Spherical Regression: Learning Viewpoints, Surface Normals and 3D Rotations on *n*-Spheres

Shuai Liao Efstratios Gavves Cees G. M. Snoek
QUVA Lab, University of Amsterdam

Abstract

Many computer vision challenges require continuous outputs, but tend to be solved by discrete classification. The reason is classification's natural containment within a probability n-simplex, as defined by the popular softmax activation function. Regular regression lacks such a closed geometry, leading to unstable training and convergence to suboptimal local minima. Starting from this insight we revisit regression in convolutional neural networks. We observe many continuous output problems in computer vision are naturally contained in closed geometrical manifolds, like the Euler angles in viewpoint estimation or the normals in surface normal estimation. A natural framework for posing such continuous output problems are n-spheres, which are naturally closed geometric manifolds defined in the $\mathbb{R}^{(n+1)}$ space. By introducing a spherical exponential mapping on n-spheres at the regression output, we obtain well-behaved gradients, leading to stable training. We show how our spherical regression can be utilized for several computer vision challenges, specifically viewpoint estimation, surface normal estimation and 3D rotation estimation. For all these problems our experiments demonstrate the benefit of spherical regression. All paper resources are available at https://github.com/leoshine/Spherical_Regression.

1. Introduction

Computer vision challenges requiring continuous outputs are abundant. Viewpoint estimation [28, 29, 33, 35], object tracking [12, 17, 18, 34], and surface normal estimation [1, 8, 30, 39] are just three examples. Despite the continuous nature of these problems, regression based solutions that seem a natural fit are not very popular. Instead, classification based approaches are more reliable in practice and, thus, dominate the literature [20, 24, 33, 34, 35]. This leads us to an interesting paradox: while several challenges are of continuous nature, their present-day solutions tend to be discrete.

In this work we start from this paradox and investigate why regression lags behind. When juxtaposing the mechanics of classification and regression we observe that classification is naturally contained within a probability n-simplex geometry defined by the popular softmax activation function. The gradients propagated backwards to the model are constrained and enable stable training and convergence. In contrast, regression is not contained by any closed geometry. Hence, the gradients propagated backwards are not constrained, potentially leading to unstable training or convergence to suboptimal local minima. Although classification solutions for continuous problems suffer from discretization

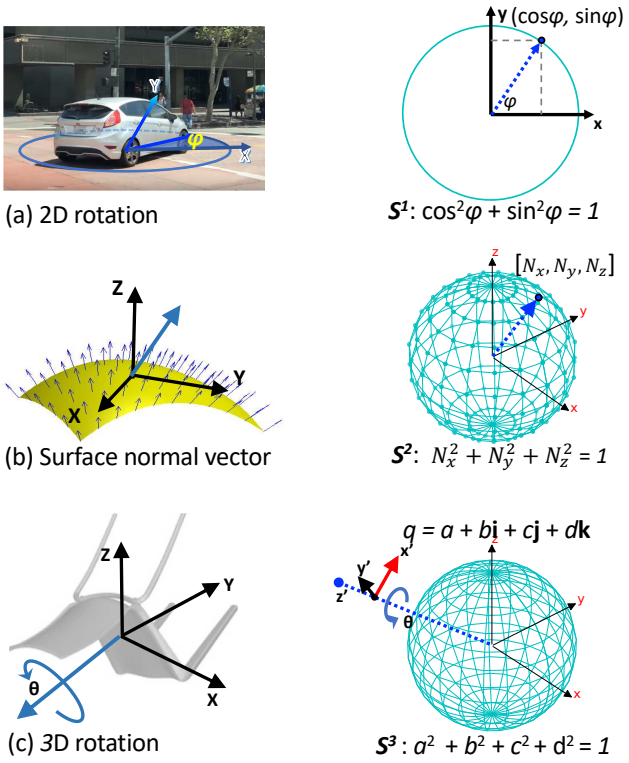


Figure 1. **Many computer vision problems can be converted into a n-sphere problem.** n-spheres are naturally closed geometric manifolds defined in the $\mathbb{R}^{(n+1)}$ space. Examples are a) viewpoint estimation, b) surface normal estimation, and c) 3D rotation estimation. This paper proposes a general regression framework that can be applied on all these n-sphere problems.

errors in annotations and predictions, they typically lead to more reliable learning [20, 24].

Founded on the relation between classification, regression and closed geometric manifolds, we revisit regression in deep networks. Specifically, we observe many continuous output problems in computer vision are naturally contained in closed geometrical manifolds defined by the problem at hand. For instance, in viewpoint estimation, angles cannot go beyond the $[-\pi, \pi]$ range. Or, in surface normal estimation the ℓ_2 norm of the surface normals must sum up to 1 to form unit vectors that indicate directionality. It turns out that a natural framework for posing such continuous output problems are the n-spheres S^n [7, 10], which are naturally closed geometric manifolds defined in the $\mathbb{R}^{(n+1)}$ space. We, therefore, rethink regression in continuous spaces in the context of n-spheres, when permitted by the application. It turns out that if we introduce a proposed spherical exponential mapping on n-spheres at the regression output we obtain regression gradients that are constrained and well-behaving, similar to classification-based learning. We refer to regression using the proposed spherical exponential mappings on S^n spheres as S^n spherical regression.

In this work we make the following contributions. First, we link the framework of n-spheres to continuous output computer vision tasks. By doing so, they are amenable to the properties of the n-spheres formulation, leading to spherical regression. Second, we propose a novel nonlinearity, the spherical exponential activation function, specifically designed for regressing on S^n spheres. We show the activation function improves the results obtained by regular regression. Third, we show how the general spherical regression framework can be utilized for particular computer vision challenges. Specifically, we show how to recast existing methods for viewpoint estimation, surface normal estimation and 3D rotation estimation to the proposed spherical regression framework. Our experiments demonstrate the benefit of spherical regression for these problems.

We now first describe in Section 2 the motivation behind the deep learning mechanics of classification and regression. Based on the insights derived, we describe in Section 3 the general framework for spherical regression on S^n spheres. We then explain how to specialize the general frameworks for particular applications, see Fig. 1. We describe the related work for these tasks in Section 4. In Section 5, we evaluate spherical regression for the three applications.

2. Motivation

Deep classification and regression networks. We start from an input image $\mathbf{x}$ of an object with a supervised learning task in mind, be it classification or regression. Regardless the task, if we use a convolutional neural network (CNN) we can split it into two subnetworks, the base network and the prediction head, see (eq. 1).

$$\underbrace{\mathbf{x} \xrightarrow[\text{base network}]{H(\cdot)} \boldsymbol{O} = \begin{bmatrix} o_0 \\ o_1 \\ \vdots \\ o_n \end{bmatrix}}_{\text{Base network}} \xrightarrow[\text{activation}]{g(\cdot)} \underbrace{\boldsymbol{P} = \begin{bmatrix} p_0 \\ p_1 \\ \vdots \\ p_n \end{bmatrix} \xleftrightarrow[\text{loss}]{\mathcal{L}(\cdot,\cdot)} \boldsymbol{Y}}_{\text{Prediction head}} \tag{1}$$

The base network considers all the layers from input $\mathbf{x}$ till layer $\boldsymbol{O}$. It defines a function $\boldsymbol{O} = H(\mathbf{x})$ that returns an intermediate latent embedding $\boldsymbol{O} = [o_0, o_1, ..., o_n]^\top$ of the raw input $\mathbf{x}$. The function comprises a cascade of convolutional layers intertwined with nonlinearities and followed by fully connected layers, $H = h_l \circ h_{l-1} \cdots \circ h_k \circ \cdots \circ h_2 \circ h_1$, where h_k is the θ-parameterized mapping of k-th layer. Given an arbitrary input signal $\mathbf{x}$, the latent representation $\boldsymbol{O}$ is unconstrained, namely $\mathbf{x} = H(\mathbf{x}) \rightarrow \mathbb{R}^{(n+1)}$.

The prediction head contains the last $(n+1)$-dimensional layer $\boldsymbol{P}$ before the loss function, which is typically referred to as the network output. The output is obtained from an activation function $g(\cdot)$, which generates the output $\boldsymbol{P} : p_k = g(o_k; \boldsymbol{O})$ using as input the intermediate raw embedding $\boldsymbol{O}$ returned by the base network. The activation function $g(\cdot)$ imposes a structure to the raw embdedding $\boldsymbol{O}$ according to the task at hand. For instance, for a CNN trained for image classification out of $1,000$ classes we have a $1,000$-dimensional output layer $\boldsymbol{P}$ that represents softmax probabilities. And, for a CNN trained for 2D viewpoint estimation we have a 2-dimensional output layer $\boldsymbol{P}$ that represents the trigonometric functions $\boldsymbol{P} = [cos\phi, sin\phi]$. After the prediction head lies the loss function $\mathcal{L}(\boldsymbol{P}, \boldsymbol{Y})$ that computes the distance between the prediction $\boldsymbol{P}$ and the ground truth $\boldsymbol{Y} = [y_0, y_1, ...]^\top$, be it cross entropy for classification or sum of squared errors for regression.

The dimensionalities of $\boldsymbol{O}$ and $\boldsymbol{P}$ vary according to the type of classification or regression that is considered. For classification $\boldsymbol{P}$ represents the probability of $(n+1)$ discretized bins. For regression, $\boldsymbol{P}$ depends on the assumed output representation dimensionality, *e.g.*, regression 1D [28], regression 2D [2, 28] or regression 3D [27] and beyond can have different output dimensions. Together the subnetworks comprise a standard deep architecture, which is trained end-to-end.

Training. During training, the k-th layer parameters are updated with stochastic gradient descent, $\theta_k \leftarrow \theta_k - \gamma \frac{\partial \mathcal{L}}{\partial \theta_k}$, where γ is the learning rate. Expanding by the chain rule of calculus we have that

$$\frac{\partial \mathcal{L}}{\partial \theta_k} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{P}} \frac{\partial \boldsymbol{P}}{\partial \boldsymbol{O}} \left(\frac{\partial \boldsymbol{O}}{\partial h_{l-1}} \cdots \frac{\partial h_{k+1}}{\partial h_k}\right) \frac{\partial h_k}{\partial \theta_k} \tag{2}$$

Training is stable and leads to consistent convergence when the gradients are constrained, otherwise gradient updates

may cause bouncing effects on the optimization landscape and may cancel each other out. Next, we examine the behavior of the output activation $\boldsymbol{P}$ and the loss functions for classification and regression.

Classification. For classification the standard output activation and loss functions are the softmax and the cross entropy, that is $g(o_i; \boldsymbol{O}) = \{p_i = e^{o_i} / \sum_j e^{o_j}, i = 0 \cdots n\}, \mathcal{L}(\boldsymbol{O}, \boldsymbol{Y}) = -\sum_i y_i log(p_i)$. The p_i and y_i are the posterior probability and the one-hot vector for the i-th class, and d is the number of classes. Note that softmax maps the raw latent embedding $O \in \mathbb{R}^{(n+1)}$ to a structured output $\boldsymbol{P}$, known as n-simplex, where each dimension is positive and the sum equals to one, *i.e.* $\sum_i p_i = 1$ and $p_i > 0$. The partial derivative of the probability output with respect to the latent activation equals to

$$\frac{\partial p_j}{\partial o_i} = \begin{cases} p_j \cdot (1 - p_j), & \text{when } j = i \\ -p_i \cdot p_j, & \text{when } j \neq i \end{cases} \tag{3}$$

Crucially, we observe that the partial derivative $\frac{\partial p_j}{\partial o_i}$ does not directly depend on $\boldsymbol{O}$. This leads the partial derivative of the loss function with respect to o_i, namely

$$\frac{\partial \mathcal{L}}{\partial o_i} = -\sum_k \frac{y_k}{p_k} \cdot \frac{\partial p_k}{\partial o_i} = p_i - y_i, \tag{4}$$

to be independent of $\boldsymbol{O}$ itself. As $\boldsymbol{P}$ corresponds to a probability distribution that lies inside the n-dimensional simplex, it is naturally constrained by its ℓ_1 norm, $p_j < 1$. Thus, the partial derivative $\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}$ depends only on a quantity that is already constrained.

Regression. In regression usually there is no explicit activation function in the final layer to enforce some manifold structure. Instead, the raw latent embedding $\boldsymbol{O}$ is directly compared with the ground truth. Take the smooth-L1 loss as an example,

$$\mathcal{L} = \begin{cases} 0.5|y_i - o_i|^2 & \text{if} |y_i - o_i| \leqslant 1 \\ |y_i - o_i| - 0.5 & \text{otherwise.} \end{cases} \tag{5}$$

The partial derivative of the loss with respect to o_i equals to

$$\frac{\partial \mathcal{L}}{\partial o_i} = \begin{cases} -(y_i - o_i) & \text{if} |y_i - o_i| \leqslant 1 \\ -\text{sign}(y_i - o_i) & \text{otherwise.} \end{cases} \tag{6}$$

Unlike classification, where the partial derivatives are constrained, for regression we observe that the $\frac{\partial \mathcal{L}}{\partial o_i}$ directly depends on the raw output $\boldsymbol{O}$. Hence, if $\boldsymbol{O}$ has high variance, the unconstrained gradient will have a high variance as well. Because of the unconstrained gradients training may be unstable.

Conclusion. Classification with neural networks leads to stable training and convergence. The reason is that the partial derivatives $\frac{\partial \mathcal{L}}{\partial \boldsymbol{P}} \cdot \frac{\partial \boldsymbol{P}}{\partial \boldsymbol{O}}$ is constrained, and, therefore, the gradient updates $\frac{\partial \mathcal{L}}{\partial \theta_k}$, are constrained. The gradients are constrained because the output $\boldsymbol{P}$ itself is constrained by the ℓ_1 norm of the n-simplex, $\sum_i p_i = 1$. Regression with neural networks may have instabilities and sub-optimal results during training because gradient updates are unconstrained. We examine next how we can define a similar closed geometrical manifold also for regression. Specifically, we focus on regression problems where the target label Y lives in a constrained n-sphere manifold.

3. Spherical regression

The n-sphere, denoted with S^n, is the surface boundary of an $(n+1)$-dimensional ball in the Euclidean space. Mathematically, the n-sphere is defined as $S^n = \left\{\mathbf{x} \in \mathbb{R}^{n+1} : \|\mathbf{x}\| = r\right\}$ and is constrained by the ℓ_2 norm, namely $\sum_i x_i^2 = 1$. Fig. 1 gives examples of simple n-spheres, where S^1 is the circle and S^2 the surface of a 3D ball. Where the n-simplex constrains classification by the ℓ_1 simplex norm, we next present how to constrain regression by the ℓ_2 norm of an n-sphere.

3.1. Constraining regression with n-spheres

To encourage stability in training regression neural networks on S^n spheres, one reasonable objective is to ensure the gradients are constrained. To constrain the gradient $\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}$, we propose to insert an additional activation function in regression after the raw embedding layer $\boldsymbol{O}$. The activation function should have the following properties.

I The *output* of the activation, $\boldsymbol{P} = \{p_k\}$, must live on n-sphere, namely its ℓ_2 norm $\sum_{k=1} p_k^2 = 1$ must be constant, *e.g.*, $cos^2\phi + sin^2\phi = 1$. This is necessary for spherical targets.

II Similar to classification, the *gradient* $\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}$ must not directly depend on the input signal. That is, $\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}$ must not depend directly on the raw latent embedding $\boldsymbol{O} \in \mathbb{R}^{(n+1)}$.

To satisfy property I, we pick our activation function such that it produces normalized values. We opt for the ℓ_2 normalization form: $p_j = g(o_j; \boldsymbol{O}) = \frac{f(o_j)}{\sqrt{\sum_k f(o_k)^2}}$, where $f(\cdot)$ corresponds to any univariate mapping. The partial derivative of the output with respect to the latent O then becomes:

$$\begin{aligned} \frac{\partial p_j}{\partial o_i} &= \frac{\partial \left[\frac{f(o_j)}{\sqrt{\sum_k f(o_k)^2}} \right]}{\partial o_i} \\ &= \begin{cases} \left(\frac{df(o_i)}{do_i} \cdot \frac{1}{A} \right) \cdot (1 - {p_i}^2), & \text{when } j = i \\ \left(\frac{df(o_i)}{do_i} \cdot \frac{1}{A} \right) \cdot (-p_i \cdot p_j), & \text{when } j \neq i \end{cases} \end{aligned} \tag{7}$$

where $A = \sqrt{\sum_k f(o_k)^2}$ is the normalization factor.

Still, $\frac{\partial p_j}{\partial o_i}$ is potentially depending on the raw latent embedding $\boldsymbol{O}$ through the partial function derivatives $\frac{df(o_j)}{do_i}$ and the normalization factor A. To satisfy property II and make $\frac{\partial p_i}{\partial o_j}$ independent from the raw output $\boldsymbol{O}$, and thus constrained, we must make sure that $\left(\frac{df(o_j)}{do_j} \cdot \frac{1}{A}\right)$ becomes independent of $\boldsymbol{O}$. In practice, there are a limited number of choices for $f(\cdot)$ to satisfy this constraint. Inspired by the softmax activation function, we resort to the exponential map $f(o_i) = e^{o_i}$, where $\frac{df(o_i)}{do_i} = f(o_i)$ and $\frac{\partial f(o_i)}{\partial o_i} \cdot \frac{1}{A} = \frac{f(o_i)}{A} = p_i$. Thus Eq. 7 is simplified as

$$\frac{\partial p_j}{\partial o_i} = \begin{cases} p_i \cdot (1 - p_i^2), & \text{when } j = i \\ -p_i^2 \cdot p_j, & \text{when } j \neq i \end{cases} \tag{8}$$

removing all dependency on $\boldsymbol{O}$.

Since our activation function has a similar form as softmax, which is also known as normalized exponential function, we refer to our activation function as *Spherical Exponential Function*. It maps inputs from $\mathbb{R}^{n+1}$ to the positive domain of the n-Sphere, *i.e.* $S_{exp}(\cdot) : \mathbb{R}^{n+1} \rightarrow \mathbb{S}^n_+$:

$$p_j = S_{exp}(o_j; \boldsymbol{O}) = \frac{e^{o_j}}{\sqrt{\sum_k (e^{o_k})^2}} \tag{9}$$

Converting Eq. 8 into matrix provides Jacobian as $\mathbf{J}_{S_{exp}} = (\boldsymbol{I} - \boldsymbol{P} \otimes \boldsymbol{P}) \cdot diag(\boldsymbol{P})$ where $\otimes$ denotes outer product (see supplementary material for details). Notice that if we only do ℓ_2 normalization without exponential, the Jacobian is given as $\mathbf{J}_{\mathcal{S}_{flat}} = (\boldsymbol{I} - \boldsymbol{P} \otimes \boldsymbol{P}) \cdot \frac{1}{||\boldsymbol{O}||}$, which is influenced by the magnitude of $\boldsymbol{O}$ in gradient, which is unconstrained.

Unfortunately, the exponential map in $S_{exp}(\cdot)$ restricts the output to be in the positive range only, whereas our target can be either positive or negative. To enable regression on the full range on n-sphere coordinates we rewrite each dimension into two parts: $p_i = sign(p_i) \cdot |p_i|$. We then use the output from the spherical exponential function to learn the absolute values $|p_i|, i = 0, 1, ..., n$ only. At the same time, we rely on a separate classification branch to predict the sign values, $sign(p_i), i = 1, ..., d$ of the output. The overall network is shown in Fig. 2:

Conclusion. Given the spherical exponential mapping for $g(\cdot)$, the gradient $\frac{\partial \boldsymbol{P}}{\partial \boldsymbol{O}}$ is detached from $\boldsymbol{O}$, and $\boldsymbol{P}$ is constrained by the n-Sphere. Thus, to make the parameter gradients also constrained, we just need to pick a suitable loss function. It turns out that there are no significant constraints for the loss function. Given ground truth $\boldsymbol{Y}$, we can set the loss to be the negative dot product $\mathcal{L} = -\langle |\boldsymbol{P}|, |\boldsymbol{Y}| \rangle$. Since both $\boldsymbol{P}$ and $\boldsymbol{Y}$ are on sphere with ℓ_2 norm equal to 1 (*i.e.* $||\boldsymbol{P}||_2 = ||\boldsymbol{Y}||_2 = 1$), this is equivalent to optimize with cosine proximity loss or L2 loss [1]. In this case, the gradients

[1]For cosine proximity loss: $\mathcal{L} = -\frac{\langle |\boldsymbol{P}|, |\boldsymbol{Y}| \rangle}{||\boldsymbol{P}||_2 \cdot ||\boldsymbol{Y}||_2} = -\langle |\boldsymbol{P}|, |\boldsymbol{Y}| \rangle$.

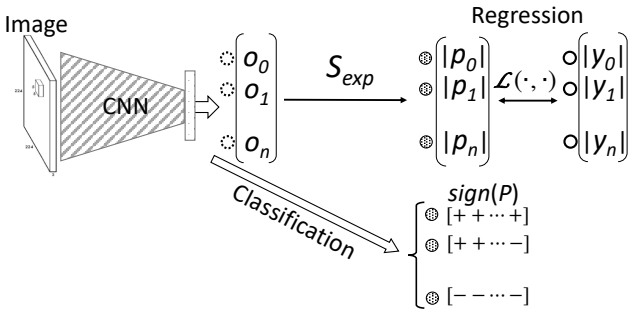


Figure 2. **Regressing on n-spheres** with targets $Y = [y_0, ..., y_n]$, *i.e.* $\sum_i y_i^2 = 1$. The model processes the input image and first returns a raw latent embedding $\boldsymbol{O} = [o_0, ..., o_n] \in \mathbb{R}^{(n+1)}$. Then, a regression branch using the proposed spherical exponential acctivation S_{exp} maps $\boldsymbol{O}$ to a structured output $|\boldsymbol{P}| = [|p_0|, ..., |p_n|]$. A classification branch is also used to learn the sign labels of $\boldsymbol{P}$. Prediction is made by $\boldsymbol{P} = sign(\boldsymbol{P}) \cdot |\boldsymbol{P}|$.

are $\frac{\partial \mathcal{L}}{\partial p_i} = -sign(p_i)|y_i|$ and only relate to $\boldsymbol{P}$. We could also treat the individual outputs $\{p_1^2, p_2^2 ...\}$ as probabilities with a cross-entropy loss on continuous labels y_i^2, in which case we would have that $H(\boldsymbol{Y}^2, \boldsymbol{P}^2) = \sum_i y_i^2 \log \frac{1}{p_i^2}$. We conclude that the Spherical Regression using the spherical exponential mapping allows for constrained parameter updates and, thus, we expect it to lead to stable training and convergence. We verify this experimentally on three different applications and datasets.

3.2. Specializing to S^1, S^2 and S^3

Next, we show how to specialize the general n-sphere formulation for different regression applications that reside on specific n-spheres.

S^1 case: Euler angles estimation. Euler angles are used to describe the orientation of a rigid body with respect to a fixed coordinate system. They are defined by 3 angles, describing 3 consecutive rotations around fixed axes. Specifically, each of the angles $\phi \in [0, 2\pi]$ can be represented by a point on a unit circle with 2D coordinate $[cos\phi, sin\phi]$, see Fig. 1. Since $cos^2\phi + sin^2\phi = 1$, estimating these coordinates is an S^1 sphere problem. Consequently, our prediction head has two components: *i)* a regression branch with spherical exponential activations for absolute values $|\boldsymbol{P}| = [|cos\phi|, |sin\phi|]$ and, *ii)* a classification branch to learn all possible sign combinations between $sign(cos\phi)$ and $sign(sin\phi)$, that is a 4-class classification problem: $sign(\boldsymbol{P}) \in \{(+,+), (+,-), (-,+), (-,-)\}$. We could also predict the signs independently and have fewer possible outputs, however, this would deprive the classifier from the opportunity to learn possible correlations.

During training time, we jointly minimize the regression loss (cosine proximity) and the sign classification loss

For L2 loss: $\mathcal{L} = ||\boldsymbol{P} - \boldsymbol{Y}||_2^2 = ||\boldsymbol{P}||_2^2 + ||\boldsymbol{Y}||_2^2 - 2\langle |\boldsymbol{P}|, |\boldsymbol{Y}| \rangle = 2 - 2\langle |\boldsymbol{P}|, |\boldsymbol{Y}| \rangle$.

(cross-entropy). For the inference, we do the final prediction by merging the absolute values and sign labels together:

$$\begin{cases} cos\phi = sign(cos\phi) \cdot |cos\phi| \\ sin\phi = sign(sin\phi) \cdot |sin\phi| \end{cases} \tag{10}$$

Beyond Euler angles, other 2D rotations can be learned in the same fashion.

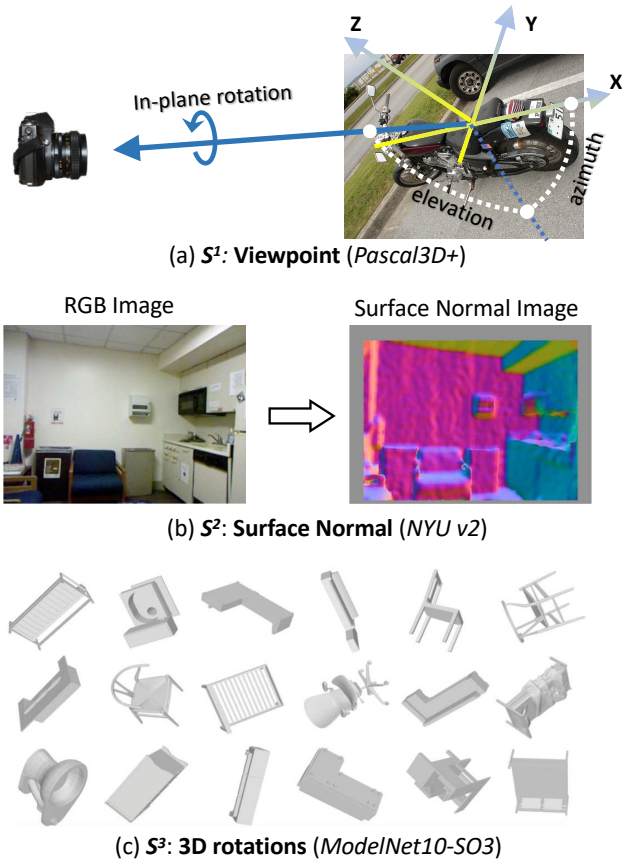
S^2 case: Surface normal estimation. A surface normal is the direction that is perpendicular to the tangent plane of the point on the surface of objects in a 3D scene, see Fig. 1.(b). It can be represented by a unit 3D vector $\mathbf{v} = [N_x, N_y, N_z]$ for which $N_x^2 + N_y^2 + N_z^2 = 1$. Thus, a surface normal lies on the surface of a unit 3D ball, *i.e.* an S^2 sphere. Surface normal estimation from RGB images makes pixel-wise predictions of surface normals of the input scene.

It is worth noticing that all surface normals computed by a 2D image should always be pointing outwards from the image plane, that is $N_z < 0$, since only these surfaces are visible to the camera. This halves the prediction space to a semi-sphere of S^2. Again, when designing the spherical regressor for surface normals, we have a regression branch to learn the absolute normal values $[|N_x|, |N_y|, |N_z|]$ and a classification branch for learning all combinations of signs for N_x and N_y. The total number of possible sign classes is 4, similar to Euler angle estimation. The training and inference is similar to Euler angles as well. Other S^2 problems include learning the direction of motion in 2D/3D flow fields, geographical locations on the Earth sphere and so on.

S^3 case: 3D rotation estimation. Rotational transformations are relevant in many computer vision tasks, for example, orientation estimation, generalized viewpoint and pose estimation beyond Euler angles or camera relocation. Rotational transformations can be expressed as orthogonal matrices of size n with determinant +1 (rotation matrices). We can think of the set of all possible rotation matrices to form a group that acts as an operator on vectors. This group is better known as the *special orthogonal Lie group* $\mathcal{SO}(n)$ [14]. Specifically, the $\mathcal{SO}(2)$ represents the set of all 2D rotation transformations, whereas $\mathcal{SO}(3)$ represents the set of all possible 3D rotations.

We have already shown that 2D rotations can be mapped to a regression on an S^1 sphere, thus the set $\mathcal{SO}(2)$ of all 2D rotations is topologically equivalent to the S^1 sphere. Interestingly, the topology of 3D rotations is not as straightforward [14], namely there is no n-sphere that is equivalent to $\mathcal{SO}(3)$. Instead, as shown in Fig. 1.(c) a 3D rotation $\mathcal{SO}(3)$ can be thought of as first choosing a rotation axis $\mathbf{v}$ and then rotating by an angle θ. This approach leads to the well known S^3 representation of quaternions [15], which is the closest equivalent to the 3D rotation [31].

A unit quaternion is equal to $q = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, where $a^2 + b^2 + c^2 + d^2 = 1$. As q and $-q$ give the same rotation, we restrict ourselves to $a > 0$, which again halves the output space. We, therefore, need to predict the signs of only 3 imaginary components $\{b, c, d\}$ to a total of 8 (2^3) classes. The design of the prediction heads and the loss functions are similar to the case of surface normal prediction on S^2, only now having 8 sign classes. Given the axis-angle representation $(\theta, \mathbf{v})$ of $\mathcal{SO}(3)$, we can, therefore, rewrite a quaternion into $q = (cos\frac{\theta}{2}, sin\frac{\theta}{2}\mathbf{v})$. Constraining $a > 0$ is equivalent to restricting the rotation angle $\theta \in [0, \pi]$. Furthermore, predicting the 8 sign categories is equivalent to predicting to which of the 8 quadrants of the 3D rotation space the $\mathbf{v}$ belongs.



(a) ***S¹*: Viewpoint** (*Pascal3D+*)

(b) ***S²*: Surface Normal** (*NYU v2*)

(c) ***S³*: 3D rotations** (*ModelNet10-SO3*)

Figure 3. **We assess spherical regression on 3 computer vision tasks.** (a) S^1: Viewpoint estimation on Pascal3D+ [38], which needs to predict 3 Euler angles: azimuth, elevation and in-plane rotation. (b) S^2: Surface normal estimation on NYU v2 [32], where pixel-wised dense surface normal prediction is required. (c) S^3: 3D rotation on our newly proposed ModelNet10-SO3, where given one rendered view of a CAD model, we predict the underlying 3D rotation that aligns it back to standard pose.

4. Related work

Viewpoint Estimation. In general, viewpoint estimation focuses on recovering the 3 Euler angles, namely, azimuth, elevation and in-plane rotation (see Fig. 3-(a)). Tulsiani

and Malik [35] discretize continuous Euler angles into multiple bins and convert viewpoint estimation into a classification problem. Su *et al*. [33] propose a finer-grained discretization that divides the Euler angles into 360 bins. However, training for all possible outputs requires an enormous amount of examples that can only be addressed by synthetic renderings.

Albeit more natural, regression-based viewpoint estimation is less popular. Because of the periodical nature of angles, most approaches do not regress directly on the linear space of angles, $a, e, t \in [-\pi, \pi]$. The reason is that ignoring the angle periodicity leads to bad modeling, as the 1° and 359° angles are assumed to be the furthest apart. Instead, trigonometric representations are preferred, with [2, 28, 29] proposing to represent angles by $[cos\phi, sin\phi]$. They then learn a regression function $h : x \mapsto [cos\phi, sin\phi]$, without, however, enforcing the vectors to lie on S^1. In comparison to viewpoint classification, regression gives continuous and fine-grained angles. In practice, however, training regression for viewpoint estimation is not as easy. Complex loss functions are typically crafted, *e.g*., smooth L_1 loss [24], without reaching the accuracy levels of classification-based alternatives.

In this paper, we continue the line of work on regression based viewpoint estimation. Built upon the S^1 representations $[cos\phi, sin\phi]$ of Euler angles [2, 28, 29], we assess our spherical regression for viewpoint prediction.

Surface Normal Estimation. Surface normal estimation is typically viewed as a 2.5D representation problem, one that carries information for the geometry of the scene, including layout, shape and even depth. The surface normal is a 3-dim vector that points outside the tangent plane of the surface. In the surface normal estimation task, given an image of a scene, a pixel-wise prediction of the surface normal is required [1, 8, 11, 21, 30, 32, 36, 39] (see Fig. 3-(b)).

Fouhey *et al*. [11] infer the surface normal by discovering discriminative and geometrically 3D primitives from 2D images. Building on contextual and segment-based cues, Ladicky *et al*. [21] build their surface normal regressor from local image features. They both use hand crafted features. Eigen and Fergus [8] propose a multi-scale CNN architecture adapted to predicting depth, surface normals and semantic labels. While the network outputs are ℓ_2 normalized, the gradients are not constrained. Bansal *et al*. [1] introduce a skip-network model optimized by the standard sum of squared errors regression loss, without enforcing any structure to the output. Zhang *et al*. [39] propose to predict normals with deconvolution layers and rely on large scale synthetic data for training. Similar to [8], they also enforce an ℓ_2 norm on the output but have unconstrained gradients. Recently, Qi *et al*. [30] proposed two-stream CNNs that jointly predict depth and surface normals from a single image and also rely on the sum of squared errors loss for training.

In our work we propose a spherical exponential mapping for performing spherical regression. This new mapping can be directly applied to any of the surface normal estimation methods that rely on a regression loss on n-spheres and improve their accuracy, as we show in the experiments.

3D Rotation Estimation. 3D Rotations are a component of several tasks in computer vision and robotics, including viewpoint and pose estimation or camera relocation. The rotation matrix for 3D rotation is a 3×3 orthogonal matrix (determinant$= 1$). Direct regression on the rotation matrix via neural networks is difficult, as the output lies in the $\mathbb{R}^9$ (3×3) space. Moreover, regressing a rotation matrix directly cannot guarantee its orthogonality. Recently, Falorsi *et al*. [9] take a first step toward regressing 3D rotation matrices. Instead of predicting the 9 elements of rotation matrix directly, they pose the 3D rotation as an $S^2 \times S^2$ representation problem reducing the number of elements to regress on to a total of 6.

Viewpoint [2, 5, 24, 24, 26, 33, 35] and pose [27, 28] consider the relative 3D rotation between object and camera. With 3 consecutive rotation angles, see Fig. 3 (a), *Euler Angles* can uniquely recover the rotation matrix. As such a decomposition is easy to be interpreted and able to cover most of the viewpoint distribution, it has been widely adopted. However, this approach leads to the gimbal lock problem [16], where the degrees of freedom for the rotations are reduced.

Mahendran *et al*. [22] studied an axis-angle representation for viewpoint estimation by first choosing a rotation axis and then rotating along it by an angle θ. To constrain the angle $\theta \in [0, \pi)$ and the axis $v_i \in [-1, 1]$, they propose a $\pi \cdot tanh$ non-linearity. Also, instead of a standard regression loss, *e.g*. cosine proximity or sum of squared errors loss, they propose a geodesic loss which directly optimizes the 3D rotations in $\mathcal{SO}(3)$. Do *et al*. [6] consider the Lie-algebra $\mathcal{SO}(3)$ representation to learn the 3D rotation of the 6 DoF pose of an object. It is represented as $[x, y, z] \in R^3$, and can be mapped to a rotation matrix via the Rodrigues rotation formula [3]. They conclude that an ℓ_1 regression loss yields better results.

Last, both Kendall *et al*. [19] and Mahendran *et al*. [22] consider quaternion for camera re-localization and viewpoint estimation. As quaternions allow for easy interpolation and computations on the S^3 sphere, they are also widely used in graphics [4,31] and robotics [25]. Although Do *et al*. [6] argue that quaternion is over-parameterized, we see this as an advantage that gives us more freedom to learn rotations directly on the n-sphere.

Despite the elegance and completeness of the aforementioned works, modelling 3D rotations is hard and methods specialized for the task at hand, instead, typically reach better accuracies. Unlike most of the aforementioned works,

Table 1. S^1**: Viewpoint estimation with Euler angles.** Comparison with state-of-the-art on Pascal3D+. Adding our S^1_{exp} spherical regression on top of the backbone network of [28] leads to best accuracy. We report a class-wise comparison in supplementary.

	MedErr↓	**Acc@** $\frac{\pi}{6}$↑
Mahendran *et al*. [22]	16.6	N/A
Tulsiani and Malik [35]	13.6	80.8
Mousavian *et al*. [26]	11.1	81.0
Su *et al*. [33]	11.7	82.0
Penedones *et al*. [28]†	11.6	83.6
Prokudin *et al*. [29]	12.2	83.8
Grabner *et al*. [13]	10.9	83.9
Mahendran *et al*. [23]	10.1	85.9
This paper: [28]†+ S^1_{exp}	**9.2**	**88.2**

† Based on our implementation.

we learn to regress on the Euclidean space directly. Furthermore, we present a framework for regressing on n-spheres with constrained gradients, leading to more stable training and good accuracy, as we show experimentally.

5. Experiments

5.1. S^1: Viewpoint estimation with Euler angles

Setup. First, we evaluate spherical regression on S^1 viewpoint estimation on Pascal3D+ [38]. Pascal3D+ contains 12 rigid object categories with bounding boxes and noisy rotation matrix annotations, obtained after manually aligning 3D models to the 2D object in the image. We follow [23, 26, 29, 33, 35] and estimate the 3 Euler angles, namely the azimuth, elevation and in-plane rotation, given the ground truth object location. A viewpoint prediction is correct when the geodesic distance $\Delta(R_{gt}, R_{pr}) = \frac{||log R_{gt}^T R_{pr}||_{\mathcal{F}}}{\sqrt{2}}$ between the predicted rotation matrix R_{pr} (constructed from the predicted Euler angles) and the ground truth rotation matrix R_{gt} is smaller than a threshold θ [35]. The evaluation metric is the accuracy $Acc@\pi/6$ given threshold $\theta = \pi/6$. We use ResNet101 as our backbone architecture, with a wider penultimate fully connected layer in the prediction head that is shared by the regression branch and classification branch (see supplementary material for details). As many of the annotations are concentrated around the x-axis, we found that rotating all annotations by 45° during training (and rotating back at test time) leads to more balanced distribution of annotations and better learning. For training data, we also use the synthetic data provided by [33], without additional data augmentations like in [22, 23].

Results. We report comparisons with the state-of-the-art in Table 1. Note that our spherical exponential mapping can be easily used by any of the regression-based methods with S^1 representation $[cos\phi, sin\phi]$ [2, 28]. In this experiment we combine it with Penedones *et al*. [28], who tried to directly regress 2D representation [$cos\phi$, $sin\phi$] of angels, obtaining a significant improvement in accuracy over other regression and classification baselines. That said, during experiments we observed that classification-based methods are more amenable to large data sets, most probably because of their increased number of parameters. As expected, the continuous outputs by the spherical regression are better suited for finer and finer evaluations , that is $Acc@\pi/12$ and $Acc@\pi/24$ (supplementary material). We conclude that spherical regression is successful for viewpoint estimation with Euler angles.

Table 2. S^2**: Surface normal estimation** Comparison with state-of-the-art on NYU v2. Adding our S^2_{exp} spherical regression on top of the backbone network of Zhang *et al*. [39] leads to best accuracy.

	Mean↓	**Median**↓	11.25°↑	22.5°↑	30.0°↑
Fouhey *et al*. [11] §	37.7	34.1	14.0	32.7	44.1
Ladicky *et al*. [21] §	35.5	25.5	24.0	45.6	55.9
Wang *et al*. [36] §	28.8	17.9	35.2	57.1	65.5
Eigen and Fergus [8]	22.3	15.3	38.6	64.0	73.9
Zhang *et al*. [39]	21.7	14.8	39.4	66.3	76.1
This paper: [39] + S^2_{exp}	**19.7**	**12.5**	**45.8**	**72.1**	**80.6**

§ Copied from [8].

5.2. S^2: Surface normal estimation

Setup. Next, we evaluate spherical regression for S^2 surface normal estimation on the NYU Depth v2 [32]. The NYU Depth v2 dataset contains 1,449 video frames of indoor scenes associated with Microsoft Kinect depth data. We use the ground truth surface normals provided by [32]. We consider all valid pixels across the whole test set during evaluation [39]. The evaluation metrics are the (*Mean* and *Median*), as well as the accuracy based metric, namely the percentage of correct predictions at given threshold $11.24°$, $22.5°$ and $30°$). We implement our S^2_{exp} spherical regression based on the network proposed by Zhang *et al*. [39], which is built on top of VGG-16 convolutional layers, and a symmetric stack of deconvolution layers with skip connections for decoding. As in viewpoint estimation, we also rotate the ground truth around the z-axis by 45° to yield better results. We follow the same training setup as [39], that is we first pre-train on the selected 568K synthetic data provided by [39] for 8 epochs, and fine-tune on NYU v2 for 60 epochs.

Results. We report results in Table 2. Replacing regular regression in [39] with spherical regression on S^2 improves the estimation of the surface normals considerably. We found the improvement is attentuated by the fact that for surface normal estimation we perform one regression

Table 3. S^3**: 3D Rotation estimation with quaternions.** Comparison on newly established ModelNet10-SO3. Adding our S^3_{exp} spherical regression on top of an AlexNet or VGG16 backbone network leads to best accuracy.

	MedErr↓	Acc@$\frac{\pi}{6}$↑	Acc@$\frac{\pi}{12}$↑	Acc@$\frac{\pi}{24}$↑
AlexNet (Direct+smooth-L1)	46.1	32.5	11.2	2.5
AlexNet + S_{flat}	33.3	53.5	34.1	13.9
AlexNet + S^3_{exp}	**25.3**	**65.4**	**48.5**	**24.4**
VGG16 (Direct+smooth-L1)	36.8	46.7	29.4	13.4
VGG16 + S_{flat}	25.9	63.5	48.7	29.5
VGG16 + S^3_{exp}	**20.3**	**70.9**	**58.9**	**38.4**

per pixel location. As each one of these regressions could return unstable gradients, bounding the total sum of losses with spherical regression is beneficial. Especially for the finer regression thresholds of $11.25°, 22.5°$. We conclude that spherical regression is successful also for surface normal estimation.

5.3. S^3: 3D Rotation estimation with quaternions

Setup. Last, we evaluate S^3_{exp} spherical regression on 3D rotation estimation on S^3 with quaternions. For this evaluation we introduce a new dataset, *ModelNet10-SO3*, composed of images of 3D synthetic renderings. ModelNet10-SO3 is based on *ModelNet10* [37], which contains 4,899 instances from 10 categories of 3D CAD models. In ModelNet10 the purpose is the classification of 3D shapes to one of the permissible CAD object categories. With ModelNet10-SO3 we have a different purpose, we want to evaluate 3D shape alignment by predicting its 3D rotation matrix w.r.t. the reference position from single image. We construct ModelNet10-SO3 by uniformly sampling per CAD model 100 3D rotations on $\mathcal{SO}(3)$ for the training set and 4 3D rotations for the test set. We render each view with white background, thus the foreground shows the rotated rendering only. We show some examples in Fig. 3-(c).

Relying on Euler angles for ModelNet10-SO3 is not advised because of the Gimbal lock problem [16]. Instead, alignment is possible only by predicting the quaternion representation of the 3D rotation matrix. For this task, we test the following 3 regression strategies:

(I) Direct regression with smooth-L1 loss. It may cause the output to no longer follow unit ℓ_2 norm.

(II) Regression with ℓ_2 normalization $\mathcal{S}_{flat}$.

(III) Regression with $\mathcal{S}_{exp}$ (this paper).

We report results based on AlexNet and VGG16 as our CNN backbones, with a class-specific prediction head. We borrow the evaluation metric from viewpoint estimation, namely *MedErr* and $Acc@\{\pi/6, \pi/12, \pi/24\}$ so that we also examine finer-grained predictions.

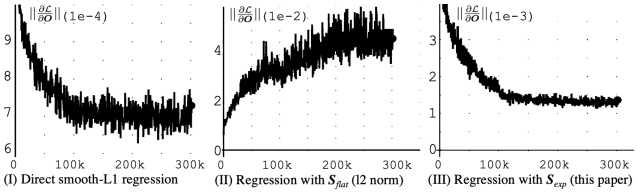


Figure 4. **Variance of the average gradient norm** $||\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}||$**.** Spherical exponentiation $\mathcal{S}_{exp}$ yields lower variance on mini-batch over entire train progress.

Results. We report results in Table 3. First, both S_{flat} and S^3_{exp} regression on quaternions improve over direct regression baselines. This shows the importance of constraining the output space to be on sphere when regress spherical target. Second, putting l_2 normalization constraint on output space, S^3_{exp} improves over S_{flat} with both AlexNet and VGG16. For AlexNet we obtain about $8-12\%$ improvement across all metrics. VGG16 is higher overall, but the improvement over the baseline is less. This shows that with the VGG16 we are potentially getting closer to the maximum possible accuracy attainable for this hard task. That can be explained by the fact that the shapes have no texture. Thus, a regular VGG16 is close to what can be encoded by a good RGB-based model. Note that estimating the 3D rotation with a discretization and classification approach [5, 24, 33, 35] would be impossible because of the vastness of the output space on $\mathcal{SO}(3)$ manifold.

Further, we investigate the variance of the gradients $\frac{\partial \mathcal{L}}{\partial \boldsymbol{O}}$ by recording its average ℓ_2 norm during training progress. The results are shown in Fig. 4. We observe the gradient norm of the spherical exponential mapping has much lower variance. Spherical exponentiation achieves this behavior naturally without interventions, unlike other tricks (*e.g.* gradient clipping, gradient reparameterization) which fix the symptom (gradient instability/vanishing/exploding) but not the root cause (unconstrained input signals). We conclude that spherical regression is successful also for the application of 3D rotation estimation.

6. Conclusion

Spherical regression is a general framework which can be applied to any continuous output problem that lives in n-spheres. It obtains regression gradients that are constrained and well-behaving for several computer vision challenges. In this work we have investigated three such applications, specifically viewpoint estimation, surface normal estimation and 3D rotation estimation. Generally, we observe that spherical regression improves considerably the regression accuracy in all tasks and different datasets. We conclude that spherical regression is a good alternative for tasks where continuous output prediction are needed.

References

[1] Aayush Bansal, Bryan Russell, and Abhinav Gupta. Marr revisited: 2d-3d alignment via surface normal prediction. In *CVPR*, 2016.

[2] Lucas Beyer, Alexander Hermans, and Bastian Leibe. Biternion nets: Continuous head pose regression from discrete training labels. In *GCVPR*, 2015.

[3] R. W. Brockett. Robotic manipulators and the product of exponentials formula. In *Mathematical Theory of Networks and Systems*. 1984.

[4] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*. Datalogisk Institut, Københavns Universitet, 1998.

[5] Gilad Divon and Ayellet Tal. Viewpoint estimation—insights & model. In *ECCV*, 2018.

[6] Thanh-Toan Do, Trung Pham, Ming Cai, and Ian Reid. Real-time monocular object instance 6d pose estimation. In *BMVC*, 2018.

[7] Maura Eduarda and David G Henderson. *Experiencing geometry: On plane and sphere*. Prentice Hall, 1996.

[8] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015.

[9] Luca Falorsi, Pim de Haan, Tim R Davidson, Nicola De Cao, Maurice Weiler, Patrick Forré, and Taco S Cohen. Explorations in homeomorphic variational auto-encoding. *arXiv preprint arXiv:1807.04689*, 2018.

[10] Harley Flanders. *Differential Forms with Applications to the Physical Sciences by Harley Flanders*, volume 11. Elsevier, 1963.

[11] David F Fouhey, Abhinav Gupta, and Martial Hebert. Data-driven 3d primitives for single image understanding. In *ICCV*, 2013.

[12] Jin Gao, Haibin Ling, Weiming Hu, and Junliang Xing. Transfer learning based visual tracking with gaussian processes regression. In *ECCV*, 2014.

[13] Alexander Grabner, Peter M Roth, and Vincent Lepetit. 3d pose estimation and 3d model retrieval for objects in the wild. In *CVPR*, 2018.

[14] David Gurarie. Symmetries and laplacians: Introduction to harmonic analysis, group representations and applications. *Bull. Amer. Math. Soc*, 29, 1993.

[15] William Rowan Hamilton. On quaternions; or on a new system of imaginaries in algebra. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 1844.

[16] David Hoag. Apollo guidance and navigation: Considerations of apollo imu gimbal lock. *Cambridge: MIT Instrumentation Laboratory*, 1963.

[17] Seunghoon Hong, Tackgeun You, Suha Kwak, and Bohyung Han. Online tracking by learning discriminative saliency map with convolutional neural network. In *ICML*, 2015.

[18] Zdenek Kalal, Krystian Mikolajczyk, and Jiri Matas. Tracking-learning-detection. *PAMI*, 2010.

[19] Alex Kendall and Roberto Cipolla. Geometric loss functions for camera pose regression with deep learning. In *CVPR*, 2017.

[20] Lubor Ladicky, Jianbo Shi, and Marc Pollefeys. Pulling things out of perspective. In *CVPR*, 2014.

[21] Lubor Ladicky, Bernhard Zeisl, and Marc Pollefeys. Discriminatively trained dense surface normal estimation. In *ECCV*, 2014.

[22] Siddharth Mahendran, Haider Ali, and René Vidal. 3d pose regression using convolutional neural networks. In *ICCV*, 2017.

[23] Siddharth Mahendran, Haider Ali, and Rene Vidal. A mixed classification-regression framework for 3d pose estimation from 2d images. In *BMVC*, 2018.

[24] Francisco Massa, Renaud Marlet, and Mathieu Aubry. Crafting a multi-task cnn for viewpoint estimation. In *BMVC*, 2016.

[25] J Michael McCarthy. *Introduction to theoretical kinematics*. MIT press, 1990.

[26] Arsalan Mousavian, Dragomir Anguelov, John Flynn, and Jana Košecká. 3d bounding box estimation using deep learning and geometry. In *CVPR*, 2017.

[27] Margarita Osadchy, Yann Le Cun, and Matthew L Miller. Synergistic face detection and pose estimation with energy-based models. *Journal of Machine Learning Research*, 2007.

[28] Hugo Penedones, Ronan Collobert, Francois Fleuret, and David Grangier. Improving object classification using pose information. Technical report, Idiap, 2012.

[29] Sergey Prokudin, Peter Gehler, and Sebastian Nowozin. Deep directional statistics: Pose estimation with uncertainty quantification. In *ECCV*, 2018.

[30] Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. Geonet: Geometric neural network for joint depth and surface normal estimation. In *CVPR*, 2018.

[31] Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH*, 1985.

[32] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[33] Hao Su, Charles R Qi, Yangyan Li, and Leonidas J Guibas. Render for cnn: Viewpoint estimation in images using cnns trained with rendered 3d model views. In *ICCV*, 2015.

[34] Ran Tao, Efstratios Gavves, and Arnold W M Smeulders. Siamese instance search for tracking. In *CVPR*, 2016.

[35] Shubham Tulsiani and Jitendra Malik. Viewpoints and keypoints. In *CVPR*, 2015.

[36] Anran Wang, Jiwen Lu, Gang Wang, Jianfei Cai, and Tat-Jen Cham. Multi-modal unsupervised feature learning for rgb-d scene labeling. In *ECCV*, 2014.

[37] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. 3d shapenets: A deep representation for volumetric shapes. In *CVPR*, 2015.

[38] Yu Xiang, Roozbeh Mottaghi, and Silvio Savarese. Beyond pascal: A benchmark for 3d object detection in the wild. In *WACV*, 2014.

[39] Yinda Zhang, Shuran Song, Ersin Yumer, Manolis Savva, Joon-Young Lee, Hailin Jin, and Thomas Funkhouser. Physically-based rendering for indoor scene understanding using convolutional neural networks. In *CVPR*, 2017.