中国科学技术大学

University of Science and Technology of China

图像颜色变换

Color Transformation of Images

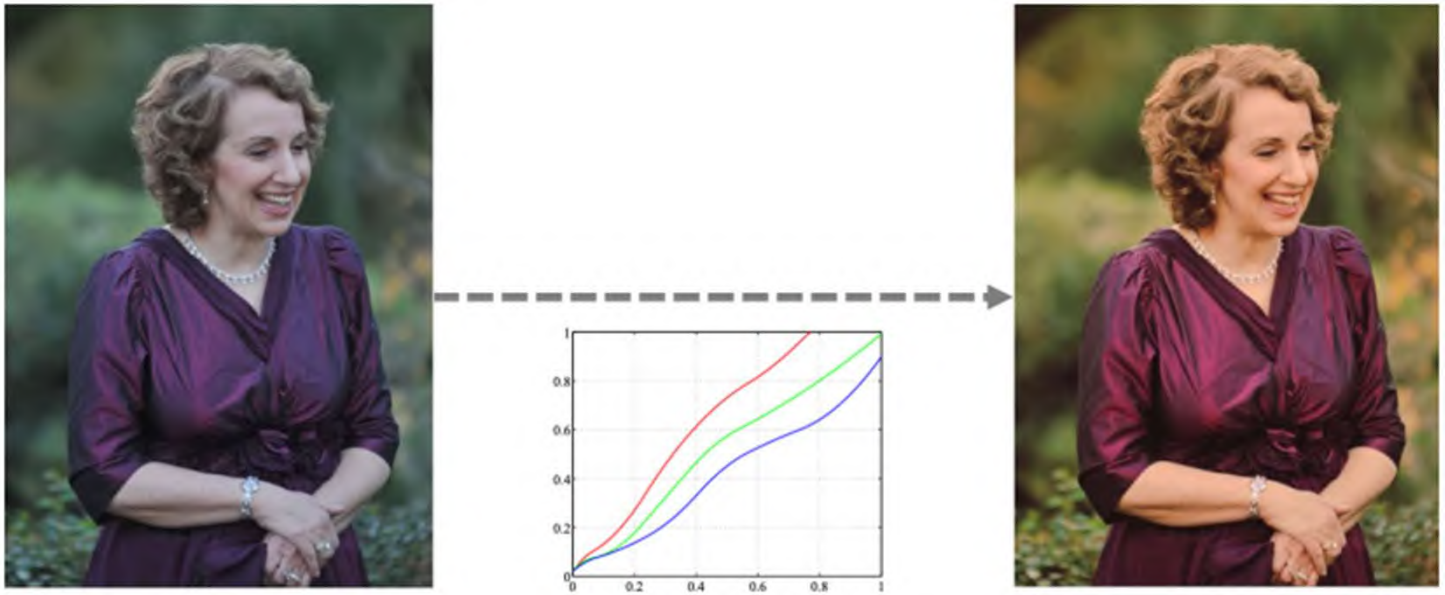
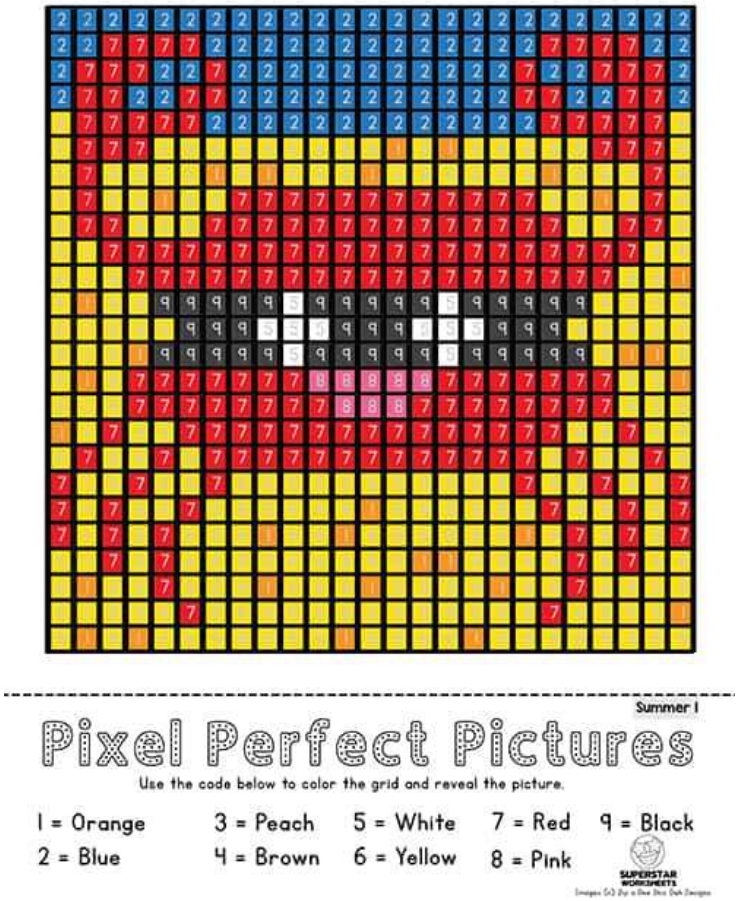


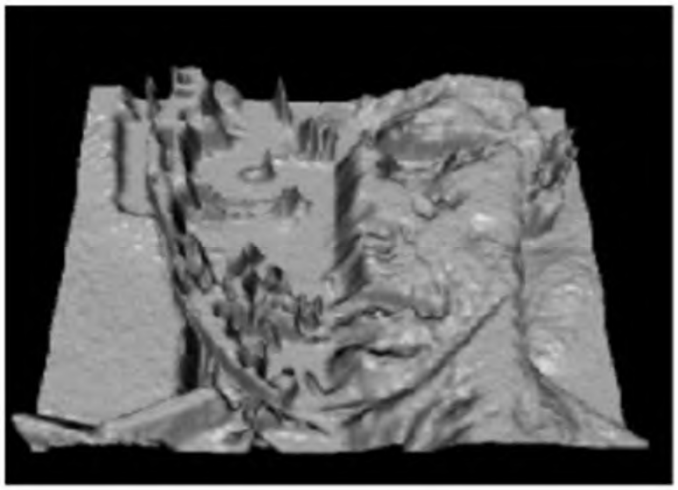
Image as function of Coordinates

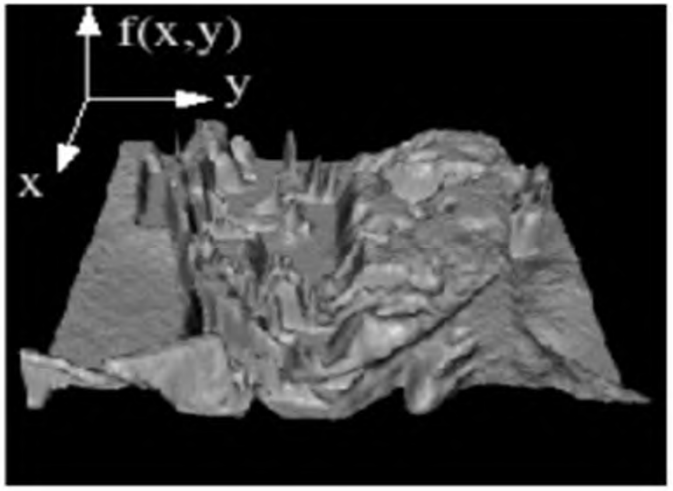


$$f(x,y) = \boldsymbol{c}, R^2 \rightarrow R^d, d \in \{1,3\}$$

$$x \in [0,w), y \in [0,h)$$

Image as function of Coordinates





Visualization of single channel grayscale image $f(x, y)$

Color Transform v.s. Geometry Transform

image warping: change ***domain*** of image

$$g(x) = f(T(x))$$

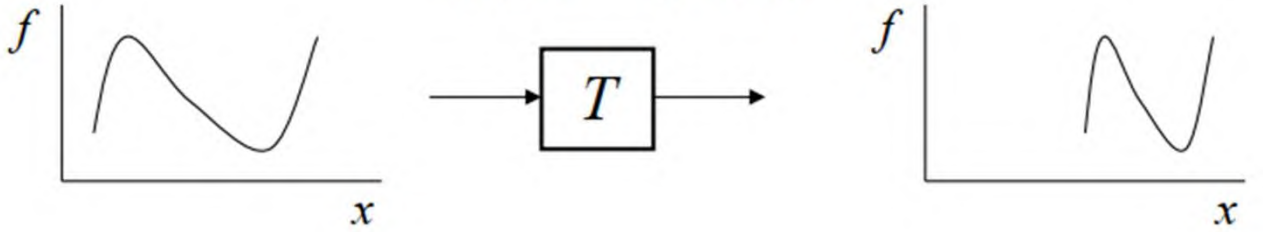
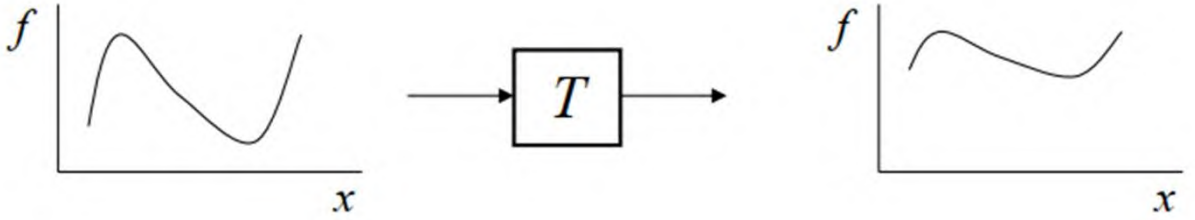


image filtering: change ***range*** of image

$$g(x) = T(f(x))$$



Color Transform v.s. Geometry Transform

image warping: change ***domain*** of image

$$g(x) = f(T(x))$$

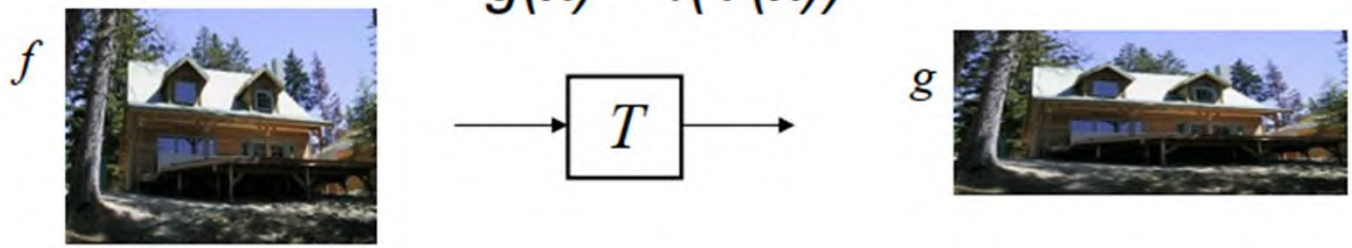
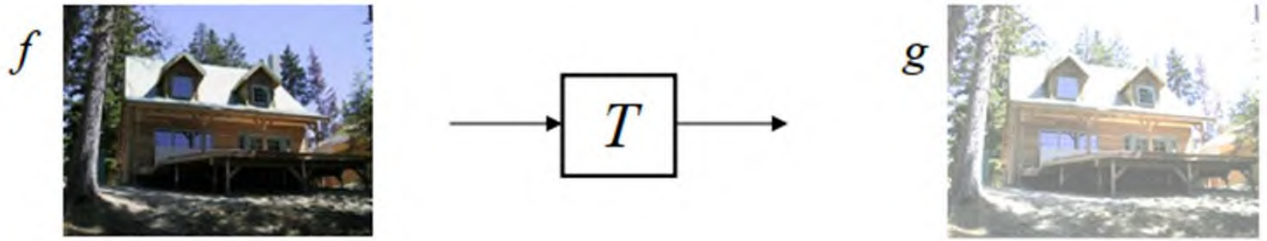


image filtering: change ***range*** of image

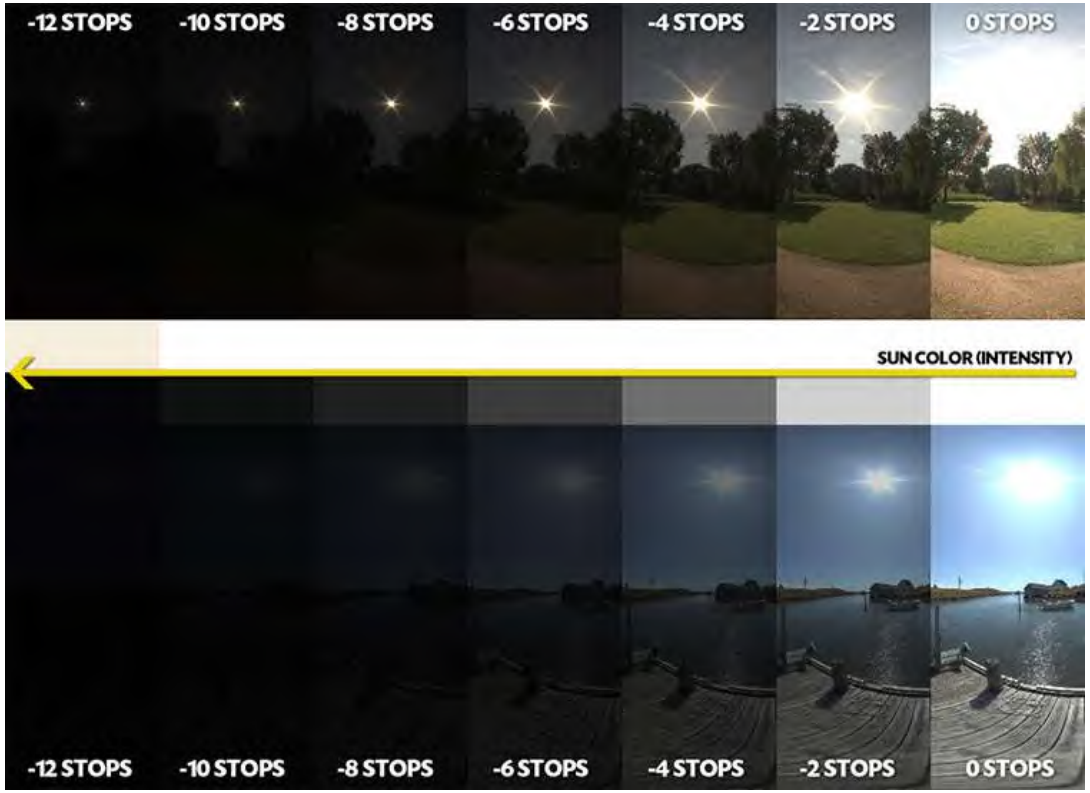
$$g(x) = T(f(x))$$



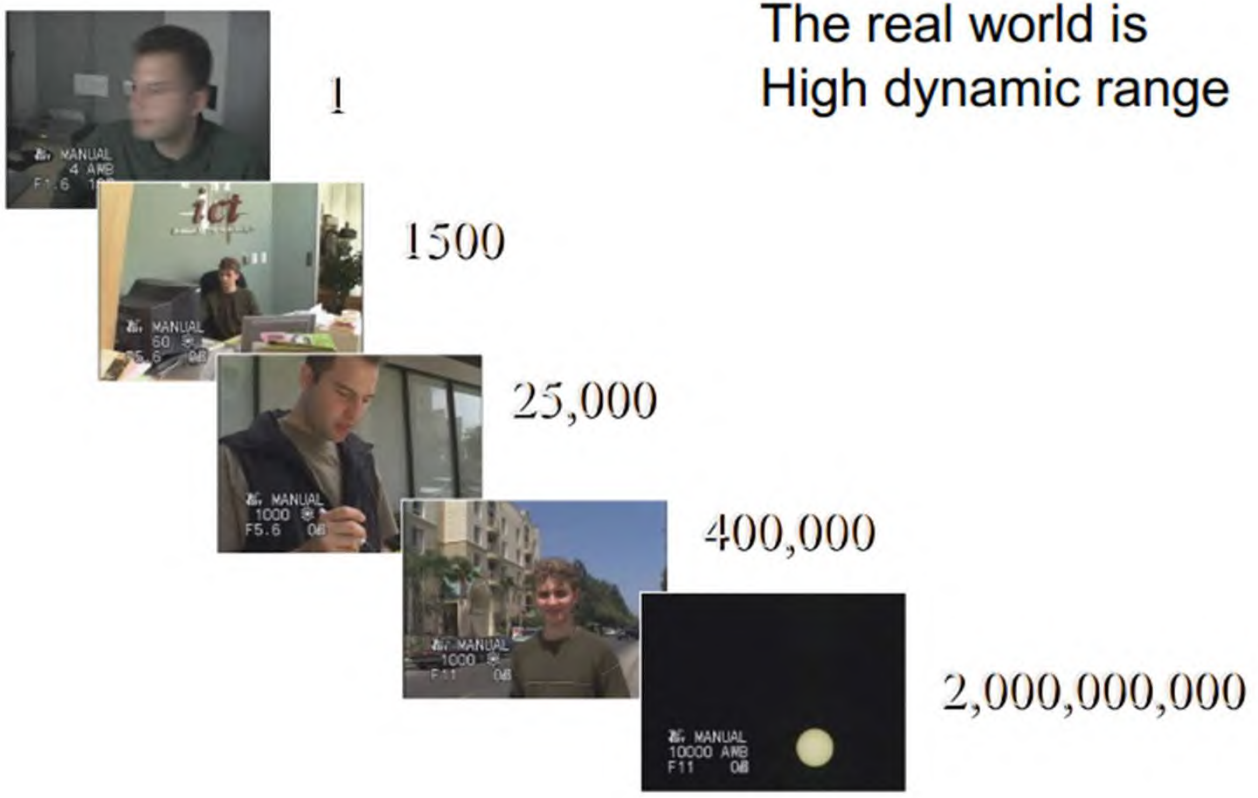
An example: HDR Image

What Is Dynamic Range in Photography?

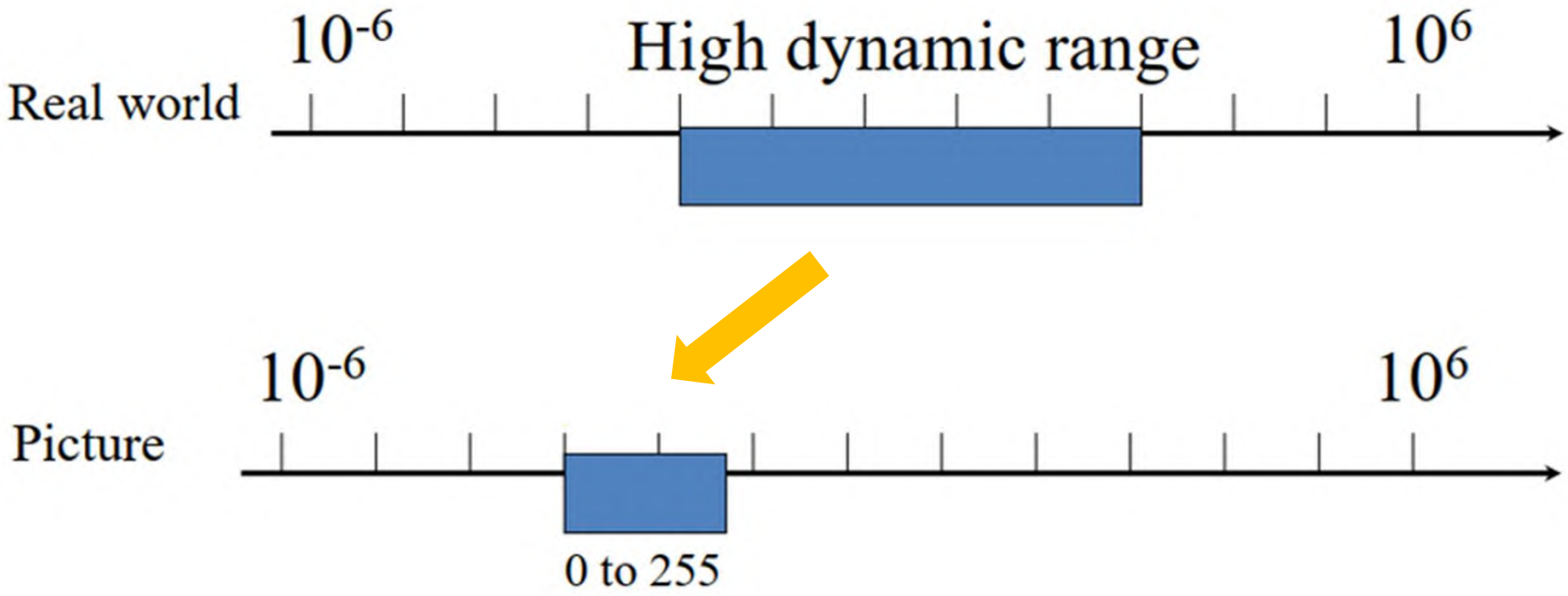
In photography, dynamic range is the contrast ratio between the darkest and brightest color tones that a camera can capture in a single exposure. Maximum dynamic range is the greatest range of light a digital camera sensor or strip of film can capture. Dynamic range is measured in stops. Each stop indicates a doubling of the level of brightness captured. While the human eye can see up to 20 stops of dynamic range, even high-end DSLR and mirrorless cameras can only reach around 14 stops.



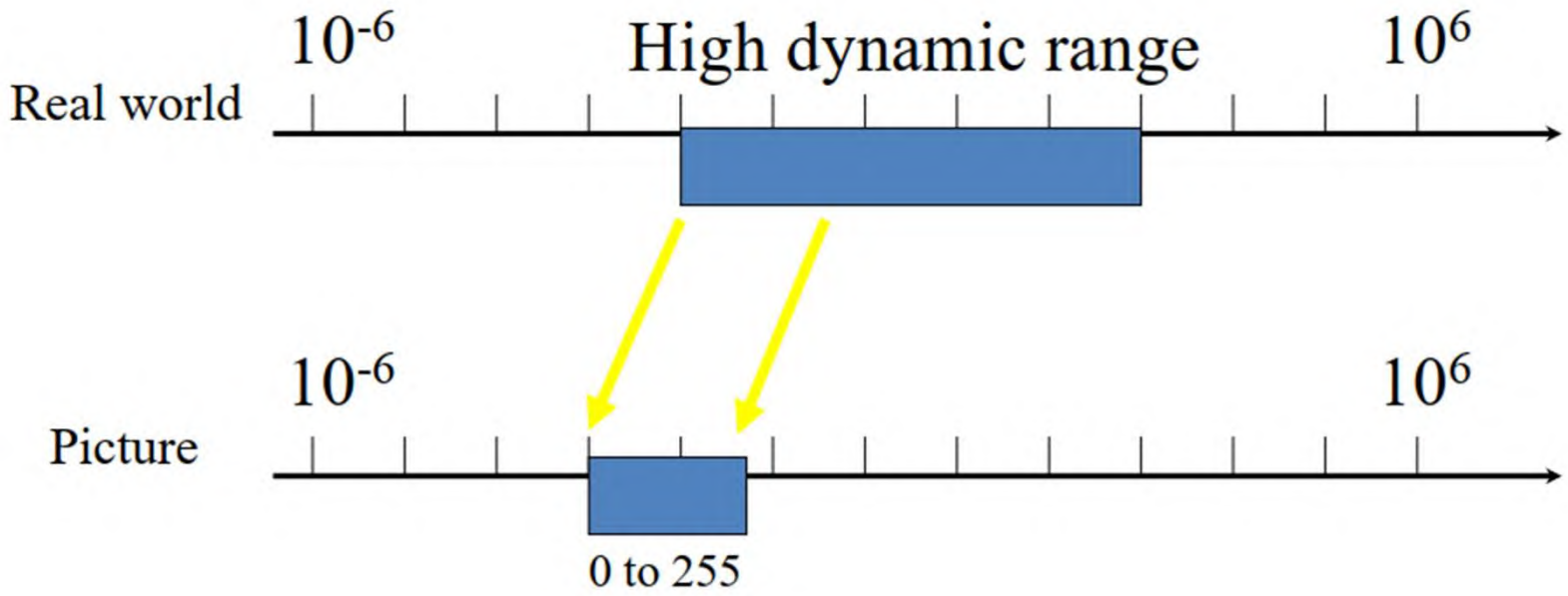
An example: HDR Image



An example: HDR Image

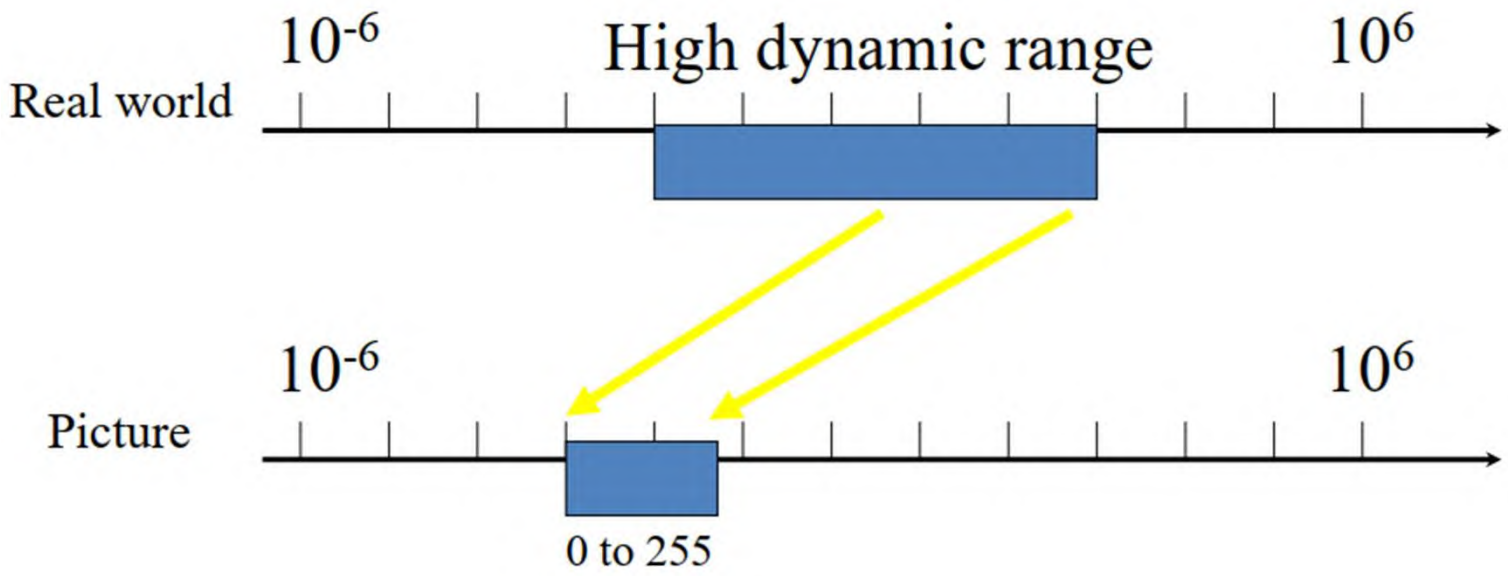


Capture Dark Light



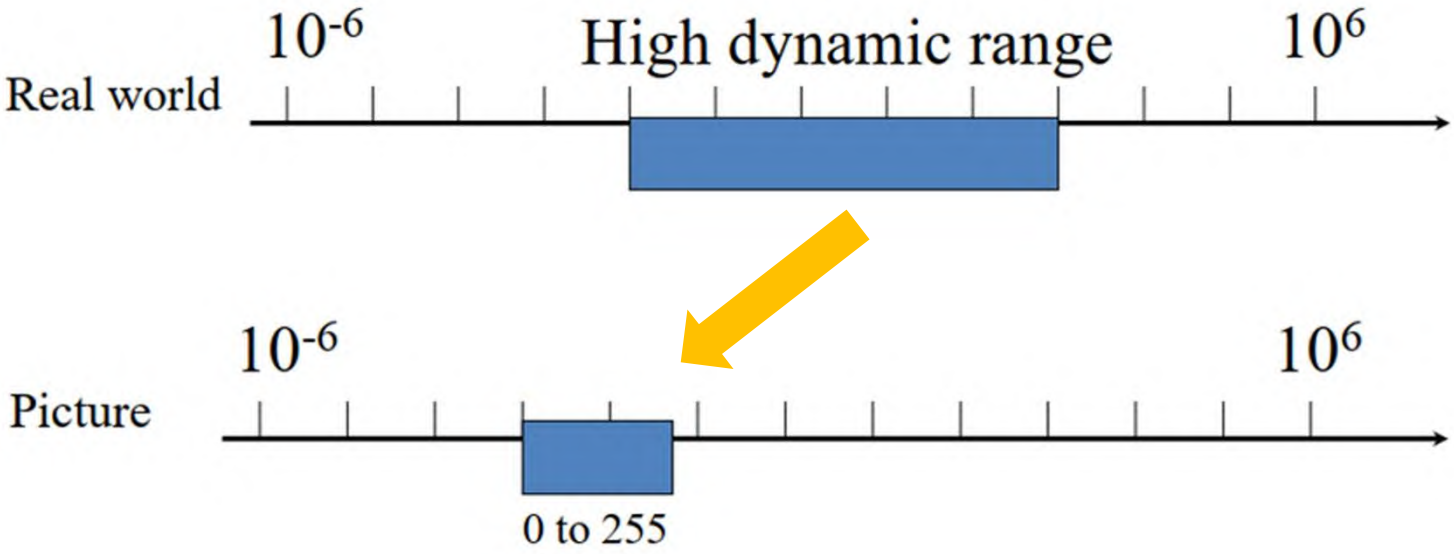
长时间曝光 Long Exposure

Capture Bright Light



短时间曝光 Short Exposure

An example: HDR Image



Basic Transform: Enhancement

a b
c d

FIGURE 3.9
(a) Aerial image. (b)–(d) Results of applying the transformation in Eq. (3.2-3) with $c = 1$ and $\gamma = 3.0, 4.0$, and 5.0, respectively. (Original image for this example courtesy of NASA.)

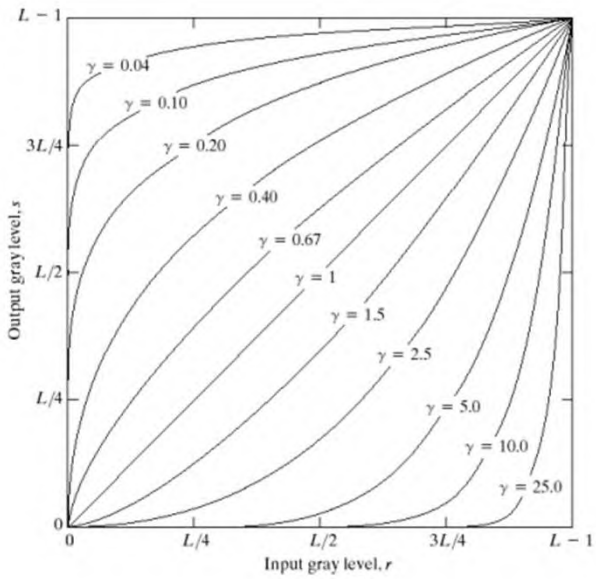


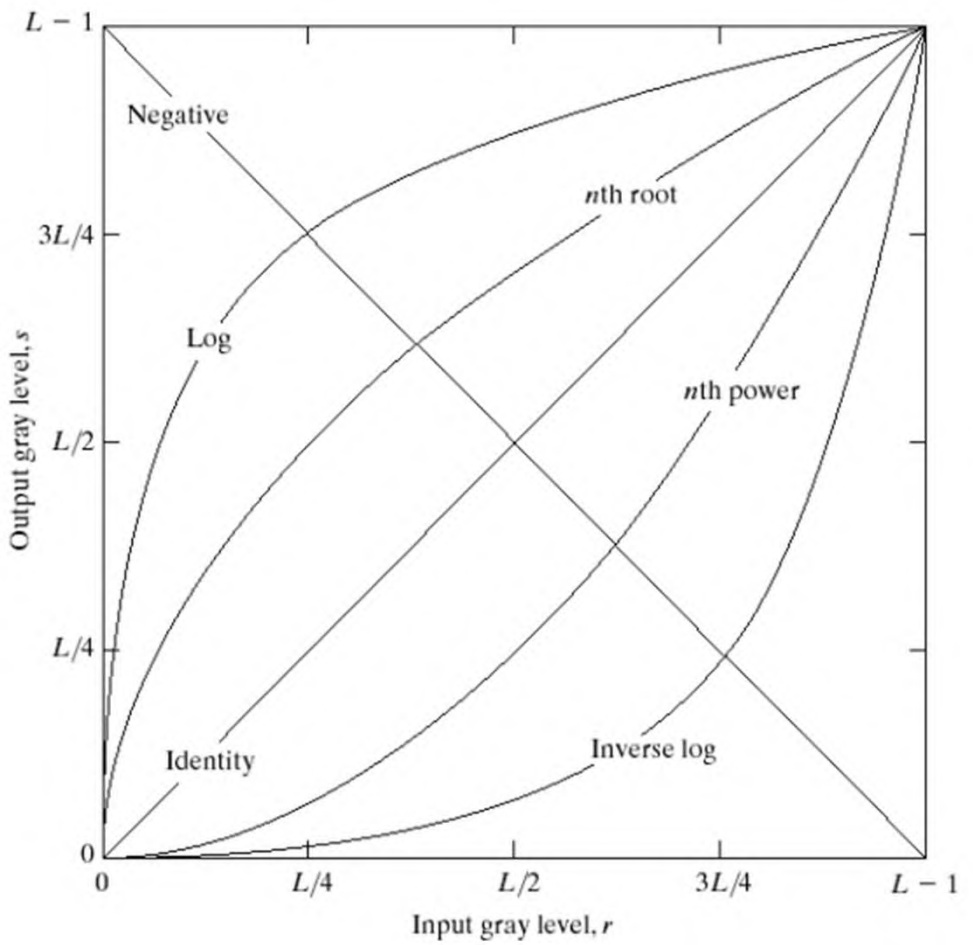
FIGURE 3.6 Plots of the equation $s = cr^{\gamma}$ for various values of γ ($c = 1$ in all cases).

$$c \cdot f(x,y)^{\gamma}$$

Power-law transformations

Basic Transform: Enhancement

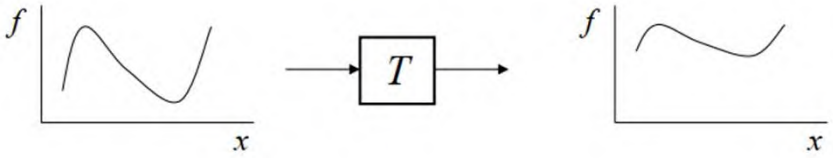
FIGURE 3.3 Some basic gray-level transformation functions used for image enhancement.



Different T for different tasks

image filtering: change ***range*** of image

$g(x) = T(f(x))$



Gamma Correction

$$T(c;\gamma) = c^{\gamma}$$

上述变换的特点

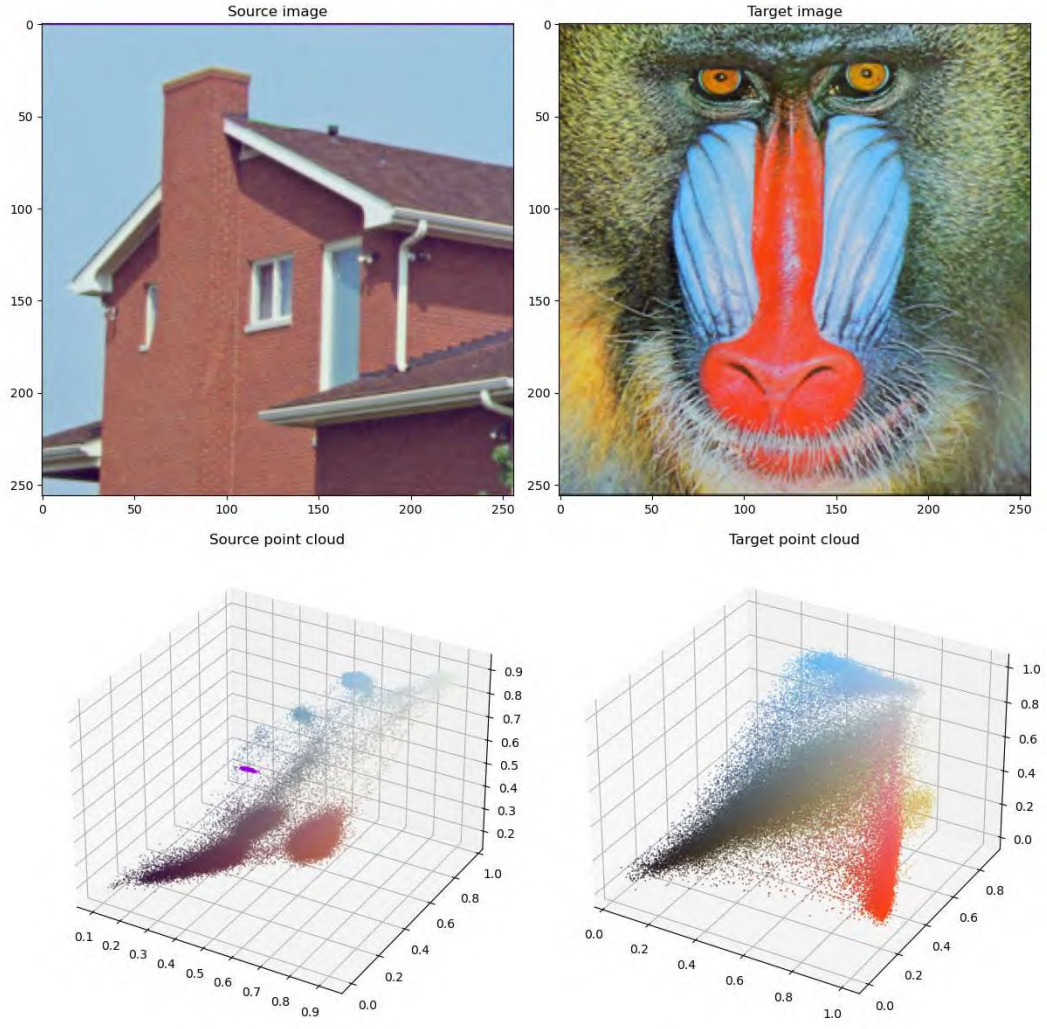
Shared transformation for all pixels

- In a digital image, point = pixel.
- Point processing transforms a pixel's value as function of its value alone;
- it does not depend on the values of the pixel's neighbors.

Point Processing of Images

类比：三维几何点云的刚性变换

图像看作3D点云

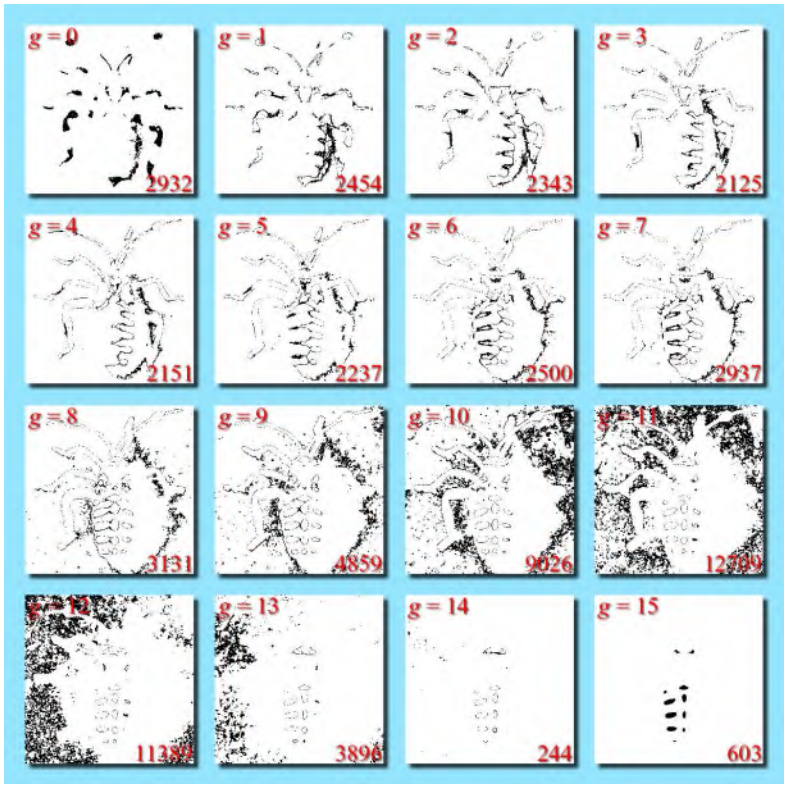


每个像素的颜色都是三维空间的一个样本

Histogram of (Grayscale) Image

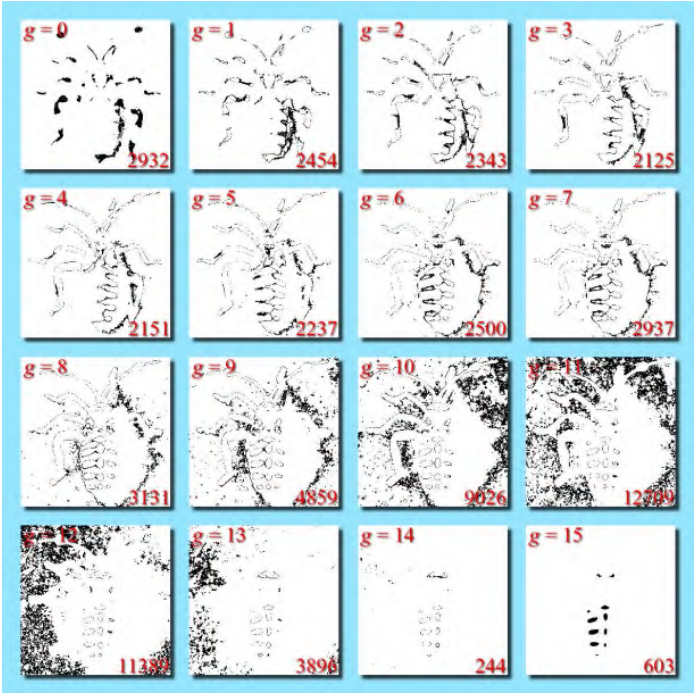
16-level (4-bit) image

lower RHC: number of pixels with intensity g

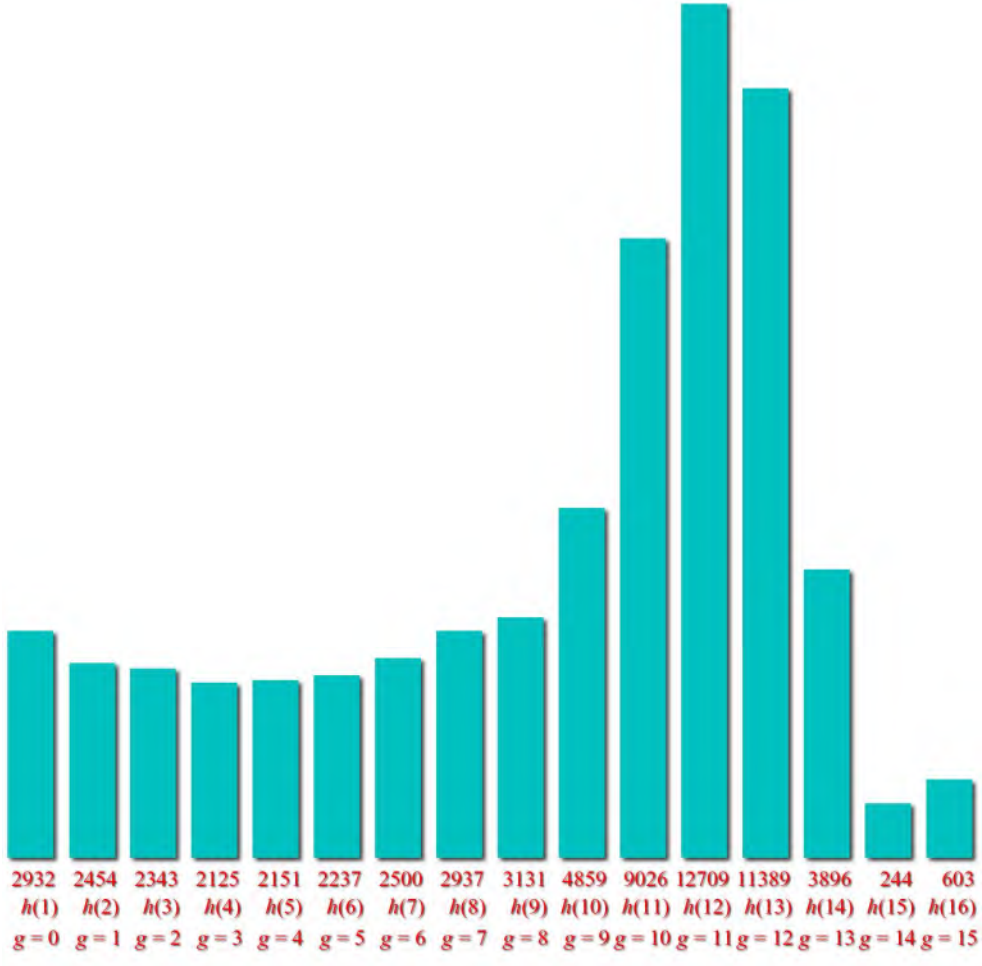


black marks pixels with intensity g

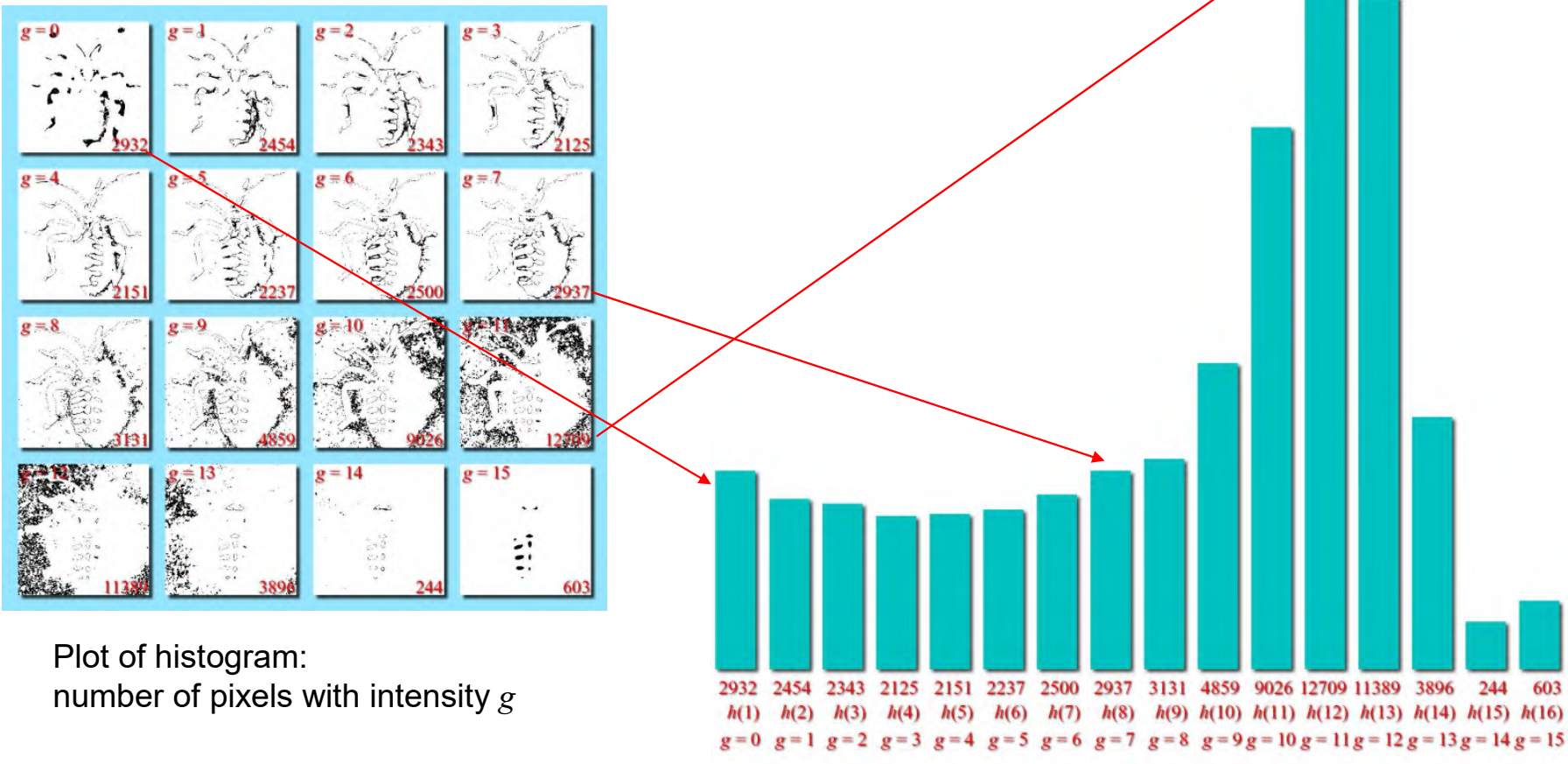
Histogram of Grayscale Image



Plot of histogram:
number of pixels with intensity g



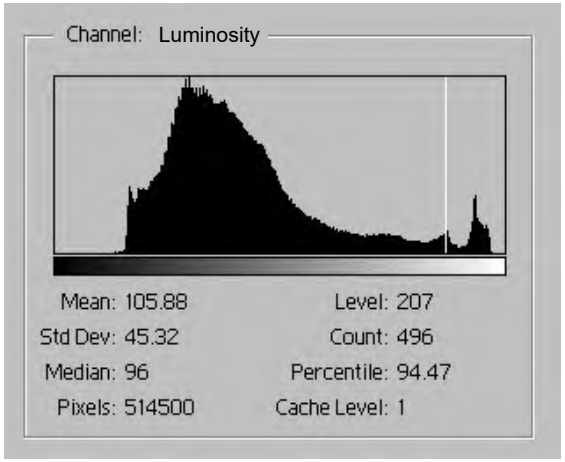
Histogram of Grayscale Image



Plot of histogram:
number of pixels with intensity g

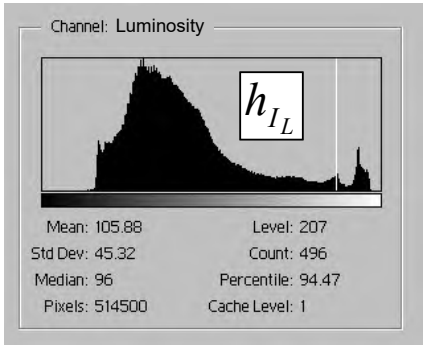
Histogram of Grayscale Image

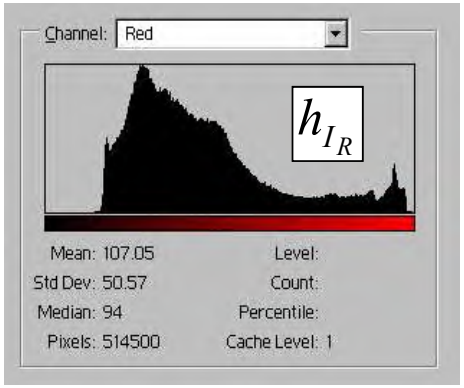
$h_{\mathbf{I}}(g+1) =$ the number of pixels in $\mathbf{I}$ with graylevel g.

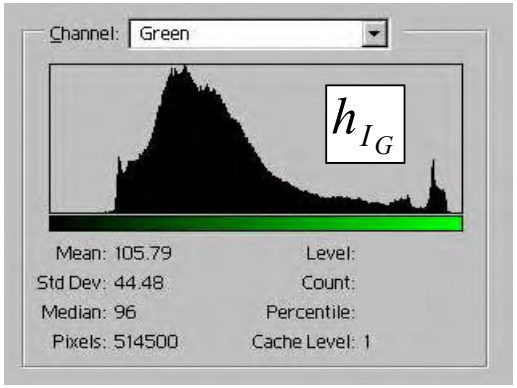


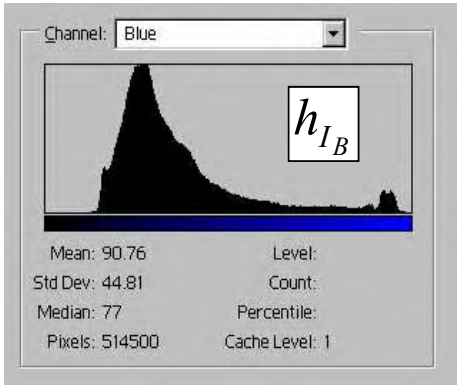
Histogram of Color Image

There is one histogram per color band R, G, & B. Gray histogram 0.299*R+0.587*G+0.114*B



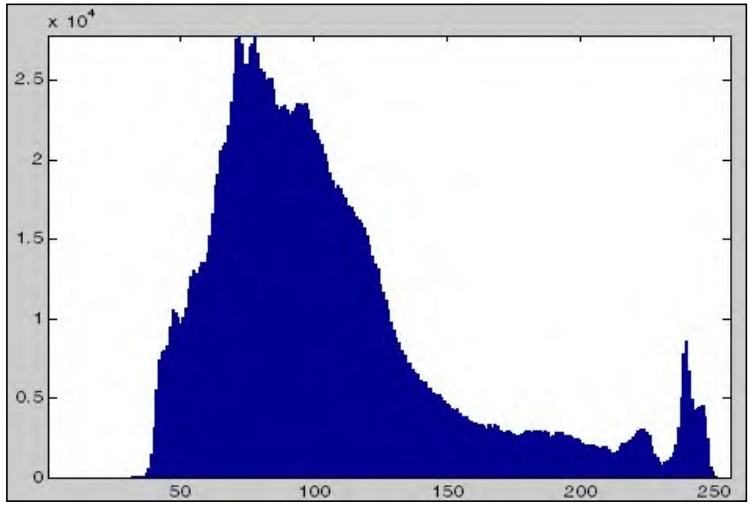






Transformations in Histogram

Kinkaku-ji (Temple of the Golden Pavilion), also known as Rokuon-ji (Deer Garden Temple), is a Zen Buddhist temple in Kyoto, Japan.

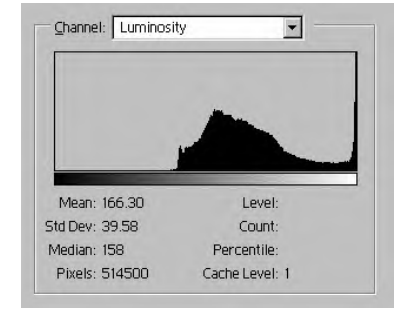


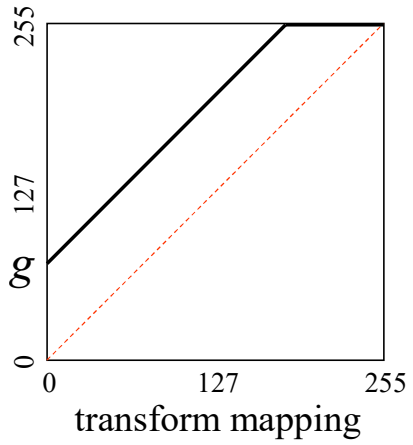
Luminance Histogram

Mean ++

$$\mathbf{J}(r,c,b) = \begin{cases} \mathbf{I}(r,c,b) + g, & \text{if } \mathbf{I}(r,c,b) + g < 256 \\ 255, & \text{if } \mathbf{I}(r,c,b) + g > 255 \end{cases}$$

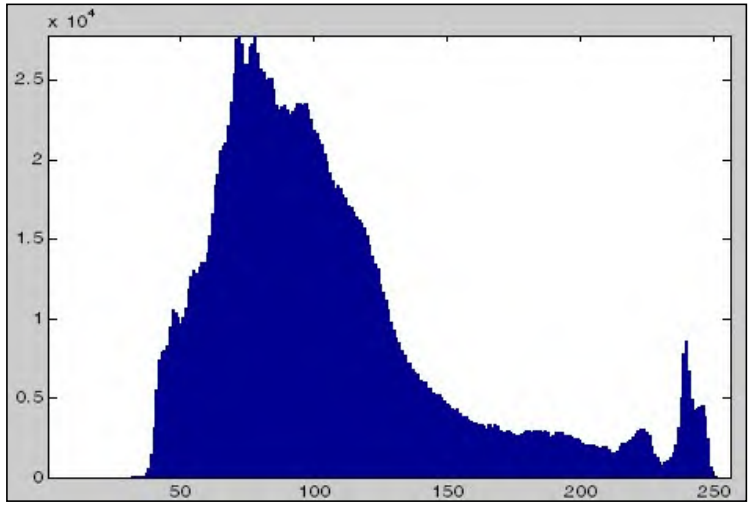
$g \geq 0$ and $b \in \{1,2,3\}$ is the band index.





Transformations in Histogram

Kinkaku-ji (Temple of the Golden Pavilion), also known as Rokuon-ji (Deer Garden Temple), is a Zen Buddhist temple in Kyoto, Japan.



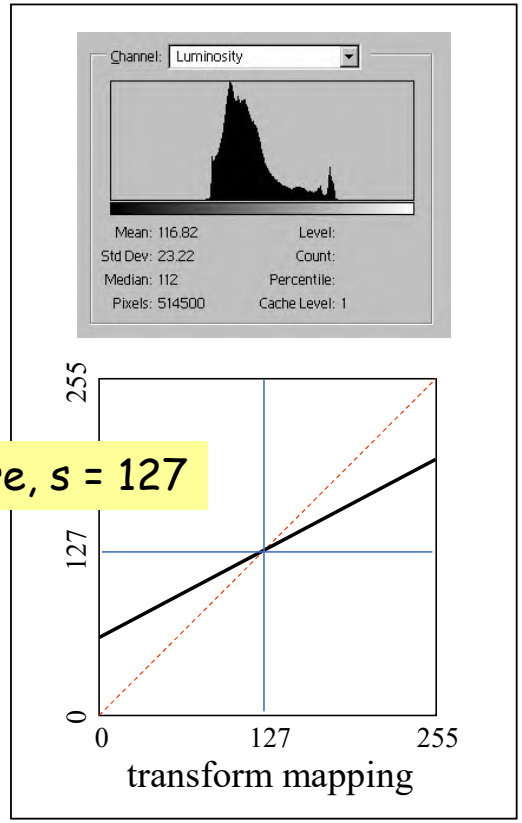
Luminance Histogram

Decrease Contrast

$$\mathbf{T}(r,c,b) = a[\mathbf{I}(r,c,b) - s] + s,$$
where $0 \le a < 1.0$,
$s \in \{0,1,2,\rightleftharpoons,255\}$, and
$b \in \{1,2,3\}$.

s is the center of the contrast function.

Here, s = 127

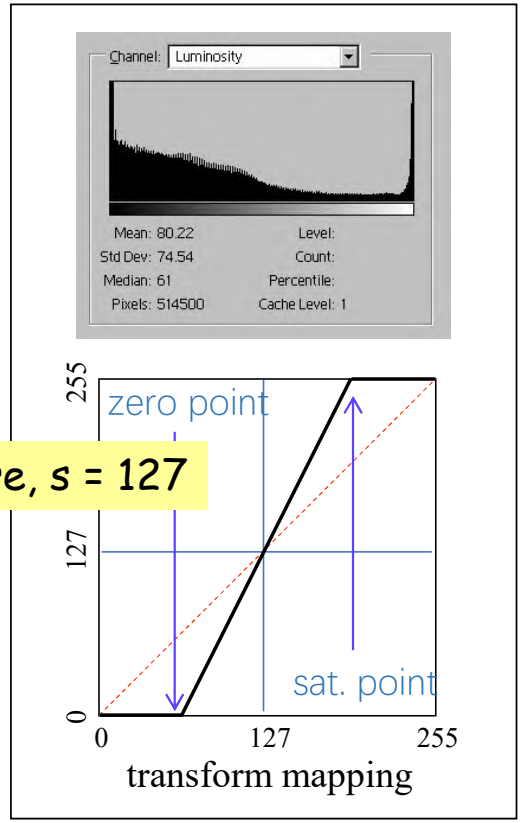


Increase Contrast

$$\mathbf{T}(r,c,b) = a\left[\mathbf{I}(r,c,b) - s\right] + s$$

$$\mathbf{J}(r,c,b) = \begin{cases} 0, & \text{if } \mathbf{T}(r,c,b) < 0, \\ \mathbf{T}(r,c,b), & \text{if } 0 \le \mathbf{T}(r,c,b) \le 255, \\ 255, & \text{if } \mathbf{T}(r,c,b) > 255. \end{cases}$$

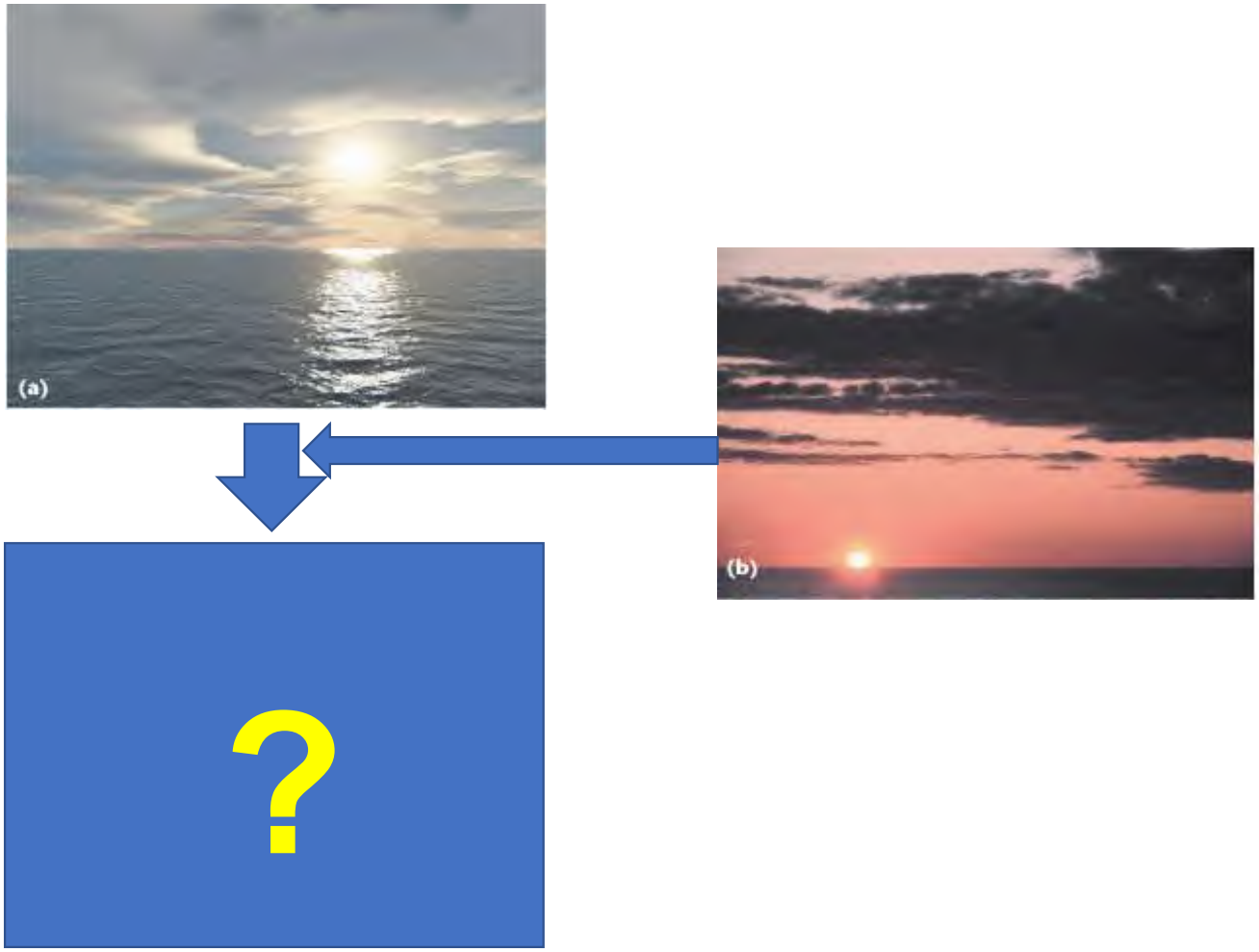
$$a > 1, \quad s \in \{0, \rightleftharpoons, 255\}, \quad b \in \{1,2,3\}$$



Here, s = 127

More Applications

Eaxmple1: Color Transfer



Color Transfer Algorithm

$$l^* = l - \langle l \rangle$$

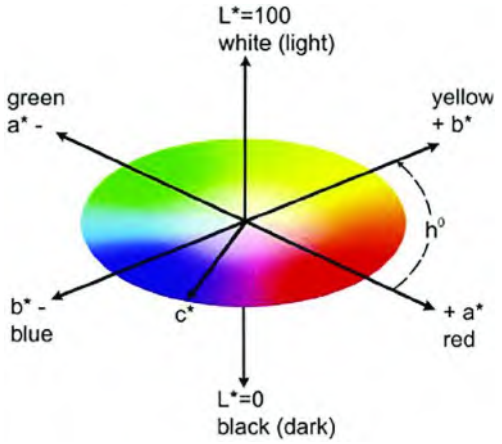
$$\alpha^* = \alpha - \langle \alpha \rangle$$

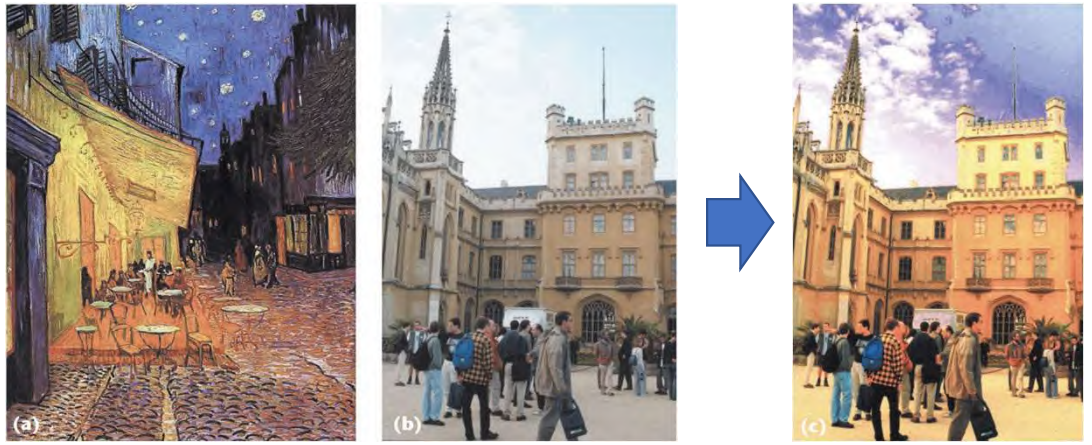
$$\beta^* = \beta - \langle \beta \rangle$$

$$l' = \frac{\sigma_t^l}{\sigma_s^l} l^*$$

$$\alpha' = \frac{\sigma_t^\alpha}{\sigma_s^\alpha} \alpha^*$$

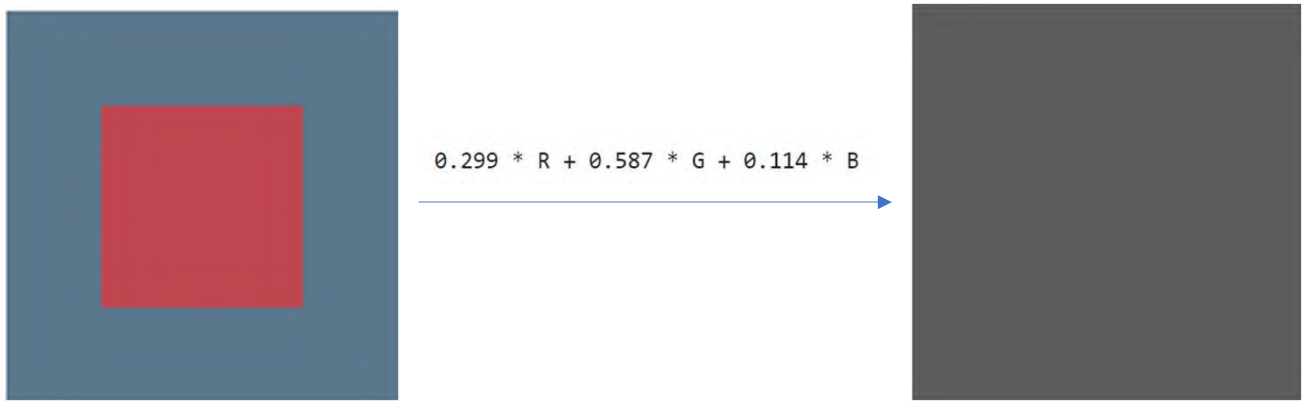
$$\beta' = \frac{\sigma_t^\beta}{\sigma_s^\beta} \beta^*$$





Color Transfer between Images. IEEE CG&A, 2001.

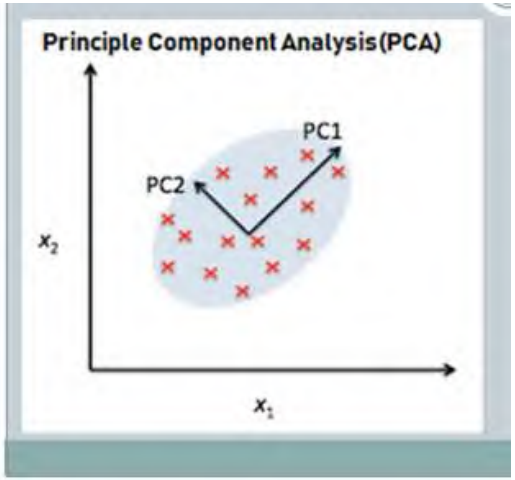
Example2: Color2Gray



Example2: Color2Gray

Color2Gray Algorithm

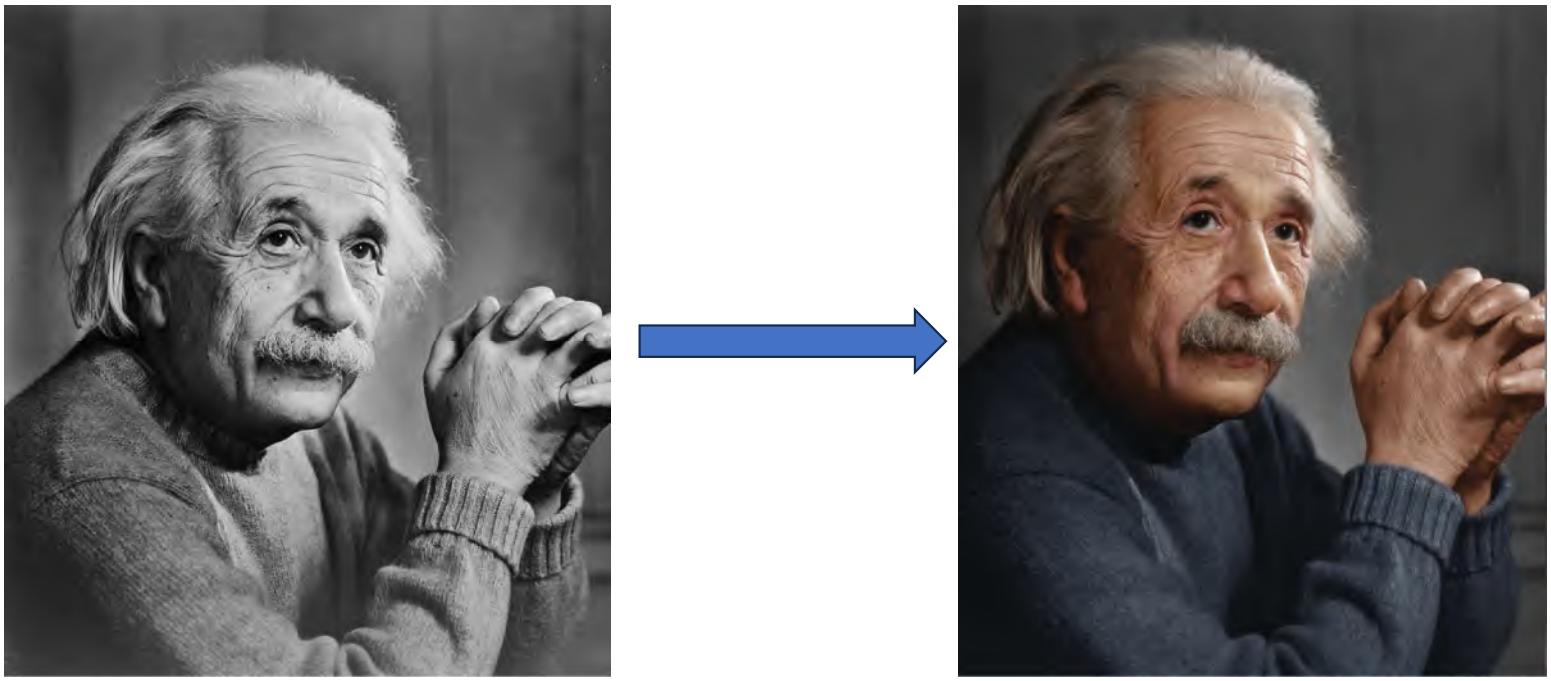
Dim Reduction



$$f(g) = \sum_{(i,j)\in\mathcal{K}} ((g_i - g_j) - \delta_{ij})^2$$

Color2Gray: Salience-Preserving Color Removal. SIGGRAPH 2005.

Example3: Gray2Color (or Colorization)



Inherent Ambiguity

1D->3D, 欠定问题

Inherent Ambiguity

1D->3D, 欠定问题

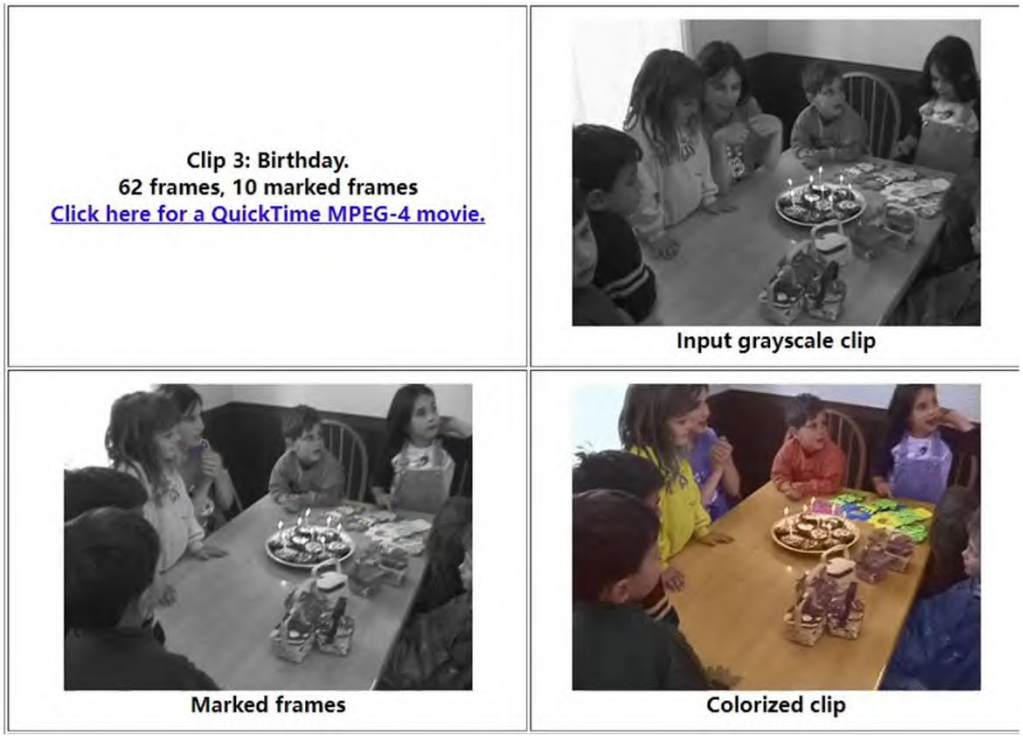
Strand based Colorization

$$J(U) = \sum_{\mathbf{r}} \left(U(\mathbf{r}) - \sum_{\mathbf{s} \in N(\mathbf{r})} w_{\mathbf{rs}} U(\mathbf{s}) \right)^2$$

$$w_{\mathbf{rs}} \propto 1 + \frac{1}{\sigma_{\mathbf{r}}^2} (Y(\mathbf{r}) - \mu_{\mathbf{r}})(Y(\mathbf{s}) - \mu_{\mathbf{r}})$$

Colorization using Optimization. SIGGRAPH 2004.

Strand based Colorization



Colorization using Optimization. SIGGRAPH 2004.

Reference based Colorization

Deep Exemplar-based Colorization. SIGGRAPH 2018.

Example based Colorization

Reference Target Colorized Result

$$sim^i_{T\to R}(p) = d(F^i_T(p), F^i_R(\phi_{T\to R}(p)) \,,$$
$$sim^i_{R\to T}(p) = d(F^i_T(\phi_{R\to T}(\phi_{T\to R}(p))), F^i_R(\phi_{T\to R}(p))) \,. \quad (1)$$

As mentioned in Liao et al. [2017], cosine distance performs better in measuring feature similarity since it is more robust to appearance variances between image pairs. Thus, our similarity metric $d(x, y)$ between two deep features is defined as their cosine similarity:

$$d(x, y) = \frac{x^T y}{|x||y|} \,. \quad (2)$$

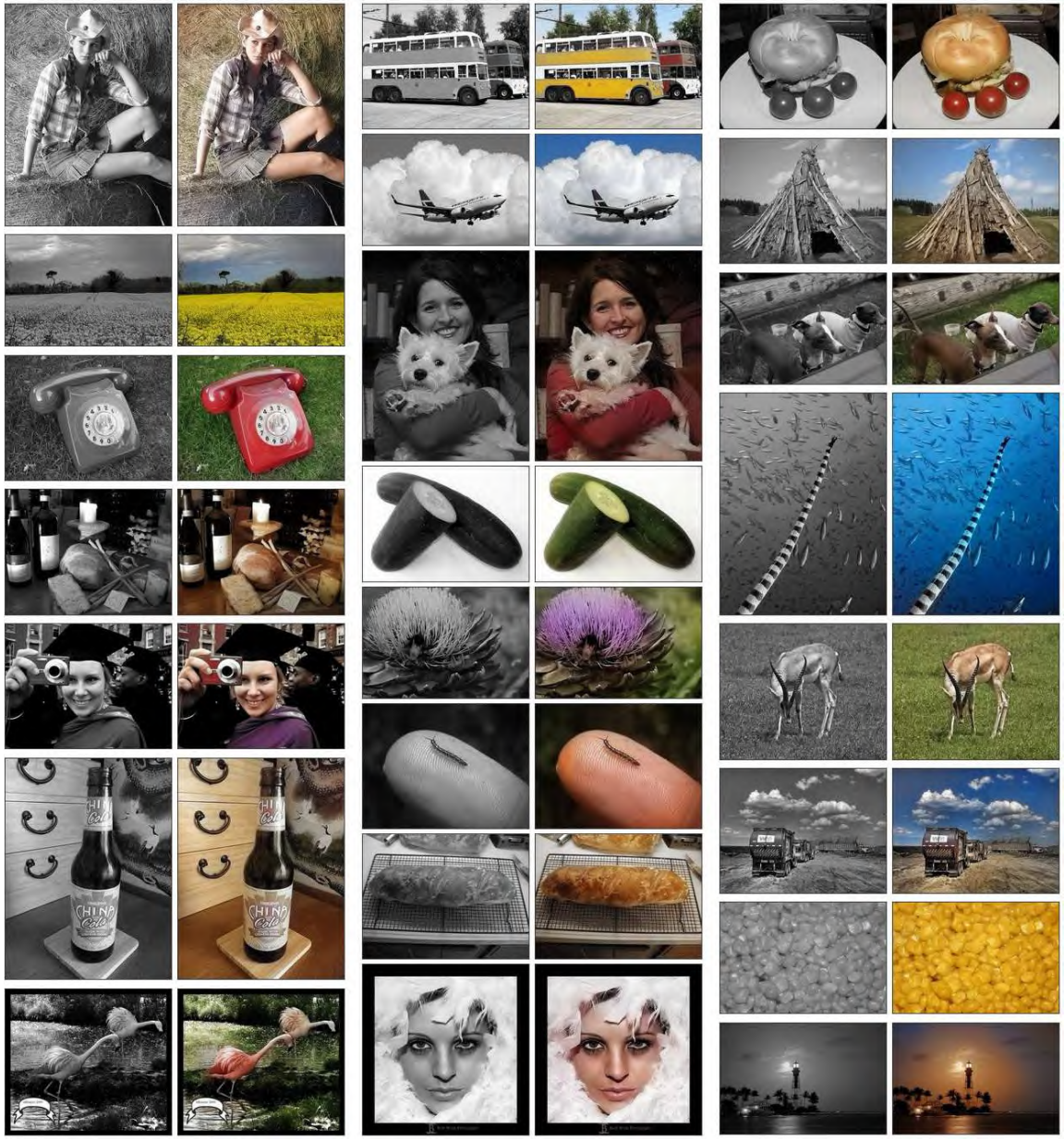
Deep Exemplar-based Colorization. SIGGRAPH 2018.

Reference free Colorization

Colorful Image Colorization. ECCV 2016.

Data Driven

Color2gray is easy



Reference free Colorization

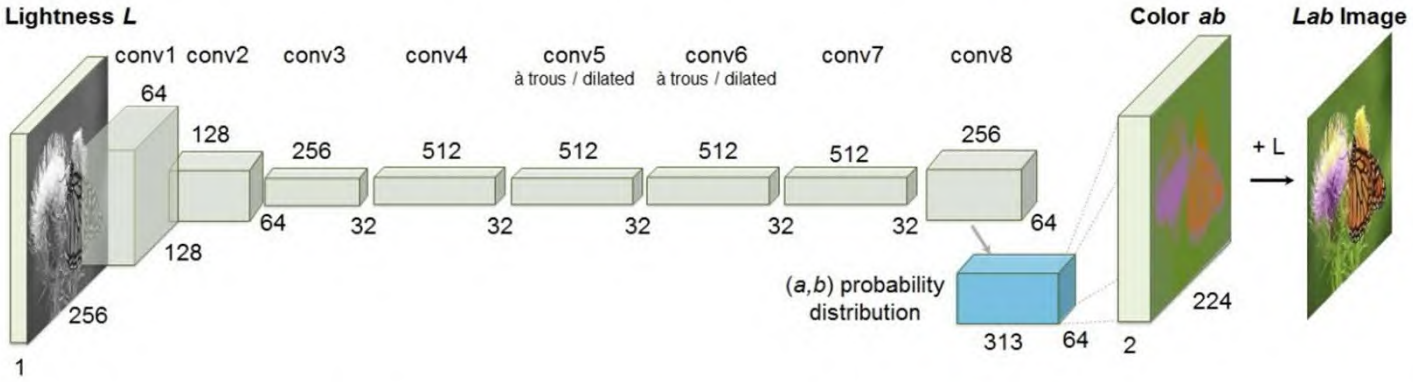


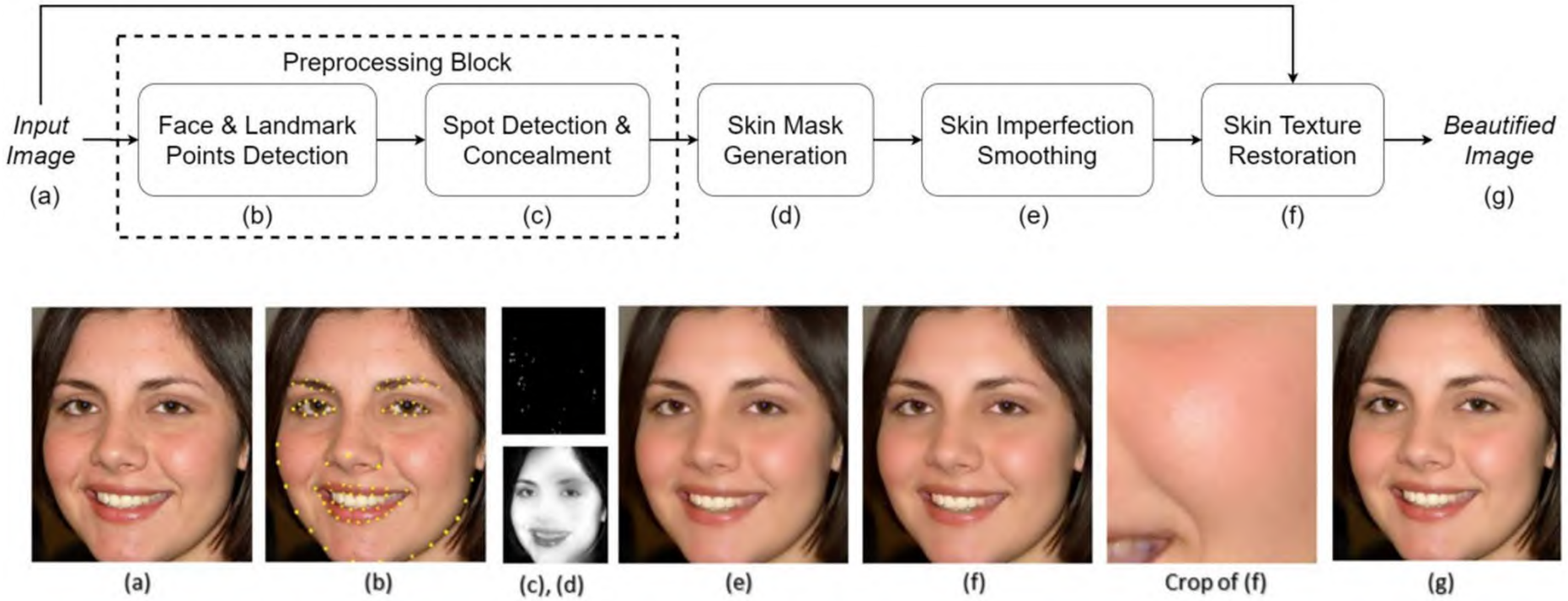
Fig. 2. Our network architecture. Each `conv` layer refers to a block of 2 or 3 repeated `conv` and `ReLU` layers, followed by a `BatchNorm` [30] layer. The net has no `pool` layers. All changes in resolution are achieved through spatial downsampling or upsampling between `conv` blocks.

$$\mathrm{L}_2(\widehat{\mathbf{Y}}, \mathbf{Y}) = \frac{1}{2}\sum_{h,w} \|\mathbf{Y}_{h,w} - \widehat{\mathbf{Y}}_{h,w}\|_2^2$$

Colorful Image Colorization. ECCV 2016.

Example4: Face Retouch

Face Retouch Algorithm



Face Beautification via Dynamic Skin Smoothing, Guided Feathering, and Texture Restoration. CVPRW 2020.

Face Retouch Algorithm

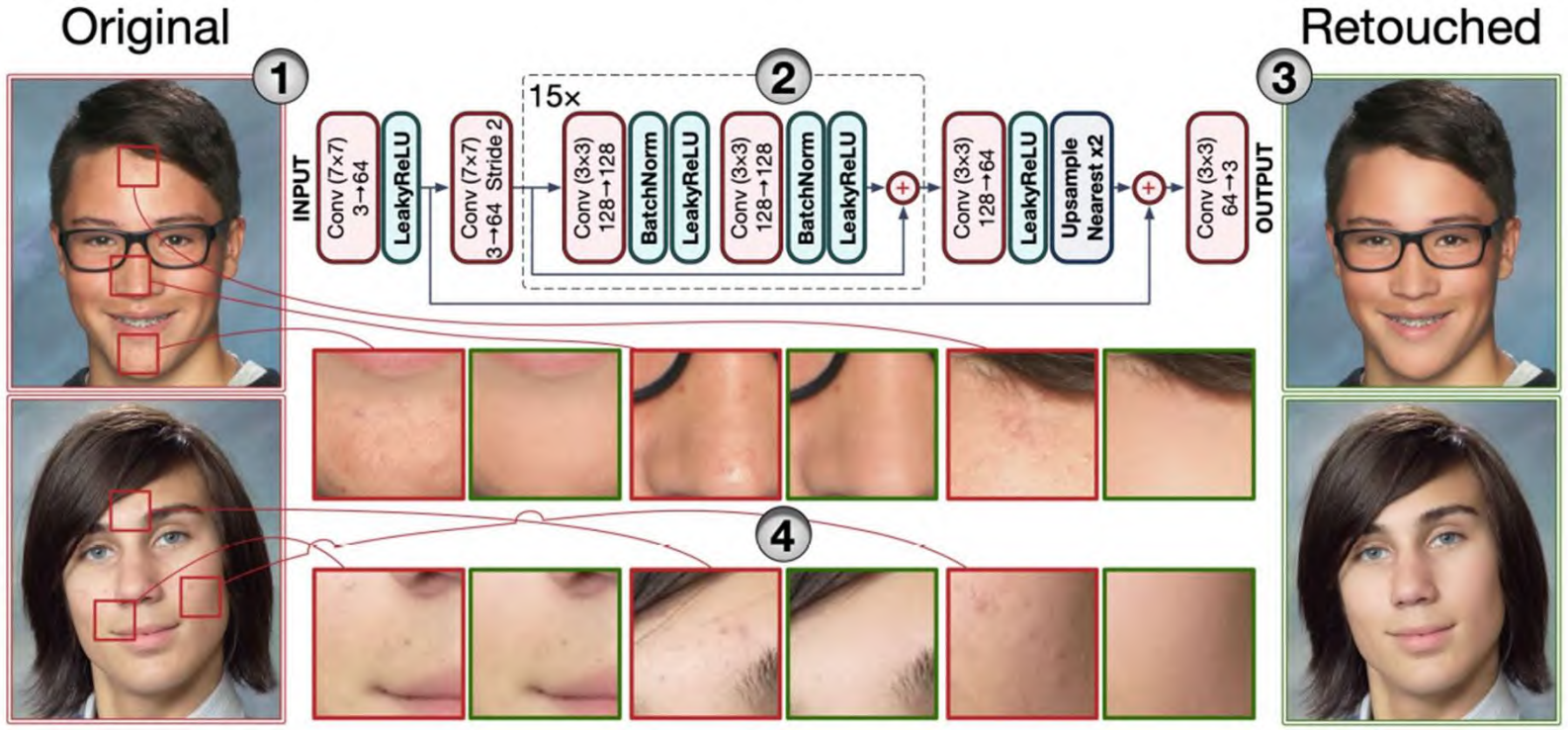
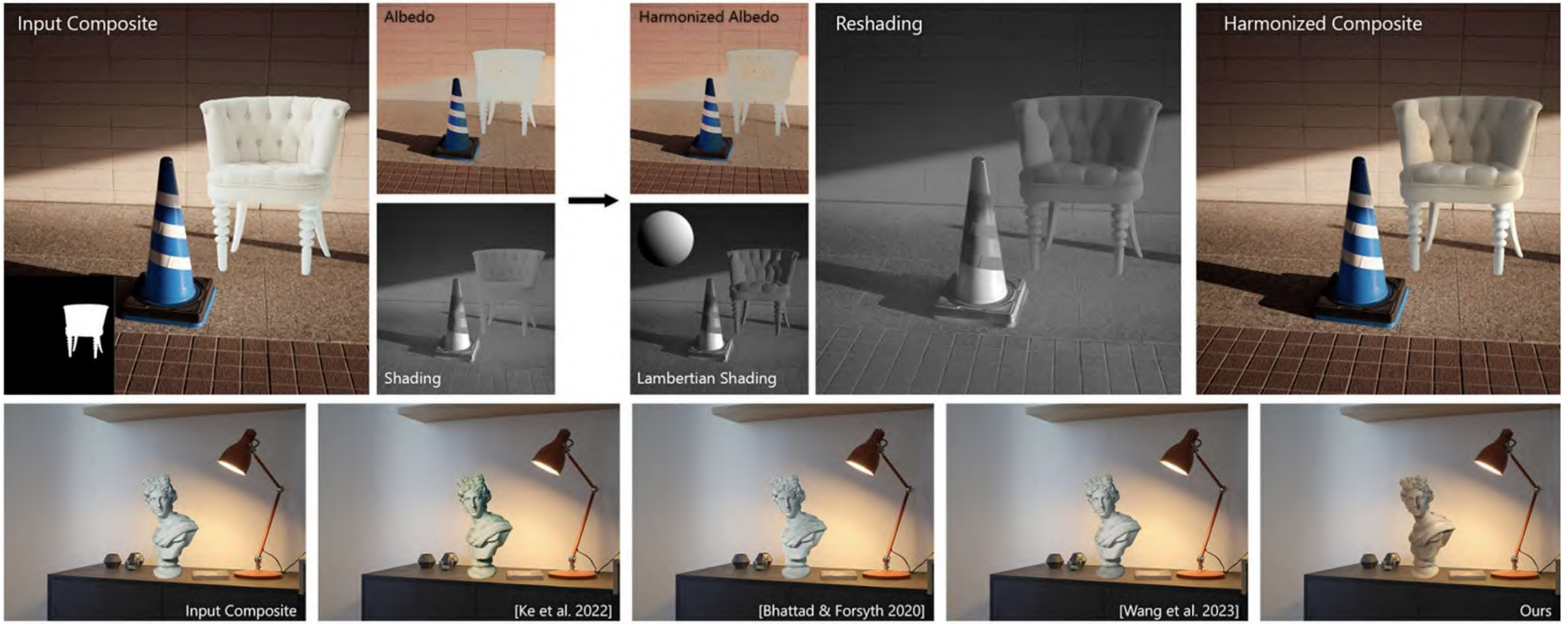


Figure 1: Overview of AutoRetouch: (1) input portraits cropped to the head, (2) the images pass through the fully convolutional architecture of AutoRetouch, (3) the output portraits are retouched while preserving fine textures. (4) Sample before-after patches scaled up. We empirically show that our final models have desirable generalization properties.

AutoRetouch: Automatic Professional Face Retouching. WACV 2021.

Example5: Image Composition



Intrinsic Harmonization for Illumination-Aware Compositing. SIGGRAPH Asia 2023.

A Simple Image Composition

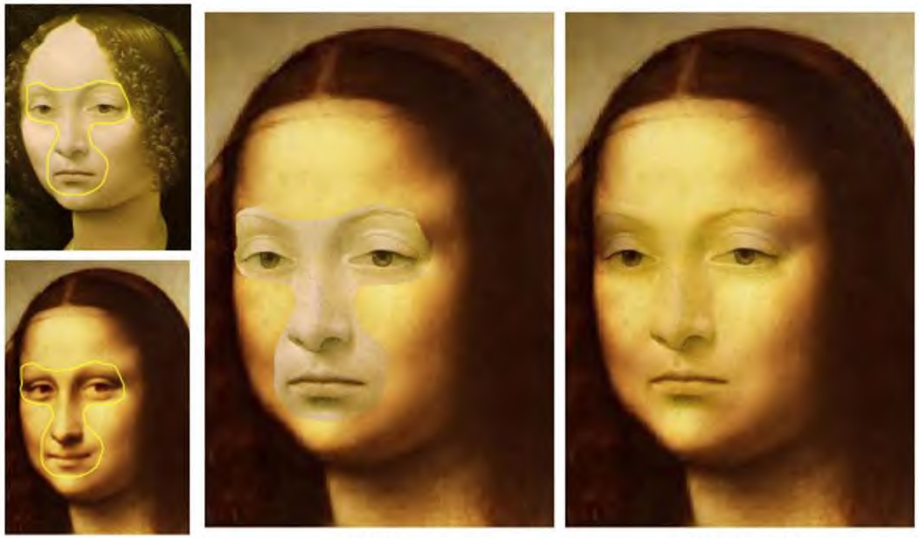
sources/destinations

cloning

seamless cloning

A Simple Image Composition

A Simple Image Composition



source/destination cloning seamless cloning

A *guidance field* is a vector field $\mathbf{v}$ used in an extended version of the minimization problem (1) above:

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{3}$$

whose solution is the unique solution of the following Poisson equation with Dirichlet boundary conditions:

$$\Delta f = \text{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{4}$$

where $\text{div}\mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $\mathbf{v} = (u, v)$. This is the

Poisson Image Editing. TOG 2003.

Example Review

$$J(U) = \sum_{\mathbf{r}} \left(U(\mathbf{r}) - \sum_{\mathbf{s} \in N(\mathbf{r})} w_{\mathbf{rs}} U(\mathbf{s}) \right)^2$$

$$w_{\mathbf{rs}} \propto 1 + \frac{1}{\sigma_{\mathbf{r}}^2} (Y(\mathbf{r}) - \mu_{\mathbf{r}})(Y(\mathbf{s}) - \mu_{\mathbf{r}})$$

source/destination cloning seamless cloning

A *guidance field* is a vector field $\mathbf{v}$ used in an extended version of the minimization problem (1) above:

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{3}$$

whose solution is the unique solution of the following Poisson equation with Dirichlet boundary conditions:

$$\Delta f = \text{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{4}$$

where $\text{div}\mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $\mathbf{v} = (u, v)$. This is the

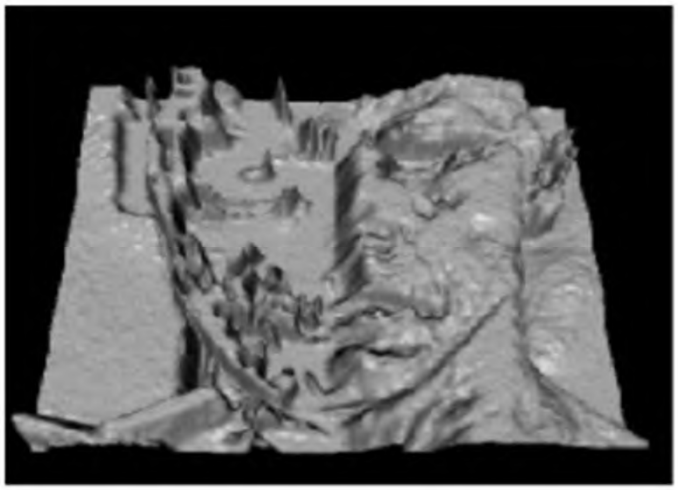
Shared transformation for all pixels

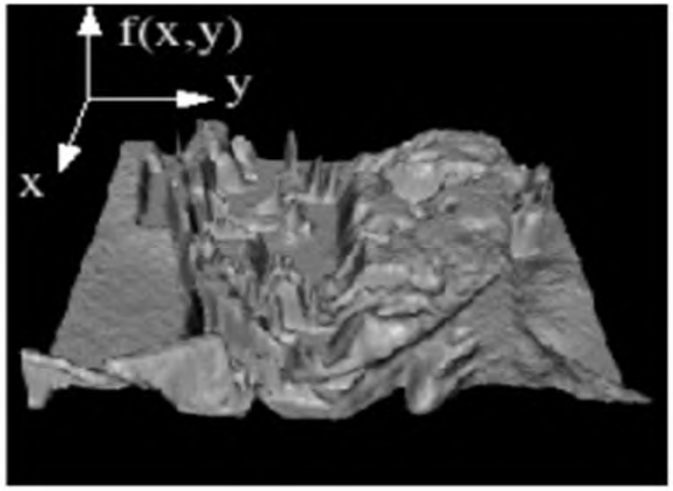
- In a digital image, point = pixel.
- Point processing transforms a pixel's value as function of its value alone;
- it does not depend on the values of the pixel's neighbors.

Point Processing of Images
类比：三维几何点云的刚性变换

Neighboring / Coordinate Dependent Transform

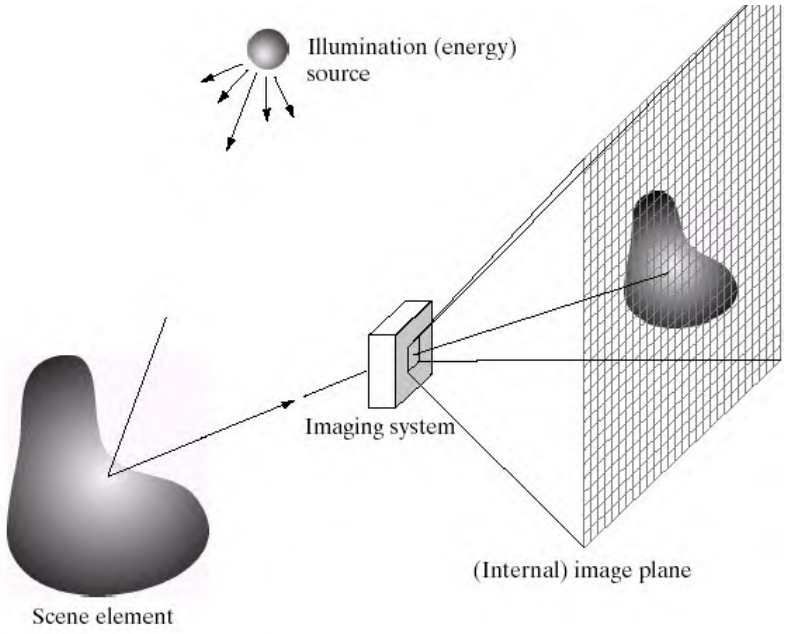
Image as Continuous function of Coordinates



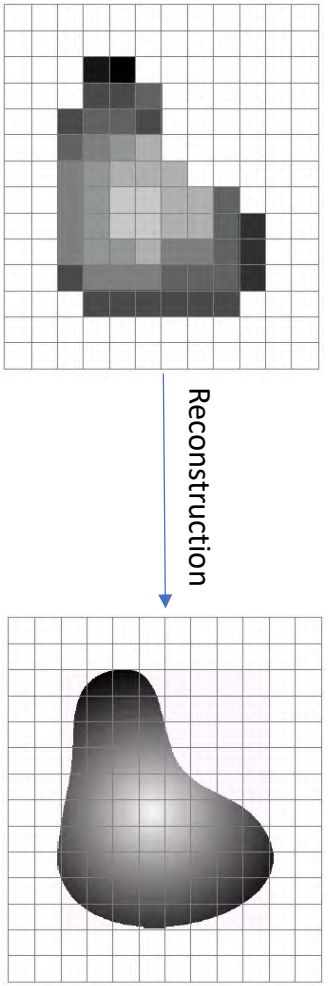


Visualization of single channel grayscale image $f(x, y)$

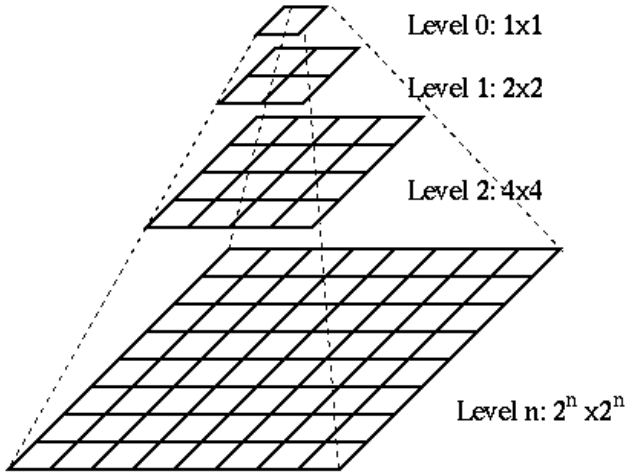
Sampling (Non-continuous) and Reconstruction

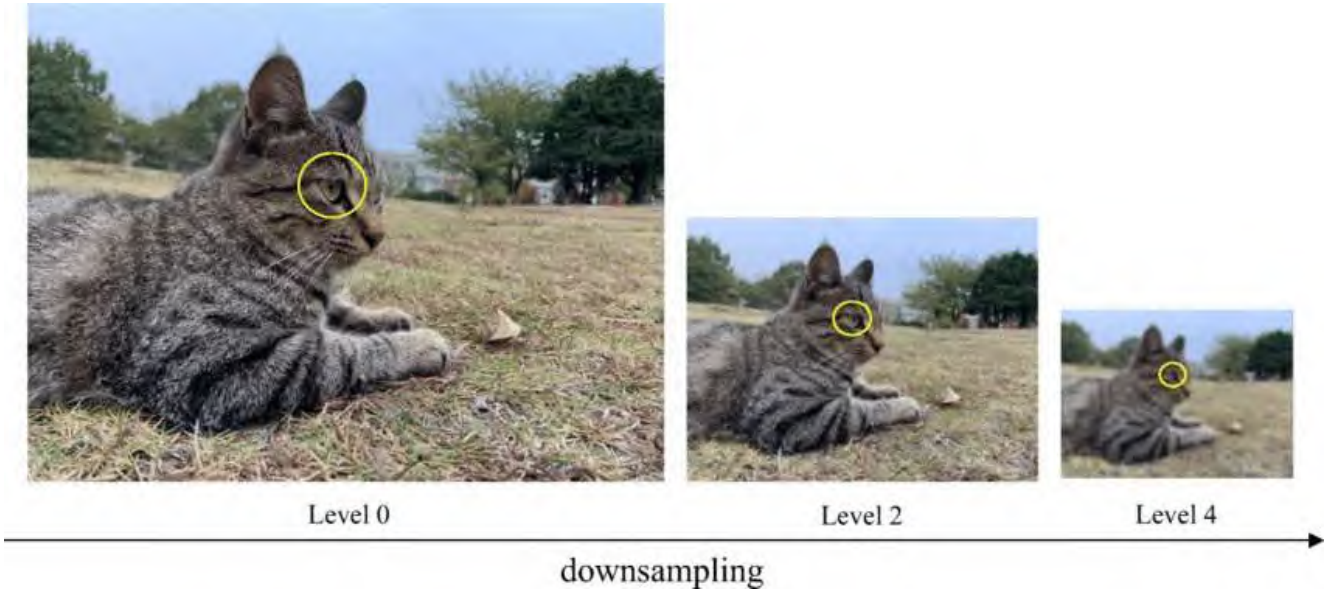


Sampling

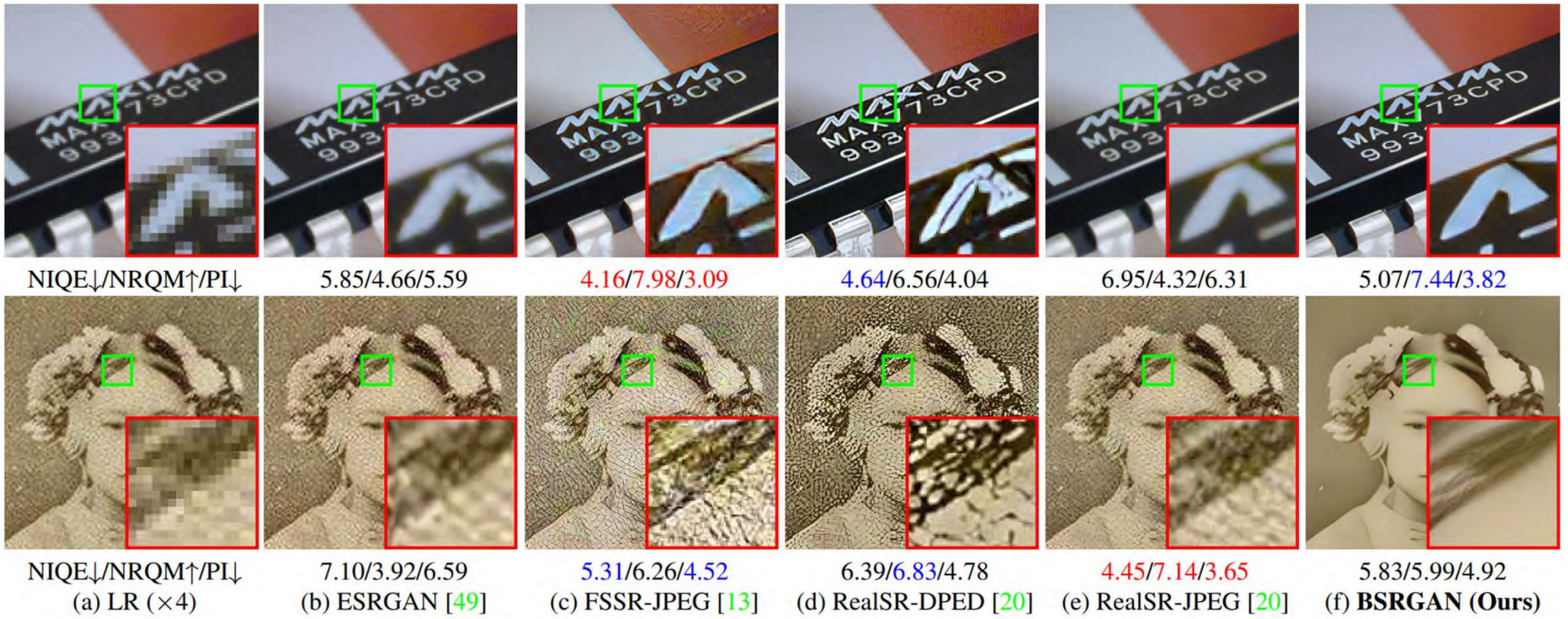


Downsampling





Upsampling (Super-resolution)

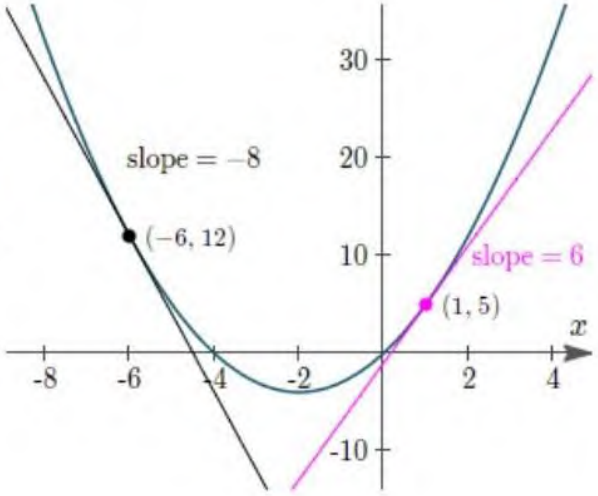


Discrete Operations

DIFFERENTIAL CALCULUS

Gradient of any point on CURVE

$$\frac{dy}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$



T. Prashant

Image Gradient

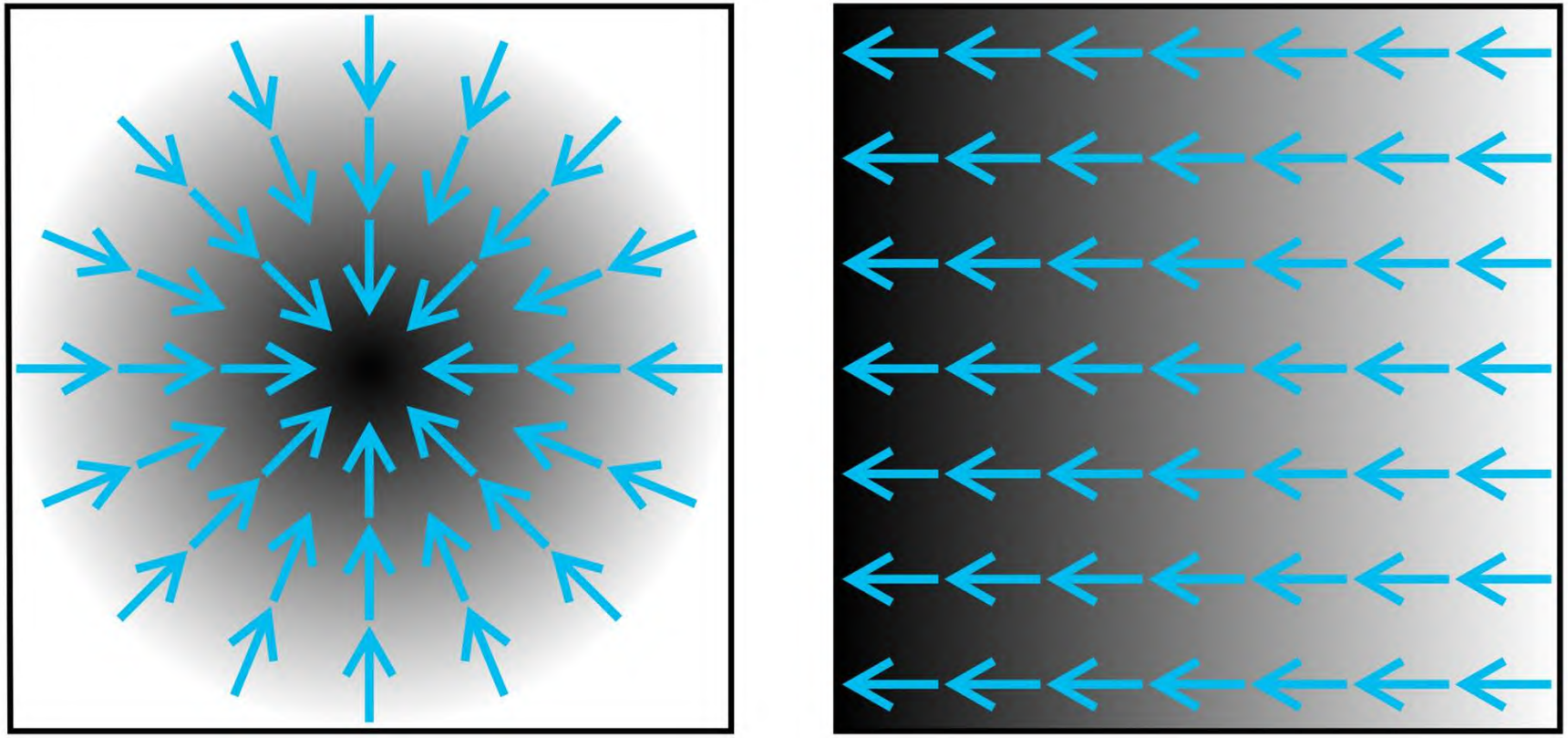
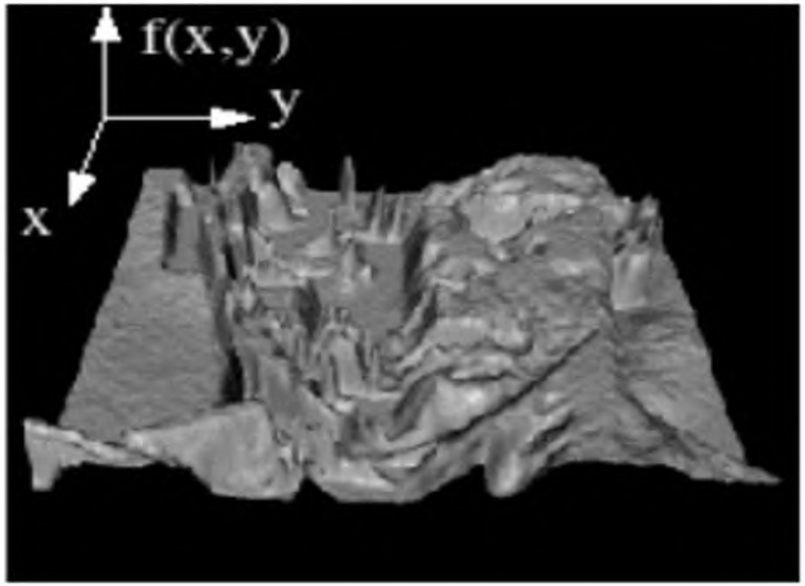


Image Gradient

$$\nabla f = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$



$$g_x = f(x+1, y) - f(x, y)$$

$$g_y = f(x, y+1) - f(x, y)$$

or

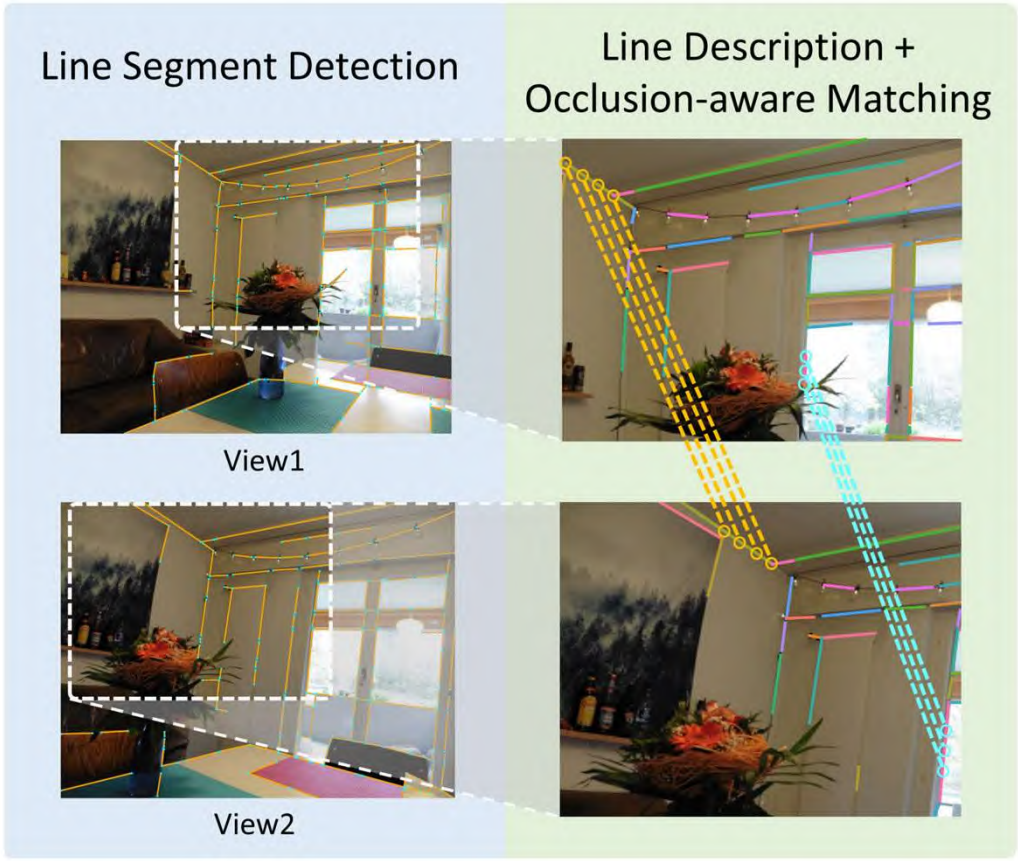
$$g_x = f(x+1, y) - f(x-1, y)$$

$$g_y = f(x, y+1) - f(x, y-1)$$

Image Gradient Visualization

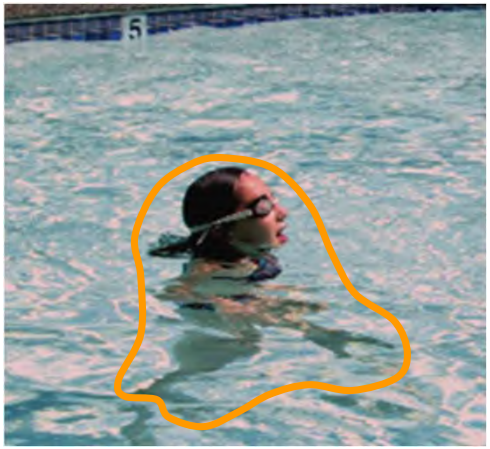
X-gradient Y-gradient

Application: Image -> Edge (Sketch)



SOLD2: Self-supervised Occlusion-aware Line Description and Detection. CVPR 2021.

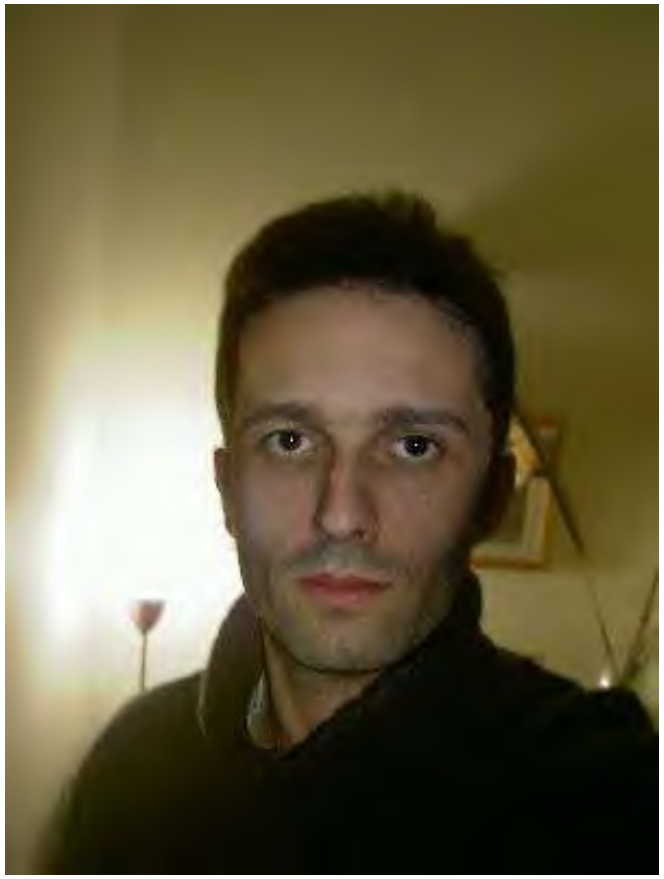
Application: Image Composition



Application: Texture Exchange

Application: Wrinkle Removal

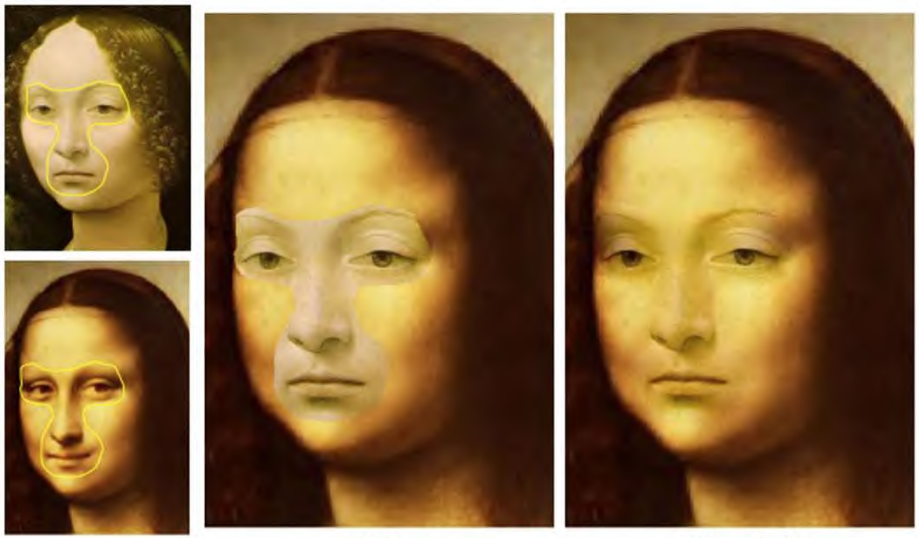
Application: Lighting Interpolation



Application: Color Transfer

Application: Color Transfer

How to efficiently implement



source/destination cloning seamless cloning

A *guidance field* is a vector field $\mathbf{v}$ used in an extended version of the minimization problem (1) above:

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{3}$$

whose solution is the unique solution of the following Poisson equation with Dirichlet boundary conditions:

$$\Delta f = \text{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \tag{4}$$

where $\text{div}\mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $\mathbf{v} = (u, v)$. This is the

O(N*N)

Filtering / Convolution

$$(f * h)(x, y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} f(x - i, y - j) \times h(i, j)$$

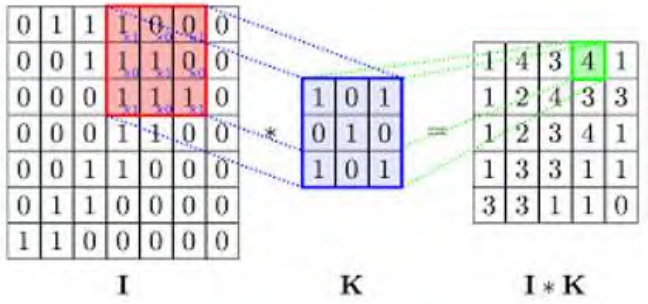


Figure 1: Convolution Algorithm.

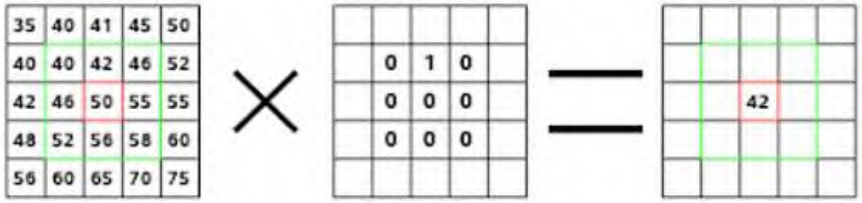
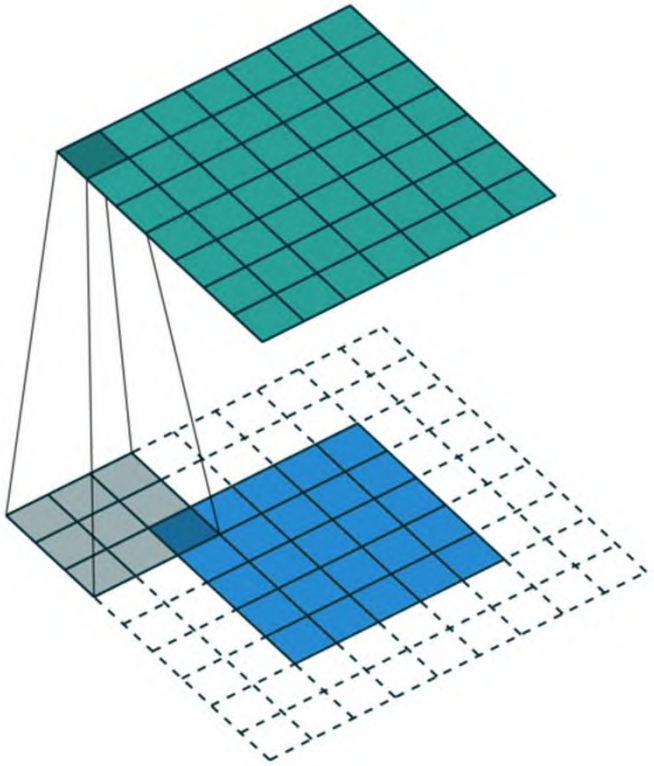


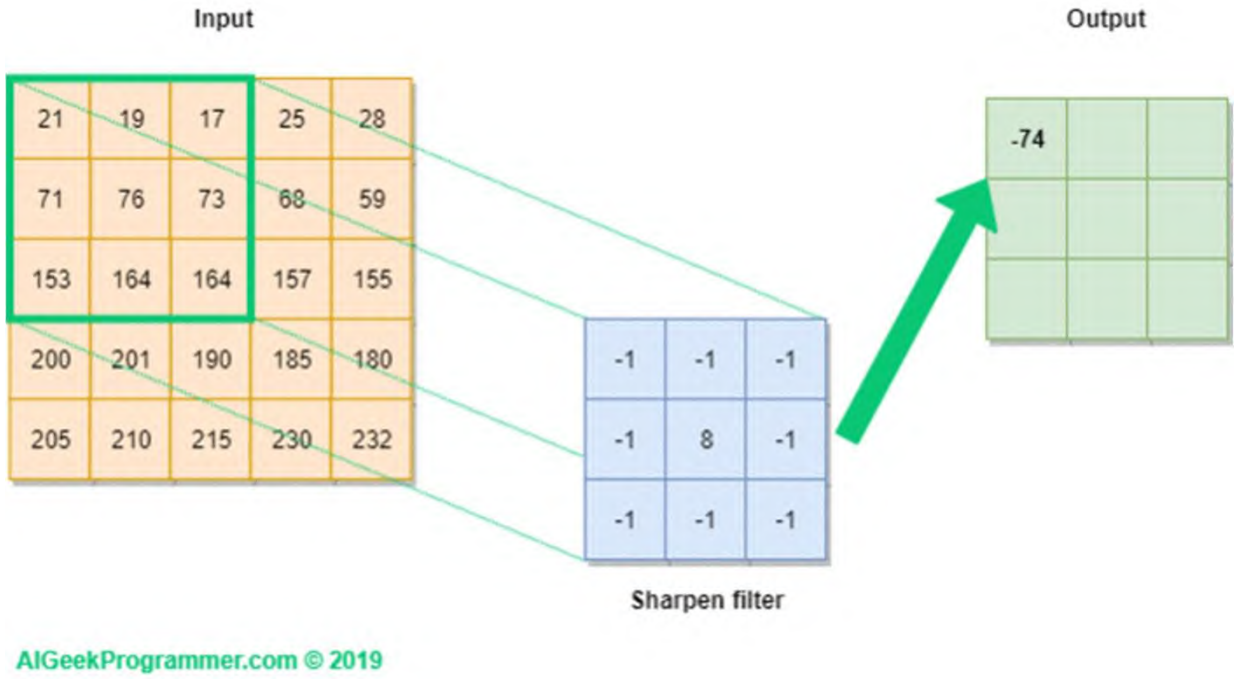
Figure 2: Example of convolution

Example of adjusted convolution mask:

Box blur: $\frac{1}{9}\begin{bmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{bmatrix}$ => at the edge $\frac{1}{4}\begin{bmatrix}1 & 1\\ 1 & 1\end{bmatrix}$

Image Filtering / Convolution





2D Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood

- Achieve smoothing effect (remove sharp features)

$$h[\cdot,\cdot]$$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

2D Box Filter

$$h[\cdot,\cdot]$$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Gaussian Blur

$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

0.075	0.124	0.075
0.124	0.204	0.124
0.075	0.124	0.075

Sharpen

$$\begin{pmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{pmatrix}$$

XY-Gradient

X – Direction Kernel

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

Y – Direction Kernel

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

Laplacian

0	−1	0
−1	4	−1
0	−1	0

−1	−1	−1
−1	8	−1
−1	−1	−1

$$\begin{aligned} & \big(f(x+1,y)-f(x,y)\big)-\big(f(x,y)-f(x-1,y)\big) \\ + \quad & \big(f(x,y+1)-f(x,y)\big)-\big(f(x,y)-f(x,y-1)\big) \end{aligned}$$

Mixture of Filters

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in

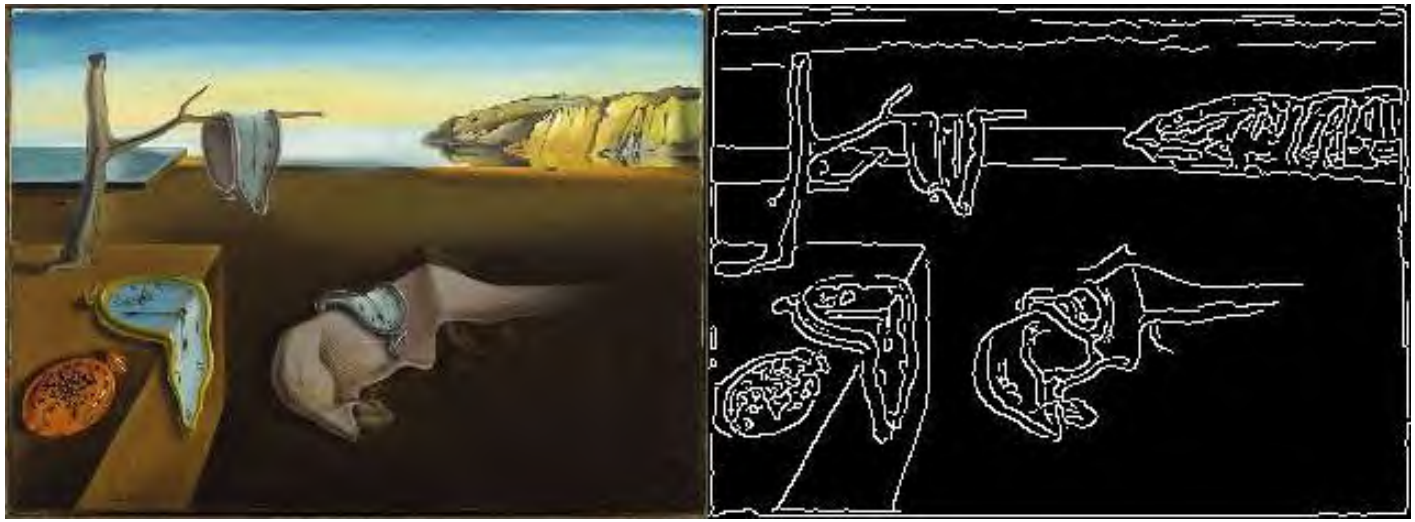
1. It is a multi-stage algorithm and we will go through each stages.
2. **Noise Reduction**

   Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter. We have already seen this in previous chapters.

3. **Finding Intensity Gradient of the Image**

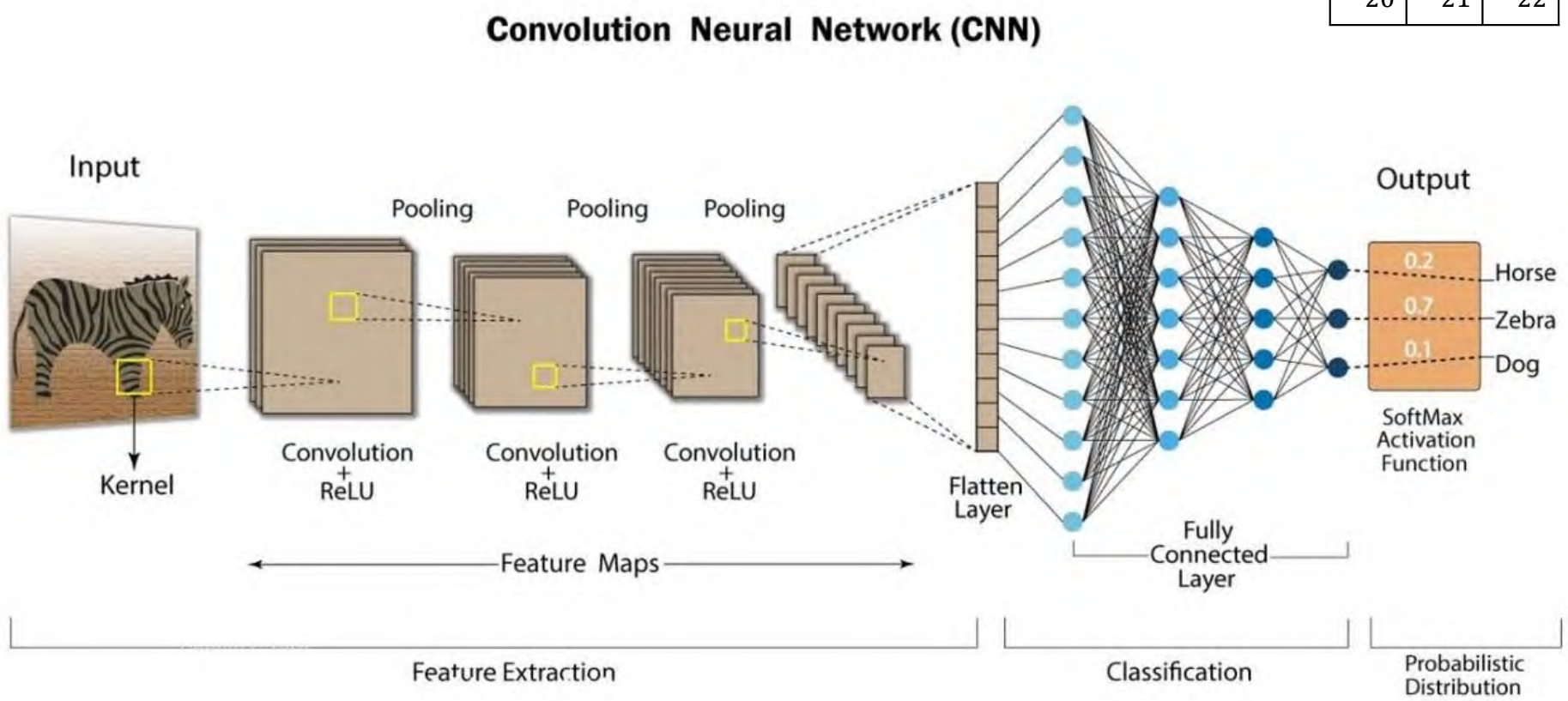
   Smoothened image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (G_x) and vertical direction (G_y). From these two images, we can find edge gradient and direction for each pixel as follows:

$$Edge\_Gradient\,(G) = \sqrt{G_x^2 + G_y^2} Angle\,(\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$



Mixture of Learnable Filters

h_{00}	h_{01}	h_{02}
h_{10}	h_{11}	h_{12}
h_{20}	h_{21}	h_{22}



Efficiently Conv—With Fourier Transform

$$F(k,l) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} f(i,j) e^{-2\pi \mathbf{i} \left(\frac{ki}{N} + \frac{lj}{N} \right)}$$

Fast Fourier transform

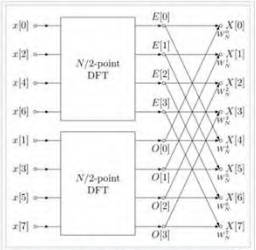
29 languages

Article Talk Read Edit View history Tools

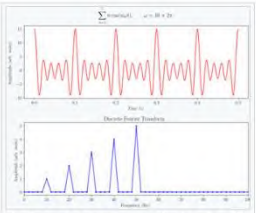
From Wikipedia, the free encyclopedia

"FFT" redirects here. For other uses, see FFT (disambiguation).

A **fast Fourier transform** (**FFT**) is an algorithm that computes the Discrete Fourier Transform (DFT) of a sequence, or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa. The DFT is obtained by decomposing a sequence of values into components of different frequencies.[1] This operation is useful in many fields, but computing it directly from the definition is often too slow to be practical. An FFT rapidly computes such transformations by factorizing the DFT matrix into a product of sparse (mostly zero) factors.[2] As a result, it manages to reduce the complexity of computing the DFT from $O(n^2)$, which arises if one simply applies the definition of DFT, to $O(n \log n)$, where n is the data size. The difference in speed can be enormous, especially for long data sets where n may be in the thousands or millions. In the presence of round-off error, many FFT algorithms are much more accurate than evaluating the DFT definition directly or indirectly. There are many different FFT algorithms based on a wide range of published theories, from simple complex-number arithmetic to group theory and number theory.



An example FFT algorithm structure, using a decomposition into half-size FFTs



A discrete Fourier analysis of a sum of cosine waves at 10, 20

Frequency Magnitude
Center at (0,0)

Convolution Theorem

In mathematics, the **convolution theorem** states that under suitable conditions the Fourier transform of a convolution of two functions (or signals) is the product of their Fourier transforms. More generally, convolution in one domain (e.g., time domain) equals point-wise multiplication in the other domain (e.g., frequency domain). Other versions of the convolution theorem are applicable to various Fourier-related transforms.

Convolution theorem

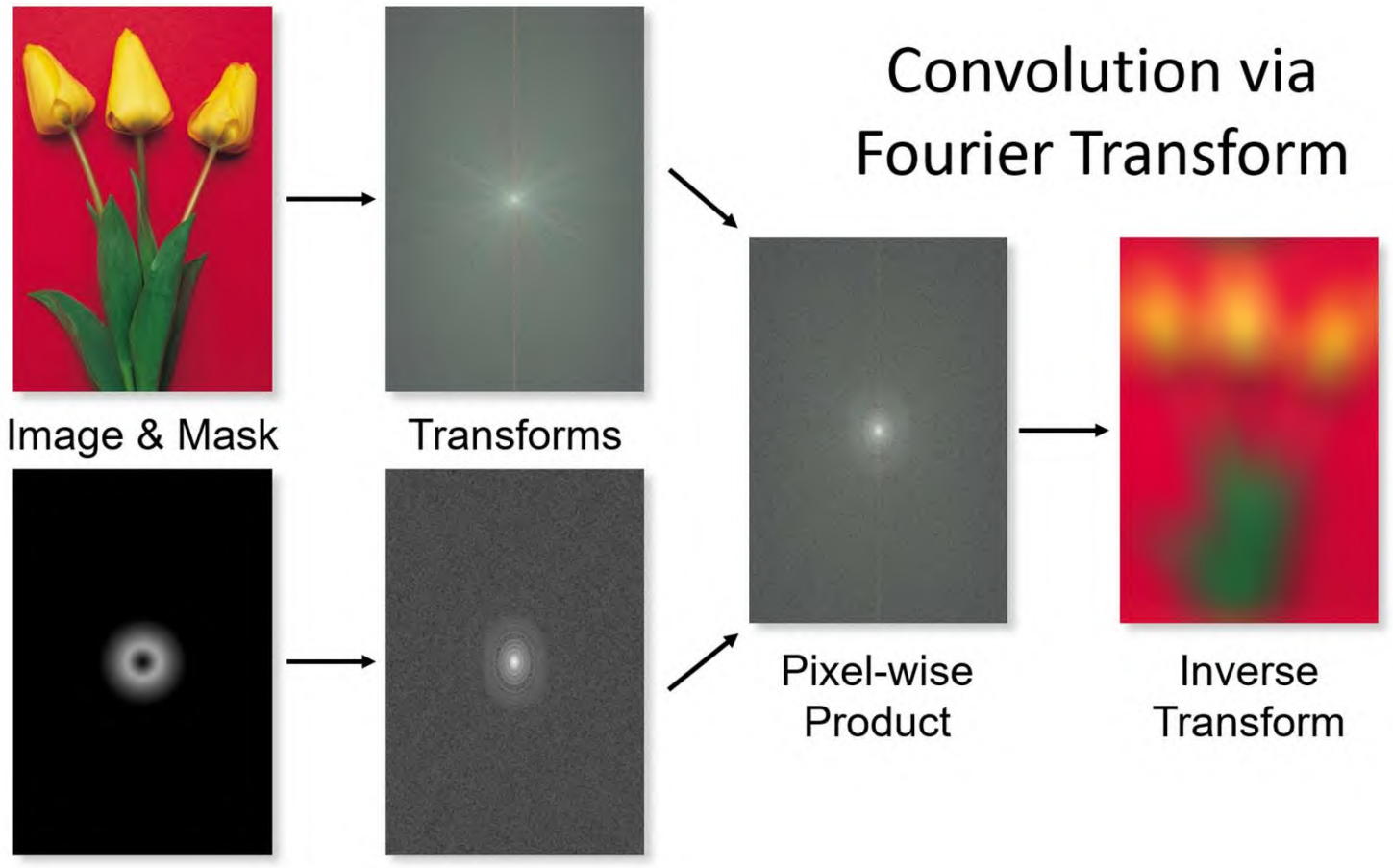
$$r(x) = \{u * v\}(x) = \mathcal{F}^{-1}\{U \cdot V\}. \quad \textbf{(Eq.1b)}$$

$$F(k,l) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1} f(i,j) e^{-2\pi \mathbf{i}\left(\frac{ki}{N}+\frac{lj}{N}\right)}$$

$$f(i,j) = \frac{1}{N^2}\sum_{k=0}^{N-1}\sum_{l=0}^{N-1} F(k,l) e^{2\pi \mathbf{i}\left(\frac{ki}{N}+\frac{lj}{N}\right)}$$

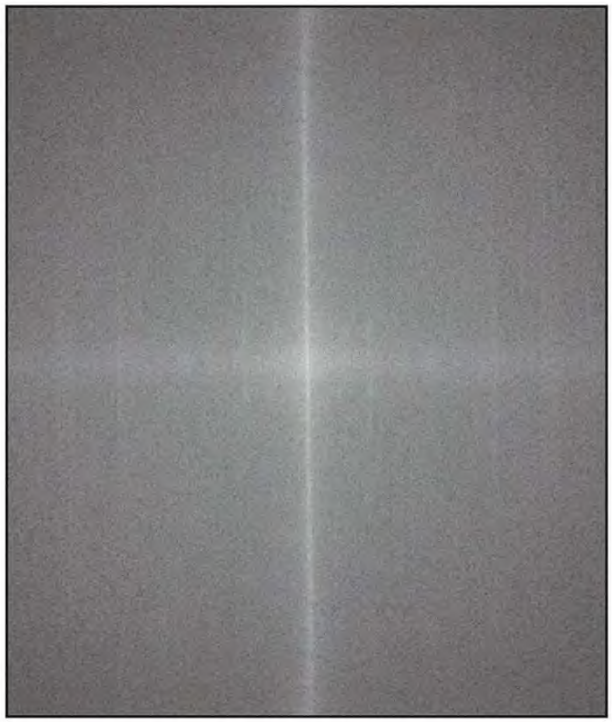
$$f(i,j) = \sum_{k=0}^{N-1}\sum_{l=0}^{N-1}\left(\sum_{m=0}^{N-1}\sum_{n=0}^{N-1} f(m,n) e^{-2\pi \mathbf{i}\left(\frac{km}{N}+\frac{ln}{N}\right)}\right) e^{2\pi \mathbf{i}\left(\frac{ki}{N}+\frac{lj}{N}\right)}$$

Fast Conv Computation



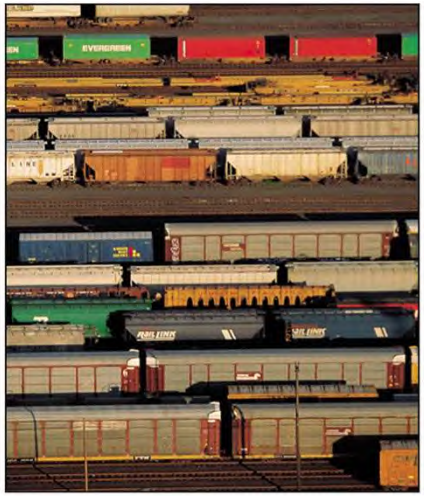
So called filtering

original image



power spectrum

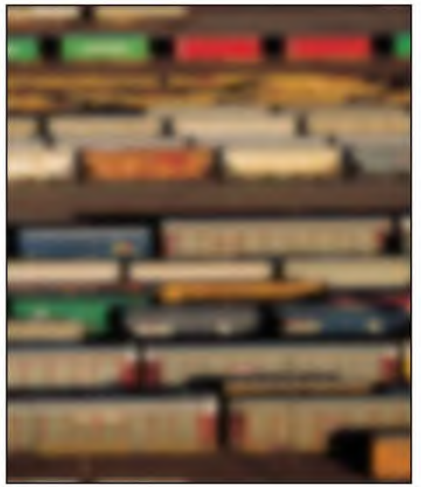
So called filtering



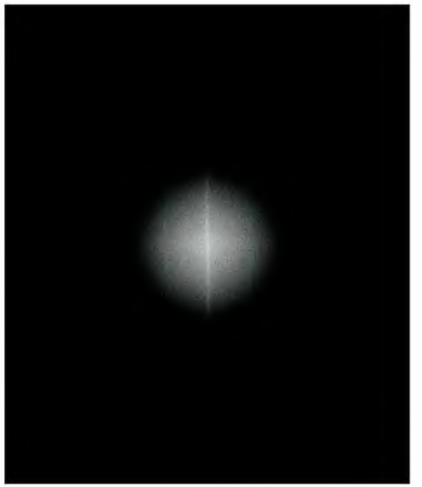
original image

power spectrum

Low-Pass

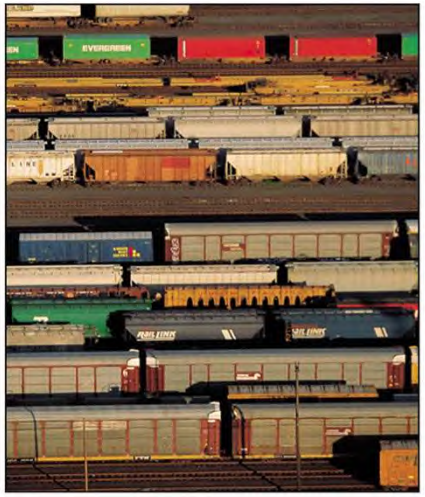


blurred image



power spectrum

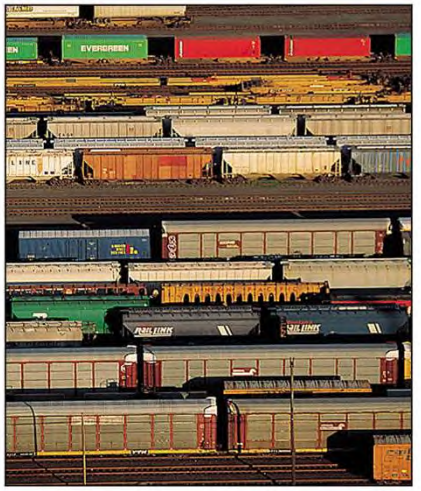
So called filtering



original image

power spectrum

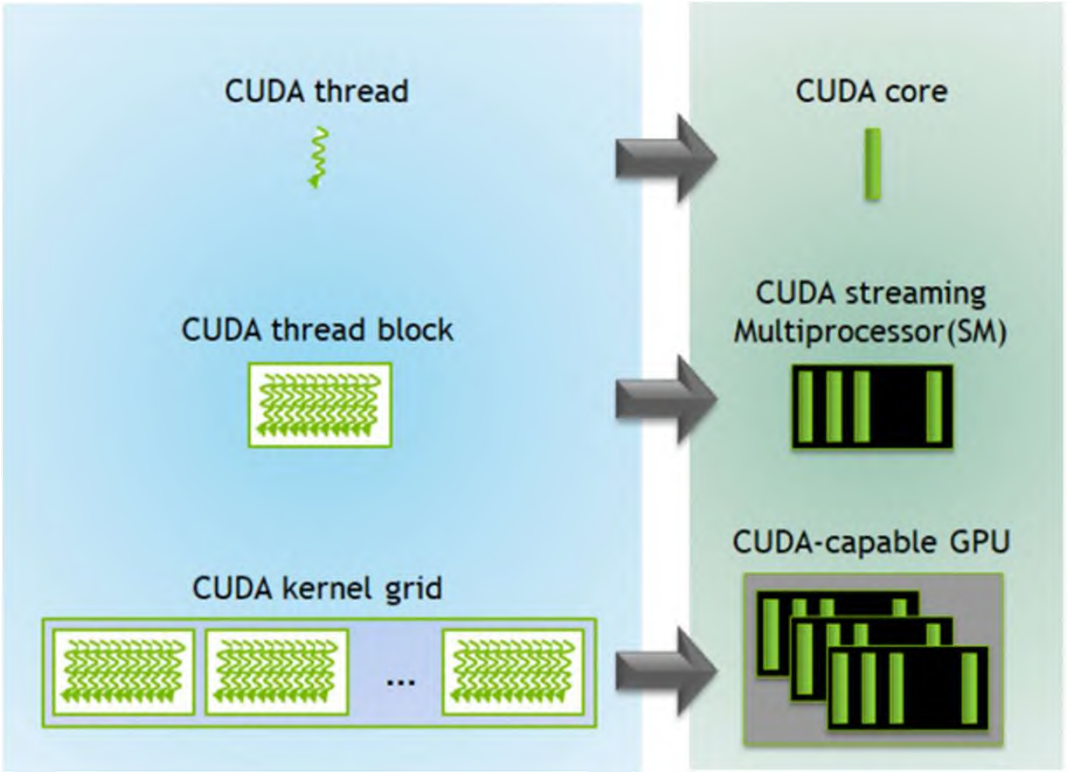
More High

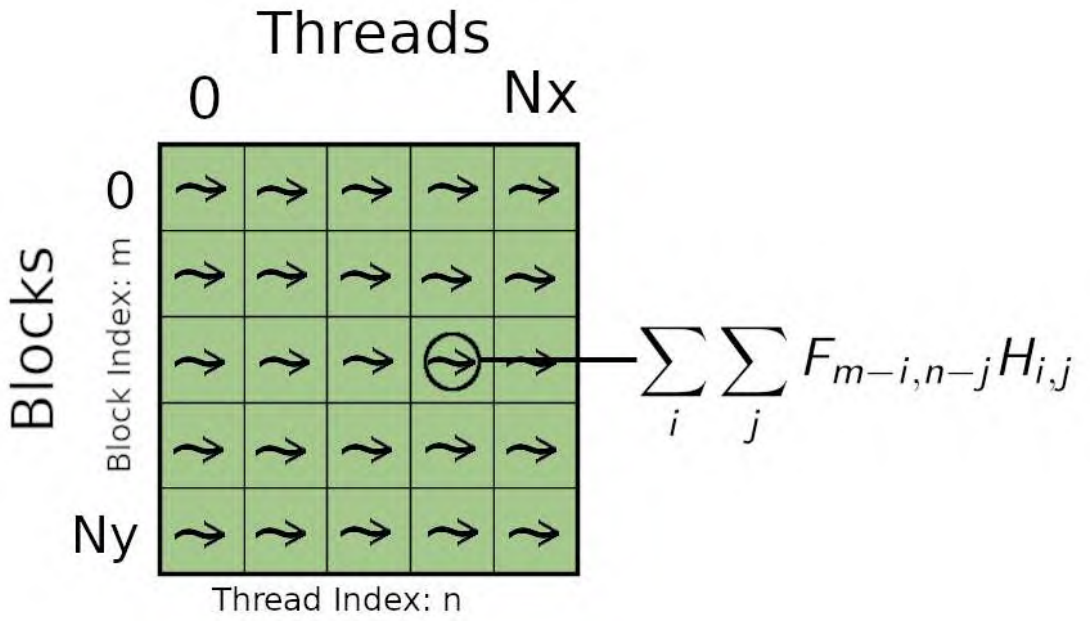


sharpened image

power spectrum

But not used in Modern GPU





Easy to Use

```python
import cv2
import numpy as np

image = cv2.imread('test.jpg')

# Print error message if image is null
if image is None:
    print('Could not read image')

# Apply identity kernel
kernel1 = np.array([[0, 0, 0],
                    [0, 1, 0],
                    [0, 0, 0]])

identity = cv2.filter2D(src=image, ddepth=-1, kernel=kernel1)

cv2.imshow('Original', image)
cv2.imshow('Identity', identity)

cv2.waitKey()
cv2.imwrite('identity.jpg', identity)
cv2.destroyAllWindows()

# Apply blurring kernel
kernel2 = np.ones((5, 5), np.float32) / 25
img = cv2.filter2D(src=image, ddepth=-1, kernel=kernel2)

cv2.imshow('Original', image)
cv2.imshow('Kernel Blur', img)

cv2.waitKey()
cv2.imwrite('blur_kernel.jpg', img)
cv2.destroyAllWindows()
```

torch.nn.functional.conv2d

torch.nn.functional.conv2d(*input*, *weight*, *bias=None*, *stride=1*, *padding=0*, *dilation=1*, *groups=1*) → Tensor

Applies a 2D convolution over an input image composed of several input planes.

This operator supports TensorFloat32.

See `Conv2d` for details and output shape.

NOTE

In some circumstances when given tensors on a CUDA device and using CuDNN, this operator may select a nondeterministic algorithm to increase performance. If this is undesirable, you can try to make the operation deterministic (potentially at a performance cost) by setting `torch.backends.cudnn.deterministic = True`. See Reproducibility for more information.

NOTE

This operator supports complex data types i.e. `complex32, complex64, complex128.`

Parameters

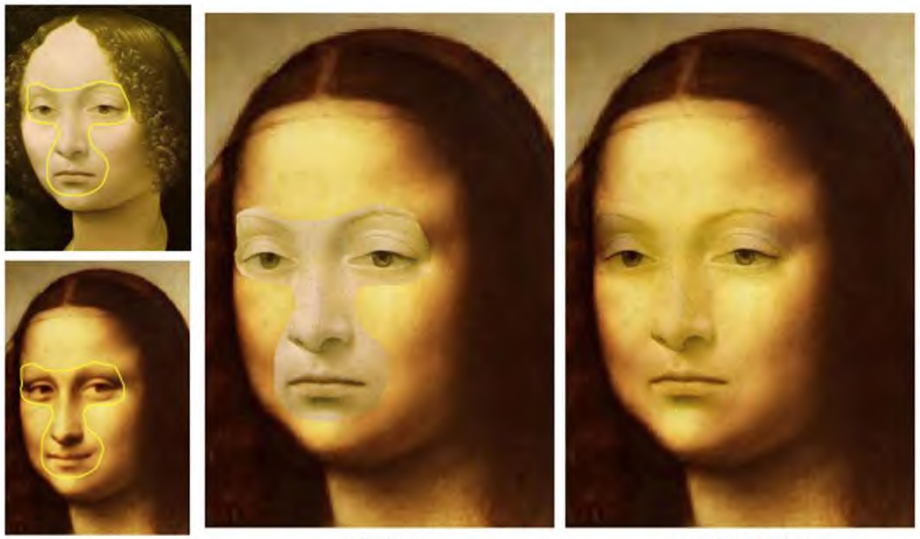
- **input** – input tensor of shape $(\text{minibatch}, \text{in\_channels}, iH, iW)$
- **weight** – filters of shape $(\text{out\_channels}, \frac{\text{in\_channels}}{\text{groups}}, kH, kW)$
- **bias** – optional bias tensor of shape (out_channels). Default: `None`
- **stride** – the stride of the convolving kernel. Can be a single number or a tuple (*sH, sW*). Default: 1
- **padding** –
  implicit paddings on both sides of the input. Can be a string {'valid', 'same'}, single number or a tuple (*padH, padW*). Default: 0 `padding='valid'` is the same as no padding. `padding='same'` pads the input so the output has the same shape as the input. However, this mode doesn't support any stride values other than 1.

  WARNING

  For `padding='same'`, if the `weight` is even-length and `dilation` is odd in any dimension, a full `pad()` operation may be needed internally. Lowering performance.

- **dilation** – the spacing between kernel elements. Can be a single number or a tuple (*dH, dW*). Default: 1
- **groups** – split input into groups, both in_channels and out_channels should be divisible by the number of groups. Default: 1

How to implement PIE



A *guidance field* is a vector field $\mathbf{v}$ used in an extended version of the minimization problem (1) above:

$$\min_f \iint_\Omega |\nabla f - \mathbf{v}|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \quad (3)$$

whose solution is the unique solution of the following Poisson equation with Dirichlet boundary conditions:

$$\Delta f = \text{div}\mathbf{v} \text{ over } \Omega, \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega}, \quad (4)$$

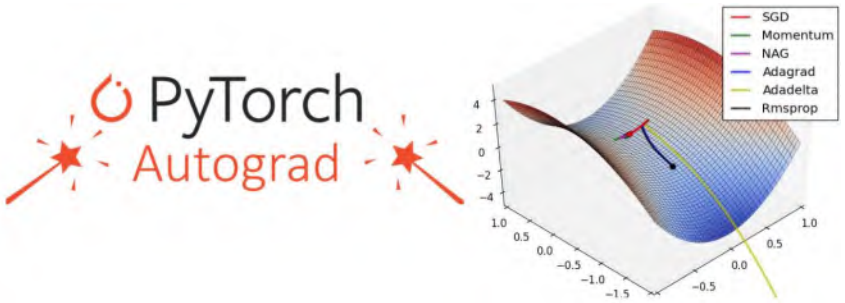
where $\text{div}\mathbf{v} = \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y}$ is the divergence of $\mathbf{v} = (u, v)$. This is the

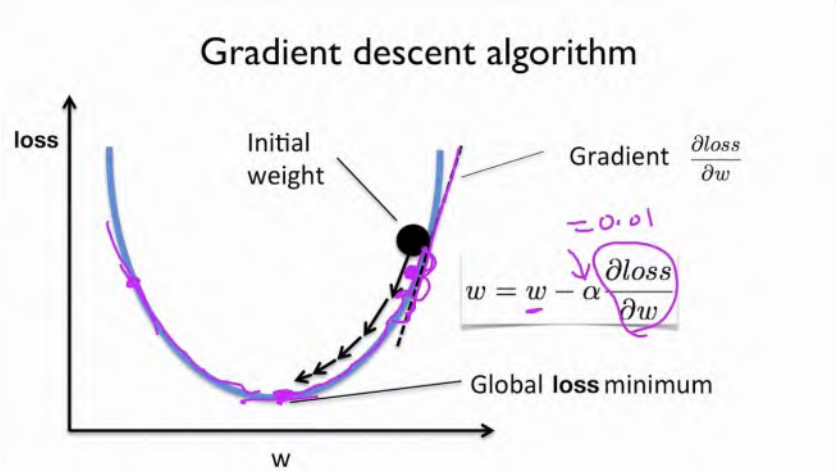
0	−1	0
−1	4	−1
0	−1	0

torch.nn.functional.conv2d(I, K) or

I[1:-1, 1:-1]*4 - I[1:-1, :-2] - I[1:-1, 2:] - I[:-2, 1:-1] - I[2:, 1:-1]

Optimize using Gradient Descent






```python
import torch
import torch.optim as optim

# 构造数据，假设我们的目标函数是 y = 2x^2 + 3x + 1
x = torch.tensor([1.0, 2.0, 3.0, 4.0, 5.0])
y_true = 2 * x * x + 3 * x + 1

# 初始化需要优化的参数 a, b, c，设置为需要梯度的变量
a = torch.tensor(0.0, requires_grad=True)
b = torch.tensor(0.0, requires_grad=True)
c = torch.tensor(0.0, requires_grad=True)

# 定义优化器，使用随机梯度下降
optimizer = optim.SGD([a, b, c], lr=0.01)

# 开始训练
for epoch in range(1000):
    # 预测 y = ax^2 + bx + c
    y_pred = a * x * x + b * x + c

    # 计算均方误差损失
    loss = torch.mean((y_pred - y_true) ** 2)

    # 反向传播计算梯度
    optimizer.zero_grad()
    loss.backward()

    # 更新参数
    optimizer.step()

    # 每 100 个 epoch 打印一次损失和参数
    if (epoch + 1) % 100 == 0:
        print(f'Epoch [{epoch+1}/1000], Loss: {loss.item():.4f}, a: {a.it
```

中国科学技术大学

University of Science and Technology of China

Q & A