The **brightness** and **contrast** of an image whose greyscale value is given by $f(i,j)$ is the expected value and contrast of f.

$$B = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M} f(i,j), \quad C = \frac{1}{NM}\sum_{i=1}^{N}\sum_{j=1}^{M}\big(f(i,j)-B\big)^2$$

Taking K **replicas** of the same image (f) with noise ε_i and taking their means creates a new image $f + \frac{1}{K}\sum_{i=1}^{k}\varepsilon_i$, assuming ε_i are IID the variance of the noise here is $\frac{1}{K}\sigma_\varepsilon^2$, ie. the noise is reduced by a factor of K.

The **histogram** of an image f is the probability function p_f where $p_f(n)$ is the relative number of pixels of greyscale value n To improve the **contrast** of an image, use **histogram equalization**. Histogram equalization flattens the histogram as much as possible, increasing contrast.

In general to perform a **histogram transformation** from one histogram p_f to another p_g compute the cumulative probability densities F_f and F_g, create a conversion vector v where $v[i] = j$ where $F_f(i) = F_g(j)$ (in discrete spaces, this must be approximated, ie. choose j which fits this best) note here that i iterates over all the possible greyscale values, ie $0 \dots 255$. Then convert $f(x,y)$ to $v\big[f(x,y)\big]$. **Histogram Equalization** is a histogram transformation to a flat histogram.

For a **binary image** B, its **area** is $A = \sum_{i=1}^{N}\sum_{j=1}^{M} B[i,j]$. The **center of mass** of B is given by the coordinates

$$x = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M} iB[i,j]}{A}, \quad y = \frac{\sum_{i=1}^{N}\sum_{j=1}^{M} jB[i,j]}{A}$$

Edge detection in a binary image: The edge of an object S in a binary image is the set of pixels in S 4-neighboring a pixel not in S. To find these pixels, begin scanning left-right, top-bottom, the binary image to find the first pixel $s \in S$. Once s is found, repeat until we reach s again: if the current cell is in S, add it to the boundary and turn left and continue, otherwise it is not in S so turn right and continue without updating the boundary. Note that "left" and "right" mean relative to the current orientation of how you entered the current cell, eg at the beginning when you find s, you are going left to right relative to the image so turning left orients you upward relative to the image.

Connected component labeling: A connected component in a binary image is a set of pixels such that every pixel is reachable from every other pixel via its neighbors. To find and enumerate/tag these connected components: scan the image left-right, top-bottom, and once you find a pixel with value 1: if its left neighbor or top neighbor are tagged if they have the same tag or only one is tagged, copy that tag to the current pixel. Otherwise if they have different tags, copy one of the tags and in an equivalence table label the tags of the neighbors as equivalent. Otherwise both neighbors are not tagged, so create a new tag with the minimum possible value. After scanning the entire image, for each equivalence class find the minimum tag and for every pixel in the image whose tag is in that equivalence class, change the tag to this minimum value.

To find the **distance** between pixels in a binary image (logical value 1) to the background (logical value 0), can be done iteratively:

$$f^0[i,j] = f[i,j], \quad f^m[i,j] = f[i,j] + \min\{f^{m-1}[u,v]\}$$

where $[u,v]$ is a 4-neighbor of $[i,j]$. The iteration ends once $f^m = f^{m+1}$, this gives $\rho([i,j],\bar{S})$ – the distance between $[i,j]$ to the background $\bar{S}$. The **skeleton** of an image is the set of pixels $[i,j]$ whose distance from the background is locally maximal, ie $\rho([i,j],\bar{S}) \geq \rho([u,v],\bar{S})$.

Segmentation of an image I is the division of the image into similar regions. In other words, given a **homogeneous predicate** P our goal is to find a partition of I into regions $I = \bigsqcup_{i=1}^{n} R_i$ such that $P(R_i) = \mathsf{true}$ for every i, but $P\big(P_i \sqcup P_j\big) = \mathsf{false}$ for every $i \neq j$. Example predicates are: $\max R - \min R < \varepsilon$, $\operatorname{Var} R < \varepsilon$, etc.

Split and merge segmentation: choose a region R, if $P(R) = \mathsf{false}$ then divide R into 4 equal subregions. Once all the regions are homogeneous, begin merging: if a region R and a neighbor R' satisfy $P(R \sqcup R') = \mathsf{true}$, merge R and R'. Continue this process of merging until exhaustion.

Recall that the **gradient** of a function $f(x,y)$ is defined as the vector function $\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)(x,y)$, where $\frac{\partial f}{\partial x}(x,y)$ is the **partial derivative** of f relative to x at the point (x,y):

$$\frac{\partial f}{\partial x}(x,y) = \lim_{h\to 0}\frac{f(x+h,y)-f(x,y)}{h}$$

and the partial derivative of f relative to y is defined similarly. The **directional derivative** of f relative to a *unit vector* $\vec{u}$ is defined as

$$D_u f(x,y) = \lim_{h\to 0}\frac{f\big((x,y)+h\vec{u}\big)-f(x,y)}{h}$$

this is equal to $D_u f(x,y) = \vec{u}\cdot\nabla f(x,y) = \|\nabla f(x,y)\|\cdot\cos\theta$ (dot product) where θ is the angle between $\vec{u}$ and the gradient, and so f has the highest rate of change in the direction of its gradient ($\theta = 0$).

Edge detection of an image can be done by finding where the rate of change of an image is maximal. Since the magnitude of the gradient gives the maximal rate of change of the image at a point, it can be used to detect edges. There are various ways of approximating the gradient of an image, the most basic is $\partial_x f[i,j] \approx f[i,j+1] - f[i,j]$ and similar for y, these give convolution kernels of $G_x = \begin{bmatrix}-1 & 1\end{bmatrix}, G_y = \begin{bmatrix}1\\-1\end{bmatrix}$, these can be extended to have larger sizes. **Roberts Operator** gives two other kernels:

$$G_x = \begin{bmatrix}1 & 0\\0 & -1\end{bmatrix}, \quad G_y = \begin{bmatrix}0 & -1\\1 & 0\end{bmatrix}$$

Sobel's Operator:

$$G_x = \begin{bmatrix}-1 & 0 & 1\\-2 & 0 & 2\\-1 & 0 & 1\end{bmatrix}, \quad G_y = \begin{bmatrix}1 & 2 & 1\\0 & 0 & 0\\-1 & -2 & -1\end{bmatrix}$$

Prewitt's Operator: like Sobel's but replace 2s with 1s.

Canny Edge Detection: Canny edge detection utilizes gradient kernel approximations (eg above), and creates a new binary image (the edges image) which indicates if a pixel is an edge or not. The algorithm takes three inputs: the image and a lower and higher threshold (t_ℓ and t_h).

(1) The image is smoothed using a Gaussian filter to get rid of noise.

(2) Then the gradient of the image is calculated using approximations (eg Sobel's operator), and from this we create two new images: $\|\nabla f[x,y]\|$ and $\big(\tan\theta\big)[x,y]$ by computing $\sqrt{G_x^2+G_y^2}$ and $\frac{G_y}{G_x}$ respectively where G_x and G_y are approximations of the x and y partial derivatives respectively.

(3) For every $[x,y]$ if $\|\nabla f[x,y]\| < t_\ell$ it is considered not to be an edge, so set $E[x,y] = 0$ (E is the edges image) and if $\|\nabla f[x,y]\| > t_h$ set $E[x,y] = 1$. Otherwise if any of its neighbors are edges, set $E[x,y] = 1$ (this requires two passes, one left-top to right-bottom and another in reverse).

(4) Next for every $E[x,y] = 1$ if $\|\nabla f[x,y]\|$ is not a local maximum in the direction of $\nabla f[x,y]$ then set $E[x,y] = 0$ (non max suppression). This is done by using the $\tan\theta$ image, we can figure out the major direction of the gradient (if $-0.414 < \tan\theta < 0.414$ then the gradient is flat, etc). Once we have the major direction (0 degrees, 45 degrees, 90 degrees, 135 degrees), we then check that $\|\nabla f[x,y]\|$ is larger than its neighbors in this direction.

Another method of edge detection is by using the **Laplacian Operator** which approximates the second derivative.

$$\nabla^2 = \begin{bmatrix}0 & 1 & 0\\1 & -4 & 1\\0 & 1 & 0\end{bmatrix}$$

Recall that ∇f is maximal/minimal if $\nabla^2 f = 0$. Here after convolving with the Laplacian, an edge will be indicated by a jump from a large positive number to a large negative number (perhaps with zero/s between). It is wise to smooth the image with a Gaussian filter before edge detection.

The purpose of the **Hough Transform** is for **curve fitting**: to detect and fit curves (circles, lines, etc) to the image. The idea is once we have the edges of an image, we map each edge pixel from the image space to the parameter space, where we have parameters for the curve we are detecting. If the edge point $[x, y]$ is on the curve parameterized by $[\alpha_1, \dots, \alpha_n]$ we increment $P[\alpha_1, \dots, \alpha_n]$ in the parameter space. Then we find local maximums in P (after smoothing with a mean kernel) which are above some predefined threshold (which may be a function of the parameters) which define a curve in the image space. This can be inefficient, for example with circles, so we wish to reduce how we parameterize the curves. For circles, we can for example only iterate over circles whose center is at an angle from the point equal to the angle of its gradient.

If f is a periodic function of period T then it can be written as a sum of sinusoidal functions:

$$f(x) = \frac{A_0}{2} + \sum_{n=1}^{\infty} A_n \cdot \cos\big(n \cdot 2\pi \cdot f_0 x\big) + \sum_{n=1}^{\infty} B_n \cdot \sin\big(n \cdot 2\pi f_0 x\big)$$

$$A_n = \frac{2}{T}\int_0^T f(x) \cdot \cos\big(n \cdot 2\pi f_0 x\big)\, dx, \quad B_n = \frac{2}{T}\int_0^T f(x) \cdot \sin\big(n \cdot 2\pi f_0 x\big)\, dx$$

and f_0 is f's **frequency**: $f_0 = \frac{1}{T}$. This is the **Fourier Series** of f.

For functions which are not necessarily periodic, we define the **Fourier Transform** of f by:

$$\mathcal{F}(f)(u) = \int_{-\infty}^{\infty} f(x) \cdot e^{-i2\pi ux}\, dx, \quad \mathcal{F}(f) \colon \mathbb{R} \longrightarrow \mathbb{C}$$

In multiple dimensions this can be generalized to:

$$\mathcal{F}(f)(\vec{u}) = \int_{\mathbb{R}^n} f(\vec{x}) \cdot e^{-i2\pi \vec{u}\cdot\vec{x}}\, d\vec{x}, \qquad \begin{matrix} f \colon \mathbb{R}^n \longrightarrow \mathbb{R} \\ \mathcal{F}(f) \colon \mathbb{R}^n \longrightarrow \mathbb{C} \end{matrix}$$

The inverse of the transform is:

$$\mathcal{F}^{-1}(F)(x) = \int_{-\infty}^{\infty} F(u) \cdot e^{i2\pi ux}\, du$$

for multiple dimensions, the inverse is similar (dot product instead of real product, integrate over $\mathbb{R}^n$).

The Fourier Transform is a linear transformation: $\mathcal{F}(f+g) = \mathcal{F}(f) + \mathcal{F}(g)$ and $\mathcal{F}(\alpha f) = \alpha\,\mathcal{F}(f)$. Furthermore:

$$\mathcal{F}(f(\alpha x, \beta y)) = \frac{1}{|\alpha\beta|} \cdot \mathcal{F}(f)\left(\frac{u}{\alpha}, \frac{v}{\beta}\right)$$

And the Fourier Transform of a rotated image is equal to the rotation of the Fourier transform of the original image.

The **Dirac Delta Function** is a "function" (or measure) δ such that for every function $\int_{-\infty}^{\infty} f(x) \cdot \delta(x)\, dx = f(0)$.
And so $\int_{-\infty}^{\infty} f(x) \cdot \delta(x - x_0)\, dx = f(x_0)$. Furthermore $\delta(\alpha x) = \frac{1}{\alpha x}\delta(x)$. So the Fourier Transform of δ is 1.

We define continuous convolution of two functions by:

$$\big(f * g\big)(x) = \int_{\Omega} f(\tau) \cdot g(x - \tau)\, d\tau$$

Where Ω is equal to the domain of f and g ($\mathbb{R}^k$ for some k). Convolutions are commutative and associative.

The **Convolution Theorem** states that the image of a convolution is a product and the image of a product is a convolution, ie:

$$f * g \xleftrightarrow{\mathcal{F}} \mathcal{F}(f) \cdot \mathcal{F}(g), \quad f \cdot g \xleftrightarrow{\mathcal{F}} \mathcal{F}(f) * \mathcal{F}(g)$$

If we want to **sample** a function f with frequency τ (every τ units) we define the **sampled function** $f_s(x) = f(x) \cdot \text{Ш}\big(\frac{x}{\tau}\big)$. Where

$$\text{Ш}(x) = \sum_{n=-\infty}^{\infty} \delta(x - n) \Longrightarrow \text{Ш}\Big(\frac{x}{\tau}\Big) = \tau \sum_{n=-\infty}^{\infty} \delta(x - \tau n)$$

And so we get that

$$\mathcal{F}(f_s(x)) = \sum_{n=-\infty}^{\infty} \mathcal{F}(f)\Big(u - \frac{n}{\tau}\Big)$$

Notice how we sampled f with a frequency of τ but the Fourier Transform of f_s is the sum of differences with frequency of $\frac{1}{\tau}$. So if τ is too large, then the sum in the Fourier transform of f_s will overlap, this is called **undersampling**.

The **Discrete Fourier Transform** is analogous to its continuous cousin, but its purpose is for transforming finite (or periodic) sequences. Given a sequence $x[0], \dots, x[N-1]$ its Fourier Transform, X is defined as follows:

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot e^{-i\frac{2\pi}{N}\cdot kn}, \qquad k = 0, \dots, N-1$$

in some conventions this is multiplied by $\frac{1}{N}$.

The **Inverse Discrete Fourier Transform** is given by (if the sequence is $X[0], \dots, X[N-1]$):

$$x[n] = \sum_{k=0}^{N-1} X[k] \cdot e^{i\frac{2\pi}{N}\cdot kn}, \qquad n = 0, \dots, N-1$$

if the Discrete Fourier Transform is multiplied by $\frac{1}{N}$, so is this.

The sequence $x[0], \dots, x[N-1]$ corresponds to sampling a signal f at N points in intervals of Δx (f is sampled at $0, \Delta x, \dots, (N-1)\Delta x$) and its resulting transform X is equal to the sampling of the continuous transform of f at points $0, \Delta u, \dots, (N-1)\Delta u$ where $\Delta u = \frac{1}{N\Delta x}$. This can be seen since:

$$\mathcal{F}(f_s)(u) = \sum_{n=0}^{N-1} \int f(x) e^{-2\pi i \cdot ux} \delta(x - n\Delta x) = \sum_{n=0}^{N-1} f(n\Delta x) \cdot e^{-2\pi i \cdot un\Delta x}$$

And so $\mathcal{F}(f_s)(k\Delta u) = \sum f(n\Delta x) \cdot e^{-2\pi i \cdot nk\Delta x\Delta u}$ so when $\Delta x \cdot \Delta u = \frac{1}{N}$ this is equal to $X[k]$ (recall $x[n] = f(n\Delta x)$).

Similarly for a 2 dimensional matrix $x[0 \dots N-1\,,\, 0 \dots M-1]$ its Fourier Transform is a matrix $X[0 \dots N-1\,,\, 0 \dots M-1]$:

$$X[u, v] = \sum_{j=0}^{M-1} \sum_{i=0}^{N-1} x[i, j] \cdot e^{-2\pi\left(\frac{ui}{N} + \frac{vj}{M}\right)}$$

Some conventions multiply this by $\frac{1}{NM}$. Notice that if we find the Discrete Fourier Transform of every row of x and then take the Discrete Fourier Transform of each column, we get X (the reverse order also gives X). The inverse 2 dimensional DFT is given by:

$$x[i, j] = \sum_{v=0}^{M-1} \sum_{u=0}^{N-1} X[u, v] \cdot e^{i2\pi\left(\frac{ui}{N} + \frac{vj}{M}\right)}$$

potentially multiplied by $\frac{1}{NM}$.

DFT can be computed by the **Fast Fourier Transform** (FFT) algorithm in $O(n \log n)$ time. The first component in a DFT is called the **DC component** and is equal to the sum of the values of the sequence (mean if multiplied by $\frac{1}{N}$ or $\frac{1}{NM}$). Further notice that $X[u, v] = X[u+N, v+M] = X[u+N, v] = X[u, v+M]$ (DFT is cyclic). Another important function is **FFTshift** which swaps the order of a DFT so that the DC is in the center.

Frequency filters are filters applied in the frequency domain. Due to the Convolution Theorem, multiplication in the frequency domain is equivalent to convolving in the image domain, so these filters are applied via elementwise multiplication. A **Low Pass Filter** (LPF) is a filter $H(u, v)$ which filters *out* high frequencies and preserves lower frequencies. A LPF will have that higher indexes of H are smaller (closer to 0) and lower indexes are larger. A **High Pass Filter** (HPF) is a filter $H(u, v)$ which filters out low frequencies and preserves higher frequencies. If H is a HPF then its higher indexes of H are larger and lower indexes are smaller. A **Band Pass Filter** (BPF) is a filter which filters (in) only frequencies within a region.

Since higher frequencies correlate with areas of higher rates of change, HPFs will create edge images. Similarly, LPFs will create blurred/smoothed images.

A **High Frequency Emphasis** filter (HFE) is a filter of the form $k_0 + k_1 H$ where H is a HPF, its purpose is to emphasize higher frequencies (edges) and preserve lower frequencies (so $k_1 = 1$ usually). HFEs are usually used in conjunction with **Histogram Equalization**.