**Noname manuscript No.**
(will be inserted by the editor)

# Learning to Generate Wasserstein Barycenters

**Julien Lacombe · Julie Digne · Nicolas Courty · Nicolas Bonneel**

Received: date / Accepted: date

**Abstract** Optimal transport is a notoriously difficult problem to solve numerically, with current approaches often remaining intractable for very large scale applications such as those encountered in machine learning. Wasserstein barycenters – the problem of finding measures in-between given input measures in the optimal transport sense – is even more computationally demanding as it requires to solve an optimization problem involving optimal transport distances. By training a deep convolutional neural network, we improve by a factor of 80 the computational speed of Wasserstein barycenters over the fastest state-of-the-art approach on the GPU, resulting in milliseconds computational times on $512 \times 512$ regular grids. We show that our network, trained on Wasserstein barycenters of pairs of measures, generalizes well to the problem of finding Wasserstein barycenters of more than two measures. We demonstrate the efficiency of our approach for computing barycenters of sketches and transferring colors between multiple images.

Julien Lacombe
INSA Lyon, Univ. Lyon
Lyon, France
E-mail: jlacombe@protonmail.com

Julie Digne
CNRS, Univ. Lyon
Lyon, France

Nicolas Courty
CNRS, IRISA, Univ. Bretagne Sud
Vannes, France

Nicolas Bonneel
CNRS, Univ. Lyon
Lyon, France

**Keywords** Wasserstein barycenter · Optimal Transport · Convolutional Neural Network · Color Transfer

## 1 Introduction

Optimal transport is becoming widespread in machine learning, but also in computer graphics, vision and many other disciplines. Its framework allows for comparing probability distributions, shapes or images, as well as producing interpolations of these data. As a result, it has been used in the context of machine learning as a loss (Frogner et al., 2015; Arjovsky et al., 2017), for building a manifold for dictionary learning (Schmitz et al., 2018), clustering (Mi et al., 2018) and metric learning applications (Heitz et al., 2019), as a way to sample an embedding (Liutkus et al., 2019) and transfer learning (Courty et al., 2014), and many other applications (see Sec. 2.3). However, despite recent progress in computational optimal transport, in many cases these applications have remained limited to small datasets due to the substantial computational cost of optimal transport, in terms of speed, and also memory. Among all Optimal Transport concepts, Wasserstein Barycenters are particularly interesting for data analysis. Lacombe et al. (2018a) use it for persistence diagram analysis. Feydy et al. (2019a) used Wasserstein barycenters as a probabilistic atlas of a set of density maps. It can even help as a greedy approximation for solving PDEs (Ehrlacher et al., 2020). The goal of our paper is to provide an efficient way to compute such barycenters.

We tackle the problem of efficiently computing Wasserstein barycenters of measures discretized on regular grids, a setting common to several of these machine learning applications. Wasserstein barycenters are interpolations of two or more probability distributions under

optimal transport distances. As such, a common way to obtain them is to perform a minimization of a functional involving optimal transport distances or transport plans, which is thus a very costly process. Instead, we directly predict Wasserstein barycenters by training a Deep Convolutional Neural Network (DCNN) specific to this task.

An important challenge behind our work is to build an architecture that can handle a variable number of input measures with associated weights without needing to retrain a specific network. To achieve that, we specify and adapt an architecture designed for and trained with two input measures, and show that we can use this modified network, without retraining, to compute barycenters of more than two measures. Directly predicting Wasserstein barycenters avoids the need to compute a Wasserstein embedding (Courty et al., 2017), and our experiments suggest that this results in better Wasserstein barycenters approximations. Our implementation is publicly available[1].

*Contributions* This paper introduces a method to compute Wasserstein barycenters in milliseconds. It shows that this can be done by learning Wasserstein barycenters of only two measures on a dataset of random shapes using a DCNN, and by adapting this DCNN to handle multiple input measures without retraining. This proposed approach is 80x faster than the fastest state-of-the-art GPU library, and performs better than Wasserstein embeddings.

## 2 Related Work

### 2.1 Wasserstein distances and approximations

Optimal transport seeks the best way to warp a given probability measure $\mu_0$ to form another given probability measure $\mu_1$ by minimizing the total cost of moving individual "particles of earth". We restrict our description to discrete distributions. In this setting, finding the optimal transport between two probability measures is often achieved by solving a large linear program (Kantorovich, 1942) – more details on this theory and numerical tools can be found in the book of Peyré et al. (2019). This minimization results in the so-called *Wasserstein distance*, the mathematical distance defined by the total cost of reshaping $\mu_0$ to $\mu_1$. This distance can be used to compare probability distributions, in particular in a machine learning context. It also results in a *transport plan*, a matrix $P(x, y)$ representing the amount of mass of $\mu_0$ traveling from location $x$ in $\mu_0$ towards location $y$ in $\mu_1$.

However, the Wasserstein distance is notoriously difficult to compute – the corresponding linear program is huge, and dedicated solvers typically solve this problem in $\mathcal{O}(N^3 \log N)$, with $N$ the size of the input measures discretization. Recently, numerous approaches have attempted to approximate Wasserstein distances. One of the most efficient methods, the so-called Sinkhorn algorithm introduces an entropic regularization, allowing to compute such distances by iteratively performing fast matrix-vector multiplications (Cuturi, 2013) Adding an entropic regularization simplifies drastically the structure of the transport problem: the solution can be computed as the fixed point of Sinkhorn iterations, which simply alternate Bregman projections over the set of marginal constraints Benamou et al. (2015). Another possible acceleration is to use convolutions in the case of regular domains (Solomon et al., 2015). However, this comes at the expense of smoothing the transport plan and removing guarantees regarding this mathematical distance (in particular, the regularized cost $W_\epsilon(\mu_0, \mu_0) \neq 0$). These issues are partly addressed by *Sinkhorn divergences* (Feydy et al., 2018; Genevay et al., 2017). This approach symmetrizes the entropy-regularized optimal transport distance, adding guarantees on this divergence (now, the cost $S_\epsilon(\mu_0, \mu_0) = 0$ by construction, although triangular inequality still does not hold) but also effectively reduces blur, while maintaining a relatively fast numerical algorithm. They show that this divergence interpolates between optimal transport distances and Maximum Mean Discrepancies. Sinkhorn divergences are implemented in the *GeomLoss* library (Feydy, 2019), relying on a specific computational scheme on the GPU (Feydy et al., 2019b, 2018; Schmitzer, 2019) and constitutes the state-of-the-art in term of speed and approximation of optimal transport-like distances. In general, the amount of entropy added to the problem constitutes a trade-off between computational speed, robustness and blur, as lower entropy induces smaller blur but reduced convergence speed and increased numerical errors (unless dealt with at additional computational cost, such as log-domain stabilization techniques (Schmitzer, 2019)).

### 2.2 Wasserstein barycenters

The Wasserstein barycenter of a set of probability measures corresponds to the Fréchet mean of these measures under the Wasserstein distance (i.e., a weighted mean under the Wasserstein metric). Wasserstein barycenters allow to interpolate between two or more probability measures by warping these measures (contrarily

[1] https://github.com/jlacombe/learning-to-generate-wasserstein-barycenters

to Euclidean barycenters that blends them). These interpolations can be computed using different methods, often considering a regularization term, using subgradients (Cuturi and Doucet, 2014), iterative Bregman projections (Benamou et al., 2015), or convolutions (Solomon et al., 2015; Janati et al., 2020). Similarly to Wasserstein distances, Wasserstein barycenters are very expensive to compute. An entropy-regularized approach based on Sinkhorn-like iterations also allows to efficiently compute blurred Wasserstein barycenters. Reducing blur via Sinkhorn divergence is also doable. This can be done by iteratively minimizing a functional via automatic differentiation tools – a direction taken by the GeomLoss library (Feydy, 2019) – or by using iterative projections with a convolutional approach similar to Sinkhorn iterations (Janati et al., 2020), though our experiments suggests the latter approach is significantly slower at equivalent quality (see Sec. 4.4). In our approach, we rely on Sinkhorn divergence-based barycenters to feed training data to a Deep Convolutional Neural Network, and thus aim at speeding up the generation of approximate Wasserstein barycenters. Other fast transport-based barycenters include that of sliced and Radon Wasserstein barycenters, obtained via Wasserstein barycenters on 1-d projections (Rabin et al., 2011b; Bonneel et al., 2015), which we compare to.

A recent trend seeks linearizations or Euclidean embeddings of optimal transport problems. Notably, Nader and Guennebaud (2018) approximate Wasserstein barycenters by first solving an optimal transport map between a uniform measure towards $n$ input measures, and then linearly combining Monge maps. This allows for efficient computations – typically of the order of half a second for 512x512 images. A similar approach is taken within the documentation of the GeomLoss library (Feydy, 2019)[2], where a single step of a gradient descent initialized with a uniform distribution is used, which effectively corresponds to such linearization. We use this technique in our work to train our network. Wang et al. (2013), Moosmüller and Cloninger (2020) and Mérigot et al. (2020) use a similar linearization, possibly using a non-uniform reference measure, with theoretical guarantees on the distorsion introduced by the embedding. Instead of explicitly building an embedding via Monge maps, such an embedding can be learned. Courty et al. (2017) propose a siamese neural network architecture to learn an embedding in which the Euclidean distance approximates the Wasserstein distance. Wasserstein barycenters can then be approximated by interpolating within the Euclidean embedding, without requiring explicit computations of transport plans. They show accurate barycenters on a number of datasets of low resolution ($28 \times 28$). However, in general, it is unclear whether Wasserstein metrics embed into Euclidean spaces. Negative results were shown for 3d optimal transport onto a Euclidean space (Andoni et al., 2016). Interestingly, in the reversed direction, Wasserstein spaces have been used to embed other metrics (Frogner et al., 2019).

Fan et al. (2020) propose a model based on input convex neural networks (ICNN) developed by Amos et al. (2017) to approximate Wasserstein barycenters of continuous input measures. Also in the continuous setting, other neural networks based methods have also been proposed to compute optimal transport barycenters (Li et al., 2020; Korotin et al., 2021). While these approaches make it possible to compute the Wasserstein barycenter when only samples from the input distributions are available, they require several minutes to obtain a barycenter given samples from two 2D inputs and are therefore not competitive in terms of runtime in our setting.

### 2.3 Applications to machine learning

For its ability to compare probability measures, optimal transport has met an important success in machine learning. This is particularly the case of Wasserstein GANs (Arjovsky et al., 2017) that compute a very efficient approximation of Wasserstein distances as a loss for generative adversarial models. The optimal transport loss has also been used in the context of dictionary learning (Rolet et al., 2016). Other fast approximations have allowed to perform domain adaptation for transfer learning of a classifier, by advecting samples via a computed transport plan (Courty et al., 2014). Among these approximations, Sliced optimal transport has been used to sample an embedding learned by an auto-encoder, by computing a flow between uniformly random samples and the image of encoded inputs (Liutkus et al., 2019).

Regarding the Wasserstein barycenters we are interested in, they have been used for the task of learning a dictionary out of a set of probability measures (Schmitz et al., 2018), for combining subset posteriors (Srivastava et al., 2015), for computing Wasserstein barycentric coordinates of probability measures (Bonneel et al., 2016) or for metric learning (Heitz et al., 2019). These have been performed by automatic-differentiation of Wasserstein barycenters obtained through Sinkhorn iterations and non-linear optimization, and have thus been limited to small datasets, both due to speed and memory limitations. An adaptation of k-means clustering for optimal transport was proposed by Mi et al. (2018) and (Domazakis et al., 2020). Backhoff-Veraguas et al. replaces

[2] See `https://www.kernel-operations.io/geomloss/_auto_examples/optimal_transport/plot_wasserstein_barycenters_2D.html`

maximum a posteriori (MAP) estimation or Bayesian model average, by computing Wasserstein barycenters of posterior distributions (Backhoff-Veraguas et al., 2018) using a stochastic gradient descent scheme. In the context of reinforcement learning, Wasserstein barycenters are used by Metelli et al. (2019) as a way to regularize the update rule and offer robustness to uncertainty. PCA in the Wasserstein space require the ability to compute Wasserstein barycenters ; they have been studied by Bigot et al. (2017) but could only be computed in 1-d where theory is simpler. They can be extended to higher dimensions, by relaxing the definition of geodesics (Seguy and Cuturi, 2015), with application to 2D images. In the work of Dognin et al. (2019), Wasserstein barycenters are used for model ensembling , i.e., averaging the predictions of several models to build a more robust model. Another application of Wasserstein barycenters concerns the interpolation of persistence diagrams in the context of topological data analysis (Lacombe et al., 2018b).

In this work, we do not focus on a single application but instead provide the tools to efficiently approximate Wasserstein barycenters on 2-d regular grids.

## 3 Learning Wasserstein barycenters

This section describes our neural network and our proposed solution to train it in a scalable way.

### 3.1 Proposed Model

Our model aims at obtaining approximations of Wasserstein barycenters from $n \geq 2$ probability measures $\{\mu_i\}_{i=1..n}$ discretized on $512 \times 512$ regular grids, and their corresponding barycentric weights $\{\lambda_i\}_{i=1..n}$. Based on the observation that the Sinkhorn algorithm is mainly made of successive convolutions, we propose to directly predict a Wasserstein barycenter through an end-to-end neural network approach, using a Deep Convolutional Neural Network (DCNN) architecture.

We propose a network consisting of $n$ contractive paths $\{\varphi_i\}_{i=1..n}$ and one expansive path $\psi$ (see Fig. 1). Importantly enough, $n$ is not fixed and can vary at test time. In fact, the contractive paths are $n$ duplicates of the same path with the same architecture and sharing the same weights. The $n$ contractive paths are made of successive blocks, each block consisting of two convolutional layers followed by a ReLU activation. We further add average pooling layers between each block in order to decrease the dimensionality. The expansive path is symetrically constructed, each block also being made of 2 convolutional layers with ReLU activations. To better invert average poolings, we use upsampling layers with nearest-neighbor interpolation. Finally, to recover an output probability distribution, we use a softmax activation at the end of the expansive path. All the 2D convolutions of our model use $3 \times 3$ kernels with a stride and a padding equal to 1. The architecture might look similar to the U-Net architecture introduced by Ronneberger et al. (2015), because of the nature of the contractive and expansive paths. However the similarities end here, since our architecture uses a variable number of contractive paths to handle multiple inputs. The connections we use from the contractive paths to the expansive path also highly differ: first, we take all the feature maps from each contractive path and not only a part of it as it is done in U-Net, and, second, we compute a weighted sum of all these activations using barycentric weights which results in a weighted feature map which is then symmetrically concatenated to the corresponding activations in the expansive path. Our network is deeper than U-Net and we do not use the same succession of layers nor the same downsampling and upsampling methods which are respectively max-pooling and up-convolutions in the case of U-Net. We also use Instance Normalization (Ulyanov et al., 2016) which has empirically shown better results than Batch Normalization for our model. These normalization layers are placed before each ReLU activation.

The connections going from the contractive paths to the expansive path are defined as follows: after each block in a contractive path $\varphi_i$ at depth level $j$, we take the resulting activations $\{F_{ij}\}_{i=1..n}$, compute their linear combination $F'_j = \sum_{i \in n} \lambda_i F_{ij}$, and concatenate it symmetrically to the corresponding activations in the expansive path (see figure 1). In our architecture we use 6 depth levels, chosen as a speed/quality trade-off (see Fig. 14 in appendix for additional experiments).

### 3.2 Training

Our solution allows to generalize a network trained for computing the barycenter of two measures to an arbitrary number of input measures while remaining fast to train.

*Variable number of inputs.* We expect our network to produce accurate results without constructing an explicit embedding whose existence remains uncertain (Andoni et al., 2008). However, a Euclidean embedding trivially generalizes to an arbitrary number of input measures. A key insight to our work is that, since contractive paths weights are shared, our network can be trained using only two contractive paths for the task of predicting Wasserstein barycenters of two probability measures.

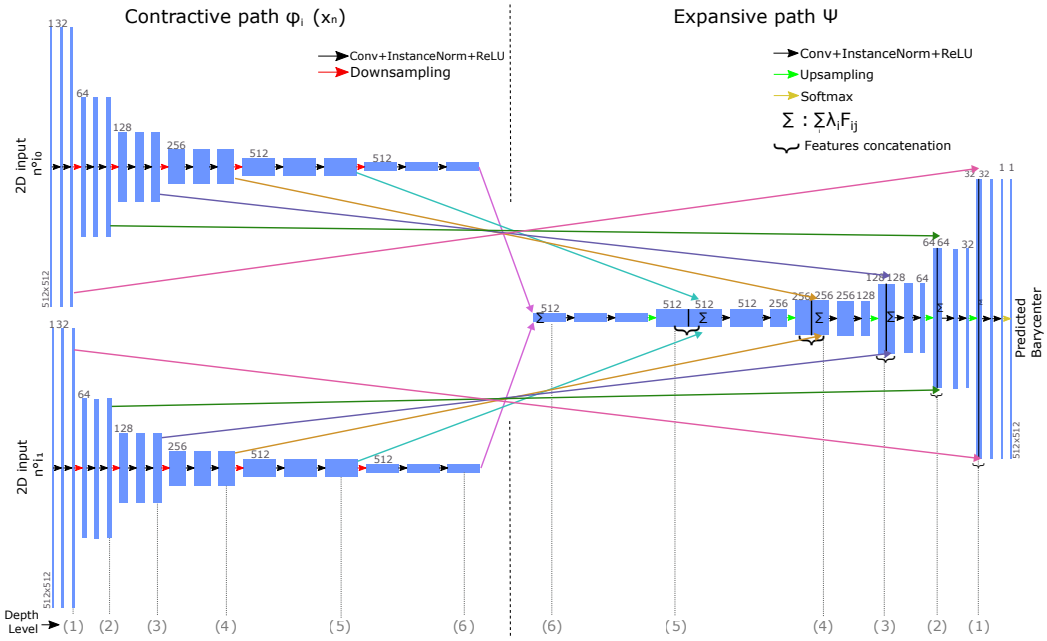



**Fig. 1** Our model is divided into $n$ contractive paths $\varphi_i$, sharing the same architecture and weights, and one expansive path $\psi$. Blue rectangles represent feature maps and arrows denote the different operations we use (see legend). We train our network with $n = 2$ as illustrated here, but by duplicating the contractive paths, we can adapt to the $n$ measures barycenter problem at test time, without needing to retrain the network.

Once trained, contractive paths can be duplicated to the desired number of input measures. In practice, we found this procedure to yield accurate barycenters (see Sec. 4).

*Loss function.* Training the network requires comparing the predicted Wasserstein barycenter to a groundtruth Wasserstein barycenter. Ideally, such comparison should be performed via an optimal transport cost – those are ideal to compare probability distributions. However, computing optimal transport costs on large training datasets would be intractable. Instead, we resort to a Kullback-Leibler divergence between the output distribution and the desired barycenter. KL divergence enforces that mass should be present wherever the groundtruth barycenter is nonzero (otherwise it causes an extreme cost), but there can be mass at places where the groundtruth barycenter is zero. It thus favors some one-way consistency of the mass support, which can be useful for line drawings. We also experimented with a sliced Wasserstein loss, and it yielded poor numerical results, with important visual artifacts.

*Optimizer.* To optimize the model parameters, we use a stochastic gradient descent with warm restarts (SGDR) (Loshchilov and Hutter, 2016). The exact learning rate schedule we used for our models is shown in appendix A, Fig. 12.

*Training data.* We strive to train our network with datasets that would cover a wide range of input sketches. To achieve this, we built a dataset made of $100k$ pairs of $512 \times 512$ random shape contours with random barycentric weights and their corresponding 2D Wasserstein barycenter. This dataset is itself divided into 3 distinct sets: a training set ($80k$ shapes), a validation set ($10k$ shapes) and a test set ($10k$ shapes). Thereafter we call this dataset *ContoursDS*. The 2D shapes are generated in a Constructive Solid Geometry fashion: we randomly assemble primitives shapes using logical operators and detect contours in post-processing. A primitive corresponds to a filled ellipse, triangle, rectangle or a line. We assemble these primitives together by using the classical boolean operators OR, AND, XOR, NOT. To generate a shape, we initialize it with a random primitive. Then we combine it with another random primitive using a randomly chosen operator, and repeat this operation $d$ times ($0 \leq d \leq 50$) where $d$ follows the probability distribution $d \sim \frac{1}{3}(\mathcal{U}(0, 50) + \mathcal{N}(0, 2.5) + \mathcal{N}(50, 2.5))$ which promotes simple ($d$ close to 0) and complex ($d$ close to 50) shapes. Finally, we apply a Sobel filter to create contours. We thus create $10k$ random 2D shapes from which we build $100k$ Wasserstein barycenters.

We then use the GeomLoss library (Feydy, 2019) to build good approximations of Wasserstein barycenters in a reasonable time, with random pairs of inputs sampled from the set of generated shape contours. Given

two 2D input distributions $\mu_1$ and $\mu_2$ with their corresponding barycentric weights $\lambda_1$ and $\lambda_2 = 1 - \lambda_1$, their barycenter $b^*$ can be found by minimizing: $b^* = \arg\min_b \lambda_1 S_\epsilon(b, \mu_1) + \lambda_2 S_\epsilon(b, \mu_2)$ where $S_\epsilon$ corresponds to the Sinkhorn divergence with quadratic ground metric, and $\epsilon$ the regularization parameter (we use $\epsilon = 1e{-}4$). We use a Lagrangian gradient descent scheme that first samples the distributions as $b = \sum_{j=1}^{N} b_j \delta_{x_j}$ and then performs a gradient descent using $x_j^{(k+1)} = x_j^{(k)} + \lambda_1 v_j^{\mu_1} + \lambda_2 v_j^{\mu_2}$ where $v_j^{\mu_i}$ is the displacement vector. This vector is computed as the gradient of the Sinkhorn divergence with squared $\ell 2$ as the ground cost function between points: $v_j^{\mu_i} = -\frac{1}{b_j} \nabla_{x_j} S_\epsilon(b, \mu_i)$. These successive updates can be computationally expensive when inputs are large. GeomLoss operates in the log domain for numerical stability, which prevents efficient vectorized separable convolution. However, it implements many other optimizations (GPU, multiscale, kernel truncation) which makes it one of the fastest library available. To speed up computations, we use a linearized approach that performs a single descent step, starting from a uniform distribution. In practice, this allows to precompute one optimal transport map between a uniform distribution and each of the input measures in the database, and obtain approximate Wasserstein barycenters by using a weighted average of these transport maps. Since the training dataset needs only to be generated once, it might be tempting to use more than one iteration. However this would lead to an unreasonable computation time increase. If we want to generate N 2-way barycenters of M shapes, our approach requires $M\times$gradient step$+N\times$combination (combinations are trivial in practice), while, if we use $K$ gradient steps, it would require $N \times K \times 2$. In our case $M = 10k$ and $N = 80k$, which means that going from 1 step to even two steps ($K = 2$) would multiply the computation time by 32. It has also been shown that the quality of barycenters is only marginally improved by adding more iterations (Feydy, 2020).

While it is quite obvious that our model trained with an application-specific dataset will produce the best results for this application, our model trained exclusively on *ContoursDS* achieves results that are close enough and which can be in practice sufficient for the applications we consider. Figure 9 demonstrates this in the context of color transfer. Interpolated color histograms are clearly best predicted by our model trained with the application-specific dataset; however the final color transfer results are very similar to the ones obtained using the histograms predicted by our model trained on ContourDS.

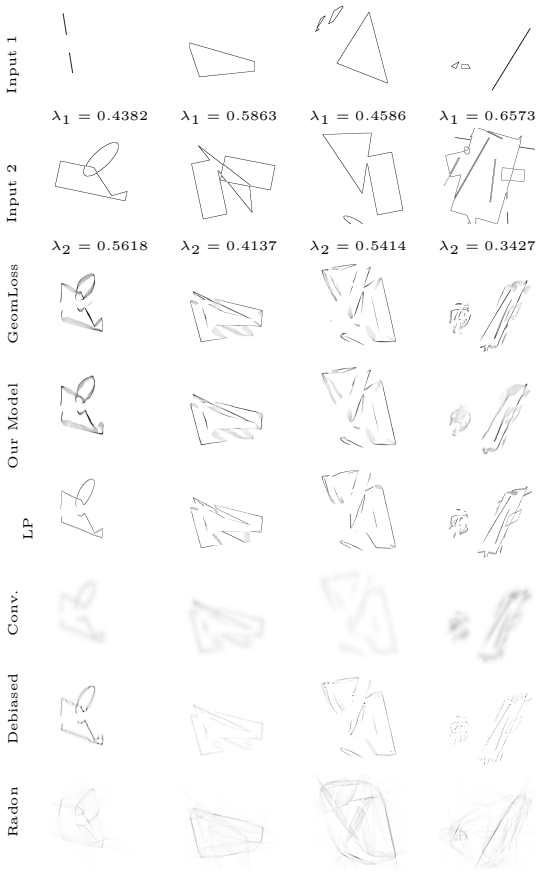



**Fig. 2** We illustrate 4 typical results, and comparisons to GeomLoss (Feydy, 2019), a linear program (LP) via a network simplex (Bonneel et al., 2011), regularized convolutional (Conv.) barycenters (Solomon et al., 2015) with a regularization parameter of $10^{-3}$, regularized debiased barycenters (Janati et al., 2020) also with a regularization parameter of $10^{-3}$ and Radon barycenters (Bonneel et al., 2015). The parameter used for regularized methods is the smallest regularization we found that did not introduce numerical instabilities.

## 4 Experimental Results

While our model is exclusively trained on our synthetic *ContoursDS* dataset, at test time we also consider three additional datasets : the *Quick, Draw!* dataset from Google (2020), the *Coil20* dataset (Nane et al., 1996) and *HistoDS*, a dataset of chrominance histograms. The *Quick, Draw!* dataset contains 50 million grayscale drawings divided in multiple classes and has been created by asking users to draw with a mouse a given object in a limited time. The *Coil20* is made of images of 20 objects rotating on a black background and contains 72 images per object for a total of 1440 images. We rasterized these two datasets to $512 \times 512$ images. Finally, *HistoDS* contains $100k$ $512 \times 512$ chrominance histograms extracted from 10350 Flickr[3] images of various content

[3] `https://www.flickr.com/`

and sizes obtained using the Flickr API. Our method was implemented in PyTorch (Paszke et al., 2019), and uses Matplotlib for visualization (Hunter, 2007).

### 4.1 Two-way interpolation results

In Fig. 2, we show a visual comparison between barycenters obtained with Geomloss and our method. Wassertein barycenters are taken from the test dataset and the corresponding predictions are shown. We also compare these results to classical approaches (linear program, regularized barycenters) and to another approximation method known as Radon barycenters (Bonneel et al., 2015).

To further visually assess that the barycenters we are approximating are close to the exact ones, we also present a comparison with the method of Claici et al. (2018) in Fig. 3. The number of points for Claici et al.'s method is low, which makes the comparison hard, but computing even this small number of points required 37 hours of computation, and adding more points would have been untractable in practice. Input distributions are taken from the *Quick, Draw!* dataset.

We also qualitatively validate the equivariance of our network under translation, rotation and scaling of the input measures in appendix B.

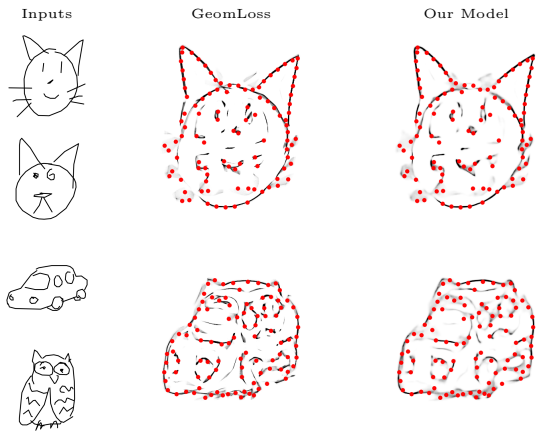



**Fig. 3** We superimpose the centroids ($\lambda_1 = \lambda_2 = 0.5$) found by the method of Claici et al. (2018) (in red) using 100 Dirac masses over the ones computed by GeomLoss and by our model, on images from the *Quick, Draw!* dataset. The solution of Claici et al. (2018) was found within 37 hours of computation

We compare our method with the Deep Wasserstein Embedding (DWE) model developed by Courty et al. (2017) on *Quick, Draw!* images. We propose two versions of DWE. The first version relies on the exact original architecture which can only process $28 \times 28$ images, retrained on a downsampled version of our shape contours dataset – see Fig. 5 for this comparison. In the second version, we adapt their network to process $512 \times 512$ inputs. The encoder and decoder of this second version have the same architecture as the contractive and expansive paths that we use in our model without our skip connections, but is used to compute the embedding rather than directly predicting barycenters – see Fig. 6. The main difference between our method and DWE, besides layer sizes, lies in our n-way generalization and the skip connections in our approach. This has radical consequences: our method does not compute an embedding of the shapes. DWE has thus theoretically a larger scope, since it can compute barycenters directly by exploiting the embedding. However the barycenter quality is much lower from both numerical (Fig. 4) and visual (Figs 5, 6) points of view.

In Fig.4, we show a numerical comparison of approximation errors between our model and DWE adapted to $512 \times 512$ images of our shape contours dataset, in terms of KL-divergence and L1 distance. Our results clearly show that our method is able to approximate more accurately the Wasserstein barycenter on $512 \times 512$ input measures.

Finally, we evaluate how our network generalizes on very different data using the *Coil20* dataset (Nane et al., 1996) that consists of grayscale photographs of objects on a black background (Fig. 7). Results are of limited quality, which indicates that for some applications retraining on a case-specific dataset might be better.

### 4.2 N-way barycenters

Even if our model has been trained using only barycenters computed from pairs of inputs, we can apply it to predict barycenters of more than two measures. This section illustrates N-way barycenters on 2-d sketch images and color distributions.

*Sketch interpolation.* We display interpolations between respectively three and five input measures in Fig.8, which surprisingly tends to show that our model can generalize what it learned on pairs of inputs, at least partially. Additional results on *Quick, Draw!* are also shown in appendix C, Fig. 15. A 100-way barycenter comparison can be found in appendix Sec. C, Fig. 16.

Numerically when the number of inputs is greater than 2, our model also achieve to find better approximations than the ones obtained with DWE, as shown in Fig. 4.

*Interpolating color distributions.* We also propose color transfer between images as another application of our

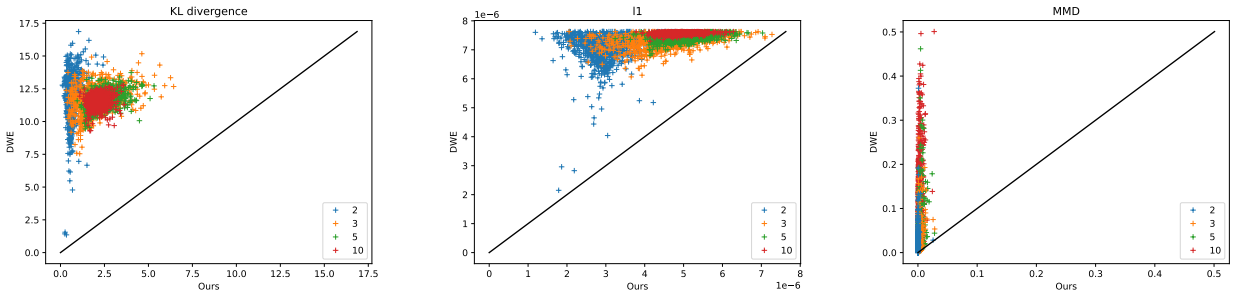



**Fig. 4** Approximation error of our model compared to the ones of DWE (version adapted to handle $512 \times 512$ images), respectively measured in terms of KL-Divergence, L1 distance, and Maximum Mean Discrepancy (MMD) (c), on images coming from our synthetic test dataset. Each point corresponds to a barycenter. The x-axis represents the error measured between the GeomLoss barycenter and the barycenter predicted by our model while the y-axis represents the one between the GeomLoss barycenter and the barycenter predicted by DWE. The color of a point associated to a barycenter represents its number of inputs (2,3,5 or 10).

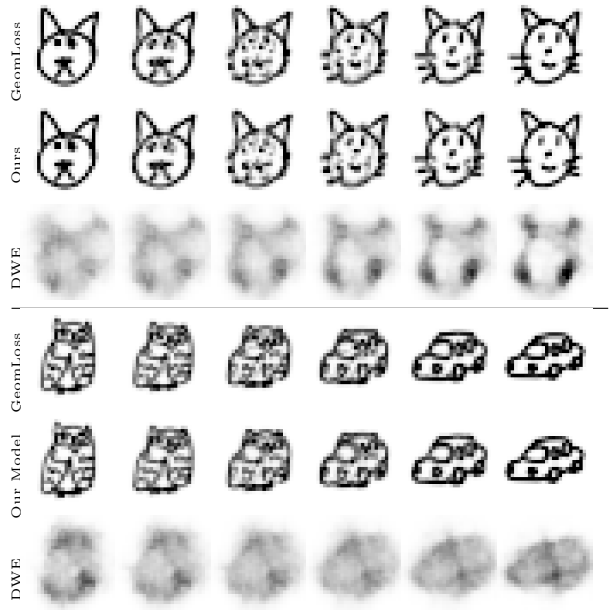



**Fig. 5** Interpolations between two $28 \times 28$ images from the *Quick, Draw!* dataset using Geomloss, our model and the original Deep Wasserstein Embedding (DWE) method from (Courty et al., 2017). Our model directly considers $512 \times 512$ inputs and its results are downsampled from $512 \times 512$ to $28 \times 28$


**Fig. 6** Interpolations between two $512 \times 512$ images from the *Quick, Draw!* dataset using Geomloss, our model and the Deep Wasserstein Embedding (DWE) method from (Courty et al., 2017) adapted to handle $512 \times 512$ images

method in the n-way case, as performed in the literature (Solomon et al., 2015; Bonneel et al., 2015). More particularly here we focus on a color grading application: given $n$ images, we are interested in the weighted interpolation of their color histograms. Then we alter the color histogram of a target image so that it matches the interpolated histogram in order to transfer colors. Based on recommendations by Reinhard and Pouli (2011), we consider images in the CIE-Lab space and we perform the transfer by modifying the luminance and the chrominance channels independently. While the transfer of luminance only requires 1D optimal transport plan, chrominance has 2 dimensions. In order to transfer it, we follow the procedure detailed in (Solomon et al., 2015): we first compute the $n$ 2D chrominance histograms $\{\mu_i\}_{i=1..n}$ and also $\nu$, that of the target image. These histograms are stored as $512 \times 512$ images. We then interpolate the $\{\mu_i\}_{i=1..n}$ using our model in order to obtain their barycenter $\hat{\mu}$ for given weights. Color transfer requires an explicit knowledge of the transport plan $\pi$ between $\nu$ and $\hat{\mu}$. In our method, $\pi$ is computed using the OT solver GeomLoss to retrieve the dual potentials $f$ and $g$ which are combined yielding $\pi = \exp \frac{1}{\epsilon}(f + g - C) \cdot \nu \otimes \hat{\mu}$ where $C$ corresponds to the cost matrix and with $\epsilon^2 = 5.10^{-2}$. Note that for this part we do not use Sinkhorn divergences and we

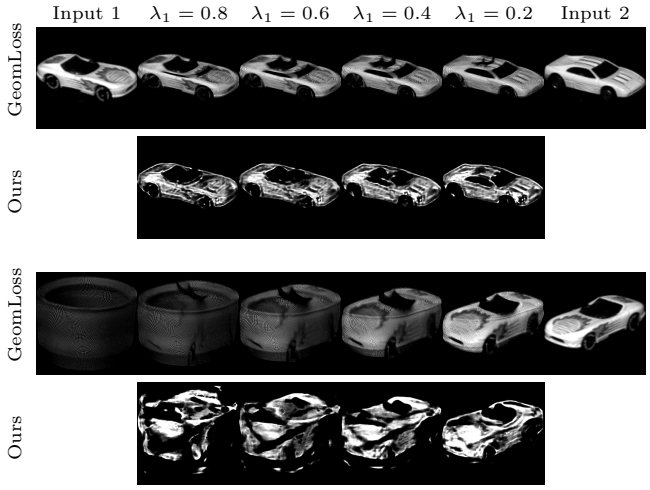



**Fig. 7** Interpolations between two $512 \times 512$ images from the *Coil20* dataset (Nane et al., 1996) using GeomLoss and our model trained with synthetic shape contours. For visualization purposes, white values represent high mass concentration, while dark values represent low mass concentration. The second interpolation in particular shows a failure case of our model due to our training images being significantly different from these images.

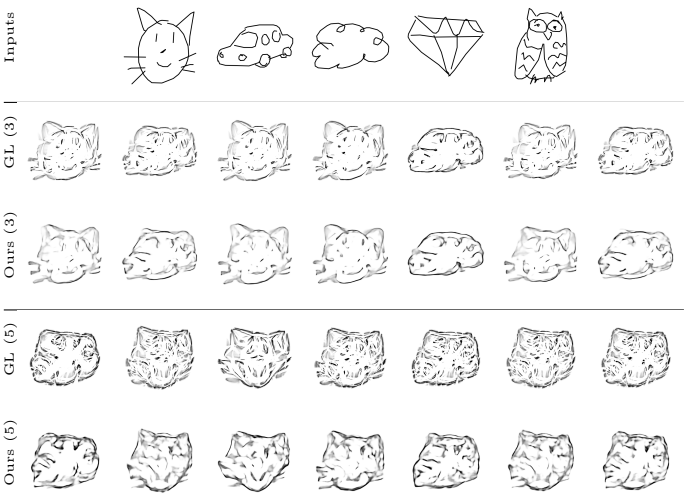



**Fig. 8** Wasserstein barycenters of three inputs (top rows) and five inputs (bottom rows) from *Quick, Draw!*, respectively computed with GeomLoss (GL) and with our model trained with only pairs from our synthetic training dataset. Barycentric weights are randomly chosen

instead consider the regularized OT problem in order to retrieve $f$ and $g$. This transport plan $\pi$ is used to retrieve the chrominance $T$ associated with the target image: $T(i) = \frac{1}{\nu(i)} \sum_{j \in M} \pi_{ij} j$ where $i, j \in M$ the set of all the possible discretized chrominance values.

After this color transfer step, similarly to (Bonneel et al., 2015), we apply a post-processing technique from (Rabin et al., 2011a) using iterative guided filtering in order to reduce visual artefacts caused by the color transfer. We repeat this color transfer for each image shown in each of the pentagons of figure 9 (last column). This figure presents a comparison of barycenter and color transfer results obtained with GeomLoss, our model trained on synthetic shape contours (*ContoursDS*) and our model directly trained with chrominance histograms extracted from images from the Flickr dataset (*HistoDS*). Even if predicted chrominance histograms are clearly better with *HistoDS*, the predictions made with *ContoursDS* are good enough to obtain a consistent and visually pleasing color transfer which is close to the one obtained using GeomLoss. Additional results are provided in appendix C, figure 18.

### 4.3 Embedding analysis

The connections between layers in the contractive paths and expansive path prevent our network to explicitly provide an embedding. However, one can still analyze the innermost layer and interpret it as an embedding by removing these connections once the network has been trained. We provide a visualization of a 2d UMap McInnes et al. (2018) embedding of the innermost layer activations of 200 QuickDraw Google (2020) shapes (20 drawings each from 10 randomly chosen classes: angel, bat, campfire, flower, knee, octopus, parachute, pear, pencil, power outlet) in Fig. 10. Even though important network connections have been removed, similar shapes still roughly cluster.

### 4.4 Speed

In order to assess computational times, we obtain average running time over 1000 barycenter computations – on average, our model on GPU predicts barycenters of two images in 0.0070 seconds. We compare the average speed of our model with GeomLoss, also on GPU, in two different settings. The first one considers the full $512 \times 512$ images – GeomLoss computes such barycenters in 1.58 seconds. The second setting takes advantage of the sparsity of our images and only uses the 2D coordinates of the points with non-zero mass – in this case, GeomLoss computes barycenters in 0.61 seconds. Our method provides nearly 87× speedup compared with this last approach. In comparison, an exact barycenter computation of two (sparse) measures using the C++ implementation of Bonneel et al. (2011) based on a network simplex ranges from 4 to 80 seconds for typical shape contours images that contains few thousands of pixels carrying mass. The time required to compute barycenters using the implementation of (Claici et al., 2018) depends on the number of iterations, in our set-

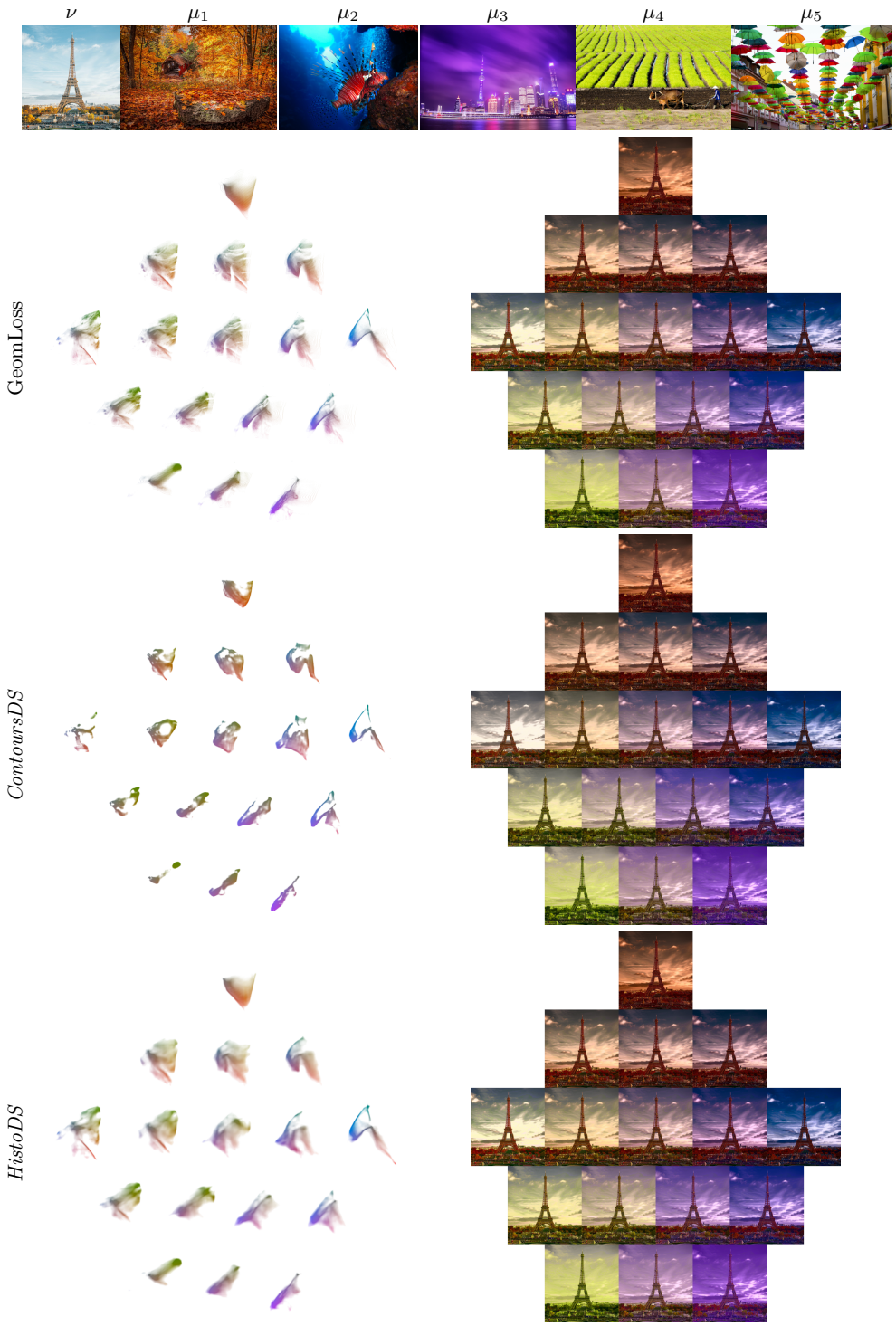



**Fig. 9** Color grading obtained by transferring the colors of $n = 5$ images onto a target image. Results are shown in pentagons (Left: interpolated chrominance histograms; Right: corresponding transfer results). The images corresponding to the target chrominance histogram $\nu$ and to the histograms $\mu_i$ - which are interpolated to obtain a barycenter - are shown in top row. Each $\mu_i$ corresponds to a vertex of the pentagon in a clockwise order beginning with $i = 1$ at the uppermost vertex. Each row presents the results for a different method, from top to bottom: GeomLoss, our model trained on synthetic shape contours (*ContoursDS*) and our model trained on chrominance histograms from Flickr images (*HistoDS*)

ting 100 iterations with the inputs shown in figure 3 require 37 hours while 50 iterations are achieved in 14

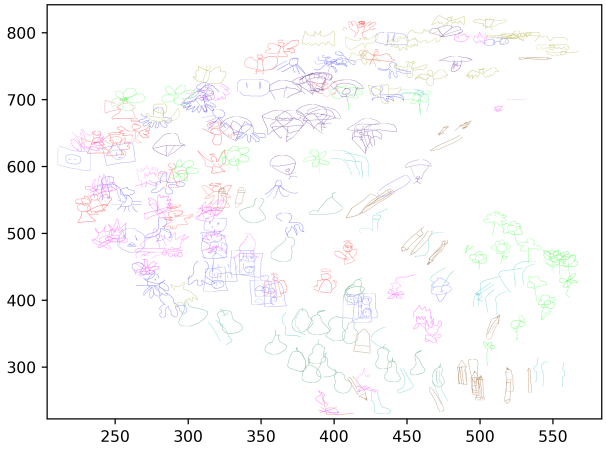



**Fig. 10** UMap representation of the activation of the innermost layer for 20 drawings of 10 classes of the QuickDraw dataset Google (2020).

hours. The convolutional method of Solomon et al. (2015) applied to our setting needs 4.7s on average with a standard deviation of 2.7s and the debiased approach (Janati et al., 2020) requires 41s on average (median of 28s) with a standard deviation of 40s. Note that the runtimes associated to these last two methods - relying on a PyTorch implementation on GPU - depend on the value of the regularization parameter and on the convergence criterion which we set to the values leading to the visually sharpest barycenters with no numerical instabilities. The convergence criterion we use corresponds to the same used in (Janati et al., 2020), namely the maximum relative change of the barycenters, set to $10^{-6}$ for debiased barycenters and to $10^{-8}$ for convolutional ones. Finally a $512 \times 512$ Radon barycenter (Bonneel et al., 2015) relying on a MatLab implementation requires 0.53s on average for 720 projection directions, but remains far from the expected barycenter. These runtimes were measured with a CPU Intel(R) Core(TM) i9-10900X CPU @ 3.70GHz and a NVIDIA GPU Geforce GTX 1660.

## 5 Discussion and conclusion

While our method produces good approximations of Wasserstein barycenters of $n$ inputs, some shapes are surprisingly difficult to handle. The barycenter of simple translated and scaled shapes such as lines or ellipses should theoretically also be lines or ellipses, but are failure cases for our model (Fig. 11), while more complex shapes are well handled (Fig. 8). In addition, we rely on a linearized barycenter to train our network (Nader and Guennebaud, 2018; Wang et al., 2013; Moosmüller and Cloninger, 2020; Mérigot et al., 2020), which incurs some error. This can be seen in appendix Sec. D,

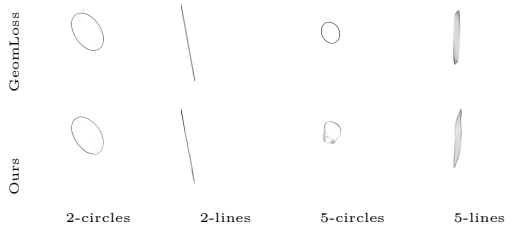



**Fig. 11** Wasserstein barycenters of sets of lines or ellipses should result in lines (resp. ellipses). Our prediction for two-way barycenters (here, with equal weights) of such shapes remains correct (left). However, the predicted barycenter is highly distorted for 5-way barycenters of simple shapes (right) although it remains plausible for more complex shapes (see Fig. 8)

Fig. 19. While using more iterations of gradient descent yields more accurate results and removes this linearity, it also prevents easy combination and makes the dataset generation intractable. Nevertheless, in many cases our DCNN is able to synthesize a barycenter from an arbitrary number of inputs. The main strength of our approach lies in its capacity to be trained from only 2-inputs barycenters examples and to generalize to any number of inputs. We showed that the results exceeded the ones obtained by explicit Wasserstein Embedding computation while having a very low computation time. Hence, while it is not aimed at computing *exact* barycenters, it sets a new milestone for *fast and approximate* computation. We hope our fast approach will accelerate the adoption of optimal transport in machine learning applications.

*Acknowledgements* This work was granted access to the HPC resources of IDRIS under the allocations 2020-AD011011538 and 2020-AD011012218 made by GENCI. We also thank the authors of all the images used in our color transfer figures.

*Funding* Partial financial support was received from the ANR ROOT (RegressiOn with Optimal Transport): ANR-16-CE23-0009 and ANR AI chair OTTOPIA under reference ANR-20-CHIA-0030

*Conflicts of interest / Competing interests.* The authors have no conflicts of interest to declare that are relevant to the content of this article.

*Code availability* Our implementation is publicly available at `https://github.com/jlacombe/learning-to-generate-wasserstein-barycenters`

## References


Amos B, Xu L, Kolter JZ (2017) Input convex neural networks. In: International Conference on Machine Learning, pp 146–155

Andoni A, Indyk P, Krauthgamer R (2008) Earth mover distance over high-dimensional spaces. In: SODA, vol 8, pp 343–352

Andoni A, Naor A, Neiman O (2016) Impossibility of sketching of the 3d transportation metric with quadratic cost. In: 43rd International Colloquium on Automata, Languages, and Programming (ICALP 2016), Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik

Arjovsky M, Chintala S, Bottou L (2017) Wasserstein gan. 1701.07875

Backhoff-Veraguas J, Fontbona J, Rios G, Tobar F (2018) Bayesian learning with wasserstein barycenters. arXiv preprint arXiv:180510833

Benamou JD, Carlier G, Cuturi M, Nenna L, Peyré G (2015) Iterative bregman projections for regularized transportation problems. SIAM Journal on Scientific Computing 37(2):A1111–A1138

Bigot J, Gouet R, Klein T, López A, et al. (2017) Geodesic pca in the wasserstein space by convex pca. In: Annales de l'Institut Henri Poincaré, Probabilités et Statistiques, Institut Henri Poincaré, vol 53, pp 1–26

Bonneel N, van de Panne M, Paris S, Heidrich W (2011) Displacement Interpolation Using Lagrangian Mass Transport. ACM Transactions on Graphics (SIGGRAPH ASIA 2011) 30(6)

Bonneel N, Rabin J, Peyré G, Pfister H (2015) Sliced and radon wasserstein barycenters of measures. Journal of Mathematical Imaging and Vision 51(1):22–45

Bonneel N, Peyré G, Cuturi M (2016) Wasserstein Barycentric Coordinates: Histogram Regression Using Optimal Transport. ACM Transactions on Graphics (SIGGRAPH 2016) 35(4)

Claici S, Chien E, Solomon J (2018) Stochastic wasserstein barycenters. arXiv preprint arXiv:180205757

Courty N, Flamary R, Tuia D (2014) Domain adaptation with regularized optimal transport. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, pp 274–289

Courty N, Flamary R, Ducoffe M (2017) Learning wasserstein embeddings. arXiv preprint arXiv:171007457

Cuturi M (2013) Sinkhorn distances: Lightspeed computation of optimal transport. In: Advances in neural information processing systems, pp 2292–2300

Cuturi M, Doucet A (2014) Fast computation of wasserstein barycenters. In: International conference on machine learning, PMLR, pp 685–693

Dognin P, Melnyk I, Mroueh Y, Ross J, Santos CD, Sercu T (2019) Wasserstein barycenter model ensembling. arXiv preprint arXiv:190204999

Domazakis G, Drivaliaris D, Koukoulas S, Papayiannis G, Tsekrekos A, Yannacopoulos A (2020) Clustering measure-valued data with wasserstein barycenters. arXiv preprint arXiv:191211801

Ehrlacher V, Lombardi D, Mula O, Vialard FX (2020) Nonlinear model reduction on metric spaces. Application to one-dimensional conservative PDEs in Wasserstein spaces. ESAIM: Mathematical Modelling and Numerical Analysis DOI 10.1051/m2an/2020013, URL https://hal.inria.fr/hal-02290431

Fan J, Taghvaei A, Chen Y (2020) Scalable computations of wasserstein barycenter via input convex neural networks. arXiv preprint arXiv:200704462

Feydy J (2019) Geometric loss functions between sampled measures, images and volumes. URL https://www.kernel-operations.io/geomloss/

Feydy J (2020) Geometric data analysis, beyond convolutions. Theses, Université Paris-Saclay, URL https://tel.archives-ouvertes.fr/tel-02945979

Feydy J, Séjourné T, Vialard FX, Amari SI, Trouvé A, Peyré G (2018) Interpolating between optimal transport and mmd using sinkhorn divergences. arXiv preprint arXiv:181008278

Feydy J, Roussillon P, Trouvé A, Gori P (2019a) Fast and Scalable Optimal Transport for Brain Tractograms. In: MICCAI 2019, Shenzhen, China, URL https://hal.telecom-paris.fr/hal-02264177

Feydy J, Roussillon P, Trouvé A, Gori P (2019b) Fast and scalable optimal transport for brain tractograms. In: International Conference on Medical Image Computing and Computer-Assisted Intervention, Springer, pp 636–644

Frogner C, Zhang C, Mobahi H, Araya-Polo M, Poggio T (2015) Learning with a wasserstein loss. arXiv preprint arXiv:150605439

Frogner C, Mirzazadeh F, Solomon J (2019) Learning embeddings into entropic wasserstein spaces. arXiv preprint arXiv:190503329

Genevay A, Peyré G, Cuturi M (2017) Learning generative models with sinkhorn divergences. arXiv preprint arXiv:170600292

Google I (2020) The quick, draw! dataset. URL https://github.com/googlecreativelab/quickdraw-dataset

Heitz M, Bonneel N, Coeurjolly D, Cuturi M, Peyré G (2019) Ground Metric Learning on Graphs. Tech. Rep. arXiv:1911.03117

Hunter JD (2007) Matplotlib: A 2d graphics environment. Computing in Science & Engineering 9(3):90–

95, DOI 10.1109/MCSE.2007.55

Janati H, Cuturi M, Gramfort A (2020) Debiased sinkhorn barycenters. In: International Conference on Machine Learning, PMLR, pp 4692–4701

Kantorovich L (1942) On the transfer of masses (in russian). In: Doklady Akademii Nauk, vol 37, pp 227–229

Korotin A, Li L, Solomon J, Burnaev E (2021) Continuous wasserstein-2 barycenter estimation without minimax optimization. arXiv preprint arXiv:210201752

Lacombe T, Cuturi M, OUDOT S (2018a) Large scale computation of means and clusters for persistence diagrams using optimal transport. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R (eds) Advances in Neural Information Processing Systems, Curran Associates, Inc., vol 31, URL https://proceedings.neurips.cc/paper/2018/file/b58f7d184743106a8a66028b7a28937c-Paper.pdf

Lacombe T, Cuturi M, Oudot S (2018b) Large scale computation of means and clusters for persistence diagrams using optimal transport. arXiv preprint arXiv:180508331

Li L, Genevay A, Yurochkin M, Solomon J (2020) Continuous regularized wasserstein barycenters. arXiv preprint arXiv:200812534

Liutkus A, Simsekli U, Majewski S, Durmus A, Stöter FR (2019) Sliced-wasserstein flows: Nonparametric generative modeling via optimal transport and diffusions. In: International Conference on Machine Learning, PMLR, pp 4104–4113

Loshchilov I, Hutter F (2016) Sgdr: Stochastic gradient descent with warm restarts. arXiv preprint arXiv:160803983

McInnes L, Healy J, Melville J (2018) Umap: Uniform manifold approximation and projection for dimension reduction. arXiv preprint arXiv:180203426

Mérigot Q, Delalande A, Chazal F (2020) Quantitative stability of optimal transport maps and linearization of the 2-wasserstein space. Proceedings of Machine Learning Research, vol 108, pp 3186–3196

Metelli AM, Likmeta A, Restelli M (2019) Propagating uncertainty in reinforcement learning via wasserstein barycenters. In: Advances in Neural Information Processing Systems, pp 4333–4345

Mi L, Zhang W, Gu X, Wang Y (2018) Variational Wasserstein clustering. In: Proceedings of the European Conference on Computer Vision (ECCV), pp 322–337

Moosmüller C, Cloninger A (2020) Linear optimal transport embedding: Provable fast wasserstein distance computation and classification for nonlinear problems. 2008.09165

Nader G, Guennebaud G (2018) Instant transport maps on 2d grids. ACM Trans Graph 37(6)

Nane S, Nayar S, Murase H (1996) Columbia object image library: Coil-20. Dept Comp Sci, Columbia University, New York, Tech Rep

Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, Chintala S (2019) Pytorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R (eds) Advances in Neural Information Processing Systems 32, Curran Associates, Inc., pp 8024–8035, URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Peyré G, Cuturi M, et al. (2019) Computational optimal transport. Foundations and Trends® in Machine Learning 11(5-6):355–607

Rabin J, Delon J, Gousseau Y (2011a) Removing artefacts from color and contrast modifications. IEEE Transactions on Image Processing 20(11):3073–3085

Rabin J, Peyré G, Delon J, Bernot M (2011b) Wasserstein barycenter and its application to texture mixing. In: International Conference on Scale Space and Variational Methods in Computer Vision, Springer, pp 435–446

Reinhard E, Pouli T (2011) Colour spaces for colour transfer. In: International Workshop on Computational Color Imaging, Springer, pp 1–15

Rolet A, Cuturi M, Peyré G (2016) Fast dictionary learning with a smoothed wasserstein loss. In: Artificial Intelligence and Statistics, pp 630–638

Ronneberger O, Fischer P, Brox T (2015) U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention, Springer, pp 234–241

Schmitz MA, Heitz M, Bonneel N, Mboula FMN, Coeurjolly D, Cuturi M, Peyré G, Starck JL (2018) Wasserstein dictionary learning: Optimal transport-based unsupervised non-linear dictionary learning. SIAM Journal on Imaging Sciences 11(1)

Schmitzer B (2019) Stabilized sparse scaling algorithms for entropy regularized transport problems. SIAM Journal on Scientific Computing 41(3):A1443–A1481

Seguy V, Cuturi M (2015) Principal geodesic analysis for probability measures under the optimal transport metric. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R (eds) Advances in Neural Information Processing Systems, Curran Associates, Inc., vol 28, URL https://proceedings.neurips.cc/paper/

2015/file/f26dab9bf6a137c3b6782e562794c2f2-Paper.pdf

Solomon J, De Goes F, Peyré G, Cuturi M, Butscher A, Nguyen A, Du T, Guibas L (2015) Convolutional wasserstein distances: Efficient optimal transportation on geometric domains. ACM Transactions on Graphics (TOG) 34(4):1–11

Srivastava S, Cevher V, Dinh Q, Dunson D (2015) Wasp: Scalable bayes via barycenters of subset posteriors. In: Artificial Intelligence and Statistics, PMLR, pp 912–920

Ulyanov D, Vedaldi A, Lempitsky V (2016) Instance normalization: The missing ingredient for fast stylization. arXiv preprint arXiv:160708022

Wang W, Slepčev D, Basu S, Ozolek JA, Rohde GK (2013) A linear optimal transportation framework for quantifying and visualizing variations in sets of images. International journal of computer vision 101(2):254–269

## A Learning Strategy

Instead of using a fixed learning rate or a decreasing learning rate, we choose a learning rate schedule with warm restart as proposed by Loshchilov and Hutter (2016). The learning schedule is shown in Figure 12: the learning rate decreased and is periodically restarted to its initial value, the period increasing as the number of epochs grows. This schedule was chosen after comparing with both Adam and SGD with stepwise schedules, and yielded better convergence in practice.

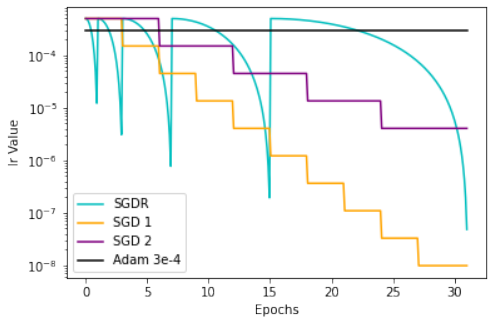



**Fig. 12** Learning rate schedule used to train our models (in blue), following the SGDR method described by Loshchilov and Hutter (2016). Our training runs for a total of 31 epochs. Compared to Adam with a learning rate of $3.10^{-4}$ or to SGD with stepwise schedules, SGDR has empirically shown a better convergence in our context. Note that while the gap between the final obtained KL divergence loss obtained using SGD with stepwise schedules (0.798 for SGD1 and 0.688 for SGD2) and the one obtained with SGDR (0.616) is significant, the one between Adam (0.622) and the SGDR schedule used is smaller, such that one may consider using Adam instead of SGDR

## B Test of equivariance

Wasserstein barycenters are equivariant under rotation, translation and scaling. This amounts to $Barycenter(\{T(\mu_i), \lambda_i\}_i) = T(Barycenter(\{\mu_i, \lambda_i\}_i))$ for $T$ a rotation, translation or scaling. We verify this behaviour qualitatively on the output of our network on two examples shown in Fig. 13.

## C Additional results

While the model presented in this paper uses 6 depth levels (following the convention of figure 1), we provide additional experiments showing the differences between our model with different number of depth levels in figure 14. Note that while the gap in performance is particularly important between the model with 4 depth levels (DL), 5DL and 6DL, there is much less difference between 6DL and 7DL.

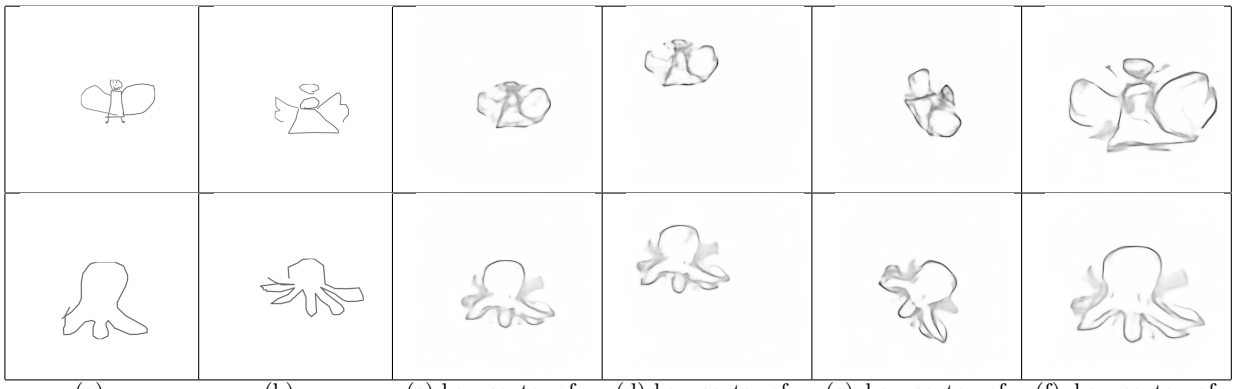
We provide additional experiments showing barycenters of 5 sketches on Figure 15. The weights evolve linearly inside the pentagon. As a stress test, we also show a barycenter of 100 cats with equal weights in Fig. 16 and compare it with a barycenter computed with GeomLoss. While both results recover more or less the global shape of the cat, details are clearly lost and our result looks much smoother.

In the debiasing approach of Janati et al. (2020), a single iteration of Bregman projections to minimize the functional in the variable $d$ is used in their paper. However, their available implementation uses 10 iterations. Fig. 17 shows the (minor) difference in quality between these two approaches. Our comparisons were performed against the original implementation.

Finally, we provide an additional color transfer experiment in figure 18 reproducing an experiment from Bonneel et al. (2015) with our model trained with *ContoursDS* and *HistoDS*.

## D Linearized barycenters

Fig. 19 shows the error introduced by using a linearized version of Wasserstein barycenters (Nader and Guennebaud, 2018; Wang et al., 2013; Moosmüller and Cloninger, 2020; Mérigot et al., 2020). Our predicted barycenters reflect this error.



(a)$\mu_0$ (b) $\mu_1$ (c) barycenter of $\{(\mu_0, \frac{1}{2}), (\mu_1, \frac{1}{2})\}$ (d) barycenter of translated measures (e) barycenter of rotated measures (f) barycenter of scaled measures

**Fig. 13** To qualitatively assess the equivariance of our network, we produce barycenters of measures $\mu_0$ (a) and $\mu_1$ (b) with equal weights as shown in (c), and barycenters of these measures translated by 120 (top) and 100 pixels (bottom) both horizontally and vertically (d), rotated by 60 (top) and 45 (bottom) degrees (e) and scaled by a factor of 2 (top) and 1.4 (bottom) (f). The barycenters of the transformed measures resemble the transformed barycenter of the input measures.

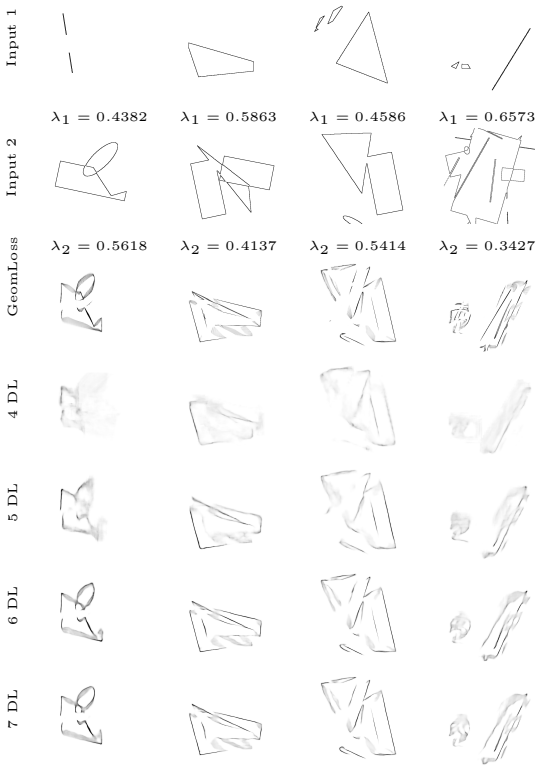



**Fig. 14** Comparisons between our model with different numbers of depth levels (DL), following the convention of figure 1. The corresponding final KL divergence losses obtained on the test dataset are : 1.5975 for 4DL, 1.0550 for 5DL, 0.6176 for 6DL (the model presented in this paper) and 0.5729 for 7DL. While the gap in performance is both visually and numerically important between 4DL and 5DL and 6DL, it is limited between 6DL and 7DL. The different associated training times for these models are : 38h50m for 4DL, 48h44m for 5DL, 53h28m for 6DL and 64h05m for 7DL. These runtimes have been measured when training the models with 2 NVIDIA GPU V100 SMX2 and a CPU Intel(R) Xeon(R) Gold 6226 CPU @ 2.70GHz (HPC resources of IDRIS)

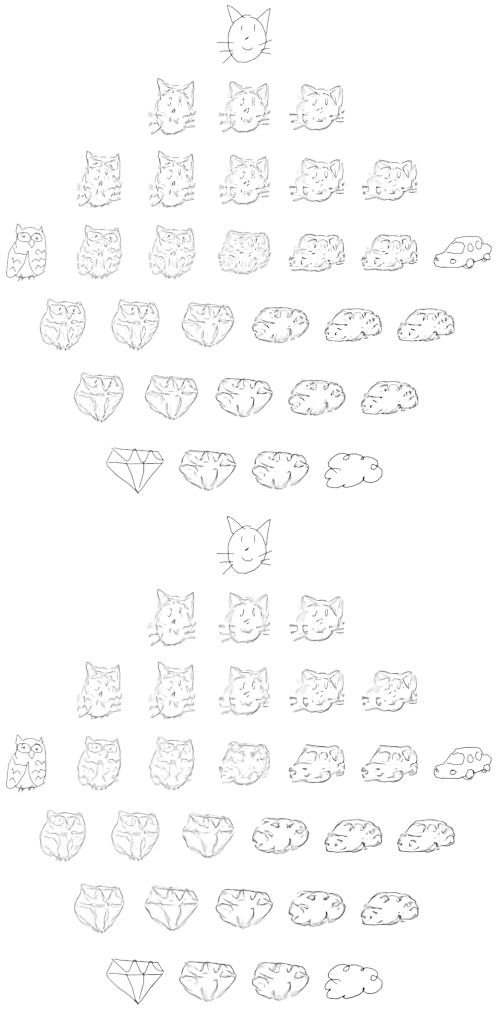
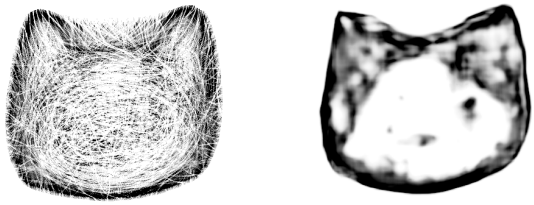


**Fig. 15** Interpolations between 5 inputs from *Quick, Draw!*, shown as pentagons. Upper pentagon corresponds to GeomLoss barycenters while the lower one shows predictions of our model trained on our synthetic dataset



GeomLoss Our DCNN

**Fig. 16** Stress test. We predict a barycenter of 100 cats of the *Quick, Draw!* dataset, with equal weights


**Fig. 17** Comparisons when using a single or ten Bregman projection steps to minimize the Sinkhorn divergence functional in the debiased barycenter approach of Janati et al. (2020). While using ten iterations makes it slower to converge, this leads to sharper results and is the default approach adopted in the authors' implementation: we thus used this value in the rest of the paper.

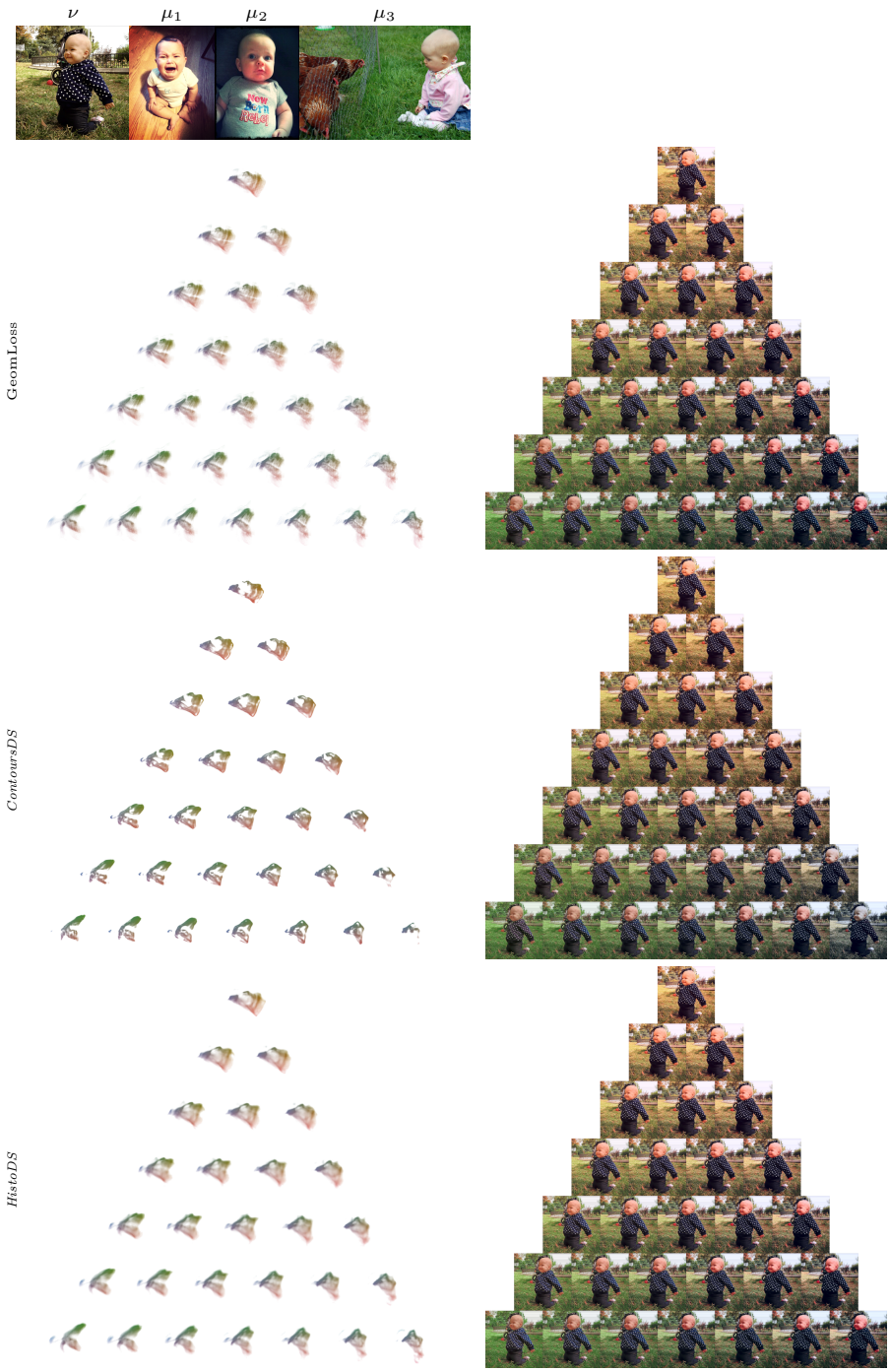



**Fig. 18** Color grading obtained by transferring the colors of $n = 3$ images onto a target image, aiming at reproducing with our method the results from (Bonneel et al., 2015), figure 12. Results are shown in triangles (Left: interpolated chrominance histograms; Right: corresponding transfer results). The images corresponding to the target chrominance histogram $\nu$ and to the histograms $\mu_i$ - which are interpolated to obtain a barycenter - are shown in top row. Each $\mu_i$ corresponds to a vertex of the triangle in a clockwise order beginning with $i = 1$ at the uppermost vertex. Each row presents the results for a different method, from top to bottom: GeomLoss, our model trained on synthetic shape contours (*ContoursDS*) and our model trained on chrominance histograms from Flickr images (*HistoDS*)

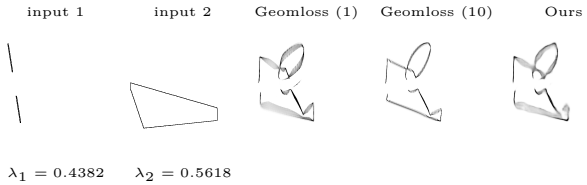



**Fig. 19** Wasserstein barycenter computed from a pair of inputs respectively using Geomloss with only one descent step, Geomloss with 10 descent steps and using our model trained on our synthetic training dataset