End-to-End 3D Bézier Surface Reconstruction Using Convolutional Neural Networks

Miki (David) Pokryvailo

August 12, 2019

ABSTRACT

Surface reconstruction is an important problem that lies in the intersection of computer graphics and computer vision. The challenge is to reconstruct a surface given its incomplete representation, which may be in the form of a point cloud, an image, or several images. State-of-the-art algorithms are able to reconstruct complex surfaces from raw images, but are usually very complex, and/or computationally intensive. Given the limited compute and time resources for this research, I focused on reconstructing 3D Bézier surfaces from single images, using convolutional neural networks (CNNs). This approach showed promise, as the network was able to learn the shapes of the surfaces, despite having relatively primitive data to learn from.

1 Introduction

Surface reconstruction is a problem with important applications in many fields, including medical imaging, computer animation, and more recently, virtual and augmented reality. Some approaches are able to generate dense point cloud representations of complex objects from single images [1]. However, they use large neural network architectures, which require similarly large amounts of compute resources and rich training data. The authors in [1] used CAD models consisting of 100k surface points as the ground truth for their model. I wanted to see if a CNN could be trained to reconstruct 3D surfaces with limited label features and limited spatial training data. To answer this question, I chose to focus a simpler version of the above problem: reconstructing a 3D Bézier surface by predicting its control points from a single generated grayscale image.

Training networks on computer-generated surface data is not new, but existing approaches have tackled the problem of estimating the output of the blending function directly, given its parameters [2].[1] The authors were able to accurately model the blending function of a B-spline using a relatively simple, fully connected 4-layer network. Given that this problem is not particularly challenging to solve using neural networks, and its lack of practical applications, an end-to-end approach for predicting surfaces was chosen instead.

The objective of this research was to find if CNNs could predict the control points of a 3D Bézier surface from such limited data as a single grayscale image. Since input data was was not information-rich, high accuracy was not expected, but rather a general ability to predict surface shape. CNNs were chosen specifically for their well-known ability to learn spatial features [3].

There were several subproblems to tackle in order to answer this question. First, a loss function had to be formalized in order to establish the kind of labels needed. Then, surface image data and their corresponding labels needed to be generated, with enough variance to ensure the network does not simply learn a small number of very similar surfaces. Finally, a suitable neural network architecture had to found and trained to effectively predict control points from the image input.

[1]It is possible (likely) that research very similar to this has been done, but I could not find any such papers.

2 Mathematical Background

2.1 Bézier Surfaces

A Bézier surface is a parametric surface defined by a number of fixed points called *control points* [4]. A two-dimensional Bézier surface, which is used in this study, defined by a set of $(n+1)(m+1)$ control points, can be defined by two parameters, u and v, and described by the following equation:

$$\mathbf{p}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)\ B_j^m(v)\ \mathbf{k}_{i,j}$$

where $B_i^n(u)$ is a Bernstein polynomial defined by:

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

and u and v are defined on the interval $[0,1]$.

Bézier surfaces were chosen as the data for this study because they "stretch" towards their control points, making them visually intuitive, and therefore good candidates for a CNN, which has historically been used on image data.

2.2 Neural Network Overview

Before CNNs can be explained, one must understand the concept of supervised learning and multilayer perceptrons (MLP).

Supervised learning is a method of training that works by supplying the learning function with an input and a ground truth (sometimes called a label), and minimizing a predefined loss function that measures the difference between the predictions and the labels.

MLPs are sometimes referred to as "vanilla" neural networks, as they are the most basic and traditional form of a neural network [5]. They consist of layers of neurons, which use an activation function (usually nonlinear) to control their output. There is a single input layer, one or more hidden layers, and one output layer. An MLP "learns" by iteratively optimizing each neuron's weight to minimize a specified loss function.

MLPs usually learn with a variation of the gradient descent algorithm, a popular iterative optimization algorithm [6]. Since the number of neurons and layers can get very large, a process called backpropagation is used to efficiently calculate the gradients of the network's loss function with respect to its parameters [7], which are then used by the optimization algorithm to perform necessary calculations. The derivation for backpropagation is beyond the scope of this paper, however, for gradient descent, a simplified derivation will be shown.

Gradient descent is an intuitive optimization algorithm that aims to minimize a cost function by iteratively stepping in the opposite direction of the function's gradient. A derivation of the gradient descent algorithm for a commonly used loss function, Mean Squared Error (MSE) will be shown below [8], with a simplified prediction function of linear regression using two weights. In practice, neural networks often have millions of weights, but the general derivation is the same.

Consider the MSE loss function:

$$J(w_0, w_1) = \frac{1}{m}\sum_{i=1}^{m}(P_w(x_i) - y_i)^2$$

where w_0 and w_1 are the weights, m is the number of training samples, x_i is the ith sample, P_w is the prediction function, in this case linear regression, defined by:

$$P_w(x) = w_0 + w_1 x$$

and y_i is the ith ground truth.

The aim is to minimize $J(w_0, w_1)$, which is done by calculating $\frac{\partial}{\partial w_i}J(w_0, w_1)$ for each w_i, and moving w_i in the opposite direction of the its partial derivative scaled by a specified amount α, also known as the *learning rate* [9]:

$$\begin{aligned} w_0 &:= w_0 - \alpha\frac{\partial}{\partial w_0}J(w_0, w_1) \\ w_1 &:= w_1 - \alpha\frac{\partial}{\partial w_1}J(w_0, w_1) \end{aligned}$$

The derivation of $\frac{\partial}{\partial w_0}$ is shown below:

$$\begin{aligned} J(w_0, w_1) &= \frac{1}{m}\sum_{i=1}^{m}(P_w(x_i) - y_i)^2 && (1) \\ \frac{\partial}{\partial w_0}J(w_0, w_1) &= \frac{\partial}{\partial w_0}\left(\frac{1}{m}\sum_{i=1}^{m}(P_w(x_i) - y_i)^2\right) && (2) \\ &= \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_0}(P_w(x_i) - y_i)^2 && (3) \\ &= \frac{2}{m}\sum_{i=1}^{m}(P_w(x_i) - y_i) && (4) \end{aligned}$$

Deriving (3) from (2) requires use of the sum rule:

$$\frac{d}{dx}\sum u = \sum\frac{du}{dx}$$

And deriving (4) from (3) requires use of the chain rule and derivation of $\frac{\partial P}{\partial w_0}$, which is equal to 1.

2.3 Convolutional Networks

Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers [10]. A convolution on two functions produces a third function that expresses how the shape of one function is modified by the other [11]. With function f and g, this can be expressed as:

$$(f * g)(t) \triangleq \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\, d\tau$$

For image data, which is a 2D grid of pixels, a 2D matrix, or *kernel*, "slides" over the input image, performing convolutions to extract features, such as vertical/horizontal edges, curves, etc. The size of the kernel and the amount it slides (called the *stride*), can be adjusted to detect more or less detailed features, and to decrease the input size to the next layer, which reduces the number of parameters of the network and its training time. Learning with a CNN consists of adjusting the weights of the kernel to minimize the loss function.

3 Training Details

3.1 Data Generation

For training and testing data, $40{,}000$ grayscale images of Bézier surfaces were generated. To generate random surfaces, a 3x3 grid of control points was generated by the following method. The uniform distribution over $[0, 2)$ was sampled for x and y dimensions, to generate dx and dy, which were incremented to the last row/column value, respectively, to generate a grid. For the z dimension the uniform distribution over $[0, 3)$ was sampled, without incrementing to generate surface height. Uniform sampling over these intervals provided adequate variation in surfaces, without producing surfaces with too many irregularities.

The resulting grid of 9 control points was used as the ground truth, and also as the input to the algorithm described in section 2.1 to generate a Bézier surface, with n uniformly distributed u and v over $[0, 1]$. n was a random integer sampled uniformly over $[50, 75]$. Randomizing the density of the Bézier parameters was done in order to provide more noisy input to the CNN, and hopefully result in a more robust network.

(a) 50 values per parameter (b) 75 values per parameter

Figure 1: The same Bézier surface, with different parameter densities.

The image data was loaded into a 120x120 array, and cropped to a size of 110x110. This size was chosen because it retained enough detail, and kept training time adequately low. Finally, the image was normalized by diving each pixel value by 255.

3.2 Network Architecture

The CNN used starts with 2 convolutional layers, both using a 3x3 kernel. The first convolutional layer has 128 filters, with a stride of 1, and then second has 64 filters, with a stride of 2. These layers are followed by a max-pooling layer [12], used to decrease network size and training time. This is followed by another convolutional layer, with 32 filters, a 5x5 kernel, and a stride of 1, and another max-pooling layer. Finally, there is a fully-connected layer with 128 units, followed by a dropout layer with a dropout rate of 0.2 to prevent overfitting [13]. The output layer consists of 27 units, one for each dimension of the

9 three-dimensional control points. The loss function minimized in training is the MSE function, derived previously.

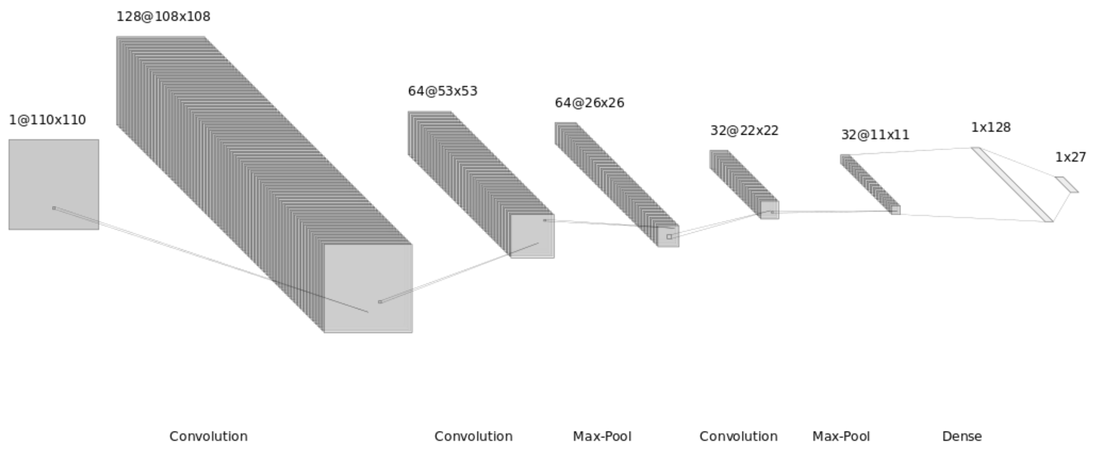


Figure 2: CNN architecture used.

This architecture was picked after trying several architectures, ranging in layer size and depth. Architectures with more layers and/or parameters showed overfitting. and simpler architectures had high loss. To diagnose overfitting, training and validation losses were recorded, and if validation loss started increasing while training loss decreased, this was considered a sign of overfitting, and training was stopped.

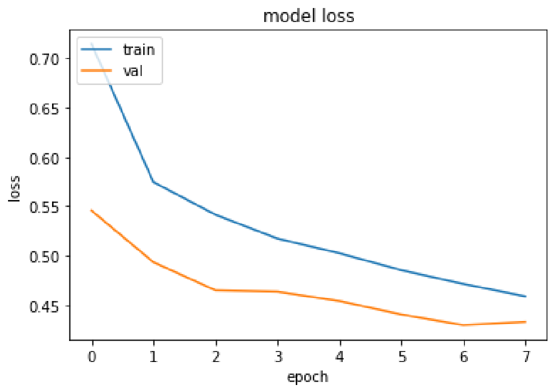
3.3 Training Parameters

To train the model, the Adam optimizer [14] was used with a learning rate (α) of $1\mathrm{e}{-3}$ was used until the model exhibited signs of overfitting, as described in the previous section. Then, the model was fine-tuned with a learning rate of $1\mathrm{e}{-5}$. The maximum number of epochs was set to 30, but the model started overfitting much earlier. Finally, 20% of the data was used for validation, while the rest was used for training.

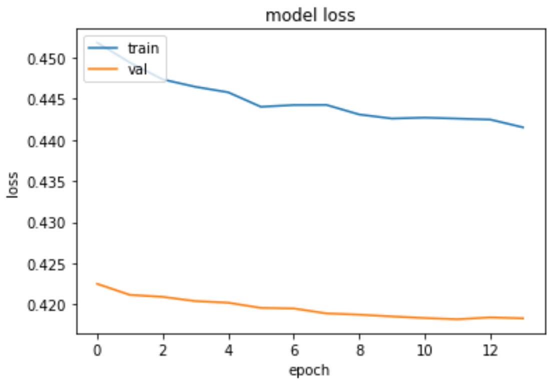
4 Results

4.1 Model Loss

The final model achieved a validation MSE of 0.41821, after initial training for 7 epochs, and fine-tuning for 13 epochs. The fine-tuning decreased loss very little, but took quite a bit of time to finish. Other fine-tuning approaches, perhaps such as using an adaptive learning rate are worth exploring in the future. Training history for both initial training and fine-tuning is shown below.



(a) Initial training ($\alpha = 1e{-3}$)



(b) Fine-tuning ($\alpha = 1e{-5}$)

Figure 3: Model loss over time

4.2 Surface Predictions

Several plots of surfaces and control points the model predicted are shown below. Predicted surfaces were generated using the same algorithm as the ground truth surfaces, simply using the predicted control points as input. The true control points are red, and the model's predicted control points are green. The true surfaces are colored black, and the model's predicted surfaces are orange.

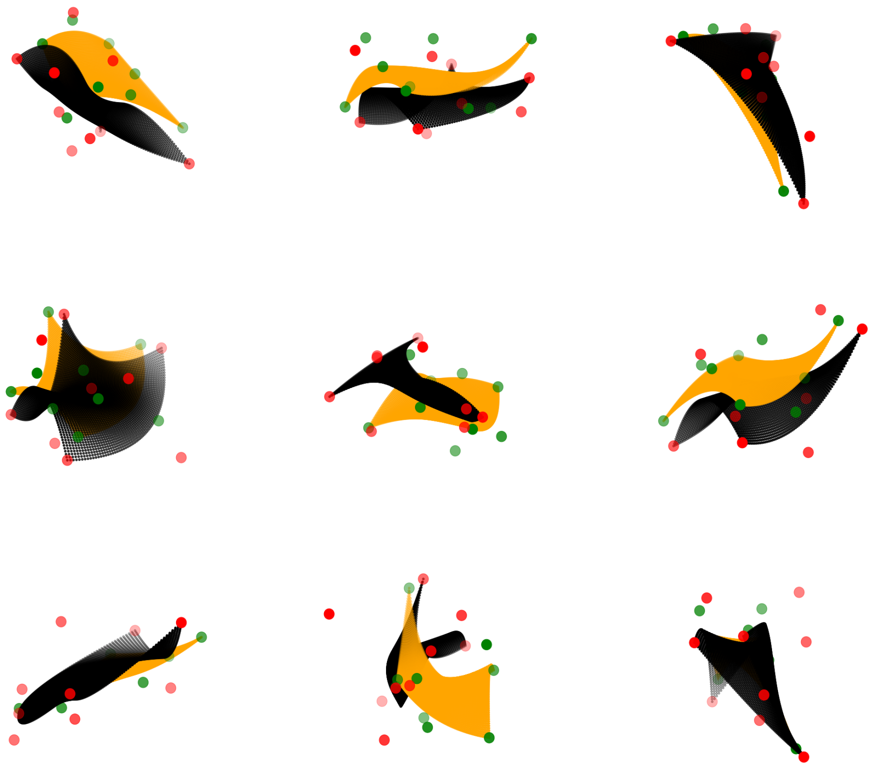


Figure 4: Model predictions vs. ground truth.

One can clearly see the model's predictions have a similar shape to the actual surfaces (especially in the top and middle-right figures), demonstrating that the model has the ability to infer control point location from raw image data. It is also clear however, that the model's prediction are far from exact, and are not usable in applications that need to accurately map images to models. Sometimes, the model predictions are very different from the ground truth, as seen in the middle figure.

Note: A larger version of the above figure is available as part of this paper's resources.

5 Conclusion

This study has achieved its objective, which was to show that CNNs are able to approximately reconstruct 3D Bézier surfaces from limited raw pixel data. It was shown that even a relatively simple CNN was able to infer nontrivial geometric information from the limited raw pixel data it was provided. Since surfaces were generated randomly, they were often seen from a perspective obscuring important information. Nevertheless, the hypothesis that predictions would not be highly accurate, but rather generally shape-preserving, was confirmed.

For further research, it would be interesting to compare performance of this CNN with another that is provided several perspectives of the same surface as input. My hypothesis is that a significant limiting factor of this CNN is the lack of features its input contains. In addition, exploring the usage of Generative Adversarial Networks (GANs) [15], instead of CNNs would be a worthwhile endeavor. GANs have showed much promise in generating image data, as demonstrated in studies such as [15] and [16].

References

[1] Chen-Hsuan Lin, Chen Kong, and Simon Lucey. Learning efficient point cloud generation for dense 3d object reconstruction. *CoRR*, abs/1706.07036, 2017.

[2] P Gu and X Yan. Neural network approach to the reconstruction of freeform surfaces for reverse engineering. *Computer-Aided Design*, 27(1):59 – 64, 1995.

[3] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, Aug 2017.

[4] Wikipedia contributors. Bézier surface — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=B%C3%A9zier_surface&oldid=866864194`, 2018. [Online; accessed 8-August-2019].

[5] Wikipedia contributors. Multilayer perceptron — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Multilayer_perceptron&oldid=901974638`, 2019. [Online; accessed 8-August-2019].

[6] Wikipedia contributors. Gradient descent — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Gradient_descent&oldid=907632275`, 2019. [Online; accessed 8-August-2019].

[7] Wikipedia contributors. Backpropagation — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Backpropagation&oldid=909631210`, 2019. [Online; accessed 8-August-2019].

[8] Chris McCormick. Gradient descent derivation, Mar 2014.

[9] Wikipedia contributors. Learning rate — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Learning_rate&oldid=898255673, 2019. [Online; accessed 9-August-2019].

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[11] Wikipedia contributors. Convolution — Wikipedia, the free encyclopedia, 2019. [Online; accessed 12-August-2019].

[12] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In Konstantinos Diamantaras, Wlodek Duch, and Lazaros S. Iliadis, editors, *Artificial Neural Networks – ICANN 2010*, pages 92–101, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[13] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

[15] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

[16] Dong Nie, Roger Trullo, Jun Lian, Caroline Petitjean, Su Ruan, Qian Wang, and Dinggang Shen. Medical image synthesis with context-aware generative adversarial networks. In Maxime Descoteaux, Lena Maier-Hein, Alfred Franz, Pierre Jannin, D. Louis Collins, and Simon Duchesne, editors, *Medical Image Computing and Computer Assisted Intervention - MICCAI 2017*, pages 417–425, Cham, 2017. Springer International Publishing.