Technion - Israel Institute of Technology
GIP Lab

Project in Image Processing and Analysis
234329

Learning Unique Invariant Signatures of Non-Rigid Point Clouds

Authors:

Sari Hleihil & Idan Shenfield.

Supervisor:

Ido Imanuel.

Table of Contents

1 Abstract

We propose a metric learning framework for the construction of invariant signatures of non-rigid 3D point clouds under the isometry transformations group. We leverage the representational power of convolutional neural networks to compute these signatures and show that in comparison with classical methods, we achieve superior results that allow for higher classification accuracy using the invariant signature, and a lower pose dependency, with the additional advantage of much lower complexity, allowing for the calculation of invariant signatures for larger point clouds with orders of magnitude less time, this is achieved without the use of edge information that is commonly used for such applications.

2 Introduction

Non-rigid objects such as humanoids or any shape[1] with joints pose a difficulty for computational shape identification. Consider a rigid shape. For any scan of the same shape, we have 6 degrees of freedom, 3 for the rotation, and 3 for the translation, allowing for a relatively simple embedding space; in contrast, non-rigid shapes can bend their joints, and thus each joint can add 1 or more degrees of freedom in a simple articulated model, or even more if we consider elastic deformations of the surface around the joint as a result of it moving e.g., skin folding. Hence, shapes such as human beings with many joints can have a relatively large and complex embedding space due to the large pose[2] space, which leads to a more challenging identification task.

In practice, this means that it can be hard to computationally differentiate between the identity of a shape and its pose; a pose Invariant representation of shapes is required. We ask for this representation to be different for different shapes, but to be the same for the same shape under a change of pose and orientation.

Formally, it is common practice to model shapes as Riemannian manifolds, and the changes between them as a distance preserving transformation, which is referred to as an isometry. This is because only the joints are moved, but all other parts of the shape are not moved with respect to their neighborhood, hence if the metric was to be altered, it would only be in the joints' areas due to elastic deformations of the skin, muscle, and fat. We neglect effects of this deformation in our current model, but since we make use of real scans, our model is trained on data containing such deformations and thus theoretically it could learn to account for them.

Henceforth, let X be the set of all non-rigid objects[3] and $S \subseteq X^X$ be the set of all isometric transformations, our objective is to find a function ϕ such that:

$$(1)\ \forall x, y \in X:\ \phi(x) = \phi(y) \Leftrightarrow \exists \psi \in S, \psi(x) = y$$

This definition is not the standard definition of an invariant signature, as it also asks for the signature to be unique to each object. This is important if our goal is to learn a useful signature, since removing this condition means that trivial signatures are accepted as viable signatures e.g., signatures that are constant over all shapes, but such signatures contain no information.

Since objects are continuous, we obviously cannot deal with them directly using computers, and therefore we choose to represent them using point clouds sampled from the object. We show that even at lower resolutions and without using

[1] We shall use the word shape to describe an object regardless of its pose\orientation, e.g., a specific person is an object.

[2] We shall use the word pose to describe the way a shape appears (regardless of the object itself), such as the setting of its joints and its orientation.

[3] We shall use the word object to describe a shape in a specific pose, such that there is no ambiguity in this definition.

edge information our method achieves satisfactory results.

To find a function satisfying the above conditions, we make use of deep neural networks, a slightly altered PointNet [14] is used to encode the objects into normalized 1024-dimensinoal vectors that are pose invariant and unique to each shape.

To achieve this we train our network in a Siamese network setup, where different objects are fed into encoders with shared weights. The loss function motivates the network to learn pose invariant signatures by motivating close signatures for objects representing the same shape in different poses, and distant signatures for objects representing different shapes.

We compare our signatures in a shape classification task against MDS-type algorithms and an end-to-end classifier that shares its architecture with our algorithm. Interestingly, our algorithm beats the classifier trained with the same architecture, showing that our training scheme motivated learning meaningful signatures.

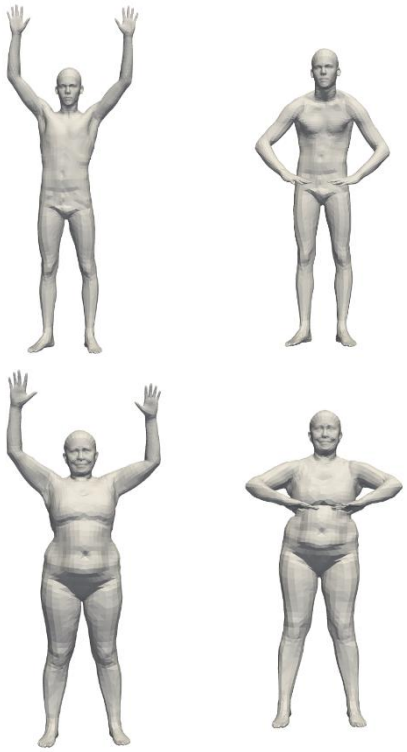


Figure: the upper two tringular meshes are of the same person (shape) in different poses, and the bottom two are two different poses of another shape.

An invariant signature extractor would produce the same signature for the upper two meshes, and a different signature the is identical for the bottom two.

3 Previous Work

Learning Invariant Representations of Planar Curves

Deep learning methods have been successfully used to learn invariant signatures of planar curves by [1] where they rely on the fundamental theorem of differential invariants [22], which states that every differential invariant of a planar curve with respect to some group of transformations is a function of the derivatives of a unique differential invariant (of a bounded order) with respect to an arclength parameterization.

This means that every differential invariant of a planar curve is a function of an infinitesimal neighborhood of each point, meaning that no global context is needed to calculate the differential invariants, and thus convolutional networks could be used.

To do this, [1] uses a convolutional network while assuming that the points are ordered correctly, meaning that the output corresponding to each point would be a function of the receptive field of the convolution which is a neighborhood of the point itself. The signature is trained using a Siamese network setup. By learning an invariant signature under the Euclidean transformation group, Gautam et al. show that this method successfully learns a function strongly correlated to the Gaussian curvature of planar curves which is the first differential invariant of a planar curve under the Euclidean transformation group.

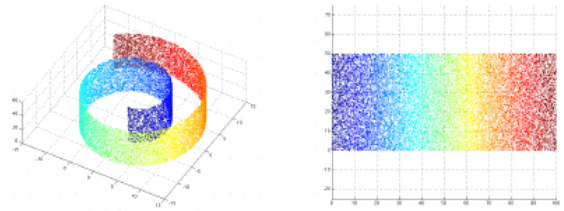
3.1 MDS

Multi-Dimensional-Scaling [2] is a classical algorithm, or rather a family of classical algorithms, that embed data of a high dimensional general metric space (not necessarily Euclidean) into lower dimension (usually) Euclidean space; the algorithm attempts to minimize the pairwise distance matrix of the object before and after the embedding. We note that these methods require such distances to be provide/computable, and thus using points-clouds is not trivial, and usually triangulated meshes are used.

Formally, let $\{x_i\}_{i=1}^n \subseteq X$ where (X, d_X) is a general metric space, define $d_{ij} = d_X(x_i, x_j)$, the classical MDS algorithm solves the following optimization problem:

$$\{y_i^*\}_{i=1}^n \coloneqq \underset{\{y_i\}_{i=1}^n}{\operatorname{argmin}} \left\{ \sum_{i=1}^n \left(d_{ij}^2 - \|y_i - y_j\|_2^2 \right) \right\}$$

Such problems could be solved in many ways, some are iterative and others are axiomatic & deterministic.



[3] *right: the popular 3D Swiss roll, lift: MDS embedding of the Swiss roll.*

Note: that one could see that the geodesic distances of points in the left object are represented by the Euclidean distance on the right.

As mentioned previously, objects are modeled as metric spaces. The MDS algorithm could applied to meshes describing such spaces to embed the 2-dimensional manifold into a Euclidean space of higher dimension. The metric of original metric space is defined by the geodesic distances between the points of the mesh which could be calculated using known methods such as Fast Marching [4] or the Heat Method [5], many variations of this have been suggested over the years such as in [6],[7] and [8]. Because we model pose transformations as isometries, the embeddings into Euclidean space for any two poses of the same object are also isometric, but an isometry in the Euclidean space is a Euclidean transformation. This means that the question of weather two shapes are isometric is equivalent to the question of weather their embeddings are a result of a Euclidean transformation of one another.

3.2 NMDS

Non-Metric MDS[9] is a variant of multi-dimensional scaling that attempts to conserve the order of the distances and not the distances themselves, meaning that if we were to replace each element of the distance matrices (before and after the embedding) by its rank, optimally the resulting matrix would be untouched by the NMDS embedding.

To do this, the algorithm attempts to optimize a term we call the stress, which is defined by the following equation:

$$Stress \coloneqq \sqrt{\frac{\sum (f(x) - d)^2}{\sum d^2}}$$

Where:

- x is the vector of proximities.
- d the point distances.
- f is a monotonic transformation of x, which could be found by using Isotonic regression [10].

3.3 Geodesic Distance Descriptors

Geodesic Distance Descriptors [11], is an algorithm that attempts to find an orthogonal set that minimizes the MSE of the best representation of the distance matrix as linear combinations of the elements of the set (noting that we do not call it a basis as it does not have to be of the dimension of the space, and thus reducing the dimensionality). Mathematical approximations

are used to make this process more efficient and less heavy in terms of computational resources; this leads to a decomposition of the approximation $\widetilde{D} = Q\Lambda Q^T$, where Λ is a diagonal matrix, and thus $X \coloneqq Q\sqrt{\Lambda}$ is well defined, and noting that $XX^T = \widetilde{D}$ one could say that X encodes the full information found in $\widetilde{D}$, and thus X could be our embedding where $X \in \mathbb{C}^{n\times c}$ where n is the number of points and c is the chosen dimension of the embedding space.

This means that X could be thought of as the set of pointwise descriptors for each point. Additionally, we note that since the entire process only takes into consideration the geodesic distances between points. One can prove that the resulting descriptors are permutation, rotation, and translation invariant.

Additionally, as discussed above, we model non-rigid transformations as isometric transformations, meaning that the geodesic distance matrix should not be affected, and thus resulting in a descriptor invariant to pose changes.

4 Training For Invariance

We observe that our objective (1) for the learned function ϕ could be thought of as two conditions:

1- Given a shape in some pose $x \in X, \forall \psi \in S\ \phi(x) = \phi\big(\psi(x)\big)$. (2)
2- Given two different shapes $x, y \in X$ (i.e., $\forall \psi \in S, \psi(x) \neq y$), $\phi(x) \neq \phi(y)$. (3)

Noting that if and only if both atoms are satisfied, equation (1) is satisfied.

Analyzing our new set of objectives, we can see that the first one asserts that pose invariance, while the second asserts that different objects produce different signatures, and thus excluding trivial invariant signatures.

One possible method to learn such a signature is adversarial training; an encoder network learns the signature, while two classifier heads are trained, where one learns to classify the shapes, and the other learns to classify the poses; the heads are trained in interleaving intervals with the encoder, such that the encoder's loss is defined in so that it minimizes the shape classifier's loss and maximizes the pose classifier's loss.

The problem with the above approach is that it requires labels for the poses, while for data acquired from animations, for example, there are thousands of poses, which might be unique, and therefore the classification task would be hard and there might not be enough examples of each pose to generalize (the curse of dimensionality). Animations are interesting since 3D scans of the same shape in different poses are not common, and a great source for such scans is 3D animations that include hundreds of different frames of the same shape in different poses.

Our solution to this problem is to use a metric learning scheme, where we only need to know whether the two given point clouds represent the same shape, and if so, we need to know whether they are the same pose or not. This means that we have 3 options: same shape different pose, same shape and pose & different shapes. The reasoning behind us not caring about the pose of different shapes could be seen from our objective, if two shapes are different, we want the signatures to be different regardless of the pose.

Additionally, we note that objects of the same shape and pose are identical, meaning that trivially the network produces the same output for both, and therefore we do not need to keep track of them. This leaves us with only two options: the same shape in a different pose and different shapes, which we term positive and negative examples respectively.

After establishing a dataset with the above annotations, we sample triplets such that the first object is called the original sample, along with negative and positive examples of that object. we build our loss function such that it maximizes some metric between the signatures of the negative examples and the original samples' and minimizes it between the positive examples' signature and the original samples'.

4.1 The Signature

We choose our signature to be a vector of some come dimension n, but We limit the signature vector to vectors with a unit norm meaning that $\phi \in \mathcal{S}^{n^X}$. This gave us more stable results numerically, reduces the degrees of freedom by only 1, and encourages a linear separation. Another important reason for this choice is that it makes the network encode the similarity of different vectors into their direction rather than their range, which is more precise and allows for the use of the cosine similarity measure:

$$\cos\big(\angle(x,y)\big) \coloneqq \frac{\langle x,y\rangle_{std}}{\|x\|_2\|y\|_2} = \langle x,y\rangle_{std}$$

Where the last equality is a direct result of normalizing the vectors, this means that this measure could be easily calculated.

4.2 Loss function

As will be described in section 5.1, we experimented with a variety of loss functions, until we finally arrived at what we termed the 'LinearInfoNCE', which is inspired by the InfoNCE loss[12].

Let G be a set of objects, denote $G = \{O\} \sqcup G_+ \sqcup G_-$, where O is the original sample, and G_+, G_- are the sets are positive and negative examples respectively. The LinearInfoNCE function acting on the encoder ϕ and the sample set G, is defined as:

$$l(\phi; G) \coloneqq \frac{1}{|G|-1}\left[\sum_{s\in G_+} -\log\left(\frac{\langle\phi(s),\phi(O)\rangle+1}{2}\right) + \sum_{s\in G_-} -\log\left(1-\frac{\langle\phi(s),\phi(O)\rangle+1}{2}\right)\right]$$

One way to understand the above function is that we model the probability of two objects being the same to be a linear (or more precisely affine) function of the cosine of the angle between the two vectors, this a result of the vectors being unit vectors and therefore:

$$\langle\phi(s),\phi(O)\rangle = \cos(\angle\big(\phi(s),\phi(O)\big)$$

Since the range of the cosine function is [-1,1], while a probability's range is [0,1], we define the model the probability of two signatures being outputs of the same shape as $\frac{\langle\phi(s),\phi(O)\rangle+1}{2}$ which has the adequate range. Notice that for two similar signatures the angle would be small, and thus the cosine would be close to 1 (its maximal value), leading to that the probability too would be close to 1 as expected. If the angle is large, the closer it gets to $\pi\ [rad]$, the smaller the cosine becomes, so that finally at $\pi\ [rad]$ it becomes -1, making the probability 0 as we would expect.

Another important property of the cosine measure of the angle is that it is symmetric, and thus it is a function of the amount of difference in angle and not the direction, which is more fitting, as there is no specific ordering that should be followed between the shapes or their signatures and adding such an ordering can make the distribution on the sphere biased.

Our objective is to maximize the probability when the example is a positive one and to minimize it otherwise, and since we are using a probability measure that is a linear function of the cosine of the angle, this would lead to minimizing the angle between signatures of the same shapes which satisfies (2) and maximizing it for different shapes which satisfies (3), meaning that the minimization of such a loss achieves our original objective.

We observe that maximizing the probability when the example is a positive one, and minimizing it otherwise, could be modeled as a binary classification problem, and thus a Binary Cross-Entropy is used applied the above-described probabilities, finally giving us the LinearInfoNCE loss.

4.3 Architecture

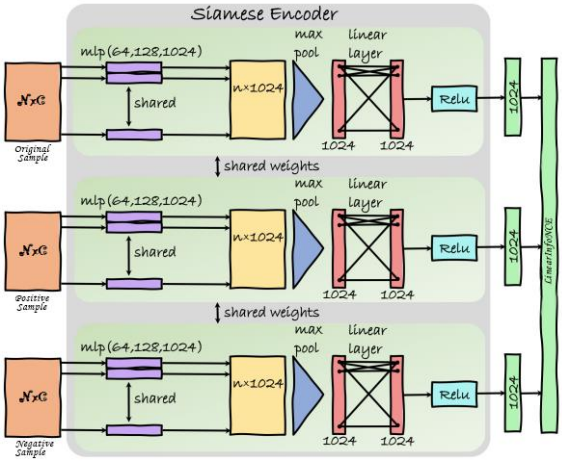


Figure modified from [13]

Our focus was not to engineer a highly sophisticated network that fits our problem exactly, rather we aimed to use a simple general network, but to optimize the training scheme such that it can be used with a wide range of datasets and different networks.

We make no assumption on the point cloud vertex ordering, and as such our method is applicable to general point clouds and natural depth projections; furthermore, we do not assume a triangulated shape or knowledge of the geodesic distances, making our algorithm simple, efficient and general.

We used an encoder inspired by O. Halimi et. Al's encoder [13] used in a Siamese network setup, and then added a neural network with either 1 or 2 Linear layers (commonly termed MLP) for classification tasks.

The encoder is a simple PointNet [14] inspired network, meaning that it takes the features of each point which in our case are either the coordinates alone or the coordinates alongside the components of the second order moments (i.e., x^2, y^2, z^2, xy, zy and xz as detailed in the section 4.5.1), and then feeds those into an MLP that its weights shared by all the points. These are all concatenated and a max-pooling operation is applied along the points axis, giving a vector invariant to permutations on the order of points, and thus this vector could be fed into an MLP to get the final encoding.

4.4 Dataset

We used the FAUST and DFAUST datasets, both of which are 3D registered scans of humanoid figures, and have a trivial vertex correspondence for their 6890 vertices. Due to computational considerations, we utilize only 1035 points from each point cloud, where the sampling was done using Pyvista's default down sampling technique. The shapes are provided triangular meshes but we extract simpler point-cloud representations from each triangulated mesh for the sake of simplicity and generalizability.

4.4.1 DFAUST:

The DFAUST dataset consists of approximately 40,000-point clouds in 3D, which consist of 10 different people (shapes), each with around 10-15 different animations containing around 200-400 3D registered triangular meshes.

To build our training, validation, and testing sets, we had to reduce the correlation between the different sets as much as possible without making the dataset smaller than it already is.

5 full-length animations are shared amongst every person in the dataset, out of these 2 were chosen to be the validation set, and another 2 to be the test set, we do this for every given person, and this results in approximately an 80,10,10 split (32K, 4K, 4K).

As will be described in section 4.5.2, one of the methods that we use to solve the problem of rotation invariance is PCA normalization. We note that this is a computationally heavy task which increases the training time by an order of magnitude. To deal with this issue, we created a version of the DFAUST dataset that had already been normalized using PCA, but as we described in section 4.5.2, the PCA normalization in unique up to flipping the signs of each coordinate independently, hence we implement an augmentation that does exactly that; when training we load the pre-normalized data and apply the augmentation which is computationally

light and does not increase our training time, but applying Lemma 1 (see below) we receive an equivalency between the simple augmentation and applying a random rotation and then PCA normalization.

4.4.2 FAUST:

To understand how well our model generalizes to unseen shapes, we use the FAUST dataset, which contains 10 people in 10 different poses each.

Here we went with a uniformly random 80% train and 20% test split over each shape in the dataset.

4.5 Rotation Invariance

The basic networks were able to perform reasonably well on the original data, as we show in the section 1.1.1, but when using the proposed network whilst augmenting the data by adding a random rotation makes the network work considerably worse, yielding unsatisfactory results.

Rotation Invariance is desirable since real-world data might not always be aligned, e.g., cameras might be tilted, surfaces might be uneven and shapes might be rotated; an algorithm that

Multiple solutions were considered, most notably:

1. Alignment networks[14], described in detail in the section 5.4.
2. Providing second order moments as inputs to the network.
3. Using PCA to normalize the inputs.

Following is an explanation for each of the above methods.

4.5.1 Utilizing Second Order Moments as Inputs

A solution that was inspired by [15], instead of providing the network a point cloud where each point's features are its x, y and z coordinates, we calculate the second order moments i.e., x^2, y^2, z^2, xy, xz and yz, and provide them alongside the point-cloud coordinates as inputs for the network. This solution is simpler than the one shown in [15], as it doesn't use K-NN to feed each points the moments of neighboring point. Our approach clearly doesn't provide localization information, but still provides information that would be otherwise hard to calculate.

One might initially assume that this is unnecessary as the network is capable of learning the above-mentioned moments by itself if needed. However, since they are not linearly dependent on first moments, it has been shown to take number of layers logarithmic to the largest multiplication, which would make the network deeper and hence more susceptible to exploding gradients, and undesirably increase the size of the model hypothesis space.

In addition, higher moments provide us with meaningful geometric information. For example, together they construct the covariance matrix. The spectral decomposition of the covariance matrix supplies us with the principle components, the primary directions of change for each point cloud, acting as orientation axis, meaning that this information could be used in order to learn a canonical orientation for shapes.

Results using the above method are provided in section 6.2.

4.5.2 PCA Normalization

Principle component analysis [16], is a classic algorithm that is used traditionally for linearly embedding data into lower-dimensional spaces. This is done by analyzing the second-moment matrix of the data (a.k.a., the covariance matrix of the data) and determining which eigenvectors correspond to the largest eigenvalues. This could be shown to achieve the minimal MSE between the original point-cloud and any representation using a set of n vectors.

Formally, we aim to find T an orthogonal projection matrix such that:

$$T = \underset{U \in \mathbb{R}^{n \times m}}{\operatorname{argmin}} \{\mathbb{E}_x[\|UU^* x - x\|_2^2]\}$$

Where:

- x is a sample from our data distribution, $x \in \mathbb{R}^m$.

- m is the dimension of a sampled point from our data.
- n is the chosen embedding dimension.
- U is the target matrix of orthonormal column vectors.

Note that we could generalize the problem to general linear projections instead of orthogonal ones, but it could be easily proven that there exists an optimal solution of the generalized problem which is also an orthonormal matrix, and thus an optimal solution for the problem above is optimal for the generalized problem.

The solution to the above problem is:

$$T = \begin{pmatrix} | & | & ... & | \\ \hat{u}_1 & \hat{u}_2 & ... & \hat{u}_3 \\ | & | & ... & | \end{pmatrix}$$

Where $\hat{u}_i$ is the normalized eigenvector with the i[th] highest eigenvalue of the covariance matrix $\mathbb{E}[XX^*]$.

This could be shown to be equivalent to $\hat{u}_i$ satisfying:

$$\hat{u}_i = \underset{\substack{\hat{u}_i \in \mathbb{R}^m \, s.t. \\ \forall 0<j<i, \hat{u}_j \perp \hat{u}_i \\ \wedge \|\hat{u}_i\|_2 = 1}}{\operatorname{argmin}} \left\{ \mathbb{V}ar_x\left(\hat{u}_i^T x\right) \right\}$$

Meaning that these are the directions in which the orthogonal projections of the points have the largest variance, this could be exploited to gain information about the orientation of the shape [23], as in our case the shapes are humanoid figures, and thus for most positions the direction with the highest variance is the one that goes parallel to the spine.

And therefore, if we solve optimization problem but use $n = m$, we get $T \in \mathbb{R}^{3\times3}$ which is a rotation matrix, which orients each mesh such that the first coordinate is the one with the highest variance, the second is the second highest and finally the third is the one with the least amount of change, we do this in hopes of reducing the degrees of freedom enough so that the network learns a signature invariant to the slight orientation differences that might be the result of different poses. For example, a person in the splits position (legs wide open) might have a higher variance in the direction of the legs (to his sides) rather than his spine.

This allows our network to learn a simpler task on PCA oriented shapes, and when combining the PCA-normalization layer into our algorithm this gives a general pipeline that deals with isometric and Euclidean transformation, without requiring the network to learn to deal with the general Euclidean transformations by itself.

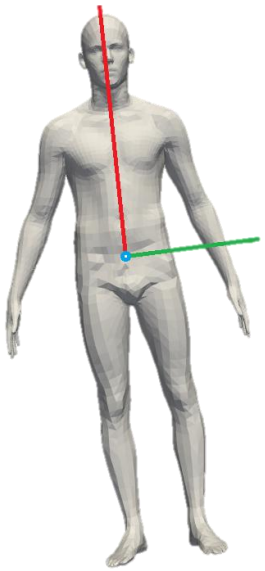


Figure: A mesh with the principle components plotted over it where red, green, and blue correspond to the eigenvalues in descending order.

Lastly, we show that this provides us with an algorithm that is invariant to rotations up to multiplication an axis mirroring rotation matrix.

Lemma 1: the above described normalization method is invariant to rotations up to a multiplication with an axis mirroring rotation matrix:

$$diag\{\pm1, \pm1, \pm1\}$$

Note: the elements are independent, meaning that there are 8 different options in total.

Proof. Given a rotation matrix R, we define $Y = RX$, and thus the covariance matrix of Y is:

$$\mathbb{E}[YY^T] = \mathbb{E}[RXX^TR^T] = R\mathbb{E}[XX^T]R^T = R\mathbb{E}[XX^T]R^T$$

And thus for v an eigenvector of $\mathbb{E}[XX^T]$ with eigenvalue λ, one can easily see that $u = Rv$ is an eigenvector of $\mathbb{E}[YY^T]$ with the same eigenvalue:

$$\mathbb{E}[YY^T]u = \mathbb{E}[YY^T]Rv = R\mathbb{E}[XX^T]R^TRv = R\mathbb{E}[XX^T]v = R\lambda v = \lambda Rv = \lambda u$$

And the opposite direction is trivial since $v = R^Tu$, and therefore it is a direct result of the above result.

A direct result from this is that if $T = (\hat{u}_1, .. \hat{u}_n)$ is a solution to the original problem, then after rotation $\tilde{T} = (R\hat{u}_1, .. R\hat{u}_n) = R(\hat{u}_1, .. \hat{u}_n) = RT$ is the solution for the rotated problem, and therefore the final embedding would be:

$$\tilde{T}^TY = T^TR^T(RX) = T^TX$$

$$R_i^2 \coloneqq \tilde{\alpha},$$

Which is exactly the original solution.

As for the 8 solutions we mentioned above, we note that for every normalized eigenvector u, $-u$ is a valid solution as well, and therefore we might in different runs get u and $-Ru$, and thus the difference in sign.

■

Lastly, we note that theoretically if two or more eigenvectors have the same eigenvalue, the number of possible solutions would become infinite, since they would form a solution space of a dimension higher than 1, meaning that there is an infinite number of orthonormal basis for this space. We neglect such cases for two reasons. Firstly, having two directions have the same eigenvalue is uncommon for general natural shapes. Secondly, due to numeric instabilities, the probability of actually getting two exactly equal eigenvalues drops substantially. We have asserted this assumption holds empirically.

5 Experiments

5.1 Loss Functions

In section 4.2 we showed our proposed loss function which we call LinearInfoNCE, but other loss functions were tested to determine the best fitting loss function, among those are:

5.1.1 Contrastive Loss

Contrastive loss [17] basic well-known loss that aims to minimize the L2 distance between the signatures that come from the same shape, and aims to maximize the L2 distance between the signatures that come from different shapes.

$$\mathcal{L}(W_1, W_2, Y) = (1 - Y)\frac{1}{2}D_{W_{12}}^2 + Y\frac{1}{2}\max\left(0, \mu - D_{W_{12}}\right)^2$$

Where:

- W_i is the signature of the i[th] object.
- $D_{W_{ij}}$ is the L2 distance between W_i, W_j meaning $D_{W_{ij}} = \left\|W_i - W_j\right\|_2$.
- Y is a binary variable, which receives the value 1 iff W_1, W_2 come from different shapes.
- μ is a hyperparameter, which defines the 'cutoff'.

As one can see, if the signatures come from the same shape, Y would be 0, and therefore the loss would be:

$$L_{Positive}(W_1, W_2) = \frac{1}{2}D_{W_{12}}^2$$

Meaning that optimizing the loss would minimize the distance as planned.

Otherwise, if the signatures come from different shapes, Y would be 1, and therefore the loss would be:

$$L_{negative}(W_1, W_2) = \frac{1}{2}\max\left(0, \mu - D_{W_{12}}\right)^2$$

Optimizing this term would maximize $D_{W_{12}}$ as planned, but it would stop if the distance exceeded the given threshold μ, this is very necessary, as if we were to use $Y\frac{1}{2}{D_{W_{12}}}^2$ instead, the network could learn to prefer increasing the difference between signatures of different shapes, but not focus as much on bringing the signatures of the same shape close, because increasing a large number that is squared has stronger effect than decreasing a small squared number, in other words it is more beneficial to increase the distance between negative example that to reduce the distance between positive examples.

5.1.2 Triplet Loss

Triplets loss[18] is yet another common loss function, it too uses the L2 norm, but it differs slightly, its goal could be interpreted as maximizing the difference between the average distance between signatures of the same shape and the average distance between signatures of different shapes.

To achieve this the proposed loss function is modified to:

$$\mathcal{L}(x, x_p, x_n) \coloneqq \max\Big(0, \gamma\|f(x_n) - f(x)\|_2^2 - (1-\gamma)\big\|f(x_p) - f(x)\big\|_2^2 + \alpha\Big)$$

Where:

- x is some sample.
- x_p is a positive sample, meaning a sample of the same shape in a different pose.
- x_n is a negative sample, meaning a sample of a different shape.
- $\gamma \in (0,1)$ is a weighting hyperparameter.
- α is the difference threshold hyperparameter.
- f is the encoder function.

The gamma allows us to pick a value that normalizes both terms to be of the same order of magnitude or to give more emphasis to one of them. on the other hand, the α combined with the clipping operation means that if the (signed) distance drops below $-\alpha$ (meaning that the difference is over α) the loss would become zero. This way the network would not be tempted to overfit a subset of the training data while neglecting the rest, instead it stops optimizing the given triplet when it reaches a resolution ability of α.

5.1.3 SigmoidInfoNCE

This loss has the same motivation described for LinearInfoNCE, but it differs only in it how it transforms the cosine similarity into a probability measure, LinearInfoNCE, as the name implies, does this linearly (or rather affinely), but SigmoidInfoNCE does this by exploiting the sigmoid function, and therefore it is calculated by the equation:

$$l(\phi; G) \coloneqq \frac{1}{|G|-1}\left[\sum_{s\in G_+} -\log\left(\sigma\left(\frac{\langle\phi(s), \phi(O)\rangle}{\tau}\right)\right) + \sum_{s\in G_-} -\log\left(1-\sigma\left(\frac{\langle\phi(s), \phi(O)\rangle}{\tau}\right)\right)\right]$$

Where:

- τ is a temperature hyperparameter that controls the sensitivity of the probability to changes in the cosine similarity.

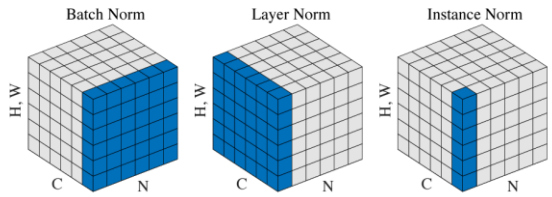
In other words, this is just a Binary Cross Entropy loss where the probabilities are calculated by $\sigma\left(\frac{\langle\phi(s),\phi(O)\rangle}{\tau}\right)$.

5.2 Normalization

Normalization of the outputs of the layers of the network has been repeatedly shown to improve stability by reducing the covariate shift [19], reducing overfitting and accelerating the training of the network [20].

To reap the full benefits of normalization, one must pick the right kind of normalization for the task, the top candidates for our specific task were Batch Normalization [24], Layer Normalization [25], and Instance Normalization [26] (which using Pytorch terminology is a special case of

layer norm, but for the sake of clarity, in the following section we are going to refer to them by different names).



As could be seen in the above figure[21], N is the batch size, C is the number of channels and H,W represent any additional dimensions. The different normalization schemes differ in what dimension each of them normalizes, the pixels annotated in blue in the above figure are normalized using the same standard deviation and mean.

Batch Norm, given a batch of samples, calculates the standard deviations and means of the set of all features of each channel, and then normalizes each channel of every sample in the batch using the corresponding standard deviation and mean. This gives independence between the channels but creates a dependence between the features of each channel and the samples of the batch.

Layer Norm, given a sample, calculates the standard deviation and mean of the set of all features, and then normalizes every feature using these values. Means that we normalize each sample independently of other samples, and assume a shared distribution of each sample features.

Instance Norm, given a sample, calculates the standard deviations and means of the set of all features of each channel, and then normalizes the features of each channel of the given sample using the corresponding standard deviation and mean. This gives each sample independence of other samples in the batch, as well as independence between different channels of the sample.

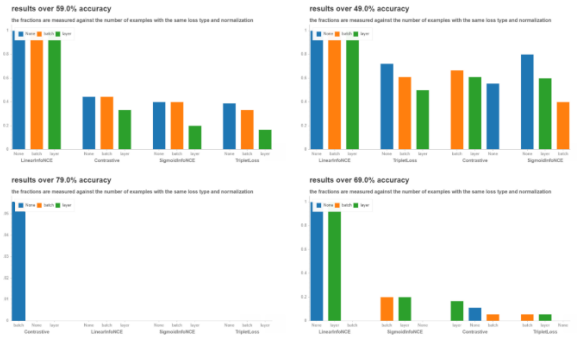
In our case we tested Batch Normalization and Instance Normalization where the dimension that we normalize along is the points, meaning that we learn a statistic for the distribution of the points and normalize across it. Notice that is a permutation invariant operation, which is of top importance, as we do not assume some ordering on the points, meaning that 'the same point' could appear in different positions in the vector in different samples.

5.3 Loss Functions and Normalization

To check which normalization technique and loss function fit our goals best, we used the FAUST dataset, to run a simple shape classification test (we elaborate on this in section 6.1.1) using a K-Fold validation technique, and then we used 4 different thresholds of accuracy $\{49\%, 59\%, 69\%, 79\%\}$, and calculated the percentage of models that surpassed each threshold.

The reason for checking pairs comprised of a loss function and a normalization technique, rather than checking each individually, is that there might be a correlation between the two, and thus a need for checking the entire gird instead of sampling points on each axis (where the first axis is the loss function and the second is the normalization technique).

Hence for each pair of a loss function and a normalization technique we perform K-Fold validation for K=5, and calculate the percentage of the K trained models that got an accuracy above each given threshold.



As can be seen from the above figure, the LinearInfoNCE performs more consistently than other loss functions, and with the added benefit of it being the only function with no hyperparameters, it was a clear decision.

The normalization on the other hand was not as clear of a decision, but a slight advantage of using

no normalization could be observed in the above testing, furthermore while training on DFAUST not using normalization increases the accuracy by 5% and thus it was decided not to use it.

Many reasons could play a role in this interesting result. Our theory is that neither of the normalization techniques makes sense given the type of data, as different meshes should have different distributions of their points and channels because of the differences in the pose. The difference in these statistics could be important. For example, a high standard deviation in some direction could indicate that this is the direction of the spine of a standing mesh, on the other hand, a low standard deviation could indicate the depth direction of a mesh. Normalizing the data might hide away this information and thus make it harder to orient the shape.

5.4 Alignnet

One possible solution for the rotation invariance problem that was described above is training an alignment network as proposed in [14], where the network attempts to learn the 6D pose vector (meaning the 3D translation, and a 3D representation of a general 3D rotation using Euler angles).

To train such a network, we note that by default the DFAUST point clouds are aligned perfectly, by applying random 3D rotations sampled uniformly and 3D translations sampled from a multivariate normal distribution to its samples, we build a dataset of shapes transformed by a Euclidean transformation alongside the transformations' parameters. Using this dataset, a network could be trained to predict a Euclidean transformation's parameters, given a transformed object.

As for the loss, different losses showed different results, for the translation a simple L2 metric was all that was needed, as for the rotation, there were more options, notably:

1. L2 loss over the difference between the predicted Euler angles representation of the predicted and ground truth rotations.

   Note: in order to avoid problems pertaining to the cyclicity of the Euler angles representation, the last non-linear layer's output is bounded to one cycle of the angle's range.

2. L2 loss over the difference between the rotation matrix induces by the predicted Euler angles and the ground truth rotation matrix.
3. L2 loss between the rotation matrix transposed multiplied by the ground truth matrix, and the unit matrix, the reasoning behind this loss is that it is equivalent to the predicted rotation matrix's inverse canceling out the ground truth rotation, which is the original objectives.

Testing showed slight differences between the different losses, but overall the networks were unable to transfer over to FAUST after training on DFAUST and scored lower overall on DFAUST on the subsequent classification task. We assume that this happens because we trained the network on DFAUST which contains a low number of different shapes, and thus the network could learn to overfit them (e.g., classify the shape and then understand its orientation based on its identity), which would make the network unable to generalize to other datasets such as FAUST.

6 Results & Comparison

6.1 Testing Method

To quantitively measure the quality of the learned signatures, we argue that if our signature satisfies eq (1), then it should contain all the necessary information to classify the different shapes, but it should contain minimal to no information about the pose, and thus two main tests were proposed.

6.1.1 Shape Classification

The first test is a simple shape classification test, where we add an MLP head that takes the signature from the pre-trained encoder (which we freeze when testing), and then train the head to classify the different shapes (in our case, humanoid point-clouds).

A high accuracy suggests that the signature is unique. For each different shape there exists a different signature, and thus they can be distinguished one from the other, satisfying eq. (3).

Notice that this does not mean that the network has learned a pose invariant signature since each shape can still get different outputs for each pose, but the meaning of this is that each shape's signature space (spanned by the embeddings it gets for all its different poses) has a small or no intersection with other shapes' signature spaces, and thus the satisfaction of eq. (2) is not implied by high accuracy in this test.

Technical Details:

We make use of two tests, one where the MLP head is a one-layer MLP, making the test a linear classification test. If successful it means that a linear separation was achieved in the signature space.

The other test is a two-layer MLP, allowing it to learn a more complex separation, but without having it be able to become too specific\overfit the data.

We run the test in two settings, once on DFAUST, and once for cross-dataset generalization on FAUST. In both cases, the signature network is trained on DFAUST, this gives us two measures one for accuracy on similar data measured on DFAUST, and one for the generalization ability measured on FAUST.

6.1.2 Pose Classification

The second test is a pose classification test. The is similar to the shape classification test described above, meaning that we add an MLP head as described and learn a classification task on the signatures where the target is to predict the correct pose.

A low top accuracy on this test means that the signature retains low (or no) information about the pose of the shape, meaning that it has achieved pose invariance, hence satisfying eq. (2).

Technical Details:

The same scheme that was used for the shape classification task was used here, with one difference, DFAUST contains 4000 frames for each shape on average. Learning such a classification task would have us learn an output vector of dimension 4000, but since our data is relatively sparse (40 samples), this means that on average each frame appears 10 times, this is not sufficient for learning to predict the pose.

As a result, the pose network is tested only on FAUST.

6.1.3 Putting It All Together

To say that an algorithm has achieved our goals (to a certain degree), it needs to score high on the shape classification test, but low on the pose classification test, since together they mean that the algorithm satisfies both eq. (2) and eq. (3) which as explained in the introduction mean that the algorithm satisfies eq. (1), our mathematical definition of learning an invariant signature.

6.2 Results

6.2.1 Algorithms

The models that we show below are:

- Baseline classifier: the encoder network that we described, with a classification head attached. Note that for testing a new head is attached and retrained.
- SN w/o moments: Siamese network (our proposed method), where the inputs are the point clouds, and the features are the x,y, and z values.
- SN w/ moments: Siamese network (our proposed method), where the inputs are the point clouds, and the features are the x, y, and z values, alongside the second order moments $(x^2, y^2, z^2, xy, yz\ and\ xz)$.
- PCA algorithm: the inputs are normalized using the PCA normalization method described in section 4.5.2, and then fed into the algorithm.

- GDD: GDD descriptors are used as descriptors for the object after being flattened, notice that since both DFAUST and FAUST have a 1:1 correspondence, and we do not permute the points, the signature (because of the constant ordering) is also a global signature, and thus could be classified using our usual MLP methods.

6.2.2 Tests

- Object classification:
  - DFAUST – 1-layer head: classifying the signatures into shapes using a one-layer MLP head on the DFAUST dataset.
  - DFAUST – 2-layer head: classifying the signatures into shapes using a two-layer MLP head on the DFAUST dataset.
  - FAUST 2-layer head: classifying the signatures into shapes using a two-layer MLP head on the FAUST dataset.
- Pose classification:
  - FAUST – 2-layer head: classifying the signatures into poses using a two-layer MLP head on the FAUST dataset.

6.2.3 Without Rotation:

c	Object classification (↑)			Pose Classification (↓)
Algorithm \ Test	DFAUST – 1 layer head	DFAUST – 2 layer head	FAUST – 2 layer head	FAUST – 2 layer head
Baseline classifier	91.11%	91.08%	65.23%	40.46%
SN w/o moments	99.54%	99.1%	65.23%	15.8%
SN w/ moments	99.86%	99.54%	40.65%	35.64%
GDD	12.60%	92.29%	25.53%	10%

6.2.4 With Rotation:

	Object classification (↑)			Pose Classification (↓)
Algorithm \ Test	DFAUST – 1 layer head	DFAUST – 2-layer head	FAUST – 2-layer head	FAUST – 2-layer head
Baseline classifier	82.44%	82.43%	30.09%	70.26%
SN w/o moments	75.14%	74.65%	25.96%	33.74%
SN w/ moments	77.47%	80.34%	50.61%	36.61%
PCA baseline	12.64%	11.61%	30.46%	23.98%
PCA SN w/o moments	88.22%	86.34%	65.86%	10.25%
PCA SN w/ moments	91.48%	92.89%	55.39%	18.06%
GDD	12%	92.52%	25.22%	10.99%

- Yellow indicates the best result, Grey indicates the second-best result.

7 Conclusions

Our framework achieves a 9% improvement over the baseline and 8% over the SOTA achieving near 100% accuracy on the aligned data. This means that our framework encourages learning a meaningful descriptor that does not overfit the training data as much as end-to-end classification would.

We argue that the improvement over aligned data is one of practical use as well. In the general case for humanoid scans we can assume that the rotation would only be 2 dimensional, this is because in most cases people are in upright positions. The same goes for furniture, vehicles, etc.

Most notably, on data augmented with random rotations our framework achieved results consistently better than the same architecture trained as a classifier (the baseline). This implies that our training scheme urges the network to learn valuable information that generalizes better since on the test set it still retains a higher

accuracy than the baseline. In other words, given the same data and the architecture our proposed training scheme performs better than training a classifier in an end-to-end fashion.

Additionally, we see that since the baseline has no explicit reason to learn pose invariant signatures, the results show that the pose could be classified better from the signatures it produces, unlike the Siamese training scheme that we propose that directly punishes such behavior.

As for the GDD, as could be seen from its construction, it is completely rotation invariant, and almost completely pose invariant (the almost is the result of the geodesic distances being distorted near the joints and skin folds; GDD achieves the best pose classification accuracy – as expected). But interestingly, using one layer for classification achieves very bad results, pointing to a non-linear separation between the objects, but as soon as we use a 2 layer MLP we get 92% accuracy which is a tie with our best algorithm.

However, of FAUST, we can see that the GDD descriptor does not achieve as good an accuracy, since it's a much smaller dataset and the variances might be small enough that it is hard to learn a good separation between the different objects.

We conclude that our method achieves results that are competitive with the SOTA, alongside a simpler (linear) separation, and thus allowing for simpler post-processing algorithms.

Additionally, we note that our network is comprised of linear and convolutional layers and thus it is much more efficient than GDD and other MDS counterparts that have cubic terms in their complexity.

To summarize, we propose a deep learning-based approach that utilizes a Siamese metric learning scheme to learn a pose and orientation signatures that is both competitive and more efficient than the SOTA on rotated data, and superior to the SOTA on aligned data.

8 References

[1] G. Pai, A. Wetzler, and R. Kimmel, “L EARNING I NVARIANT R EPRESENTATIONS O F,” pp. 1–11, 2017.

[2] R. S. Society, “Review of the Development of Multidimensional Scaling Methods Author (s): A . Mead Source : Journal of the Royal Statistical Society . Series D (The Statistician) , 1992 , Vol . Published by : Wiley for the Royal Statistical Society Stable URL : https,” vol. 41, no. 1, pp. 27–39, 1992.

[3] J. A. Burgoyne and S. McAdams, “Non-linear scaling techniques for uncovering the perceptual dimensions of timbre,” *Int. Comput. Music Conf. ICMC 2007*, no. April 2014, pp. 73–76, 2007.

[4] J. A. Sethian, “Fast Marching Methods,” *SIAM Rev.*, vol. 41, no. 2, pp. 199–235, 1999, doi: 10.1137/S0036144598347059.

[5] K. Crane, C. Weischedel, and M. Wardetzky, “The heat method for distance computation,” *Commun. ACM*, vol. 60, no. 11, pp. 90–99, 2017, doi: 10.1145/3131280.

[6] G. Shamai, Y. Aflalo, M. Zibulevsky, and R. Kimmel, “Classical scaling revisited,” *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 International Conference on Computer Vision, ICCV 2015, pp. 2255–2263, 2015, doi: 10.1109/ICCV.2015.260.

[7] E. Peterfreund and M. Gavish, “Multidimensional scaling of noisy high dimensional data,” *Appl. Comput. Harmon. Anal.*, vol. 51, pp. 333–373, 2021, doi: 10.1016/j.acha.2020.11.006.

[8] S. Martin and J. P. Watson, “Non-manifold surface reconstruction from high-dimensional point cloud data,” *Comput. Geom. Theory Appl.*, vol. 44, no. 8, pp. 427–441, 2011, doi: 10.1016/j.comgeo.2011.05.002.

[9] S. M. Holland, “Non-Metric Multidimensional Scaling (MDS),” *J. Cell Biol.*, no. May, p. 8, 2008.

[10] E. Beutner and U. Kamps, “Order restricted statistical inference for scale parameters based on sequential order statistics,” *J. Stat. Plan. Inference*, vol. 139, no. 9, pp. 2963–2969, 2009, doi: 10.1016/j.jspi.2009.01.017.

[11] G. Shamai and R. Kimmel, “Geodesic distance descriptors,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 3624–3632, 2017, doi: 10.1109/CVPR.2017.386.

[12] A. van den Oord, Y. Li, and O. Vinyals, “Representation Learning with Contrastive Predictive Coding,” 2018, [Online]. Available: http://arxiv.org/abs/1807.03748.

[13] O. Halimi *et al.*, “Towards Precise Completion of Deformable Objects,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12369 LNCS, pp. 359–377, 2020, doi: 10.1007/978-3-030-58586-0_22.

[14] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep learning on point sets for 3D classification and segmentation,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 77–85, 2017, doi: 10.1109/CVPR.2017.16.

[15] M. Joseph-Rivlin, A. Zvirin, and R. Kimmel, “Momenet: Flavor the moments in learning to classify objects,” *Proc. - 2019 Int. Conf. Comput. Vis. Work. ICCVW 2019*, pp. 4085–4094, 2019, doi: 10.1109/ICCVW.2019.00503.

[16] J. Kang and A. K. Patterson, “Principal component analysis of mRNA levels of genes related to inflammation and fibrosis in rats treated with TNBS or glutamine,” *Inflammatory Bowel Diseases*, vol. 17, no. 7. pp. 1630–1631, 2011, doi: 10.1002/ibd.21544.

[17] R. Hadsell, S. Chopra, and Y. LeCun, “Dimensionality reduction by learning an invariant mapping,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2, pp. 1735–1742, 2006, doi: 10.1109/CVPR.2006.100.

[18] F. Schroff, D. Kalenichenko, and J. Philbin, “FaceNet: A unified embedding for face recognition and clustering,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June, pp. 815–823, 2015, doi: 10.1109/CVPR.2015.7298682.

[19] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *32nd Int. Conf. Mach. Learn. ICML 2015*, vol. 1, pp. 448–456, 2015.

[20] T. Salimans and D. P. Kingma, “Weight normalization: A simple reparameterization to accelerate training of deep neural networks,” *Adv. Neural Inf. Process. Syst.*, no. Nips, pp. 901–909, 2016.

[21] Y. Wu and K. He, “Group Normalization,” *Int. J. Comput. Vis.*, vol. 128, no. 3, pp. 742–755, 2020, doi: 10.1007/s11263-019-01198-w.

[22] Olver, Peter J. Classical Invariant Theory . Cambridge: Cambridge University Press, 1999. Print.

[23] Paquet, E. “Nefertiti: a Query by Content Software for Three-Dimensional Models Databases Management.” International Conference on Recent Advances in 3-D Digital Imaging and Modeling : Proceedings, May 12-15, 1997, Ottawa, Ontario, Canada. [Place of publication not identified]: IEEE Computer Society Press, 1997. 345–352. Web.

[24] Ioffe, Sergey, and Christian Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift." International conference on machine learning. PMLR, 2015.

[25] Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton. "Layer normalization." arXiv preprint arXiv:1607.06450 (2016).

[26] Ulyanov, Dmitry, Andrea Vedaldi, and Victor Lempitsky. "Instance normalization: The missing ingredient for fast stylization." arXiv preprint arXiv:1607.08022 (2016).