PVNN: A neural network library for photometric vision

Ye Yu and William A. P. Smith
Department of Computer Science, University of York, UK
{yy1571,william.smith}@york.ac.uk

Abstract

In this paper we show how a differentiable, physics-based renderer suitable for photometric vision tasks can be implemented as layers in a deep neural network. The layers include geometric operations for representation transformations, reflectance evaluations with arbitrary numbers of light sources and statistical bidirectional reflectance distribution function (BRDF) models. We make an implementation of these layers available as a neural network library (PVNN) for Theano. The layers can be incorporated into any neural network architecture, allowing parts of the photometric image formation process to be explicitly modelled in a network that is trained end to end via backpropagation. As an exemplar application, we show how to train a network with encoder-decoder architecture that learns to estimate BRDF parameters from a single image in an unsupervised manner.

1. Introduction

Physics-based vision seeks to solve problems by modelling the underlying physical processes that give rise to an observed image. Such a mathematical model provides insight into the problem, allowing conclusions to be drawn about the existence and uniqueness of solutions. These physical models are then either explicitly inverted or a solution obtained by energy minimisation in an analysis-by-synthesis framework. Classical problems in physics-based vision include shape-from-shading, inverse rendering, illumination estimation and reflectance measurement. An attractive property of physics-based approaches is that they provide an explanation of appearance in terms of physically-meaningful parameters that can be edited for re-rendering or used for tasks such as material classification.

In some ways, deep learning is an entirely orthogonal approach to physics-based vision. A deep neural network (DNN) is a black box that has learnt a vector to vector mapping that transforms images into a useful representation. DNNs, and specifically convolutional neural networks (CNNs), have proven remarkably successful at doing this on a wide range of vision problems. This includes classical physics-based vision problems. In its simplest form, a black box CNN can be trained to directly regress the physical parameter of interest from an image. The challenge in such supervised approaches is the acquisition of training data since acquiring images with registered depth maps, illumination estimates and so on is itself an open problem. One way around this is to use synthetic training data, where the images are rendered from models. However, the ability of the CNN to generalise to real world data is limited by the realism of the synthetic training data. Moreover, such a black box reveals nothing about the underlying physical processes and the model must be retrained in order to estimate different physical quantities or solve different problems.

A promising alternative is to combine model- and learning-based vision. One way to do this is to incorporate physical models into a CNN by introducing layers that explicitly implement physical models. This potentially offers a route to unsupervised learning of physics-based vision tasks with CNNs. For example, a black box encoder that transforms images to physically meaningful parameters can be trained by pairing it with a physically-based decoder that renders an image from the estimated parameters allowing an unsupervised loss to be computed between the original and rendered images. Many other architectures are also possible that exploit these physics-based layers.

In this paper we present PVNN, a Theano [1] toolbox that implements operations often used in photometric vision as part of a neural network. We see this toolbox as a photometric analog of the geometric functionality provided by the Geometric Vision with Neural Networks (gvnn) toolbox [14]. In a forward pass, combinations of our layers can act as a differentiable, physically-based renderer. In an encoder-decoder architecture as described above, PVNN can be used to train CNNs to solve physics-based vision problems in an unsupervised manner. In Section 3 we describe the layers that make up PVNN. In Section 4 we show how these layers can be used for differentiable forward rendering. In Section 5 we demonstrate an exemplar application by using unsupervised learning to train a CNN to predict to BRDF parameters from single images.

2. Related work

In order to include physics-based models in a CNN, the key component is a differentiable rendering layer that can reconstruct input-like images whilst enabling backpropogation during training. Hence, we begin by reviewing related work that utilises trainable or fixed, differentiable renderers for vision tasks as well as previous work on BRDF estimation and modelling.

Differentiable rendering. In energy minimisation approaches to inverse graphics problems, there is a need for differentiable renderers that model the physical process of image intensity formation. This allows minimisation of an appearance error using gradient descent or other first-order optimisation algorithms. Loper et al. [22] developed differentiable renderer package called OpenDR as a framework. OpenDR could reproduce the observed images with latent variables like appearance, geometry and camera, then the error between reproduced image intensity map and the observation can be minimised. Zienkiewicz et al. [39] applied a similar idea for estimating height maps in a real-time robotic system. The rendering objective is height map rather than RGB intensities in their case.

Trainable render layer. Some work has investigated the idea of a trainable renderer. Dosovitskiy et al. [3] proposed a neural network for image reconstruction from extracted feature maps resulting from different hand-crafted or learned descriptors like HOG, SIFT or trained deterministic CNN. They trained an up-convolutional network against the loss function comparing original inputs and reconstruction. Similarly, using differentiable rendering layers inside a network to perform image reconstruction were implemented by [4, 13, 8, 38, 21]. All trainable rendering networks introduced by these works are generative models simulating the probabilistic process of RGB image synthesis. In contrast, our network models a discriminative process on statistical BRDF parameter estimation problem and uses a fixed rendering architecture to formulate the loss function. Both of [12, 28] employed adversarial architectures in their networks so that incorporating a recurrent-racing mechanism between the generative and discriminative models. This could improve the realism of their rendered images. It is different from the pure generative models, since both of them trained a generative model followed by a discriminative model simultaneously. Jaderberg et al. [18] proposed a plugin renderer layer to seamlessly work along with existing CNN architectures, in order to activate the network's capability on recognising images with spatial transformation. The work in [19] presented a sophisticated extension from its early counterpart [16] from Hinton et al. Nalbach et al. [26] explored the graphical shading problem by using rendering layers. Specifically, their Deep Shading net explicitly takes as input the meaningful scenery attributes defined in common shaders and use well-trained neural network to act as a self-taught shader as opposite to manually designed shader.

Render layers in CNNs. Another group of differentiable rendering layers embedded in CNN are untrainable rendering layers that usually make their network attack a specific problem. Our proposed network fall into this group of work. The attractive feature of such a fixed rendering layer is that the decoders well understood physical theory as a prior that can provide constraint during unsupervised training. Richardson et al. [30] facilitated such a hand-crafted rendering layer for face images. Their fixed differentiable rendering layer works on finetuning results provided from a former discriminative model, and it plays the role of encouraging similarity between rendered images and original input face images. Tung et al. [5] concentrated on a similar idea of inverse graphics and incorporate an adversarial architecture into the network. Through unsupervised learning, their network can deal with many physics-based vision problems upon physics-specific rendering layers in conjunction with GANs architecture [12]. A variety of work [7, 11, 14, 34, 37] focused on specific physics-based vision problems, and realised unsupervised learning by utilising deep learning in conjunction with a problem-oriented rendering layer. Other than using rendering layers as a tool for learning in an unsupervised setting, Cole et al. [2] achieved great frontal-face rendering results by using a warping layer as a rendering subnet combined with landmark and texture extraction networks. Their loss function consists of two portions: errors of landmark and texture predictions and error of rendered frontal-face image and both of them need ground truth in training. Tang et al. [33] assumed Lambertian reflectance in their render layer and perform a multiplicative rendering process on the top of their generative model. Tulsiani et al. [35] presented an unsupervised learning scheme on a monocular depth estimation problem through their special ray tracing loss function, which is similar in function to a render layer.

BRDF estimation and modelling. BRDF estimation using CNNs has recently been considered. Kim et al. [20] assumed the Ward BRDF model and trained their network to predict the Ward model parameters of objects. One of their proposed networks contains a fixed layer to perform pre-processing on voxel clouds which are inputs for the following CNN architecture. Georgoulis et al. [9] used their DeLight-Net to estimate Phong model parameters and environment maps given a reflectance map of a captured material and illumination. Rematas et al. [29] tried to extract the reflectance map from single RGB image through their deep network. To achieve a better performance, they proposed to used fixed sampling layer and domain conversion layer bridging two subnets that work on surface normal prediction and dense reflectance map prediction respectively.

Other than CNN-based BRDF estimation, traditional BRDF estimation and modelling often places much greater restrictions on data capture. Our main inspiration comes from one of the non-CNN work [27]. Nielsen et al. [27] in this work proposed a statistical model of BRDFs, but their BRDF reconstruction requires images under constrained, calibrated capture conditions. Just like Nielsen's work, this branch of work heavily rely on sampling algorithm and the reconstruction quality is determined by sampling results. In early work, White et al. [36] set up a gonioreflectometer gantry to capture the 4D BRDF in a brute force way. The exhaustive sampling is time consuming and unwieldy, some other lightweight sampling tricks have been proposed. Marschner et al. [23] performed sampling on a sphere or curved object to simplify the gonioreflectometer measurement. Ghosh et al. [10] proposed to sample reflectance upon spherical zonal basis function. Fuchs et al. [6] applied adaptive sampling over their setup to overcome the most common aliasing problems. Very close to the statistical model in the heart of our network [27], Matusik et al. [25] derived similar BRDF model from MERL dataset but with linear PCA method that is less able to capture the distinctive features of a BRDF.

3. Photometric Vision with Neural Networks (PVNN)

In this section we present the layers that are provided by PVNN. All layers are implemented by a series of differentiable operations provided by Theano, which could be defined as customised layers in CNN.

3.1. Geometric transformations

Most commonly, photometric vision is concerned with single viewpoint problems and so usually operates in the depth or surface normal domains (as opposed to an object-space mesh representation). We use such a viewer-centred depth map representation in PVNN. One advantage of this formulation is that we need not model self-occlusions of the surface. Occlusion is a binary, and hence discontinuous, function. This means that it is not differentiable and therefore occlusion information cannot be exploited for learning during backpropagation.

We assume here orthographic projection, though it would be straightforward to modify our layers to account for perspective projection (taking into account the intrinsic parameters of the camera).

3.1.1 Surface height differentiation

This layer transforms a depth map into estimates of the surface gradient in the horizontal and vertical directions. It takes as input a depth map, $\mathbf{z} \in \mathbb{R}^{m\times n}$, that is assumed to be smooth containing the depth values for $m \times n$ pixels: $\mathbf{G} = \begin{bmatrix}\mathbf{D}_x * \mathbf{z} & \mathbf{D}_y * \mathbf{z}\end{bmatrix}$, where $\mathbf{D}_x \in \mathbb{R}^{3\times3}$ and $\mathbf{D}_y \in \mathbb{R}^{3\times3}$ evaluate the surface gradient in the horizontal and vertical directions respectively using forward finite differences and convolution:

$$\mathbf{D}_x = \frac{1}{12}\begin{bmatrix}-1 & 0 & 1\\ -4 & 0 & 4\\ -1 & 0 & 1\end{bmatrix}, \mathbf{D}_y = \frac{1}{12}\begin{bmatrix}1 & 4 & 1\\ 0 & 0 & 0\\ -1 & -4 & -1\end{bmatrix} \tag{1}$$

3.1.2 Gradient to normal vector

This layers applies the function $\mathbf{n} : \mathbb{R}^2 \mapsto \mathbb{R}^3$ which transforms the gradient vector into a vector whose direction is normal to the surface: $\mathbf{n}(\mathbf{g}) = \begin{bmatrix}-\mathbf{g}^T & 1\end{bmatrix}^T$. This function is applied to the gradient estimate at each pixel, yielding u surface normal vectors, where u is length of flatten vector from $m \times n$ map.

3.1.3 Vector normalisation

It is often convenient to work with surface normal vectors of unit length. So, this layer applies the function $\bar{\mathbf{n}} : \mathbb{R}^3 \mapsto \mathbb{R}^3$ which normalises a vector to have unit length such that $\|\bar{\mathbf{n}}\| = 1$. This function is simply: $\bar{\mathbf{n}}(\mathbf{n}) = \mathbf{n}/\|\mathbf{n}\|$.

3.1.4 Unit vector to spherical coordinates

Sometimes, it is useful to transform the surface normal vector $\bar{\mathbf{n}}$ into spherical coordinates (α, θ) in a viewer-centred coordinate system. The azimuth angle is computed by the function $\alpha : \mathbb{R}^3 \mapsto [0, 2\pi)$: $\alpha(\bar{\mathbf{n}}) = \text{atan2}(\bar{n}_2, \bar{n}_1)$. And the zenith angle is from function $\theta : \mathbb{R}^3 \mapsto [0, \pi]$: $\theta(\bar{\mathbf{n}}) = \arccos(\bar{n}_3)$.

3.1.5 Conversion to Rusinkiewicz coordinates

We assume that the BRDFs are isotropic and hence can be expressed as three dimensional functions. For convenience and compatibility with the statistical BRDF model used later, we parameterise BRDFs in terms of the three angles proposed by Rusinkiewicz [31]. Concretely, the Rusinkiewicz coordinates parameterise the local reflectance geometry in terms of three angles relative to the halfway vector:

$$\mathbf{h}(\mathbf{s}, \mathbf{v}) = \bar{\mathbf{n}}(\mathbf{s} + \mathbf{v}), \tag{2}$$

where $\mathbf{s}$ and $\mathbf{v}$ are unit vectors in the light source and viewer directions respectively. The angle $\theta_h(\mathbf{n}, \mathbf{s}, \mathbf{v}) \in [0, \pi/2]$ is the angle between $\mathbf{h}$ and $\mathbf{n}$ while $\theta_d(\mathbf{n}, \mathbf{s}, \mathbf{v}) \in [0, \pi/2]$ and $\phi_d(\mathbf{n}, \mathbf{s}, \mathbf{v}) \in [0, \pi]$ are the spherical coordinates of $\mathbf{s}$ in a coordinate system in which $\mathbf{h}$ is at the north pole.

Hence, we define a layer that takes as input the per-pixel surface normals $\mathbf{N} \in \mathbb{R}^{u\times 3}$ for u pixels, light directions

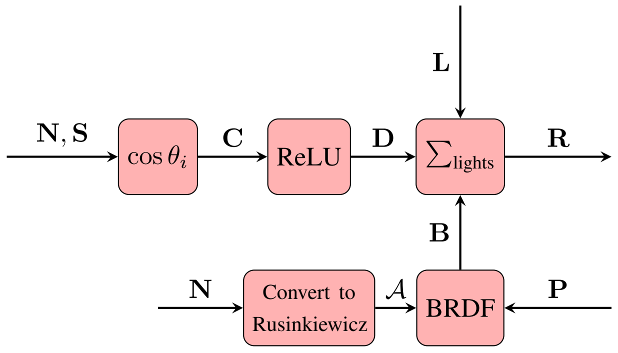


Figure 1: A reflectance evaluation sub-network. It takes normals $\mathbf{N}$, light source directions $\mathbf{S}$, light source colours $\mathbf{L}$ and BRDF parameters $\mathbf{P}$ as input and outputs radiance values per-pixel $\mathbf{R}$.

$\mathbf{S} \in \mathbb{R}^{s\times 3}$ for s light sources and per-pixel viewer directions $\mathbf{V} \in \mathbb{R}^{u\times 3}$ which are all $[0\ 0\ 1]$ for an orthographic projection. The output is a tensor $\mathcal{A} \in \mathbb{R}^{u\times s\times 3}$ containing the three reflectance angles for each pixel/light source combination, such that $\mathcal{A}_{ij1} = \theta_h(\mathbf{n}_i, \mathbf{s}_j, \mathbf{v})$, $\mathcal{A}_{ij2} = \theta_d(\mathbf{n}_i, \mathbf{s}_j, \mathbf{v})$ and $\mathcal{A}_{ij3} = \phi_d(\mathbf{n}_i, \mathbf{s}_j, \mathbf{v})$

3.2. Reflectance evaluation network

A reflectance evaluation network is a sub-network that computes the radiance reflected towards the viewer, for a given BRDF, surface normal direction ($\mathbf{n}$), viewer direction ($\mathbf{v}$) and lighting environment. A BRDF is a function $\rho(\theta_h, \theta_d, \phi_d)$ that returns the ratio of reflected radiance to the incident irradiance for a pair of incoming and outgoing directions. We write $\rho(\mathbf{n}, \mathbf{s}, \mathbf{v})$ as shorthand for $\rho(\theta_h(\mathbf{n}, \mathbf{s}, \mathbf{v}), \theta_d(\mathbf{n}, \mathbf{s}, \mathbf{v}), \phi_d(\mathbf{n}, \mathbf{s}, \mathbf{v}))$, where $\mathbf{s}$ is the incident radiance direction.

For a continuous lighting environment, the reflected radiance, L_o, is given by:

$$L_o = \int_{S^2} \rho(\mathbf{n}, \boldsymbol{\omega}_i, \mathbf{v}) L(\boldsymbol{\omega}_i) \max(0, \mathbf{n}\cdot\boldsymbol{\omega}_i) d\boldsymbol{\omega}_i, \tag{3}$$

where $L(\boldsymbol{\omega}_i)$ is the incident radiance from direction $\boldsymbol{\omega}_i$. In practice, we discretise over s point light sources and evaluate:

$$\mathbf{r} = \sum_{i=1}^{s} \rho(\mathbf{n}, \mathbf{s}_i, \mathbf{v}) \mathbf{l}_i \max(0, \mathbf{n}\cdot\mathbf{s}_i) \tag{4}$$

where $\mathbf{r} \in \mathbb{R}^3$ is the RGB colour radiance vector, $\mathbf{l}_i \in \mathbb{R}^3$ the colour of the ith light source and $\mathbf{s}_i \in \mathbb{R}^3$, $\|\mathbf{s}_i\| = 1$ the direction of the ith light source.

The individual layers that we use to implement a reflectance evaluation are shown in Figure 1. We describe each of these layers in the following sections.

3.2.1 BRDF

The BRDF layer evaluates a parametric BRDF function $\rho(\theta_h, \theta_d, \phi_d, \mathbf{p})$ for each pixel/light source where $\mathbf{p}$ is the vector of additional parameters required by the BRDF. Any BRDF function can be used so long as it is differentiable. Physically valid BRDF models must obey the three properties of positivity, Helmholtz reciprocity and conservation of energy. A particularly simple, physically valid BRDF is the Lambertian model. This models a perfectly diffuse surface, i.e. where light is scattered equally in all directions, and has a single parameter k_d (the diffuse albedo):

$$\rho_{\text{Lambert}}(\theta_h, \theta_d, \phi_d, k_d) = \frac{k_d}{\pi}. \tag{5}$$

Concretely, the BRDF layer takes as input a tensor containing the reflectance angles for each pixel/light source $\mathcal{A} \in \mathbb{R}^{u\times s\times 3}$ and a matrix $\mathbf{P} \in \mathbb{R}^{p\times u}$ or vector $\mathbf{p} \in \mathbb{R}^p$ containing the p parameters (per-pixel or per-image respectively). It outputs a BRDF value for each pixel/light source as $\mathbf{B} \in \mathbb{R}^{u\times s}$:

$$\mathbf{B}_{ij} = \rho(\mathcal{A}_{ij1}, \mathcal{A}_{ij2}, \mathcal{A}_{ij3}, \mathbf{P}_i). \tag{6}$$

3.2.2 Statistical BRDF network

Parametric BRDF models are popular and widely used in graphics and vision. However, no single parametric model is capable of generalising to the wide range of reflectance properties observed in the real world. For this reason, statistical [27] or dictionary-based [17] models built from measured data are becoming increasingly popular.

As an alternative to parametric BRDFs, PVNN also supports statistical BRDF models; specifically, the model of Nielsen et al. [27] that is trained on the MERL reflectance database [24]. This is implemented as a subnetwork (see Figure 2) that evaluates the statistical model to generate an empirical BRDF, inverts the log-relative mapping used in the Nielsen [27] model and finally performs differentiable interpolation into the empirical BRDF.

Statistical BRDF model The statistical BRDF model layer evaluates a linear statistical model of empirical BRDFs. An empirical BRDF $\mathcal{X} \in \mathbb{R}^{90\times 90\times 180}$ is a representation of a discretely measured BRDF at $k = 90 \times 90 \times 180$ samples over $(\theta_h, \theta_d, \phi_d)$ space. We compute a vectorised representation $\mathbf{x} = \text{vec}(\mathcal{X}) \in \mathbb{R}^k$ as:

$$\mathbf{x} = \mathbf{Q}\mathbf{p} + \boldsymbol{\mu} \tag{7}$$

where $\mathbf{Q} \in \mathbb{R}^{k\times p}$ contains the p principal component vectors, $\boldsymbol{\mu} \in \mathbb{R}^k$ the mean and $\mathbf{p} \in \mathbb{R}^p$ is a vector of weights, which by appropriate scaling of the principal components, has elements that follow the standard normal distribution.

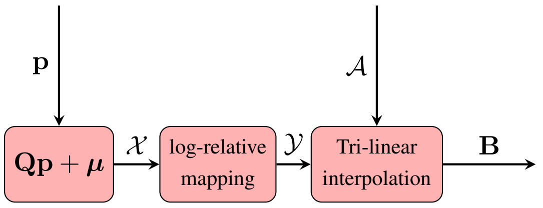


Figure 2: A statistical BRDF model sub-network. This network can be plugged in place of a parametric BRDF model in a reflectance evaluation network (i.e. in place of the "BRDF" node in Figure 1). It takes statistical BRDF parameters $\mathbf{p}$ and per-pixel Rusinkiewicz coordinates $\mathcal{A}$ as input and outputs BRDF values $\mathbf{B}$.

The layer takes $\mathbf{p}$ as input and outputs $\mathcal{X}$. Note that this layer is exactly equivalent to a fully connected layer in which $\mathbf{Q}$ plays the role of the weights and $\boldsymbol{\mu}$ the biases. Hence, it would be straightforward to make this layer trainable and, in principle, to learn statistical BRDF models as part of the network training. We do not do this here though and keep the model fixed to that of Nielsen et al. [27].

Note that the statistical model of Nielsen et al. [27] contains some missing values. For simplicity, we interpolated these missing values by their nearest neighbours. We precompute the relative relationship between missing values and known values, and directly perform mapping through precomputed mapping look-up table. Although a more sophisticated missing data scheme could be used, we notice no visual artefacts through the use of nearest neighbour.

Log-relative mapping Nielsen et al. [27] proposed a novel "log-relative" mapping that is applied to empirical BRDFs prior to statistical modelling. The purpose of this mapping is to alleviate the high dynamic range which resulted in the specular peak completely dominating the statistical model and reconstruction. Hence, in our network, after reconstruction we must invert this mapping to recover an actual empirical BRDF. In [27] all the samples in the MERL dataset were mapped by:

$$\rho_{\text{map}} = \ln\left(\frac{\rho\ \cos_{\text{weight}} + \epsilon}{\rho_{\text{ref}}\ \cos_{\text{weight}} + \epsilon}\right), \tag{8}$$

where

$$\cos_{\text{weight}} = \max\left[\cos(\mathbf{n}\cdot\mathbf{s})\cos(\mathbf{n}\cdot\mathbf{v}), \epsilon\right], \tag{9}$$

is a weight applied to compensate for extreme grazing-angle values, ρ_{ref} is a reference BRDF (set as the median BRDF value for each sampled angles) and ϵ is the numerical stabilisation term, set equal to 10^{-3}.

In our statistical BRDF network, we reconstruct a discretely sampled BRDF $\mathcal{X}$ and then invert (8) using:

$$\mathcal{Y} = \frac{\exp(\mathcal{X}) \odot (\mathcal{X}_{\text{ref}} \odot \cos_{\text{weight}} + \epsilon)\ -\ \epsilon}{\cos_{\text{weight}}}, \tag{10}$$

where $\odot$ is the Hadamard (element-wise) product and the division is also applied element-wise.

BRDF value sampling Given an empirical BRDF, $\mathcal{Y}$, with the log-relative mapping inverted and per-pixel/light source Rusinkiewicz 3D coordinates, $\mathcal{A}$, we interpolate an exact BRDF value by tri-linear interpolation. The purpose of this step is to find appropriate BRDF values from a discrete BRDF lookup table when the indices $(\theta_h, \theta_d, \phi_d)$ are continuous. Differentiable tri-linear interpolation can be performed using:

$$\rho(\theta_h,\theta_d,\phi_d) = \sum_{i=1}^{90}\sum_{j=1}^{90}\sum_{k=1}^{180}\mathcal{Y}_{ijk}\max(0, 1-|\theta_h\frac{180}{\pi}-i|)\times$$
$$\max(0, 1-|\theta_d\frac{180}{\pi}-j|)\max(0, 1-|\phi_d\frac{180}{\pi}-k|).$$

We perform such a look up for the Rusinkiewicz coordinates for each pixel/light source and output a matrix of BRDF values, $\mathbf{B}$.

3.2.3 Cosine weight layer

This layer computes cosine weights, $\cos(\theta_i) = \mathbf{n}\cdot\mathbf{s}$, for each pixel:

$$\mathbf{C} = \mathbf{N}\mathbf{S}^T, \tag{11}$$

where $\mathbf{N} \in \mathbb{R}^{u\times 3}$ contains the surface normal vectors for u pixels and $\mathbf{S} \in \mathbb{R}^{s\times 3}$ contains the lighting direction vectors for s light sources.

3.2.4 Clamping layer

This layer implements the max operator in (4) which simulates the effect of self-shadows (i.e. incident angle greater than 90°). It takes as input the output of the cosine layer and computes:

$$\mathbf{D} = \max(0, \mathbf{C}) \tag{12}$$

Note that this is exactly a Rectified Linear Unit (ReLU) layer and so can be implemented using a standard ReLU layer implementation.

3.2.5 Colour scale and sum over light sources

This layer takes as input the clamped-cosine and BRDF values, element-wise multiplies them, uses these to scale the light source radiances and finally sums over light sources:

$$\mathbf{R}_i = \sum_{j=1}^{s}\mathbf{B}_{ij}\mathbf{D}_{ij}\mathbf{L}_j. \tag{13}$$

This provides the final output of the reflectance evaluation network: per-pixel reflected radiance values.

4. Forward rendering

We now illustrate how the layers in PVNN can be combined to implement forward rendering. Such a network takes as input a depth map, one or more light source directions and colours and the BRDF parameters or an empirically measured BRDF and outputs a rendered image. To demonstrate the rendering capability of this network, we render spheres with environment map illumination. In Figure 3 we show results using well reconstructed BRDF presented by [27]. In Figure 4 we show results using a single set of statistical BRDF parameters and three different illumination environments. We note that our renderings are comparable to the output from conventional renderers as well as the statistical reconstructions shown by Nielsen et al. [27].

To illustrate the simplicity with which PVNN layers can be combined to achieve physically-based rendering, the following code snippet is all that is required to create images such as those in Figure 4:

```
# initialise renderer subnet
pvnn = pvnn_module.pvnn(shapes_var, mask_var)
# calculate surface normal map
normals = pvnn.depthToNormal(shapes_var)
# cosine weights for foreshortening
cosWeights = pvnn.cosWeight(normals, lights_var)
# filter out negatives
clampCosWeights = pvnn.clamping(cosWeights)
# reconstruct BRDF from predicted vector
brdfFn = pvnn.brdfFunction(prediction, './
    precomputedData')
# interpolate missing values in reconstruction
brdfFn = pvnn.brdfFnInterp(brdfFn, 'mapping.npy')
# indexing BRDF values for each pixel
brdfMatrix = pvnn.brdfValues(normals, \
    lights_var, brdfFn)
# intensities formation
imgs = pvnn.brdfIntensity(brdfMatrix, \
    clampCosWeights, lightColors_var, mask_var)
# transpose output to have same shape with input
imgs = imgs.transpose(0,3,1,2)
```

5. Example application: BRDF estimation

We now present an example of using PVNN in a vision application. We show how to train a CNN to directly regress BRDF parameters from a single image. Moreover, we train the network in an unsupervised fashion in the sense that BRDF parameters are not provided at training time. Instead, we use an encoder-decoder architecture in which the loss function is the error between a rendering using the predicted parameters and the input image.

5.1. Architecture

An overview of the network is shown in Figure 5 including both training and inference subnets. The inference part inside the dot-line box is a standard down-scaling ConvNet, taking input images and producing BRDF vector as output.

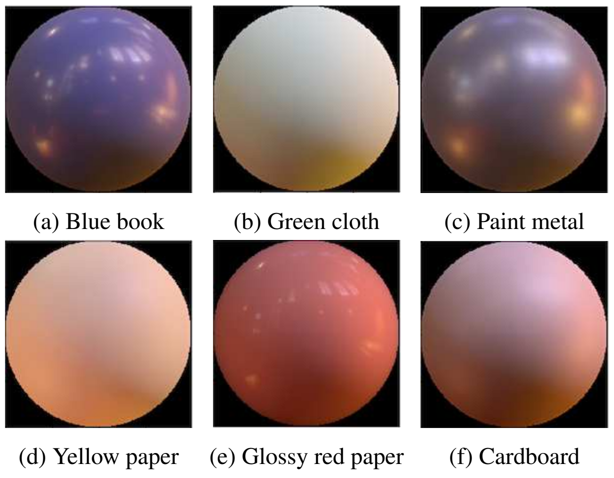


Figure 3: Images are rendered by our differentiable renderer. The inputs for the renderer is the BRDF function of the known material and environment map and depth map of the object, the output is the realistic rendered images.

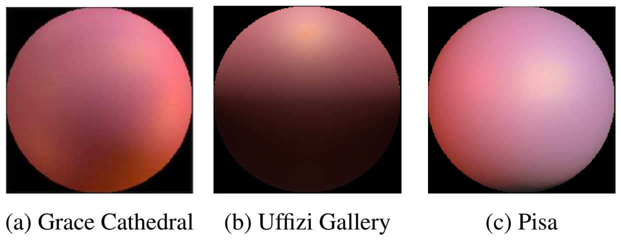


Figure 4: Three rendered examples from BRDF parameters under different HDR Environment Maps. All of our results are using same 5D parameters.

The other parts of the network are untrainable and designed for unsupervised training. During training procedure, the weights in the ConvNet will be updated by the Error layer given output from the Renderer layer. As described in Section 3, the Renderer layer takes as input the depth maps, light sources and BRDF parameters and outputs realistic RGB images. The Error layer, placed at the end of our network, calculates a loss function by the L2 norm between the original input image and the rendered intensity map. In addition to this intensity loss, the network parameter regularisation term is another part of our loss function. The coefficient of regularisation is 5×10^{-4}.

5.2. Convolutional Neural Network

The architecture of our ConvNet is realised by a stack of bottlenecks and residual shortcuts mechanism introduced by He et al. [15]. Since our problem is to estimate BRDF parameters from only one single image, a deep network with powerful learning ability is required. We construct our network with 50 layers following a resnet layout. Also, following the downscale rule proposed by Simonyan et al.

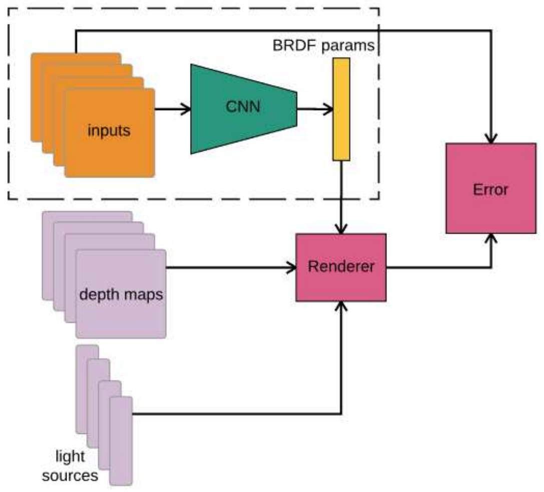
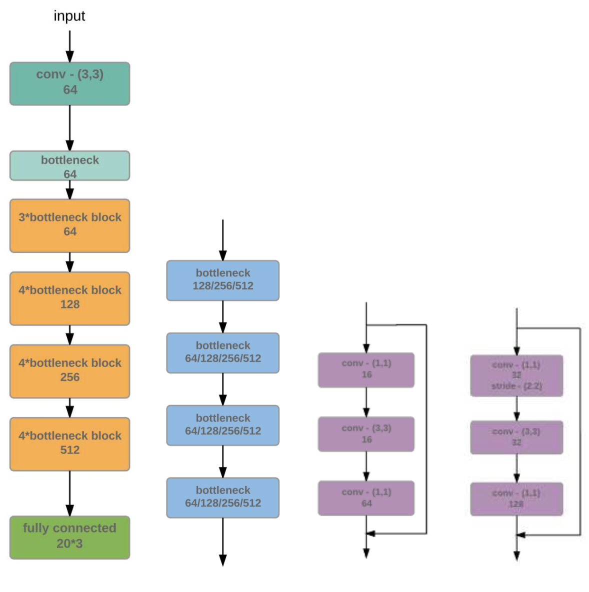


Figure 5: The overall architecture of our neural network. The dot-line box highlights the part for used for BRDF parameter inference during testing.



(a) (b) (c) (d)

Figure 6: Network architecture. (a) Overall architecture; (b) Bottleneck stack, shown as orange boxes placed in the middle of (a); (c) Convolutional layers inside bottleneck; (d) Convolutional layers inside the first bottleneck of each block, which needs to increase the dimension of feature map and decrease spatial resolution at same time.

[32] in their VGG net, our network performs pooling operations followed by dimension increasing on the feature map. All the convolutional layers are separated by 4 blocks after which spatial pooling is performed, and dimensional increments are achieved by the first layer of the next block. To concatenate 48 bottleneck convolutional layers and 1 fully-connected layer and 1 actuation convolutional layer, each convolutional block contains 1 dimension increment bottleneck and 3 normal bottlenecks. The actuation layer in our network is placed next to the input layer, converting RGB input images to initial feature maps with 64 channels. The fully-connected layer at the end of the net map output from convolutional layer to 15-D vectors, which represents three 5 dimensional BRDF statistical model parameters for each colour channel respectively. The visualisation of our network is shown in Figure 6. Note that the first bottleneck right after the actuation layer does not increase the dimension but maintains its spatial size and dimension. Except for the first bottleneck block, the pooling and dimension increment operations are proceeded by the first bottleneck in each block. For simplicity the pooling is done by using stride of size $(2, 2)$, whose efficiency has been proven by [15]. The architecture of the bottlenecks can be found in Figure 6 as well. For each operational box, the numbers written at the bottom indicate the dimension of the output feature map. Figures 6d and 6c are examples of the first bottleneck block.

Dataset	**PCs Params.**	**Full BRDF**
Training	0.01425	1.951
Testing	0.01437	1.958

Table 1: RMSE on training and testing set.

5.3. Inference

From the results in [27], here we use first 5 projected principal components in the statistical model, which is sufficient for good BRDF reconstructions. Although it would be straightforward to estimate more parameters, we use the same number as the original paper to simplify the problem and make calibration easy by using [27]. To fully reconstruct the BRDF look up table using our result vector, as part of PVNN we port their nonlinear reconstruction algorithm to Theano to output a standard `.binary` file.

5.4. Training

We employed SGD algorithm with batch size of 20 in backpropagation and run training 300 epochs. The learning schedule started by a relatively small value 0.001, which is good for network convergence as illustrated in [15]. A larger learning rate 0.01 is then substituted to speed up training. Since the learning process is unsupervised (via indirect comparison between prediction and ground truth), we found the network is slower to stabilise than supervised training which directly encourage prediction to approach ground truth. As a result, more epochs used in training are necessary. So we keep using 0.01 as the learning rate for 200 epochs, which is longer than typical learning schemes.

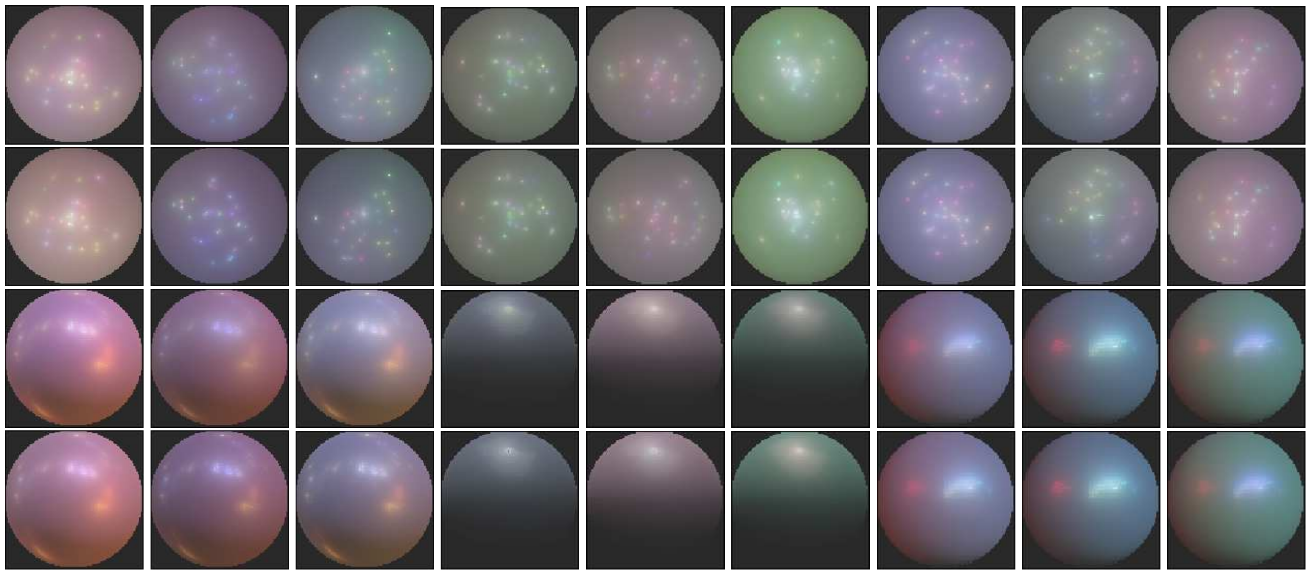


Figure 7: BRDF estimation results. The first row in each block is ground truth, and second row is reconstruction. Each column is the relighting results on the same material. The first two rows show the inputs for the network (row 1) and relighting using predicted BRDF (row 2). The second two rows of images are renderings under new lighting for both ground truth (row 3) and predicted (row 4) BRDF.

For training we use 7000 synthetic images and 3000 synthetic images for validation and testing. Every image is rendered by using randomly drawn 5D BRDF parameters, 30 light sources with random direction and colour and spherical depth map. The BRDF parameters are drawn from a normal distribution with the variance for each dimension coming from the PCA model.

6. Experiments

The experiments are deployed on synthetic data described in last section. We quantitatively evaluate the accuracy of the BRDF parameter estimates made by our inference network in two ways. First we compute the RMSE between the predicted and actual 5D BRDF parameters (Table 1, first column). Second we compute the RMSE between the BRDF reconstructed by our parameter estimate the actual BRDF (Table 1, second column). This second error can be compared directly to the values in [27], showing that we are achieving comparable accuracy without knowledge of the lighting conditions and using only a single image. We show qualitative results in Figure 7. The first two rows show actual input images (first row) and relightings using the estimated BRDF and ground truth lighting (second row). In the bottom two rows we show reilluminations of the ground truth (third row) and estimated (fourth row) BRDFs under environment lighting. In both cases the estimated BRDF gives very close visual appearance to ground truth and captures the key features of the reflectance properties.

7. Conclusions

In this paper we have presented a set of novel layers that can be incorporated into a neural network to incorporate explicit models of photometric image formation. We release these layers as the PVNN toolbox for Theano[1]. We have illustrated how these layers can be used in a simple photometric vision application, though the setup was quite simple (the images are always of spheres so the inference network essentially "knows" the shape of the object even though this is not explicit). However, there are a wide range of much more ambitious applications where PVNN could be used. For example, the same encoder-decoder architecture could be used to train a network that predicts all of depth, lighting and reflectance parameters. We plan to train such a network using photometric stereo datasets where Siamese training can be used to ensure the same depth and reflectance parameters are estimated from pairs of images where only lighting varies. In terms of limitations, the most obvious extensions for PVNN would be to add support for perspective projection and alternate lighting models, such as spherical harmonics, to improve efficiency.

Acknowledgements

We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan X Pascal GPU used for this research.

[1]https://github.com/YeeU/PVNN.git

References

[1] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-farley, and Y. Bengio. Theano: A CPU and GPU math compiler in python. In *Proc. Python in Science Conference*, 2010. 1

[2] F. Cole, D. Belanger, D. Krishnan, A. Sarna, I. Mosseri, and W. T. Freeman. Face synthesis from facial identity features. *CoRR*, abs/1701.04851, 2017. 2

[3] A. Dosovitskiy and T. Brox. Inverting convolutional networks with convolutional networks. *CoRR*, abs/1506.02753, 2015. 2

[4] A. Dosovitskiy, J. T. Springenberg, and T. Brox. Learning to generate chairs with convolutional neural networks. *CoRR*, abs/1411.5928, 2014. 2

[5] H.-Y. Fish Tung, A. Harley, W. Seto, and K. Fragkiadaki. Adversarial Inversion: Inverse Graphics with Adversarial Priors. *ArXiv e-prints*, May 2017. 2

[6] M. Fuchs, V. Blanz, H. P. Lensch, and H.-P. Seidel. Adaptive sampling of reflectance fields. *ACM Trans. Graph.*, 26(2), June 2007. 3

[7] R. Garg, V. K. B. G, and I. D. Reid. Unsupervised CNN for single view depth estimation: Geometry to the rescue. *CoRR*, abs/1603.04992, 2016. 2

[8] L. A. Gatys, A. S. Ecker, and M. Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015. 2

[9] S. Georgoulis, K. Rematas, T. Ritschel, M. Fritz, L. J. V. Gool, and T. Tuytelaars. Delight-net: Decomposing reflectance maps into specular materials and natural illumination. *CoRR*, abs/1603.08240, 2016. 2

[10] A. Ghosh, S. Achutha, W. Heidrich, and M. O'Toole. Brdf acquisition with basis illumination. In *Proc. ICCV*, pages 1–8, 2007. 3

[11] C. Godard, O. Mac Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CoRR*, abs/1609.03677, 2016. 2

[12] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Proc. NIPS*, pages 2672–2680, 2014. 2

[13] K. Gregor, I. Danihelka, A. Graves, and D. Wierstra. DRAW: A recurrent neural network for image generation. *CoRR*, abs/1502.04623, 2015. 2

[14] A. Handa, M. Blösch, V. Patraucean, S. Stent, J. McCormac, and A. J. Davison. gvnn: Neural network library for geometric computer vision. *CoRR*, abs/1607.07405, 2016. 1, 2

[15] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 6, 7

[16] G. Hinton, A. Krizhevsky, and S. Wang. Transforming auto-encoders. *Proc. ICANN*, pages 44–51, 2011. 2

[17] Z. Hui and A. C. Sankaranarayanan. A dictionary-based approach for estimating shape and spatially-varying reflectance. In *Proc. ICCP*, pages 1–9, 2015. 4

[18] M. Jaderberg, K. Simonyan, A. Zisserman, et al. Spatial transformer networks. In *Advances in Neural Information Processing Systems*, pages 2017–2025, 2015. 2

[19] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu. Spatial transformer networks. *CoRR*, abs/1506.02025, 2015. 2

[20] K. Kim, J. Gu, S. Tyree, P. Molchanov, M. Nießner, and J. Kautz. A lightweight approach for on-the-fly reflectance estimation. *CoRR*, abs/1705.07162, 2017. 2

[21] T. D. Kulkarni, W. Whitney, P. Kohli, and J. B. Tenenbaum. Deep convolutional inverse graphics network. *CoRR*, abs/1503.03167, 2015. 2

[22] M. M. Loper and M. J. Black. OpenDR: An approximate differentiable renderer. In *Proc. ECCV*, pages 154–169. Springer, 2014. 2

[23] S. R. Marschner, S. H. Westin, E. P. F. Lafortune, K. E. Torrance, and D. P. Greenberg. Image-based BRDF measurement including human skin. In D. Lischinski and G. W. Larson, editors, *Proc. Rendering Techniques*, pages 131–144, Vienna, 1999. Springer Vienna. 3

[24] W. Matusik, H. Pfister, M. Brand, and L. McMillan. A data-driven reflectance model. *ACM Transactions on Graphics*, 22(3):759–769, July 2003. 4

[25] W. Matusik, H. Pfister, M. Brand, and L. McMillan. Efficient isotropic brdf measurement. In *Proc. Eurographics Workshop on Rendering*, EGRW '03, pages 241–247, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association. 3

[26] O. Nalbach, E. Arabadzhiyska, D. Mehta, H. Seidel, and T. Ritschel. Deep shading: Convolutional neural networks for screen-space shading. *CoRR*, abs/1603.06078, 2016. 2

[27] J. B. Nielsen, H. W. Jensen, and R. Ramamoorthi. On optimal, minimal brdf sampling for reflectance acquisition. *ACM Transactions on Graphics (TOG)*, 34(6):186:1–186:11, November 2015. 3, 4, 5, 6, 7, 8

[28] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional gen-

erative adversarial networks. *CoRR*, abs/1511.06434, 2015. 2

[29] K. Rematas, T. Ritschel, M. Fritz, E. Gavves, and T. Tuytelaars. Deep reflectance maps. *CoRR*, abs/1511.04384, 2015. 2

[30] E. Richardson, M. Sela, R. Or-El, and R. Kimmel. Learning detailed face reconstruction from a single image. *CoRR*, abs/1611.05053, 2016. 2

[31] S. M. Rusinkiewicz. A new change of variables for efficient brdf representation. In *Rendering techniques 98*, pages 11–22. Springer, 1998. 3

[32] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 7

[33] Y. Tang, R. Salakhutdinov, and G. Hinton. Deep lambertian networks. *arXiv preprint arXiv:1206.6445*, 2012. 2

[34] A. Tewari, M. Zollhöfer, H. Kim, P. Garrido, F. Bernard, P. Pérez, and C. Theobalt. Mofa: Model-based deep convolutional face autoencoder for unsupervised monocular reconstruction. *CoRR*, abs/1703.10580, 2017. 2

[35] S. Tulsiani, T. Zhou, A. A. Efros, and J. Malik. Multi-view supervision for single-view reconstruction via differentiable ray consistency. *CoRR*, abs/1704.06254, 2017. 2

[36] D. R. White, P. Saunders, S. J. Bonsey, J. van de Ven, and H. Edgar. Reflectometer for measuring the bidirectional reflectance of rough surfaces. *Applied optics*, 37(16):3450–3454, 1998. 3

[37] J. Wu, T. Xue, J. J. Lim, Y. Tian, J. B. Tenenbaum, A. Torralba, and W. T. Freeman. Single image 3d interpreter network. In *European Conference on Computer Vision (ECCV)*, 2016. 2

[38] A. Zhmoginov and M. Sandler. Inverting face embeddings with convolutional neural networks. *CoRR*, abs/1606.04189, 2016. 2

[39] J. Zienkiewicz, A. Davison, and S. Leutenegger. Real-time height map fusion using differentiable rendering. In *Proc. IROS*, pages 4280–4287, 2016. 2