Convolutional Neural Networks on Surfaces via Seamless Toric Covers

H. MARON, M. GALUN, N. AIGERMAN, M. TROPE, N. DYM, Weizmann Institute of Science
V. G. KIM, E. YUMER, Adobe Research
Y. LIPMAN, Weizmann Institute of Science

The recent success of convolutional neural networks (CNNs) for image processing tasks is inspiring research efforts attempting to achieve similar success for geometric tasks. One of the main challenges in applying CNNs to surfaces is defining a natural convolution operator on surfaces.

In this paper we present a method for applying deep learning to sphere-type shapes using a global seamless parameterization to a planar flat-torus, for which the convolution operator is well defined. As a result, the standard deep learning framework can be readily applied for learning semantic, high-level properties of the shape.

An indication of our success in bridging the gap between images and surfaces is the fact that our algorithm succeeds in learning semantic information from an input of raw low-dimensional feature vectors.

We demonstrate the usefulness of our approach by presenting two applications: human body segmentation, and automatic landmark detection on anatomical surfaces. We show that our algorithm compares favorably with competing geometric deep-learning algorithms for segmentation tasks, and is able to produce meaningful correspondences on anatomical surfaces where hand-crafted features are bound to fail.

CCS Concepts:
• Computing methodologies → Neural networks; Shape analysis;

Additional Key Words and Phrases: Geometric deep learning, Convolutional neural network, Shape analysis, shape segmentation

ACM Reference format:

1 INTRODUCTION

A recent research effort in the geometry processing and vision communities is to translate the incredible success of deep convolutional neural networks (CNN) to geometric settings. One particularly interesting scenario is applying CNNs for supervised learning of functions or labels over curved two-dimensional sphere-like surfaces. This is a common problem in analyzing human bodies, anatomical, and medical data.

Applying CNNs to extrinsic surface representation such as volumetric grids [Qi et al. 2016b] or depth map projections on extrinsic 2D cameras [Wei et al. 2016] requires working with 3D grids, or dealing with many camera and lighting parameters, and is very sensitive to deformations (e.g., human pose changes). While it might be possible to learn a representation that is invariant to these deformations, this requires substantial amount of training data. In contrast, the goal of our paper is to provide an intrinsic representation that would enable applying CNNs directly to the surfaces.

One of the main challenges in applying CNN to curved surfaces is that there is no clear generalization of the convolution operator. In particular, two properties of the convolution operator that are considered pivotal in the success of the CNN architectures are locality and translation invariance. It is possible to parameterize a surface locally on a geodesic patch around a point [Masci et al. 2015], however, this representation lacks global context. Sinha et al. [2016] proposed geometry images to globally parameterize sphere-like surface into an image, however, although continuous, their representation is not seamless (the parameterization is dependent on the cuts made for its computation), their space of parameterizations, namely area-preserving maps has a very high number of degrees of freedom (i.e., it requires an infinite number of point constraints to uniquely define a mapping) and therefore can represent the same shape in many different arbitrary ways in the image (see Figure 2 for an example). Lastly, the convolution on the geometry image is not translation invariant.

Fig. 1. Defining convolutional neural networks on sphere-like surfaces: we construct a torus (top-right) from four exact copies of the surface (top-left) and map it to the flat-torus (bottom-right) to define a local, translation invariant convolution (bottom-left). This construction is unique up to a choice of three points on the surface (colored disks).
Defining a local translation invariant convolution operator on surfaces is not trivial. The first hurdle is topological: the only surface type for which a translation invariant convolution is well defined is the torus (this will be explained in Section 4). However, clearly it is not possible to map sphere-like surfaces to a torus without introducing discontinuities, that is, mapping some neighboring points on the surface to distant ones on the torus. Information will propagate differently through discontinuities and locality of the convolution would be lost. The second difficulty is geometric: We want mappings (preserving orthogonal directions) to the flat torus using a very efficient algorithm. This defines a local translation invariant convolution on the 4-cover, see example of a single convolution stencil in Figure 1 - bottom row. This construction is unique up to a choice of three points on the surface; the convolution depicted in Figure 1 is created by the three points (shown as colored disks) in the bottom-left inset.

This construction provides a six dimensional space of seamless convolutions on a sphere-like surface: Every choice of a triplet of points corresponds to a unique conformal map which in turn defines a convolution operator, or equivalently, a conformal flat-torus structure on the 4-cover of the surface. Since isometries are in particular conformal maps this construction is also invariant to isometric deformations of the shapes. The relatively low dimension of the convolution space allows efficient sampling of this space in the context of data augmentation. The conformality preserves the directionality of the translation directions on the flat-torus but introduces scale changes; in that sense the triplet acts as a magnifying glass - zooming into different parts of the surface.

We employ the above constructions for supervised learning over surfaces. Our goal is to learn a non-linear relation between "easy" vector valued functions over surfaces (e.g., coordinate functions, normals, curvature or other commonly used geometric features) and target "hard" vector valued functions (e.g., semantic segmentation or landmark labeling). The conformal map from the 4-cover to the flat-torus will be used to seamlessly transfer these functions to the flat-torus which will be used as our domain for training. To leverage existing image-based CNN architecture and optimization algorithms on the flat-torus domain, we provide three novel technical components: (i) A cyclic-padding layer replaces zero padding to achieve fully-translational invariance over the flat-torus; (ii) a projection layer on the function space of the surface to properly map functions between the original surface and its 4-cover, and (iii) an aggregation procedure to infer prediction from multiple triplets.

Experiments show that our method is able to learn and generalize semantic functions better than state of the art geometric learning approaches in segmentation tasks. Furthermore, it can use only basic local data (Euclidean coordinates, curvature, normals) to achieve high success rate, demonstrating ability to learn high-level features from a low-level signal. This is the key advantage of defining a local translation invariant convolution operator. Finally, it is easy to implement and is fully compatible with current standard CNN implementations for images.

2 PREVIOUS WORK

Recent advances in convolutional neural networks (CNNs) motivated many researchers to apply these methods to geometric data. Extrinsic representations, such as 3D volumetric grid [Qi et al. 2016b; Wu et al. 2015], 2D projections [Kalogerakis et al. 2016; Su et al. 2015; Wei et al. 2016], or point coordinates [Qi et al. 2016a], have many shortcomings when analyzing non-rigid 2D manifolds: they are sensitive to articulations, they do not leverage the intrinsic structure of the geometry, and only allow very coarse representations. While these limitations can be addressed by analyzing the manifolds directly, applying CNNs to surfaces is challenging because they do not come with a planar parameterization, and thus are not directly suitable for deep learning. One possible way to address this limitation is to represent a surface as a graph of triangles and use spectral convolutions [Bruna et al. 2013; Defferrard et al. 2016; Henaff et al. 2015]. However, this representation does not take advantage of the fact that we are analyzing 2-manifold that can be parameterized in 2D domain. Another disadvantage of spectral methods (which is targeted by [Yi et al. 2016]) is their difficulty with cross-shape learning which stems from the fact that the spectral decomposition of each shape can be inconsistent. We next discuss existing techniques which preform deep learning on 2-manifolds and parameterization methods they use.

For segmentation task, Guo et al. [2015] proposed to lay out per-triangle features to a single 2D grid, and used CNN to classify each triangle. This approach cannot use contextual information on relationships between different triangles on the same surface unless this relationships are encoded in the input features.

The first paper adapting neural networks to surfaces, Masci et al. [2015], use local geodesic polar coordinates to parameterize a surface patch around a point; and map features to this patch. This parameterization requires modifying the traditional convolution filter to account for angular coordinate ambiguity, essentially ignoring patch orientation. In a follow up work, [Boscaini et al. 2016] use anisotropic heat kernels in order to generate a local description of the input function and incorporate orientation.

For classification tasks, Sinha et al. [2016] parameterize the entire surface using geometry images [Praun and Hoppe 2003] combined with spherical area-preserving parameterizations. As mentioned briefly above, geometry images are not seamless and introduce a direction jump at the cut, see Figure 2. Additionally, the convolution over the geometry image is not translation invariant since it represents a sphere-like topology. Finally, since geometry images are computed using area-preserving mappings, which have an infinite number of degrees of freedom, they can produce a wide variability of different planar realizations which will make the learning process
we only need to solve a sparse system of linear equations per triplet
vector valued functions over the surface
functions for which no simple algorithm exists, such as a semantic
can be computed efficiently over the surfaces, such as coordinate
and ground-truth “hard” functions
3.1 Overview
surfaces.

Our goal is to adapt CNN architectures to geometric
networks that leverages the spatial relationships of local regions
learning easier. Our map computation routine is very effective, as
we only need to solve a sparse system of linear equations per triplet
of points.

3 METHOD
Convolutional neural networks (CNN) is a specific family of neural
networks that leverage the spatial relationships of local regions
using mostly convolutional layers. Deep CNN’s with multiple con-
secutive convolutions followed by nonlinear functions have shown
be immensely successful in image understanding [Krizhevsky
et al. 2012]. Our goal is to adapt CNN architectures to geometric
surfaces.

3.1 Overview
Problem definition. Our training data consists of a set of triplets
{(Si, xi, yi)}iei of sphere-like surface meshes Si ∈ R3, “easy” d-
vector valued functions over the surface Si, denoted xi ∈ \mathcal{F}(Si, R^d),
and ground-truth “hard” functions yi ∈ \mathcal{F}(Si, L), where L =
{1, \ldots, L} is a label set. By “easy” functions we mean functions that
can be computed efficiently over the surfaces, such as coordinate
functions, curvature, normals or shape features; by “hard” we mean
functions for which no simple algorithm exists, such as a semantic
label (e.g., object part) that has to be prescribed manually.

Our goal is to find a non-linear mapping relating the “easy” and
“hard” functions on surfaces. Mathematically we are looking for a
function F,
\[ F: \mathcal{F}(S_i, \mathbb{R}^d) \rightarrow \mathcal{F}(S_i, \mathbb{R}^L) \] (1)
that takes as input a d-vector valued (“easy”) function over a surface
Si and produces a confidence L-vector valued (“hard”) function
over the same surface. That is, it produces a vector of confidences
F(x_i)[p] ∈ R^L per point p ∈ Si that correctly predicts its ground-
truth label y_i[p] ∈ L (i.e., the maximal value in F(x_i)[p] is achieved
at its y_i[p]-th coordinate).

CNN on the flat-torus \mathcal{T}. While CNN is a powerful tool for map-
ing “easy” to “hard” functions, existing architectures cannot run
directly over S. Therefore we propose to transfer functions to a
flat torus, denoted \mathcal{T}, and train CNN over that domain. The flat
torus space is favorable since we can use a traditional CNN with 2D
convolutions directly to solve the problem over \mathcal{T}, by discretiz-
ing the flat torus space as an m × n grid (we used m = n = 512).

Mapping S to \mathcal{T} is not trivial because these two domains have
different topologies. We address this issue by considering a new
topological construction S^4 (Section 3.2). The domain S^4 consists
of four copies of the surface cut in the same way to a disk topology
and stitched together to one (boundaryless) torus-like surface. We
map S^4 conformally to the plane, where these 4 surface copies
seamlessly tile the flat-torus. Note that this mapping is not unique,
and is defined by a triplet of points on S. Each triplet provides a
different image over \mathcal{T} where resolution (surface area scaling) is
non-uniform, and varies over S.

We address the mapping ambiguity by modifying network ar-
chitecture, training data, and the way we interpret the network
output (Section 3.3). First, we add a new cyclic-padding layer en-
abling translation-invariant convolutions (i.e., invariance to torus
symmetry). Second, we incorporate a projection operator layer that
ensures that our network’s output is invariant to symmetries of S^4
(i.e., multiple copies of the input surface). Both of these layers are
differentiable and support end-to-end training. Third, we sample
multiple triplets to produce multiple training images over \mathcal{T}, sub-
stantially augmenting our training data. And finally, as we analyze
a surface at test time, we aggregate several predictions over the
surface (produced with different triplets).

3.2 Transferring functions between S and \mathcal{T}
A key component of our approach is transferring functions between
the surface S and flat torus \mathcal{T}. That is, given a function x_i over
the surface Si we want to transfer it to the flat-torus in a seamless
way that preserves locality and topological (i.e., neighboring) relations.
We also want this transfer to be as-unique-as-possible and invariant
to isometric deformations of Si to avoid the need for unnecessary
data augmentation. We next show that there is a unique transfer
operator for a given triplet of points \mathcal{P} = {p_1, p_2, p_3} ⊂ S_i.

Since S and \mathcal{T} have different topologies, to create a desired seamless
map between these two domains we define an intermediate surface,
S^4, a torus-type 4-cover of S (branched cover, similarly to [Kälberer
et al. 2007]). To create S^4 we first make four copies of the

1The flat-torus is the planar square [0, 1]^2 with its opposite sides identified.
would produce the same surface \( S^4 \). Although this representation is more compact it will not be different orientations to points on different sides of the cut.

...and cut each one along the path

\[ p_1 \to p_2 \to p_3 \]

to obtain disk-like surfaces (Figure 3, left). Next, we stitch the four surfaces according to the instructions shown in Figure 1, top-right, to get a torus-like surface, \( S^4 \). Note that this construction is indifferent to the actual cuts used (e.g., \( a, b \) in Figure 3) and different cuts would produce the same surface \( S^4 \). Lastly, we compute a map \( \Phi_p : S^4 \to T \) taking \( S^4 \) conformally to \( T \) (see Figure 3, middle). In practice, we compute this map by first mapping a single disk-like copy of \( S \) in \( S^4 \) (e.g., Figure 3, left) to the plane using the method in [Aigerman and Lipman 2015] (we used the \( \{\pi/2, \pi, \pi/2\} \) orbifold) followed by duplicating the resulting planar mesh until we cover the representative square tile of the flat torus, namely \([0,1]^2\). For weights we used cotan weights with negative values clamped to \( 10^{-2} \) to ensure positivity and hence bijective mapping. Per triplet \( P \), this approximately-conformal map can be computed very efficiently by solving a sparse system of linear equations, where the resulting map defines a 2D position for each vertex of the disk-like \( S \).

We use \( \Phi_p \) to transfer functions between the surface \( S \) and the flat-torus \( T \). Given a function \( x \in \mathcal{F}(S, \mathbb{R}^d) \) we define its push-forward to the flat-torus, \( \text{push}_p(x) \in \mathbb{R}^{m \times n \times l} \), by

\[ \text{push}_p(x) = x \circ \Psi \circ \Phi_p^{-1}. \]

where \( \Psi : S^4 \to S \) is the projection map taking each point in \( S^4 \) to its corresponding point in \( S \). That is, given a cell \( i.e., \) pixel on the discretized torus, we map its centroid to \( S^4 \) via the map \( \Phi_p^{-1} \), and then to \( S \) via the map \( \Psi \). We evaluate \( x \) at that point and assign the corresponding \( d \)-dimensional vector value to the cell. In practice, we use Matlab’s “patch” command to generate each channel of \( \text{push}_p(x) \). Figure 4 visualizes “easy” functions \( x \) and their push-forward to \( T \).

An interesting alternative to the above construction of \( S^4 \) and the mapping to the flat torus \( S^4 \to T \) would be to use a single copy of the surface, \( S \), and a mapping to a sphere-type flat representation (Euclidean orbifold) \( S \to \mathcal{O} \). This corresponds to taking one quarter of the flat torus \( e.g., \) upper left quarter of the squares in Figure 4). Although this representation is more compact it will not be topologically correct as the convolution kernels will be applied in different orientations to points on different sides of the cut.

3.3 Neural Networks on the flat-torus \( T \)

Now that we are able to map functions between \( S \) and \( T \) we explain how we train CNN over the flat torus. A CNN over the flat-torus is defined as a non-linear function taking \( d \)-vector valued functions over the torus to \( L \)-vector valued function over the torus. Therefore we denote:

\[ f(\cdot, w) : \mathbb{R}^{m \times n \times d} \to \mathbb{R}^{m \times n \times L}, \]

where \( w \) denotes the network parameters.

We first describe the appropriate architecture for \( f \) on \( T \) that takes into account translational symmetries of \( T \) and the fact it is covered by four copies of the surface \( i.e., S^4 \). To train \( f \), we use multiple triplets \( P_k \) to push training examples on the surface \((S_i, x_i, y_i)\) to the flat-torus, augmenting our training data by mapping the same surface in different ways. We use these training examples to optimize for \( w \), parameters of the CNN. Lastly, we explain how our trained network can be used to analyze an input test shape.

**Network architecture for CNN on \( T \).** The input and output of the network \( f(\cdot, w) \) is represented in the form of discrete two dimensional images, and there are many state-of-the-art network architectures that have proven highly successful for this type of data. In this work, we used the FCN32 CNN architecture of [Long et al. 2015] with two main differences: First, since we need \( f \) to be fully-translation invariant and well-defined on the flat-torus we employ a cyclic padding instead of the standard zero padding used for images. This is crucial for seamless learning (as demonstrated by experiments in Section 5). Second, since there are four copies of \( S \) in \( S^4 \), several predictions over the flat-torus might correspond to the same point on \( S \). Thus, for the final output of the network \( f(\cdot, w) \in \mathbb{R}^{m \times n \times L} \) to be well-defined on \( S \) (so that we can use push\(^{-1}\)) we incorporate a projection operator that considers values in the \( m \times n \) grid that correspond to the same point in \( S \) and replaces them with their max. Similar to standard pooling layers, averaging corresponding values resulted in inferior results. We implemented two differentiable layers that correspond to these modifications, enabling end-to-end learning for \( f(\cdot, w) \). Figure 5 shows the new layers and their incorporation in the network’s architecture.
Convolutional Neural Networks on Surfaces via Seamless Toric Covers

We note that the max-projection layer mentioned above has certain resemblance to the TI-pooling operator introduced in [Laptev et al. 2016] which pools over corresponding pixels of multiple transformed versions of the input image and aim at learning transformation-invariant features. In contrast, our layer pools over corresponding points on the surface in order to get a single prediction at every point on the surface.

Data generation. To train the network, we first need to push the training data to images defined over the flat-torus $T$. Given training data $\{(S_i, x_i, y_i)\}_{i \in I}$, for each $i$ we sample $\rho$ triplets $P = (p_1, p_2, p_3) \subset S_i$ from $S_i \times S_i \times S_i$. Then for each $P$ we create a pair

$$(X_k, Y_k) = (\text{push}_P(x_i), \text{push}_P(y_i)), \quad (4)$$

where each pair corresponds to training input $X_k \in \mathbb{R}^{m \times n \times d}$ and output $Y_k \in \mathbb{R}^{m \times n \times d}$ directly compatible with $f(\cdot, w)$, and $k$ is an index for $|I| \times \rho$ such pairs. The choice of triplets follow the rationale of well-covering the surface to allow each point to be represented with a reasonable (=not too low) scale factor at-least in one map. Hence, we sampled a small number (5-20) of uniformly spread points (including the AGD local maxima [Kim et al. 2011]) on the surface and randomly sampled triplets from this set.

Training CNN on $T$. We use this data to train CNN by finding locally optimal parameters $w$ with respect to the following loss function:

$$E(w) = \sum_{k} \sigma(f(X_k, w), Y_k), \quad (5)$$

where $\sigma$ is the standard softmax loss per pixel, weighted by $1/(\delta + c)$; $c$ is the size of the pixel’s ground-truth class, and $\delta = 4000$ is a regularizer. We used Matconvnet [Vedaldi and Lenc 2015] for training using its SGD (Stochastic gradient descent) with learning rate of 0.0001 as in the original implementation of [Long et al. 2015].

We initialized the network with the parameters of FCN32, removing and/or duplicating channels to fit our data dimension.

Aggregating CNN output on $S$. Given a new surface $S$ and corresponding vector valued function $x$, we use the trained CNN to define the final predictor $F$ via:

$$F(x) = \sum_{P} (S(P) \odot \text{push}_{P}^{-1}\left(\text{push}_{P}(x), w\right)), \quad (6)$$

where $P$ is a triplet from a set of $\rho$ random triplets, $S(P)$ is a weight function chosen to compensate for the scale changes induced by the mapping $\Phi_P$, and $\odot$ is pointwise multiplication of functions. The weight function $S(P)$ is taken to be the scale factor of the mapping $\Phi_P$. It is defined at every vertex of the surface using a triangle-area weighted average of the adjacent triangles’ scale. Our aggregation method is motivated by the observation that the scale factor can serve as a confidence measure for the CNN prediction at each point of the surface.

Figure 6 depicts an aggregation example where the four left models show the contribution of four different triplets $P$ visualized using orange disks (gray color corresponds to points with low scale factor), and the model on the right is the final result.

4 PROPERTIES

In this section we justify the choice of the flat torus as the target domain and explain the translation invariance of the new convolution operators. Specifically, we show that the convolution operator on $S^4$ is invariant to a two dimensional group of conformal translations.

Convolution on the flat torus. We start by considering the Euclidean case, namely the flat torus $T$. A translation is an isometry $\tau_v : T \rightarrow T$ defined by $\tau_v(x) = x - v$. Translations act on functions over the torus $\tau_v : \mathcal{F}(T, \mathbb{R}) \rightarrow \mathcal{F}(T, \mathbb{R})$ via $\tau_v(f)(x) = f(\tau_v(x)) = f(x - v)$. Translation invariance of the convolution operator means it commutes with the translation operator as was just defined:

$$\tau_v(f * g) = \tau_v(f) * g$$

Conversely, under certain conditions, one can show that a linear and translation invariant operator is a convolution with some fixed function $g$ [Davies 2007]. Therefore, linearity and translation invariance are defining properties of convolution.

Translations on surfaces. To define a convolution operator on a surface \( S \), a natural requirement is that it would be linear and translation invariant. But what is a translation \( \tau : S \to S \)? In tune with the definition of a surface, a translation on a surface should locally be a Euclidean translation. That is, a flow along a non-vanishing vector field. According to the Poincaré-Hopf Theorem [Milnor 1965] the only surfaces with non-vanishing vector fields are of Euler characteristic zero - in case of closed orientable surfaces, the torus. This implies that the only surfaces on which we can define translations in the above mentioned way are toric.

The pullback convolution. Given two toric (not necessarily flat) surfaces \( T_1, T_2 \) and a homeomorphism \( \Phi : T_1 \to T_2 \) one can define a convolution operator \( *_1 \) on \( T_1 \) from a convolution operator \( *_2 \) defined on \( T_2 \) via

\[
    f *_1 g = \left( f \circ \Phi^{-1} \right) *_2 \left( g \circ \Phi^{-1} \right) \circ \Phi,
\]

The pullback convolution \( *_1 \) is linear and translation invariant w.r.t. the pullback translations \( \Phi^{-1} \circ \tau \circ \Phi \), where \( \tau \) represents translations in \( T_2 \) for which \( *_2 \) is invariant to. This is proved in the appendix.

In our case \( T_1 = S^4, T_2 = T \) the flat-torus with the Euclidean convolution and translations (modulo 1), and the mapping \( \Phi : S^4 \to T \) is a conformal homeomorphism. As the convolution on the flat-torus is invariant to Euclidean translations \( \tau \) of the type \( x \mapsto x - v(\text{mod } 1) \), the pullback convolution on \( S^4 \) is invariant to the pullback translations \( \Phi^{-1} \circ \tau \circ \Phi \). Since \( \Phi, \Phi^{-1}, \tau \) are all conformal maps, these pullback translations are conformal maps as well. To visualize an example of these translations consider Figure 1 (bottom left) and imagine that each square on the David surface moves to a well-defined "right" neighbor.

5 EVALUATION

In this section, we compare our method to alternative approaches for surface parameterization and network architecture. We compare to three other methods: The first two methods focus on the parameterization component of our algorithm, the last one on the architecture:

(i) GIM - where we used the geometry images parameterization method of [Sinha et al. 2016] followed by application of the FCN32 network [Long et al. 2015]. To the best of our knowledge this is the only parameterization method used with CNNs.

(ii) Tutte - we consider parameterization using Tutte’s embedding [Tutte 1963] to the unit square followed by application of the FCN32 network [Long et al. 2015]. To generate the embedding, we use the same cut as in our method and map the boundary in a length-preserving manner (up to global scale) to the boundary of the unit square. This is a natural selection for comparison, since Tutte’s embedding can be computed as efficiently as our method by solving a sparse system of linear equations.

(iii) Seamless+FCN - where we use our parameterization technique but without the two additional layers of cyclical padding and projection. This is equivalent to considering the flat-torus as a disk with its opposite sides disconnected.

We perform the evaluation on the task of segmenting the head in human models. Our training set for this task is composed of 370 models from the SCAPE [Anguelov et al. 2005], FAUST [Bogo et al. 2014] and MIT animation datasets [Vlasic et al. 2008]. The ground truth labels are head indicator functions that are manually labeled. Our test data are the 18 sphere-like human models from the SHREC dataset [Giorgi et al. 2007]. We use intersection-over-union as our evaluation metric: Denoting by \( GT_{\text{head}} \) the set of faces labeled “head” and by \( Alg_{\text{head}} \) the faces labeled “head” by the algorithm, this metric as defined as

\[
    \frac{|GT_{\text{head}} \cap Alg_{\text{head}}|}{|GT_{\text{head}} \cup Alg_{\text{head}}|}
\]

weighted by triangle areas. In all experiments, as input features (“easy” functions over the surfaces), we use a set of 26 basic shape features: 21 WKS features (equally spaced), 4 curvature features (max, min, arithmetic mean and geometric mean of the principal curvatures) and average geodesic distance (AGD) as input. We initialize training with the parameters obtained by the FCN32 net [Long et al. 2015]. We trained all networks with the same parameters and same number of epochs.

For all methods we considered \( p = 10 \) different parameterizations per mesh, resulting in a dataset with 3700 segmented images of
Convolutional Neural Networks on Surfaces via Seamless Toric Covers

Table 1. Evaluation of CNN on surfaces with different parameterization methods and network architectures.

<table>
<thead>
<tr>
<th>Intersection-over-union</th>
<th>1. Geometry images + FCN</th>
<th>0.625</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Tutte + FCN</td>
<td>0.567</td>
</tr>
<tr>
<td></td>
<td>3. Seamless + FCN</td>
<td>0.503</td>
</tr>
<tr>
<td></td>
<td>4. Ours</td>
<td>0.710</td>
</tr>
</tbody>
</table>

Table 2. Semantic segmentation results.

<table>
<thead>
<tr>
<th>Method</th>
<th>Features used</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ours</td>
<td>10 normals, Euclidean, curv.</td>
<td>81.6%</td>
</tr>
<tr>
<td>Guo</td>
<td>10 normals, Euclidean, curv.</td>
<td>43.6%</td>
</tr>
<tr>
<td>Ours</td>
<td>26 WKS, AGD, curv.</td>
<td>88.0%</td>
</tr>
<tr>
<td>Guo</td>
<td>26 WKS, AGD, curv.</td>
<td>76.0%</td>
</tr>
<tr>
<td>Guo</td>
<td>600 HKS, WKS, AGD, curv.</td>
<td>87.8%</td>
</tr>
</tbody>
</table>

6 APPLICATIONS

In this section we demonstrate the usefulness of our method for two applications: semantic segmentation and automatic landmark detection on anatomical surfaces.

6.1 Semantic segmentation of surfaces

We applied our algorithm to the task of semantic segmentation of human models. As training data we used 370 models from SCAPE, FAUST, MIT (excluding two classes which are not suitable for full-body segmentation), and Adobe Fuse [fus 2016]. All models are manually segmented into eight labels according to the labels in [Kalogerakis et al. 2010]. Our test set is again the 18 models from the SHREC07 dataset in human category (all sphere-like models). Note that, in contrast with previous works, the training set does not include any models from the SHREC07 dataset which we use solely for testing. Figure 8 shows examples from the training set.

We generated data from 300 triplets per model in the training set which resulted in approximately 110K segmented images. The networks converged after 30-50 epochs. We compared our method to the state-of-the-art CNN method for mesh segmentation [Guo et al. 2015]. Both methods were trained on the same training data. Gou et al. also use convolutional nets, but on per-triangle features reshaped to a square grid. These convolutions do not enable aggregating information from nearby regions on the surface, so to get global context Gou et al. need to leverage global features. The training set for Guo et al. was created by randomly sampling $\sim 360k$ triangles from the set of training models. We applied their method with different sets of training features (ranging from 10 to 600) and as shown in Table 2 their performance drops considerably as the number of training features decreases; accuracy is measured as ratio of correctly labeled triangles, weighted by triangle areas. In contrast, our method can learn features by leveraging convolution defined...
over the surface, demonstrating consistently better performance with the same number of features. In fact, our method, using only 26 features, modestly outperforms the method of Guo et al. with 600 features, see Table 2 (for evaluation we used $\rho = 260/480$ triplets for 26/10 features, respectively). Figure 9 shows several results obtained with our algorithm.

Figure 10 shows the segmentation accuracy for the above 26 feature experiment as a function of the number of triplets $\rho$ used for aggregation. Using 5 triplets already provides meaningful results, while 10 triplets are almost identical to the final result with $\sim 300$ triplets.

Figure 11 shows successful application of the human segmentation method to shapes from other classes: robot [Yob 2016], armadillo and four-legged [Giorgi et al. 2007]. Note that although the training set contains only standard human segmentation data (see Figure 8), the network still produces plausible segmentations on this very different set of models. This demonstrates the robustness and generalization power of our method.

The robot example also demonstrates a possible way to bypass the sphere topology restriction of our method: In this case the input surface contains multiple connected components and non-manifold edges. We approximated the input with a genus-0 surface using a simple reconstruction algorithm [Zhao et al. 2000] and applied our method to it. The result (shown on the approximated surface) is plausible.

6.2 Landmark detection on anatomical surfaces

Our algorithm can be applied to automatic landmark detection on 3D shapes in general and on biological data in particular. Here we show results of an experiment we conducted on models of animal teeth from the [Boyer et al. 2011] dataset. On each tooth 6 biologically significant landmarks where manually marked by experts. Our task here is to detect these landmarks on an unseen tooth. For this purpose we took 81 teeth from this dataset, converted them to sphere topology using [Ju 2004] and marked each landmark area using a geodesic disk of constant radius. We trained on a random subset of 73 teeth and tested on 8 models. For each tooth we generated 125 triplets, resulting in a training set of approximately 9K images.

For the input ‘easy’ function we only used curvature and (the logarithm of) the conformal scale factor. The network converged after 50 epoches.

Figure 12 shows the results we obtained on all eight test models. In a dashed rectangle we show the ground-truth labeling for the tooth in the middle of the first row. Although we used only a five-dimensional vector of basic features our method was successful in identifying most of the landmarks despite the local as-well as global variability in the data.

This dataset contains both right-side as-well as left-side teeth. The landmarks are therefore reflected when comparing right- and left-side teeth. Remarkably, our algorithm is able to correctly label landmarks on both right- and left-side teeth according to their biological meaning and is not ‘fooled’ by their orientation (see the first tooth in the second row).
We present average running times. Computing the parameterization was detected successfully. The failure in this case may be related (possibly due to the lack of the ridge on which the missing landmark is usually marked). In the last tooth on the right only one landmark was detected successfully. The failure in this case may be related to the fact that both the genus of the animal and the specific peak structures are not represented in the training set [Maier 1984]. This experiment further demonstrates the ability of our method to learn high-level semantic data from low-level information on the surface.

Using the output of our algorithm, we extract the point landmarks by computing the geodesic centroid of each label. Figure 13 shows a quantitative evaluation of the keypoints we extracted using the above method compared to ground truth. We plot the fraction of the predicted points (y-axis) that are within a certain geodesic error threshold of their true position (x-axis). We compared our algorithm to a baseline random forest classifier [Breiman 2001] (using matlab’s implementation) which was recently shown to be a successful classifier for shape analysis problems [Rodola et al. 2014]. The input per-face feature vectors consisted of the 600 WKS, HKS, curvature and AGD as before, with additional (logarithm of) the conformal scale factor generate with 125 triplet (generated as above). We tried two versions - the first with 50 trees and 73K sampled faces and the second with 100 trees and 292K sampled faces. Our algorithm was able to extract more accurate landmarks despite the large number and expressive power of the features fed to the baseline.

In Figure 14 we show a smooth bijective map between a pair of teeth obtained by interpolating the 6 landmarks identified by our method. The map was obtained using the method of [Aigerman and Lipman 2016]. This provides a fully automatic pipeline for producing semantically correct mappings between biological surfaces.

6.3 Timing
We present average running times. Computing the parameterization on a mesh with 12.5K vertices takes 0.862 seconds for a given triplet of cones. Computing the scale factor of the parameterization (used for aggregation) takes 0.534 seconds. The training can process about one image with 5-26 channels per second for a single GPU in Nvidia K80. We used three such dual GPUs which made the training 6 times faster. For a dataset containing 110k images (as in the full segmentation experiment) a single epoch takes about 5 hours. Feed-forward calculation in the network (using a single GPU of Nvidia K80) takes 0.35 seconds on average for a single image with 5-26 channels. Full prediction for a single triplet (feed-forward and pull-back of functions to the surface) takes about 2.94 seconds, and this process can be parallelized for multiple triplets. Consequently, in case of sequential runs, a prediction on a single model with 1/10/50 triples takes 3/30/150 seconds. Using 50 triples, it takes 45 minutes to calculate predictions on the human class of SHREC07 and 20 minutes on the teeth dataset. These experiments were done on an Intel Xeon E5 CPU with 64GB of RAM.

7 CONCLUSION
We presented a methodology and an algorithm for applying deep convolutional neural networks to geometric surfaces. The algorithm is based on seamless, conformal mapping of surfaces to the flat-torus on which convolution is well defined. Standard CNN architecture can then be used with minor modifications to perform supervised learning on the flat-torus. We demonstrated the usefulness of our approach for semantic segmentation and automatic landmark detection on anatomical surfaces, and showed it compares favorably to competing methods.

A limitation of our technique is that it assumes the input shape is a mesh with a sphere-like topology. An interesting direction for future work is extending our method to meshes with arbitrary topologies. This problem is especially interesting since in certain cases shapes from the same semantic class may have different genus. Another limitation is that currently aggregation is done as a separate post-process step and not as a part of the CNN optimization. An interesting future work in this regard is to incorporate the aggregation in the learning stage and produce end-to-end learning framework.

8 ACKNOWLEDGEMENTS
This research was supported in part by the European Research Council (ERC Starting Grant "Surf-Comp", Grant No. 307754), I-CORE program of the Israel PBC and ISF (Grant No. 4/11). The authors would like to thank Ayan Sinha and Karthik Ramani for sharing their code, Shahar Kovalsky and Julia Winchester for providing biological insights, and the anonymous reviewers for their useful comments and suggestions.

REFERENCES

JINGGUO WEI, Qinqing Huang, Duygu Ceylan, Etienne Vouga, and Hao Li. 2016. Dense Human Body Correspondences Using Convolutional Networks. In Proc. CVPR.

Appendix

We prove that given a bijection $\Phi : T_1 \to T_2$ and assuming $\tau(\circ \Phi) = \tau$ we have that $\tau(\circ \Phi) = \tau$. As before, we use the notation $\tau(f) = f \circ \tau$. First,

$$\tau(f \circ \Phi) = (f \circ \Phi) \circ \tau = (f \circ \Phi) \circ \tau \circ \Phi = (f \circ \Phi \circ \tau \circ \Phi) \circ \Phi$$

On the other hand we have

$$\tau(f \circ \Phi) = (f \circ \Phi) \circ \tau = (f \circ \Phi \circ \tau) \circ \Phi$$

where in the last equality we used the invariance of $\tau$ to $\tau$. We proved invariance of $\tau$ to $\tau$. We