Foot Depth Map Point Cloud Completion using Deep Learning with Residual Blocks

Nolan Lunscher University of Waterloo, ON, Canada
John Zelek University of Waterloo, ON, Canada

Abstract

Fit is extremely important in footwear as fit largely determines performance and comfort. Current footwear fit estimation mainly uses only shoe size, which is extremely limited in characterizing the shape of a foot or the shape of a shoe. 3D scanning presents a solution to this, where a foot shape can be captured and virtually fit with shoe models. Traditional 3D scanning techniques have their own complications however, stemming from their need to collect views covering all aspects of an object. In this work we explore a deep learning technique to compete a foot scan point cloud from information contained in a single depth map view. We examine the benefits of implementing residual blocks in architectures for this application, and find that they can improve accuracies while reducing model size and training time.

1 Introduction

Foot shape is complex and to properly characterize requires many measures of lengths, widths, girths and angles [1]. The shoe size system currently in use typically only relies on a single length measurement which does not do enough to characterize shape. 3D scanning can be used to capture more information about a persons foot, which can then be used to better match with a well fitting piece of footwear. A number of scanning solution have already been developed including the Vorum Yeti[1] and the Volumental scanner[2], however they are not commonly used and tend to be expensive or difficult to operate.

In recent years, RGBD cameras have become popular devices in a number of applications including their use in low cost 3D scanners, as they are able to provide a fast and accurate depth map. The typical process of 3D scanning requires the capture of images from many positions that cover the entire surface of an object. Capturing these images typically requires either a moving camera [2] or camera array system [3, 4]. In the application of scanning a person, the faster camera arrays are a better solution as people are not rigid and will often move during a lengthy scan. In order to further simplify and reduce the costs of the scanning process, fewer cameras with fewer capture steps would be ideal. Towards this goal, learning models can be used to extrapolate from limited inputs, such as single images, and synthesize object images from alternate views [5, 6, 7], and even depth map images containing 3D shape [8]. In this way, synthesized depth maps can be used to form a 3D point cloud of an object from a limited viewpoint.

In our case, our goal is to produce a complete 3D foot point cloud from a single depth map. When the input depth map is taken from the profile of the foot, nearly half the points required are already present. Synthesizing a second view that covers the opposite surface of the foot is sufficient to produce a near complete point cloud of the foot surface. In this work we expand on our previous work [9], and explore the impact of using different network architectures. We implement two new architecture utilizing residual blocks, which are a fundamental component of ResNet [10], one of the most powerful deep networks to date.

2 Methods

We train a deep neural network to take as input a depth map of the profile view of a foot, and to synthesize a depth map of the same foot from the opposite viewpoint [9]. The points from this second depth map can be merged with the points from the input depth map to create a near complete 3D point cloud. To simplify the process of registering the two sets of points, we synthesize the points from the second depth map from the same camera pose as the input. Figure 1 illustrates this process.

[1] vorum.com/footwear/yeti-3d-foot-scanner
[2] volumental.com

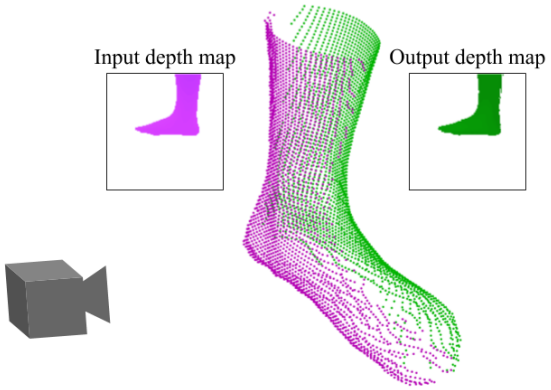

Fig. 1: Depth map input/output configuration. Purple: points from the input depth map, Green: points from the synthesized output depth map.

Table 1: Camera pose parameters for the network input.

Pose Parameter	Value Range	Step
Radius (mm)	640 to 700	15
Azimuth (deg)	70 to 110 or 250 to 290	1
Elevation (deg)	-20 to 20	1
Roll (deg)	-5 to 5	1

2.1 Dataset

Due to a lack of freely available foot scan data, a depth map dataset was constructed from renderings of body models from MPII Human Shape [11] built from the CAESAR database [12]. The feet of each body model were isolated, and virtually scanned at various azimuth angles, elevation angles, roll angles and at varying radii around the profile of a foot. Depth map images are rendered using Panda3D[3] at a resolution of 128x128, using random camera poses in the ranges described in Table 1, with additional Kinect noise applied [13]. A total of 8602 foot objects were available, which were split into two sets, 80% for training and 20% for testing.

2.2 Network Architectures

Our first architecture, denoted Net1, closely follows that used in previous RGBD view synthesis works [8], and was used in our previous works [9]. This network encodes information all the way down to a compact 1 dimensional representation before decoding the output depth map. This network does not use any batch normalization [14].

Our second architecture, denoted Net2, uses an architecture similar to ResNet-34 [10] as its encoder and decoder. Sixteen residual blocks of two 3x3 convolutional layers are used in both the encoder and decoder, with projection shortcuts used to match dimensions. Up-scaling in the decoder's residual blocks is done using strided deconvolutional [15] layers. We do not include any fully connected layers in Net2 in order to save parameters, as well as to allow for more 2D spatial information to pass through the network. Batch normalization was used in all layers except for the last three residual blocks and the final deconvolution. We found that batch normalization on the final layers made it difficult for the output depth map pixels to take on the necessary values. This network contains less parameters than Net1, but contains far more processing at each representation size.

Our third architecture, denoted Net3, uses less residual blocks than Net2, and does not down scale its representations to as small a size, allowing it to have far less parameters than either Net1 or Net2. Here we again use batch normalization everywhere except

[3] panda3d.org

Table 2: Network architectures used. Abbreviations used: Convolution (c), Deconvolution (d), Stride of 2 (/2), Fully connected (fc), Residual Block (res), Residual Block with deconvolutional upscaling (dres).

Output Size	Net1	Net2	Net3
128x128	Input	Input	Input
64x64	c 5x5, 32, /2	c 7x7, 16, /2	c 7x7, 16, /2
	-	res, 16	res, 16
	-	res, 16	res, 16
	-	res, 16	res, 16
32x32	c 5x5, 32, /2	res, 32, /2	res, 32, /2
	-	res, 32	res, 32
	-	res, 32	res, 32
	-	res, 32	res, 32
16x16	c 5x5, 64, /2	res, 64, /2	res, 64, /2
	-	res, 64	res, 64
	-	res, 64	res, 64
	-	res, 64	-
	-	res, 64	-
	-	res, 64	-
8x8	c 3x3, 128, /2	res, 128, /2	-
	-	res, 128	-
	-	res, 128	-
4x4	c 3x3, 256, /2	-	-
1024	fc	-	-
4096	fc	-	-
8x8	d 3x3, 128, /2	res, 128	-
	-	res, 128	-
16x16	d 3x3, 64, /2	dres, 64, /2	-
	-	res, 64	-
	-	res, 64	-
	-	res, 64	-
	-	res, 64	res, 64
	-	res, 64	res, 64
32x32	d 5x5, 32, /2	dres, 32, /2	dres, 32, /2
	-	res, 32	res, 32
	-	res, 32	res, 32
	-	res, 32	res, 32
64x64	d 5x5, 32, /2	dres, 16, /2	dres, 16, /2
	-	res, 16	res, 16
	-	res, 16	res, 16
	-	res, 16	res, 16
128x128	d 5x5, 1, /2	d 7x7, 1, /2	d 7x7, 1, /2
	Output	Output	Output
Parameters	9,286,977	2,666,785	581,153

on the last three residual blocks and the final deconvolution. In all three architectures, ReLU activations are used in all layers except for the final deconvolution, which uses tanh. Our network architectures are shown in Table 2.

2.3 Implementation Details

We implement our networks in Tensorflow[4] on a Linux machine with an Nvidia K80 GPU. Training was done with mini-batch sizes of 64 and the Adam optimizer [16] in all cases. We used the mean L1 difference between the output and the ground truth pixels as the Loss function. Net1 was trained with a learning rate of 5e-5, while Net2 and Net3 used 5e-4.

In order to construct a complete point cloud using our networks, we first project both the input depth map and output depth map to world coordinate points. We then post process the points using MATLAB's *pcdenoise* function and 3D cropping. The combination of the input depth map points, and the synthesized depth map points create a near complete overall point cloud.

[4] tensorflow.org

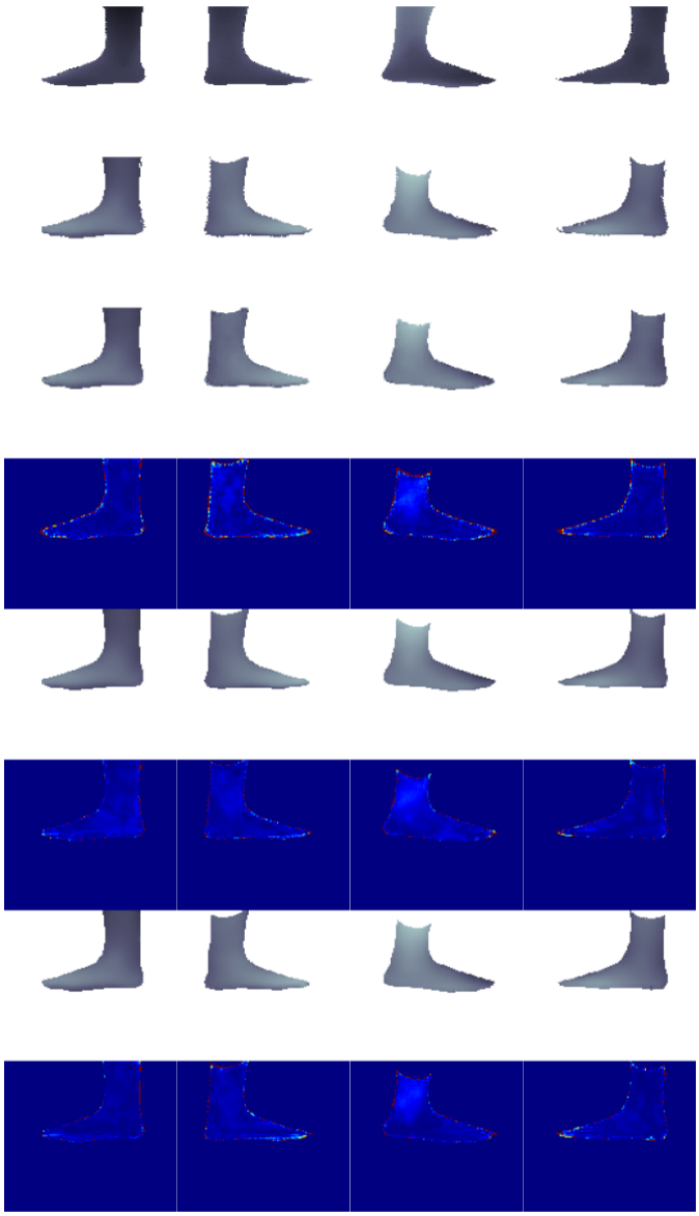

Fig. 2: Depth map samples. Row 1: input depth map, Row 2: ground truth, Row 3: Net1 output depth map, Row 4: Net1 depth map error, Row 5: Net2 output depth map, Row 6: Net2 depth map error, Row 7: Net3 output depth map, Row 8: Net3 depth map error.

3 Results

For each foot object in the test set, we generate 64 views to measure network performance. We evaluate performance with two metrics. Our first metric measures the depth map error using a the pixel-wise L1 difference between the synthesized depth map and the ground truth. Samples of depth maps from each architecture are shown in Figure 2 along with their distribution of error across the image. We additionally measure the accuracy of the 3D point cloud produced using the synthesized depth map. We use a metric similar to that used by Luximon et al. [17]. Our metric is a two way nearest neighbour euclidean distance measure. Were each point in the synthesized depth map is compared using its euclidean distance to the nearest point in the ground truth and visa-versa. This distance is averaged over the depth maps to produce an overall point cloud error. Samples of synthesized point clouds are shown in Figure 3. Our results are shown in Table 3.

As can be seen, Net2 and Net3 both significantly out-perform Net1 in both metrics, while requiring less training time. The use of residual blocks with batch normalization allows for higher learning rates reduced training time. The lack of fully connected layers allows for smaller models with less parameters, while the residual block skip connection allows for deeper networks, with more processing at each representation size.