African Institute for Mathematical Sciences — Cameroon
Structured Master's, Data Science Stream · Computer Vision

The Most Important Operation in Vision

Week 1 · Day 3: Noise, Convolution, Filtering and Histograms

Yusuf Brima, Ph.D.

Introduction to Computer Vision · AIMS Cameroon, 2026–2027

Today

1. **Noise**: where it comes from, what it looks like, how to measure it
2. **Convolution**: the sliding weighted sum at the heart of every CNN
3. **Filtering**: blurring, denoising, sharpening, and choosing the right filter
4. **The frequency view**: images as sums of waves
5. **Histograms**: reading them, stretching contrast, equalisation

By the end of today you should be able to

- compute a convolution by hand, including its output size;
- choose a filter for a given type of noise;
- explain blurring and sharpening in terms of frequencies;
- derive and apply histogram equalisation.

Where we are

Yesterday

- A pixel is a measurement of light: sampled in space, quantised in value.
- Photon counting is random: images are *noisy*.
- Shrinking without blurring first causes aliasing.

Today

- Operations that turn an image into a *better* image: **image processing**.
- The tool behind almost all of them is **convolution**. Learn it deeply today; it will carry you through Week 2.

Two families of operations

Point operations change each pixel using only its own value (brightness, contrast, histograms).

Neighbourhood operations change each pixel using its neighbours too (blur, sharpen, edges). Convolution is the linear neighbourhood operation.

Part 1

Noise

What is noise?

A simple model

$$\underbrace{\tilde{I}[m,n]}_{\text{what we record}} = \underbrace{I[m,n]}_{\text{the true scene}} + \underbrace{\eta[m,n]}_{\text{noise}}$$

- Noise is the part of the measurement that does *not* come from the scene.
- It is random: two photos of a still scene differ.

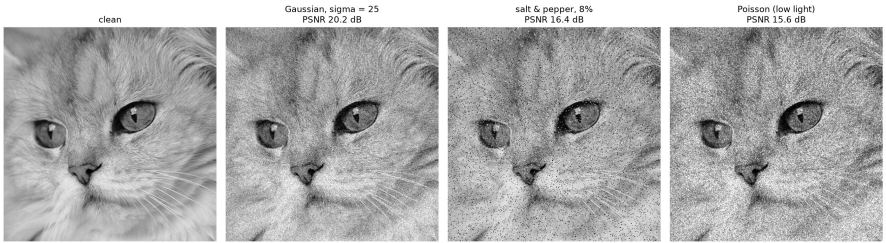
Where it comes from

- **Photon (shot) noise**: light arrives in random packets.
- **Electronic noise**: heat in the sensor and amplifier.
- **Quantisation**: rounding to integers.
- **Transmission**: corrupted bits, dead pixels.
- **Compression**: JPEG blocks and ringing.

Intuition

Denoising means *guessing* I from $\tilde{I}$, using an assumption such as "neighbouring pixels are usually similar".

Three noise models on a real photograph



Gourdo 1 at en.wikipedia, Public domain

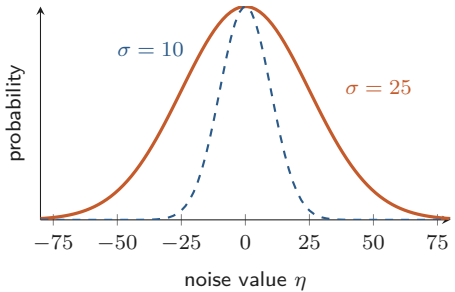
Gaussian: every pixel nudged by a small random amount. Electronics, heat.

$$\eta \sim \mathcal{N}(0, \sigma^2)$$

Salt & pepper: a few pixels completely wrong (0 or 255). Bit errors, dead pixels.

Poisson (shot): variance equals the signal. Dominant in low light and X-ray.

Gaussian noise, intuitively

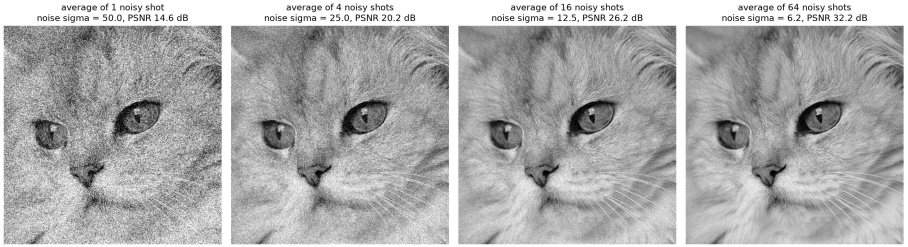


- Most pixels are nudged a little, a few a lot.
- About 68% of pixels change by less than σ; about 95% by less than 2σ.
- Independent from pixel to pixel: neighbours' noise is unrelated.

Intuition

Independence is the key. The *signal* in neighbouring pixels is similar; the *noise* is not. Averaging neighbours keeps the signal and cancels the noise. Every denoiser today exploits this.

Averaging cancels noise: many photos of the same scene



Gourdo 1 at en.wikipedia, Public domain

Averaging N independent measurements

$$\sigma_{\text{average}} = \frac{\sigma}{\sqrt{N}}$$

Four shots halve the noise; 64 shots cut it by a factor of 8. Phone "night mode" does exactly this: it takes a burst of photos, aligns them and averages.

Measuring quality: MSE and PSNR

Mean squared error and peak signal-to-noise ratio

$$\mathrm{MSE} = \frac{1}{MN} \sum_{m,n} \big(I[m,n] - \hat{I}[m,n]\big)^2$$

$$\mathrm{PSNR} = 10 \log_{10} \frac{255^2}{\mathrm{MSE}} \ \text{dB}$$

$\hat{I}$ is our estimate (noisy or denoised); I is the clean reference.

- Higher PSNR is better. $+3$ dB $\approx$ half the MSE.
- 8-bit rules of thumb: below 20 dB visibly poor; above 30 dB good; above 40 dB nearly indistinguishable.

Gaussian noise, $\sigma = 25$

Each pixel's squared error averages σ^2, so $\mathrm{MSE} \approx 625$.

$$\begin{aligned} \mathrm{PSNR} &= 10 \log_{10} \tfrac{65\,025}{625} \\ &= 10 \log_{10} 104 \approx 20.2 \ \text{dB} \end{aligned}$$

Exactly what we measured on the cat.

PSNR is not the whole story

- PSNR treats every pixel error equally, wherever it is.
- Our eyes do not: an error on an edge or a face is far more visible than in a textured lawn.
- Two images can have the same PSNR yet look very different: one slightly blurred, one with a few bright specks.

Better-aligned measures

- **SSIM**: compares local brightness, contrast and structure.
- **Learned metrics** (e.g. LPIPS): compare features inside a trained network.
- **Task metrics**: does the denoised X-ray still let the detector find the nodule?

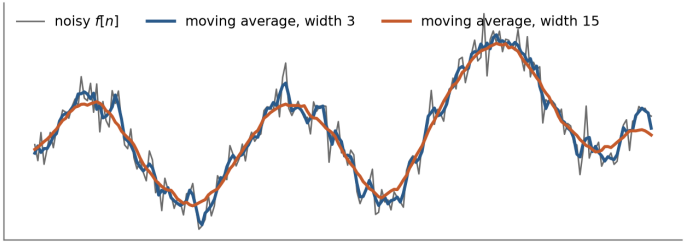
Practitioner's Note

Always ask *quality for what?* A denoiser that maximises PSNR can erase the faint ring-form malaria parasites a microscopist needs.

Part 2

Convolution

Start in 1D: the moving average



Replace each sample by the average of itself and its neighbours. The noise shrinks. A wider window removes more noise but also rounds off the sharp step.

Intuition

The **noise vs. detail trade-off**: averaging cannot tell noise from genuine fine detail. Every smoothing filter today makes this trade, more or less cleverly.

1D convolution: the formula, read slowly

1D discrete convolution

$$g[n] = (f * h)[n] = \sum_{k=-a}^{a} h[k]\, f[n-k]$$

- f: the input signal. h: the **kernel** (or filter), a small list of weights. g: the output.
- Each output is a **weighted sum** of nearby inputs. The kernel holds the weights.
- Moving average of width 3: $h = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$.

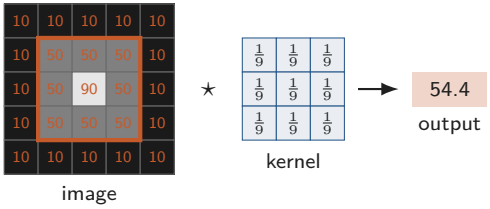
By hand

$f = [2,\ 4,\ 8,\ 4,\ 2]$, $h = [\frac{1}{3}, \frac{1}{3}, \frac{1}{3}]$.

$g[1] = \frac{2+4+8}{3} = 4.67$
$g[2] = \frac{4+8+4}{3} = 5.33$
$g[3] = \frac{8+4+2}{3} = 4.67$

The peak of 8 is spread out to 5.33: smoothing.

2D convolution: a sliding window



1. Place the kernel over a 3×3 patch.
2. Multiply each pixel by the weight on top of it.
3. Add up: that is one output pixel.
4. Slide one step and repeat, everywhere.

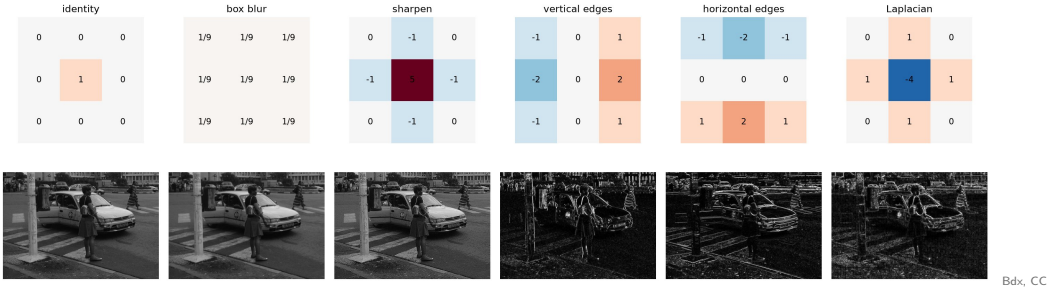
The centre pixel

$\frac{8 \cdot 50 + 90}{9} = \frac{490}{9} \approx 54.4$. The bright 90 is pulled towards its neighbours.

2D discrete convolution

$$(I * h)[m, n] = \sum_{i=-a}^{a} \sum_{j=-a}^{a} h[i, j]\, I[m - i,\, n - j]$$

One image, six kernels



BY-SA 4.0 Bdx, CC

Same operation, different weights, completely different results: a copy, a blur, a sharpened image, vertical edges, horizontal edges, and all edges.

Intuition

The *kernel* decides what the filter looks for. Week 1: we choose the kernel by hand. Week 2: a CNN *learns* thousands of kernels from data.

Reading a kernel

- **Weights sum to 1** (blur): a flat region stays the same brightness.
- **Weights sum to 0** (edges): a flat region becomes 0; only *changes* survive.
- **Positive centre, negative surround** (sharpen, Laplacian): exaggerates the difference between a pixel and its neighbours.
- **Positive on one side, negative on the other** (Sobel): detects a change in one direction.

The vertical-edge kernel on a step

Kernel $[-1,\ 0,\ +1]$ (one row of Sobel).
Row of pixels: $[20,\ 20,\ 20,\ 200,\ 200,\ 200]$.

On the flat left part: $-20 + 0 + 20 = 0$.
Across the step: $-20 + 0 + 200 = 180$.
On the flat right part: 0.

The output lights up *only at the edge*.

Intuition

A kernel is a *template*: the output is large where the image locally looks like it. CNN neurons do exactly this.

Convolution vs. correlation

The only difference: a flip

Convolution:

$$(I * h)[m, n] = \sum_{i,j} h[i, j]\, I[m - i,\, n - j]$$

Correlation:

$$(I \star h)[m, n] = \sum_{i,j} h[i, j]\, I[m + i,\, n + j]$$

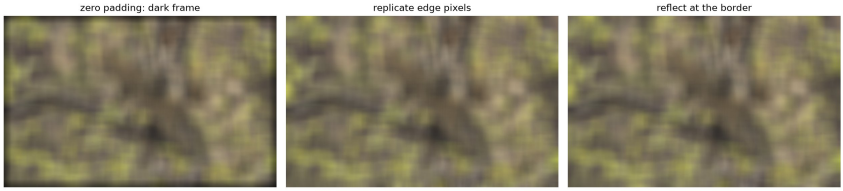
Convolution flips the kernel by 180° before sliding it; correlation does not.

- For symmetric kernels (box, Gaussian) the two are identical.
- For Sobel, the flip changes the sign of the output.
- Convolution has nicer maths (it is commutative and associative).

Practitioner's Note

Deep-learning "convolution" layers actually compute *correlation*. It does not matter: the weights are learned, so the network simply learns the flipped kernel.

What happens at the border?



Davidvraju, CC BY-SA 4.0

At the edge of the image the kernel hangs over nothing. We must invent the missing pixels (**padding**):

- **zero**: assume black: darkens the border when blurring;
- **replicate**: repeat the edge pixel;
- **reflect**: mirror the image at the edge.

Practitioner's Note

CNNs usually use zero padding. That lets a network know where the border is, which can make it sensitive to an object's absolute position.

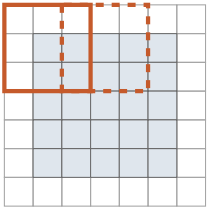
Padding, stride and output size

Output size, per dimension

$$O = \left\lfloor \frac{N + 2p - K}{s} \right\rfloor + 1$$

N input size, K kernel size, p padding, s stride.

- **valid** ($p = 0$): output shrinks by $K - 1$.
- **same** ($p = (K-1)/2$, $s = 1$): output size $=$ input size.
- **stride** s: jump s pixels at a time: output about N/s.



5×5 input, $p = 1$ (grey ring), $K = 3$, stride $s = 2$: output 3×3

Worked examples: output sizes

Same padding

$N = 32$, $K = 5$, $p = 2$, $s = 1$: $O = \lfloor 31/1 \rfloor + 1 = 32$.

The first layer of ResNet

$N = 224$, $K = 7$, $s = 2$, $p = 3$:

$$O = \left\lfloor \frac{224+6-7}{2} \right\rfloor + 1 = \lfloor 111.5 \rfloor + 1 = 112$$

Five valid 3×3 layers

Each removes $K - 1 = 2$ pixels: $224 \to 222 \to \cdots \to 214$.

Halving a 64×64 map

$K = 3$, $s = 2$, $p = 1$: $\lfloor 63/2 \rfloor + 1 = 32$. Or $K = 2$, $s = 2$, $p = 0$: $\lfloor 62/2 \rfloor + 1 = 32$.

Practitioner's Note

You will compute these numbers constantly in Week 2 when designing networks. Most "shape mismatch" errors in PyTorch are this formula, applied wrongly.

Why convolution is special

Two defining properties

- **Linear**: filtering a sum = sum of the filtered images; doubling the input doubles the output.

$$(\alpha I_1 + \beta I_2) * h = \alpha\, I_1 * h + \beta\, I_2 * h$$

- **Shift-equivariant**: shift the input, and the output shifts by the same amount.

The theorem: every operation that is linear *and* shift-invariant is a convolution with some kernel.

Shift-equivariance in plain words

A cat in the top-left corner and a cat in the bottom-right are processed by *exactly the same rule*. The filter does not need to learn "cat" separately for every position.

Associativity

$(I * h_1) * h_2 = I * (h_1 * h_2)$: two blurs in a row are one bigger blur. Two stacked 3×3 convolutions see a 5×5 region.

Separable kernels: a big speed-up

Separability

If $h[i,j] = u[i]\,v[j]$ (an outer product), then

$$I * h = (I * u) * v$$

Filter the rows with v, then the columns with u.

Sobel is separable

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

A smoothing in one direction and a difference in the other.

Cost of a 15×15 Gaussian

Direct: $15^2 = 225$ multiplications per pixel.
Separable: $15 + 15 = 30$ per pixel.

For a 12-megapixel photo: 2.7 billion vs. 0.36 billion operations, **7.5 times faster**.

Practitioner's Note

Box and Gaussian kernels are separable; OpenCV exploits this automatically. Modern efficient CNNs (Week 2) use a similar factorisation trick.

A convolution output is a neuron

One output pixel

$$y = \sum_{i,j} w_{ij}\, x_{ij} = \mathbf{w}^\top \mathbf{x}$$

$\mathbf{x}$: the patch under the kernel, flattened. $\mathbf{w}$: the kernel, flattened.

- That is a dot product: exactly what an artificial neuron computes (before its activation).
- Sliding the kernel = the *same* neuron applied at every position: **weight sharing**.

Parameters

A 3×3 kernel on a 224×224 image: **9** weights, reused at 50 176 positions.

A fully connected layer mapping the same image to an image of the same size: $50\,176^2 \approx 2.5$ **billion** weights.

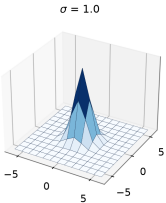
Intuition

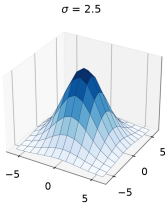
Weight sharing is why CNNs are trainable at all. This slide is the bridge to Day 5 and Week 2.

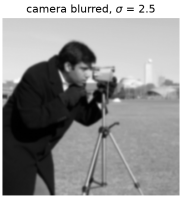
Part 3

Filtering: blur, denoise, sharpen

Smoothing filters: box and Gaussian







- **Box**: every weight $1/K^2$. Cheap, but its hard edges create blocky artefacts.
- **Gaussian**: nearer pixels count more. One parameter σ. Kernel width $\approx 6\sigma$, rounded to an odd number.

2D Gaussian kernel

$$h(i,j) \propto \exp\Big(-\frac{i^2+j^2}{2\sigma^2}\Big)$$

normalised so the weights sum to 1.

Intuition

The Gaussian is the "natural" blur: it is what an out-of-focus lens roughly does, it has no preferred direction, and blurring twice with Gaussians gives another Gaussian.

Worked example: building a 3×3 Gaussian kernel

$\sigma = 1$

Unnormalised weights $e^{-(i^2+j^2)/2}$:

centre $(0,0)$: $e^0 = 1$
edge $(\pm 1, 0)$: $e^{-0.5} = 0.607$
corner $(\pm 1, \pm 1)$: $e^{-1} = 0.368$

Sum $= 1 + 4(0.607) + 4(0.368) = 4.90$

Normalised kernel:

$$\begin{bmatrix} 0.075 & 0.124 & 0.075 \\ 0.124 & 0.204 & 0.124 \\ 0.075 & 0.124 & 0.075 \end{bmatrix}$$

- The centre counts most; weights fall off smoothly with distance.
- Dividing by the sum makes the weights add to 1, so flat regions keep their brightness.
- With $\sigma = 1$ a 3×3 window cuts the Gaussian short; the 6σ rule suggests 7×7.

Choosing σ: the scale of detail you keep



Minette Lontsie, CC BY-SA 4.0

At $\sigma = 2$ faces and number plates blur; at $\sigma = 5$ individual people merge; at $\sigma = 12$ only the layout of the crowd and the road remains.

Intuition

σ sets a **scale**: details smaller than about σ pixels are removed. Blurring at many σ values gives a *scale space*, the basis of SIFT tomorrow.

The median filter: a non-linear alternative

- Replace each pixel by the **median** of its window: sort the values, take the middle one.
- Not a weighted sum, so *not* a convolution.
- A single extreme value cannot pull the median far: it is **robust to outliers**.
- It also keeps sharp edges sharp: at a step, the median is either the dark or the bright value, never something in between.

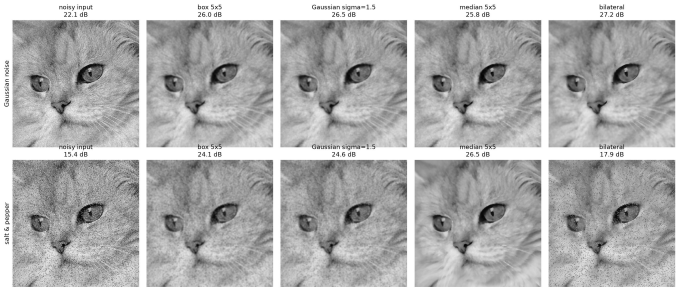
A salt pixel

A 3×3 patch: eight pixels of 128, centre of 255 ("salt").

Box filter: $\frac{8 \cdot 128 + 255}{9} = \frac{1279}{9} \approx 142$.
The error is smeared over the neighbourhood.

Median: sorted values $128, \ldots, 128, 255$; the middle (5th) one is 128.
The outlier simply disappears.

The right filter for the right noise: a real comparison



Gourdo 1 at en.wikipedia, Public domain

Gaussian noise (top): smoothing filters all help; the bilateral filter is best (27.2 dB). **Salt & pepper** (bottom): the median wins clearly (26.5 dB); the bilateral filter barely helps (17.9 dB).

Intuition

There is no best filter, only a best filter *for a noise model*. Know your sensor before you choose.

Edge-preserving smoothing: the bilateral filter

Bilateral filter

$$\hat{I}(\mathbf{x}) = \frac{1}{W} \sum_{\mathbf{y}} \underbrace{G_{\sigma_s}(\|\mathbf{x}-\mathbf{y}\|)}_{\text{close in space}} \underbrace{G_{\sigma_r}(|I(\mathbf{x})-I(\mathbf{y})|)}_{\text{similar in value}} I(\mathbf{y})$$

W normalises the weights to sum to 1.

- Average only neighbours that are *near* and *look similar*.
- Across an edge, the neighbours look different, so they get almost no weight: the edge survives.

Intuition

A Gaussian blur asks "who is near me?". The bilateral filter asks "who is near me *and* like me?". It is the first filter today whose weights depend on the image itself, an idea that returns in *attention* (Week 2).

Practitioner's Note

It fails on salt & pepper: an outlier pixel has no similar neighbours, so it averages with nothing and survives.

The family of denoisers

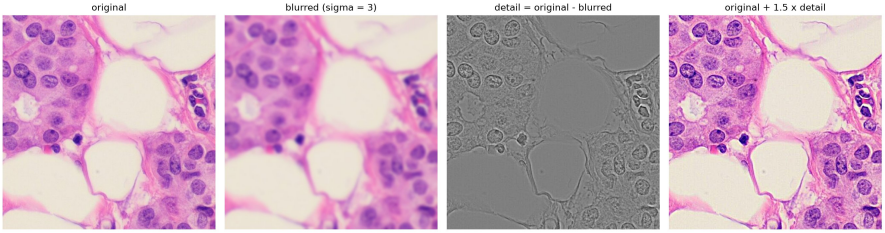


- Moving right, the assumption about "what clean images look like" gets richer.
- **Non-local means**: a brick in a wall looks like other bricks far away; average with *them*.
- **Learned denoisers** learn the assumption from millions of clean images.

Intuition

A diffusion model (Week 3) is a denoiser so good that, starting from pure noise and denoising step by step, it produces a brand-new image.

Sharpening: add back the detail



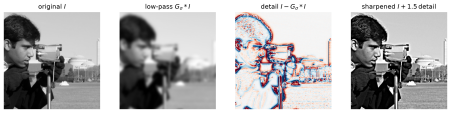
Mikael Häggström, CC0

Unsharp masking

$$I_{\text{sharp}} = I + \alpha \left(\underbrace{I - G_\sigma * I}_{\text{detail}} \right)$$

Blur the image, subtract the blur to isolate the detail (cell membranes, nuclei edges), and add it back amplified. The name comes from a darkroom technique using a blurred ("unsharp") negative as a mask.

Sharpening has a price



- The detail image $I - G_\sigma * I$ holds edges and texture, but also **noise**; sharpening amplifies all of them.
- Overdo it and you get bright **halos** along edges.

Unsharp masking is one kernel

$(1+\alpha)\,\delta - \alpha\, G_\sigma$, where δ is the identity kernel. With a 3×3 box blur and $\alpha = 1$: centre weight $2 - \frac{1}{9} = \frac{17}{9}$, all others $-\frac{1}{9}$.

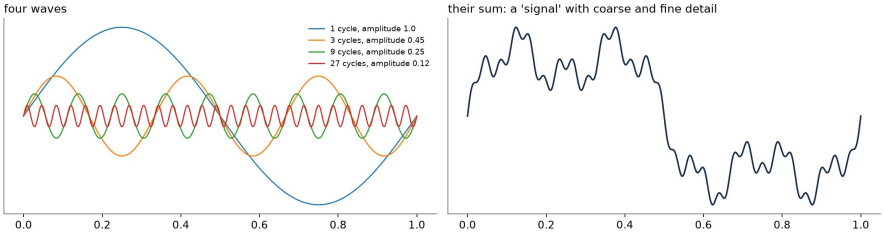
Practitioner's Note

Denoise first, then sharpen, never the other way round. Phone cameras over-sharpen, a source of halo artefacts in datasets.

Part 4

The frequency view

Any signal is a sum of waves

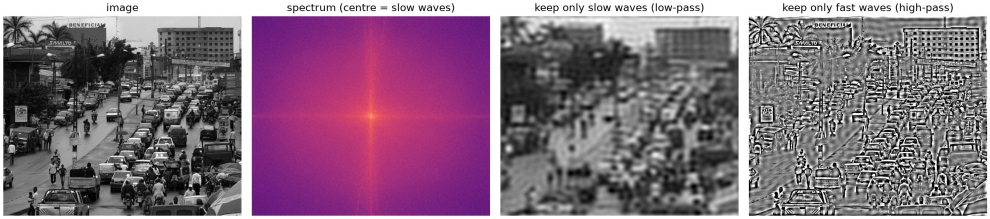


A slow wave gives the overall shape; faster waves add finer and finer wiggles. With enough waves of the right sizes you can build *any* signal: this is Fourier's insight (1807).

Intuition

Images work the same way in 2D: slow waves give smooth shading and layout; fast waves give edges, texture and noise.

The frequency content of a real photograph



Happiraphael, CC BY-SA 4.0

The **spectrum** shows how much of each wave the image contains: slow waves at the centre, fast waves at the edges. Most energy is near the centre. Keeping only slow waves gives a blur; keeping only fast ones gives edges.

Intuition

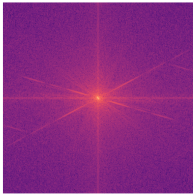
The bright cross in the spectrum comes from the many horizontal and vertical lines (buildings, road edges) in the street.

The convolution theorem

image



log |Fourier spectrum|



keep low frequencies



keep high frequencies



Convolution theorem

$$\mathcal{F}\{I * h\} = \mathcal{F}\{I\} \cdot \mathcal{F}\{h\}$$

Convolving in space = multiplying in frequency.

- **Blur** = **low-pass**: keeps slow waves, suppresses fast ones.
- **Edge detection** = **high-pass**.
- Noise lives mostly at high frequencies: that is *why* blurring removes it.

Connecting the dots

Things that now make sense

- **Aliasing** (Day 2): fast waves that cannot be represented at the new resolution fold back as slow ones. A low-pass blur removes them *before* sampling.
- **JPEG** (Day 2): stores slow waves accurately and fast waves coarsely.
- **Sharpening**: boosts fast waves.
- **Big kernels**: can be applied faster as a multiplication in the frequency domain.

Two views, one operation

Space view: "average each pixel with its neighbours".

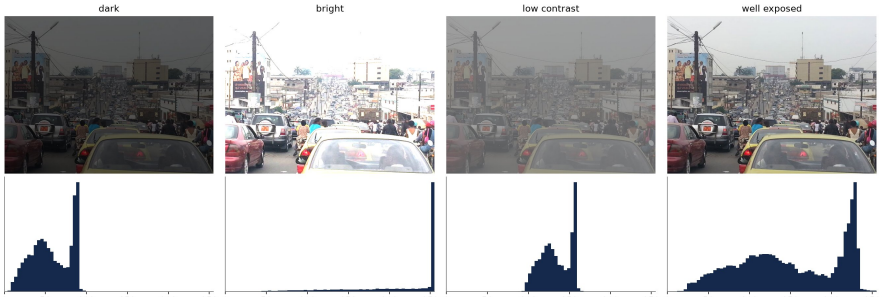
Frequency view: "turn down the high notes".

Like an audio equaliser: a blur is the bass knob, sharpening the treble knob. Some problems are easy in one view and hard in the other.

Part 5

Histograms and contrast

What a histogram tells you

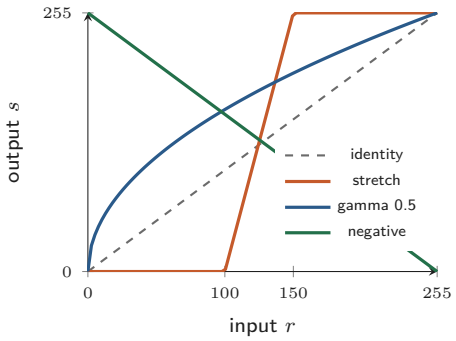


Guaka, CC BY-SA 4.0

A **histogram** counts how many pixels have each brightness: $h(r_k) = n_k$. It forgets *where* pixels are and keeps only *how bright*.

Dark: piled up on the left. **Bright**: piled up on the right, with a spike at 255 (clipping). **Low contrast**: squeezed into a narrow band. **Well exposed**: spread across the range.

Point operations: transform each pixel on its own



- A point operation is a curve $s = T(r)$ applied to every pixel.
- **Contrast stretching**: map a narrow band $[r_{\min}, r_{\max}]$ onto $[0, 255]$.
- **Gamma**: $s = 255\,(r/255)^{\gamma}$; $\gamma < 1$ brightens shadows.
- **Negative**: $s = 255 - r$ (X-ray viewing).

Stretching

Pixels lie in $[100, 150]$. Use $s = (r - 100) \cdot \frac{255}{50}$. $r = 100 \to 0$, $\quad r = 125 \to 127.5$, $\quad r = 150 \to 255$.

Histogram equalisation: the idea

- Stretching uses a straight line. But what if the pixels are crowded in *two* narrow bands, or unevenly?
- **Goal**: find the curve T that makes the output histogram as **flat** as possible: every brightness used equally often.
- **Answer**: use the image's own **cumulative histogram** (CDF) as the curve.

Intuition

Imagine 100 students queuing by height. Equalisation assigns new "heights" by *rank*: the shortest gets 0, the median gets 128, the tallest gets 255. Crowded height ranges get spread out; empty ranges get squeezed.

The rule

$$s = (L-1) \cdot \mathrm{CDF}(r)$$

$\mathrm{CDF}(r)$: fraction of pixels with brightness $\leq r$.

Why the CDF? A short derivation

Treat brightness as a random variable r with density p_r and CDF F_r. We want a monotone map $s = T(r)$ that makes s uniform.

Claim. $U = F_r(r)$ is uniform on $[0, 1]$.

Proof. For any $u \in [0, 1]$:

$$P(U \le u) = P\big(r \le F_r^{-1}(u)\big) = F_r\big(F_r^{-1}(u)\big) = u.$$

That is the CDF of a uniform variable. $\square$

Equalisation transform

Continuous:

$$s = (L-1) \int_0^r p_r(w)\, dw$$

Discrete (n pixels in total):

$$s_k = \text{round}\Big((L-1) \sum_{j=0}^{k} \frac{n_j}{n}\Big)$$

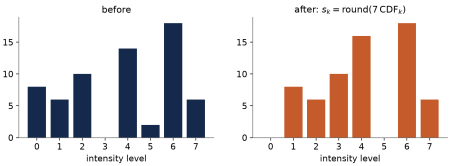
Intuition

Where the histogram is crowded, the CDF is steep, so nearby levels get pulled apart. Where it is sparse, the CDF is flat, so levels get pushed together.

Worked example: an 8-level image with 64 pixels

k	0	1	2	3	4	5	6	7
n_k	8	6	10	0	14	2	18	6
CDF	.125	.219	.375	.375	.594	.625	.906	1
7·CDF	0.9	1.5	2.6	2.6	4.2	4.4	6.3	7.0
s_k	**1**	**2**	**3**	**3**	**4**	**4**	**6**	**7**

Levels 4 and 5 merge into 4; level 6 (the most crowded) jumps to 6 and is now well separated from 4.



Worked example: why the result is never perfectly flat

16 pixels, $L = 4$ levels

Counts $n = (10, 4, 2, 0)$.

CDF: $(0.625,\ 0.875,\ 1,\ 1)$
$3{\cdot}$CDF: $(1.875,\ 2.625,\ 3,\ 3)$
$s = (2,\ 3,\ 3,\ 3)$

New histogram: 10 pixels at level 2, 6 pixels at level 3. Not flat at all.

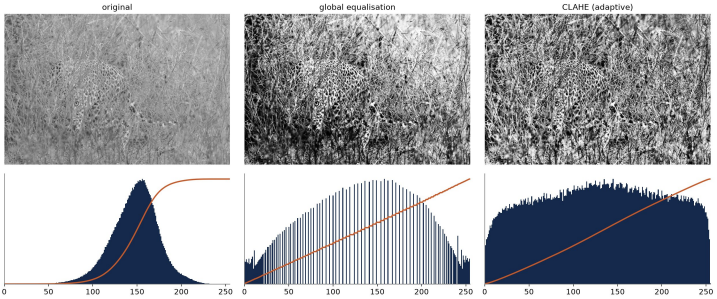
Intuition

Equalisation moves whole *bins*; it never splits a bin. Ten pixels that share a value must all get the same new value. So a discrete histogram can only become *flatter*, never perfectly flat.

Practitioner's Note

With 256 levels and millions of pixels, the result is usually close to flat. With few levels (or a large uniform background) it is not.

Equalisation on a real photograph



Senthiaathavan, CC BY-SA 4.0

The camouflaged leopard from Day 1, in grayscale. The original uses a narrow band of greys. **Global** equalisation spreads them out (the CDF becomes a straight line). **CLAHE** does it locally, tile by tile.

Intuition

Equalisation makes the leopard's spots easier *for you* to see. It adds no information: a computer comparing pixel values gains nothing that was not already there.

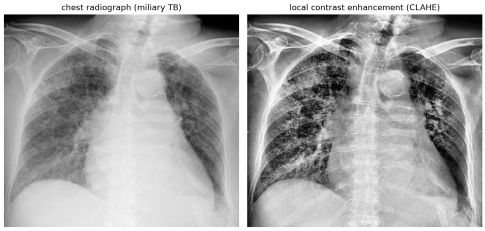
Adaptive equalisation: CLAHE

Problems with global equalisation

- One curve for the whole image ignores *local* contrast: a dark corner and a bright sky get the same treatment.
- Large flat regions (sky, background) get their noise amplified.

CLAHE: contrast-limited adaptive histogram equalisation

1. Split the image into tiles (e.g. 8×8).
2. Equalise each tile on its own.
3. *Clip* each tile's histogram first, to limit how much noise can be amplified.
4. Blend between tiles to hide the seams.

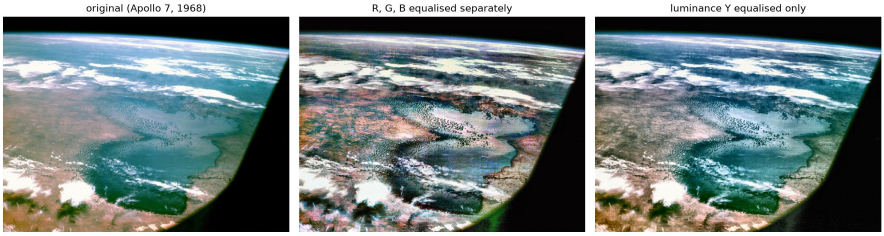


B. Herreros et al., CC BY 4.0

Practitioner's Note

Standard preprocessing for X-rays, fundus photographs and hazy satellite images, and a cheap data augmentation.

Equalising colour images



NASA, Public domain

Equalising R, G and B *separately* changes their ratios, so colours shift (the false pinks and greens in the middle). Instead convert to a space with a separate brightness channel (YCbCr, HSV, Lab), equalise *only brightness*, and convert back.

Intuition

The same lesson as Day 2: pick the representation in which your operation means what you want. Here: change brightness, keep colour.

Histogram matching: making devices comparable

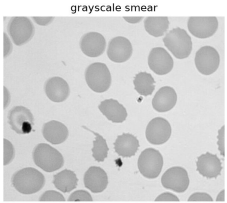
- Equalisation maps a histogram to a *flat* one. **Histogram matching** maps it to *any target* histogram.
- Recipe: apply your own CDF (making it uniform), then the *inverse* of the target's CDF.

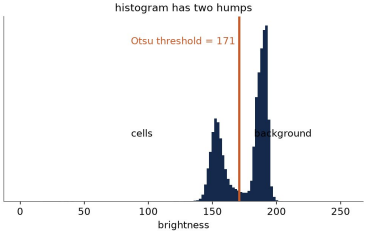
$$s = F^{-1}_{\text{target}}\big(F_{\text{source}}(r)\big)$$

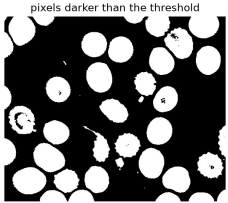
Practitioner's Note

Two hospitals stain pathology slides slightly differently; two microscopes give different colour casts. Matching each image to a reference reduces this *domain shift* before a model sees it. In scikit-image: `exposure.match_histograms`.

Thresholding: from grey to black-and-white







OpenFlexure Project, CC BY-SA 4.0

A threshold is the simplest point operation: $s = 1$ if $r < t$, else 0. When the histogram has two humps (cells, background), a good t sits in the valley between them.

Otsu's method (1979)

Try every t; pick the one that makes the two groups as *internally similar* as possible (minimum within-class variance). Here $t = 171$.

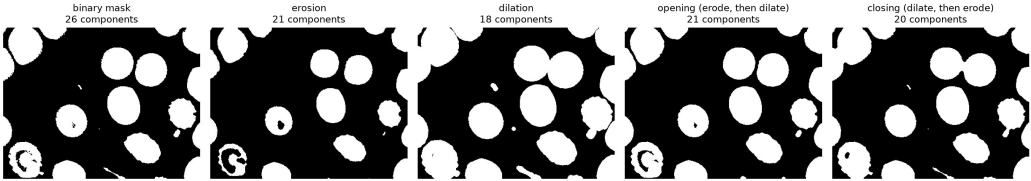
Morphology: cleaning up binary masks

- Neighbourhood operations on *shapes*, using a small **structuring element** (e.g. a disc).
- **Erosion**: a pixel stays white only if the whole disc around it is white. Objects shrink; specks vanish.
- **Dilation**: white if *any* pixel under the disc is white. Objects grow; holes fill.
- **Opening** = erode, then dilate: removes small specks and thin bridges, keeps big shapes.
- **Closing** = dilate, then erode: fills small holes and gaps.

Intuition

Erosion is like a min filter, dilation like a max filter: non-linear cousins of convolution.

Morphology on a real blood-smear mask



OpenFlexure Project, CC BY-SA 4.0

Erosion separates touching cells and deletes specks, but shrinks everything. Dilation merges neighbours. Opening removes specks while keeping cell size; closing fills the pale centres of cells. The component count changes with each.

Practitioner's Note

Threshold, then open/close, then label connected components: a complete classical counting pipeline. The crack and malaria examples on Day 1 used exactly this.

All of today in a few lines of code

```python
import cv2, numpy as np
# filtering
box   = cv2.blur(img, (5, 5))
gauss = cv2.GaussianBlur(img, (0, 0), sigmaX
    =1.5)
med   = cv2.medianBlur(img, 5)
bil   = cv2.bilateralFilter(img, 9, 60, 7)
k = np.array([[0,-1,0],[-1,5,-1],[0,-1,0]],
    np.float32)
sharp = cv2.filter2D(img, -1, k)   #
    correlation!
```

```python
# histograms
eq = cv2.equalizeHist(gray)
clahe = cv2.createCLAHE(clipLimit=2.0,
                        tileGridSize=(8, 8)
    )
eq2 = clahe.apply(gray)
# colour: equalise luminance only
ycc = cv2.cvtColor(rgb, cv2.COLOR_RGB2YCrCb
    )
ycc[..., 0] = clahe.apply(ycc[..., 0])
out = cv2.cvtColor(ycc, cv2.COLOR_YCrCb2RGB
    )
```

Practitioner's Note

Write `conv2d` and `equalize` yourself once, with loops, and check them against the library. After that, always use the library: it is faster and tested.

Part 6

Wrapping up

Key takeaways

1. **Noise** is random and independent between pixels; the signal is not. Averaging exploits this: $\sigma/\sqrt{N}$.
2. **Convolution** is a sliding weighted sum. The kernel is a template.
3. Output size: $O = \lfloor (N + 2p - K)/s \rfloor + 1$.
4. Linear + shift-invariant $\Leftrightarrow$ convolution: the foundation of CNNs.
5. Gaussian blur for Gaussian noise, median for outliers, bilateral to keep edges.
6. Blur = low-pass; sharpening = adding back high-pass (and noise).
7. Histograms show exposure and contrast; equalisation applies the CDF; CLAHE does it locally.

Before the next lecture

Practise (tutorial notebook `W1D3_Filtering_and_Histograms`)

- Write `output_size` and `conv2d` from scratch; match `scipy.ndimage.correlate`.
- Time loops vs. NumPy vs. OpenCV vs. separable filtering.
- Run your own denoising shoot-out with a PSNR table.
- Implement `equalize` from the CDF.

Read

- Course notes, Chapter 3
- Szeliski (2022), §3.2–3.4

Next lecture: classical features

- Image gradients: where brightness changes
- The Canny edge detector, stage by stage
- Harris corners
- Scale space and SIFT
- HOG: describing shape with gradients

Credits

Photographs from Wikimedia Commons under the licences shown. Other images: scikit-image sample data (`camera`, `moon`).

`cat_persian`: Gourdo 1 at en.wikipedia, Public domain
`yaounde_taxis`: Bdx, CC BY-SA 4.0
`leopard_tree`: Davidvraju, CC BY-SA 4.0
`douala_ndokoti`: Minette Lontsie, CC BY-SA 4.0
`histo_breast`: Mikael Häggström, CC0

`douala_deido`: Happiraphael, CC BY-SA 4.0
`douala_2014`: Guaka, CC BY-SA 4.0
`malaria_smear`: OpenFlexure Project, CC BY-SA 4.0
`leopard_grass`: Senthiaathavan, CC BY-SA 4.0
`xray_tb`: B. Herreros et al., CC BY 4.0
`lake_chad_apollo7`: NASA, Public domain