Module 3
Point Operations & MLP s

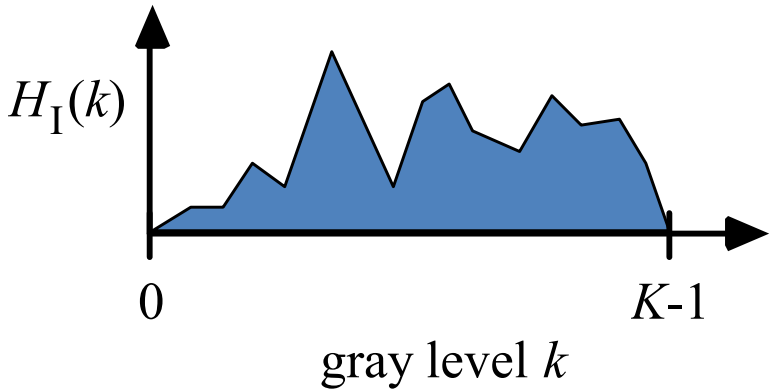
- Linear Point Operations
- Nonlinear Point Operations
- Histogram Shaping and Matching
- Arithmetic Image Operations
- Geometric Image Operations
- Multilayer Perceptrons

Simple Histogram Operations

- Recall: the **gray-level histogram** $\boldsymbol{H}_{\mathbf{I}}$ of an image $\mathbf{I}$ is a graph of the frequency of occurrence of each gray level in $\mathbf{I}$.

- $\boldsymbol{H}_{\mathbf{I}}$ is a one-dimensional function with domain $0, \ldots, K\text{-}1$.

- $\boldsymbol{H}_{\mathbf{I}}(k) = n$ if gray-level k occurs exactly n times in $\mathbf{I}$ (for each $k = 0, \ldots, K\text{-}1$).



- $\boldsymbol{H_I}$ contains **no spatial information** - only the relative frequency of intensities (first-order information only).

- Much useful info is obtainable from $\boldsymbol{H_I}$.

- Image quality is affected (enhanced, modified) by altering $\boldsymbol{H_I}$.

Average Optical Density (AOD)

- The average intensity or brightness of the $M \times N$ image $\mathbf{I}$:

$$\mathrm{AOD}(\mathbf{I}) = \mathrm{L}_{\mathbf{AVE}}(\mathbf{I}) = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \mathrm{I}(m,n)$$

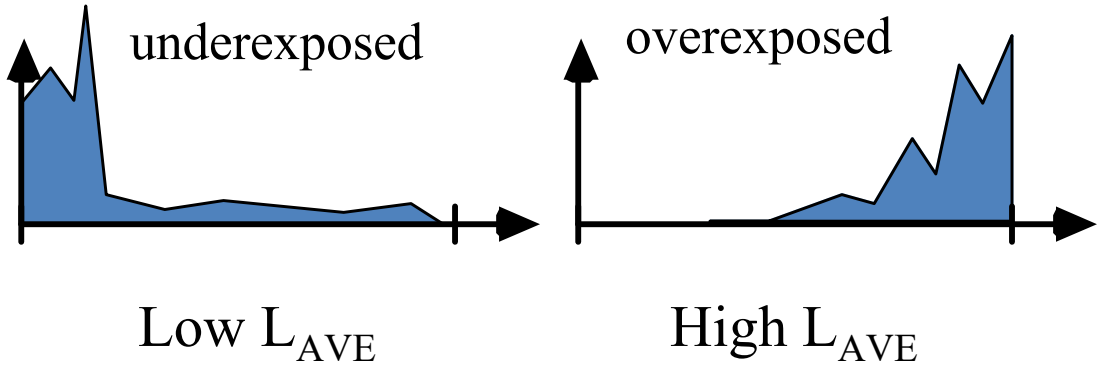
- Can compute it from the histogram as well:

$$\mathrm{AOD}(\mathbf{I}) = \frac{1}{MN} \sum_{k=0}^{K\text{-}1} k H_{\mathbf{I}}(k)$$

- "L" often used for "Luma" or "Luminance" – closely related to the idea of "brightness"

Average Brightness

- Examining the histogram can reveal possible errors in the imaging process:



- By operating on the histogram, such errors can be ameliorated.

Point Operations

- **Point operation:** a **function** f on **single pixels** in $\mathbf{I}$:
  $\mathrm{J}(m,n) = \mathrm{f}[\mathrm{I}(m,n)],\ 0 \le m \le M\text{-}1,\ 0 \le n \le N\text{-}1$

- The same function f is applied at every (m,n).

- Unlike **local operation**s (e.g. OPEN), does not use the **neighbors** of $\mathrm{I}(m,n)$.

- Point operations **don't** modify **spatial relationships**. They **do** change the **histogram**, and therefore the appearance of the image.

Linear Point Operations

- The simplest class of point operations. They **offset** and **scale** the image intensities.
  - **offset**, also called **bias**, is like the "brightness" control on your television.
  - **scale**, also called **gain**, is like the "contrast" control on your television.
- **offset (or "bias"):**

$$\mathrm{J}(m,n) = \mathrm{I}(m,n) + L$$

- **gain (or "scale"):**

$$\mathrm{J}(m,n) = P \cdot \mathrm{I}(m,n)$$

Image Offset

- If $L > 0$, then $\mathbf{J}$ is a **brightened** version of $\mathbf{I}$.
  If $L < 0$, then it is a **dimmed** version of $\mathbf{I}$.
- The histogram is shifted by an amount L:

$$\boldsymbol{H}_{\mathbf{J}}(k) = \boldsymbol{H}_{\mathbf{I}}(k\text{-}L)$$

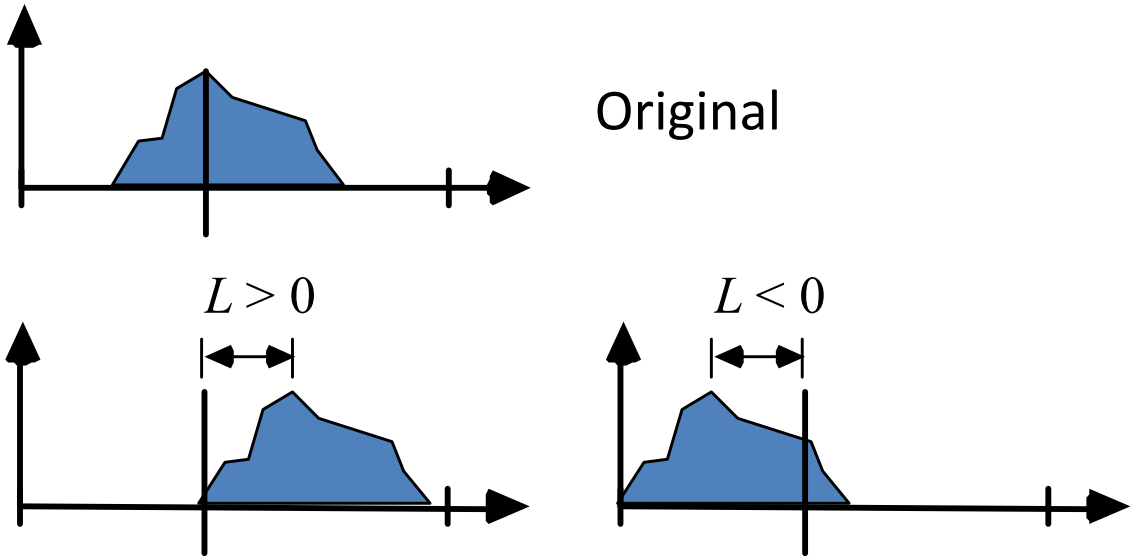


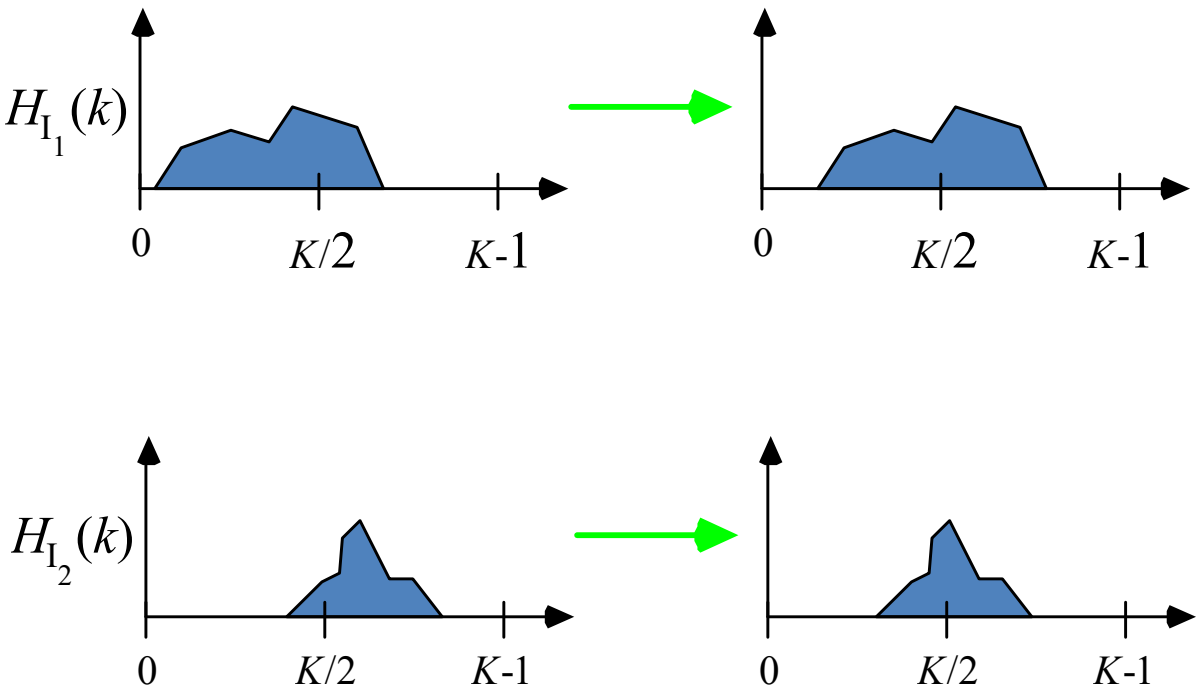
Image Offset Example

- **Compare** multiple images $\mathbf{I}_1, \mathbf{I}_2, \ldots, \mathbf{I}_p$ of the same scene taken with different exposures or lighting conditions.

- A solution: **equalize** the image AODs, e.g., set them all to $K/2$ (for gray scale range $0, \ldots, K\text{-}1$).

- Let $L_q = \mathrm{AOD}(\mathbf{I}_q)$, $q = 1, \ldots, p$. Then equalize via

$$\mathrm{J}_q(m,n) = \mathrm{I}_q(m,n) - L_q + K/2$$



AOD Equalization

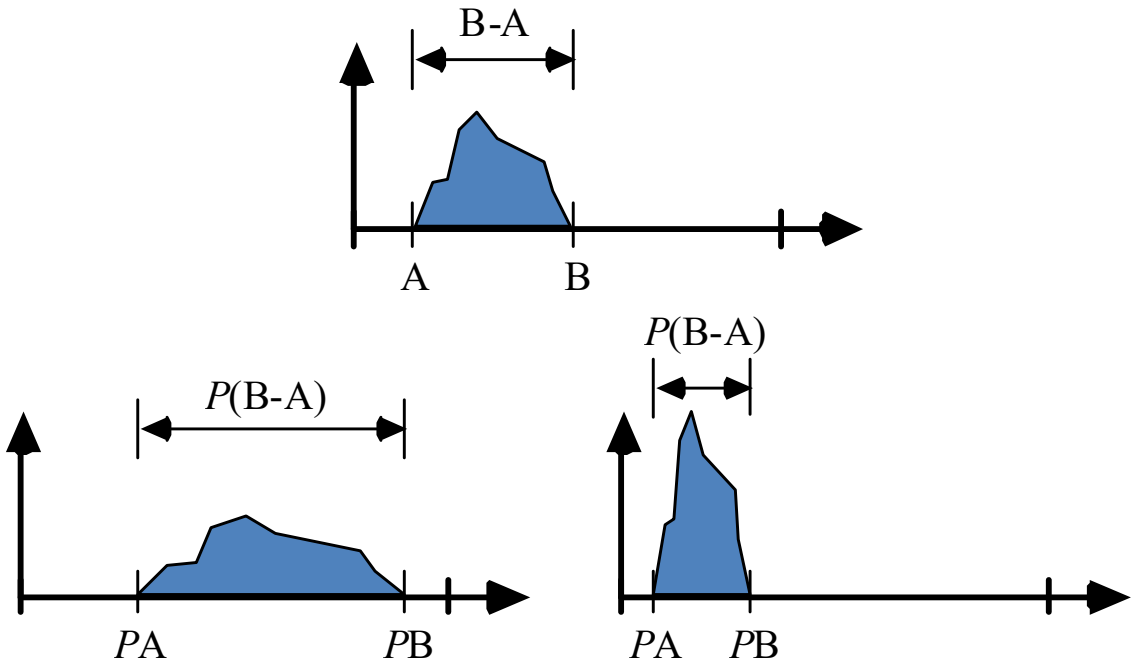
Image Scaling

- $\mathrm{J}(m,n) = P \cdot \mathrm{I}(m,n)$
- If $|P| > 1$, the **intensity range** is **widened**.
- If $|P| < 1$, the **intensity range** is **narrowed**.
- Multiplying by P **stretches** or **compresses** the image histogram by a factor P:

$$\boldsymbol{H}_{\mathbf{J}}(k) = \boldsymbol{H}_{\mathbf{I}}(k/P) \qquad \text{(continuous)}$$

$$\boldsymbol{H}_{\mathbf{J}}(k) = \boldsymbol{H}_{\mathbf{I}}[\mathrm{INT}(k/P)] \quad \text{(discrete)}$$

Image Scaling



- An image with a compressed gray level range generally has a **poor appearance** – washed out, not visually striking.

Linear Point Operations: Offset & Scaling

- Given reals L and P, a **linear point operation** on $\mathbf{I}$ is a function

$$\mathrm{J}(m,n) = P \cdot \mathrm{I}(m,n) + L$$

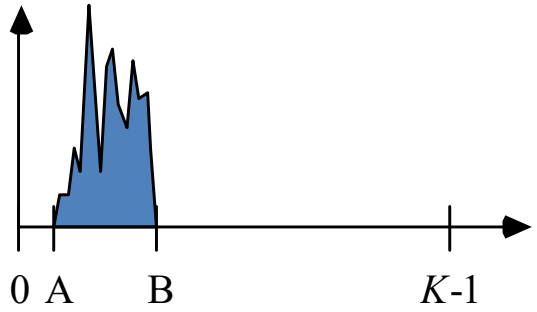
comprising both offset and scaling.

- If $P < 0$, the histogram is **reversed**, creating a **negative** image. In this case, usually $P = -1$ and $L = K\text{-}1$:

$$\mathrm{J}(m,n) = (K\text{-}1) - \mathrm{I}(m,n) \quad \text{(negative)}$$

Full-Scale Contrast Stretch (FSCS)

- The **most common** linear point operation. Suppose $\mathbf{I}$ has a compressed histogram:



- Let A and B be the min and max gray levels in $\mathbf{I}$.
- Define

$$\mathrm{J}(m,n) = P \cdot \mathrm{I}(m,n) + L$$

such that $P\mathrm{A} + L = 0$ and $P\mathrm{B} + L = (K\text{-}1)$.

- This maps the old gray level A to the new gray level 0 and the old gray level B to the new gray level K-1.

Full-Scale Contrast Stretch

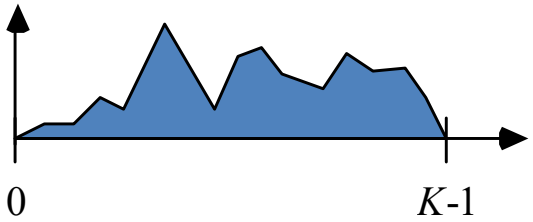
- Solving these **2 equations in 2 unknowns** yields:

$$P = \frac{K-1}{\mathrm{B}-\mathrm{A}} \text{ and } L = -\mathrm{A}\frac{K-1}{\mathrm{B}-\mathrm{A}}$$

or

$$\mathrm{J}(m,n) = \frac{(K-1)}{(\mathrm{B}-\mathrm{A})}\left[\mathrm{I}(m,n) - \mathrm{A}\right]$$

- The result is an image **J** with a full-range histogram:



Full-Scale Contrast Stretch

- **NOTE:** If the resulting values $\mathrm{J}(m,n)$ are **floating point**, then each pixel should be rounded to the nearest **integer** as a final step.
  - This will happen, for example, when we talk about histogram flattening and histogram shaping later in this module.
- **Many** image processing operations will produce an output image having pixels with floating point values and/or pixels with values that are **not** normalized to the range [0, 255].
  - Examples: LTI filters, nonlinear point operations, etc.
  - After such an operation, a full-scale contrast stretch (FSCS) is almost always applied to map the pixels of the output image back into the range [0, 255] for display.

Nonlinear Point Operations

- Now consider **nonlinear point functions** f

$$\mathrm{J}(m,n) = \mathrm{f}\,[\mathrm{I}(m,n)].$$

- A very broad class of functions!
- Commonly used:

$$\mathrm{J}(m,n) = |\mathrm{I}(m,n)| \qquad \text{(magnitude)}$$

$$\mathrm{J}(m,n) = [\mathrm{I}(m,n)]^2 \qquad \text{(square-law)}$$

$$\mathrm{J}(m,n) = [\mathrm{I}(m,n)]^{1/2} \qquad \text{(square root)}$$

$$\mathrm{J}(m,n) = \log[1+\mathrm{I}(m,n)] \qquad \text{(logarithm)}$$

$$\mathrm{J}(m,n) = \exp[\mathrm{I}(m,n)] = \mathrm{e}^{\mathrm{I}(m,n)} \qquad \text{(exponential)}$$

- Most of these are special-purpose, for example…

Logarithmic Range Compression

- Small groupings of **very bright pixels** may dominate the perception of an image at the expense of other rich information that is **less bright and less visible**.

- Astronomical images of faint nebulae and galaxies with dominating stars are an excellent example.

The Rosette Nebula

Logarithmic Range Compression

- **Logarithmic transformation**

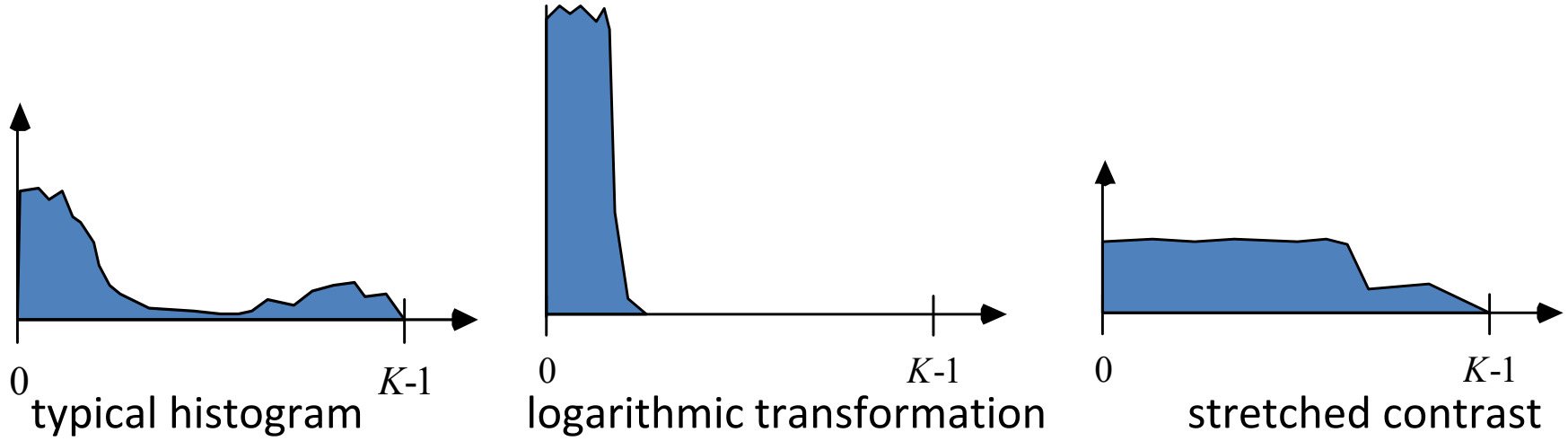
$$\mathrm{J}(m,n) = \log[1+\mathrm{I}(m,n)]$$

nonlinearly **compresses** and **equalizes** the gray-scales.

- Bright intensities are compressed much more heavily - thus **faint details** emerge.

Logarithmic Range Compression

- A full-scale contrast stretch then utilizes the full gray-scale range:



Contrast Stretched Rosette

Rosette in color

Gamma Correction

- **Monitors** that **display images** and **videos** often have a **nonlinear response**.
- Commonly an **exponential nonlinearity**

$$\text{Display}(m,n) = [\text{I}(m,n)]^{\gamma}$$

- **Gamma correction** is (digital) pre-processing to **correct the nonlinearity**:

$$\text{J}(m,n) = [\text{I}(m,n)]^{(1/\gamma)}$$

- Then

$$\text{Display}(m,n) = [\text{J}(m,n)]^{\gamma} = \text{I}(m,n)$$

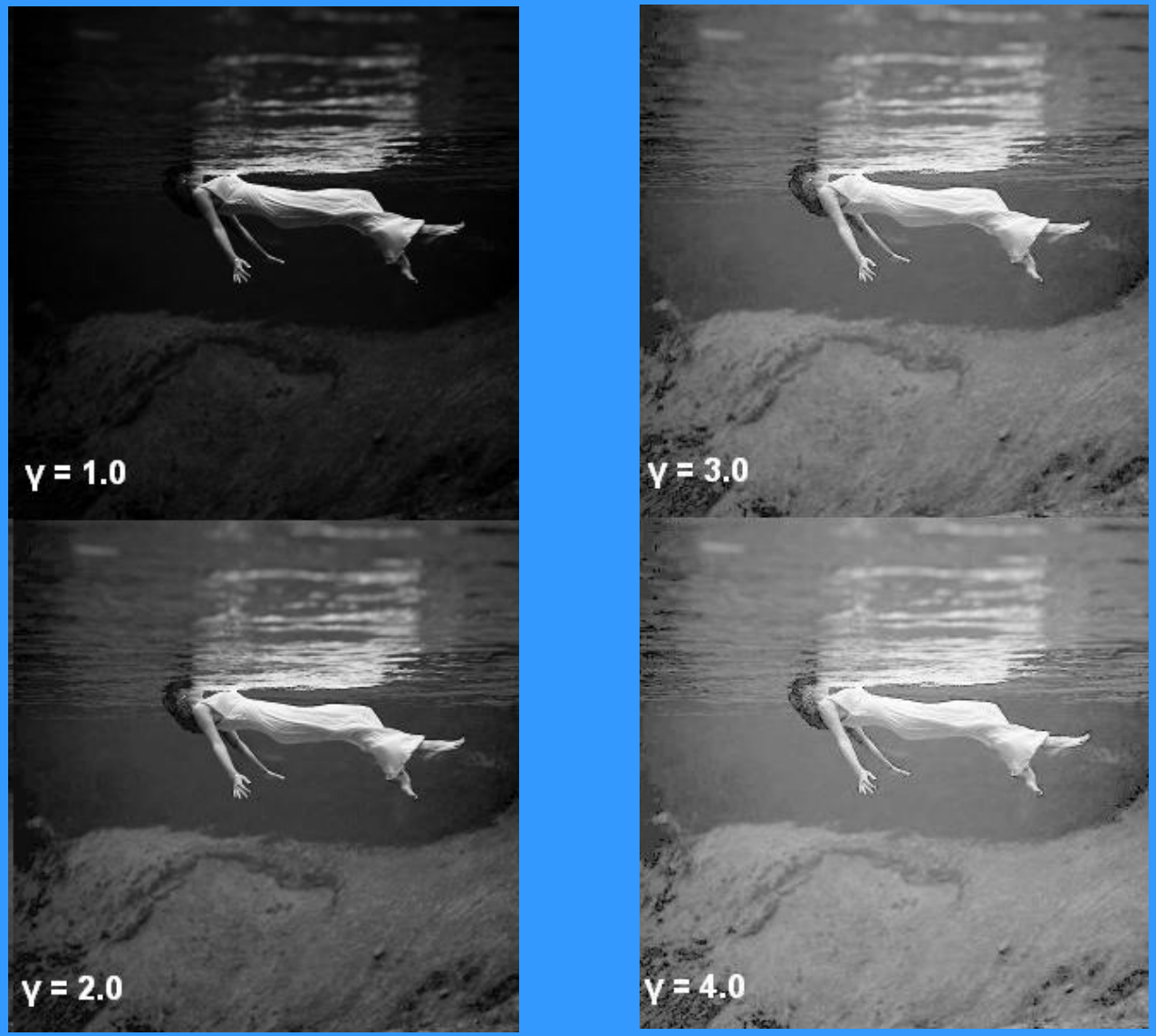
Gamma Correction

- For a CRT (e.g., analog NTSC TV), typically

$$\gamma = 2.2$$

- This is accomplished by **mapping** all **luminances** (or chrominances) to a $[0, 1]$ range.
- Hence **black** (0) and **white** (1) are unaffected.
- **Plasma** and **LCD** have **linear** characteristics**,** hence do not need gamma correction. But many devices that feed them still gamma correct; hence reverse nonlinearity often needed.

Gamma Correction



Histogram Equalization or "Flattening"

- An image with a **flat** histogram makes rich use of the available gray-scale range. This might be an image with

  - Smooth changes in intensity across many gray levels

  - Lots of texture covering many gray levels

- We can obtain an image with an **approximately** flat histogram using nonlinear point operations.
- This is called "histogram flattening" or "histogram equalization."
- It is a special case of "histogram shaping."

Normalized Histogram

- Define the **normalized histogram**:

$$p_{\mathbf{I}}(k) = \frac{1}{MN} H_{\mathbf{I}}(k) \; ; \; k \; = \; 0 \; , \ldots , \; K \; - 1$$

- The normalized histogram **sums to one**:

$$\sum_{k=0}^{K-1} p_{\mathbf{I}}(k) \; = \; 1$$

- Note that $p_{\mathbf{I}}(k)$ is the **probability** that gray level k occurs in the image (at any given coordinate): can be interpreted as an empirical probability density function (pdf) for the image.

Cumulative Histogram

- The **cumulative histogram** is

$$P_{\mathbf{I}}(r) = \sum_{k=0}^{r} p_{\mathbf{I}}(k) \; ; \; r = 0 \; ,..., \; K\text{-}1$$

  which is **non-decreasing**; also, $P_{\mathbf{I}}(K\text{-}1) = 1.$

- Probabilistic interpretation: at any (m,n):

$$P_{\mathbf{I}}(r) = \Pr\{\mathrm{I}(m,n) \leq r\}$$

$$p_{\mathbf{I}}(r) = P_{\mathbf{I}}(r) \text{ - } P_{\mathbf{I}}(r\text{-}1) \; ; \; r = 0,..., \; K\text{-}1$$

Relation to Probability Theory

- The functions $p_{\mathbf{I}}(x)$ and $P_{\mathbf{I}}(x)$ may be rigorously interpreted as the probability density function (pdf) and cumulative distribution function (cdf) of the image $\mathbf{I}$.
- As with any pdf & cdf, we have $p_{\mathbf{I}}(x) = \mathrm{d}P_{\mathbf{I}}(x)/\mathrm{d}x$
- For theoretical work, it is useful to assume that:
  - the grayscales x are continuous: $x \in \mathbb{R}$.
  - $P_{\mathbf{I}}(x)$ is one-to-one, so that $P_{\mathbf{I}}^{-1}(x)$ exists.
- We'll describe histogram flattening and shaping for these assumptions, then adapt them for the more practical discrete case.

Histogram Equalization

- Let $\mathbf{I}$ be an image with continuous gray scales $x \in \mathbb{R}$.
- Let $p_{\mathbf{I}}(x)$ and $P_{\mathbf{I}}(x)$ be the pdf and cdf of $\mathbf{I}$.
- Assume that $P_{\mathbf{I}}^{-1}(x)$ exists.
- Define the histogram equalized image $\mathbf{J}$ by

$$\mathbf{J} = P_{\mathbf{I}}(\mathbf{I}) \quad \left(\mathrm{J}(m,n) = P_{\mathbf{I}}[\mathrm{I}(m,n)] \ \forall \ (m,n) \right)$$

- **Claim: J** has a flat histogram, i.e.,

$$p_{\mathbf{J}}(x) = 1$$

Proof: Since $p_{\mathbf{I}}(x) = \dfrac{d}{dx} P_{\mathbf{I}}(x),$ we have from the fundamental theorem of calculus that

$$\mathrm{J}(m,n) = P_{\mathbf{I}}[\mathrm{I}(m,n)] = \int_{-\infty}^{\mathrm{I}(m,n)} p_{\mathbf{I}}(\theta)\, d\theta.$$

Now, the cdf of image $\mathbf{J}$ is given by

$$\begin{aligned} P_{\mathbf{J}}(x) &= \Pr\{\mathrm{J}(m,n) \le x\} \\ &= \Pr\{P_{\mathbf{I}}[\mathrm{I}(m,n)] \le x\}. \end{aligned} \qquad (1)$$

If we do a one-to-one positive semidefinite mapping to both sides of the inequality in (1), it will not change any of the probabilities. Since $P_{\mathbf{I}}^{-1}(x)$ is such a mapping, we can rewrite (1) as

$$\begin{aligned} P_{\mathbf{J}}(x) &= \Pr\{P_{\mathbf{I}}^{-1}[P_{\mathbf{I}}[\mathrm{I}(m,n)]] \le P_{\mathbf{I}}^{-1}(x)\} \\ &= \Pr\{\mathrm{I}(m,n) \le P_{\mathbf{I}}^{-1}(x)\}. \end{aligned} \qquad (2)$$

To make this easier to understand, let's temporarily write ☺ for $P_{\mathbf{I}}^{-1}(x)$.

Then (2) becomes $P_{\mathbf{J}}(x) = \Pr\{\mathrm{I}(m,n) \leq ☺\}$.

But from the definition of the cdf on p. 29, this is the same as

$$P_{\mathbf{J}}(x) = P_{\mathbf{I}}(☺).$$

Plugging back in the definition of ☺ from p. 32, we have

$$P_{\mathbf{J}}(x) = P_{\mathbf{I}}[P_{\mathbf{I}}^{-1}(x)] = x.$$

Since the pdf is the derivative of the cdf, we have then that

$$p_{\mathbf{J}}(x) = \frac{d}{dx} P_{\mathbf{J}}(x) = \frac{d}{dx} x = 1.$$

So $p_{\mathbf{J}}(x) = 1$ and the histogram of $\mathbf{J}$ is flat, as claimed. ■

Histogram Shaping

- More generally, suppose we would like to transform the image $\mathbf{I}$ into a new image $\mathbf{J}$ with pdf $p_{\mathbf{J}}(x)$ and cdf $P_{\mathbf{J}}(x)$ that are given by some particular desired functions $q(x)$ and $Q(x)$ with $q(x) = \mathrm{d}Q(x)/\mathrm{d}x$.
- This is called **histogram shaping**.
- Histogram equalization (flattening) is the special case where $q(x) = 1$ and $Q(x) = x$
  - which also means that $Q^{-1}(x) = Q(x) = x$.

- For **arbitrary** $q(x)$ and $Q(x)$, the histogram shaping algorithm is given by

$$\mathbf{J} = Q^{-1}[P_{\mathbf{I}}(\mathbf{I})]$$

$$\left(\mathrm{J}(m,n) = Q^{-1}\{P_{\mathbf{I}}[\mathrm{I}(m,n)]\} \ \forall \ (m,n)\right)$$

- This works since the cdf of $\mathbf{J}$ is:

$$\begin{aligned}\Pr\{\mathrm{J}(m,n) \le x\} &= \Pr\{Q^{-1}[P_{\mathbf{I}}[\mathrm{I}(m,n)]] \le x\} \\ &= \Pr\{P_{\mathbf{I}}[\mathrm{I}(m,n)] \le Q(x)\} \\ &= \Pr\{\mathrm{I}(m,n) \le P_{\mathbf{I}}^{-1}[Q(x)]\} \\ &= P_{\mathbf{I}}\{P_{\mathbf{I}}^{-1}[Q(x)]\} = Q(x)\end{aligned}$$

- For a practical digital image $\mathbf{I}$ with a discrete histogram, all of this can only be **approximated**.

Practical Histogram Flattening

- To approximately **flatten** the histogram of the digital image $\mathbf{I}$:
- Define the **cumulative histogram image**

$$\mathbf{J} = P_{\mathbf{I}}(\mathbf{I})$$

so that

$$\mathrm{J}(m,n) = P_{\mathbf{I}}[\mathrm{I}(m,n)] = \sum_{k=0}^{\mathrm{I}(m,n)} p_{\mathbf{I}}(k)$$

- This is the cumulative histogram of image $\mathbf{I}$ evaluated at the gray level of pixel (m,n).

Practical Histogram Flattening

- Note that

$$0 \leq \mathrm{J}(m,n) \leq 1$$

- The elements of $\mathbf{J}$ are **approximately** linearly distributed between 0 and 1.
- Finally, let $\mathbf{K} = \mathrm{FSCS}(\mathbf{J})$ to get the histogram equalized image with full scale contrast.

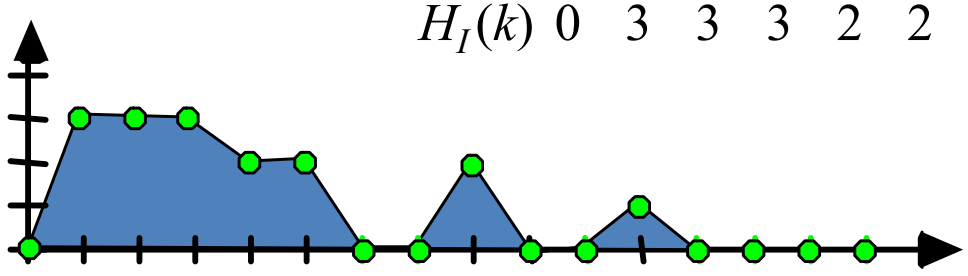
Histogram Flattening Example

- Let $\mathbf{I}$ be a 4×4 image with gray-level range $\{0, \ldots, 15\}$ (K-1 = 15):

$$\mathbf{I} = \begin{array}{|c|c|c|c|}\hline 1 & 1 & 3 & 4 \\ \hline 2 & 5 & 3 & 2 \\ \hline 8 & 1 & 8 & 2 \\ \hline 4 & 5 & 3 & 11 \\ \hline \end{array}$$

- The histogram:

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$H_I(k)$	0	3	3	3	2	2	0	0	2	0	0	1	0	0	0	0



- The normalized histogram…

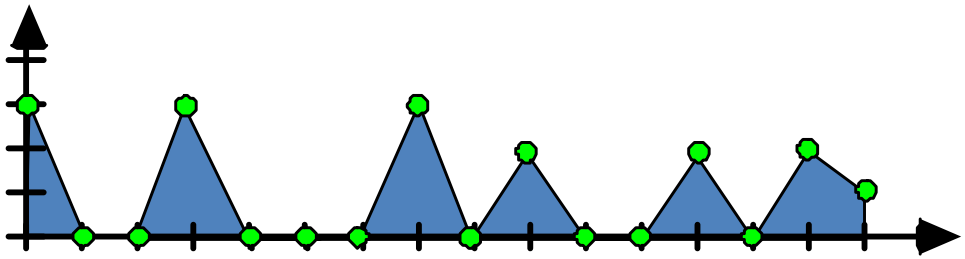
k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$p_I(k)$	0	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{2}{16}$	$\frac{2}{16}$	0	0	$\frac{2}{16}$	0	0	$\frac{1}{16}$	0	0	0	0
$P_I(k)$	0	$\frac{3}{16}$	$\frac{6}{16}$	$\frac{9}{16}$	$\frac{11}{16}$	$\frac{13}{16}$	$\frac{13}{16}$	$\frac{13}{16}$	$\frac{15}{16}$	$\frac{15}{16}$	$\frac{15}{16}$	$\frac{16}{16}$	$\frac{16}{16}$	$\frac{16}{16}$	$\frac{16}{16}$	$\frac{16}{16}$

- The equalized image **J** is computed and contrast stretched to obtain the final result **K** (as on p. 3.15 and after **rounding**):

$$\mathbf{J} = \begin{bmatrix} 3/16 & 3/16 & 9/16 & 11/16 \\ 6/16 & 13/16 & 9/16 & 6/16 \\ 15/16 & 3/16 & 15/16 & 6/16 \\ 11/16 & 13/16 & 9/16 & 16/16 \end{bmatrix} \qquad \mathbf{K} = \begin{bmatrix} 0 & 0 & 7 & 9 \\ 3 & 12 & 7 & 3 \\ 14 & 0 & 14 & 3 \\ 9 & 12 & 7 & 15 \end{bmatrix}$$

- The new **flattened** histogram looks like this:

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$H_I(k)$	3	0	0	3	0	0	0	3	0	2	0	0	2	0	2	1



- This is an **approximation** of the continuous algorithm.
- What it basically does is move the bins around to space them out more equally – making the histogram **more flat**.
- Because of the rounding, some bins may also get combined.
- The **spaces** that appear between bins in the new histogram are characteristic of equalization; for a typical image like mandrill, the new histogram will often have a "comb-like" appearance.

Histogram Shaping

- The idea is to create a new image $\mathbf{K}$ with an approximate **specified** histogram shape, such as a triangle or bell-shaped curve.

- Let $H_{\mathbf{K}}(k)$ be the **desired** histogram shape, with corresponding normalized values (probabilities) $p_{\mathbf{K}}(k)$.

Histogram Shaping

- Define the cumulative histogram image as before:

$$\mathrm{J}(m,n) \;=\; P_{\mathbf{I}}\big[\mathrm{I}(m,n)\big] \;=\; \sum_{k=0}^{\mathrm{I}(m,n)} p_{\mathbf{I}}(k)$$

- Also define the cumulative probabilities

$$P_{\mathbf{K}}(r) \;=\; \sum_{k=0}^{r} p_{\mathbf{K}}(k)$$

Histogram Shaping Algorithm

- Let $r(m,n)$ denote the **minimum value** of r such that

$$P_{\mathbf{K}}(r) \geq \mathrm{J}(m,n)$$

- Then take $\mathrm{K}(m,n) = r(m,n)$.
- This is a **convention** for approximating

$$\mathrm{K}(m,n) = P_{\mathbf{K}}^{-1}[\mathrm{J}(m,n)]$$

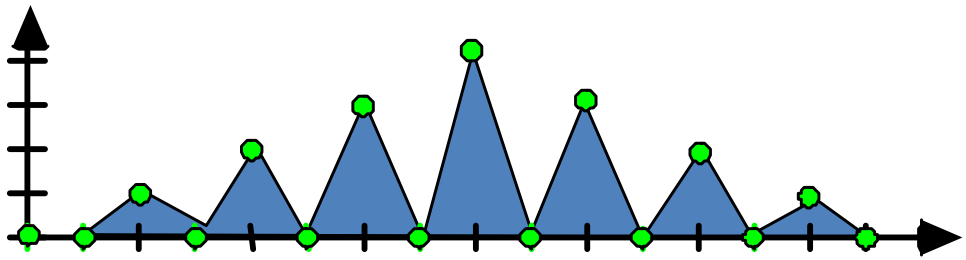
Histogram Shaping Example

- Consider the same image as in the last example. We had

$$\mathbf{I} = \begin{array}{|c|c|c|c|}\hline 1 & 1 & 3 & 4 \\ \hline 2 & 5 & 3 & 2 \\ \hline 8 & 1 & 8 & 2 \\ \hline 4 & 5 & 3 & 11 \\ \hline \end{array} \qquad \mathbf{J} = \begin{array}{|c|c|c|c|}\hline 3/16 & 3/16 & 9/16 & 11/16 \\ \hline 6/16 & 13/16 & 9/16 & 6/16 \\ \hline 15/16 & 3/16 & 15/16 & 6/16 \\ \hline 11/16 & 13/16 & 9/16 & 16/16 \\ \hline \end{array}$$

- Fit this to a (triangular) histogram:

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$H_{\mathbf{K}}(k)$	0	0	1	0	2	0	3	0	4	0	3	0	2	0	1	0
$p_{\mathbf{K}}(k)$	0	0	$\frac{1}{16}$	0	$\frac{2}{16}$	0	$\frac{3}{16}$	0	$\frac{4}{16}$	0	$\frac{3}{16}$	0	$\frac{2}{16}$	0	$\frac{1}{16}$	0



- The cumulative probabilities:

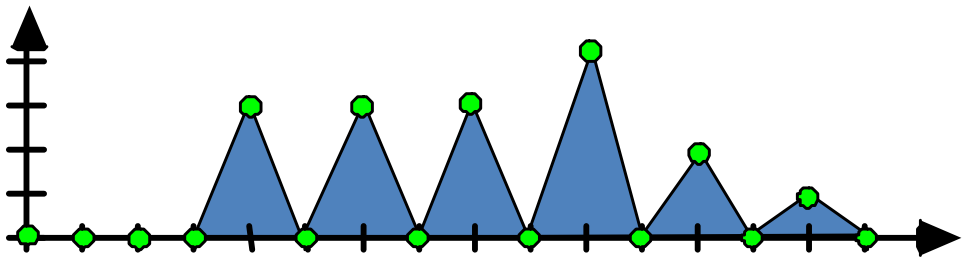
r	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$P_{\mathbf{K}}(r)$	0	0	$\frac{1}{16}$	$\frac{1}{16}$	$\frac{3}{16}$	$\frac{3}{16}$	$\frac{6}{16}$	$\frac{6}{16}$	$\frac{10}{16}$	$\frac{10}{16}$	$\frac{13}{16}$	$\frac{13}{16}$	$\frac{15}{16}$	$\frac{15}{16}$	$\frac{16}{16}$	$\frac{16}{16}$

- Using this table, a **careful** visual inspection of $\mathbf{J}$ then lets us form the new image $\mathbf{K}$.

$$\mathbf{K} = \begin{array}{|c|c|c|c|} \hline 4 & 4 & 8 & 10 \\ \hline 6 & 10 & 8 & 6 \\ \hline 12 & 4 & 12 & 6 \\ \hline 10 & 10 & 8 & 14 \\ \hline \end{array}$$

- Here's the new histogram:

k	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$H_{\mathbf{K}}(k)$	0	0	0	0	3	0	3	0	3	0	4	0	2	0	1	0



Histogram Matching

- A **special case** of histogram shaping.
- The histogram of image $\mathbf{I}$ is matched to the histogram of **another** image $\mathbf{I}'$.
- The procedure is identical, once the cumulative probabilities are computed for the model image $\mathbf{I}'$.
- Useful application: **Comparing** similar images of the same scene obtained under different conditions (e.g., lighting, time of day). Extends the concept of equalizing AOD described earlier.

Arithmetic Image Operations

- Suppose we have two $M \times N$ images $\mathbf{I}_1$ and $\mathbf{I}_2$. The basic arithmetic operations are:

Pointwise Addition

$$\mathbf{J} = \mathbf{I}_1 + \mathbf{I}_2 \text{ ; if } \mathrm{J}(m,n) = \mathrm{I}_1(m,n) + \mathrm{I}_2(m,n)$$

Pointwise Subtraction

$$\mathbf{J} = \mathbf{I}_1 - \mathbf{I}_2 \text{ ; if } \mathrm{J}(m,n) = \mathrm{I}_1(m,n) - \mathrm{I}_2(m,n)$$

Pointwise Multiplication

$$\mathbf{J} = \mathbf{I}_1 \otimes \mathbf{I}_2 \text{ ; if } \mathrm{J}(m,n) = \mathrm{I}_1(m,n) \times \mathrm{I}_2(m,n)$$

Pointwise Division

$$\mathbf{J} = \mathbf{I}_1 \Delta \mathbf{I}_2 \text{ ; if } \mathrm{J}(m,n) = \mathrm{I}_1(m,n) / \mathrm{I}_2(m,n)$$

- The operations $\otimes$ and Δ are mainly useful when manipulating **Fourier Transform matrices**.

Applying Arithmetic Operations

- We will look at two simple but **important** applications of arithmetic operations on images:
  - Frame averaging for noise reduction
  - Frame differencing for motion detection
- Note: the individual images in a video sequence are usually called **frames**.

Frame Averaging for Noise Reduction

- An image $\mathbf{J}$ is often corrupted by **additive noise**:
  - Surface radiation scatter
  - Noise in the camera
  - Thermal noise in a computer circuit
  - Channel transmission noise
- **Model:** The received image $\mathbf{J}$ is the sum of a **desired** image $\mathbf{I}$ and an **undesired** noise image $\mathbf{N}$:

$$\mathbf{J} = \mathbf{I} + \mathbf{N},$$

- the elements $\mathrm{N}(m,n)$ of $\mathbf{N}$ are **random variables**.

Zero-Mean Noise Images

- We won't explore the mathematics of 2D random processes in any detail yet.
- For now, just assume that each pixel $\mathrm{N}(m,n)$ is a random variable and that they all have **zero mean**.
- A sequence of noise images $\mathbf{N}_1, \mathbf{N}_2, \ldots, \mathbf{N}_M$ can be considered as a set of realizations of these random variables… just like repeatedly tossing a coin can be considered as realizations of a single random variable.
- If M is large enough, then the time average of each pixel (m,n) is expected to converge towards the mean of $\mathrm{N}(m,n)$, which is zero.
- So pointwise time averaging of the noise images $\mathbf{N}_i$ should give

$$\frac{1}{M}\sum_{i=1}^{M}\mathbf{N}_i \approx \mathbf{0} \quad \text{(image of zeroes)}$$

Frame Averaging for Noise Reduction

- Suppose we acquire M images $\mathbf{J}_1, \ldots, \mathbf{J}_M$ of the **same scene**

  in rapid succession, so that there is **little motion** between frames;

  or there is **no motion** in the scene.

- However, the frames are **noisy:**

$$\mathbf{J}_i = \mathbf{I}_i + \mathbf{N}_i \text{ for } i = 1, \ldots, M.$$

Frame Averaging for Noise Reduction

- Suppose we **average** the frames:

$$\mathbf{J} = \frac{1}{M}\sum_{i=1}^{M}\mathbf{J}_i = \frac{1}{M}\sum_{i=1}^{M}\left[\mathbf{I}_i + \mathbf{N}_i\right] = \frac{1}{M}\sum_{i=1}^{M}\mathbf{I}_i + \frac{1}{M}\sum_{i=1}^{M}\mathbf{N}_i$$

- Since $\mathbf{I}_1 \approx \cdots \approx \mathbf{I}_M \approx \mathbf{I}$, we have that

$$\frac{1}{M}\sum_{i=1}^{M}\mathbf{I}_i \approx \mathbf{I} \quad \text{and also} \quad \frac{1}{M}\sum_{i=1}^{M}\mathbf{N}_i \approx \mathbf{0}$$

Frame Averaging for Noise Reduction

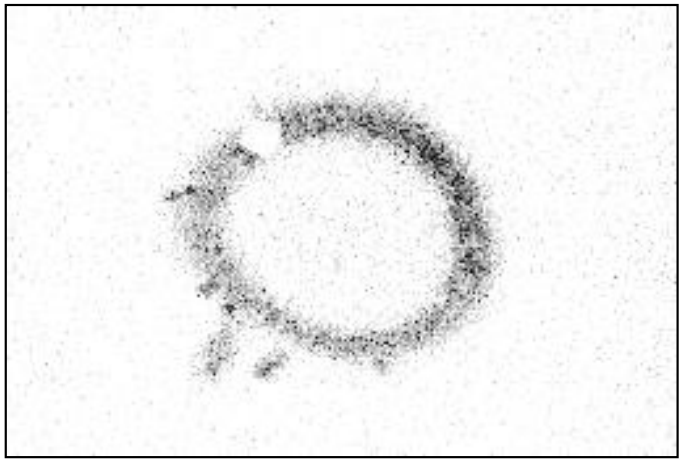
- So

$$\mathbf{J} = \frac{1}{M}\sum_{i=1}^{M}\mathbf{I}_i + \frac{1}{M}\sum_{i=1}^{M}\mathbf{N}_i \approx \mathbf{I} + \mathbf{0} = \mathbf{I}$$

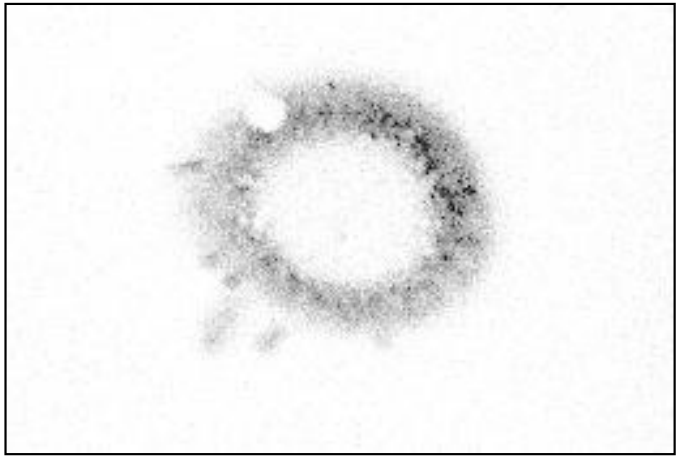
if enough frames (M) are averaged together.

Frame Averaging Example

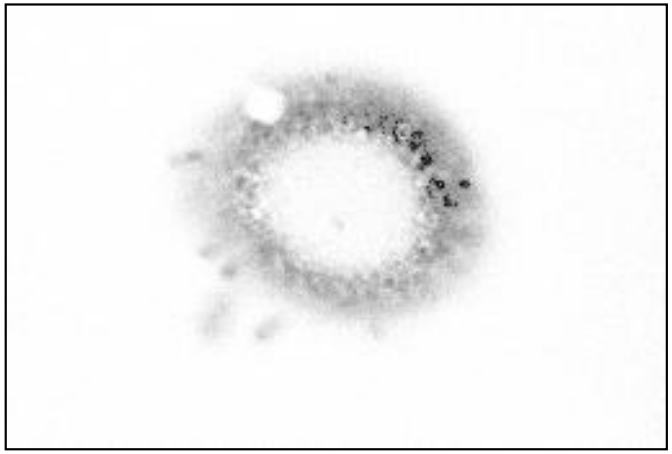
Macroalga Valonia microphysa, imaged with laser scanning confocal microscope (LSCM). The ring is chlorophyll fluorescing under Ar laser excitation.



Single frame (no averaging)



Average of four frames



Average of sixteen frames.

Motion Detection by Frame Differencing

- Often it is of interest to detect **object motion** between frames.
- **Many applications**: video compression, target recognition, tracking, security, surveillance, automated inspection, etc.
- Frame differencing is a **simple** but **effective** approach.

Motion Detection by Frame Differencing

- Let $\mathbf{I}_1, \mathbf{I}_2$ be consecutive frames in close time proximity, e.g., from a video camera.
- Assumptions: there is a moving object and the background doesn't change much.
- Form the absolute difference image.

$$\mathbf{J} = |\mathbf{I}_1 - \mathbf{I}_2|$$

- The background is approximately subtracted out.
- Large values typically remain at both the old **and** new locations of the object.
- Presence/absense of motion can be detected by thresholding $\mathbf{J}$.

Geometric Image Operations

- Certain geometric image operations are widely used in image processing.
- Many concepts also overlap **computer graphics**.
- **Image processing** is primarily concerned with **correcting** or **improving** images of the **real world**.
- **Computer graphics** is primarily concerned with **creating** images of an **unreal world**.

Geometric Image Operations

- In a sense, geometric image operations are the **opposite** of point operations: they modify **spatial positions** but not **gray levels**.
- A geometric operation generally requires **two steps:**

  - A spatial mapping of image coordinates

  - Interpolation

Spatial Mapping

- The image coordinates are mapped to create the new image:

$$\mathrm{J}(m,n) = \mathrm{I}(m',n') = \mathrm{I}[\mathrm{a}(m,n), \mathrm{b}(m,n)]$$

- But the coordinates $\mathrm{a}(m,n)$ and $\mathrm{b}(m,n)$ **might not be integers!**
- For example if

$$\mathrm{a}(m,n) = m/3.5 \text{ and } \mathrm{b}(m,n) = n/4.5$$

then $\mathrm{J}(m,n) = \mathrm{I}(m/3.5, n/4.5)$, which are undefined image coordinates!

Interpolation

- It is necessary to **interpolate** non-integer coordinates $\mathrm{a}(m,n)$ and $\mathrm{b}(m,n)$ to integer values.
- We will look at two methods:
  - Nearest neighbor interpolation
  - Bilinear interpolation

Nearest Neighbor Interpolation

- Simple-minded.
- The geometrically transformed coordinates are mapped to the **nearest integer coordinates**:

$$J(m,n) = I\{\mathrm{INT}[a(m,n)+0.5], \mathrm{INT}[b(m,n)+0.5]\}$$

- **Serious drawback**: Pixel replication can occur, creating a "jagged" edge effect in non-smooth regions.

Nearest Neighbor Interpolation

Caveat

- If for some coordinate (m,n) either

$$\text{INT}[\text{a}(m,n)+0.5] < 0 \quad \text{or} \quad \text{INT}[\text{b}(m,n)+0.5] < 0$$

or

$$\text{INT}[\text{a}(m,n)+0.5] > \text{N-1} \quad \text{or} \quad \text{INT}[\text{b}(m,n)+0.5] > \text{N-1}$$

then

$$\text{J}(m,n) = \text{I}\{\text{INT}[\text{a}(m,n)+0.5], \text{INT}[\text{b}(m,n)+0.5]\}$$

is not defined.

- We usually set $\text{J}(m,n) = 0$ in these cases.

Bilinear Interpolation

- Produces a **much smoother** interpolation than nearest neighbor approach.
- Given four neighboring image coordinates $\mathrm{I}(m_0,n_0)$, $\mathrm{I}(m_1,n_1)$, $\mathrm{I}(m_2,n_2)$, and $\mathrm{I}(m_3,n_3)$, the new image $\mathrm{J}(m,n)$ is computed as

$$\mathrm{J}(m,n) = \mathrm{A}_0 + \mathrm{A}_1 \cdot m + \mathrm{A}_2 \cdot n + \mathrm{A}_3 \cdot m \cdot n$$

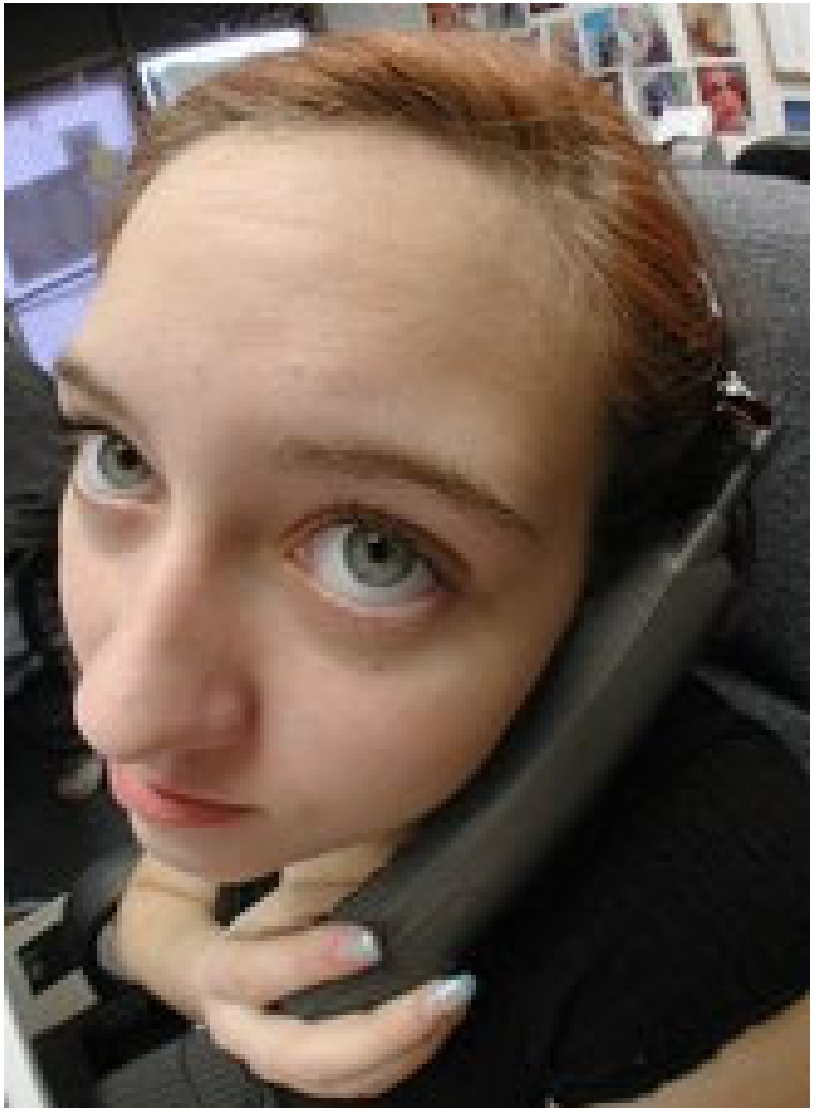
where the **bilinear weights** A_0, A_1, A_2, A_3 are found by solving

$$\begin{bmatrix} \mathrm{A}_0 \\ \mathrm{A}_1 \\ \mathrm{A}_2 \\ \mathrm{A}_3 \end{bmatrix} = \begin{bmatrix} 1 & m_0 & n_0 & m_0 n_0 \\ 1 & m_1 & n_1 & m_1 n_1 \\ 1 & m_2 & n_2 & m_2 n_2 \\ 1 & m_3 & n_3 & m_3 n_3 \end{bmatrix}^{-1} \begin{bmatrix} \mathrm{I}(m_0,n_0) \\ \mathrm{I}(m_1,n_1) \\ \mathrm{I}(m_2,n_2) \\ \mathrm{I}(m_3,n_3) \end{bmatrix}$$

A **linear combination** of the four closest values. The **best planar fit** to the four nearest values.

Basic Geometric Transformations

- The basic geometric transformations are
  - Translation
  - Rotation
  - Zoom



“Fish-Eye” geometric distortion
(more complex)

Translation

- The simplest geometric operation - requires no interpolation. Let

$$\mathrm{a}(m,n) = m - m_0, \qquad \mathrm{b}(m,n) = n - n_0$$

where (m_0, n_0) are **constants**. In this case

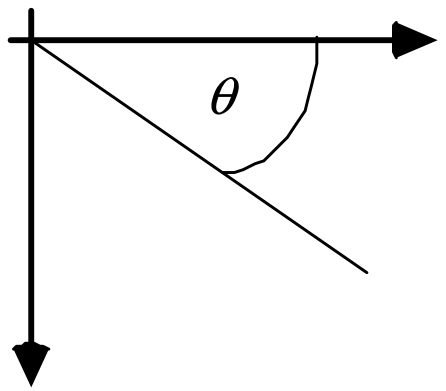
$$\mathrm{J}(m,n) = \mathrm{I}(m - m_0, n - n_0)$$

a shift of the image by amounts (m_0, n_0) in the (row, column) directions.

Rotation

- Clockwise rotation of an image by an angle θ relative to the horizontal axis is accomplished by the following transformation:

$$\mathrm{a}(m,n) = m\cos\theta - n\sin\theta$$
$$\mathrm{b}(m,n) = m\sin\theta + n\cos\theta$$



- Simplest cases:

$\theta = 90°$: $[\mathrm{a}(m,n), \mathrm{b}(m,n)] = (-n, m)$

$\theta = 180°$: $[\mathrm{a}(m,n), \mathrm{b}(m,n)] = (-m, -n)$

$\theta = -90°$: $[\mathrm{a}(m,n), \mathrm{b}(m,n)] = (n, -m)$

- A translation is **usually** require afterwards to obtain coordinate values in the nominal range, e.g.

$$(M\text{-}n, m) \;,\; (M\text{-}m, N\text{-}n) \;,\; (n, N\text{-}m)$$

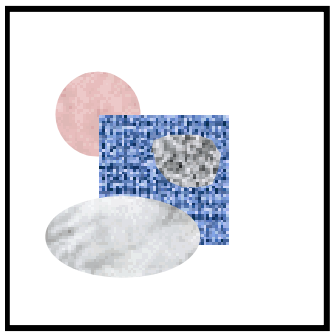
Zoom

- Zooming **magnifies** an image by the mapping functions

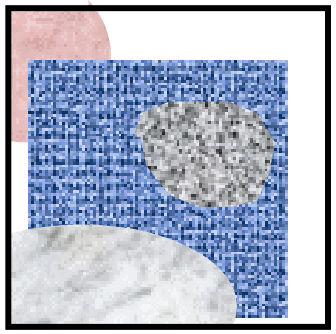
$$\mathrm{a}(m,n) = m/\mathrm{c} \quad \text{and} \quad \mathrm{b}(m,n) = n/\mathrm{d}$$

  where $\mathrm{c} \geq 1$ and $\mathrm{d} \geq 1$.

- For large magnifications, a zoom image will look "blotchy" if nearest neighbor interpolation is used. A bilinear interpolation works quite well.



original



2x zoomed

Multilayer Perceptrons

Neural Representations

- Now that we are processing **gray-scale images,** we return to **neural image representations.**
- **Perceptrons** are **limited** but are the basis for **extremely powerful** ways of **representing image data.**
- The key is **more layers.**

Multilayer Perceptrons (MLPs)

- Neural networks **began to fulfill their promise** with MLPs. Often called "vanilla" or "artificial neural networks."
- Operate by **layering perceptrons** in a **feed-forward** manner.
- Can learn to represent data that is **not linearly separable,** hence also **complex patterns,** including those in **images.**
- Uses more **powerful** and diverse **activation functions.**
- Can be **efficiently trained** via a method called **backpropagation.**

Multilayer Perceptrons

- **Add layers** between the **input layer** and the **output layer.** They will also consist of:
  - **weights**
  - **summation**
  - **activation function**

- The new **"hidden layers"** (invisible to input and output) can:
  - **potentially receive all inputs or all prior layer outputs**
  - **potentially feed all output layer or next layer nodes**

- Can **use different activation functions** than the **signum function** having better properties, like **differentiability.**

- Can easily have **multiple outputs.**

Activation Functions

- The **signum function** $\phi(x) = \text{sign}(x)$ is **discontinuous.** Other functions are possible that can **limit** or **rectify neural responses.**

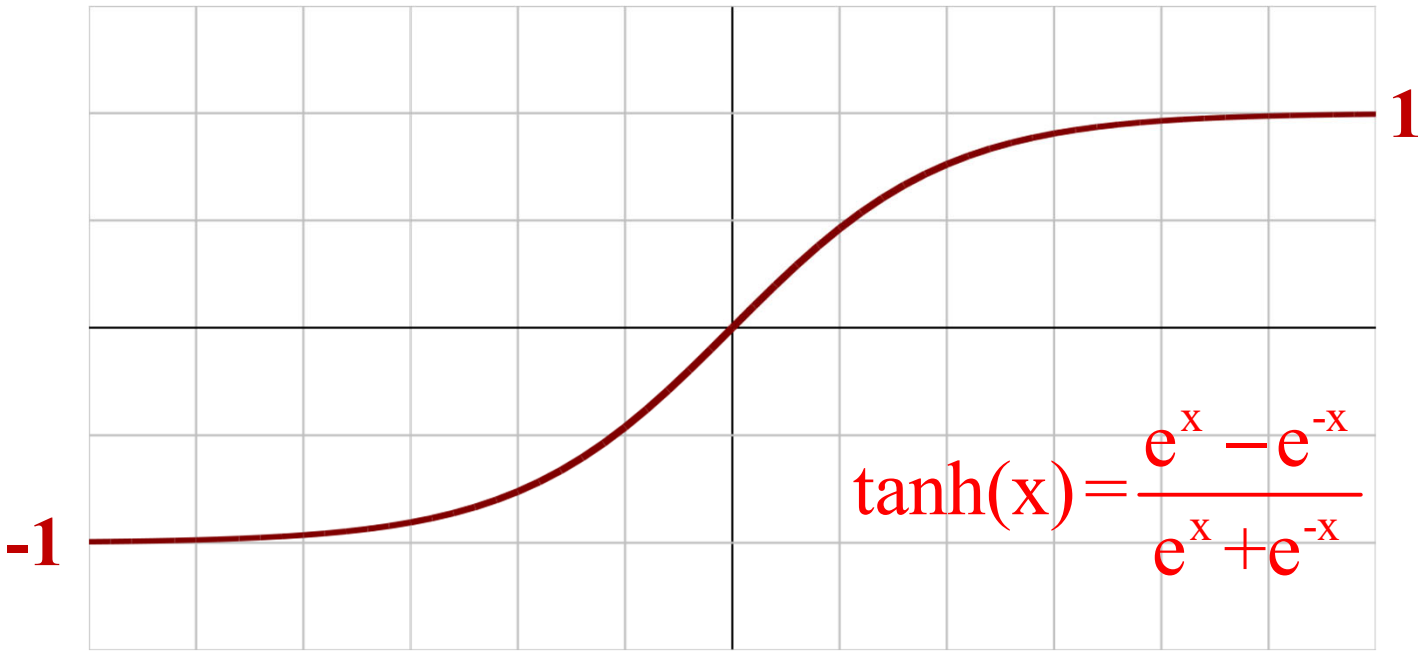
- They are **not binary,** which affords **greater freedom** (for regression, instead of just classification).

- The two most **popular activation functions** used with ANNs (before the "deep" era) were the **tanh** and **logistic functions** which are **sigmoid** (s-shaped) **functions.**

$$\phi(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\phi(x) = \text{logistic}(x) = \frac{1}{1 + e^{-x}}$$

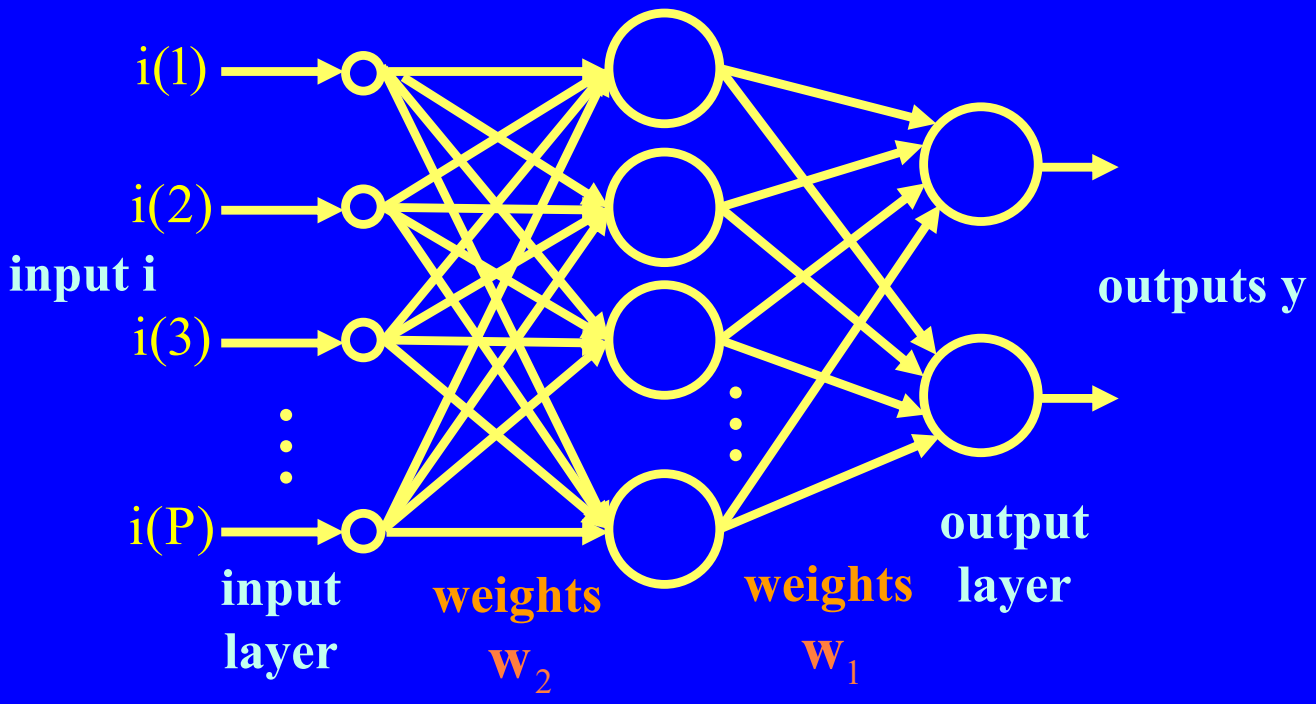
- Both **limit** and are **differentiable.**


1
-1
$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$


1
0
$\text{logistic}(x) = \frac{1}{1 + e^{-x}}$

Multilayer Perceptron Diagram

- This **example** has two outputs and is **fully connected. Every node** in a layer **feeds every node** in the next layer.



- **Each large node** is of the form $\phi\left[\sum_{r=1}^{R_q} w_q(r) i_q(r)\right]$ where i_q are the R_q inputs to a q^{th} layer with weights $\mathbf{w}_q$ and activation function ϕ.

Still Too Much Computation

- … to operate on pixels directly (back then).
- **Huge computation:**
  - P^2 weights in w_2
  - Only 2P in w_1, but there can be as many outputs as pixels.
- A 1024x1024 image ($P=2^{20}$) implies **2^{40} weights** in w_2!*
- Clearly **untenable.** The number of **nodes,** or **connections** (or **both**) must somehow be greatly limited.
- Using **small images** obviously helps. A 128x128 image ($P= 2^{14}$) implies $P= 2^{28}$ weights in w_2. But that's **not the answer** either…

***Note:** $2^{40} \approx$ 1 Trillion, while $2^{28} \approx$ 268 Million

Features

- Instead, **highly informative features** would be extracted from images. **Trained** on **many images' features.**
- **Vastly fewer** than the number of pixels: often just **single digits,** or dozens.
- Could be simple **image statistics, Fourier** features, **wavelet/bandpass** filter data, regional **colors, "busyness"** and so on. **A lot of varieties** have been used.
- We'll talk about MANY later: **SIFT, SURF, LBP,** etc etc.
- These days they are typically called **"handcrafted"** as opposed to **"learned."**

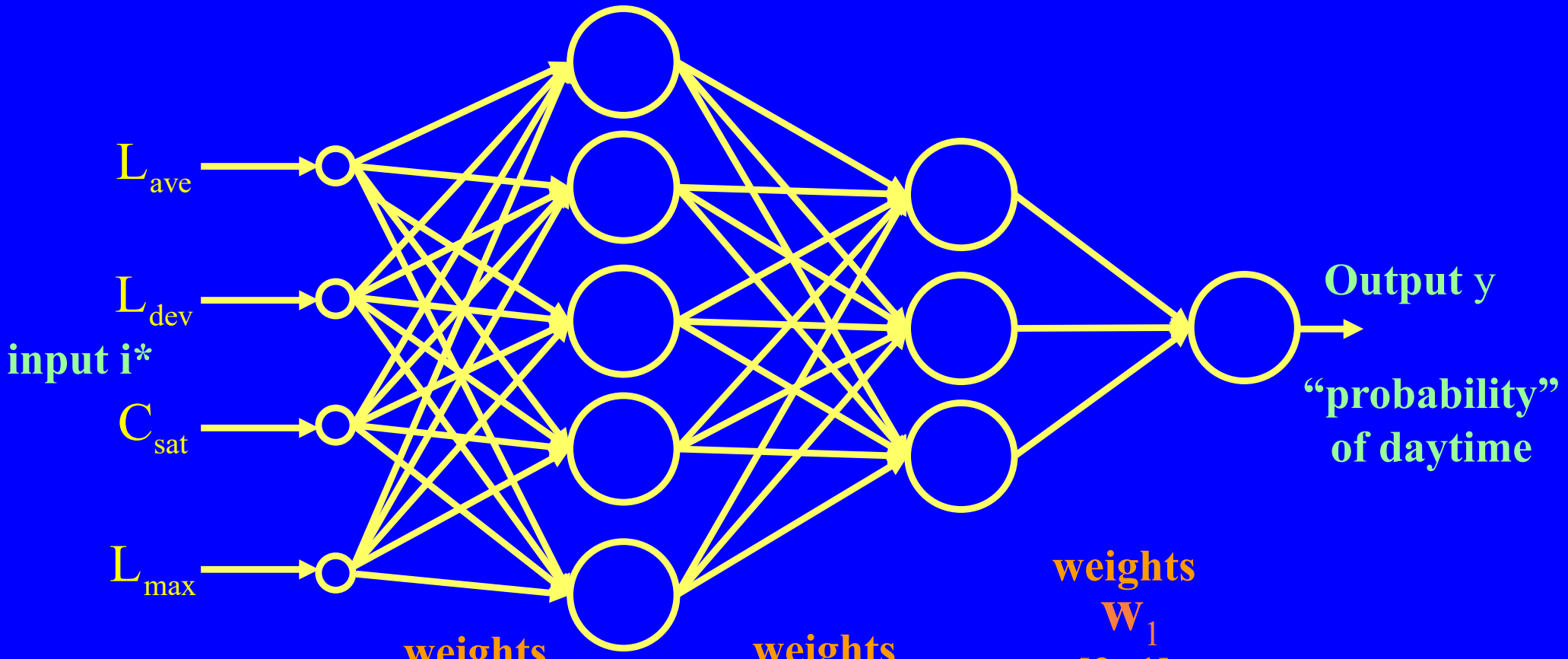
Example Feature-Driven MLP

Day vs Night Detector

- **Task:** Determine if an image was taken in **daytime** or **night time.**
- The following **intuitive features** might be **extracted** from each **training image:**
  - **Average luminance** (brightness) L_{ave} of the image
  - **Standard deviation of luminance** L_{dev} of the image
  - **Color saturation** C_{sat} of the image (how widely distributed is color)
  - **Maximum luminance** L_{max} of the image
- Are these the **"right" features?** They make sense, but **who knows?**
- **Normalization:** Usually the set of values of the features computed from each image in the training set is normalized to [0, 1] (e.g.) so no feature has an outsized effect (unless that's desired).

Day vs Night Detector Network

- A **fully connected** network**. Every node** in a layer **feeds every node** in the next layer. Output is **continuous** or **thresholded.**



*Still calling the input vector i

Training by Backpropagation

- Perhaps the **greatest advance** in the history of neural networks – except for the **Perceptron,** of course.
- "Backprop" is short for "backward propagation of errors."
- It means the **weights $w_1, \ldots, w_L$ of all layers** are adjusted to **minimize the training error** w.r.t. the known training labels (day/night, or whatever).
- The **activation function** ϕ needs to be **differentiable.**
- If $\phi(x) = \tanh(x)$, **then** $\phi'(x) = 1 - \tanh^2(x)$
- If $\phi(x) = \text{logistic}(x)$, **then** $\phi'(x) = e^{-x}\,\text{logistic}^2(x)$

Training by Backpropagation

- Given **training labels** $\mathbf{T} = \{t_p;\ 1 \le p \le P\}$ of a neural network with outputs $\mathbf{Z} = \{z_p;\ 1 \le p \le P\}$, form the **MSE loss function***

$$E = \frac{1}{2}\sum_{p=1}^{P}(z_p - t_p)^2$$

- Then for an **arbitrary neuron anywhere in the network** indexed k with output

$$y_k = \phi\left[\sum_{i=1}^{J} w_k(i)\, y_i\right]$$

- The goal is to **minimize the loss function** E over **all weights** $\mathbf{w_k} = \{w_k(j);\ 1 \le j \le J\}$ by gradient descent.

*The ½ is just to cancel a later term

Double Chain Rule

- **Differentiate** E w.r.t. each j^{th} weight

$$\frac{\partial E}{\partial w_k(j)} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial x_k}\frac{\partial x_k}{\partial w_k(j)}$$

$$y_k = \phi(x_k)$$

$$x_k = \sum_{i=1}^{J} w_k(i) y_i$$

- **Note** that

$$\frac{\partial x_k}{\partial w_k(j)} = y_j \qquad \frac{\partial y_k}{\partial x_k} = \phi'(x_k)$$

- If neuron k is in the **output layer** ($y_k = z_k$) then

$$\frac{\partial E(y_k)}{\partial y_k} = \frac{\partial E}{\partial z_k} = z_k - t_k$$

$$E = \frac{1}{2}\sum_{p=1}^{P}(z_p - t_p)^2$$

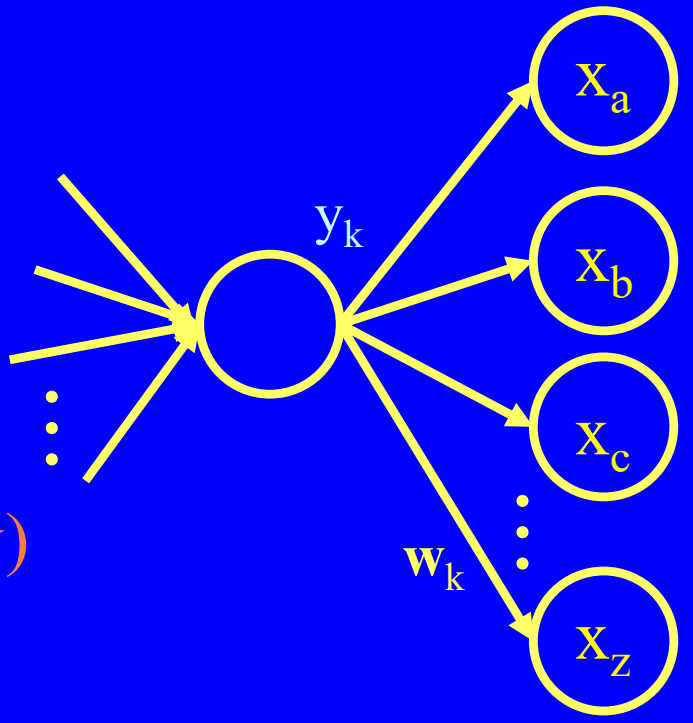
Hidden Layer Neuron

- Let neuron k be at **arbitrary location**. Then E is a function of **all neurons** **V** = {a, b, c, …, z} receiving input from k (in the next layer) and

$$\frac{\partial E(y_k)}{\partial y_k} = \frac{\partial E(x_a, x_b, ..., x_z)}{\partial y_k}$$

- Take the **total derivative:**

$$\frac{\partial E}{\partial y_k} = \sum_{v \in V} \frac{\partial E}{\partial x_v} \frac{\partial x_v}{\partial y_k} = \sum_{v \in V} \frac{\partial E}{\partial y_v} \frac{\partial y_v}{\partial x_v} w_k(v)$$



- **A recursion!** The derivative wrt y_k can be computed from the derivatives wrt the outputs $\{y_v\}$ of the **next layer.**

Putting it Together

- So **finally** for each output

$$\frac{\partial E}{\partial w_k(j)} = \delta_k y_j$$

where

$$\delta_k = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial x_k} = \begin{cases} \phi'(x_k)(y_k - t_k) & \text{(output neuron)} \\ \phi'(x_k)\sum_{v \in \mathbf{V}} w_k(v)\delta_v & \text{(hidden neuron)} \end{cases}$$

- Backprop proceeds by **gradient descent.** For a **learning rate** γ,*

$$\Delta w_k(j) = -\gamma \frac{\partial E}{\partial w_k(j)} = -\gamma \delta_k y_j$$

- We will see **other loss functions** and **activation functions** later.

*See Module 1, p. 121. Picking γ can be a trial-and-error process!

Training, Validation, and Testing

- MLPs/ANNs are tools to **construct algorithms that learn** from data (**training**) and **make predictions** (**testing**).

- **Universal Approximation Theorem** (Cybenko 1989): Even **a single-layer feed-forward MLP** can **approximate** any **continuous function** on a **compact* subset** of $\mathbf{R}^n$ **arbitrarily well, if** ϕ is a bounded, continuous function.

- Greatly suggests **potential** of MLPs to create models, but it is **no guarantee** that a given trained MLP is a **good model!**

- MLPs must be trained on **adequately sizable** and **representative data and** should be **validated/tested.**

*Closed and bounded

Training, Validation, and Testing

- **Basic process:** Given **training samples**

$$\mathbf{T} = \{(\mathbf{i}_1, t_1), (\mathbf{i}_2, t_2), \ldots, (\mathbf{i}_Q, t_Q)\}$$

- Feed **sequentially** to MLP optimizing by **backprop** via **gradient descent** (GD). Repeat epochs until **stopping criteria** is met (based on **loss**), possibly **randomly reordering** the samples (**stochastic GD or SGD**).

- **Once learned,** apply the model on a separate **validation set**

$$\mathbf{V} = \{(\mathbf{j}_1, v_1), (\mathbf{j}_2, v_2), \ldots, (\mathbf{j}_P, v_P)\}$$

on which **network parameters** are **tuned** (e.g. node density, #layers …) and a **stopping point** decided - e.g., avoid **overfitting** (too little data, usually).

- Finally, use a separate **test set** to measure the **performance** of the model. Can come from same large dataset but must be **separate from training and validation data.**

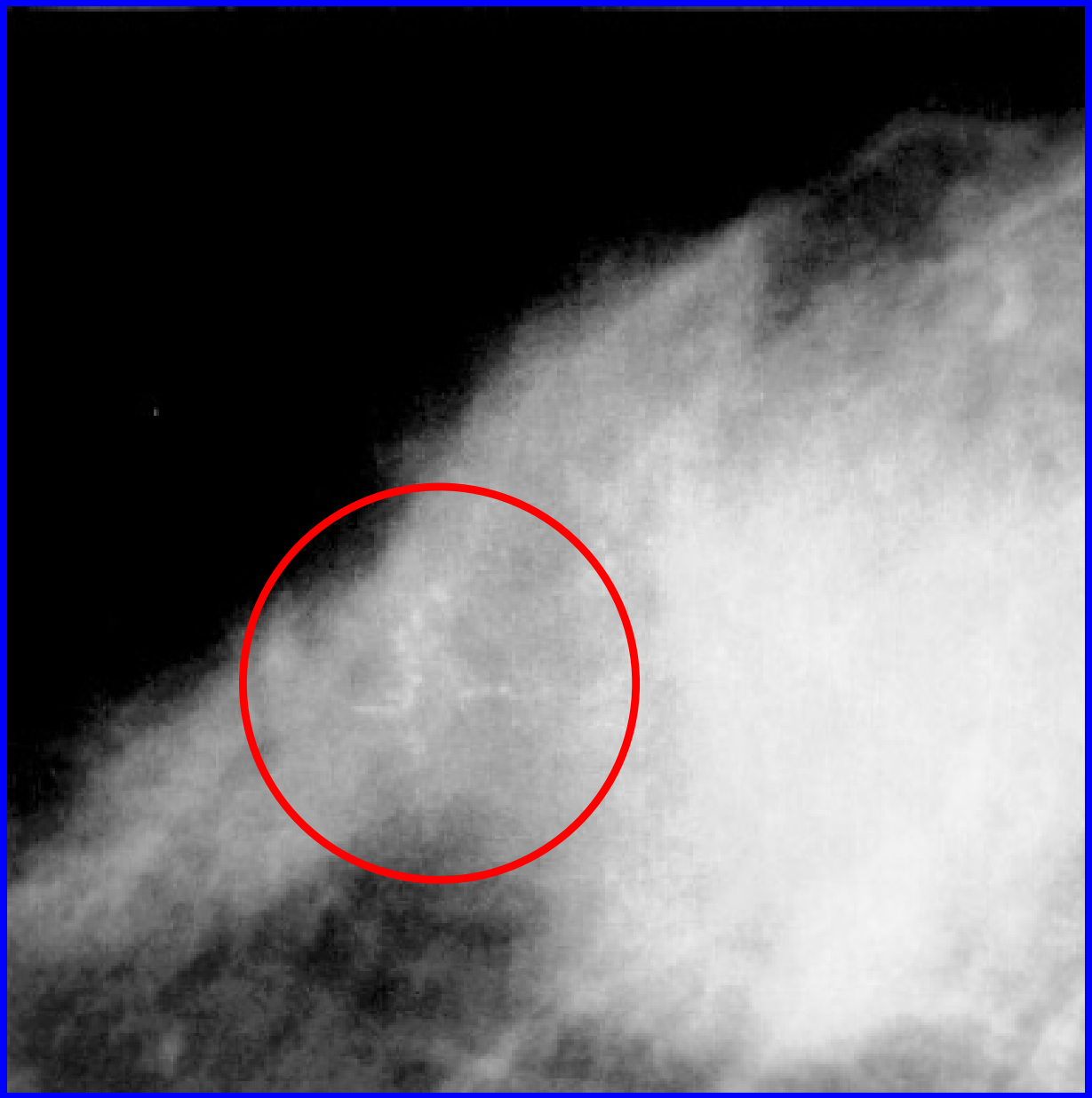
- Lastly, the **input features** are usually **normalized** to (say) [0, 1] so that the **feature range** will not **affect the results too much.**

Case Study: Classifying Microcalcifications

Classifying Microcalcifications

- Based on a paper by Chan et al (1997).
- **Idea:** use **texture features** on "regions of interest" (ROIs) of **mammograms** to train a MLP to classify as **benign** or **malignant.**
- Each **1024x1024 ROI** identified by a radiologist around a **microcalcification(s)**
- Great example of using a small number of features from a small number of images to **train a MLP classifier.**

Microcalcifications

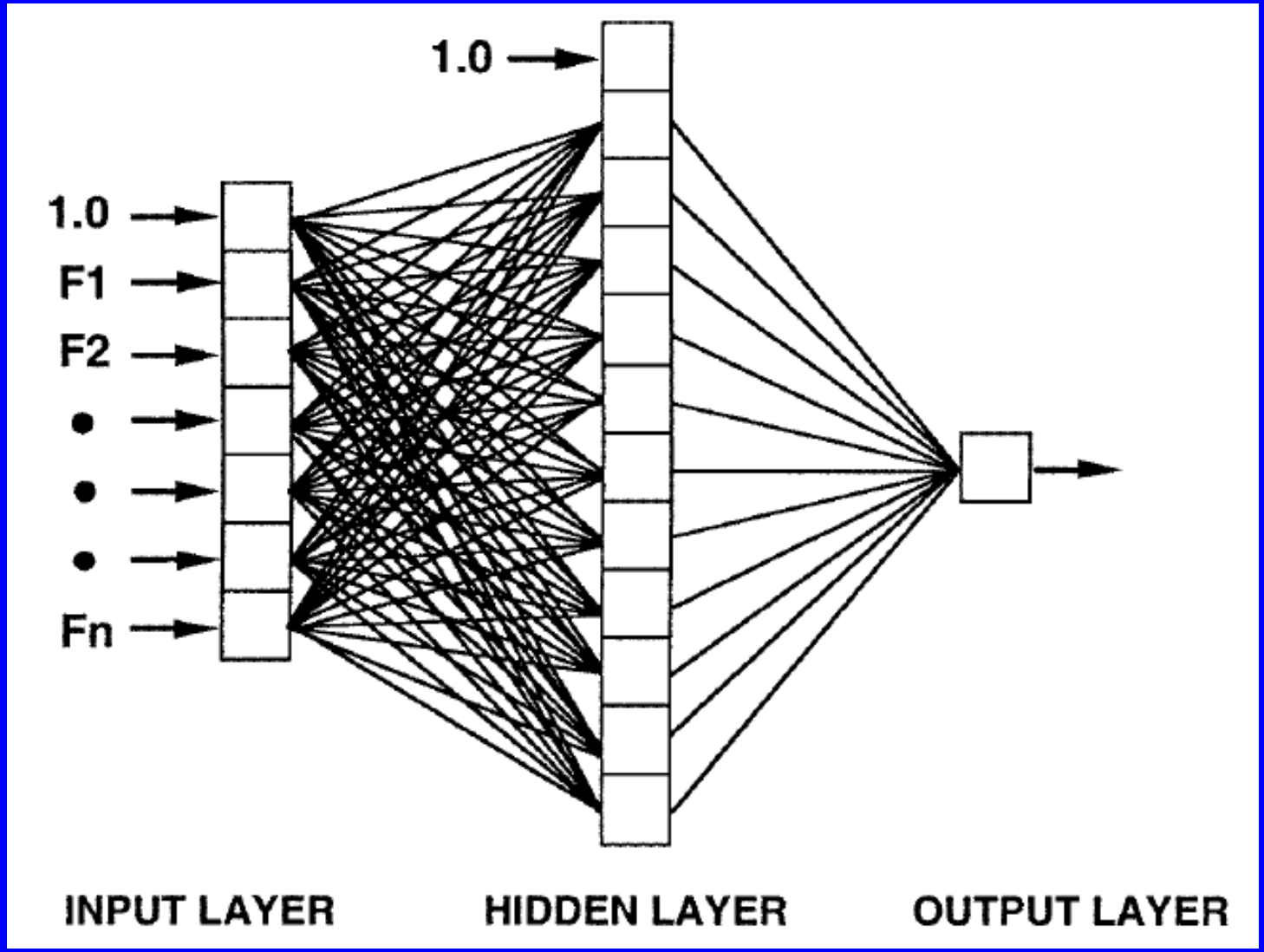


Microcalcifications in malignant breast tissue

Features

- 13 **texture features** (simple computed statistics) of ROIs:
  - sample correlation
  - sample entropy
  - sample energy
  - sample inertia
  - inverse difference moment
  - sum average
  - sum entropy
  - difference entropy
  - difference average
  - sum variance
  - difference variance
  - information measure of correlation 1
  - information measure of correlation 2

Training an MLP

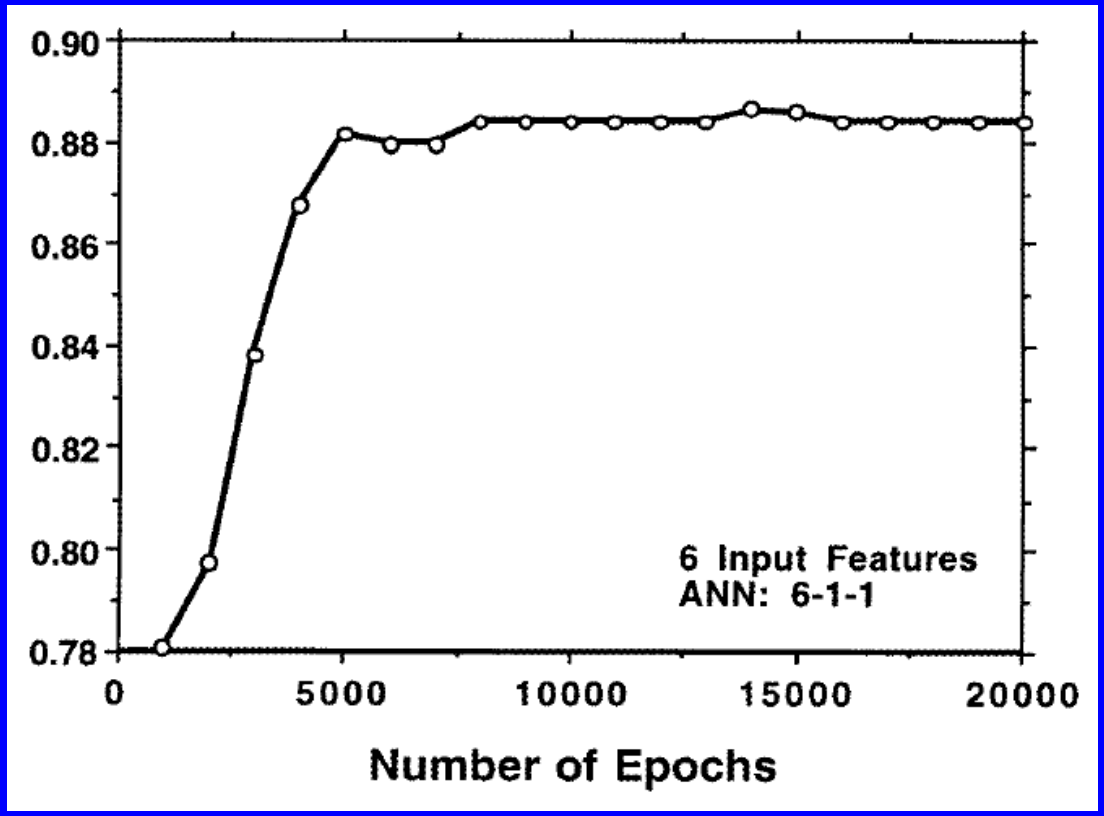


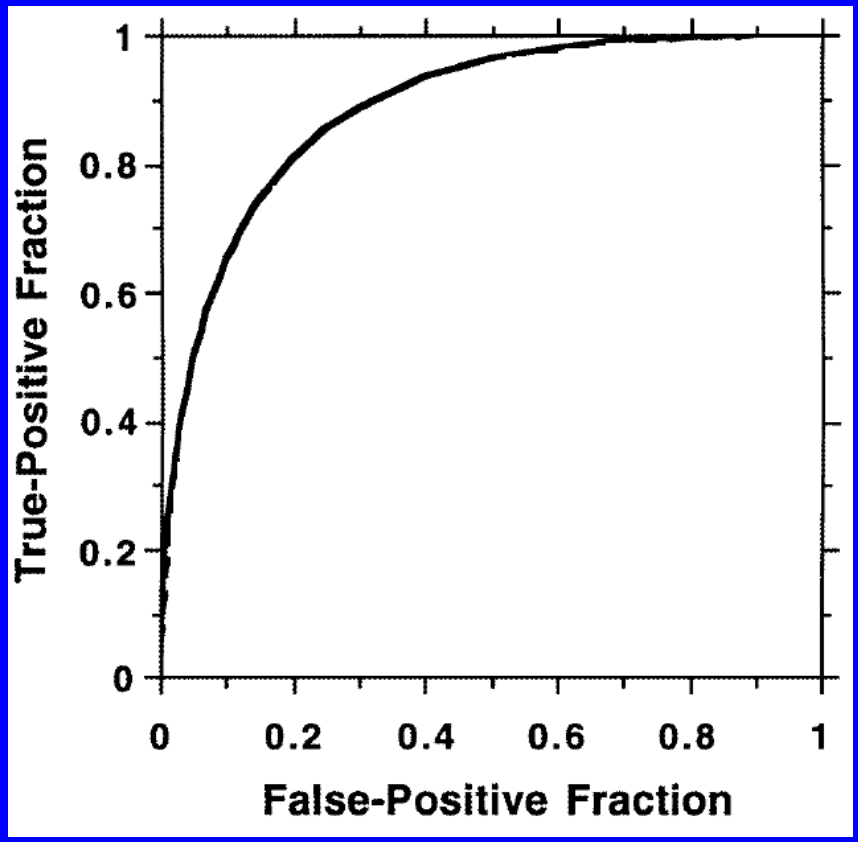
The MLP

Training, Validation, and Testing

- A total of **86 mammograms** from 54 cases were (**26 benign, 28 malignant**) were used. All recommended for surgical biopsy.
- In each training phase, MLP was trained on 85, tested on one. **Repeated 86 times.** Called **"leave one out"** training.
- MLP output was **thresholded** to **classify.**
- **26 of 26 malignant** cases identified (**100% sensitivity**)
- **11 of 28 benign** cases identified (**39% specificity**)

Convergence





Area under ROC

Final ROC

Many epochs needed before converging.

Note: ROC = Receiver Operating Characteristic is a plot of false positives vs true positives (or false negatives)

Let's Make a "Day-Night Classifier"

Day-Night Classifier

- Let's define **five simple features** on each image:

  LMAX = maximum luminance
  LMIN = minimum luminance
  LMED = median luminance
  LAVE = average luminance
  LSTD = standard deviation of luminance

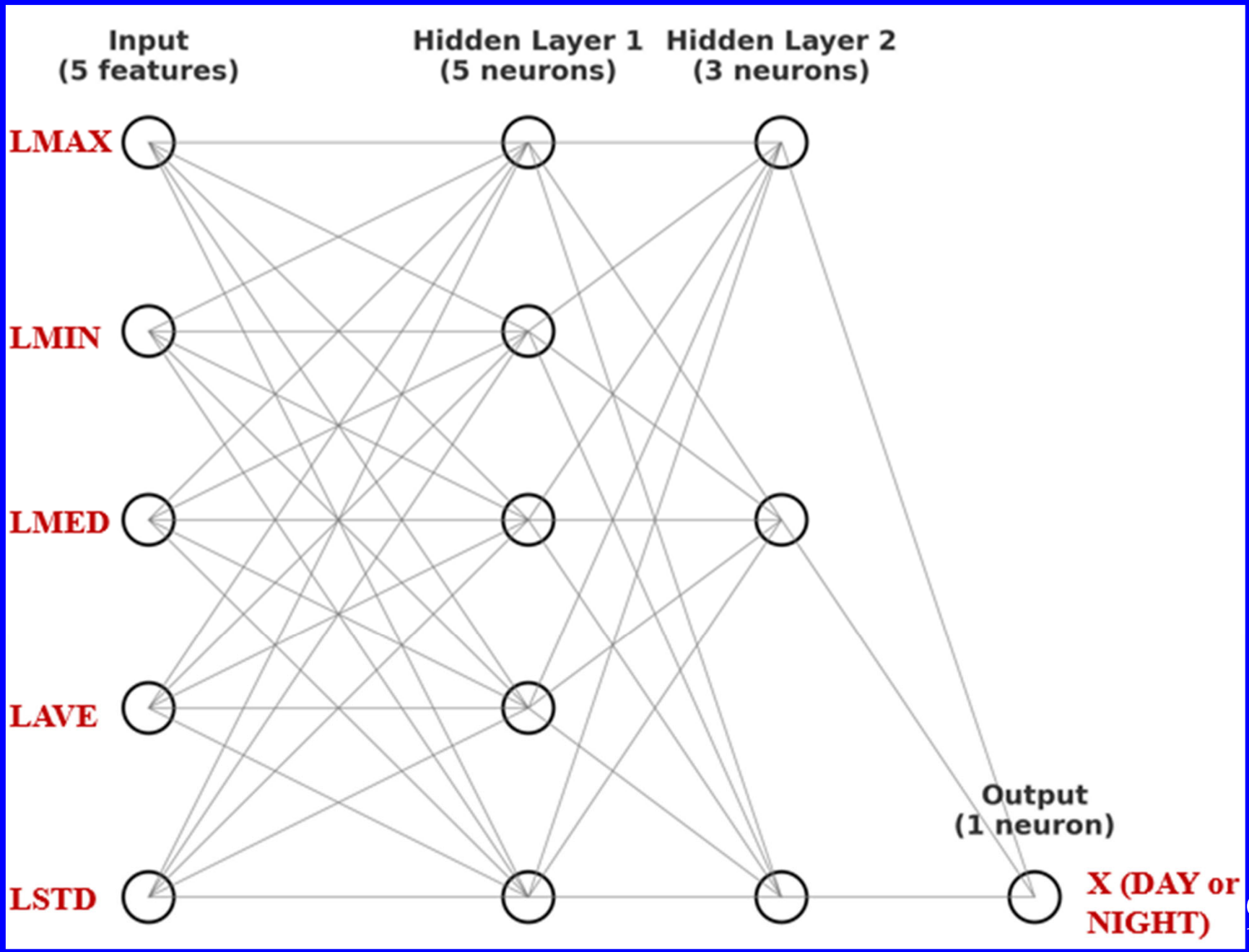
- Create a **three layer MLP** (52 learnable parameters)

  - **Input layer:** each feature feeding a neuron
  - **First layer:** 5 neurons with tanh activations (5 inputs x 5 outputs + 5 biases = 30)
  - **Second layer:** 5 neurons with tanh activations (5 inputs x 3 outputs + 3 biases = 18)
  - **Third layer:** 1 neuron with tanh activation (3 inputs x 1 output + 1 bias = 4)

- **Classification:** If output = X, then

  DAY if $X < 0$
  NIGHT if $X \geq 0$

Day-Night Classifier



$$BCE = \frac{1}{K}\sum_{k=1}^{K} -y_k \log(\hat{y}_k) - (1-y_k)\log(1-\hat{y}_k)$$

$y_k \in \{0,1\}$ are the true labels

$\hat{y}_k \in (0,1)$ is the predicted probability the label is 1 (map from tanh)

- We used a **public-domain dataset**
  - **The "Day-Night Dataset":** freely available here.
  - **Fairly sizeable:** 1719 images
    788 labeled as 'day' images
    931 labeled as 'night' images
  - You can **also find them** in a folder 'day_night_dataset' with subfolders 'day' and 'night' here.

- **Why train on the entire dataset?** A: For teaching purposes: we'll demo/test on entirely unrelated images!

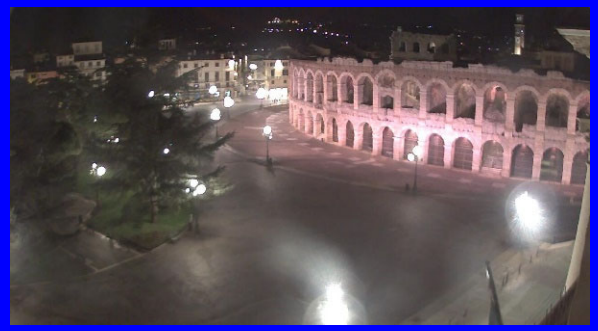
Example Day / Night Images

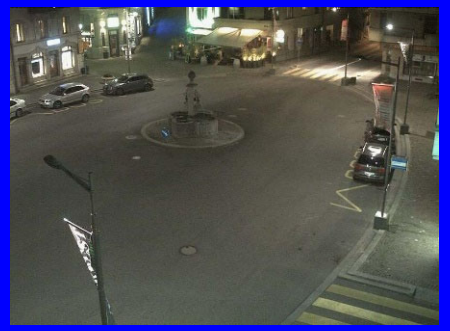










Each of the many contents was captured throughout the day/night.

The day images The night images

General Comments

- These models were interesting back then but **required too much computation** for large data.
- **Training** was a **huge** burden.
- **HENCE:** MLPs / ANNs remained a **topical interest** that **dwindled** through the 90's.
- However, **other methods** soon succeeded, and computation continued to accelerate. Today, MLPs are often integral parts of very large models.

Comments

- We will next look at **spatial frequency** analysis and processing of images.
- These methods effect both gray-levels and shapes… onward to **Module 4**!