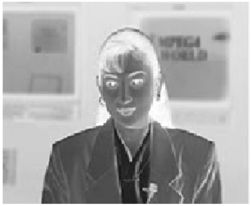
Negative image

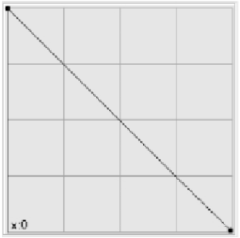
- A negative image of i(x,y) can be created using:

$$g(x,y) = 255 - i(x,y) \tag{2}$$

Negative image







A negative image of Akiyo1.jpg was created using $g(x, y) = 255 - i(x, y)$

Negative image

```
 int main(int argc, char** argv)
{
  namedWindow("image",0);
  namedWindow("image2",0);
  Mat image;
  image=imread(argv[1], CV_LOAD_IMAGE_GRAYSCALE);
  Mat image2(image.rows,image.cols,CV_8U);
  if(! image.data)  {printf("Error \n");exit(0);}
  for(int x=0; x<image.cols;x++){
    for(int y=0; y<image.rows;y++){
      Mpixel(image2,x,y)=255 - Mpixel(image,x,y);
    }
  }
  imshow("image", image);
  imshow("image2", image2);
  waitKey(0);
}
```

Threshold

non-linear functions

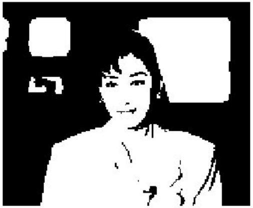
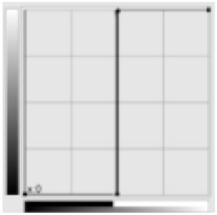
- A binary image of i(x,y) can be created using:

$$g(x,y) = \begin{cases} 0 & \text{if } i(x,y) <= 128 \\ 255 & \text{if } i(x,y) > 128 \end{cases} \tag{3}$$

Threshold

```
int main(int argc, char** argv)
{
...
   for(int x=0; x<image.cols;x++){
      for(int y=0; y<image.rows;y++){
         if (Mpixel(image,x,y)>=128) {
             Mpixel(image2,x,y)=255;
         }
         else Mpixel(image2,x,y)=0;
      }
   }
   imshow("image", image);
   imshow("image2", image2);
   waitKey(0);
}
```

Threshold

A binary image of Akiyo1.jpg was created using $g(x, y)$

Otsu's Threshold

For bimodal histograms, we can find an optimal threshold iteratively.
Otsu proposed a simple approach in 1979.

$$\sigma_w^2(t) = p_f(t)\,\sigma_f^2(t) + p_b(t)\,\sigma_b^2(t) \tag{4}$$

where: σ^2 is the variance and p is the portion of the pixels in one class or another and t is the threshold.

This approach is commonly used when the background is very different than the foreground, e.g., in microscope images.
In OpenCV:

```
threshold(greyscaleimage,threshimage,0,255,CV_THRESH_BINARY
| CV_THRESH_OTSU);
```

Contrast Stretching

equation

$$\bar{i}(x,y) = \frac{255(i(x,y) - \mu + c\sigma)}{2c\sigma} \tag{5}$$

Where:
, $c \in \Re^+$ (typical values for c are between 1 to 2)
and $0 \leq \bar{i}(x,y) \leq 255$
In a digital image, one has to cut off values that are out of range.
μ is the mean of the pixel values
σ is the standard deviation (could alternatively be σ^2) of the pixel values

Contrast Stretching

Note that after the operation the contrast is much sharper.

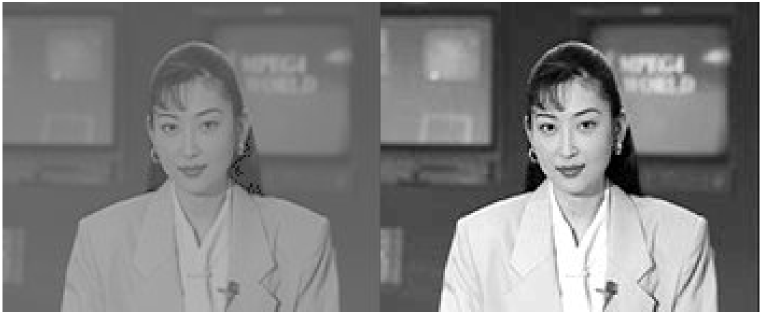


Figure 1A: Before 1B: After

Contrast Stretching

The dark image appears more clear after the operation.

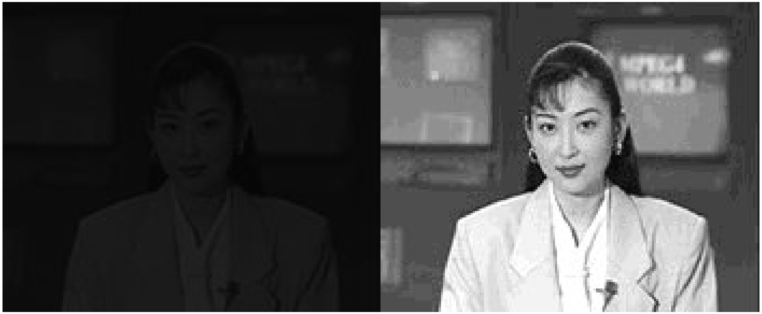


Figure 2A: Before 2B: After

Contrast Stretching code

```
int main(int argc, char** argv)
{
...
  double mean, var, sdev;
  int N = image.rows*image.cols;  float c=1.5;
  for(int x=0; x<image.cols;x++){
    for(int y=0; y<image.rows;y++){
      mean = mean + Mpixel(image,x,y);
      var = var + pow(Mpixel(image,x,y),2);
    }
  }
  mean=mean/N;
  var = (var / N ) - (mean * mean);
  sdev = sqrt(var);

...
```

Contrast Stretching (cont.)

```
...
  for(int x=0; x<image.cols;x++){
    for(int y=0; y<image.rows;y++){
      float cpixel = (255.0*(Mpixel(image,x,y) -
          mean + c*sdev))/(2*c*sdev);
      if (cpixel>255) cpixel=255;
      if (cpixel<0) cpixel=0;
      Mpixel(image2,x,y) = cvRound(cpixel);
    }
  }
  imshow("image", image);
  imshow("image2", image2);
  waitKey(0);
}
```

Histogram equalisation

another way to improve contrast

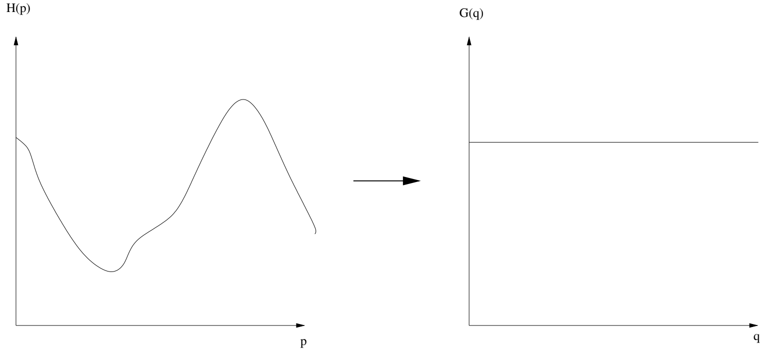
- Histograms can be equalised or flattened
- This usually (not always) improves contrast
- The resulting histogram is not truly flat

Algorithm 2

```
Require: The input is a grey-scale image and the output is a grey-scale
    image2.
    Auxiliary datastructures are two histograms $H[i]$, $HC[j]$ and a transform
    $T[k]$.
1: Create an array $H[r]$. {$r$ is the range of possible pixel values.}
2: Compute the histogram $H$. For each pixel $p$ of value $i$:
        $H[i] = H[i] + 1;$
3: Compute a cumulative histogram $HC$:
        $HC[0] = H[0]$
        $HC[i] = HC[i-1] + H[i]$ {for i=1,2,3...255 for grey-scale images}
4: Set a discrete transformation function:
        $T[i] = round(\frac{255}{width.height}.HC[i])$ for i=1,2,3...255
5: Create image2 scanning all pixels $p'$ adopting values according to:
        $p' = T[p']$
```

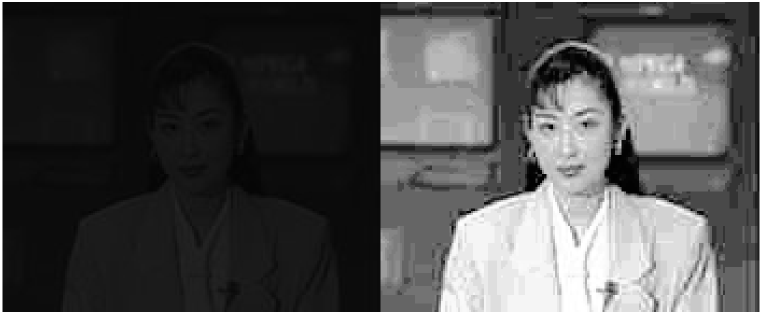
Histogram equalisation

We want a “flat” histogram...



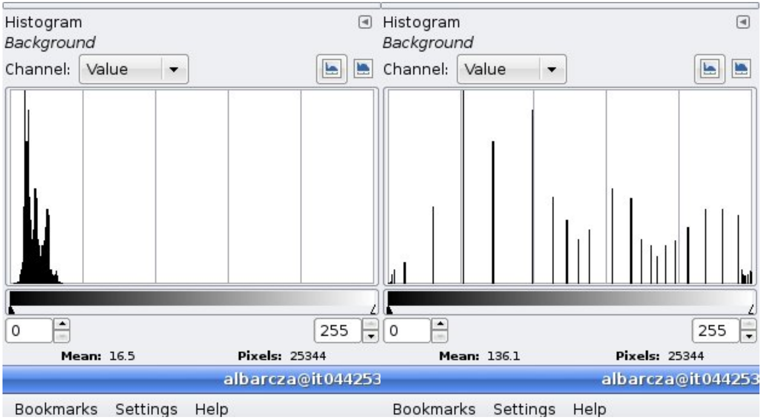
Histogram equalisation

If we do that in practice...



Histogram equalisation

...what we really get is not flat.



Mean: 16.5 Pixels: 25344

albarcza@it044253

Mean: 136.1 Pixels: 25344

albarcza@it044253

These histograms were create with GIMP.

Histogram equalisation example

We use a simple example with 3 bits (values from 0 to 7). The image is:

1	2	2	2
3	1	1	2
4	5	7	7
2	2	1	1

Histogram equalisation example

Compute the histogram H[r]:

H[0] = 0	H[1]=5
H[2] = 6	H[3]=1
H[4] = 1	H[5]=1
H[6] = 0	H[7]=2

Compute the cumulative histogram HC[r]:

HC[0] = 0	HC[1]=5
HC[2] = 11	HC[3]=12
HC[4] = 13	HC[5]=14
HC[6] = 14	HC[7]=16

Histogram equalisation example

Compute T[i] = round(7 HC[i] / W H)

T[0]=0	T[1]=7 . 5 / 16 $\approx$ 2
T[2]=7 . 11 / 16 $\approx$ 5	T[3]=7 . 12 / 16 $\approx$ 5
T[4]=7 . 13 / 16 $\approx$ 6	T[5]=7 . 14 / 16 $\approx$ 6
T[6]=7 . 14 / 16 $\approx$ 6	T[7]=7 . 16 / 16 = 7

Histogram equalisation example

Applying the transform T[i] to the image:

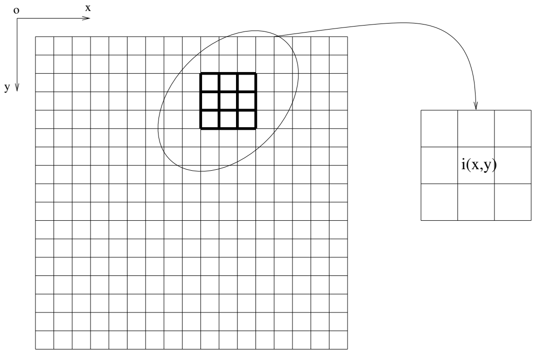
1	2	2	2
3	1	1	2
4	5	7	7
2	2	1	1

$\rightarrow$

2	5	5	5
5	2	2	5
6	6	7	7
5	5	2	2

Spatial Filtering
defining masks or kernels

Lets start with simple masks:



Spatial Filtering

swapping kernels

$$i(x, y) * M(m, n) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} M(s, t) i(x + s, y + t) \tag{6}$$

Where: $i(x, y)$ is the original image
M is a convolution mask of size m x n, $m = 2a + 1$ and $n = 2b + 1$
Common sizes for convolution masks are 3x3, 5x5 and 7x7

Smoothing masks

average of pixels

These two masks will “smooth” the edges of the image

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16

Sobel Operators

using derivatives vertically or horizontally

We have to apply the masks separately to find edges on the vertical or horizontal.

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Gradients

approximations

The masks used for finding the gradients originated from approximations of the derivatives:

$$\nabla i \approx (z_7+2z_8+z_9)-(z_1+2z_2+z_3)+(z_3+2z_6+z_9)-(z_1+2z_4+z_7)$$

Laplacian

using derivatives

These two masks are an approximation of a Laplacian of the image. Applying one of these will result in an image with the edges only.

0	1	0
1	−4	1
0	1	0

1	1	1
1	−8	1
1	1	1

Gradients

approximations

The masks used for the Laplacian originated from approximations of the second derivatives:

$$\nabla^2 i = z_2 + z_4 + z_6 + z_8 - 4z_5 \tag{7}$$

Sharpening masks

using derivatives

These two masks will "sharpen" the edges of the image

0	−1	0
−1	5	−1
0	−1	0

−1	−1	−1
−1	9	−1
−1	−1	−1

Sharpening masks

using derivatives

The sharpening masks can be obtained by subtracting the image from the Laplacian mask.
The image is the mask in the left (if you convolve the image with that mask you make a copy of the image).

$$\begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline\end{array} - \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & -4 & 1 \\ \hline 0 & 1 & 0 \\ \hline\end{array} = \begin{array}{|c|c|c|}\hline 0 & -1 & 0 \\ \hline -1 & 5 & -1 \\ \hline 0 & -1 & 0 \\ \hline\end{array}$$

Median filter

non-linear filters

The Median filter is a non Linear operator.
The central pixel is replaced by the **median** obtained in the kernel area.

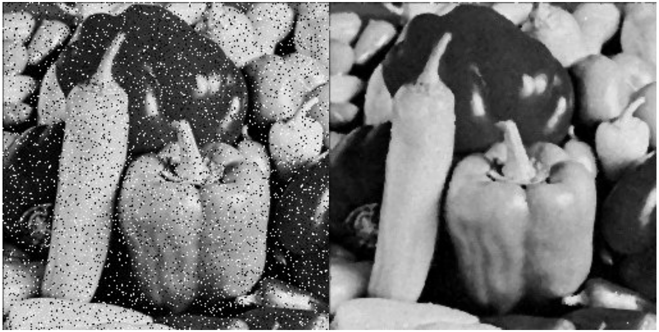


Figure 1: Median filter example.

Kuwahara filter

non-linear filters

- non-linear as well.
- the kernel is divided into four different areas.
- the kernel assumes the value of the mean brightness of the area with the smallest variance.

$$\sigma_a = \sqrt{\frac{1}{(N-1)} \sum (i(x,y) - \mu_a)^2} \tag{8}$$

Kuwahara filter

non-linear filters

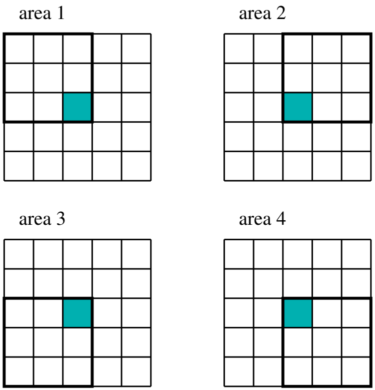


Figure 2: The Kuwahara filter.

Kuwahara filter

Example

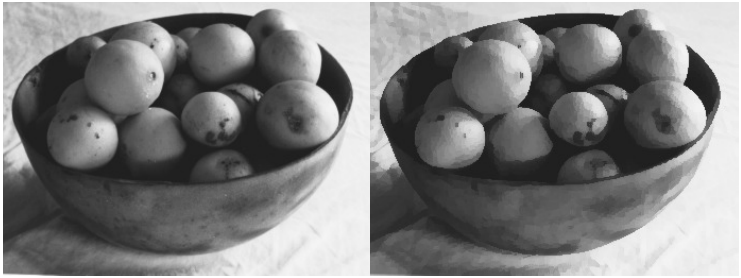


Figure 3: Kuwahara filter results for Limes.

Kuwahara filter

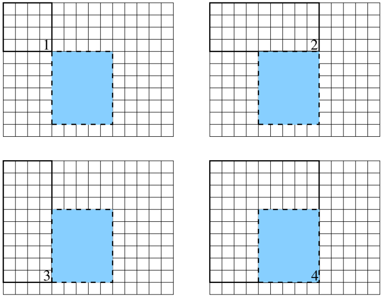
Implementation

- very slow if kernel size if large.
- How to get a better run-time?
- Use Summed-area Tables (Integral Images)

$$I(x, y) = \sum_{x \leq x_i, y \leq y_i} i(x_i, y_i) \tag{9}$$

Kuwahara filter

Implementation using SATs

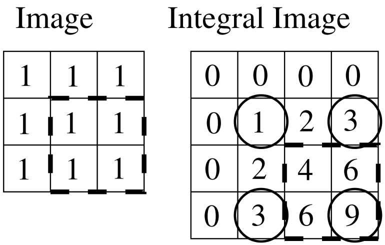


For a square area, one needs four lookups:

$$Area = pt_4 - pt_2 - pt_3 + pt_1 \tag{10}$$

SAT (Integral Image)

Implementation



1	1	1
1	1	1
1	1	1

0	0	0	0
0	1	2	3
0	2	4	6
0	3	6	9

Figure 4: Integral image example.

$$ii(x,y) = ii(x-1,y) + ii(x,y-1) - ii(x-1,y-1) + i(x,y) \quad (11)$$

Affine Transforms

General case:

General case equations: these cover rotation, translation, scaling and skewing.

$$x' = a_0 + a_1 x + a_2 y \tag{12}$$

$$y' = b_0 + b_1 x + b_2 y \tag{13}$$

Affine Transforms

Rotation:

For Rotation only (with center at the origin).

$$x' = xcos(\alpha) + ysin(\alpha) \tag{14}$$

$$y' = -xsin(\alpha) + ycos(\alpha) \tag{15}$$

Affine Transforms

Scaling:

Scaling only:

$$x' = ax \tag{16}$$

$$y' = bx \tag{17}$$

Affine Transforms

Skewing:

Skewing at angle α:

$$x' = x + ytan(\alpha) \tag{18}$$

$$y' = y \tag{19}$$

Using matrices

vector (x, y) or $\begin{vmatrix} x \\ y \end{vmatrix}$ augmented vector has the form: $\begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$

general affine transforms can be represented by:

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Rotation example:

using matrices, the rotation of an image 45 degrees centred at the origin is:

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} 0.707 & 0.707 & 0.0 \\ -0.707 & 0.707 & 0.0 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

and indeed this yields (for $\alpha = 45^\circ$):

$$x' = x\,cos(\alpha) + y\,sin(\alpha) = x\,0.707 + y\,0.707$$

$$y' = -x\,sin(\alpha) + y\,cos(\alpha) = -x\,0.707 + y\,0.707$$

Rotation with centre other than origin:

using matrices, the rotation of an image 45 degrees for a 100x100 image

$$\begin{vmatrix} x' \\ y' \end{vmatrix} = \begin{vmatrix} 0.707 & 0.707 & 0.0 \\ -0.707 & 0.707 & 0.0 \end{vmatrix} \begin{vmatrix} x \\ y \\ 1 \end{vmatrix}$$

Now we need to translate the centre of the resulting image to the centre of the original image
The old centre is (50,50). The new centre is at:
$x' = x\,0.707 + y\,0.707 = 70.71$
$y' = -x\,0.707 + y\,0.707 = 0$
So the translation needs to be: $x - x' = 100/2 - 70.71 = -20.71$
$y - y' = 100/2 - 0 = 50$
And therefore the matrix is: $\begin{vmatrix} 0.707 & 0.707 & -20.71 \\ -0.707 & 0.707 & 50.0 \end{vmatrix}$

Rotation with centre other than origin:

The matrix:

$$\begin{vmatrix} 0.707 & 0.707 & -20.71 \\ -0.707 & 0.707 & 50.0 \end{vmatrix}$$

can also be computer via OpenCV using: `Mat rotmatrix=getRotationMatrix2D(center, angle, scale);` where scale=1

To use the matrix in OpenCV: `warpAffine(image, image2, rotmatrix, image.size());`

add an extra parameter to get interpolation, e.g.: `INTER_LINEAR);`

Rotation with centre other than origin:

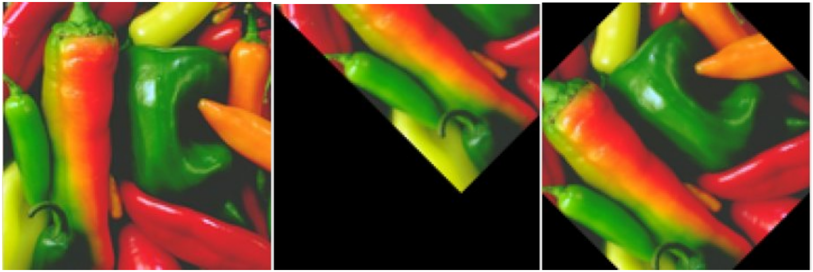


Figure 5: Rotating: original, centre at the origin, centre at the centre.

Exercise 1

1. Compute an affine matrix for: centre at the middle of the image (image size is 176x144), angle=30°, and scaling=1.
2. Write a program using OpenCV that implements rotation, translation, scaling and skewing for a grey-scale image.
3. Is there anything wrong with the resulting image? Why?
4. Fix the problem using an inverted approach.
5. Use OpenCV approach to achieve the same results. (e.g., use `warpAffine()` with a 2x3 matrix)

Exercise 2

1. Write a program that smooths the image using a linear filter. The program should allow the kernel for the linear filter to be created at different sizes (take a parameter for the kernel size).

Exercise 3

1. Write a program for histogram equalisation.
2. Modify the program for histogram *specification*. Your program should take 6 parameters:
   `equalise input.bmp output.bmp x y x’ y’`
   This should load the input.bmp image and produce the output.bmp file. The equalisation should be specified by the histogram from the rectangular area in the image defined by the points (x,y) and (x’,y’).