Metric Convolutions: A Unifying Theory to Adaptive Image Convolutions
Supplementary Material

Symbol	Type	Description
α	$\in \mathbb{R}_+$	Strictly positive number, either appearing in the dual Finsler metric or as the anisotropy gain factor parameter.
$B_1^g(x)$	$\subset X$	Unit geodesic ball (UGB) at point x.
$B_1^\mathcal{I}(x)$	$\subset T_xX$	Unit tangent ball (UTB) at point x.
c	$\in \mathbb{N}$	Feature dimensionality. Grayscale images have $c = 1$ and are the default in this work. Colour images have $c = 3$, and image-like grid data can have arbitrary number of features c.
$c(t)$	$\in X$	Curve on the manifold, with t varying from 0 to 1.
c_{in}	$\in \mathbb{N}$	Input number of channels.
c_{out}	$\in \mathbb{N}$	Output number of channels.
d	$\in \mathbb{N}$	Manifold dimensionality. For regular images, the manifold is a surface and $d = 2$.
Δ	$\subset \Omega$	Support of the convolution kernel.
Δ_x^{def}	$\subset \Omega$	Deformed kernel support when considering pixel location x.
Δ^{dil}	$\subset \Omega$	Uniformly dilated kernel support.
Δ_x^{dil}	$\subset \Omega$	Dilated kernel support when considering pixel location x.
Δ_x^{cnt}	$\subset \Omega$	Shifted kernel support when considering pixel location x.
Δ^{ref}	$\subset \Omega$	Reference support. In the discrete world, it is typically the usual kernel $k \times k$ grid.
δ_{MSE}	$\in \mathbb{R}_+$	Normalised generalisation gap of the ***MSE***.
δ_x	$\in \mathbb{R}^d$	Offset sampling vector at point x for all pixels in the reference support of the convolution kernel.
δ_x^y	$\in \mathbb{R}^d$	Offset sampling vector at point x for pixel y in the reference support of the convolution kernel.
$\delta_{x,t}$	$: \Omega \to \mathbb{R}^c$	Diffused Dirac image at time step t.
$\mathrm{dist}_F(x, y)$	$\in \mathbb{R}_+$	Geodesic distance according to the metric F from points x to y.
dt	$\in \mathbb{R}_+$	Geodesic diffusion time step.
ε	$\in \mathbb{R}_+$	Small positive number used for numerical stability.
ε_L	$\in \mathbb{R}_+$	Small scalar controlling the maximum scale of the metric.
ε_ω	$\in (0, 1]$	Hyperparameter controlling the maximum tolerated asymmetry, with 1 being no asymmetry.
η	$\in \mathbb{R}_+$	Learning rate.
f	$: \Omega \to \mathbb{R}^c$	Image with c colour channels. By default in this paper, it has $c = 1$ channel.
F	$: X \times T_xX \to \mathbb{R}_+$	Finsler metric on the tangent bundle $X \times T_xX$ of manifold X.
F^*	$: X \times T_xX \to \mathbb{R}_+$	Dual Finsler metric.
F_x	$: T_xX \to \mathbb{R}_+$	Finsler metric on the tangent plane T_xX at point x on the manifold X.
F^γ	$: X \times T_xX \to \mathbb{R}_+$	Finsler metric parametrised by γ.
F_x^γ	$: T_xX \to \mathbb{R}_+$	Finsler metric, parametrised by γ, at point x.
g	$: \Omega \to \mathbb{R}^c$	Convolution kernel. It can be viewed as an image, just like f, or we can focus only on its support. Its values $g(y)$ are the weights of the convolution kernel.
γ	$\in \mathbb{R}^p$	Metric parameters, such as M for Riemannian metrics and (M, ω) for Randers metrics.
h_x	$: \Omega \to \mathbb{R}_+$	Finsler-Gauss kernel.
ι	$\in \mathbb{R}_+$	Average metric scale parameter.
k	$\in \mathbb{N}$	Edge size of discrete convolution filters, containing $k \times k$ values for two-dimensional image convolution.
L	$\in \mathbb{R}^{d\times d}$	Lower triangular matrix with positive diagonal defining the Cholesky decomposition of $M = LL^\top$. For images, it is a 2×2 matrix with only 3 non-zero entries.
$\tilde{L}$	$\in \mathbb{R}^{d\times d}$	Slightly modified version of the lower triangular matrix L by $\tilde{L} = L + \varepsilon_L I$ to control the maximum scale of the metric.
Λ	$\in \mathbb{R}^{d\times d}$	Diagonal matrix.
λ	$\in \mathbb{R}_+$	Strictly positive scalar.
λ_i	$\in \mathbb{R}$	Scalar, i-th eigenvalue.
$\tilde{\lambda}_i$	$\in \mathbb{R}_+$	Positive and stable modification of λ_i.
λ'_i	$\in \mathbb{R}_+$	Unscaled modification of λ_i.
M	$: X \to S_d^{++}$	Riemannian tensor. It fully encodes the Riemannian metric. At point x, $M(x)$ is a symmetric positive definite matrix. For standard image manifolds, $M(x)$ is 2×2. For conciseness, M also refers to $M(x)$ without ambiguity.
$\tilde{M}$	$\in \mathbb{R}^{d\times d}$	Perturbed version of the estimated metric M for stability.
M^*	$: X \to S_d^{++}$	Riemannian tensor of the dual Randers metric of the Randers metric with parameters (M, ω).
$m_x(y)$	$\in [0, 1]$	Modulation value for the convolution weight $g(y)$ when considering the pixel location x.
$\nabla f(x)$	$\in \mathbb{R}^d$	Gradient of f at pixel x.
$\nabla f(x)^\perp$	$\in \mathbb{R}^d$	Orthogonal of the gradient $\nabla f(x)$.
$\|\cdot\|_A$	$\in \mathbb{R}_+$	L_2 norm of a vector with symmetric positive definite matrix A, meaning $\|u\|_A = \sqrt{u\top Au}$.
Ω	$\subset \mathbb{R}^d$	Parametrisation domain. For images, it can be seen as the unit square of $\mathbb{R}^2$.
ω	$: X \to T_xX$	$\omega(x)$ is the tangent vector parametrising the linear drift component of a Randers metric at point x.
ω^*	$: X \to T_xX$	Linear drift vector field of the dual Randers metric of the Randers metric with parameters (M, ω).
$\tilde{\omega}$	$: X \to T_xX$	Modified scaled version of $\omega(x)$.
P_x^γ	$: \mathbb{R}^{d+1} \to \Omega$	Stencil of points to be geodesically flowed at unit speed.
$\mathbb{R}_+$	$\subset \mathbb{R}$	Set of positive numbers (includes 0).
R	$\in \mathbb{R}^{d\times d}$	Rotation matrix (or a triangular matrix in the QR decomposition).
R	$: X \times T_xX \to \mathbb{R}_+$	Riemannian metric on the tangent bundle $X \times T_xX$ of manifold X.
R_x	$: T_xX \to \mathbb{R}_+$	Riemannian metric on the tangent plane T_xX at point x on the manifold X.
r	$\in \mathbb{R}^d$	Vector whose normalisation is the first column of a rotation matrix.
$\tilde{r}$	$\in \mathbb{R}^d$	Perturbed version of r for stability.
$\tilde{r}_\perp$	$\in \mathbb{R}^d$	Orthogonal vector to $\tilde{r}$.
s	$\in \mathbb{R}_+$	Dilation factor. It is global as it is the same at all pixel locations x. It is also used as the radius integration variable when using polar coordinates.
$\tilde{s}$	$\in \mathbb{R}_+$	Estimated eigenvalue scale.
s_0	$\in \mathbb{R}_+$	Initial scaling factor for geodesic flow of a stencil of points.
$s_{\max}$	$\in \mathbb{R}_+$	Maximum tolerated eigenvalue scale.
$s_{\min}$	$\in \mathbb{R}_+$	Minimum tolerated eigenvalue scale.
s_x	$\in \mathbb{R}_+$	Dilation factor at point x. It is local as it can differ between pixel locations x.
σ	$: \mathbb{R} \to \mathbb{R}_+$	Sigmoid function.
$\tilde{\sigma}$	$: \mathbb{R} \to \mathbb{R}_+$	Modified sigmoid function.
σ_n	$\in \mathbb{R}_+$	Standard deviation of white Gaussian noise. It encodes the noise level.
$T_{\max}$	$\in \mathbb{R}_+$	Temperature hyperparameter of the Adam optimiser.
T_xX	$\equiv \mathbb{R}^d$	Tangent space at point x of the manifold X. For standard image manifolds, it is a two-dimensional plane and can be associated as $\mathbb{R}^2$ for a fixed coordinate system.
θ	$\in [0, 2\pi]$	Angle.
θ_x	$\in [0, 2\pi]$	Angle of the image gradient.
u	$\in T_xX$	Tangent vector.
u_θ	$\in \mathbb{R}^2$	Euclidean unit vector with angle θ: $u_\theta = (\cos\theta, \sin\theta)^\top$.
X	$\subset \mathbb{R}^d$	Manifold. For images, it is the same as the parametrisation space Ω, and thus can be viewed as the unit square of $\mathbb{R}^2$. This perspective is different from a common convention where images are viewed as embedded manifolds, for instance colour images would be curved two-dimensional surfaces embedded in $\mathbb{R}^3$.
x	$\in X$	Point of the manifold. For image manifolds, it is by extension the pixel coordinate position in Ω.
y	$\in \Omega$	Pixel location of the convolution kernel.
Δ_x	$\subset \Omega$	Support of the convolution kernel when considering the pixel location x.
$y_x(\theta, \gamma)$	$\in B_1^\mathcal{I}(x)$	Point on the unit tangent circle of x with angle θ for the metric F^γ.
$\mathcal{Z}(x)$	$\in \mathbb{R}_+$	Normalisation factor in the Finsler-Gauss kernel.
z_{k,c_n}	$\in \mathbb{R}_+$	Uniform initialisation value of kernel weights.
CNN	Acronym	Convolutional neural networks.
FKW	Acronym	Fixed kernel weights. It is an experimental setting where convolution kernels have fixed uniform weights.
LKW	Acronym	Learnable kernel weights. It is an experimental setting where convolution weights are learned.
MSE	Acronym	Mean squared error.
SC	Acronym	Scratch, for training convolution kernels from scratch.
TL	Acronym	Transfer learning, for training convolution kernels using transfer learning.
UGB	Acronym	Unit geodesic ball: set of points on the manifold within unit geodesic length according to the metric.
UTB	Acronym	Unit tangent ball: convex set of tangent vectors of unit length according to the metric.

Table 5. Table of notations. Most notations are used only in the appendix.

A. Finsler and Randers Metrics: Further Details

We refer the interested reader for more information on Finsler and Randers metrics to the specialised Finsler literature, such as [5, 54, 55]. The details mentioned in this section are well-known in the community, but we put them here so that our paper is self-contained.

A.1. Finsler Metric Axioms

We provide in Fig. 7 a simple visualisation of the Finsler metric axioms. The triangular inequality is equivalent to the convexity of the unit tangent balls. Positive homogeneity implies that the metric scales with the same ratio as tangent vectors when considering only positive scaling (no flips). This differs from regular homogeneity, as in Riemannian metrics, where the metric scales with the absolute value of the ratio of tangent vectors. In a homogeneous metric, if $\lambda > 0$ and $u \in T_x X$, then λu and $-\lambda u$ both have their metric scaled by the same number $|\lambda| = \lambda$, making the metric symmetric. In contrast, in a positively homogenous metric, the metric at vectors oppositely scaled λu and $-\lambda u$ is different, as $F_x(\lambda u) = \lambda F_x(u)$ and $F_x(-\lambda u) = \lambda F_x(-u)$, where $F_x(u) \neq F_x(-u)$ in general. Note that a homogeneous metric is also positively homogeneous. For this reason, Riemannian metrics, which are root-quadratic homogeneous metrics, are special cases of Finsler metrics (see Fig. 8). Randers metrics are special cases of Finsler metrics containing Riemannian metrics, but when $\omega \neq 0$ they only satisfy the positive homogeneity.

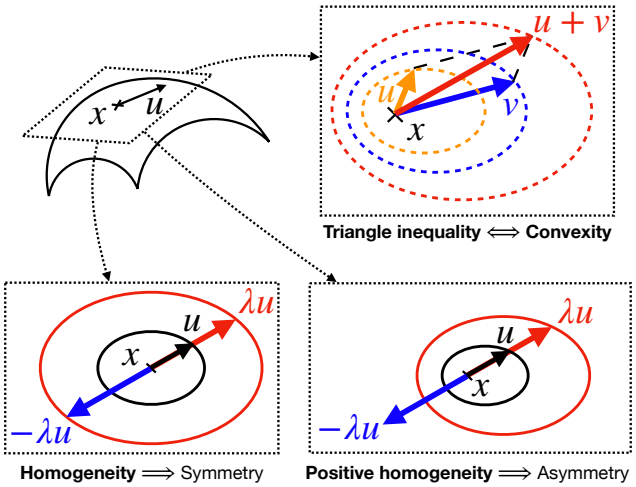

Figure 7. Illustration of metric axioms. The triangular inequality is equivalent to the convexity of unit tangent balls. Finsler metrics are positively homogeneous, which is less restrictive than regular homogeneity and allows asymmetric distances. In contrast, Riemannian metrics are homogeneous and thus always symmetric.

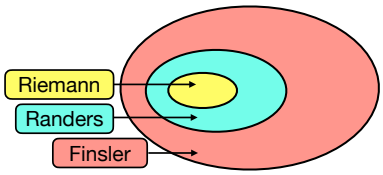

Figure 8. Venn diagram of Finsler metrics. Riemannian metrics are a special case of symmetric Finsler metrics. Randers metrics contain Riemannian ones, but are in general asymmetric.

A.2. Randers Metric Positivity

The positivity of the Randers metric F is ensured by $\|\omega\|_{M^{-1}} < 1$. In fact, we can generalise the statement in the following proposition that links the Randers metric with the regular L_2 norm.

Proposition 1. *Let $0 < \varepsilon < 1$. If for any point x on the manifold we have $\|\omega(x)\|_{M^{-1}(x)} \leq 1 - \varepsilon$, then the metric satisfies $F_x(u) \geq \varepsilon \|u\|_2$ for any $u \in T_x X$. In particular, if $\|\omega\|_{M^{-1}(x)} < 1$, then $F_x(u) > 0$ for any $u \neq 0$.*

Proof. All tangent vectors of $T_x X$ can be rewritten as $M(x)^{-1}u$. We then have

$$F_x(M(x)^{-1}u) = \sqrt{u^\top M(x)^{-1}u} + \omega(x)^\top M(x)^{-1}u \tag{5}$$

$$= \|M(x)^{-\frac{1}{2}}u\|_2 + \langle M(x)^{-\frac{1}{2}}\omega(x), M(x)^{-\frac{1}{2}}u\rangle. \tag{6}$$

The Cauchy-Schwartz inequality provides that

$$|\omega(x)^\top M(x)^{-1}u| \leq \|M(x)^{-\frac{1}{2}}\omega(x)\|_2 \|M(x)^{-\frac{1}{2}}u\|_2. \tag{7}$$

The assumption on ω can be rewritten as

$$\|M(x)^{-\frac{1}{2}}\omega(x)\| \leq 1 - \varepsilon. \tag{8}$$

Thus, for any tangent vector u, we get $F_x(M(x)^{-1}u) \geq \varepsilon \|M(x)^{-1}u\|_2$, and so for any such u, we obtain the desired result $F_x(u) \geq \varepsilon \|u\|_2$. □

A.3. Finsler Geodesic Distances

Given a Finsler metric F, the geodesic distance $\mathrm{dist}_F(x, y)$ between points x and y on the manifold is given by the minimum length of a smooth curve $c(t)$ from x to y

$$\mathrm{dist}_F(x, y) = \min_{\substack{c(t) \\ c(0)=x \text{ and } c(1)=y}} \int_{[0,1]} F_{c(t)}\big(\tfrac{\partial c}{\partial t}(t)\big) dt. \tag{9}$$

In particular, the orientation x to y is important for non Riemannian asymmetric metrics, as then $F_{c(t)}\big(\tfrac{\partial c}{\partial t}(t)\big)$ may differ from $F_{c(t)}\big(-\tfrac{\partial c}{\partial t}(t)\big)$.

A.4. Dual Randers Metric

The dual metric of a Finsler metric plays a key role in differential geometry on manifolds as it systematically appears in major differential equations, such as the Eikonal equation or the heat diffusion equation. Formally the dual metric of F is the metric F^* such that

$$F_x^*(u) = \max\{u^\top v;\ v \in T_x X, F_x(v) \leq 1\}. \tag{10}$$

One can easily verify that it satisfies the Finsler metric axioms. If F is a Randers metric parameterised by (M, ω), then the dual metric F^* is also a Randers metric. It is parameterised by (M^*, ω^*), which are explicitly given by (M, ω).

Proposition 2. *The dual of a Randers metric F parameterised by (M, ω) is a Randers metric F^*. If we denote $\alpha = 1 - \|\omega\|^2_{M^{-1}} > 0$, then the parameters (M^*, ω^*) of F^* are given by*

$$\begin{cases} M^* = \frac{1}{\alpha^2}\left(\alpha M^{-1} + (M^{-1}\omega)(M^{-1}\omega)^\top\right), \\ \omega^* = -\frac{1}{\alpha} M^{-1}\omega. \end{cases}$$

Proof. To ease notations, we drop the explicit dependence on x. Although the definition of the dual Randers metric is given in Eq. (10), since F is positive homogeneous it is also given by

$$F^*(u) = \max\left\{\frac{u^\top v}{F(v)};\ v \neq 0\right\}. \tag{11}$$

Therefore, the inverse solves the minimisation problem $\frac{1}{F^*(u)} = \min\left\{\frac{F(v)}{u^\top v};\ v \neq 0\right\}$. Likewise, by positive homogeneity, we have that the inverse of the dual metric satisfies the constrained minimisation problem

$$\frac{1}{F^*(u)} = \min\{F(v);\ u^\top v = 1\}. \tag{12}$$

We can also solve this constrained optimisation problem with Lagrangian optimisation. The Lagrangian is given by $L(v, \lambda) = F(v) + \lambda u^\top v$. To satisfy the KKT conditions, we differentiate L with respect to v and set the gradient to 0. The optimal v^* thus satisfies

$$M\frac{v^*}{\|v^*\|_M} + \omega + \lambda u = 0. \tag{13}$$

By computing the scalar product of this equation with v^*, and recalling that the constraint guarantees $u^\top v^* = 1$, we get, since $F(v^*) = \sqrt{{v^*}^\top M v^*} + {v^*}^\top \omega$, that $\lambda = -F(v^*)$. Recall that v^* solves the minimisation problem Eq. (12). Therefore,

$$\lambda = -\frac{1}{F^*(v)}. \tag{14}$$

Returning to Eq. (13), we can compute $\|\omega + \lambda u\|_{M^{-1}}$ as

$$\|\omega + \lambda u\|_{M^{-1}} = \frac{\|M v^*\|_{M^{-1}}}{\|v^*\|_M} = 1. \tag{15}$$

By squaring this equation, we obtain a polynomial of degree two for which λ is a root

$$\lambda^2 \|u\|^2_{M^{-1}} + 2\lambda\langle\omega, u\rangle_{M^{-1}} + \|\omega\|^2_{M^{-1}} - 1 = 0. \tag{16}$$

Let $\alpha = 1 - \|\omega\|^2_{M^{-1}} > 0$. The roots are then given by

$$\lambda_\pm = \frac{-\langle\omega, u\rangle_{M^{-1}} \pm \sqrt{\langle\omega, u\rangle^2_{M^{-1}} + \|u\|^2_{M^{-1}}\alpha}}{\|u\|^2_{M^{-1}}}. \tag{17}$$

Clearly, we have $\lambda_- < 0 < \lambda_+$ for $u \neq 0$. However, $\lambda < 0$ as $\lambda = -F(v^*)$ and the metric is always positive. As such, $\lambda = \lambda_-$. Inverting Eq. (14), we get

$$F^*(u) = \frac{\|u\|^2_{M^{-1}}}{\langle\omega, u\rangle_{M^{-1}} + \sqrt{\langle\omega, u\rangle^2_{M^{-1}} + \|u\|^2_{M^{-1}}\alpha}} \tag{18}$$

$$= \sqrt{u^\top \frac{1}{\alpha^2}\left(\alpha M^{-1} + M^{-1}\omega\omega^\top M^{-1}\right) u} - \frac{1}{\alpha}\langle\omega, u\rangle_{M^{-1}}, \tag{19}$$

where the classical trick $\frac{1}{x+\sqrt{y}} = \frac{x-\sqrt{y}}{x^2-y}$ was used to remove the square root from the denominator. We now identify the dual metric F^* as a Randers metric associated to (M^*, ω^*) as initially claimed

$$\begin{cases} M^* = \frac{1}{\alpha^2}(\alpha M^{-1} + (M^{-1}\omega)(M^{-1}\omega)^\top), \\ \omega^* = -\frac{1}{\alpha} M^{-1}\omega. \end{cases} \tag{20}$$

□

Going further. When studying the propagation of a wave front [54], the dual metric naturally appears yielding the Finsler Eikonal equation

$$F_x^*(-\nabla f) = 1. \tag{21}$$

Likewise, by observing that the heat equation in the Riemannian case is given by the gradient flow of the Dirichlet energy, we can descend on the dual energy $\frac{1}{2}F_x^*(u)^2$ to define the Finsler heat equation [5, 55]

$$\frac{\partial f}{\partial t} = \operatorname{div}(F_x^*(\nabla f)\nabla F_x^*(\nabla f)). \tag{22}$$

From the heat equation we can then compute various fundamental operators, mainly the Laplace-Finsler operator, a generalisation of the Laplace-Beltrami operator, that describes the shape [5]. These interesting constructions are

beyond the scope of this paper. For the interested reader, we point out the nice presentation and exploration of Finsler and Randers metrics and heat equation, leading to the Finsler-based Laplace-Beltrami operator [72], which was used instead of the traditional Laplacian to solve the shape matching problem [7–9]. . Additionally, we refer to [24] for a recent example of a beautiful application of elementary Finsler geometry for manifold learning and dimensionality reduction of asymmetric data.

A.5. Finsler and Hyperbolic geometry

Finsler geometry is a generalisation of Riemannian geometry where distances can depend on both position and direction, leading to more flexible and varied shapes. Within Finsler geometry, some spaces show properties similar to hyperbolic geometry [2, 45–47, 58, 73], a special type of Riemannian geometry having constant negative curvature. Extending further, Finsler-Lorentz geometry [1, 12, 56] allows the metric to have a Lorentzian signature, meaning it can describe not just spatial distances, but also time and causality as in relativity, all while keeping the direction-dependent qualities of Finsler spaces. This helps model more general and realistic behaviours in physics and geometry.

B. From Single-Channel Images to Manfifolds with Any Intrinsic or Feature Dimensions

B.1. Generalising our Unifying Metric Theory

Preliminaries. An image is a special type of two-dimensional manifold X, where for each point $x \in X = \Omega = [0,1]^2$ on the manifold we associate a feature vector $f(x) \in \mathbb{R}^c$. An other equivalent popular perspective for image manifolds is to see them as surfaces $X \subset \Omega \times \mathbb{R}^c \subset \mathbb{R}^{d+c}$ of the form $(x, f(x))$, but we do not adopt this perspective in this paper. The image feature dimensionality c is also called the number of channels of the image. Image manifolds can be generalised beyond two-dimensions. A d-dimensional hyperimage is a manifold $X = \Omega = [0,1]^d$ with feature function $f : X \to \mathbb{R}^c$. The manifold dimensionality of a hyperimage is d, regardless of the number of channels of the hyperimage: when embedding the hyperimage in $\mathbb{R}^{d+c}$, the manifold $(x, f(x))$ for $x \in \Omega$ is locally d-dimensional, meaning its tangent space $T_x X$ can be associated with $\mathbb{R}^d$. Points $x \in \Omega$ parametrising the hyperimage are called hypervoxels, generalising the concept of pixels when $d = 2$ and voxels when $d = 3$.

The particularity of image and hyperimage manifolds is that they possess a global uv parametrisation via $x \in \Omega = [0,1]^d$. For general manifolds, finding such a parametrisation is only possible locally[3]. Depending on the topology of the manifold, it may be possible to find such a global uv parametrisation, as for instance in genus 0 manifolds (topological planes or spheres). For general manifolds X satisfying this global uv parametrisation, e.g. a closed surface in $\mathbb{R}^3$, a manifold hyperimage can be viewed as a manifold enriched with a texture feature function $f : X \to \mathbb{R}^c$. Thanks to the global parametrisation, this formulation is mathematically equivalent to regular hyperimages where $X = \Omega = [0,1]^d$, e.g. $[0,1]^2$ for a closed surface in $\mathbb{R}^3$.

We are now ready to generalise our unifying theory on image convolutions to hyperimage convolutions.

Theoretical generalisation. Given a hyperimage of dimensionality d having c_{in} channels associated to the function $f : \Omega = [0,1]^d \to \mathbb{R}^{c_{in}}$, and a kernel hyperimage of dimensionality d with $c_{out} \times c_{in}$ channels associated to the function $g : \Omega \to \mathbb{R}^{c_{out} \times c_{in}}$. A hyperimage convolution produces a d-dimensional hyperimage with c_{out} channels, and is defined per output channel as

$$(f * g)_j(x) = \sum_{i=1}^{c_{in}} \int_\Omega g_{j,i}(y) f_i(x+y) dy, \qquad (23)$$

for $j \in \{1, \cdots, c_{out}\}$. This definition can be summarised in matrix-vector form as

$$(f * g)(x) = \int_\Omega g(y) f(x+y) dy, \qquad (24)$$

which uses the same formalism as Eq. (1).

Our entire theoretical discussion in Sec. 3 then immediately generalises to hyperimages of any dimensionality and feature channels. In particular, Theorem 1 would still apply. For instance, after discretising $\Omega = [0,1]^d$ into hypervoxels, the reference kernel support Δ can be given by a $k \times \cdots \times k$ grid with k^d hypervoxels, e.g. with hypervoxel indices $\{(i_1, \cdots, i_d) \in \{-1, 1, 0\}^d\}$ for $k = 3$. Dilation would scale this reference support by a factor s in all d dimensions. Shifted convolutions would shift the support by $\delta_x \in \mathbb{R}^d$, and deformable convolution would associate a shift vector $\delta_x^y \in \mathbb{R}^d$ for each of the k^d cells.

Note that the hypervoxel grid discretisation is a standard procedure for hyperimages as they share a global uv parametrisation via $x \in \Omega = [0,1]^d$. This procedure though is less frequent for manifolds equivalent to hyperimages, such as genus 0 manifolds like closed two-dimensional surfaces. Nevertheless, our formalism is not reliant on a specific type of sampling, it only requires the ability to approximately query feature values at non sampled locations of the manifold. This is usually possible for most parametrisations via some form of interpolation of feature values.

Finally, consider general manifolds that are not equivalent to hyperimages, where a global uv parametrisation does not exist. Our formulation still generalises to them as long

[3] For instance there is no global smooth uv parametrisation of the Klein bottle, even though it is a smooth two-dimensional manifold.

as the local kernel support Δ_x is small enough within the local neighbourhood where we can use a local uv parametrisation to approximate the manifold around point x.

B.2. Generalising Metric Convolutions

Metric convolutions easily generalise beyond single-channel image convolutions.

Feature dimensionality. As defined in Eq. (23), when images are not single-channel, the j-th output feature of the convolution is given by the aggregation of c_{in} convolutions between the single-channel images f_i and $g_{j,i}$. As such, to generalise metric convolutions to multi-channel data, it suffices to define c_{in} single-channel metric convolutions for each output dimension. This procedure would require $c_{out} \times c_{in}$ single-channel metric convolutions to compute the convolutions from c_{in} to c_{out} channel images. This generalisation procedure is the same as for generalising standard convolutions. As the required number of kernel samples, $k \times k \times c_{in} \times c_{out}$ for $k \times k$ convolutions, linearly increases with the number of input channels this can make high channel convolution operation particularly costly. To overcome the issue, the kernel g can be made sparse using separability, such as $g_{j,i} = 0$ for $j \neq i$ in depthwise convolutions [17, 33, 62, 63] or $g_{j,i} = 0$ for $j \notin C_i$ in group convolutions [39], where C_i is a group of input feature dimensions[4] for dimension j. The same can be done in metric convolutions. By using (block) separable filters, the size of the convolutions can drastically decrease, as depthwise convolutions use $k \times k \times c_{in}$ kernel locations and group convolutions use $\frac{k \times k \times c_{in} \times c_{out}}{G}$ for G groups.

Manifold dimensionality. For a d-dimensional manifold X, its tangent spaces T_xX are d-dimensional spaces and can be associated with $\mathbb{R}^d$. Given a fixed Cartesian parametrisation of $\mathbb{R}^d$, the Riemannian metric of such a manifold is defined as in two-dimensions: $R_x(u) = \sqrt{u^\top M(x) u}$. However, now $u \in T_xX = \mathbb{R}^d$ is a d-dimensional vector and the metric tensor $M(x) \in \mathbb{R}^{d \times d}$ is a $d \times d$ symmetric positive definite matrix. Likewise, Finsler metrics generalise in a similar fashion. In particular, the Randers metric of a d-dimensional manifold is then given by $F_x(u) = \sqrt{u^\top M(x) u} + \omega(x)^\top u$, where now $\omega(x) \in T_xX = \mathbb{R}^d$ is a d-dimensional vector. With these generalised definitions, we can then perform minor modifications to generalise our metric convolutions to d-dimensional manifolds. Note that we focus on hyperimage manifolds, which provide a shared universal parametrisation of the manifold and of the tangent spaces, given by the canonical Cartesian coordinates of $\mathbb{R}^d$. This avoids the need to compute changes of local coordinate systems, simplifies derivations, and allows simple interpolation of features beyond sampled values.

For illustration, we here present a generalisation of UTB metric convolution. Given the metric parameters $M(x) \in \mathbb{R}^d$ and $\omega(x) \in \mathbb{R}^d$, we need to compute explicitly the unit tangent ball, which is now a convex hypersurface of dimension $d-1$ in the tangent plane rather than a curve in the two-dimensional tangent plane. As it is convex and containing 0, it can be parametrised via $d-1$ angular spherical coordinates $\theta \in \mathbb{R}^{d-1}$. For Randers mertrics, the UTB is a hyper-ellipsoid given by the equation $u^\top M(x) u = (1 - \omega(x)^\top u)^2$ in u. Using this spherical reparametrisation, Eq. (3) still holds for points on the unit ball. Thus, the metric UTB convolution formula Eq. (4) still holds, with now θ being $d-1$-dimensional angular spherical coordinates and $s \in \mathbb{R}$ is the scalar radius coordinate.

We can generalise our discretisations of metric convolutions. Focusing once again for simplicity on UTB metric convolutions, the Cholesky decomposition approach would require encoding the symmetric positive definite matrix M as $M = LL^\top$ where L is a lower triangular matrix, thus encoded by $\frac{d(d-1)}{2}$ values. For the spectral approaches, we would need to compute d values for the eigenvalues of M. To compute the orthogonal eigenbasis U, like in the 2-dimensional case, several approaches are possible. We could use the exponential map $U = e^S$ for some skew-symmetric matrix $S = -S^\top$, parametrised by $\frac{d(d-1)}{2}$ unconstrained entries. Another option is to encode U via Givens rotations $U = R_1 \cdots R_{d(d-1)/2}$ where each Givens rotation is a planar rotation in the plane of two of the coordinate axes, requiring more parameters for encoding as we need the angles (or their cosine and sine) and the chosen axes. Another option is to take the QR decomposition, since any matrix A can be decomposed as $A = QR$ where Q is orthogonal and R is upper triangular. We could thus encode $U = AR^{-1}$ requiring $d^2 + \frac{d(d-1)}{2} = \frac{d(3d-1)}{2}$ parameters, and making sure that the diagonal of R is non-zero[5]. Regarding ω, we can encode it in the same way as in the two-dimensional case, requiring now d numbers instead of 2. These decompositions can be tweaked manually according to heuristics, or can once again be learned. Note that the numbers required to compute M can either be learned directly or can be learned as output of an intermediate standard d-dimensional convolutions, which would guarantee signal-adaptive shift-equivariant metric computation and thus shift-equivariant metric convolution, as in the two-dimensional case.

Note that an important aspect of our convolutions is knowing how to query features at non-sampled locations. When using hyperimages, the canonical axes of the domain

[4] Standard convolution has a single group, i.e. $C_i = \{1, \cdots, c_{in}\}$, whereas depthwise convolutions has as many groups as input (and output) features, i.e. $C_i = \{i\}$.

[5] For instance by feeding the diagonal to any positive non-linearity followed by a positive perturbation of ε.

$\Omega = \mathbb{R}^d$ are universal and shared by all hyperimages and are the same in each tangent plane of each pixel position. In particular, non sampled locations are guaranteed to fall within a hypercube of uniformly sampled hypervoxels grid points. This implies that feature interpolation requires only a small number of support samples with analytically known locations and analytically derivable interpolation formula. This makes the interpolation cheap and fast to implement, and would be more challenging if we were working on non-hyperimage data with irregular samples locations that differ between data instances (e.g. pointclouds).

C. Proofs of Our Unifying Metric Theory

C.1. Proof of Theorem 1

Unlike most of the community, we are rephrasing the pre-existing convolutions in the continuum. For now, we put aside modulation, which breaks the weight sharing assumption of convolution. Dilated convolutions scale by a factor s the reference support Δ^{ref}, usually uniformly in the image, to provide a dilated support Δ^{dil}: $\Delta^{dil} = s\Delta^{ref}$. Dilated convolution is thus given by

$$(f * g)(x) = \int_{\Delta^{ref}} f(x + sy)g(sy)dy. \tag{25}$$

In the non-standard case of using a different scale s_x for each pixel x, the support changes per pixel $\Delta_x^{dil} = s_x\Delta^{ref}$, and then dilated convolution is defined as

$$(f * g)(x) = \int_{\Delta^{ref}} f(x + s_x y)g(s_x y)dy. \tag{26}$$

In all cases, dilated convolutions can be rewritten as

$$(f * g)(x) = \int_{\Delta_x^{dil}} f(x + y)g(y)dy. \tag{27}$$

Shifted convolutions, also called entire deformable convolutions, simply shift the reference support Δ^{ref} by an offset δ_x shared by all cells

$$(f * g)(x) = \int_{\Delta^{ref}} f(x + y + \delta_x)g(y + \delta_x). \tag{28}$$

As such, if we denote $\Delta_x^{ent} = \delta_x + \Delta^{ref}$, we have for entire deformable convolutions

$$(f * g)(x) = \int_{\Delta_x^{ent}} f(x + y)g(y)dy. \tag{29}$$

On the other hand, deformable convolution introduces different offset vectors δ_x^y for each entry in the reference kernel support Δ^{ref}. Therefore, it is given by

$$(f * g)(x) = \int_{\Delta^{ref}} f(x + y + \delta_x^y)g(y + \delta_x^y)dy. \tag{30}$$

If we now write the deformed support $\Delta_x^{def} = \{y + \delta_x^y; \forall y \in \Delta^{ref}\}$, then deformed convolution can be rewritten as

$$(f * g)(x) = \int_{\Delta_x^{def}} f(x + y)g(y)dy. \tag{31}$$

We have thus managed to express these various convolutions with the same formulation

$$(f * g)(x) = \int_{\Delta_x} f(x + y)g(y)dy, \tag{32}$$

where Δ_x is either Δ^{ref} in the standard case, Δ_x^{dil} in the dilated case, Δ_x^{ent} in the entire deformable case, and Δ_x^{def} in the deformable case. If we now break the weight sharing assumption of any of these convolutions by introducing modulation, we have mask numbers $m_x(y) \in [0, 1]$ that multiply the kernel values $g(y)$. As such, the convolutions become

$$(f * g)(x) = \int_{\Delta_x} f(x + y)g(y)m_x(y)dy. \tag{33}$$

We can then define the distribution $dm_x(y) = m_x(y)dy$ to prove the theorem and show that all these convolutions perform weighted filtering on some neighbourhood Δ_x sampled with distribution dm_x. □

C.2. On Modulation and Weight Sharing

We here further present and discuss modulation and how it breaks the weight sharing assumption.

Weight Sharing. In order to provide deterministic sampling locations, the modulation $m_x(y)$ is not usually understood in the literature as a probabilistic density from which to sample kernel locations. Instead, it is rather viewed as modulation density, or mask, depending on the location x, that is applied to the $k \times k$ convolution weights at position x. This means that instead of computing at location x the convolution between the two functions f and g with varying sampling distribution $dm_x(y)$, the convolution uses a uniform sampling distribution dy but convolves between functions f and each $g_x : y \mapsto g(y)m_x(y)$ at point x, explaining why the weights are not shared in modulation.

Additionally, implementation strategies of modulation can further deviate from the weight sharing assumption, removing ever more the essence of what makes a convolution. There are essentially two main ways modulation is implemented in existing works.

Modulating deformations. The first approach, as in [77, 80], is to encode the modulation as an array with as many entries as kernel samples, i.e. $k \times k$ values, that is to be pointwise multiplied with the kernel weights. This strategy does not aim to construct the modulation at all possible

continuous locations in the continuous kernel, but only focuses on the sampled locations, and by doing so it is not constrained to any geometric prior. This implementation is particularly useful for sparse sampling locations, when k is small and the reference grid Δ^{ref} is heavily deformed. The mask values are then generally constrained to $[0, 1]$ and optimised in the training process. While this approach slightly boosts performance, it is theoretically unnecessary, as zeroing out sample locations is less optimal that simply moving the filtered-out samples to better locations. In practice, the resulting modulation tends to focus on sample locations close to the original pixel, and filters out distant ones.

Modulating without deformation. Given the prior that mask weights should decay with distance, the second way to encode modulation is to define a parametric mask function satisfying the prior. This function defines the mask weights at all possible continuous locations of the continuous non-deformed kernel, and then needs to be queried at the sampled locations of the discrete kernel. A famous example is the FlexConv method [59] and its Gaussian modulation. Unlike the previous implementation, this approach is tailored for large kernel sizes $k \times k$, e.g. $k = 25$ instead of $k = 3$, making it expensive, as it relies on a continuous perspective of kernels[6]. Importantly, in this strategy, the kernels are not deformed in any way $\Delta = \Delta^{ref}$. By adopting large kernel size and not deforming the kernel, the discrete kernel densely approximates the continuous kernel. Instead of explicit kernel deformation, Gaussian modulation, with learnable mean and covariance, is then pointwise applied to the kernel at each location x. This allows the operation to focus more on a part of the kernel map at each pixel location.

From our universal metric perspective, convolution operations using modulation without deformations as in FlexConv can be understood from two perspectives. The first is that unit balls are always the same and large, equal to the reference kernel Δ^{ref}, but the sampling density is not uniform. In particular, the implicit metric is the uniformly scaled standard Riemannian metric. A more interesting perspective is to understand the unit balls of the implicit metrics to fit the elliptical effective support of the Gaussian kernel, with scale given by the covariance and asymmetrically offset by the mean. In this second perspective, we must relax the unit ball sampling assumption with a sampling that can extend beyond unit balls but with a smooth rather than binary decay. This new perspective might mislead us into thinking that such constructions are equivalent to our metric convolutions, both offsetting and deforming a circle into an ellipse, with FlexConv requiring to encode a larger global weight map to be cropped. However, this is incorrect and a major difference exists between both approaches.

[6] In fact, in FlexConv the kernel is encoded continuously with an implicit neural representation via a Multi-Layer Perceptron.

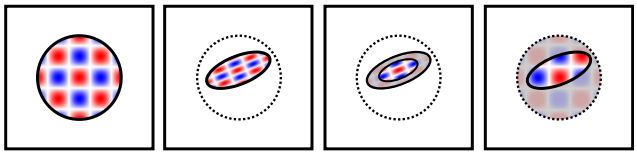

Figure 9. Illustration of different modulation strategies. Each image represents a kernel with weights given by the plotted texture. Left: reference kernel shape for standard convolution Δ^{ref}. Its texture g is the reference weight map shared by all pixels. Centre left: the reference kernel shape and its associated texture, i.e. the weights, are scaled, deformed, and shifted, resulting in a different convolution support $\Delta \neq \Delta^{ref}$ but same convolution weights. This case covers the approaches mentioned in the main manuscript, namely scaled, shifted, deformed, and our metric convolutions. Centre right: Modulation is added to the deformed convolution kernels, following the implementation strategy of [77, 80]. The weights of the kernel are preserved but some are attenuated by some factor. Right: Modulation is added by cropping the non-deformed reference kernel, e.g. via Gaussian masking as in FlexConv [59]. This requires an original kernel that is large enough, unlike the other methods, which is expensive. Additionally, weights of the kernel at the effectively sampled regions are no longer shared between points, even with binary modulation, unlike in the other methods. This second implementation implies that the statistics of the effective kernel weights are fundamentally different from those using the first strategy.

We illustrate this discussion in Fig. 9. For explanation clarity, binarise the Gaussian mask: only an ellipse of weights are then considered. In metric convolutions or other approaches deforming a reference kernel, these methods deform a reference kernel shape, e.g. a disk, and its texture to provide a new shape, with identically deformed texture, eventually modulated. On the other hand, in adaptive methods relying on modulation like FlexConv, the transformation is fundamentally different. In the second understanding of this modulation, the reference kernel shape and its texture are not deformed identically. Instead, a crop of a larger texture map is performed. This implies that there is no consistency in the convolution weights at different pixel locations, even in the simplified case where we binarise the Gaussian weight mask. FlexConv-like modulation thus fundamentally breaks the weight sharing assumption even more so than other modulation techniques, putting into question the "convolution" nomenclature of such methods.

C.3. Proof of Theorem 2

This result is well-known. It is a direct consequence of the positive homogeneity of the metric (see Fig. 10). Assume that the UTB $B_1^t(x)$ is given at any point x. Let $u \in T_x X$ be a non zero tangent vector. Then there exists a unique $v \in B_1^t(x)$ that is positively aligned with u, i.e. there exists $\lambda > 0$ for which $v = \lambda u$, that has unit metric $F_x(v) = 1$.

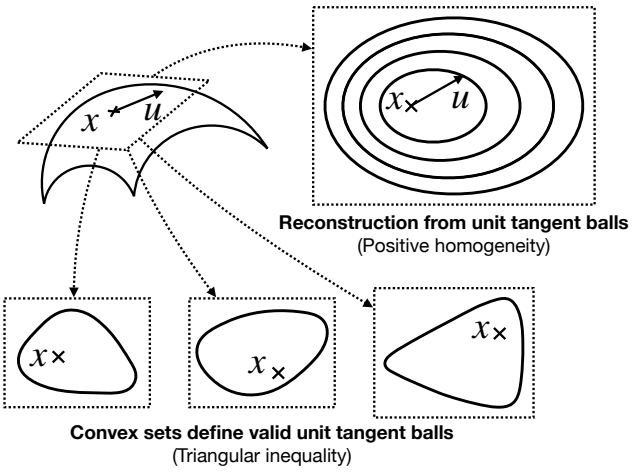

Figure 10. Thanks to their positive homogeneity, Finsler metrics are fully described by their unit tangent balls, which can be any convex set due to the triangular inequality.

Note that v is the intersection of the ray with direction u (with origin x) with the boundary of $B_1^t(x)$. We can then define $F_x(u) = \lambda$. We also define $F_x(0) = 0$. We can then easily check that F_x satisfies the positive homogeneity. Given any $\lambda' > 0$, we have $\lambda' u = \lambda' \lambda v$. By construction of F_x, we thus have $F_x(\lambda' u) = \lambda' \lambda = \lambda' F_x(u)$. As easily, we can check that F_x satisfies the triangular inequality and separability. As such, the provided UTBs implicitly defines a metric F_x. □

C.4. Reconstructing a Metric from Unit Geodesic Balls

Reconstructing the metric, or an approximation, is possible if further assumptions are introduced. For instance, if we are provided with the knowledge of distances within the unit ball, i.e. level sets within $B_1^g(x)$, or if the unit ball is sufficiently localised, i.e. $B_1^g(x)$ is sufficiently small (in Euclidean distance) to be approximated by its projection onto the tangent plane $T_x X$, then we can (approximately) reconstruct the unit tangent ball at point x and from there use Theorem 2 to recover the entire metric.

The issue for reconstructing metrics from UGBs is that geodesic distances consist in an integration of the metric along the tangents of the geodesics. By performing this summation, we can lose local information on the original metric. As an extreme counter-example, consider a small sphere, with radius smaller than $\frac{1}{2\pi r}$, then the unit ball at any point for the isotropic Riemannian metrics $M \equiv sI$ with $s \leq 1$ will cover the entire sphere, and likewise, other more complex metrics will provide the same unit ball. In this simplistic example, recovering the underlying metric is impossible without other prior knowledge.

C.5. Example of non-unique Metrics Explaining Discretised Sample Locations

Given a discretised sampling of a unit ball, the underlying continuous unit ball is ambiguous. As such, several metrics may provide continuous unit balls for which the given samples provide a good covering of its unit ball. We here provide two examples.

First, assume that we have finite samples and that an oracle provides us the information that these samples all lie on the unit tangent circle of some metric, i.e. $F_x(u) = 1$ for each of these samples u. Then any convex closed simple curve interpolating the provided samples yields a plausible Finsler metric F_x.

In general however, we are not aware of the distance of the sampled points within the unit ball. Sampled points do not necessarily lie at the same distance from x and no oracle provides their distance. Consider the following example. Assume we are provided with the reference template Δ^{ref} with 3×3 samples. We want to interpret it as a natural sampling of the unit tangent ball of some metric. A first natural possibility is to invoke the isotropic Riemannian Euclidean distance, for which the unit ball is the round disk[7]. Another possibility is to consider the traditional non-Riemannian L^∞ metric yielding unit balls in the shape of squares with straight edges parallel to those of the domain axes. Unlike the isotropic Euclidean suggestion, in the L^∞ one the samples on the border of the convex hull of the samples all lie on the unit circle of the metric.

C.6. Implicit Metrics of Existing Convolutions

We provide in Fig. 11 an intuitive visualisation of the metrics used for existing convolution methods. All of them can be well-approximated by adopting a tangent perspective to unit ball sampling of implicit Riemannian (for standard and dilated convolutions) or Finsler (for shifted and deformable convolutions) metrics. Our metric convolutions explicitly compute metrics and their unit balls, which then provide analytical sampling locations.

D. Further Discussions on the Theory of Metric Convolutions

Metric convolutions are versatile in their implementation, via heuristic or learnable metric designs. When implementing metric convolutions with learnable adaptive metric (see Fig. 12), we apply an intermediate standard convolution to learn a fixed number (5 to 7) of metric (hyper)parameters γ, from which we then extract the Randers metric parameters (M, ω) at each pixel using simple 2×2 matrix operations. From these parameters, we can then analytically compute

[7] It would be scaled so that the radius of the unit circle is in $[\sqrt{2}, 2)$ in Euclidean measurements

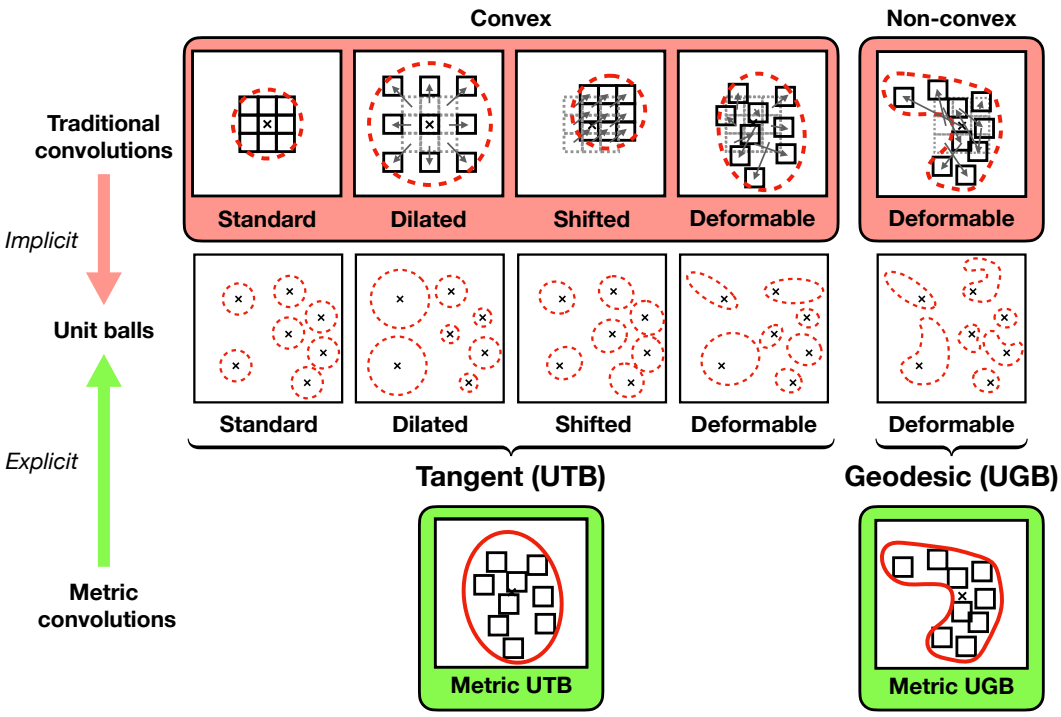

Figure 11. Traditional convolutions sample pixel neighbourhoods, which can be understood as implicitly sampling unit balls of implicit metrics. Metric convolutions explicitly compute the metric and its unit balls, which are then analytically sampled. By using Randers metrics, unit tangent ball (UTB) metric convolutions provide sampling neighbourhoods similar to those of traditional methods. Extreme non-convex sampling neighbourhoods can be provided by unit geodesic ball (UGB) metric convolutions.

kernel sampling locations of the unit tangent balls in a simple closed form that will be used for the final averaging of the metric convolution. The choice of standard convolution as intermediate hyperparameter extractor echoes the design of deformable [25, 80] and shifted [77] convolutions, providing a shift-equivariant metric convolution. A major difference though is that in these competing methods, the intermediate convolution computes directly the offset sampling locations, without any additional metric prior. Additionally, as the offsets are independent for each kernel cell in deformable convolution, the intermediate standard convolution requires $2k^2$ output channels, making this operation particularly costly as the kernel size grows. This contrasts with our metric convolution, which requires a fixed number of output channels for the intermediate convolution regardless of the number of kernel samples.

D.1. Computing the Metric from 5 Learnt Numbers: Cholesky Approach

The most general metrics we consider in this work are Randers metrics F, which are parameterised by (M, ω), where $M(x) \in \mathbb{R}^{2\times 2}$ is a symmetric definite positive matrix and $\omega(x) \in \mathbb{R}^2$ must satisfy $\|\omega(x)\|_{M^{-1}(x)} < 1$. Here, we will consider a unique location x, thus for conciseness, we drop the explicit dependence on it.

This discussion explains how to compute the metric parameters γ from 5 numbers using a Choleksy-based approach. Recall the fundamental linear algebra result that symmetric definite positive matrices possess a Cholesky decomposition and vice versa.

Proposition 3 (Cholesky decomposition)**.** *If $M \in \mathbb{R}^{d\times d}$ is a symmetric definite positive matrix, then there exists $L \in \mathbb{R}^{d\times d}$ that is lower triangular and with only positive diagonal entries such that $M = LL^\top$.*

In our case, we can reparameterise the Riemannian metric matrix M with its Cholesky decomposition matrix L requiring only three numbers instead of 4. On the other hand, ω needs two. As such, our metric convolutions require an intermediate operation to compute 5 numbers per pixel location to fully describe the metric. By analogy with deformable convolution, we chose to use a standard convolution with only 5 output channels. Nevertheless, several issues remain. First, we need to make sure that M does not become singular. This can happen through uncontrolled optimisation if for instance the lower right entry $L_{2,2}$ becomes close to 0. Secondly, we need to enforce the norm constraint on ω for the metric to remain positive. To overcome these challenges, we used the following strategy.

To avoid the non singularity of M, we construct $\tilde{L} =$

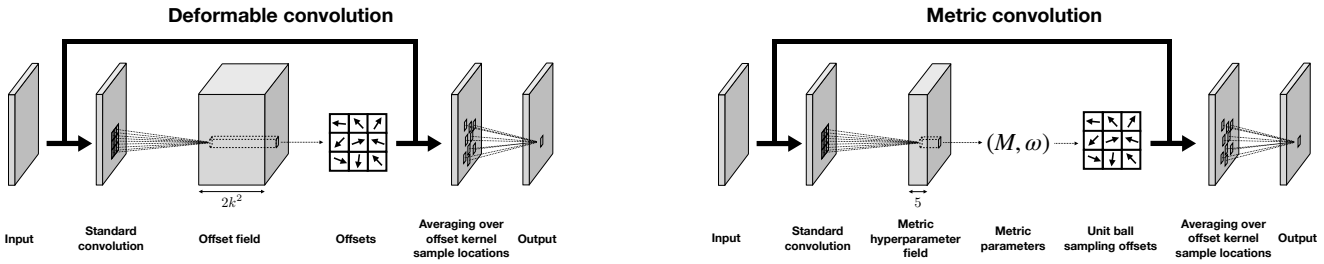

Figure 12. Metric convolutions can be implemented for learnable adaptive metrics with a similar design to deformable convolutions relying on an intermediate standard convolution. Instead of directly computing deformation offsets of the deformable convolution, which is costly as it requires $2k^2$ output channels for the intermediate convolution for a $k \times k$ convolution, we compute metric hyperparameters, which are a fixed number independent of the number of samples. From those, we can compute the metric and analytically sample unit tangent balls with simple and cheap operations involving solely 2×2 matrix multiplications. The use of standard convolution as metric parameter extractors, along with polar sampling schemes, makes metric convolutions a shift-equivariant operation by design.

$L + \varepsilon_L I$, where I is the identity matrix and $\varepsilon_L > 0$ is a small number controlling the maximum scale of the metric[8]. The Riemannian component is then given by $M = \tilde{L}\tilde{L}^\top$. In our experiments, we chose $\varepsilon_L = 0.01$.

To enforce the positivity of the metric, we introduce another hyperparameter $\varepsilon_\omega \in (0, 1]$ and devise a strategy to enforce $\|\omega\|_{M^{-1}} \leq 1 - \varepsilon_\omega$. This choice would guarantee that the metric does not accumulate around 0 as then $F_x(u) \geq \varepsilon_\omega \|u\|_2$ for any tangent vector u following Proposition 1. Taking $\varepsilon_\omega \to 1^-$ forces metric symmetry, whereas $\varepsilon_\omega \to 0^+$ allows the strongest asymmetry. Note that $\varepsilon_\omega = 1$ is equivalent to taking $\omega \equiv 0$. The strategy is the following. Compute $\|\omega\|_{M^{-1}} \in [0, \infty)$, and feed it to a modified sigmoid function to get a new number in $[0, 1 - \varepsilon_L)$. Recall that the sigmoid is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$. Our modified sigmoid is thus $\tilde{\sigma}(x) = 2(1 - \varepsilon_\omega)(\sigma(x) - \frac{1}{2})$. The computed number $\tilde{\sigma}(\|\omega\|_{M^{-1}}) \in [0, 1 - \varepsilon_\omega)$ represents the desired M^{-1}-norm of the Randers drift component. A such, we could take $\tilde{\omega} = \frac{\tilde{\sigma}(\|\omega\|_{M^{-1}})}{\|\omega\|_{M^{-1}}}\omega$. An issue arises though if we are learning the metric from data and initialise with $\omega \equiv 0$, as the square root has divergent gradient at the origin. To avoid this issue, we replace the computation of $\|\omega\|_{M^{-1}}$ by the more stable $\sqrt{\omega^\top M^{-1}\omega + \varepsilon}$, where $\varepsilon > 0$ is a small number, typically $\varepsilon = 10^{-6}$, that we systematically use to avoid instabilities, e.g. divisions by 0. As such, the modified drift component forming the metric is $\tilde{\omega} = \frac{2(1-\varepsilon_\omega)(\sigma(\sqrt{\omega^\top M^{-1}\omega+\varepsilon})-\frac{1}{2})}{\sqrt{\omega^\top M^{-1}\omega+\varepsilon}}\omega$.

We summarise the approach in Algorithm 1. Thus, when learning the metric, we use the metric defined by $(\tilde{M}, \tilde{\omega})$, and performed gradient descent on the 5 parameters of L and ω (per pixel), which is possible since $\tilde{M}$ and $\tilde{\omega}$ are given by differentiable operations from L and ω.

[8]The smaller ε_L the larger the maximum scale, quadratically.

Algorithm 1 Metric computation from 5 numbers

Input: Five numbers $L_{1,1}$, $L_{1,2}$, $L_{2,2}$, ω_1, ω_2
Hyperparameters: $\varepsilon_L > 0, \varepsilon_\omega \in (0, 1], \varepsilon > 0$
Construct $L = \begin{pmatrix} L_{1,1} & 0 \\ L_{2,1} & L_{2,2} \end{pmatrix}$ and $\omega = (\omega_1, \omega_2)^\top$
Compute $\tilde{L} = L + \varepsilon_L I$
Compute $\tilde{M} = \tilde{L}\tilde{L}^\top$
Compute $\tilde{\omega} = \frac{2(1-\varepsilon_\omega)\left(\sigma(\sqrt{\omega^\top M^{-1}\omega+\varepsilon})-\frac{1}{2}\right)}{\sqrt{\omega^\top M^{-1}\omega+\varepsilon}}\omega$
Return $(\tilde{M}, \tilde{\omega})$

D.2. Computing the Metric from 6 or 7 Learnt Numbers: Spectral Approach

The previous Cholesky-based implementation (Appendix D.1) sometimes suffers from instabilities during training when combined with neural networks. This issue persisted when using an LDLT approach, where $LDL^\top = M$ and L is lower triangular with unit diagonal entries and D is positive diagonal matrix. We here present a more stable implementation to compute γ at the cost of one or two extra numbers to encode the metric.

As in Appendix D.1, we work with Randers metrics parametrised by (M, ω) and focus on a unique location x, allowing us to drop its explicit dependence on it for conciseness. Recall the fundamental linear algebra result that symmetric matrices can be diagonalised in an orthogonal basis.

Proposition 4 (Spectral theorem)**.** *If $M \in \mathbb{R}^{d\times d}$ is a symmetric matrix, then M can be diagonalised in an orthogonal basis. This means that there exists an orthogonal matrix $R \in \mathbb{R}^{d\times d}$ and a diagonal matrix Λ such that $M = R\Lambda R^\top$.*

As the dimensionality of the image surface manifold is $d = 2$, we could encode the rotation matrix R by an angle θ as $R = \left(\begin{smallmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{smallmatrix}\right)$. Thus, only 3 numbers θ, λ_1, and

λ_2 could suffice to encode M, as in the Cholesky approach. However, regressing raw angle values is well-known to be significantly harder than estimating their cosine and sine values. This is due to periodic nature of angles: raw angle values ε and $2\pi - \varepsilon$ for small $\varepsilon > 0$ have a large difference but they correspond to almost identical angles. Instead, given two unconstrained numbers $r \in \mathbb{R}^2$, we construct the rotation matrix R using $R = \frac{1}{\|\tilde{r}\|_2+\varepsilon}(\tilde{r} \mid \tilde{r}_\perp)$, where $\tilde{r} = r + \varepsilon$ to avoid singular R on rare instances[9] and $\tilde{r}_\perp = (-\tilde{b}, \tilde{a})^\top$ if $\tilde{r} = (\tilde{a}, \tilde{b})^\top$.

Since M is also positive definite, then its eigenvalues $\lambda_1, \dots, \lambda_d$ forming the diagonal of Λ, are strictly positive. Given two unconstrained numbers λ_1 and λ_2, we could construct the eigenvalue matrix $\Lambda = \begin{pmatrix} \tilde{\lambda}_1 & 0 \\ 0 & \tilde{\lambda}_2 \end{pmatrix}$, where $\tilde{\lambda}_i = |\lambda_i| + \varepsilon_L$ for $i \in \{1, 2\}$. The Riemannian component would then be given by $\tilde{M} = R\Lambda R^\top$. This strategy requires 6 numbers to compute the metric parameters γ.

However, we obtained marginally better results when separating the scale of the eigenvalues, as it introduces regularisation on them. Let s be an additional unconstrained number used to compute the scale of the eigenvalues. Denoting $\lambda'_i = 1 + 2(\sigma(\lambda_i) - \frac{1}{2}) = 2\sigma(\lambda_i) \in [0, 2]$ the "unscaled" eigenvalues centred around 1, we compute their scale as $\tilde{s} = \frac{s_{\text{min}}+s_{\text{max}}}{2} + 2\left(\sigma(s) - \frac{1}{2}\right)(s_{\text{max}} - s_{\text{min}}) \in [s_{\text{min}}, s_{\text{max}}]$, where s_{min} and s_{max} are user-defined minimum and maximum eigenvalue scales[10]. In our experiments, we took $s_{\text{min}} = 0.1$ and $s_{\text{max}} = 1.5$. Finally we use the eigenvalues $\tilde{\lambda}_i = \lambda'_i \tilde{s}_i$ to build the matrix $\Lambda = \begin{pmatrix} \tilde{\lambda}_1 & 0 \\ 0 & \tilde{\lambda}_2 \end{pmatrix}$. The Riemannian component is then provided by $\tilde{M} = R\Lambda R^\top$. This strategy, requiring 7 numbers to compute the metric, is our preferred strategy and we only report results for this implementation when referring to the spectral implementation.

For both strategies, we compute the linear drift component from two unconstrained numbers ω as in Appendix D.1. This means that we use $\tilde{\omega} = \frac{\tilde{\sigma}(\|\omega\|_{M^{-1}})}{\|\omega\|_{M^{-1}}}\omega$ satisfying the norm constraints for the positivity of the metric. As we use the spectral approach for training stability, we found that we can also improve stability and marginally results by avoiding propagating the gradients through the invert operation M^{-1}. To this end, we detach the gradient of the factor $\frac{\tilde{\sigma}(\|\omega\|_{M^{-1}})}{\|\omega\|_{M^{-1}}}$ in the calculation of $\tilde{\omega}$. We used this strategy for all results reported in this work referring to the spectral implementation.

We summarise the spectral approaches using either 6 or 7 numbers in Algorithms 2 and 3. Our preferred version uses 7 numbers as it strongly encourages stability and leads to comparable or marginally better results. All results provided in this work using the spectral approach use 7 numbers. Thus, when learning the metric, we use the metric defined by $(\tilde{M}, \tilde{\omega})$, and performed gradient descent on the 7 parameters encoding the metric parameters γ (per pixel), which is possible since $\tilde{M}$ and $\tilde{\omega}$ are given by differentiable operations from these 7 numbers.

In our experiments using a simplistic architecture – a single convolution layer for denoising (Sec. 5.1), we did not encounter stability issues and provide results using the Cholesky implementation for computing metric parameters γ from 5 numbers. In our experiments using complex architectures – well-established CNNs for classification (Sec. 5.2), we strongly benefited from improved stability and provide results using only the spectral implementation for computing the metric parameters γ from 7 numbers.

Algorithm 2 Metric computation from 6 numbers

Input: Six numbers $r_1, r_2, \lambda_1, \lambda_2, \omega_1, \omega_2$
Hyperparameters: $\varepsilon_L > 0, \varepsilon_\omega \in (0, 1], \varepsilon > 0$
Let $r = (r_1, r_2)^\top$
Compute $\tilde{r} = r + \varepsilon$ and $\tilde{r}_\perp = (-\tilde{r}_2, \tilde{r}_1)^\top$
Construct $R = \frac{1}{\|\tilde{r}\|_2+\varepsilon}\begin{pmatrix} \tilde{r}_1 & \tilde{r}_{\perp,1} \\ \tilde{r}_2 & \tilde{r}_{\perp,2} \end{pmatrix}$
Construct $\Lambda = \begin{pmatrix} |\lambda_1| & 0 \\ 0 & |\lambda_2| \end{pmatrix} + \varepsilon_L I$
Compute $\tilde{M} = R\Lambda R^\top$
Compute $\tilde{\omega} = \frac{2(1-\varepsilon_\omega)\left(\sigma(\sqrt{\omega^\top M^{-1}\omega+\varepsilon})-\frac{1}{2}\right)}{\sqrt{\omega^\top M^{-1}\omega+\varepsilon}}\omega$
Return $(\tilde{M}, \tilde{\omega})$

Algorithm 3 Metric computation from 7 numbers

Input: Seven numbers $r_1, r_2, \lambda_1, \lambda_2, s, \omega_1, \omega_2$
Hyperparameters: $0 < s_{\text{min}} \leq s_{\text{max}}, \varepsilon_\omega \in (0, 1], \varepsilon > 0$
Let $r = (r_1, r_2)^\top$
Compute $\tilde{r} = r + \varepsilon$ and $\tilde{r}_\perp = (-\tilde{r}_2, \tilde{r}_1)^\top$
Construct $R = \frac{1}{\|\tilde{r}\|_2+\varepsilon}\begin{pmatrix} \tilde{r}_1 & \tilde{r}_{\perp,1} \\ \tilde{r}_2 & \tilde{r}_{\perp,2} \end{pmatrix}$
Compute $\lambda'_1 = 2\sigma(\lambda_1)$ and $\lambda'_2 = 2\sigma(\lambda_2)$
Compute $\tilde{s} = \frac{s_{\text{min}}+s_{\text{max}}}{2} + 2\left(\sigma(s) - \frac{1}{2}\right)(s_{\text{max}} - s_{\text{min}})$
Compute $\tilde{\lambda}_1 = \lambda'_1 \tilde{s}$ and $\tilde{\lambda}_2 = \lambda'_2 \tilde{s}$
Construct $\Lambda = \Lambda = \begin{pmatrix} \tilde{\lambda}_1 & 0 \\ 0 & \tilde{\lambda}_2 \end{pmatrix}$
Compute $\tilde{M} = R\Lambda R^\top$
Compute $\tilde{\omega} = \frac{2(1-\varepsilon_\omega)\left(\sigma(\sqrt{\omega^\top M^{-1}\omega+\varepsilon})-\frac{1}{2}\right)}{\sqrt{\omega^\top M^{-1}\omega+\varepsilon}}\omega$
Return $(\tilde{M}, \tilde{\omega})$

[9] We add ε here as if rigorously $r = 0$, which happened in practice on clean noiseless data like MNIST, then the vector $\frac{r}{\|\tilde{r}\|_2+\varepsilon}$ would still be 0 leading to a singular matrix R.

[10] Recall that due to the inverse in Eq. (3), a smaller eigenvalue scale of M leads to longer unit balls along that direction. For instance, a scale of 0.1 corresponds to stretching the ball to 10 pixels.

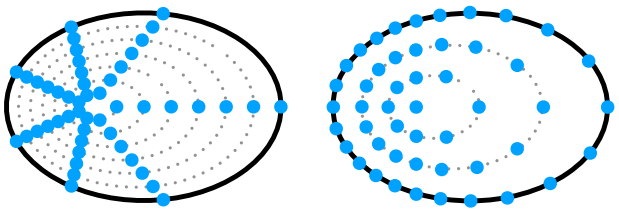

Figure 13. Polar kernel sampling strategies for $k \times k$ samples when $k = 7$. Left: grid sampling polar coordinates. Right: onion peeling strategy, analogous to what is is done in the standard square $k \times k$ grid. In both cases, the position of each sample is analytically given by a simple closed-form formula depending solely on the Randers metric parameters $\gamma = (M, \omega)$.

D.3. Polar Kernel Sampling strategies

In the continuum, the support Δ_x is given by the locations $sy_x(\theta, \gamma)$ where $s \in [0, 1]$ and $\theta \in [0, 2\pi]$. We design two complementary approaches to sample $k \times k$ grid points, as illustrated in Fig. 13.

Grid sampling. To provide a $k \times k$ sampled kernel, a natural approach is to uniformly grid sample s and θ with a $k \times k$ grid.

Onion Peeling. For very small k however, e.g. $k = 3$, polar grid sampling undersamples angles, unlike standard image convolutions using a regular $k \times k$ grid. For instance, 8 angles are considered for $k = 3$ in a regular grid. To better compare with standard convolutions for small k, we propose an onion peeling sampling strategy. A standard convolution sampling $k \times k$ grid can be understood as a succession of onion layers of pixels at L^∞ distance $k' \in \{0, \dots, \lfloor \frac{k-1}{2} \rfloor\}$ from the central pixel. In the k'-th layer, there are either 1 pixel if $k' = 0$ or $8k'$ pixels for $k' \geq 1$. We use this idea to design our onion peeling metric sampling: we sample $\lfloor \frac{k-1}{2} \rfloor + 1$ values $s \in [0, 1]$ uniformly, and for each layer index $k' \in \{0, \dots, \lfloor \frac{k-1}{2} \rfloor\}$, we sample either the original point x, which is given by any θ, if $k' = 0$ or $8k'$ angles $\theta \in [0, 2\pi)$ uniformly for $k' \geq 1$.

In our denoising experiments (Sec. 5.1), we work with larger k, thus we used a grid sampling strategy for them. In our classification experiments using well-established CNNs (Sec. 5.2), these neural architectures systematically rely on 3×3 convolutions, thus we utilized an onion peeling strategy for them.

D.4. Differentiating our Metric Convolution

We focus on the continuous case from Eq. (4). Differentiable changes of metric induce the variation of the unit ball in

$$\frac{\partial y_x(\theta, \gamma)}{\delta \gamma} = -\frac{1}{(F_x^\gamma)^2(u_\theta)} \frac{\partial F_x^\gamma}{\partial \gamma}(u_\theta) u_\theta. \tag{34}$$

Assuming that the convolution weights are fixed, i.e. $g(sy_x(\theta, \gamma)) = g_{\theta,s}$, differentiating the convolution with respect to the metric parameters gives

$$\frac{\partial (f * g)}{\partial \gamma}(x) = \int_{s,\theta} \left(s \frac{\partial y_x}{\partial \gamma} \right)^\top \nabla f(x + sy_x(\theta, \gamma)) g_{\theta,s} ds d\theta. \tag{35}$$

Likewise, if the metric is fixed, dependence on the weights is given by

$$\frac{\partial (f * g)}{\partial g_{\theta,s}}(x) = f(x + sy_x(\theta, \gamma)) \tag{36}$$

If needed, we can learn the parameters of the metric or the values of the kernel by gradient descent on some loss function L according to the dynamics

$$\begin{cases} \frac{\partial g_{\theta,s}}{\partial t} = -\frac{\partial L}{\partial g_{\theta,s}} \\ \frac{\partial \gamma}{\partial t} = -\frac{\partial L}{\partial \gamma}. \end{cases} \tag{37}$$

Naturally, we can generalise this to any descent-based optimisation algorithm and discrete optimisation steps.

We can easily extend this discussion to our discretised version of Eq. (4).

D.5. Shift-Equivariance of Metric Convolutions

Denote $\mathcal{T}_t$ the shift-operator of vector t, defined as $\mathcal{T}_t(f) : x \mapsto f(x - t)$. We say that the convolution of f with kernel g is shift-equivariant if for any vector t, we have[11] $\mathcal{T}_t(f * g) = (\mathcal{T}_t(f) * g)$. This means that shifting the input image and applying the convolution operations yields the same output image, albeit shifted by the same amount. Note that when mentioning shift-equivariance in image convolution, only periodic padding guarantees perfect shift-equivariance, otherwise the operation is approximately shift-equivariant with small differences arising at the borders of the image. Generalising the argument for standard convolutions, we here implicitly assume periodic padding, and the same issue will arise for other types of common padding, such as constant 0 padding. Let us formally prove Theorem 3.

Proof. First, the value of g at the cell with polar coordinates (s_x, θ), where $s_x = sy_x(\theta, \gamma)$, is independent of x. This implies that the convolution kernels g in metric convolutions have shared weights. Schematically, the discrete $k \times k$ convolution kernel g is an array of $k \times k$ values independent of x, that will be pointwise multiplied at each location x with the $k \times k$ array of sampled values $f(s_x, \theta)$ for the $k \times k$ sampled locations (s_x, θ), and then summed together to yield the output convolution at x.

[11] Note that shift-equivariance should not be confused with shift-invariance, which is defined as $\mathcal{T}_t(f * g) = (f * g)$. Old terminology, prior to the rise of deep learning, used to call shift-invariance what is now properly called shift-equivariance.

Second, our metric parameters γ are computed based solely on shift-equivariant objects. For heuristic designs, these objects can be image gradient or its norm, which are naturally shift-equivariant. For the advanced learnable design, we use an intermediate standard convolution to extract γ directly. Since standard convolutions are shift-equivariant, the computed array of metric parameters γ at all locations x will be shifted by the same amount as the input image. In turn, this implies that the unit tangent balls of our metric convolutions are shift-equivariant.

Third, thanks to the universal Cartesian coordinate system of images, our sampling strategies are based on angles θ with the horizontal axis of the image domain $\Omega = [0, 1]^2$, which is invariant to image shifts. This implies that our sampling scheme will always sample the same location of a given unit tangent ball, regardless of its position in the image.

Combining these three properties, the output of the metric convolution of a shifted image f with a kernel g will be the same output as the original convolution with the same kernel g, albeit shifted by the same shift amount. This means that metric convolutions are shift-equivariant. □

D.6. On Difficulties for Fast Differentiable Unit Geodesic Ball Computations

In contrast to the UTB case, computing the unit geodesic balls (UGB) is expensive as it is not given by a simple closed-form expression. It requires instead finding geodesic curves and then integrating along them. Many approaches exist for geodesic distance computations, and most of them require solving differential equations. They are usually either the Eikonal equation, which describes the propagation of a wave front on the manifold, or the heat equation, as initially heat diffuses along geodesics.

In the traditional Riemannian case, a wide variety of solvers exist, even differentiable ones, such as the recent differentiable fast marching algorithm [3, 4], Varadhan's formula [70] or the idea of [19] to flow heat initially in one small time step, normalise the obtained gradient field and interpret the normalised field as the tangential components of the geodesics. Unfortunately, existing solvers are too expensive to be used in reasonable applications, such as a neural network module, and we would need to apply these solvers at least as many times as there are pixels in the image since distances are computed from the single starting points x.

An extra layer of complexity arises when using the less common Finsler metrics, for which even discretisation of differentiable operators becomes tricky. We here discuss some of these difficulties when using the idea from [19]. In Randers geometry, deriving linear solvers to length-related differentiable equations becomes highly non trivial. In the Riemannian case, this is not an issue. For instance, the Riemannian heat equation $\frac{\partial f}{\partial t} = \operatorname{div}(\nabla f) = -\Delta_M f$ is governed by the linear Laplace-Beltrami operator Δ_M. Many works handle its possible discretisation strategies, such as the popular cotangent weight scheme [27, 53]. The linearity allows [19] to diffuse heat from a Dirac image δ_x, that are one at pixel x and zero everywhere else, by solving a set of linear equations to perform a single time-backwards iteration $(I - t\Delta_M)\delta_{x,t} = \delta_x$. The time-backwards operation essentially imagines what the heat should look like after a small time step should it have originated from the Dirac image. A forward time difference scheme would struggle to do so as the gradient of the image is zero almost everywhere, and in turn the Laplace-Beltrami operator, and so almost no heat would be flown that way in a single step. This elegant solution becomes highly non trivial in the Randers case. This is why in our metric UGB convolution, we modify the approach from [19] and revert to many smaller time forward iterations to flow heat. Also, as it is unclear how to discretise operators in Randers geometry, we use a local solution rather than the global one, which provides an approximating solution satisfying some properties of the differential equation.

D.7. Details on our Naive Implementation of our Metric Unit Geodesic Balls Convolutions

Finding global solutions to the Finsler heat equation (Eq. (22)) is difficult. However, we can easily provide local solutions [55]. Local solutions merely satisfy some local properties of the differential equation. Our local solution, the Finsler-Gauss kernel h_x [55, 76], is explicit and is given by

$$h_x(y) = \frac{1}{\mathcal{Z}(x)} \frac{1}{t} e^{-\frac{(F_x^*(y))^2}{4t}} \tag{38}$$

where F_x^* is the dual Finsler metric and is equal to the invert metric in the Riemannian case, and $\mathcal{Z}(x) = \int \frac{1}{t} e^{-\frac{(F_x^*(y))^2}{4t}} dy$ is a normalisation factor. The dual metric is beyond the scope of this paper so we refer to the Appendix A.4 for more details on it. We will just point out that the dual of a Randers metric is also a Randers metric with explicit parameters (M^*, ω^*). We then perform a standard convolution using the Finsler-Gauss kernel to diffuse the heat from a Dirac image δ_x, that are one at pixel x and zero everywhere else, to produce the diffused Dirac image $\delta_{x,t}$ according to the update rule

$$\delta_{x,t+dt}(x') = \int \delta_{x,t}(x' + y) h_{x'}(y) dy. \tag{39}$$

We can then compute the normalised gradient field $-\frac{\delta_{x,t}}{\|\delta_{x,t}\|_2}$ to get the unit speed geodesic flow field, from which we need to compute a unit ball. To get a differentiable sampling Δ_x, we can flow a stencil of points Δ^{ref} close to x given by $P_x^\gamma(s, \theta, 0) = s_0 s y_x(\theta, \gamma)$ where s_0 is an optional

scaling factor and then flow for a fixed amount of time according to $\frac{\partial P}{\partial t} = -\frac{\delta_{x,t}}{\|\delta_{x,t}\|_2}$. This deforms the stencil and the obtained unit ball is no longer necessarily convex. If the initial stencil is sampled using $k \times k$ uniform polar grid points by analogy with the UTB case, the obtained stencil can be interpreted as covering the non convex unit geodesic ball (or its approximation).

This algorithm is significantly cheaper than traditional more accurate geodesic solvers, it is fully differentiable as in particular the unit ball is not obtained via a thresholding operation. However, it is still too costly to be used in real scenarii such as neural network modules. For instance, if the image has 256×256 pixels, we need to diffuse 65536 Dirac images of the same resolution and then flow 65536 sets of stencil. This either quickly occupies all available memory in RAM for single modest commercial GPUs or implies a slow sequential bottleneck for simply computing a single convolution operation.

E. Experiments

E.1. Implementation Considerations of Heuristic Geometric Designs of Metric Convolutions and Other Methods and Results

We here show how to design sample locations from geometry. We take uniform kernel weights $\frac{1}{k^2}$ and no learning is involved. We denoise the 256×256 greyscale cameraman image using standard, dilated, and our metric UTB and UGB convolutions, along with deformable convolution using random offsets due to their lack of interpretability.

As mentioned in the main paper, a natural desire for the shape of unit balls when considering denoising is to be wide along the orthogonal gradient direction $\nabla f(x)^\perp$ and thin along the image gradient $\nabla f(x)$. This shape avoids blurring out edges.

Unit Tangent Ball. In the UTB case, unit balls are easily given in closed form. We can thus sample them directly without having to pass through the dual metric. Our anisotropic Riemannian metric of parameter M favours the direction $\nabla f(x)^\perp$ by taking it as an eigenvector with smaller eigenvalue

$$M(x) = R_{\theta_x} \begin{pmatrix} \iota\left(1 + \alpha \frac{\|\nabla f(x)\|_2}{\max \|\nabla f\|_2}\right) & 0 \\ 0 & \frac{\iota}{1+\alpha \frac{\|\nabla f(x)\|_2}{\max \|\nabla f\|_2}} \end{pmatrix} R_{\theta_x}^\top, \tag{40}$$

where $\iota = 0.1$ controls the average metric scale[12], $\alpha = 100$ is an anisotropy gain factor, and R_{θ_x} is the rotation matrix with angle θ_x such that $(\cos\theta_x, \sin\theta_x)^\top = \frac{\nabla f(x)}{\|\nabla f(x)\|_2 + \varepsilon}$. The small $\varepsilon = 10^{-6}$ is added here for stability and to avoid dividing by 0 in uniform areas of the image. The image gradient is computed using a Sobel filter of size 3×3.

It is legitimate to consider symmetric neighbourhoods for denoising, i.e. $\omega \equiv 0$. Nevertheless, we also tried asymmetric metrics by controlling the scale of ω. We first compute

$$\tilde{\omega}(x) = \frac{\nabla f(x)^\top}{\|\nabla f(x)\|_2 + \varepsilon}, \tag{41}$$

and then for various scales $(1 - \varepsilon_\omega) \in \{0, 0.5, 0.9\}$, we choose

$$\omega(x) = (1 - \varepsilon_\omega) \frac{\tilde{\omega}(x)}{\|\tilde{\omega}\|_{M^{-1}(x)} + \varepsilon}. \tag{42}$$

Unit Geodesic Ball. In the UGB case, our proof of concept algorithm requires the use of the dual metric to guide the heat flow of the Dirac images. After normalising this initial flow, we reflow a stencil of points to obtain the sample locations. We use the same metric (M, ω) as in the UTB case, except that now $\alpha = 10$ and $\iota = 1$. From this metric, we can explicitly compute (M^*, ω^*) given Proposition 2. For each pixel x, we diffuse the Dirac image δ_x, equal to 1 at pixel x and 0 everywhere, until $t = 0.1$ with time steps $dt = 0.01$. This means that starting from δ_x we iteratively convolved with the Finsler-Gauss kernel 10 times.

We then define a stencil of $k \times k$ points $P(s, \theta, 0) = s_0 s y_x(\theta, (M^*, \omega^*)))$ with a uniform uniform grid of radial values $(s, \theta) \in [0, 1] \times [0, 2\pi)$ sampled $k \times k$ times. The stencil is to flow along the diffused Dirac image $\delta_{x,t}$. We diffuse for the same amount of time as the heat diffusion with the same time steps. Tuning s_0 is of interest but was not searched, we simply took $s_0 = 2$. For small s_0, all the points in the stencil will lie in the pixel x, and when using bilinear interpolation they can decentre early from x before this drift is magnified. This behaviour is compatible with what is observed in deformable convolution. Too large values of s_0 will place stencil points in areas with unreliable normalised flow as they are at most barely reached by the heat diffusion. Flowing is done using a simple time forward scheme.

Our suggested implementation is fully differentiable but prohibitively expensive from a space and time perspective for more common application such as neural network compatibility. It merely provides a proof of concept of UGB metric convolutions.

Unit Ball Plots. To improve visualisation in Fig. 5, we slightly modified the hyperparameters in the UTB and UGB case. In the UTB plots, we take $\alpha = 10$. In the UGB plots, diffusion of the Dirac image is done with time steps $dt = 0.1$ until time $t = 0.5$, whereas the stencil is flown with time steps $dt = 0.1$ until time 1.

[12] If $\nabla f(x) = 0$ then $\iota = 0.1$ creates a ball of 10 pixel edge radius.

Details of Other Methods. All convolutions used in Fig. 14 use $k \times k$ samples with $k = 11$. Dilated convolution has a dilation factor of 3. Deformable convolution uses a dilation factor of 1 and each offset of each kernel cell for each pixel location is randomly, independently, and uniformly chosen in $[-\frac{k}{2}, \frac{k}{2}]^2$. Standard convolution covers a $k \times k$ pixel area and lacks interpolation, which further filters out noise, yet our method employs it. For fairness, we also test a non-standard interpolated convolution with $k \times k$ uniform samples covering a fixed smaller area. In practice, the interpolated standard convolution uses $k \times k$ uniform samples in a 5×5 pixel area. We chose this area as it is a common size of non interpolated standard convolutions. As such, the interpolated standard convolution can be seen as a standard 5×5 convolution equipped with the extra filtering from interpolation. Note that interpolated standard convolutions can also be understood as non-standard dilations with dilation factor less than 1.

Results. Results in Fig. 14 show our metric convolutions outperforming standard, dilated, and deformable convolution. Metric convolutions offer interpretability to flexible convolutions and strong geometric adaptable priors beneficial for basic tasks like noise filtering. The asymmetric drift component ω degrades performance for Gaussian noise filtering but could be useful in intrinsically asymmetric tasks like motion deblurring. Providing interpretable components to neural networks is fundamental as most models fundamentally lack interpretability. This is even the case for the simplest CNNs trained on the most simple tasks [22].

E.2. Learning Filtering on a Single Image

In this experiment, we learn convolution kernel shapes for deformable and UTB convolutions with fixed uniform kernel weights using the Cholesky-based implementation. Learning is performed on a single noisy cameraman image with varying noise levels $\sigma_n \in \{0.1, 0.3, 0.5\}$ and tested on another noisy version of the same image with the same noise level. Gradient descent on the Mean Squared Error (MSE) loss is employed to optimise raw offsets and metric parameters, rather than those provided by a standard convolution as we only operate on cameraman images. This experiment evaluates if convolutions learn the image surface structure or just overfit to random noise. Convolutions use $k \times k$ samples with $k \in \{5, 11, 31, 51, 121\}$, and training runs for 100 iterations. Surprisingly, both methods require high learning rates (of magnitude 10^4 to 10^8), differing from typical rates or those in the original deformable convolution works [25, 80]. A fixed learning rate did not yield meaningful results in all setups, necessitating a search for a good learning rate each time (see Appendix E.4 for details).

Full quantitative performance is given in Tab. 6, where we give the MSE on the train and test image, along with the normalised generalisation gap $\delta_{MSE} = \frac{MSE_{test} - MSE_{train}}{MSE_{train}}$. We also provide full qualitative results in Appendix E.5.

E.3. Training Details for Learning Filtering on a Dataset

We trained all models using stochastic gradient descent for 100 epochs on the MSE loss with a learning rate chosen through logarithmic grid search (see Appendix E.4) that is the same for the sample locations and the kernel weights following the default methodology of [25]. Training involved various kernel sizes $k \in \{5, 11, 31\}$ on a single small commercial GPU. For $k = 31$, the batch size was reduced to 4 to fit GPU memory, and was 32 otherwise.

E.4. Learning Rate Search for Denoising Convolution Filters

The learning rate for denoising convolution filters, whether they be of deformable convolution or our metric UTB convolution, or having fixed or learnable kernel weights, were found with a learning rate finder using a logarithmic grid search on the train data for a single epoch [65]. We report in Tabs. 7 and 8 the chosen learning rates.

E.5. Further Visual Results on Learning Convolutions on a Single Image

We provide in Figs. 15 to 20 visual comparisons of learnt deformable and metric UTB convolutions, when learning only the shape of the convolution and keeping the filtering weights fixed. These results correspond to the quantitative ones in Tab. 6.

In each of these figures, two consecutive rows of plots correspond to results with a fixed noise standard deviation σ_n and number of samples $k \times k$ for the convolution. Each of these sets of two rows of plots are organised as follows. Top left is the groundtruth image and bottom left is the train and test MSE during training. Starting from the second column, the top row corresponds to train whereas the bottom one refers to test. Starting from the second column, from left to right: input noisy image, deformable convolution result, our metric UTB convolution results with different $\varepsilon_\omega \in \{0.9, 0.1\}$ controlling the scale of ω, with $\omega \equiv 0$ for $\varepsilon_\omega = 1$. Numbers provided correspond to the PSNR with respect to the groundtruth image, with higher scores being better.

Note that using an extremely high number of samples, e.g. 121×121, does not increase the size of the sampling domain for our metric UTB convolution as the unit ball does not depend on the sample size. Larger kernels imply more samples in the same unit ball. On the other hand, deformable convolution suffers from high number of samples as it relies on the reference template of 121×121 pixels, which in our experiments is half the image size in width. As

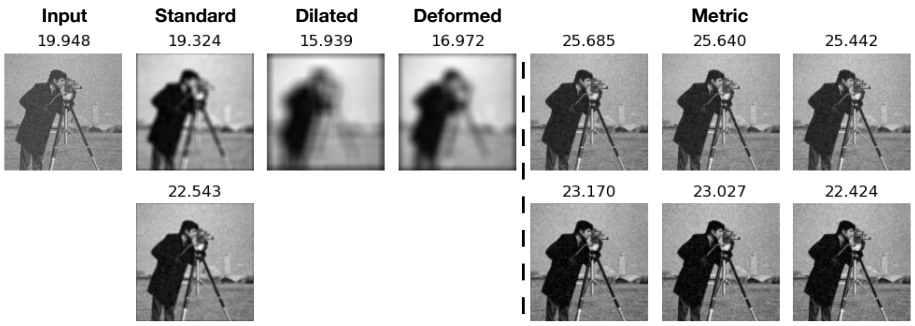

Figure 14. Denoising results using 11×11 samples from left to right: input, standard, dilated, randomly deformed, and metric convolutions with $\varepsilon_\omega = 1, 0.5, 0.1$. The bottom standard convolution uses interpolation on a smaller area. For our metric convolutions, the top row uses unit tangent balls, and the bottom row uses geodesic balls. Displayed values are the PSNR (higher is better).

		Deformable					**Unit tangent ball (ours)** $\varepsilon_\omega = 0.9$					**Unit tangent ball (ours)** $\varepsilon_\omega = 0.1$				
	k / σ_n	5	11	31	51	121	5	11	31	51	121	5	11	31	51	121
MSE	0.1	2.00E-3	4.49E-3	1.46E-2	2.24E-2	5.49E-2	1.68E-3	1.57E-3	1.54E-3	1.54E-3	1.54E-3	**1.60E-3**	**1.46E-3**	**1.44E-3**	**1.42E-3**	**1.41E-3**
	0.3	**7.26E-3**	8.97E-3	1.99E-2	2.83E-2	6.18E-2	8.13E-3	7.43E-3	7.58E-3	8.64E-3	8.28E-3	8.10E-3	**7.32E-3**	**7.45E-3**	**7.52E-3**	**7.82E-3**
	0.5	**7.86E-3**	**1.20E-2**	2.23E-2	3.07E-2	6.50E-2	1.85E-2	1.68E-2	**1.70E-2**	**1.70E-2**	1.72E-2	1.84E-2	1.70E-2	1.73E-2	1.72E-2	**1.71E-2**
δ_{MSE}	0.1	5.3	3.0	1.6	1.3	0.9	0.6	0.5	0.7	0.7	0.7	0.7	0.6	0.7	0.9	1.1
	0.3	265	74	28	18	6.6	1.1	0.9	1.1	0.8	1.2	1.3	1.1	1.3	1.4	1.5
	0.5	4554	971	210	113	35	0.8	1.3	1.8	2.0	2.2	1.3	1.3	1.6	2.1	2.9

Table 6. Denoising test MSE results (top) on a single noisy greyscale cameraman image when training on a different single noisy version, with noise level σ_n. Filters use $k \times k$ samples at each pixel location. Positional parameters of the convolutions are learnt, but weights are fixed. We also give the normalised generalisation gap δ_{MSE}. The parameter $\varepsilon_\omega \in (0, 1]$ controls the tolerated amount of metric asymmetry, with $\varepsilon_\omega = 1$ being symmetric. For all numbers, lower is better. Best test MSE are in bold, and second best are underlined.

such, in many pixel locations, the reference support overlaps with the outside of the image, where it is padded to 0, which makes it impossible for gradient-descent based strategies to learn meaningful offsets in such cases.

E.6. Implementation Considerations of CNN Classification

We here use the common ResNet terminology. All traditional ResNet architectures are a succession of *layers*. The initial layer, sometimes called *conv1*, has a single convolution module, along with other operations. Following the initial layer, comes a succession of four layers, named *layer1*, *layer2*, *layer3*, *layer4*. Each of these layers consist in a sequence of convolution *blocks*. These blocks can be *basic* for smaller networks like ResNet18, or *bottleneck* ones for larger versions like ResNet50 and ResNet152. Each block of a network has the same structure, up to a final pooling. Basic blocks have two 3×3 convolution modules, whereas bottleneck blocks have only one, when no downsampling is involved. None of these convolutions have an additional bias term. The first 3×3 convolution of the first block of every layer has a stride of 2, whereas all the other have a stride of 1. All convolutions use a dilation of 1 (no dilation).

In the experiments of Tabs. 3 and 4, we directly replace only the 3×3 convolution modules with their 3×3 adaptive counter-parts, i.e. deformable, shifted, and our UTB convolution. We use the same number of input and output channels and no bias. Only the convolutions in layer2, layer3, and layer4 are changed. Those in layer1 or conv1 are unchanged and remain standard. We also change the stride of the first convolution of the first block of layer4 from 2 to 1 and to avoid decreasing the receptive field we increase its dilation from 1 to 2. A dilation different from 1 impacts the position of the reference kernel Δ^{ref} of deformable and shifted convolutions, and does not impact our metric convolution. The methodology described here is a direct imitation of that of [25, 77, 80]. However, unlike [77, 80], we do not use modulation for simplicity as explained in the main paper: we wish to preserve the weight sharing assumption and sample uniformly the unit balls.

We propose to initialise our metric UTB convolution modules in the following way. Denoting c_{in} the number of input channels, kernel weights are initialised (and fixed in FKW) to $z_{k,c_{in}} = \frac{1}{c_{in}k^2}$. As for the weights of intermediate standard convolution with 5 output channels computing the metric parameters $L_{1,1}$, $L_{1,2}$, $L_{2,2}$, ω_1, and ω_2 in this order, we initialise them as follows per output channel: the first and third ones have uniform weights set to $z_{k,c_{in}}$, and the other ones are set uniformly to $\varepsilon = 10^{-6}$. In particular, this means that $\omega \approx 0$ initially, and the network must learn how much asymmetry is best. For simplicity however, we took $\omega = 0$ always, i.e. restricting the metric to Riemannian

	Deformable					Unit tangent ball (ours) $\varepsilon_\omega = 0.9$					Unit tangent ball (ours) $\varepsilon_\omega = 0.1$				
σ_n \ k	5	11	31	51	121	5	11	31	51	121	5	11	31	51	121
0.1	4.6E6	2.4E7	1.9E8	6.1E8	1.1E9	3.7E5	4.0E5	4.5E5	5.7E5	5.7E5	2.5E5	3.0E5	4.0E5	4.0E5	6.0E5
0.3	1.5E6	7.4E6	6.0E7	1.9E8	9.8E8	1.0E4	1.0E4	1.0E4	1.2E5	1.4E5	1.0E4	1.0E4	1.0E4	1.0E4	1.1E5
0.5	5.7E5	2.9E6	2.4E7	7.6E7	4.9E8	1.0E3	5.0E3	1.0E4	1.0E4	1.0E4	2.0E3	5.0E3	1.0E4	1.0E4	1.0E4

Table 7. Chosen learning rates for training the positional parameters on a single noisy image in the experiments of Tab. 6.

		Deformable						Unit tangent ball (ours) $\varepsilon_\omega = 0.1$						$\varepsilon_\omega = 0.9$					
		FKW			LKW			FKW			LKW			FKW			LKW		
	σ_n \ k	5	11	31	5	11	31	5	11	31	5	11	31	5	11	31	5	11	31
BSDS300	0.1	5.7E2	3.4E3	1.0E4	1.0E-2	1.0E-2	6.6E-4	6.0E0	3.1E0	7.0E-1	1.0E-2	1.0E-2	1.0E-3	6.1E0	1.1E0	1.0E0	5.0E-2	1.0E-2	1.0E-3
	0.3	3.2E2	1.5E3	1.0E4	1.0E-2	1.0E-2	6.6E-4	1.0E0	1.0E-1	1.0E-1	1.0E-1	1.0E-2	1.0E-3	1.0E0	1.1E0	1.0E-1	1.0E-2	1.0E-2	1.0E-3
	0.5	2.5E2	1.5E3	1.0E4	1.0E-2	1.0E-2	6.6E-4	6.1E-1	3.2E-1	4.5E-2	1.0E-2	1.0E-2	1.0E-3	1.0E0	3.8E-1	3.1E-2	1.0E-2	1.0E-2	1.0E-3
PASCALVOC	0.1	4.1E2	4.0E3	4.0E3	1.0E-2	1.0E-2	1.0E-3	1.2E1	4.3E0	5.0E-2	1.0E-1	1.0E-2	1.0E-3	1.4E1	1.5E0	1.0E0	1.0E-2	1.0E-2	1.0E-3
	0.3	3.0E2	4.0E3	4.0E3	1.0E-2	1.0E-2	1.0E-3	1.5E0	1.0E-1	5.0E-2	1.0E-2	1.0E-2	1.0E-3	4.0E0	5.0E-1	1.0E-1	1.0E-1	1.0E-2	1.0E-3
	0.5	4.1E2	1.1E3	4.3E3	1.0E-2	1.0E-2	1.0E-3	4.3E-1	1.0E-2	1.0E-2	1.0E-2	1.0E-2	1.0E-3	8.1E0	3.8E-1	1.0E-2	1.0E-2	5.0E-3	1.0E-3

Table 8. Chosen learning rates for training the positional parameters on noisy image datasets in the experiments of Tab. 2.

ones, by taking $\varepsilon_\omega = 1$. We also took $\varepsilon_L = 0.01$.

Like in the previous experiments, we test both fixing the kernel weights (FKW) of the non-standard convolutions to uniform values and learning only the sample locations, or learn simultaneously sample locations and the weights (LKW). Note that FKW has never been tested in the community of non-standard convolutions for neural networks. Prior works [42] start only with pretrained weights, up to module conversion, obtained on ImageNet [26] classification with vanilla modules. We argue that such a methodology does not properly reflect the strengths of convolutions with changeable supports. Indeed, we only switch a convolution with another one, thus the obtained network is still a CNN, albeit non-standard and theoretically more general. It should thus still provide good results when weights are learned from scratch. We thus train either from scratch (SC) or do transfer learning (TL) by starting from pretrained weights obtained on ImageNet.

All networks are trained for 240 epochs with the Adam optimiser [37] on the cross-entropy loss. We take a batch size of 128, a base learning rate of $\eta = 0.0001$, and we use cosine annealing [50] for scheduling the learning rate with maximal temperature $T_{\max} = 240$ as is commonly done. Following the common practice, images fed to the networks are centred and normalised following the dataset mean and standard deviation. When training on CIFAR with a single GTX 2080 Ti GPU, our metric CNN takes 7h and uses 1021MB GPU memory, compared to 5.5 hours and 940MB for a CNN using deformable convolutions (see Tab. 9). Although more expensive than deformable convolution, it is still faster than FlexConv while providing superior performance Tab. 10. Note that our unoptimised code leaves room for improvements. Optimising our code (e.g. fused CUDA kernels or C level code) could greatly improve speed. To reduce memory usage, analytical offsets (and the convolution result) could be computed in a small loop rather than being loaded simultaneously in memory, unlike in deformable convolution.

	Memory (MB)	Time (s)
FlexConv	-	127
Deformable	940	83
Metric UTB (Ours)	1021	105

Table 9. Peak GPU memory used and single epoch training time on CIFAR. FlexConv training time is taken from Tab. 3 in the original paper [59] for its best neural network FlexNet-16, but its maximum GPU utilisation is not reported there. Our method uses comparable memory to the more general deformable convolution but is slightly slower. Nevertheless, it is faster than the even more general FlexConv, while providing superior performance (see Tab. 10), even though FlexConv uses large convolution kernels with weights computed from expensive MLPs and then modulated by Gaussian masks.

Note though that although we only used fairly small datasets due to our technical limitations, they are nevertheless large enough to provide valuable insights [21, 23].

For all datasets, including both train and test splits, input images are first normalised according to the training dataset's mean and standard deviation. Since MNIST and Fashion-MNIST are curated datasets with objects centred and roughly aligned, we do not need data augmentation to train the models. However, the natural images in CIFAR-10 and CIFAR-100 are not, and therefore we apply data augmentation on training images by randomly cropping the input image to a patch, resizing the patch to the full image size, and then randomly horizontally flipping the image.

All CNNs with our metric UTB convolutions use the onion peeling sampling polar kernel sampling strategy (Appendix D.3) and the metric computation from 7 numbers

	MNIST	CIFAR10
FlexConv	99.7 ± 0.0	92.2 ($\pm 0.1\%$)
Metric UTB (Ours)	$99.7^{\dagger}$	93.1 ($\pm 0.1\%$)

Table 10. Reported performance of FlexConv from the original paper [59] (Tabs. 3 and 9) with their best model FlexNet-16, compared to our metric UTB convolution CNN. Although our network only uses 3×3 convolution samples and we did not use any modulation, our method outperforms FlexConv even though it is significantly more complex and expensive Tab. 9, requiring large convolution kernels, and thus high number of k, while also using Gaussian modulation. FlexConv results are averaged over three random seeds, whereas ours uses one on MNIST (†) and eight on CIFAR. Our performance corresponds to those in Tabs. 3 and 4 with precision rounded to the first decimal, like the results reported for FlexConv [59].

(Algorithm 3), except for those with fixed kernel weights (FKW) which use the version with 6 numbers (Algorithm 2). On CIFAR-10 and CIFAR-100, we got marginally better results when training the networks with learnable kernel weights (LKW) using an L1 regularisation loss on the weights of the intermediate convolutions with a Lagragian coefficient of 5000. On CIFAR-10 with learnable weights and transfer-learned (LKW-TL), we got even slightly better results when using 50 warmup epochs, where during the warmup the output of the intermediate convolution is multiplied by 0. This warmup imitates the baseline sampling strategy of a fixed kernel while still using our metric framework.

E.7. Further Ablation Experiments of CNN Classification

			ResNet18	ResNet50	ResNet152
TOP1	Standard		73,61 ($\pm 0,31\%$)	79,06 ($\pm 0,29\%$)	79,86 ($\pm 0,30\%$)
	Deformable		73,55 ($\pm 0,26\%$)	78,15 ($\pm 0,25\%$)	79,73 ($\pm 0,41\%$)
	Shifted		73,15 ($\pm 0,17\%$)	78,33 ($\pm 0,32\%$)	79,70 ($\pm 0,35\%$)
	Metric UTB (Ours)	Randers	**74,20** ($\pm 0,58\%$)	**79,17** ($\pm 0,60\%$)	**80,56** ($\pm 0,80\%$)
		Riemann	74,19 ($\pm 0,68\%$)	79,11 ($\pm 0,44\%$)	80,27 ($\pm 0,43\%$)
TOP5	Standard		91,83 ($\pm 0,20\%$)	94,76 ($\pm 0,17\%$)	94,53 ($\pm 0,22\%$)
	Deformable		91,50 ($\pm 0,23\%$)	94,08 ($\pm 0,17\%$)	94,51 ($\pm 0,23\%$)
	Shifted		91,37 ($\pm 0,15\%$)	94,16 ($\pm 0,15\%$)	94,30 ($\pm 0,15\%$)
	Metric UTB (Ours)	Randers	**92,31** ($\pm 0,36\%$)	**94,85** ($\pm 0,34\%$)	**94,94** ($\pm 0,32\%$)
		Riemann	92,26 ($\pm 0,28\%$)	94,81 ($\pm 0,19\%$)	94,75 ($\pm 0,23\%$)

Table 11. Mean test accuracies of different ResNet architectures with replacement of convolutions only in the last layers (*layer4*), using either standard or non-standard convolutions, with 10 independent runs per configuration. Higher is better. In parenthesis is the standard deviation (lower is better).

We here present additional ablation experiments on CIFAR100 evaluating a different replacement strategy for non-standard convolution layers and the use of Riemannian metric convolutions instead of Finsler ones.

Replacement layers. There is no standard practice for selecting which convolution layers of the CNN to replace with non-standard convolutions. However, it is common in the field to focus on deeper layers. Our design presented in the main paper and in Appendix E.6 follows the strategy of deformable convolution v2 [80], which expands on the original version [25], where only the very last layers were replaced. While the original work [25] found worsening or diminishing returns when replacing more layers, [80] argued this was due to the simplicity of the task and showed benefits when scaling up. To avoid misleading conclusions, we adopted the broader replacement strategy of [80] in the main paper.

That said, our tasks are simple, and replacing only the last layers, similar to [25] and also [42], would likely improve both speed and memory footprint while slightly boosting performance on these simple small-scale tasks. However, this might not reflect large-scale behaviour. Nevertheless, in Tab. 11, we report CIFAR100 (LW-TL) results over 10 runs with this modified setup, where only the convolutions in *layer4* are replaced, and we also test larger models. Similar to [25], we obtain better performance when replacing only the last convolution layers. However, we caution against generalising this result to larger scale problems, similar to what happened with deformable convolution [80].

As for full replacement, it is uncommon in the field. Even the recent advanced v3 version of deformable convolution – InternImage [71] tailored for large-scale foundation models – has first standard convolutions for shrinking the resolution, and also uses them for downsampling between deformable blocks. In summary, full replacement is not done due sub-optimal performance, high memory cost from early high-resolution features, as offsets are computed per pixel, and redundancy in adapting low-level filters, e.g. edge filters. Later layers capture more complex, semantic features and benefit more from sampling location adaptation. On small tasks, early layers pretrained on ImageNet are more robust, so replacing only final layers, which are more task-dependent, is preferable. Replacing earlier layers is more interesting as the task scales.

Riemann or Finsler metric convolution. Unlike standard and dilated convolutions, shifted and deformable convolutions sample asymmetrically around each pixel, requiring asymmetric metrics, i.e. Finsler, in our theory to offset unit balls (Figs. 1 to 3 and 11). We use Randers as Finsler metrics for their simple parametric form that generalises Riemannian ones. However, we can also use symmetric metrics for our metric convolutions, such as Riemannian metrics, i.e. $\omega \equiv 0$ or in our implementation $\varepsilon_\omega = 1$. We present the mean test classification performance obtained with Riemannian metric convolutions in Tab. 11, when replacing only the convolutions in the last layers (*layer4*).

As expected, performance degrades slightly from using the more general Randers metric. However, our symmetric Riemannian metric convolutions outperform the theoretically more general asymmetric deformable and shifted convolutions. This further proves that adopting an explicit metric perspective to convolution is beneficial and induces a powerful geometric regularisation.

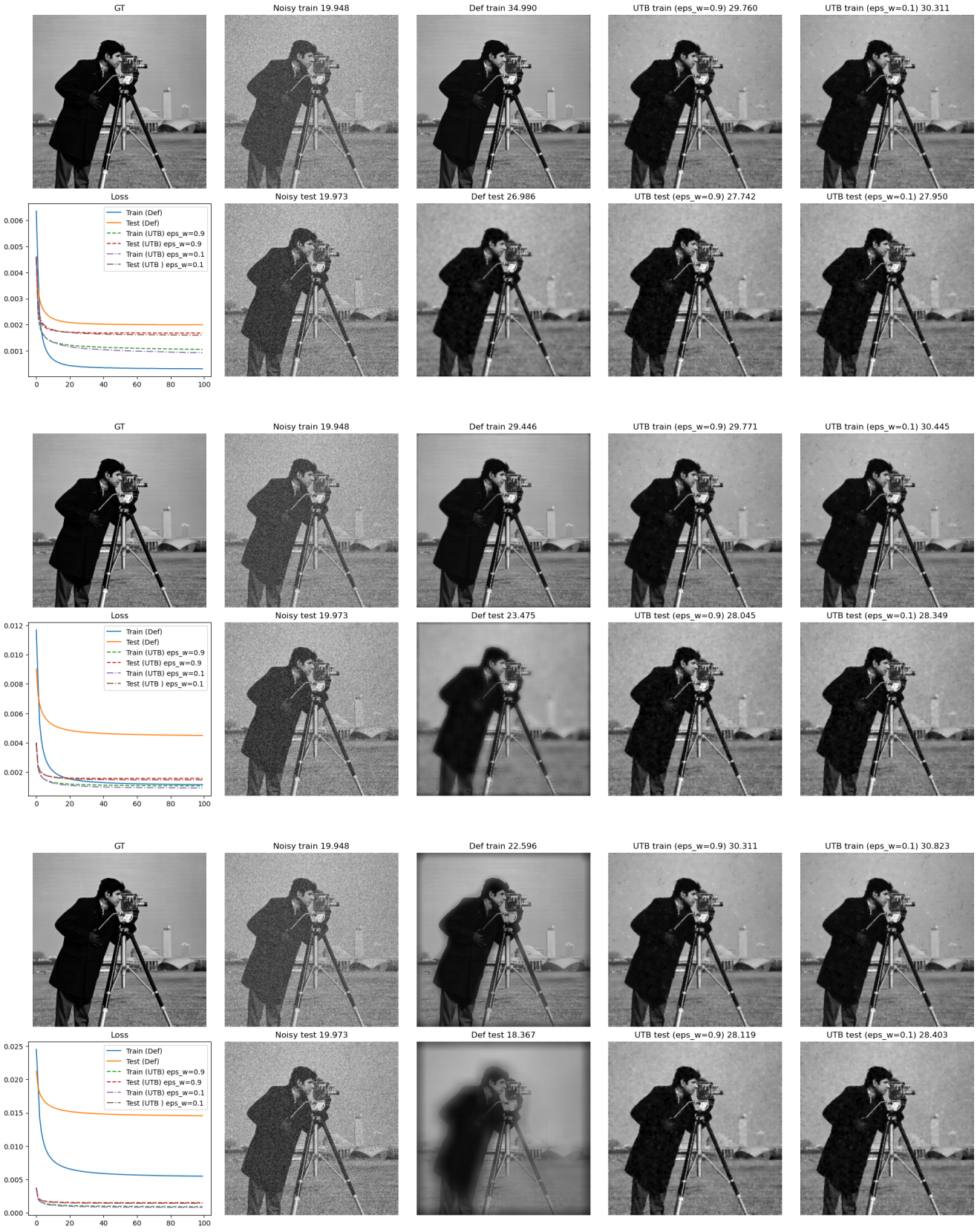

Figure 15. Results of learnt deformable and our metric UTB convolutions with $\sigma_n = 0.1$ and $k = 5, 11, 31$, from top to bottom.

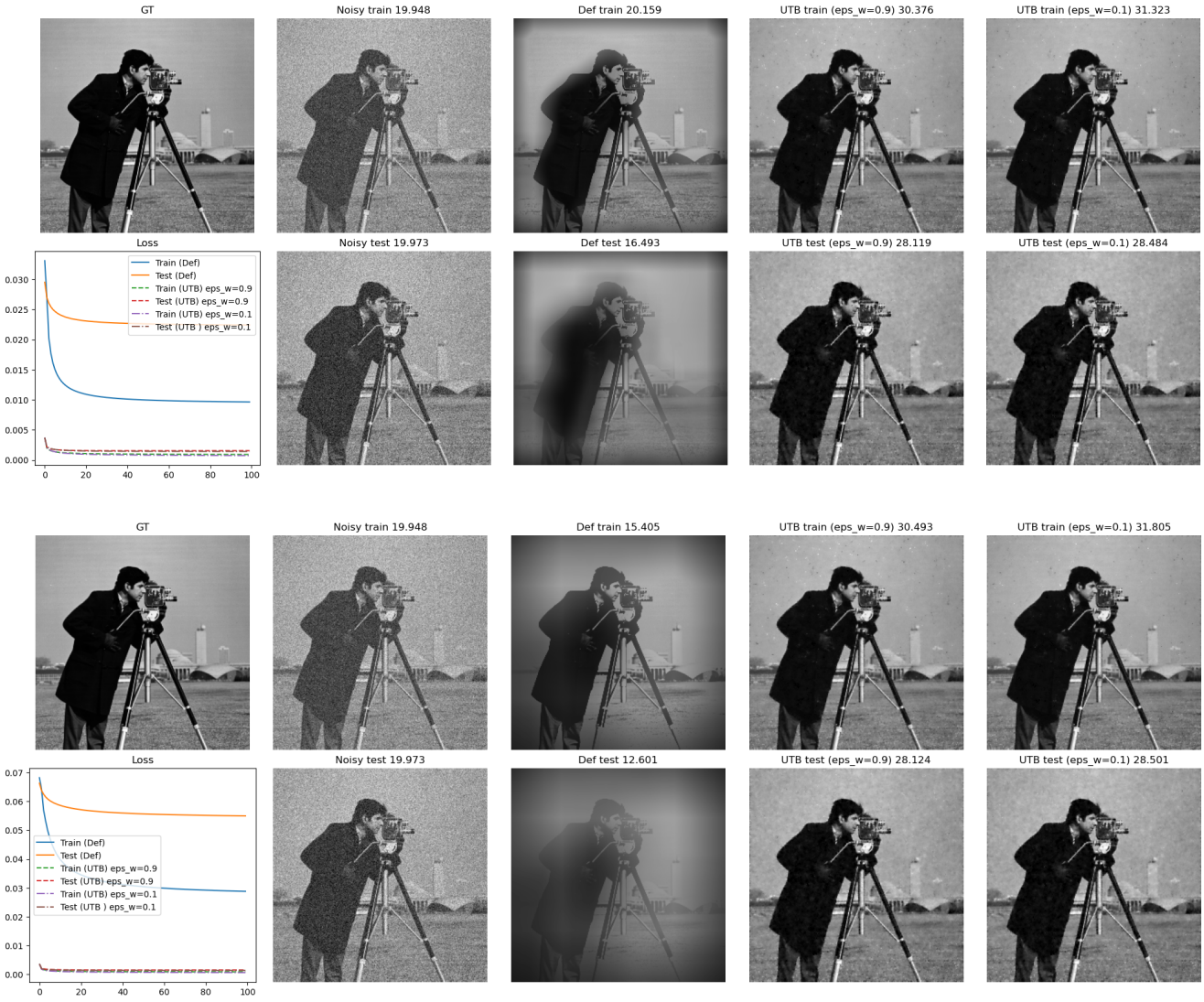

Figure 16. Results of learnt deformable and our metric UTB convolutions with $\sigma_n = 0.1$ and $k = 51, 121$ from top to bottom.

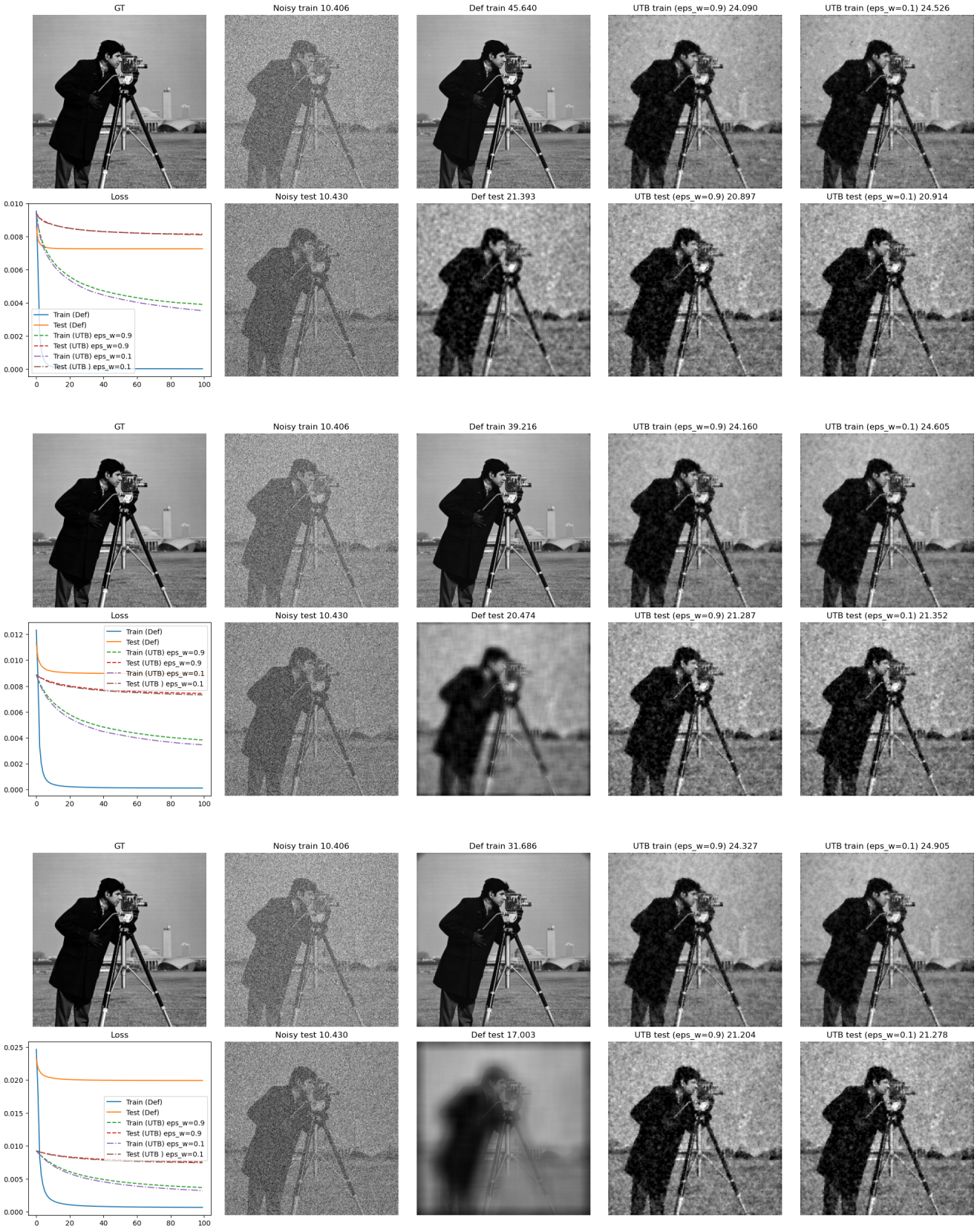

Figure 17. Results of learnt deformable and our metric UTB convolutions with $\sigma_n = 0.3$ and $k = 5, 11, 31$ from top to bottom.

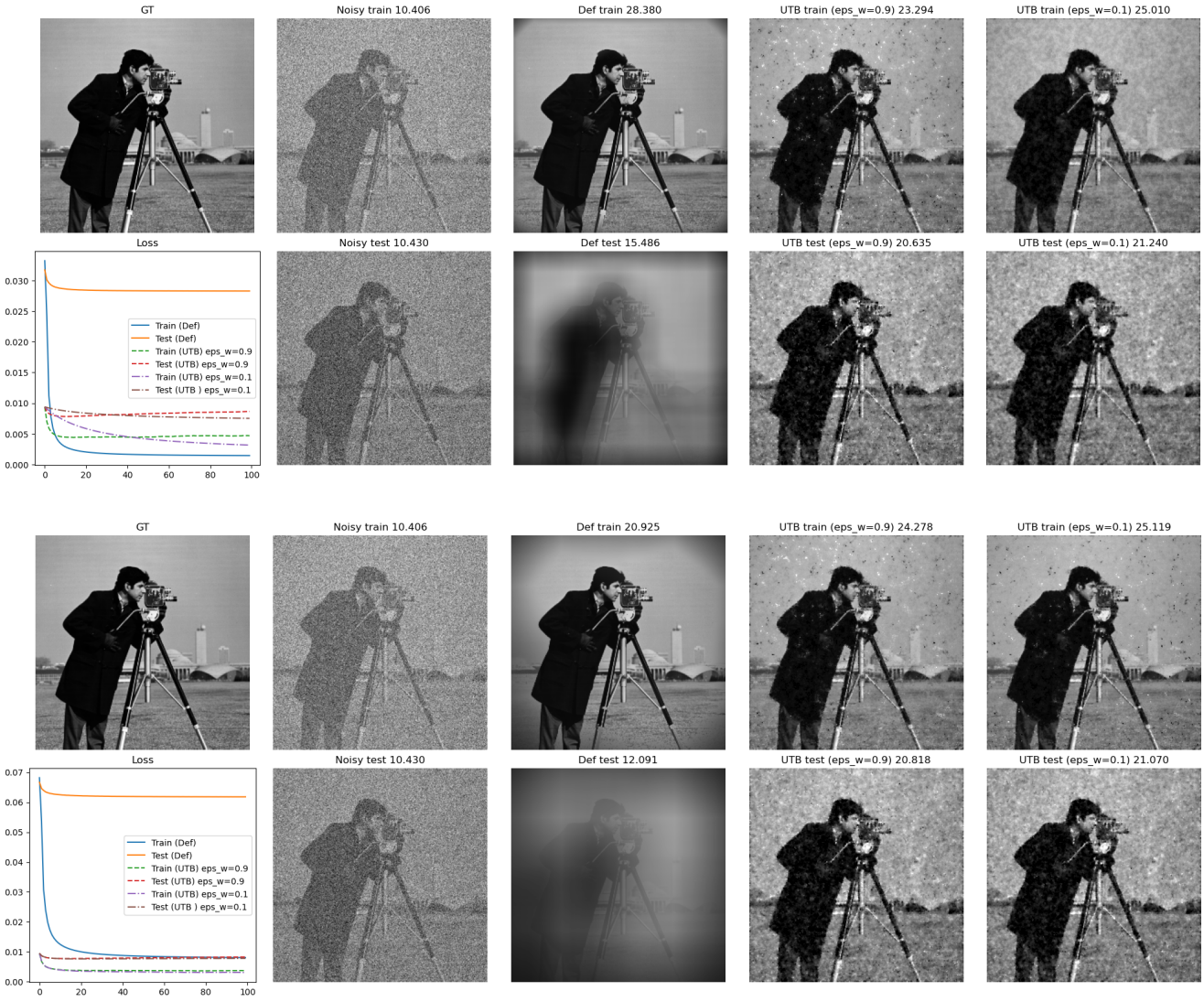

Figure 18. Results of learnt deformable and our metric UTB convolutions with $\sigma_n = 0.3$ and $k = 51, 121$ from top to bottom.

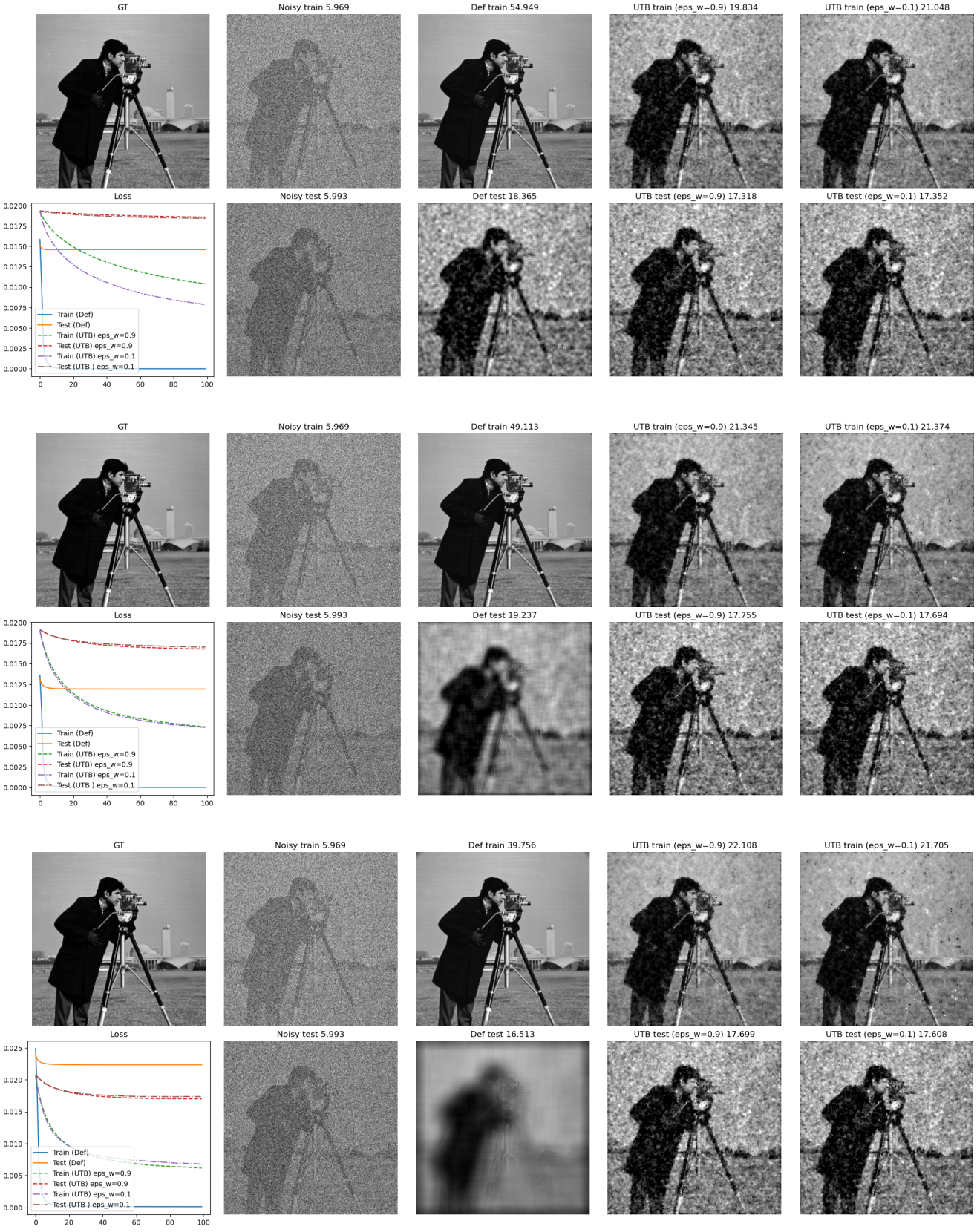

Figure 19. Results of learnt deformable and our metric UTB convolutions with $\sigma_n = 0.5$ and $k = 5, 11, 31$ from top to bottom.

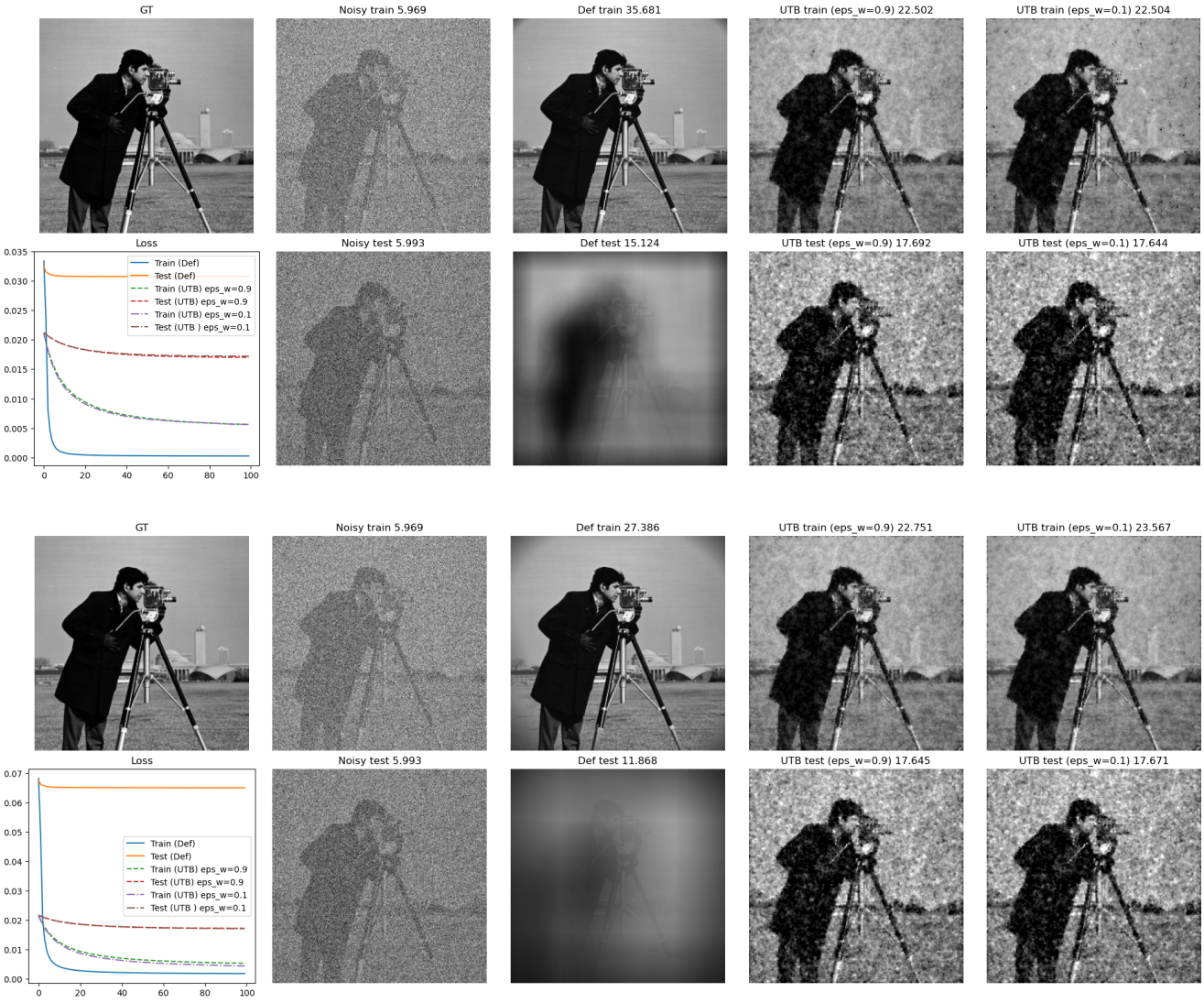

Figure 20. Results of learnt deformable and our metric UTB convolutions with $\sigma_n = 0.5$ and $k = 5, 11, 31$ from top to bottom.