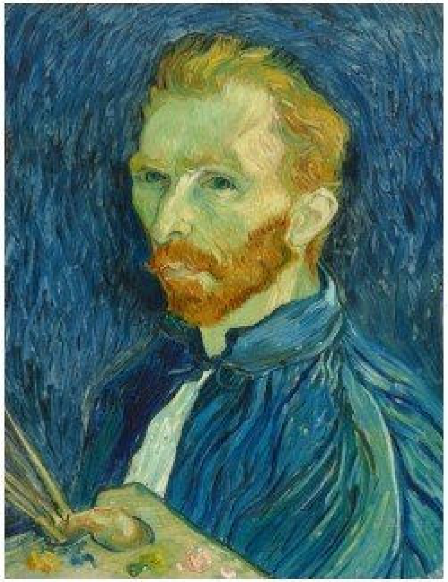
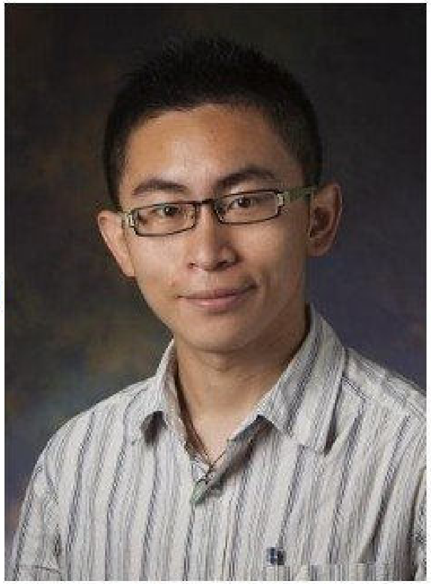
# Style transfer



Source image (**Style**)

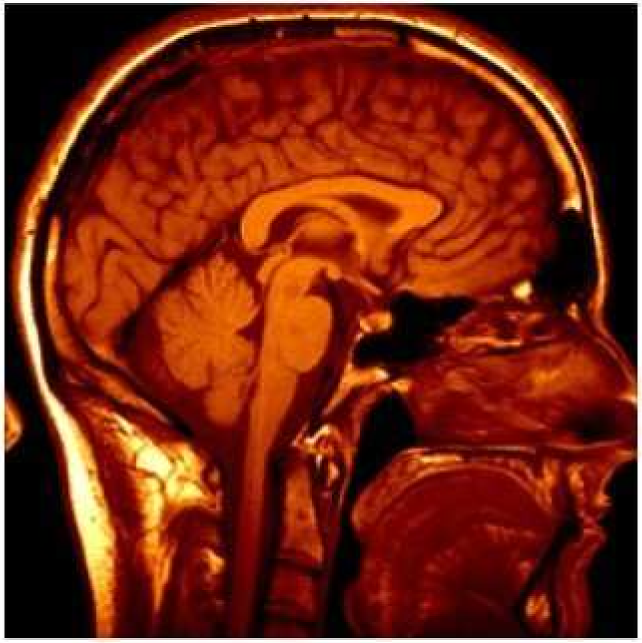


Target image (**Content**)

Output (deepart)

A Neural Algorithm of Artistic Style [Gatys et al. 2015]

# Medical imaging



3D imaging
MRI, CT

Image guided surgery
Grimson et al., MIT

# Spatial and Intensity Resolution

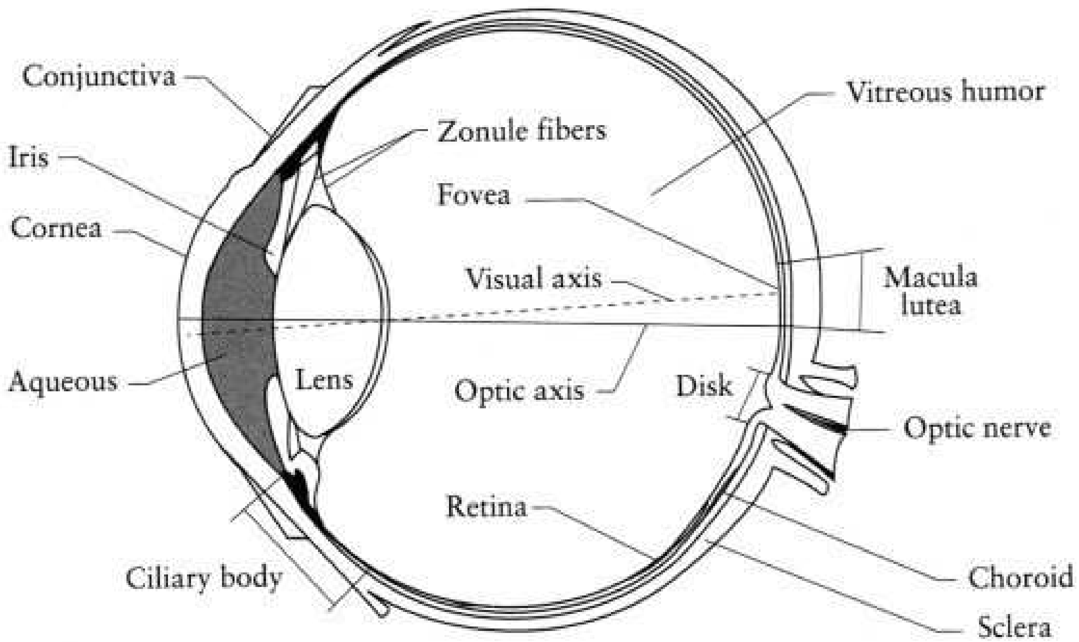
- Spatial resolution
  — A measure of the smallest discernible detail in an image
  — stated with *line pairs per unit distance, dots (pixels) per unit distance, dots per inch (dpi)*

- Intensity resolution
  — The smallest discernible change in intensity level
  — stated with *8 bits, 12 bits, 16 bits, etc.*

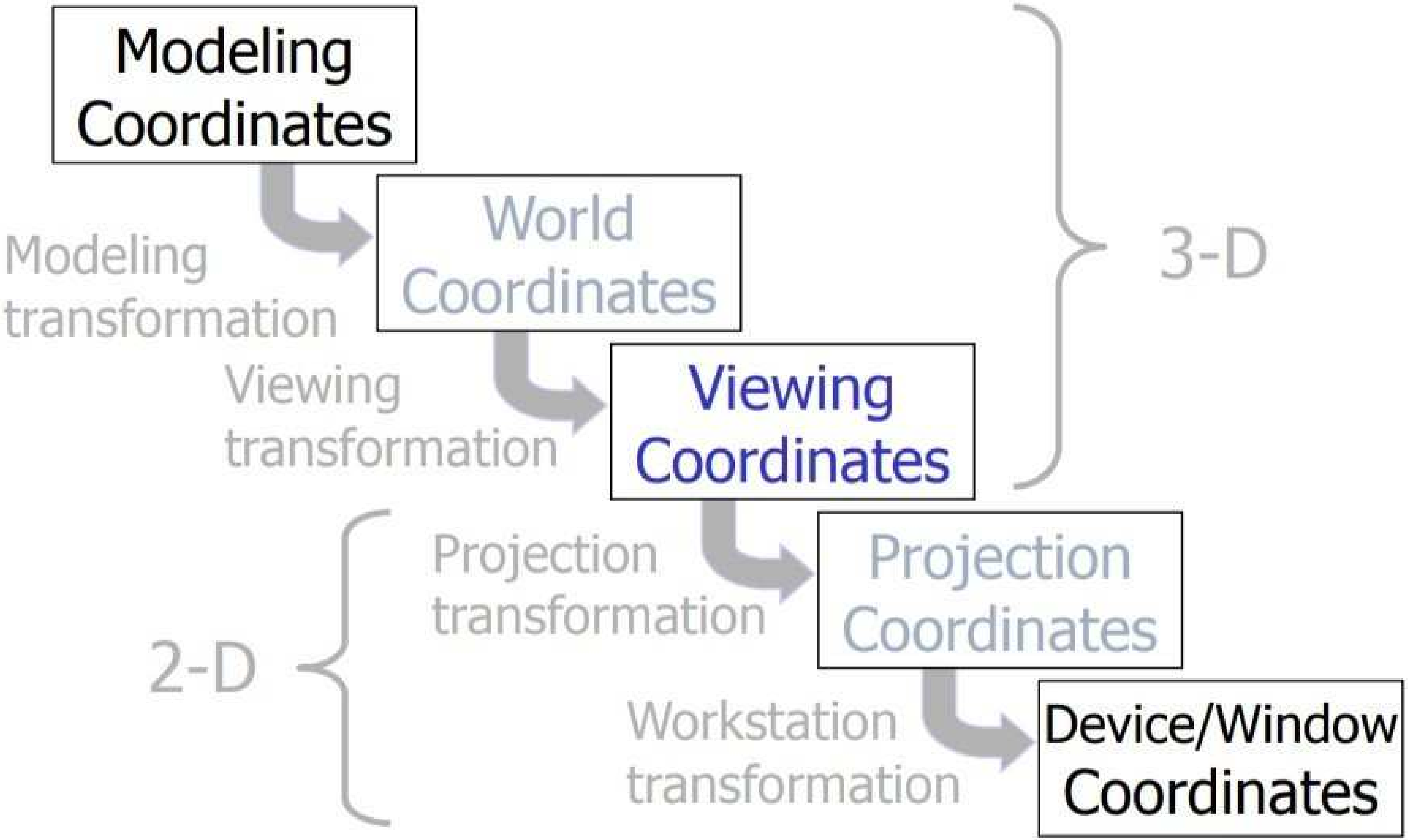
Difference ?

# The Eye

- The human eye is a camera!
  - **Iris** - colored annulus with radial muscles
  - **Pupil** - the hole (aperture) whose size is controlled by the iris
  - What's the "film"? photoreceptor cells (rods and cones) in the **retina**

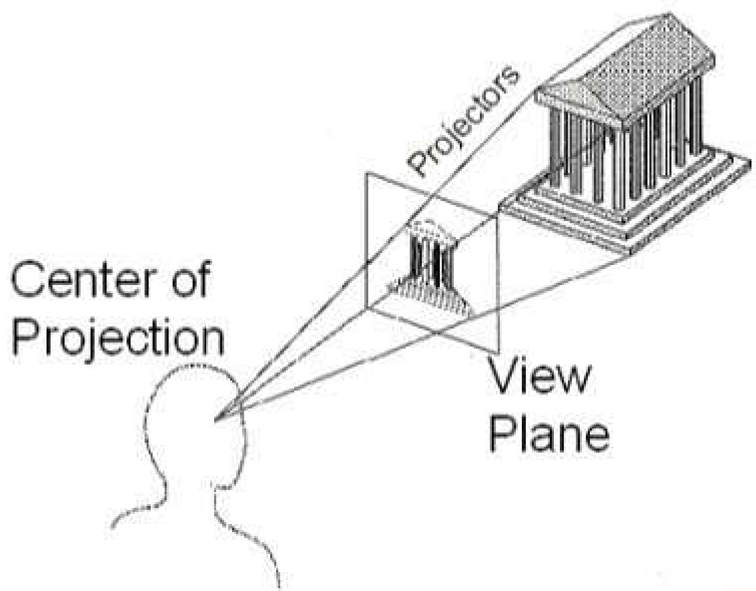


# 3-D Viewing Process



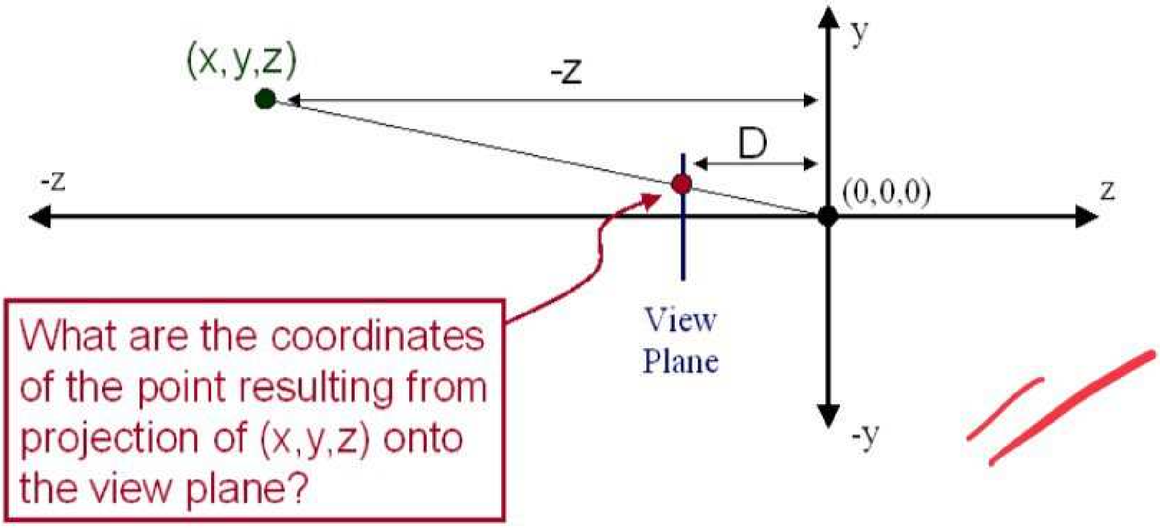
# Projection

- General definition
  - Transform points in n-space to m-space(m<n)
- In computer graphics
  - Map viewing coordinates to 2D screen coordinates



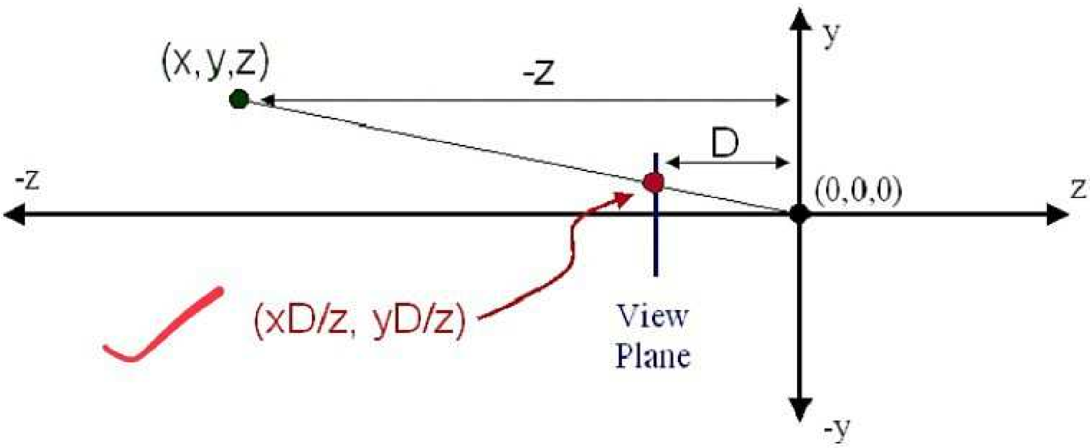
## Perspective Projection

- Compute 2D coordinates from 3D coordinates with similar triangles




## Perspective Projection

- Compute 2D coordinates from 3D coordinates with similar triangles



# Question 5

- In the following arrangement of pixels, what's the value of the chessboard distance between the circled two points?

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

# Question 6

- In the following arrangement of pixels, what's the value of the city-block distance between the circled two points?

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

# Mathematical Operations in DIP

- **Linear vs. Nonlinear Operation**

$$H\left[f(x,y)\right]=g(x,y)$$

$$H\left[a_i f_i(x,y)+a_j f_j(x,y)\right]$$

$$=H\left[a_i f_i(x,y)\right]+H\left[a_j f_j(x,y)\right]$$

**Additivity**

$$=a_i H\left[f_i(x,y)\right]+a_j H\left[f_j(x,y)\right]$$

**Homogeneity**

$$=a_i g_i(x,y)+a_j g_j(x,y)$$

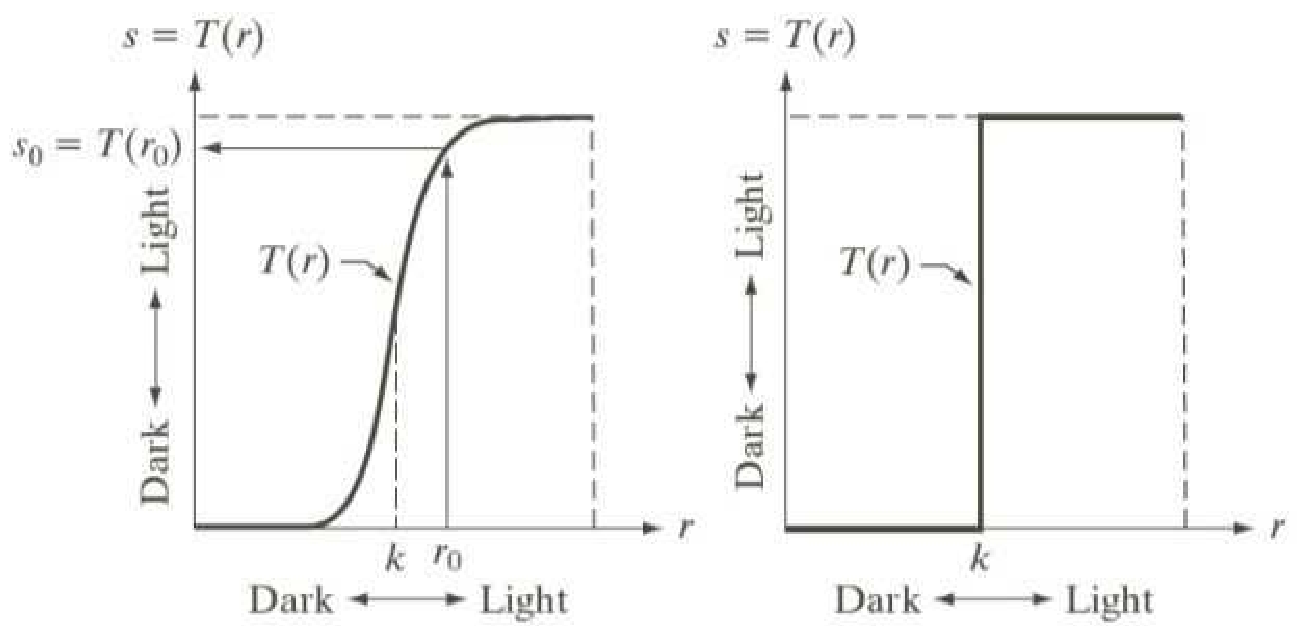
H is said to be a **linear operator;**

H is said to be a **nonlinear operator** if it does not meet the above qualification.

# Spatial Domain Process

## Intensity transformation function

$$s = T(r)$$




a b

**FIGURE 3.2** Intensity transformation functions. (a) Contrast-stretching function. (b) Thresholding function.

# Image Negatives

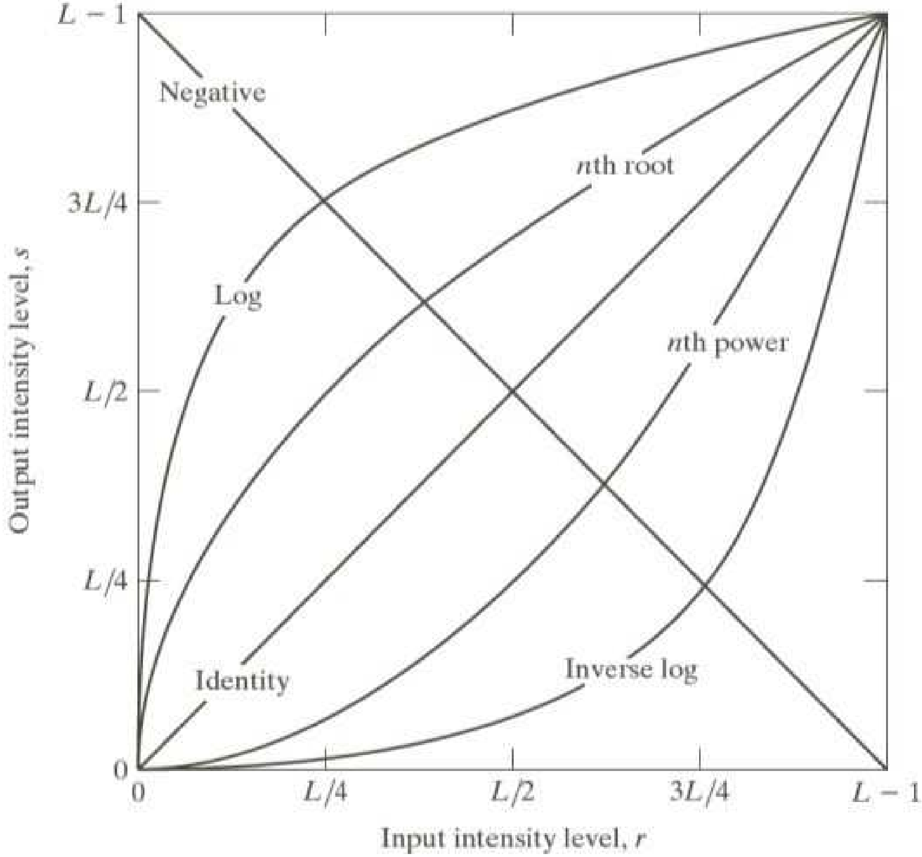
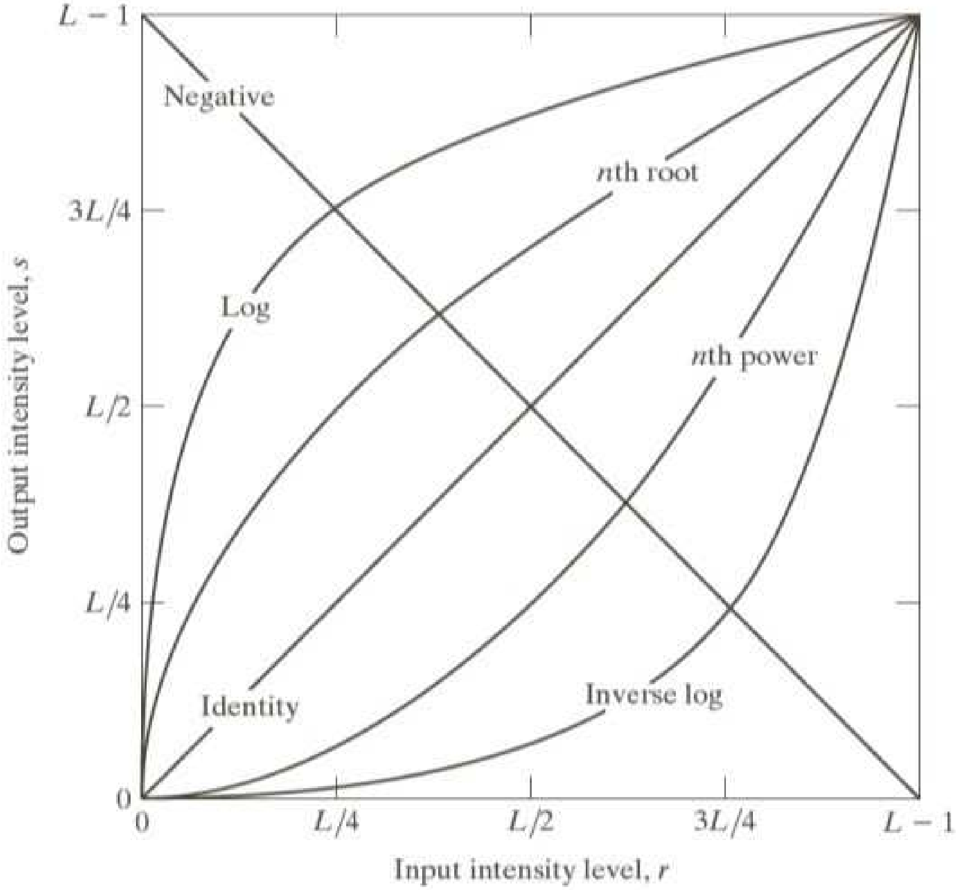



Image negatives

$$s = L - 1 - r$$

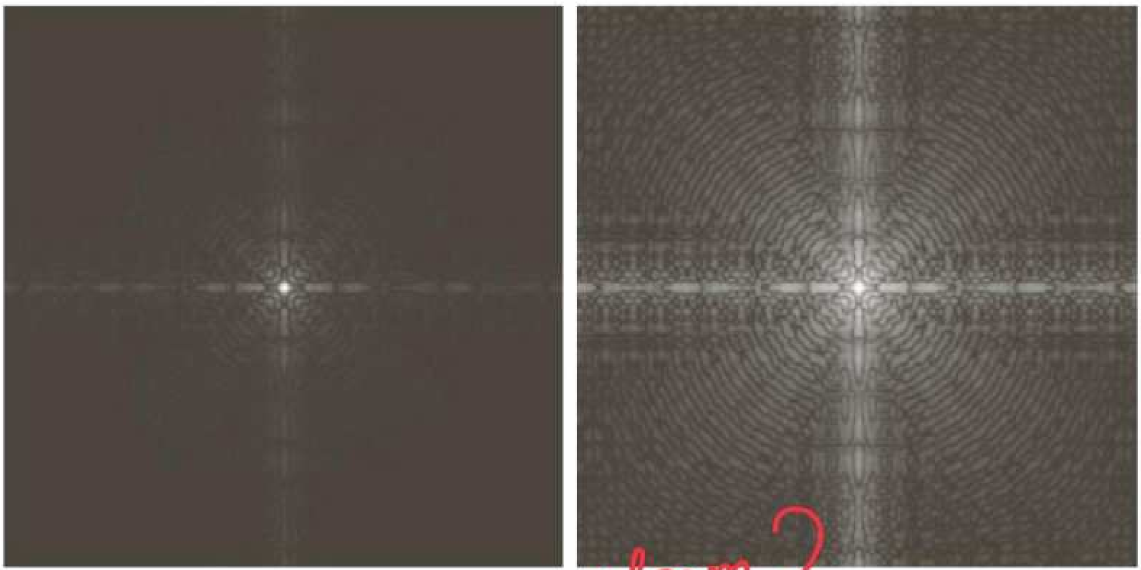
# Log Transformations




Log Transformations

$$s = c\log(1 + r)$$

## Example: Log Transformations



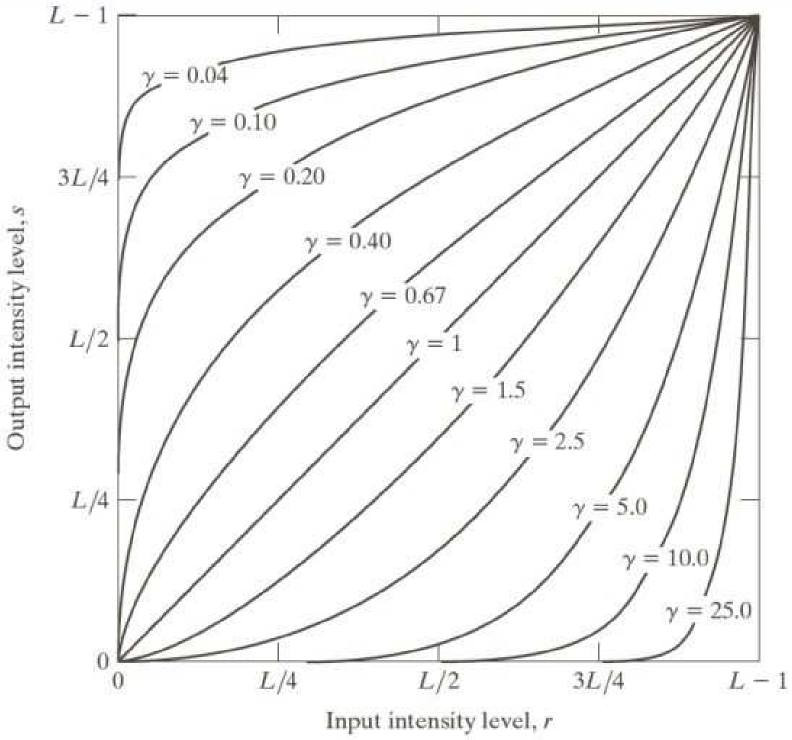
a b

**FIGURE 3.5** (a) Fourier spectrum. (b) Result of applying the log transformation in Eq. (3.2-2) with $c = 1$.

Use of log transform?

and other transforms?

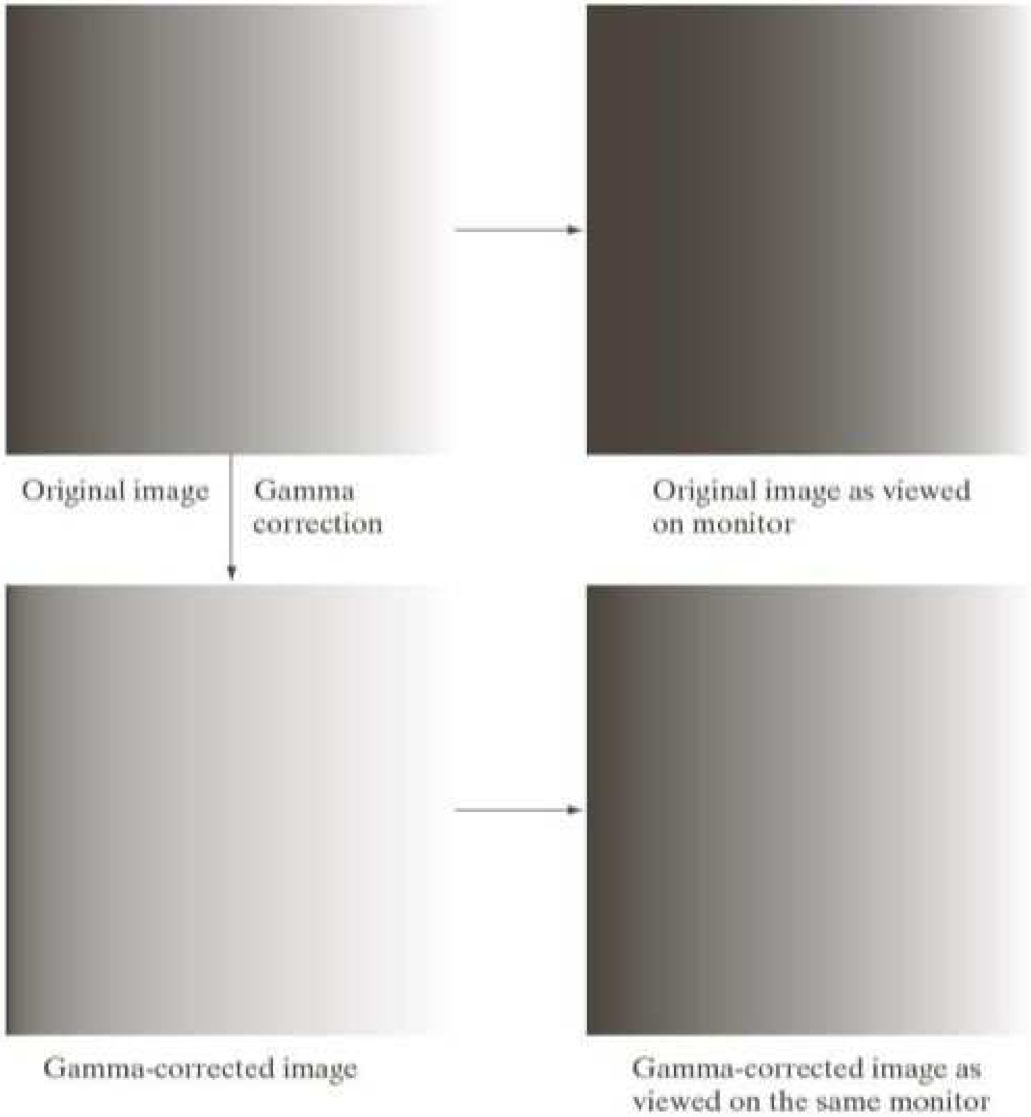
## Power-Law (Gamma) Transformations




$$s = cr^{\gamma}$$

**FIGURE 3.6** Plots of the equation $s = cr^{\gamma}$ for various values of $\gamma$ ($c = 1$ in all cases). All curves were scaled to fit in the range shown.

# Example: Gamma Transformations




Cathode ray tube (CRT) devices have an intensity-to-voltage response that is a power function, with exponents varying from approximately 1.8 to 2.5

$$s = r^{1/2.5}$$

# Piecewise-Linear Transformations

- Contrast Stretching

  — Expands the range of intensity levels in an image so that it spans the full intensity range of the recording medium or display device.

- Intensity-level Slicing

  — Highlighting a specific range of intensities in an image often is of interest.

## Bit-plane Slicing

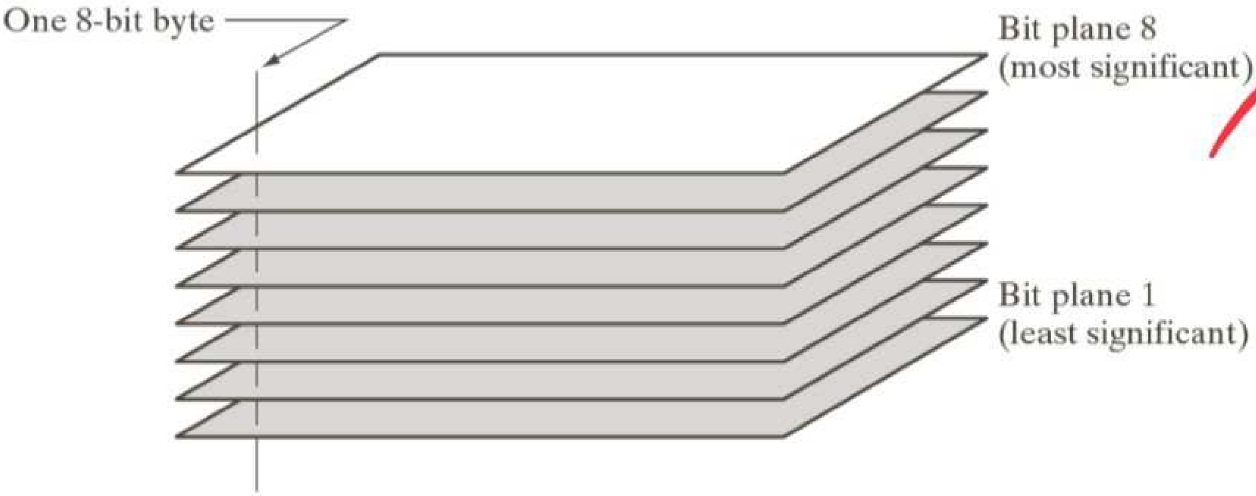



**FIGURE 3.13** Bit-plane representation of an 8-bit image.

## Bit-plane Slicing

a b c
d e f
g h i

**FIGURE 3.14** (a) An 8-bit gray-scale image of size 500 × 1192 pixels. (b) through (i) Bit planes 1 through 8, with bit plane 1 corresponding to the least significant bit. Each bit plane is a binary image.

## Histogram Processing

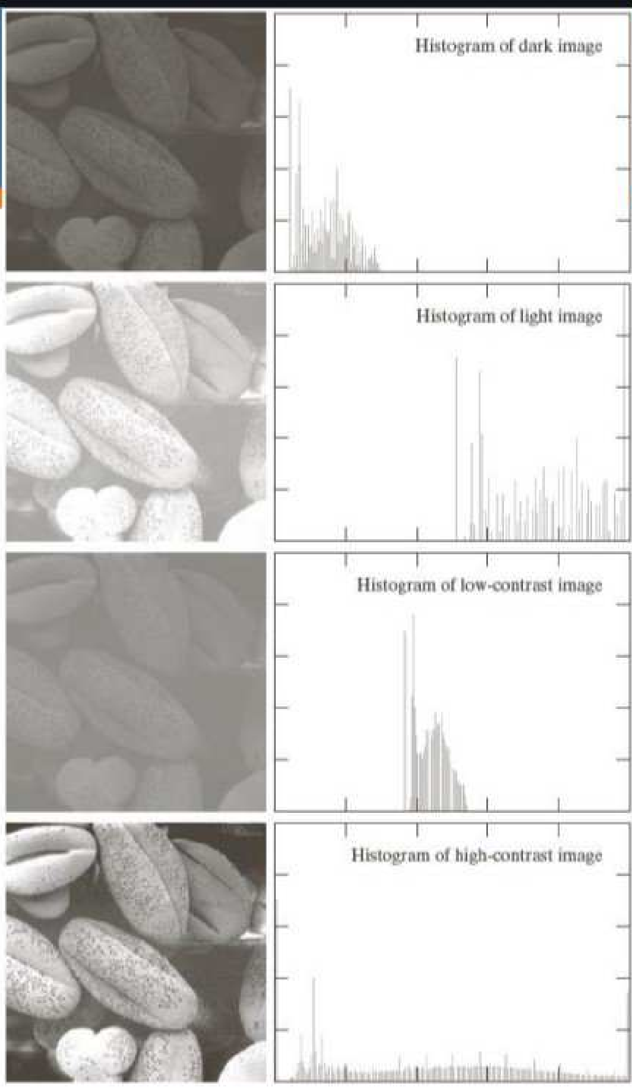
Histogram $h(r_k) = n_k$

$r_k$ is the $k^{th}$ intensity value

$n_k$ is the number of pixels in the image with intensity $r_k$

Normalized histogram $p(r_k) = \dfrac{n_k}{MN}$

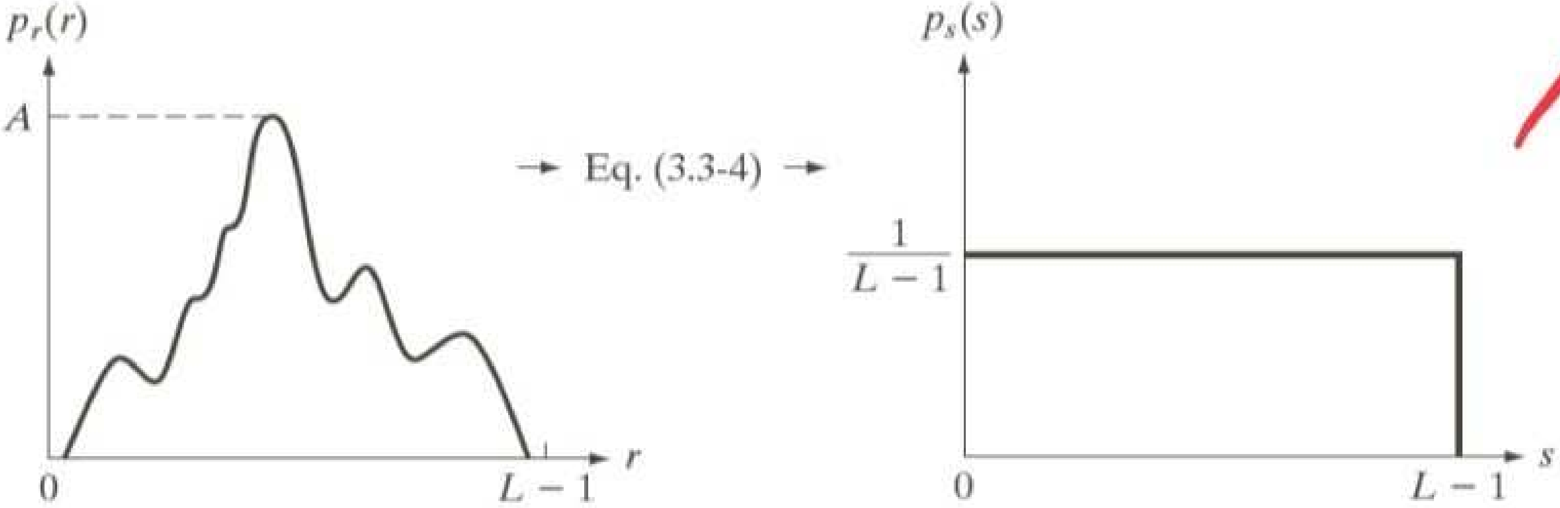
$n_k$: the number of pixels in the image of size $M \times N$ with intensity $r_k$



# Histogram Equalization

The intensity levels in an image may be viewed as random variables in the interval [0, L-1].

Let $p_r(r)$ and $p_s(s)$ denote the probability density function (PDF) of random variables $r$ and $s$.




a b

**FIGURE 3.18** (a) An arbitrary PDF. (b) Result of applying the transformation in Eq. (3.3-4) to all intensity levels, $r$. The resulting intensities, $s$, have a uniform PDF, independently of the form of the PDF of the $r$'s.

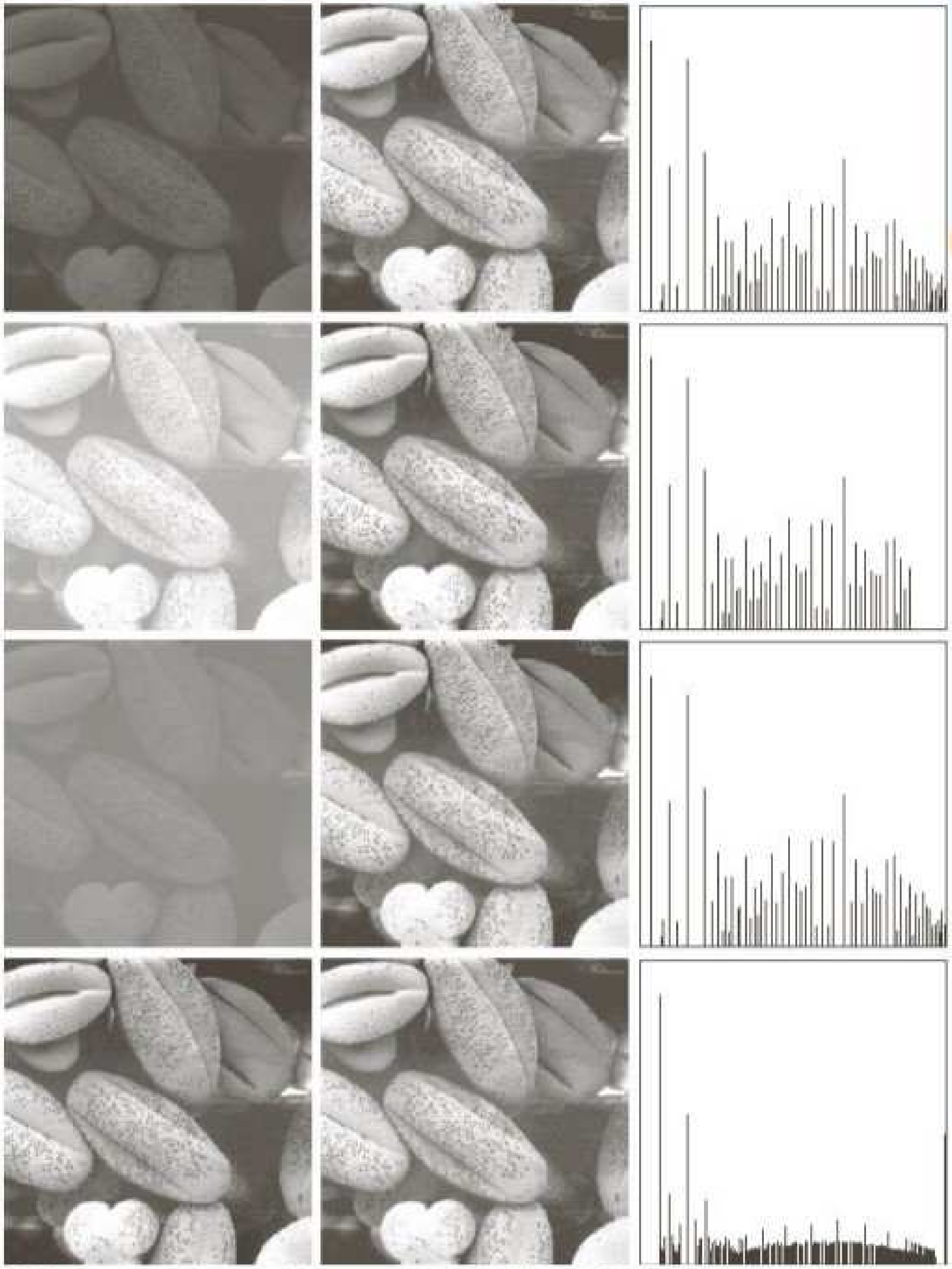


**FIGURE 3.20** Left column: images from Fig. 3.16. Center column: corresponding histogram-equalized images. Right column: histograms of the images in the center column.

Benefit of histogram equalization?

# Order-statistic (Nonlinear) Filters

— Nonlinear

— Based on ordering (ranking) the pixels contained in the filter mask

— Replacing the value of the center pixel with the value determined by the ranking result

E.g., median filter, max filter, min filter

Use ? Effects ?

# Image Enhancement in Frequency Domain

Little overview

# Fundamentals

- Let R represent the entire spatial region occupied by an image. Image segmentation is a process that partitions R into *n* sub-regions, $R_1$, $R_2$, …, $R_n$, such that

(a) $\bigcup_{i=1}^{n} R_i = R.$

(b) $R_i$ is a connected set. $i = 1, 2, ..., n.$

(c) $R_i \cap R_j = \Phi.$

(d) $Q(R_i) = \text{TRUE}$ for $i = 1, 2, ..., n.$

(e) $Q(R_i \cup R_j) = \text{FALSE}$ for any adjacent regions $R_i$ and $R_j$.

Definition of image segmentation

## Detection of Isolated Points

- The Laplacian

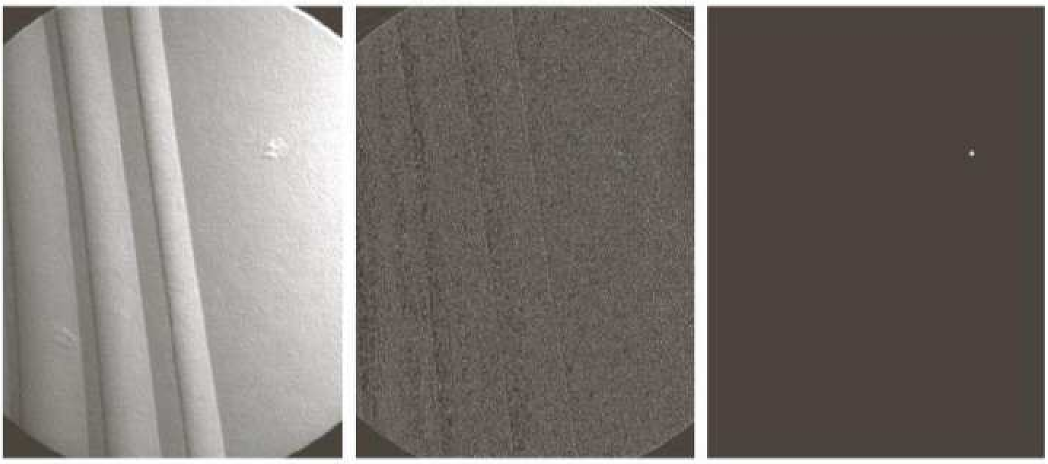
$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2}$$

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

$$-4f(x,y)$$

$$g(x,y) = \begin{cases} 1 & \text{if } |R(x,y)| \geq T \\ 0 & \text{otherwise} \end{cases} \qquad R = \sum_{k=1}^{9} w_k z_k$$

| | | |
|---|---|---|
| 1 | 1 | 1 |
| 1 | −8 | 1 |
| 1 | 1 | 1 |



a
b c d

**FIGURE 10.4** (a) Point detection (Laplacian) mask. (b) X-ray image of turbine blade with a porosity. The porosity contains a single black pixel. (c) Result of convolving the mask with the image. (d) Result of using Eq. (10.2-8) showing a single point (the point was enlarged to make it easier to see). (Original image courtesy of X-TEK Systems, Ltd.)

# Detecting Line in Specified Directions

| | | |
|---|---|---|
| −1 | −1 | −1 |
| 2 | 2 | 2 |
| −1 | −1 | −1 |

Horizontal

| | | |
|---|---|---|
| 2 | −1 | −1 |
| −1 | 2 | −1 |
| −1 | −1 | 2 |

+45°

| | | |
|---|---|---|
| −1 | 2 | −1 |
| −1 | 2 | −1 |
| −1 | 2 | −1 |

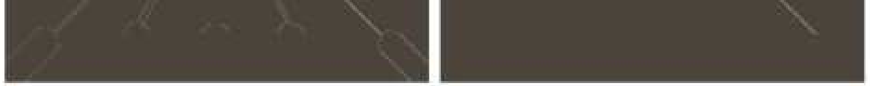
Vertical

| | | |
|---|---|---|
| −1 | −1 | 2 |
| −1 | 2 | −1 |
| 2 | −1 | −1 |

−45°

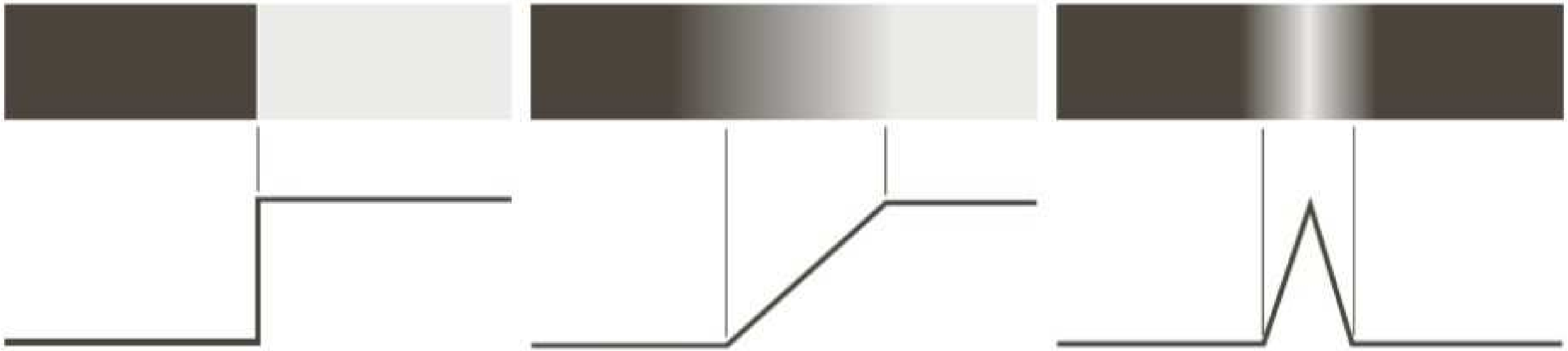
**FIGURE 10.6** Line detection masks. Angles are with respect to the axis system in Fig. 2.18(b).

- Let $R_1$, $R_2$, $R_3$, and $R_4$ denote the responses of the masks in Fig. 10.6. If, at a given point in the image, $|R_k|>|R_j|$, for all $j \neq k$, that point is said to be more likely associated with a line in the direction of mask k.



## Edge Detection

- Edges are pixels where the brightness function changes abruptly
- Edge models



a b c

**FIGURE 10.8** From left to right, models (ideal representations) of a step, a ramp, and a roof edge, and their corresponding intensity profiles.

| | | |
|---|---|---|
| $z_1$ | $z_2$ | $z_3$ |
| $z_4$ | $z_5$ | $z_6$ |
| $z_7$ | $z_8$ | $z_9$ |

| | |
|---|---|
| −1 | 0 |
| 0 | 1 |

| | |
|---|---|
| 0 | −1 |
| 1 | 0 |

Roberts

| | | |
|---|---|---|
| −1 | −1 | −1 |
| 0 | 0 | 0 |
| 1 | 1 | 1 |

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −1 | 0 | 1 |
| −1 | 0 | 1 |

Prewitt

| | | |
|---|---|---|
| −1 | −2 | −1 |
| 0 | 0 | 0 |
| 1 | 2 | 1 |

| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Sobel

a
b c
d e
f g

**FIGURE 10.14** A 3 × 3 region of an image (the $z$'s are intensity values) and various masks used to compute the gradient at the point labeled $z_5$.

| | | |
|---|---|---|
| 0 | 1 | 1 |
| −1 | 0 | 1 |
| −1 | −1 | 0 |

| | | |
|---|---|---|
| −1 | −1 | 0 |
| −1 | 0 | 1 |
| 0 | 1 | 1 |

Prewitt

| | | |
|---|---|---|
| 0 | 1 | 2 |
| −1 | 0 | 1 |
| −2 | −1 | 0 |

| | | |
|---|---|---|
| −2 | −1 | 0 |
| −1 | 0 | 1 |
| 0 | 1 | 2 |

Sobel

a b
c d

**FIGURE 10.15** Prewitt and Sobel masks for detecting diagonal edges.

# Image Classification pipeline

# An image classifier

```
def classify_image(image):
  # Some magic here?
  return class_label
```

Unlike e.g. sorting a list of numbers,

**no obvious way** to hard-code the algorithm for recognizing a cat, or other classes.

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

## Nearest Neighbor classifier

Memorize training data


```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

## Nearest Neighbor classifier


For each test image:
- Find closest train image
- Predict label of nearest image


```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

## Nearest Neighbor classifier


**Q:** With N examples, how fast are training and prediction?


```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

## Nearest Neighbor classifier

**Q:** With N examples, how fast are training and prediction?


**A:** Train O(1), predict O(N)


```python
import numpy as np
```

```python
import numpy as np

class NearestNeighbor:
  def __init__(self):
    pass

  def train(self, X, y):
    """ X is N x D where each row is an example. Y is 1-dimension of size N """
    # the nearest neighbor classifier simply remembers all the training data
    self.Xtr = X
    self.ytr = y

  def predict(self, X):
    """ X is N x D where each row is an example we wish to predict label for """
    num_test = X.shape[0]
    # lets make sure that the output type matches the input type
    Ypred = np.zeros(num_test, dtype = self.ytr.dtype)

    # loop over all test rows
    for i in xrange(num_test):
      # find the nearest training image to the i'th test image
      # using the L1 distance (sum of absolute value differences)
      distances = np.sum(np.abs(self.Xtr - X[i,:]), axis = 1)
      min_index = np.argmin(distances) # get the index with smallest distance
      Ypred[i] = self.ytr[min_index] # predict the label of the nearest example

    return Ypred
```

## Nearest Neighbor classifier

**Q:** With N examples, how fast are training and prediction?

**A:** Train O(1), predict O(N)

This is bad: we want classifiers that are **fast** at prediction; **slow** for training is ok

## Setting Hyperparameters

**Idea #1**: Choose hyperparameters that work best on the data

**BAD**: K = 1 always works perfectly on training data

| Your Dataset |
|---|

## Setting Hyperparameters

**Idea #1**: Choose hyperparameters that work best on the data

**BAD**: K = 1 always works perfectly on training data

| Your Dataset |
|---|

**Idea #2**: Split data into **train** and **test**, choose hyperparameters that work best on test data

| train | test |
|---|---|

## Setting Hyperparameters

**Idea #1**: Choose hyperparameters that work best on the data

**BAD**: K = 1 always works perfectly on training data

| Your Dataset |
|---|

**Idea #2**: Split data into **train** and **test**, choose hyperparameters that work best on test data

**BAD**: No idea how algorithm will perform on new data

| train | test |
|---|---|

## Setting Hyperparameters

**Idea #1**: Choose hyperparameters that work best on the data

**BAD**: K = 1 always works perfectly on training data

| Your Dataset |
|---|

**Idea #2**: Split data into **train** and **test**, choose hyperparameters that work best on test data

**BAD**: No idea how algorithm will perform on new data

| train | test |
|---|---|

**Idea #3**: Split data into **train**, **val**, and **test**; choose hyperparameters on val and evaluate on test

**Better!**

| train | validation | test |
|---|---|---|

## Setting Hyperparameters

# Setting Hyperparameters

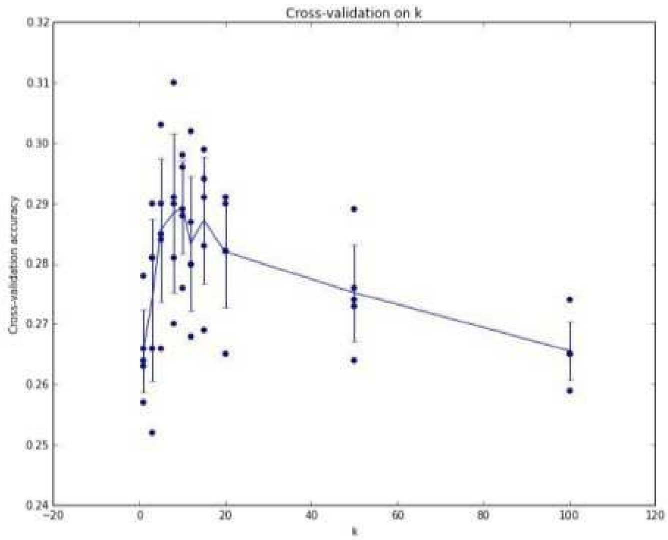
| |
|---|
| Your Dataset |

**Idea #4**: **Cross-Validation**: Split data into **folds**,
try each fold as validation and average the results

| | | | | | |
|---|---|---|---|---|---|
| fold 1 | fold 2 | fold 3 | fold 4 | fold 5 | test |
| fold 1 | fold 2 | fold 3 | fold 4 | fold 5 | test |
| fold 1 | fold 2 | fold 3 | fold 4 | fold 5 | test |

Useful for small datasets, but not used too frequently in deep learning

# Setting Hyperparameters




Example of
5-fold cross-validation
for the value of **k.**

Each point: single
outcome.

The line goes
through the mean, bars
indicated standard
deviation

(Seems that k ~= 7 works best
for this data)

## Parametric Approach

Image

Array of **32x32x3** numbers (3072 numbers total)

$$\text{Image} \longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow \text{10 numbers giving class scores}$$

**W** parameters or weights

## Parametric Approach: Linear Classifier

$$f(x,W) = Wx$$

Image

Array of **32x32x3** numbers (3072 numbers total)

$$\text{Image} \longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow \text{10 numbers giving class scores}$$

**W** parameters or weights

## Parametric Approach: Linear Classifier

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1}$$

Image

Array of **32x32x3** numbers (3072 numbers total)

$$\text{Image} \longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow \text{10 numbers giving class scores}$$

**W** parameters or weights

## Parametric Approach: Linear Classifier

$$\underbrace{\boxed{f(x,W)}}_{10\text{x}1} = \underbrace{\boxed{W}}_{10\text{x}3072}\overbrace{\boxed{x}}^{3072\text{x}1} + \boxed{b}\ \ 10\text{x}1$$

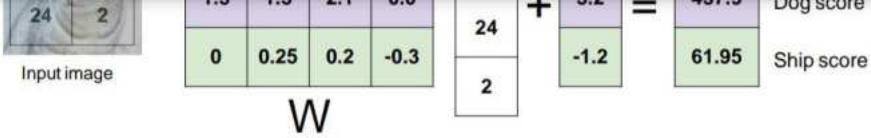
Image

Array of **32x32x3** numbers (3072 numbers total)

$$\text{Image} \longrightarrow f(\mathbf{x},\mathbf{W}) \longrightarrow \text{10 numbers giving class scores}$$

**W** parameters or weights

Example with an image with 4 pixels, and 3 classes (cat/dog/ship)




## Interpreting a Linear Classifier


$$f(x,W) = Wx + b$$

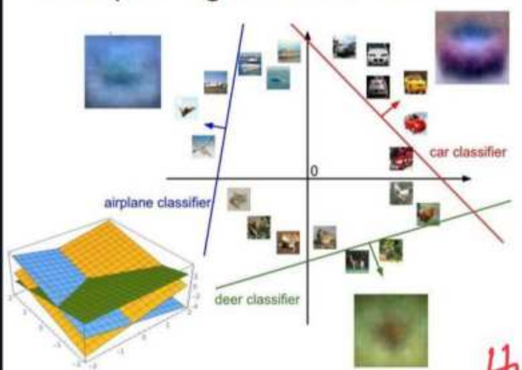
What is this thing doing?

## Interpreting a Linear Classifier


$$f(x,W) = Wx + b$$

Example trained weights of a linear classifier trained on CIFAR-10:


## Interpreting a Linear Classifier




$$f(x,W) = Wx + b$$

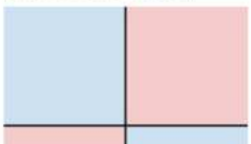
Array of **32x32x3** numbers
(3072 numbers total)

Hyperplane ?
Normal to a hyperplane

## Hard cases for a linear classifier

**Class 1**:
pixels coord > 0 odd

**Class 2**:
pixels coord > 0 even

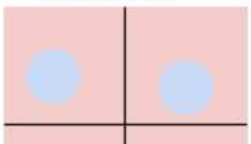


**Class 1**:
1 <= L2 norm <= 2

**Class 2**:
Everything else

**Class 1**:
Three modes

**Class 2**:
Everything else



Convolution, feature map, pooling, FCN

Basics of CNN (as taught in class)

Thank you

Here are concise notes on Image Enhancement in the Frequency Domain for a 6-mark question:

1. Fourier Transform:
    - Transforms a spatial image into its frequency domain representation.
    - Used to process and enhance images by modifying their frequency components.
2. Filters in Frequency Domain:
    - Low-Pass Filters (LPF): Smoothens images by removing high-frequency components.
        - Examples: Ideal LPF, Butterworth LPF, Gaussian LPF.
    - High-Pass Filters (HPF): Sharpens images by removing low-frequency components.
        - Examples: Ideal HPF, Butterworth HPF, Gaussian HPF.
3. Notch Filters:
    - Used to remove periodic noise by eliminating specific frequencies in the transform.
4. Selective Filters:
    - Bandreject Filters: Removes a range of frequencies.
    - Bandpass Filters: Keeps only a specific frequency band.
5. Image Sharpening:
    - Achieved by subtracting the low-frequency components from the original image.
6. Applications:
    - Used for noise removal, image smoothing, and sharpening.

Let me know if you'd like detailed explanations for any of these points.