Deep Learning & Computer Graphics: The making of 3D Models by Using Convolutional Neural Networks

Emmanuel Rochette
August 6, 2017

Abstract

The goal of this work was to automate the creation of 3D models with the help of deep learning methods. A loop system was developed to achieve this. Four 3D models are thus created in Blender, the 3D computer graphics software.

1. Introduction

Convolutional neural networks (CNN) are used for image classification [1]. Like other neural networks, they consist of an input layer and an output layer, as well as multiple hidden layers. They are made up of neurons that have learnable weights and biases. But although regular neural networks do not make any explicit assumptions on their inputs, CNN do: they assume their inputs to be images. Therefore, CNN have their first layers to be different than regular neural networks; they use convolutional and pooling layers. Without going into the details, convolutional layers are used to extract features from the input image, and the pooling layers are used to reduce the dimensionality of those features. CNN are used for image recognition and classification. For a given image, CNN will recognize an object in it with a certain probability. This is possible since CNN use a softmax function as its loss function. In Figure 1, where examples can be seen, the highest probability prediction is shown, but CNN also give a probability for every other object it was trained to recognize in pictures. This work will use the complete list of probability predictions, instead of just the highest one.

Also, since our goal is to explore the creation of 3D models with the use of CNN, rather than a study on CNN, the image classifications for this work will be perform mostly by the Google Cloud Vision API. In other words, we will outsource the work done for the

image classification. Doing so has one obvious advantage: the access to a powerful deep learning API trained with countless of images.

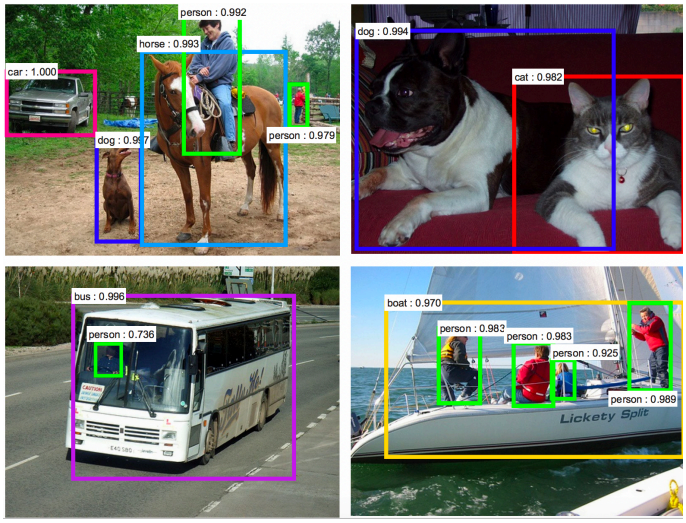


Figure 1. Examples of image classification by CNN.

Blender, also used in this work, is an open-source 3D computer graphics software. It is the environment into which all models are generated. Blender was chosen since it comes with a Python API, which allows one to write a program to create 3D models, instead of doing it manually in the software.

This last feature, combined with the use of convolutional neural networks, are the central pieces of this work. By writing a program that randomly creates millions of 3D models in Blender, and by using CNN to select the ones that represents the desired objects, it was possible to use artificial intelligence to design 3D objects. In a way, it has been shown that a computer can look at the world, and recreate what it sees in 3D.

2. The Deep3D Generator

Blender was chosen for two reasons. First, it has an impressive list of operations that can be used to create 3D models and animations. But more importantly, each of these operations has a Python command associated with it[1]. In other words, it is possible to write a Python program to generate 3D models and animations in Blender. Even more, the program can be executed outside of the Blender software. Thus, it is possible to write a program that can interact with the Blender environment, but also with the rest of the executing computer.

[1] See https://docs.blender.org/api/current/.

This last feature was used to conceive a simple loop system, on which this entire work was built. The loop is a repeating cycle of two main processes, where the first one is executed in Blender, and the second one is executed outside of it. The first process generates or modifies a 3D model in Blender. The second process checks if that 3D model corresponds to the desired object (which specified before the loop started) by using deep learning techniques. This is done outside of Blender, using, at least in our case, the Google Vision API. If Google's classifications successfully identify the 3D model as the desired object, the work done in the first process is saved somewhere in the computer; and if the 3D model is not recognize as being the desired object, nothing is saved. This completes a cycle in the loop system. The alternation of these processes continues until the loop system decides to stop it. In our case, this is when no more operations is left to be performed.

For future reference, this loop system will now be named the Deep3D Generator (DG). The name is chosen to mention the combination of *deep* learning and *3D* modeling. Let's describe it in more details.

The DG takes only two inputs in the beginning: the name of an object to be created in Blender (coffee cup, teddy bear, etc.), and a procedure to follow for the creation of 3D models. This procedure is a list of Python commands to execute in Blender, one after the other. Since no one has the time to write a list of millions of operations, the list is an algorithm, which create the next command to be performed. After executing each of these commands, pictures[2] are taken, and they are sent to Google or Clarifai through their respective API.

In its 3D environment, Blender offers an object called *camera*. As the name suggests, this is a special viewpoint from which pictures (or even films, in the case of animation) can be taken. In Figure 2, the Blender scene is seen through the camera object. When a picture is taken, a .png file is created in a specified directory with the observed scene on it – in this case, it would be a monkey head. The Deep3D Generator can simply look, afterwards, in that directory to find the pictures. Then, it sends them to Google or Clarifai. The classification predictions are returned by the API as a short list of object's names. Like in Figure 1, a probability is given for each object's names. The Deep3D Generator

[2] The number of pictures depends on the number of times the 3D model need to be rotated in front of the camera, to make sure the 3D model looks like the desired object at different angles. For example, the wine glass is symmetric, so it needs less pictures to assure this.

can then check if the name given as input is in the list, and if so, to which probability it is recognize to be it.

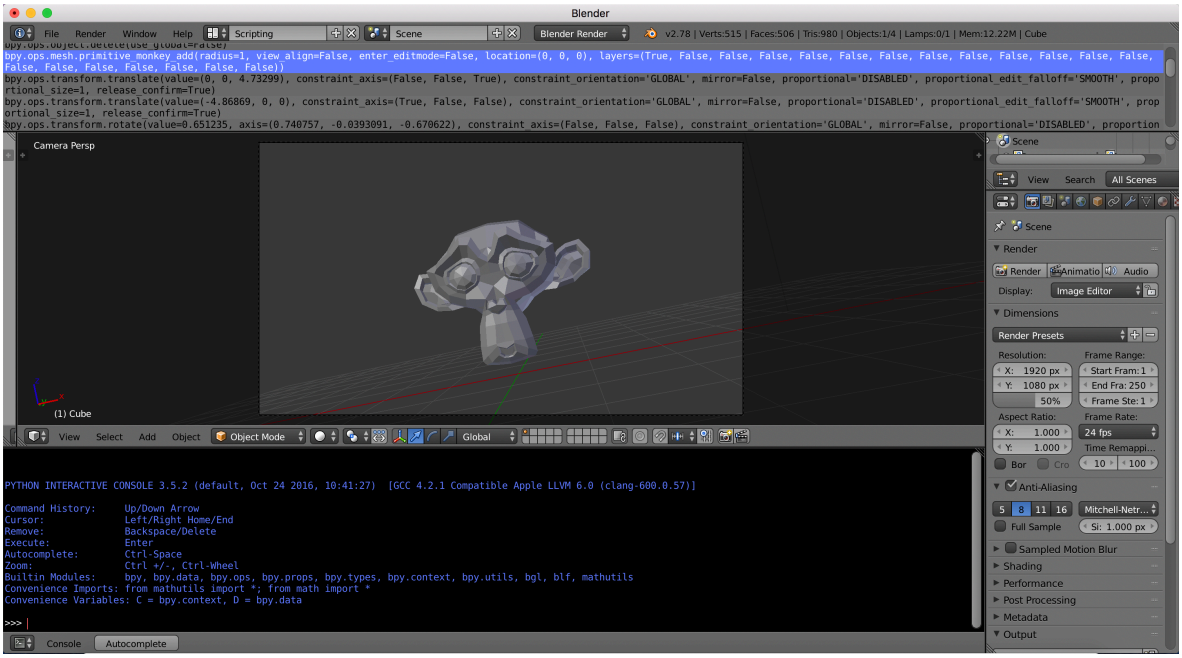


Figure 2. A screenshot of Blender 3D environment in scripting mode. At the bottom of the window, there is the console into which one can write code to create 3D models and animations. This works just like an ordinary terminal. At the top of the window, with an example selected in blue, one can see the commands executed previously. The selected example is the command used to generate the head monkey seen at the center of the screen, where the current state of 3D model is shown. The monkey head comes as a simple command in Blender, since this is the mascot. It is named Suzanne.

If the input name is not in the list, the Deep3D Generator executes the next Python command on the 3D model to modify it accordingly. Pictures are sent again to Google or Clarifai, and the loop continues.

But if the desired object is in the list, there are two cases to consider. If this is the first time the DG creates a successful 3D model, the picture is saved, and the code to generate the model is also saved (which is just the succession of commands that lead to this current model)[3]. If another model was previously successfully created, the probability of the current prediction is compared with the old one. At this point, there are three possible situations: 1) the old one is higher than the current one, 2) they are the same, and 3) the current one is higher than the old one.

[3] See Section 8.1 and 8.2 for examples. These were opened in Sublime Text Editor to have

In the first situation, the current model is simply disregarded as being unsuccessful. In the last situation, the old model is erased in memory, and the new one is saved. But in the second situation, if the probabilities are the same, the current 3D model is saved alongside with the old one. It is now seen that the Deep3D Generator can output more than one 3D model at the end of its execution. From all those saved models, it is also possible to establish the range for which the shape of an object can be transformed. In other words, and as an example, if a teddy bear with a nose of size 1 gets the same probability as another one with a nose of size 2, then it is reasonable to assume that a 3D model with a nose of size 1.5 is also a good model for a teddy bear. We thus get a range in size for the nose, which increase the number of 3D models that can be generated for a teddy bear.

The Deep3D Generator stops when all commands in the procedure list have been executed.

In summary, given as input the name of an object and the procedure to follow to possibly get to that object, the Deep3D Generator will output a set of 3D models that represent that object, for which the code is available. One can use the code to generate as many 3D models as desired, for any uses.

3. Limitations

This project had to face a major limitation: the pricing with the Clarifai was quite high, and the Google vision API was imposing a limit of 20 million predictions per month. It had the consequence of limiting our exploration, since the Deep3D Generator could not be completely random; we had to make sure it eventually reaches the desired 3D model. Otherwise, we might not have anything to show in this paper. It was thus not possible to just let the DG runs forever on a machine, outputting 3D models whenever it finds a realistic one. The objects for which we wished to create a 3D model had to be specified in advance, as well as the list of Python commands to follow to generate models leading to them. Using Keras, the open-source machine learning library, it was tried to build our own convolutional neural networks to escape the limitation. But the resources available were not sufficient to give good results. At least, it is comforting to say that, as a proof of concept, this work really shows the potential for the applications of the Deep3D Generator.

Another small limitation was consciously imposed to save space and increase the speed of execution: no pictures were saved if the current 3D model was not recognize

as being the desired object; only the pictures of successful models were saved. To also increase the speed of execution, only the wine glass presented below was tested with rendering[4], as it takes a lot of computing time.

It is finally important to mention that all figures of 3D models presented in section 4 were not taken by those seen by the DG. For a presentation purpose, the selected pictures by the DG were manually plugged in the API's interface, and a screenshot was taken.

4. Results[5]

4.1 The Coffee cup

The coffee cup was the first experimentation of this project. The 3D model started out as a cylinder and a Bézier curve. The Deep3D Generator did an approximate of 7 million operations on these two objects before arriving to a coffee cup, as shown in Figure 6. Two pictures were taken per operations, with rotation of 180 degrees. As already mentioned, the choice of operations to perform were chosen in advance: a deterministic algorithm was choosing what vertex[6] to select next, where to move it, etc. It was known that ultimately a coffee cup would be generated. But since only 139 models were recognized to be a coffee cup, and since all of them look like coffee cups, the Deep3D Generator can be said to be effective in generating the correct models. (Without the 20 million limitation, the number 139 could be greatly increased by simply dividing the current operations into more precise ones. For example, if an operation moves a vertex from coordinates 1 to coordinates 2, we could divide this operation by having an operation to move it from 1 to 1.5, and another from 1.5 to 2. In this work, the operations were moving the vertices relatively far to economize on the 20 million limit.)

To get a better idea on how the Deep3D Generator started with a cylinder and a Bézier curve, to ultimately created several coffee cups, let's explore Figure 3, 4 and 5 in more details.

---

[4] Rendering is the automatic process of generating texture on 3D models by means of computer programs. Blender offers a large variety of possible renderings. Rendering gives the glossy texture on the surface of a soccer ball, for example, or the glass property of wine glass, as seen in section 4.3.

[5] Having no prior experience in 3D computer graphics, all 3D models developed below were taking from online videos (http://www.littlewebhut.com/blender).

[6] A vertex (plural vertices) is a point on the surface of a 3D object. More precisely, it is the point where one or more lines meet. See Figure 3 for examples of how it can be used.

In Blender, the cylinder starts with 64 vertices. Once selected, they can all be translated through space. It is possible to move only one of them; but more frequently, they are selected and moved a few at a time.

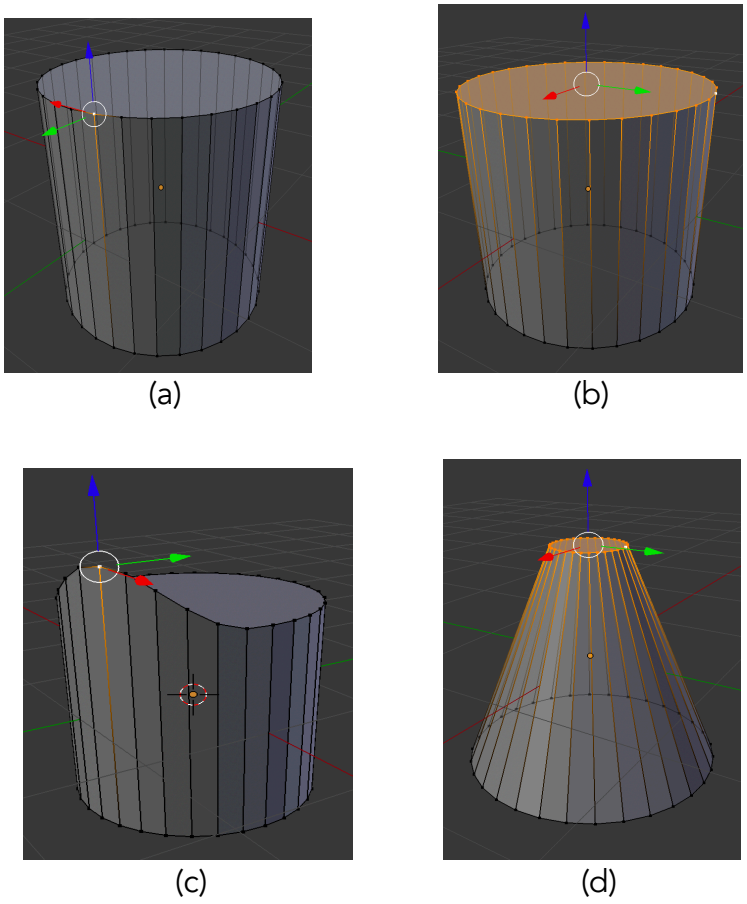
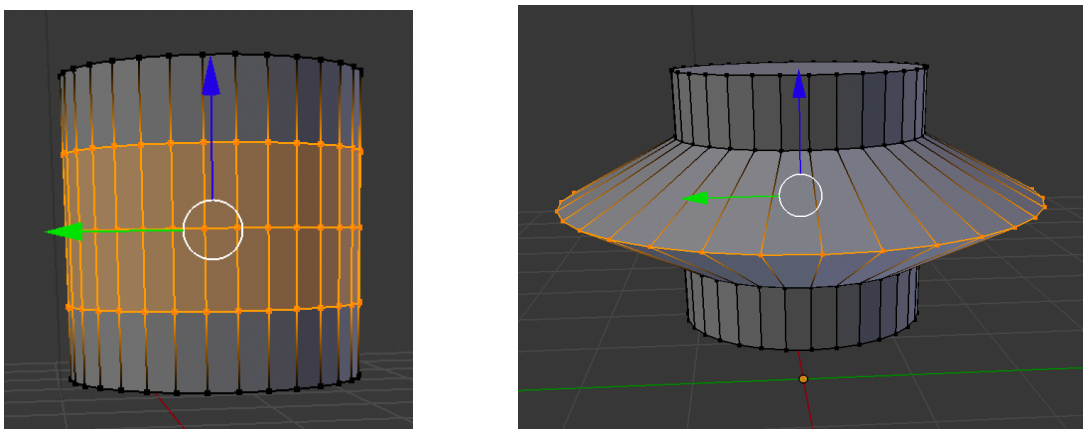


Figure 3. In (a), a cylinder is shown with one of its vertices selected. In (b), all the top vertices are selected at once. With the vertices selected as in (b), one can perform transformations by either translating them through space, or even deleting them if needed. In (c) the vertex in (a) is moved up, and in (d), all the vertices selected in (b) are moved closer together.

It is also possible to add vertices to an object, and use them to modify it. For the resulting coffee cup shown in Figure 6, the Deep3D Generator had to add three rings of vertices to the cylinder, as it is shown in Figure 5-a.



(a) (b)

Figure 4. In (a), three circles of vertices have been added to the cylinder. An example of how these vertices can be used to modify the 3D model is shown in (b).

The Bézier curve is different: it has no vertices. Only its two ends can be selected and translated in space – in fact, the Bézier curve is the line connecting these two points.

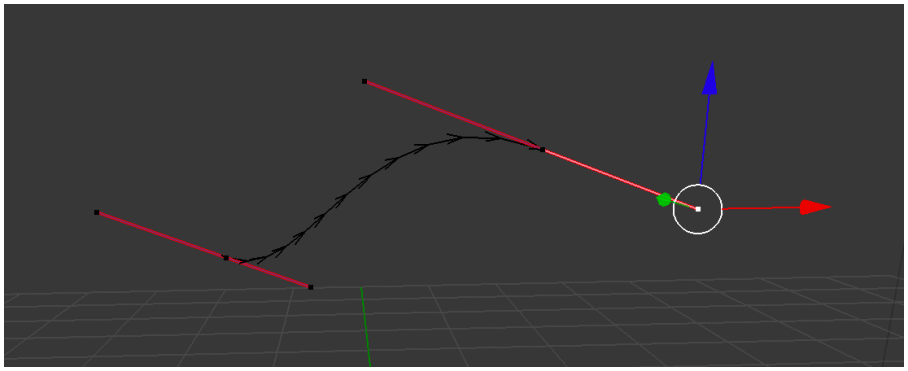
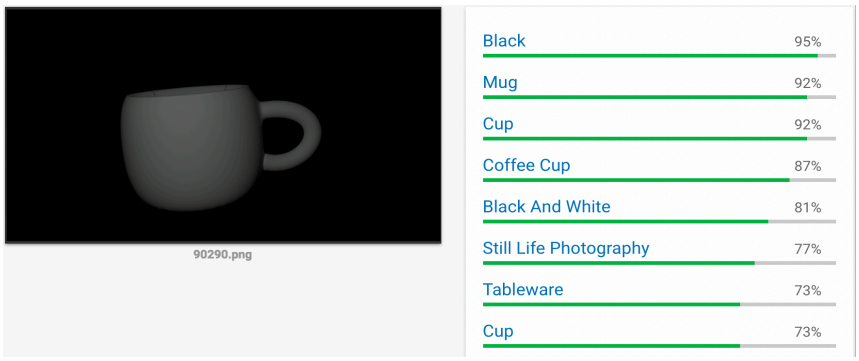
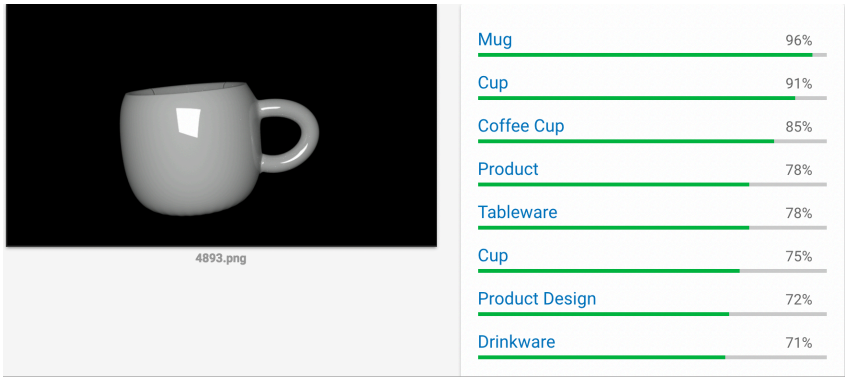


Figure 5. A Bézier curve. There is no vertex. Only the two ends of the curve can be selected through a pink line with three dots. Each dot can be selected and translated in space. The Bézier curve will follow.

The Bézier curve will ultimately be the trajectory followed by another cylinder, which became the handle of the cup. The first cylinder was transformed to become the container of the cup.



(a)



(b)

Figure 6. 3D models and Google Vision API predictions for a coffee cup (a) before rendering and (b) after rendering. See the Appendix for a view from above.

As it can be seen in Figure 6, the Google Vision API did not need the rendering of the cup to make the correct prediction. This is a very good thing since, as it has been said, rendering is really demanding energetically. The only operation applied to the cup, before the pictures are taken, is the smoothing operation. It only smooths the surface of the model to give it a uniform texture.

From this first experimentation, it has been shown that the Deep3D Generator can correctly generate the shape of an object in 3D. Section 4.4 will present a more complex example.