POLITECNICO DI MILANO
MSc in Computer Science and Engineering
Department of Electronics, Information and Bioengineering

Parametric Flower 3D Model Retrieval from Single Image with Deep Learning and Procedural Modeling

Supervisor: Prof. Marco Gribaudo
Co-Supervisor: Dr. Erica Stella

Master Thesis by:
Luciano Mariscalco, ID 914907

Academic Year 2019-2020

Abstract

With Augmented Reality (AR) and Virtual Reality (VR) technologies becoming more prevalent in entertainment, manufacturing, architectural design, training and education, the needs for 3D content are impressively growing.

In this work we tackle the problem of generating a modifiable 3D model from a real instance of an object belonging to particular wide and complex category: flowers. This category has been selected because of its major usage in the main application domains of 3D modeling, and also because its level of complexity offers a good challenge for our study.

Our strategy is to predict a parametric representation of the flower exploiting recent deep learning techniques and a procedural modeling tool (PMT) that we specifically designed for this work.

We intend to build the training dataset for the neural network directly from our flower PMT, relying also on techniques such as High Dynamic Range Imaging (HDRI) and Image Based Lighting (IBL) to add realism to the image and to augment the data. With these techniques we were able to generate as many realistic images of flowers as we want.

Once the whole network is trained, given an image of a flower, we should be able to generate its corresponding 3D model. Such a functionality could be useful in 3D modeling engines as a tool for fast modeling of environment flowers, or in a stand-alone application for educational purposes, or it could be also applied in social media applications as kind of AR filter.

Estratto

Oggi, grazie alle tecnologie della realtà aumentata (AR) e della realtà virtuale (VR) - tecnologie che rivestono un ruolo sempre più importante nel design, nell'ambito della manifattura, nell'architettura, nella didattica e nell'intrattenimento - la domanda di contenuti in 3D è cresciuta a dismisura.

In questo lavoro, abbiamo affrontato il problema di generare un modello 3D modificabile partendo da una istanza reale di un oggetto che appartiene a una categoria particolarmente ampia e complessa: quella dei fiori. Questa categoria è stata selezionata innanzitutto per il suo utilizzo assai diffuso nei principali ambiti di applicazione della modellazione 3D, e poi perché il suo alto livello di complessità offre una sfida significativa per il nostro studio.

La nostra strategia progettuale è stata quella di predire una rappresentazione parametrica del fiore sfruttando le recenti tecniche nel campo del Deep Learning e uno strumento di modellazione procedurale (PMT) che abbiamo progettato specificamente per questo scopo.

Abbiamo costruito il dataset per l'allenamento della rete neurale direttamente dal nostro PMT floreale, facendo uso di tecniche quali l'High Dynamic Range Imaging (HDRI) e l'Image Based Lighting (IBL) per aggiungere realismo alla scena e per aumentare il numero di dati. Queste tecniche ci consentono teoricamente di generare tutte le immagini realistiche di fiori che desideriamo.

Realizzato l'allenamento della rete, data un'immagine di un fiore, dovremmo essere in grado di generare il suo corrispondente modello 3D. Una simile funzionalità potrebbe risultare preziosa nei motori di modellazione 3D sottoforma di tool per la modellazione veloce di fiori per l'ambientazione, oppure potrebbe essere sfruttata in una applicazione a se stante a scopi educativi, oppure ancora potrebbe essere impiegata rella realizzazione di un filtro di tipo AR nelle applicazioni social.

Ringraziamenti

Mi è doveroso dedicare questo spazio del mio elaborato alle persone che hanno contribuito, con il loro instancabile supporto, alla realizzazione dello stesso.

Innanzitutto, ringrazio il Prof. Marco Gribaudo e la Dr. Erica Stella, sempre pronti a darmi le giuste indicazioni in ogni fase della realizzazione dell'elaborato, fin dalla scelta dell'argomento.

Grazie ai miei amici Fabio, Luca e Gerlando, per i loro preziosi consigli e il loro supporto, a Giuseppe, per il suo supporto a distanza, e a Giuseppe, costante sostegno nello studio fin dal liceo. Ringrazio anche tutti i miei colleghi universitari di Palermo, che hanno aggiunto brio alla vita universitaria, e tutti le persone che mi hanno sempre supportato.

In ultimo ringrazio mio padre, che ha preso a cuore questo progetto e mi ha seguito e sostenuto con interesse ad ogni passo, mia madre, mia sorella e mio fratello, ognuno di loro ha contribuito affinché il mio percorso di studi e questo lavoro finale venissero portati a termine con successo.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With Augmented Reality (AR) and Virtual Reality (VR) technologies becoming more prevalent in entertainment, manufacturing, architectural design, training and education, the needs for 3D content are earnestly growing. Scenes are composed of many 3D objects and sometimes, in order to make a scene more complex and realistic, variety is required.

If some information about the class of an object is known, a variance of that object can be generated changing the colour, shape or orientation of one or more of its components. Even in the case that some sections of an object change significantly from instance to instance of the same class, i.e. a total different shape or a total different distribution of colour, with a full knowledge of the class characteristics, a coherent variation of the object can be made. Such knowledge can be represented as sets of rules: procedural 3D modeling techniques create 3D models and textures from sets of rules. The sets of rules may either be embedded into the algorithm or configurable by parameters. Procedural models often exhibit database amplification, meaning that large scenes can be generated from a much smaller number of rules (or parameters), which results in a great variety. However, procedural modeling is not always easily accessible. Often, professional workers are needed to interact with their complex interfaces.

When our goal is to generate 3D content from reality, we can address 3D reconstruction. In computer vision and computer graphics, 3D reconstruction is the process of capturing the shape and appearance of real objects. The research of 3D reconstruction has always been a difficult goal. Using 3D reconstruction one can determine any object's 3D profile, as well as knowing the 3D coordinate of any point on the profile. Therefore, 3D Reconstruction aim is to build a static model of a real instance of an object, which only contains the spatial information of the object. Anyway, extracting information about the object characteristics from such a static reconstruction is hard, and variate the model is not straightforward.

In this work we try to match the two above approaches. We tackle the problem of generating a modifiable 3D model from a real instance of an object belonging to a particular wide and complex category: flowers. This category has been picked because of its major usage in the main application domains of 3D modeling, and also because its level of complexity offers a good challenge for our study.

Hence, given a target flower image, our goal is to retrieve a coherent model that represents the flower and that gives the possibility of changing it to obtain new similar instances. We also impose, as a constraint, that all the information needed have to be retrieved from a single image of the instance. The reason for this choice is that we want simplicity and usability to be preserved, since a large portion of the applications we think for this work (chapter 5) relies on a spread usage from not-professional workers.

As a matter of fact, Deep learning has brought about an encouraging performance in a multitude of tasks in computer vision, achieving impressive results over conventional methods that depend on handcrafted features. Tasks like image classification [4] , object detection [16] , and object localization and segmentation [27] have benefited a lot from applying deep learning algorithms. Most of the state-of-the-art performances in computer vision tasks are achieved by deep- learning-based methods. Anyway, reconstructing a 3D model from a single image, without any other knowledge, is an ill-posed problem. Interestingly, humans are good at solving such inverse problems by leveraging prior knowledge. We can infer the approximate size and rough geometry of objects at a glance and even guess what they would look like from another viewpoint. We can do this because all the previously seen objects and scenes have enabled us to build prior knowledge and develop mental models of what objects look like.

In our approach, we exploit this kind of knowledge through a specific procedural 3D modeling tool (PMT), a flower PMT. Therefore, our idea is to combine PMT and recent advances in Deep Learning techniques.

Among the many challenges in machine learning, data collection is often one of the critical bottlenecks. It is known that the majority of the time for running machine learning end-to-end is spent on preparing the data, which includes collecting, cleaning, analyzing, visualizing, and feature engineering [18].

Our strategy is to build a dataset from our flower PMT. Indeed, the parameters we need for the flower representation can only be attached to images produced from the PMT and cannot be retrieved from online datasets or handcrafted by humans. The images are rendered from random perspectives, using High Dynamic Range Imaging (HDRI) and Image Based Lighting (IBL). We also include random out of focus siblings of the flower to add realism to the scene. In this way we realize a sort of static data augmentation that allows us to theoretically generate infinite realistic flower images (chapter 3).

In traditional machine learning methodologies training data and testing data are taken from the same domain, such that the input feature space and data distribution characteristics are the same. However, in our case, we are not able to collect enough data to train our network from scratch. Therefore, our need is to create high-performance learners trained with more easily obtained data from different domains. This kind of methodology is referred to as transfer learning. We can exploit the benefits of transfer learning as so: we can use the knowledge learnt by the pretrained network and predict the parameters we need for our PMT. As a result, we intend to use a solid pre-trained network, i.e. VGG16 network pretrained on ImageNet dataset, and add our output layers on top of it.

Once the whole network is trained, given an image of a flower, we should be able to generate its corresponding 3D model, by passing the parameters predicted from the network to our PMT, and variate it to create any similar instance we desire.

The reminder of this thesis is organized as follows:
Chapter 2 discusses the work background around the main themes of this project and introduces the basic concepts behind deep learning, the characteristics of a Convolutional Neural Network, the VGG16 architecture.

Chapter 3 describes how we generated our dataset, collecting realistic flower images from renderings, the augmentation techniques used and the entity of the output that has to be predicted.

Chapter 4 shows the implementation of the proposed approach: the necessary data pre-processing and real time data augmentation, the network structure and the main choices for training.

Chapter 5 analyses the obtained results over the test set and real images. Eventually possible exploitations of our work are presented.

Chapter 6 represents the conclusive part of the thesis. A summary of the work is done and the goal achievement is briefly discussed. The section finishes with the introduction to possible future works starting from the outcome of this project.

Chapter 2

State of the Art

This section discusses the work background by analyzing the existing works related to this thesis' main themes. For this purpose, the section is divided into two parts to analyze the background of:

- Plants 3D model reconstruction from Single Image
- Deep learning approaches

2.1 Plants 3D model reconstruction from Single Image

Many works over the last years proposed their approaches in plants 3D model reconstruction.

Yan et al.[24] present a semi-automatic method for reconstructing flower models from a single photograph. Though 3D structure of a flower can appear ambiguous in projection, the flower head typically consists of petals embedded in 3D space that share similar shapes and form a certain level of regular structure. Therefore their technique employs these assumptions by first fitting a cone and subsequently a surface of revolution to the flower structure and then computing individual petal shapes from their projection in the photo. Flowers with multiple layers of petals are handled through processing different layers separately. Occlusions are dealt with both within and between petal layers. Yan et al. method allows users to quickly generate a variety of realistic 3D flowers from photographs and to animate an image using the underlying models reconstructed.

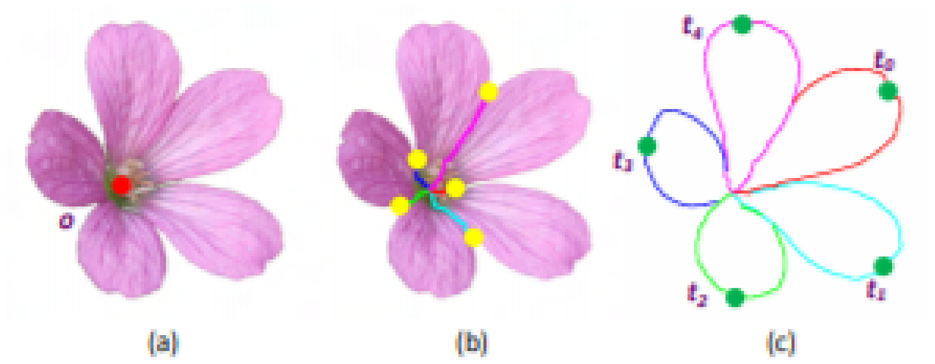


Figure 2.1: Manually location of the flower center o (red in (a)). The intersection points (yellow in (b)) are then automatically detected. Finally the tip of each petal (green in (c)) is automatically located, from [24].

Tan et al.[21] introduce a simple sketching method to generate a realistic 3D tree model from a single image. The user draws at least two strokes in the tree image: the first crown stroke around the tree crown to mark up the leaf region, the second branch stroke from the tree root to mark up the main trunk, and possibly few other branch strokes for refinement. The method automatically generates a 3D tree model including branches and leaves. Branches are synthesized by a growth engine from a small library of elementary subtrees that are pre-defined or built on the fly from the recovered visible branches. The visible branches are automatically traced from the drawn branch strokes according to image statistics on the strokes. Leaves are generated from the region bounded by the first crown stroke to complete the tree.

Guénard et al.[8] propose to generate a 3D model of a plant, using an analysis-by-synthesis method mixing information from a single image and a priori knowledge of the plant species. First, they use a dedicated skeletonisation algorithm to generate a possible branching structure from the foliage segmentation. Then, a 3D generative model, based on a parametric model of branching systems that takes into account botanical knowledge is built. The resulting skeleton follows the hierarchical organisation of natural branching structures. An instance of a 3D model can be generated. Moreover, varying parameter values of the generative model (main branching structure of the plant and foliage), it is possible to produce a series of candidate models. The reconstruction is improved by selecting the model among these proposals based on a matching criterion with the image. Realistic results obtained on different species of plants illustrate the performance of the proposed method.

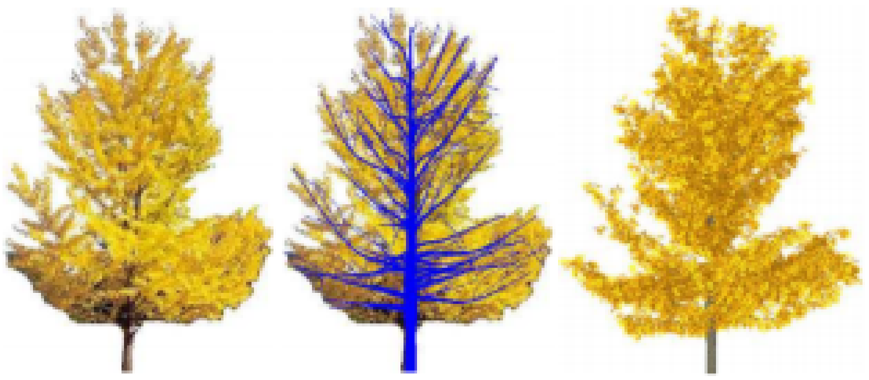


Figure 2.2: On the left, an original image of Ginkgo tree. In the middle, a possible architecture of the branching extracted with the skeletonisation method. On the right, a 3D model of this tree with the same viewpoint, from [8].

Some other works focus on tree structure reconstruction from single image, Zeng et al.[26], Chi et al.[2].

Many other works reconstruct plant 3D models from multiple views, to cite a few: Zeng and Wang et al. in Image-based tree modeling [25], Shlyakhter et al. in Reconstructing 3DTree Models from Instrumented Photographs [19].

2.2 Deep learning approaches

Since 2015, image-based 3D reconstruction using convolutional neural networks (CNN) has attracted increasing interest and demonstrated an impressive performance.

Han et al.[9] provide us a comprehensive survey of the recent developments in this field. They focus on the works which use deep learning techniques to estimate the 3D shape of generic objects either from a single or multiple RGB images. The strategies are various. The main choices regard: representation of output, training dataset, loss function, training procedures and degree of supervision. Representation of output is crucial to the choice of the network architecture. It also impacts the computational efficiency and quality of the reconstruction.

Although, our approach is not designed to point out 3D data directly from the network. Our strategy rely on retrieval of characteristic parameters to be used by our flower PMT. It is hard to find some works aimed in extracting characterizing features of flower images. Instead, there exist a huge literature about flower classification through CNN. We report here some of them, which well represent the main approaches.

Gogul et al.[5], in their research work, use a Deep learning approach using Convolutional Neural Networks (CNN) to recognize flower species with high accuracy. Images of the plant species are acquired using the built-in camera module of a mobile phone. Feature extraction of flower images is performed using a Transfer Learning approach (i.e. extraction of complex features from a pre-trained network). A machine learning classifier such as Logistic Regression or Random Forest is used on top of it to yield a higher accuracy rate. This approach helps in minimizing the hardware requirement needed to perform the computationally intensive task of training a CNN. It is observed that, CNN combined with Transfer Learning approach as feature extractor outperforms all the handcrafted feature extraction methods such as Local Binary Pattern (LBP), Color Channel Statistics, Color Histograms, Haralick Texture, Hu Moments and Zernike Moments. CNN combined with Transfer Learning approach yields impressive Rank-1 accuracies of 73.05%, 93.41% and 90.60% using OverFeat, Inception-v3 and Xception architectures, respectively as Feature Extractors on FLOWERS102 dataset.

Li et al.[14] designs a flower classification model that combines generative adversarial network and ResNet-101 transfer learning algorithm, and uses stochastic gradient descent algorithm to optimize the training process of the model. The experimental results on the international public flower recognition dataset, Oxford flower-102 dataset, show that by enhancing the original data, the accuracy of the network's recognition and classification of flowers is improved. At the same time, the model proposed in their paper is superior to other traditional network models, with higher recognition accuracy and robustness.

Cibuk et al.[3] apply deep convolutional neural networks (DCNN) based hybrid method to flower species classification. The proposed method initially employs a pre-trained DCNN model for feature extraction. To this end, two popular DCNN architectures namely, AlexNet and VGG16 models are adopted. For constructing efficient feature sets, the features from AlexNet and VGG16 models are then concatenated. Finally, a feature selection algorithm, the minimum Redundancy Maximum Relevance (mRMR) method, is used to select the more efficient features. A support vector machine (SVM) classifier with Radial Bases Function (RBF) kernel is employed to classify the flower species using the extracted features. Flower17 and Flower102 datasets which have a huge amount of category are used in the experimental works. Various experiments results show an achieved 96.39% and 95.70% accuracy performance for Flower17 and Flower102, respectively. The obtained results demonstrate the effectiveness of the proposed method, despite the relative simplicity of the approach.

Lou et al.[15] tackle the task of fine-grained flower classification based on the earlier work of Nilsback et al.[17], incorporating both traditional features like HOG, SIFT, HSV with CNN feature to better describe inter-class differences between flowers species.

2.3 Convolutional Neural Networks

2.3.1 Introduction to Deep Learning

In an effort to create systems that learn similar to how humans learn, the underlying architecture for deep learning was inspired by the structure of a human brain. For this reason, quite a few fundamental terminologies within deep learning can be mapped back to neurology. Similar to how neurons form the fundamental building blocks of the brain, deep learning architecture contains a computational unit that allows modeling of nonlinear functions called neuron. Similar to how a neuron in a human brain transmits electrical pulses throughout our nervous system, the neuron receives a list of input signals and transforms them into output signals. The neuron aims to understand data representation by stacking together many layers, where each layer is responsible for understanding some part of the input.

A layer can be thought of as a collection of computational units that learn to detect a repeating occurrence of values. Each layer of neurons is responsible for interpreting a specific pattern within the data. A network of these neurons mimics how neurons in the brain form a network, so the architecture is called neural networks (or artificial neural networks).

2.3.2 Artificial Neural Networks

This section provides an overview of the architecture behind deep learning, artificial neural networks (ANN), and discusses some of the key terminology.

Artificial Neuron An artificial neuron receives a list of inputs x_i and transforms them into output Y, using weight parameters w_i, bias parameter b and activation function f:

$$Y = f(\sum_{i=0}^{m-1} w_i x_i + b),$$

where m is the number of inputs.

The activation function f is a mathematical function that lets you transform the outputs to a desired non-linear format before it is sent to the next layer. It maps the summation result to a desired range. For example, a sigmoid function maps values to the range [0,1], which is useful when the data to be predicted is normalized. Doing so lets you model complex non-linear decision boundaries.

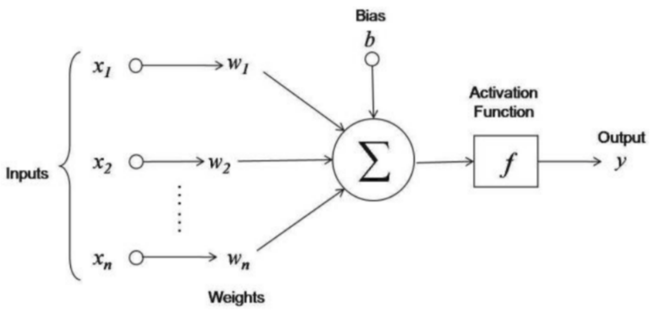


Figure 2.3: Artificial neuron scheme

Shallow Neural Network A set of neurons constitute a layer of an artificial neural network. As shown in figure 2.4, in its most basic form, a neural network contains three layers: input layer, hidden layer, and output layer.

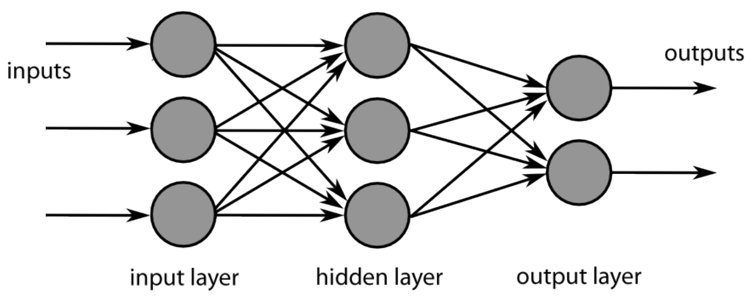


Figure 2.4: Shallow Neural Network Architecture

The computations discussed for the single neuron happen for all neurons in a neural network including the output layer, and one such pass is known as forward propagation. After one forward pass is completed, the output layer must compare its results to the actual ground truth labels and adjust the weights based on the differences between the ground truth and the predicted values. This process is a backward pass through the neural network and is known as back propagation. The basics of the process can be outlined as follows:

- The network works to minimize an objective function, for example, the error incurred across all points in a data sample. At the output layer, the network must calculate the total error (difference between actual and predicted values) for all data points and take its derivative with respect to weights at that layer. The derivative of error function with respect to weights is called the gradient of that layer.

- The weights for that layer are then updated based on the gradient. This update can be the gradient itself or a factor of it. This factor is known as the learning rate, and it controls how large the steps are that you take to change your weights.

- The process is then repeated for one layer before it and continues until the first layer is reached.

- During this process, values of gradients from previous layers can be reused, making the gradient computation efficient.

The result of one pass of forward propagation and back propagation is a change to the network layers, weights and brings the system closer toward modeling the data set provided to it. Because this process uses the gradient to minimize the overall error, the process of converging the neural networks, parameters to the optimum is called gradient descent.

2.3.3 Convolutional Neural Networks

A network with more than one hidden layer is called deep neural network (from here "deep learning"). A layer that perform the operations described in last section and connects all its neurons to its precedent layer's neurons is called fully connected layer, or dense layer. Anyway, network layers may have different functionalities.

This section briefly introduces convolutional and pooling layers, usefull to describe standard convolutional neural network architectures.

Convolutional Layers Convolutional layers are typically used to recognize spatial patterns present in the input. This spatial feature extraction is made possible by the convolution operation. Convolutional neurons convolve small sections of the input image with a kernel filter, as explained by figure 2.5.

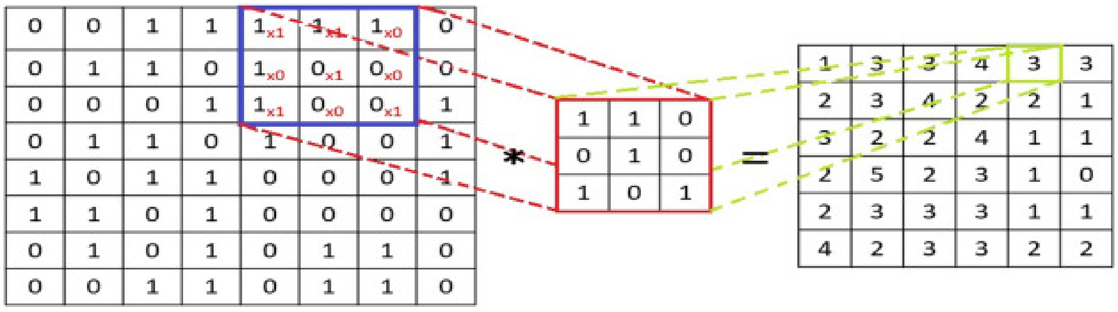


Figure 2.5: 3x3 Kernel Filter Convolution example

Convolution kernel size may vary, usually small odd numbers are used (i.e. 3x3,5x5,7x7). The number of pixel by which the center of the filter is moved from one convolution to the next one is called stride. With stride 1, every possible application of the filter is done. Stride is a parameter that works in conjunction with padding, the feature that adds blank, or empty pixels to the frame of the image for a minimized reduction of size in the output layer. Padding and stride are key parameters in any convolutional layers.

Formally, the convolution operation, for any convolutional neuron, can be expressed as follows:

$$Y_{i,j} = [X \otimes W](i,j) + b = \sum_{\alpha=1}^{f} \sum_{\beta=1}^{f} X(s * i + \alpha, s * j + \beta) W(\alpha, \beta) + b,$$

$$with\ (i,j) \in \{0, 1, ..., L\},\ and\ L = \frac{L_0 + 2p - f}{s} + 1$$

where X is the input matrix , W is the convolutional parameters matrix of the kernel filter, b is the bias parameter, f and s correspond to convolutional kernel size and convolution step (stride), Y is the output of the neuron, L is the dimension of the output, L_0 is the dimension of the input, p is the convolutional padding.

Pooling layers After feature extraction in the convolutional layer, the output feature map is often passed to a pooling layer for feature selection and information filtering. There are several non-linear functions to implement pooling among which max pooling is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum.

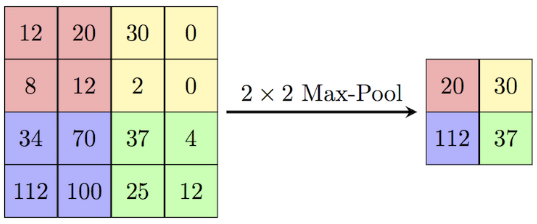


Figure 2.6: 2x2 Max Pooling with stride 2 example

CNNs architecture The standard convolutional neural network architecture consists of multiple blocks constituted by one or more convolutional layers followed by a pooling layer, and one or more fully connected layers at the top. This structure enables the CNN to take advantage of the two-dimensional structure of the input data. By this feature, CNN network gives better results on image and speech recognition than other deep learning structures. Figure 2.7 shows a standard CNN architecture.

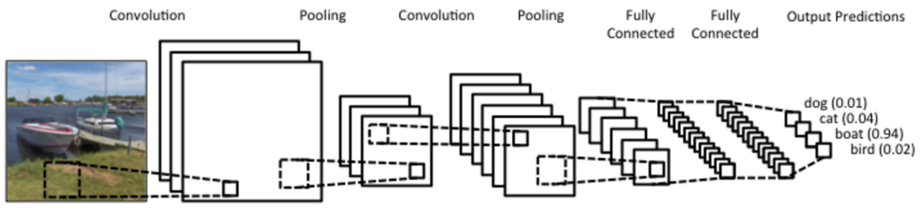


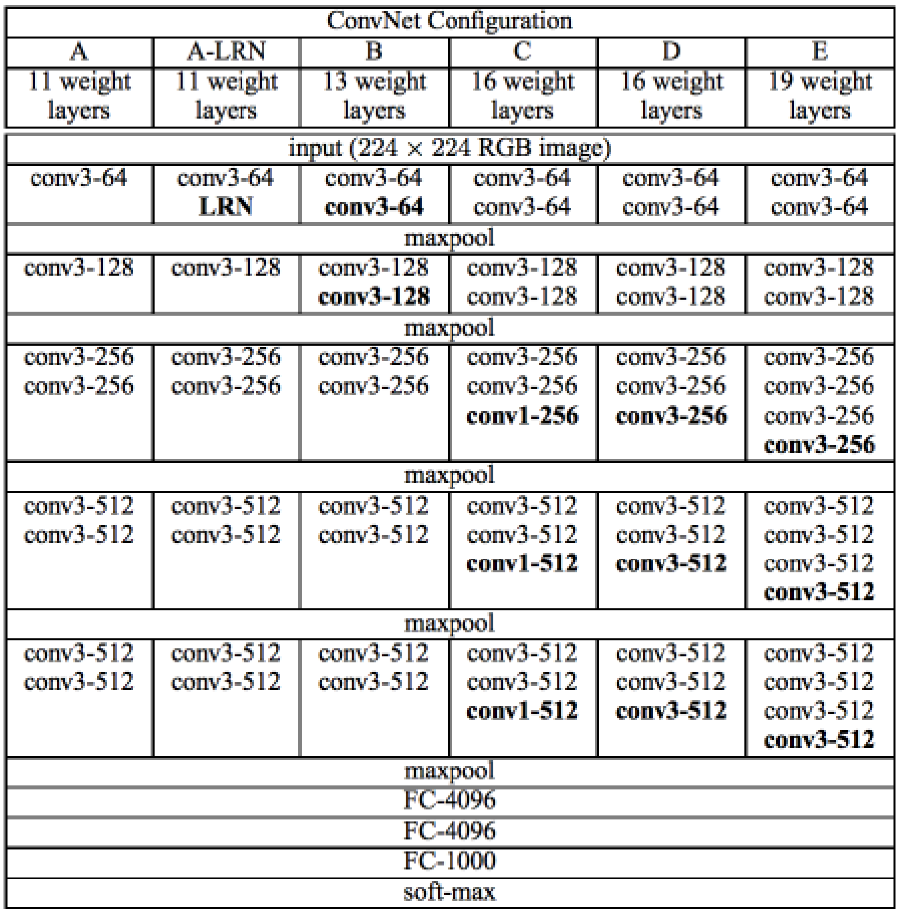
Figure 2.7: CNN architecture example: one convolutional layer and max pooling layer, one convolutional layer and max pooling layer, two fully connected layers and a final fully connected layer (output layer)

Like other deep learning algorithm, convolutional neural network usually uses Rectified Linear Unit (ReLU) as activation function. Some other variants like ReLU includes Leaky ReLU (LReLU), parametric ReLU (PReLU), randomized ReLU (RReLU), and so on Gu et al.[7].

2.4 VGG networks

Transfer learning is the reuse of a pre-trained model on a new problem. It is currently very popular in deep learning because it can train deep neural networks with comparatively little data.

Our project implementation rely on transfer learning. We choose to reuse VGG architectures and weights to exploit their vast success in image understanding. In this section, we shortly describe VGG characteristics and results.



ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.8: Table showing VGGs architectures, from "Very Deep Convolutional Networks for Large Scale Image Recognition", Simonyan and Zisserman (2014) [20]

VGG architecture The VGG network architecture was introduced by Simonyan and Zisserman in their 2014 paper, “Very Deep Convolutional Networks for Large Scale Image Recognition”[20]. This network is characterized by its simplicity, using only 3x3 convolutional layers stacked on top of each other in increasing depth. Reducing volume size is handled by max pooling. Two fully-connected layers, each with 4,096 nodes are then followed by a softmax classifier (above).

The “16” and “19” stand for the number of weight layers in the network (columns D and E in Figure 2.8). Simonyan and Zisserman found training VGG16 and VGG19 challenging (specifically regarding convergence on the deeper networks), so in order to make training easier, they first trained smaller versions of VGG with less weight layers (columns A and C) first.

The smaller networks converged and were then used as initializations for the larger, deeper networks - this process is called pre-training.

VGG, while based off of AlexNet, has several differences that separates it from other competing models: Instead of using large receptive fields like AlexNet (11x11 with a stride of 4), VGG uses very small receptive fields (3x3 with a stride of 1). Because there are now three ReLU units instead of just one, the decision function is more discriminative. There are also fewer parameters (27 times the number of channels instead of AlexNet's 49 times the number of channels). VGG incorporates 1x1 convolutional layers to make the decision function more non-linear without changing the receptive fields. The small-size convolution filters allows VGG to have a large number of weight layers; of course, more layers leads to improved performance. This is not an uncommon feature, though. GoogLeNet, another model that uses deep CNNs and small convolution filters, was also showed up in the 2014 ImageNet competition.

ImageNet ImageNet, is a dataset of over 15 millions labeled high-resolution images with around 22,000 categories. The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) evaluates algorithms for object detection and image classification at large scale. ILSVRC uses a subset of ImageNet of around 1000 images in each of 1000 categories. In all, there are roughly 1.3 million training images, 50,000 validation images and 100,000 testing images.

VGG Evaluations VGG networks are evaluated on ImageNet dataset. To obtain the optimum deep learning layer structure, a refinement study has been done as shown in the figure2.9.

- First of all, VGG-11 already obtains 10.4% error rate, which is similar

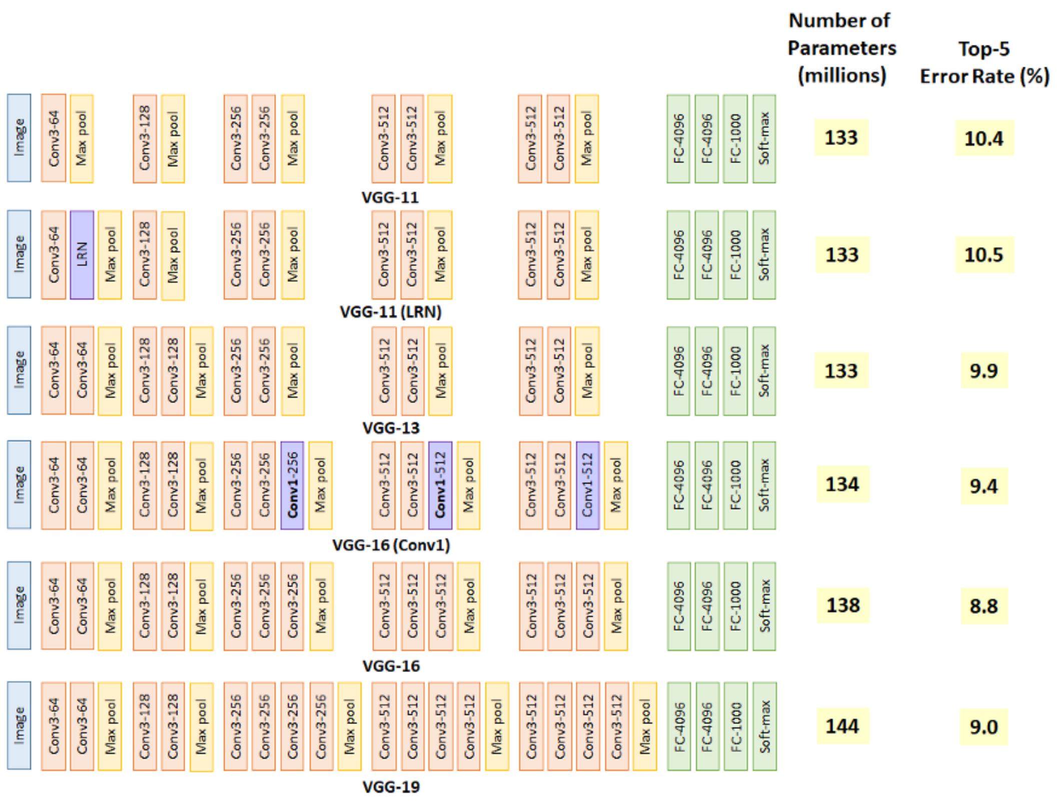


Figure 2.9: Different VGG Layer Structures Using Single Scale (256) Evaluation

to that of ZFNet in ILSVRC 2013. VGG-11 is set as benchmark.

- VGG-11 (LRN) obtains 10.5% error rate, is the one with additional local response normalization (LRN) operation suggested by AlexNet. By comparing VGG-11 and VGG-11 (LRN), the error rate does not improve which means LRN is not useful. In fact, LRN is not used any more in later on deep learning network, instead, batch normalization (BN) is used.

- VGG-13 obtains 9.9% error rate, which means the additional conv helps the classification accuracy.

- VGG-16 (Conv1) obtains 9.4% error rate, which means the additional three 1x1 conv layers help the classification accuracy. 1x1 conv actually helps to increase non-linearlity of the decision function. Without changing the dimensions of input and output, 1x1 conv is doing the projection mapping in the same high dimensionality.

- VGG-16 obtains 8.8% error rate which means the deep learning network is still improving by adding number of layers.
- VGG-19 obtains 9.0% error rate which means the deep learning network is NOT improving by adding number of layers. Thus, authors stop adding layers.

By observing the addition of layers one by one, we can observe that VGG-16 and VGG-19 start converging and the accuracy improvement is slowing down. When people are talking about VGGNet, they usually mention VGG-16 and VGG-19. In our project we are going to adopt VGG16 architecture.

Chapter 3

Dataset generation

The following chapter shows how we have generated our own dataset for the sake of the project. First, we briefly introduce the software used as a support for our scripts. Then we present the parametric representation we chose for the flowers. Eventually, we illustrate the techniques that brought us to have theoretically as much realistic flower images as desired.

3.1 The modeling tool

For the sake of our work, we created an ad hoc flower PMT. Such a tool is useful not only to reconstruct the 3D model from the flower characteristic parameters, indeed it is also necessary to generate the specific data we demand to train the network. The ad hoc flower PMT we created is designed as an add-on of a well known 3D modeling software, Blender. Before continuing, we briefly introduce Blender, and explain the reasons of our choice.

3.1.1 Introduction to Blender

Free license, open source, convenient interface multilateral aspects of design and lightweight platform by hardware requirements makes Blender a modern technological tool preferred by many users.

On October 13, 2002 the first Blender release is delivered under the GNU GPL. The aim of the movement Blender Foundation (a nonprofit public benefit corporation) is: to give designers-users worldwide a full access to 3D technology which stands Blender; to provide service to active users and developers of Blender; to maintain and improve the existing product Blender by commonly available operating system with a source code licensed under the GNU GPL; to organize a system which supports voluntarily the running costs fund.

In the last few years in parallel to the implementation of successful films and other projects, the possibilities of the program Blender in the field 3D design significantly increased. This serves as a pretext for many designers to focus on Blender. This serious choice is not only because Blender is with a free license, but also because of the following advantages that are given to users:

- If desired, it is possible to reprogram the software in a positive aspect and subsequently to be spread to a new species;
- Blender is compatible with OS Windows, Linux, Mac, FreeBSD ;
- Users receive responsiveness and maintenance, also can actively participate with comments and suggestions;
- The new versions come out relatively quickly, and added innovations can quickly be integrated into the design practice;
- Programming language Phyton.

Among the main Blender features, we report in table 3.1 the most relevant for our work:

Photorealistic Rendering	GPU & CPU rendering; Realtime viewport preview; HDR lighting support Permissive License for linking with external software.
Fast Modeling	Keyboard shortcuts for a fast workflow; N-Gon support; Edge slide, collapse and dissolve; Grid and Bridge fill; Python scripting for custom tools and add-ons.
Realistic Materials	Key features: Complete Node Support for full customization; Physically accurate shaders like glass, translucency and SSS; Open Shading Language (OSL) support for coding unique shaders.
Full Compositor	Library of nodes for creating camera fx, color grading, vignettes and much more Render-layer support; Full compositing with images and video files; Ability to render to multiLayer OpenEXR files; Multi-threaded.
File Formats	Image: JPEG, JPEG2000, PNG, TARGA, OpenEXR, DPX, Cineon, Radiance HDR, SGI Iris, TIFF; 3D: 3D Studio (3DS), COLLADA (DAE), Filmbox (FBX), Autodesk (DXF), Wavefront (OBJ), DirectX (x), Lightwave (LWO), Motion Capture (BVH), SVG, Stanford PLY, STL, VRML, VRML97, X3D.
Flexible Interface	User customize; Consistency across all platforms, no disruptive pop-up windows; crisp text (support for retina on OSX).

Table 3.1: Blender features

Blender Add-ons are scriptable plugins that serve to expand the functionality of Blender and may have various purposes. Our add-on can take as input the flower characteristics parameters, use them to build a flower model, and can export it as a 3D model. Figure 3.1 shows our add-on interface.

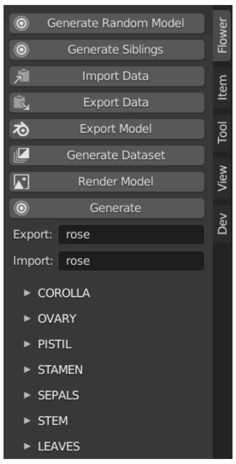


Figure 3.1: Flower Generator Add-on interface

3.1.2 Flower Characteristic Parameters

Flowers pose an interesting and important challenge for three-dimensional computer graphics modeling. They have a great number of components, such as petals, stems, and pistils, which take on highly varied 3D shapes and which are connected with intricate structures. The geometric and structural complexity makes building a flower 3D model a difficult and time-consuming task even for experienced users; for novice users, creation of beautiful and biologically plausible flowers using traditional tools is almost impossible.

Inspired by Ijiri et al.[12], we selected about 150 parameters to define a flower. Table 3.2 shows a list of them.

The parameters can be partitioned into seven classes, depending on which part of the flower they are referring to: petal parameters (46), ovary parameters (6), stamen parameters (19), pistil parameters (14), sepal parameters (22), stem parameters (10), leaf parameters (36). We can also divide the parameters into: categorical parameters (e.g. petalTopForm, petalBaseForm, petalColourType), boolean parameters (e.g. petalUnify, stamenOnPistill, includeSepals), integer parameters (e.g. stamensNumber, sepalsNumber), floating point parameters (e.g. petalLengthMin, petalLengthMax, petalTopSharpness) and vector parameters (e.g. petalColourMin, petalColourMax, stamenTopColour).

Parameters whose name finishes in "Min" (or "Max") define the minimum or initial (maximum or final) quantity for that feature, i.e. given a flower with more than one ring of petals, petalLengthMin is the length of the petals belonging to the first ring, petalLengthMax is the length of the

PETAL PARAMETERS	flowerTopInclination, petalRingsNumber, petalHeightInitial, petalHeightFinal, petalOffset, petalOffsetAlternate, petalNumberMin, petalNumberMax, petalLengthMin, petalLengthMax, petalWidthMin, petalWidthMax, petalWidthLimit, petalWidthLimitRotate, petalUnify, petalUnifyOnlyFirst, petalUnifyHeight, petalRotationMin, petalRotationMax, petalCenterHeightMin, petalCenterHeightMax, petalCenterScaleMin, petalCenterScaleMax, petalBaseForm, petalBaseSharpMin, petalBaseSharpMax, petalTopForm, petalTopSharpMin, petalTopSharpMax, petalTopFuzziness, petalBendHorizontal, petalBendHorizontal2, petalBendHorizontalStep, petalBendHorizontalStep2, petalBendHorizontalHeight, petalBendHorizontalHeight2, petalBendLateral, petalBendLateralHeight, petalBendLateralStep, petalColourType, petalColourMin, petalColourMax, petalColour2Min, petalColour2Max, petalColourHeight, petalColourHeight2
OVARY PARAMETERS	ovaryHeight, ovaryRadius, ovaryBaseHeight, ovaryColourType, ovaryColour, ovaryColour2
STAMEN PARAMETERS	stamenHeightInitial, stamenHeightFinal, stamenNumber, stamenOnPistil, stamenBaseHeight, stamenBaseThickness, stamenBaseBendHeight, stamenBaseBendLateral, stamenBaseBendForward, stamenBaseFuzziness, stamenBaseColourType, stamenBaseColour, stamenBaseColour2, stamenType, stamenTopHeight, stamenTopLength, stamenTopWidth, stamenTopColour, includeStamens
PISTIL PARAMETERS	pistilBaseHeight, pistilBaseThicknessTop, pistilBaseThicknessBottom, pistilBaseBendHeight, pistilBaseBendLateral, pistilBaseBendForward, pistilBaseFuzziness, pistilBaseColour, pistilTopHeight, pistilTopLength, pistilTopWidth, pistilTopNumber, pistilTopColour, includePistil
SEPAL PARAMETERS	sepalsOffset, sepalsNumber, sepalsLength, sepalsWidth, sepalsWidthLimit, sepalsUnify, sepalsUnifyHeight, sepalsCenterMulti, sepalsCenterHeight, sepalsCenterScale, sepalsBaseForm, sepalsBaseSharp, sepalsTopForm, sepalsTopSharp, sepalsTopFuzziness, sepalsBendHorizontal, sepalsBendHorizontal2, sepalsBendHorizontalHeight, sepalsBendHorizontalHeight2, sepalsBendLateral, sepalsColour, includeSepals
STEM PARAMETERS	stemHeight, stemThickness, stemInclinationFactor, stemBendHeight, stemBendLateral, stemBendLateral2, stemBendForward, stemBendForward2, stemBendHeight2, stemColour
LEAF PARAMETERS	leavesRingsNumber, leavesHeightInitial, leavesHeightFinal, leavesFuzziness, leavesLengthMin, leavesLengthMax, leavesWidthMin, leavesWidthMax, leavesCenterMulti, leavesCenterHeightMin, leavesCenterHeightMax, leavesCenterScaleMin, leavesCenterScaleMax, leavesBaseForm, leavesBaseSharpMin, leavesBaseSharpMax, leavesTopForm, leavesTopSharpMin, leavesTopSharpMax, leavesTopDoubleMin, leavesTopDoubleMax, leavesTopDoubleHeightMin, leavesTopDoubleHeightMax, leavesTopFuzziness, leavesBendHorizontal, leavesBendHorizontal2, leavesBendHorizontalStep, leavesBendHorizontalStep2, leavesBendHorizontalHeight, leavesBendHorizontalHeight2, leavesBendLateral, leavesBendLateralStep, leavesColourMin, leavesColourMax, includeLeaves

Table 3.2: Flower characteristic parameters

petals belonging to the last ring and the length of the intermediates is given by linearly interpolating the two measures.

Multiple instances of the same parameter, for example stemBendHeight and stemBendHeight2, are included to give more freedom in feature representation.

Some parameters' value is enabled by other boolean parameters, for instance petalTopSharpness is meaningless if petalTopForm is not "SHARP", and the whole pistil parameters can be skipped if includePistill is set to "False".

Over all the parameters chosen for the representation, there are some key parameters that we take particular care of: petalUnify, petalUnifyOnlyFirst, includeSepals, includePistil, includeStamens, includeLeaves, petalColourMin, petalColourMax, petalColour2, petalColour2Max, ovaryColour, petalTopForm, petalColourType, petalRingsNumber, petalNumberMin, petalNumberMax, petalLengthMin, petalLengthMax, petalWidthMin, petalWidthMax, petalBendHorizontal, petalBendHorizontal2, petalBendHorizontalStep, petalBendHorizontalStep2, petalBendLateral, petalBendLateralStep, ovaryHeight, ovaryRadius. These parameters describe the main shape and colour of the corolla, therefore they are fundamental for our purpose and should be treated specially.

Further details on the feature parameters chosen can be found in appendix A.

3.2 Data Collection

3.2.1 Initiatory Models

With the purpose of generate data for our dataset we started selecting 16 initiatory models significantly different from each other, figure 3.2.

In this first selection we tried to pick at least one example for each categorical and boolean parameter value. Though the majority of such models are taken as an instance of a particular real world flower species, some of the models chosen are not specifically referred to any existing flower (i.e. figure 3.2m, 3.2p). We selected them anyway, because their shape and colour are plausible and, since the goal of our dataset is to furnish our network an alphabet of flower characteristic parameters through flower images, they were perfectly suited.

From these initiatory models, we produced secondary models, by introducing some feature variations. Figure 3.3 shows some example of variation in shape and colour.

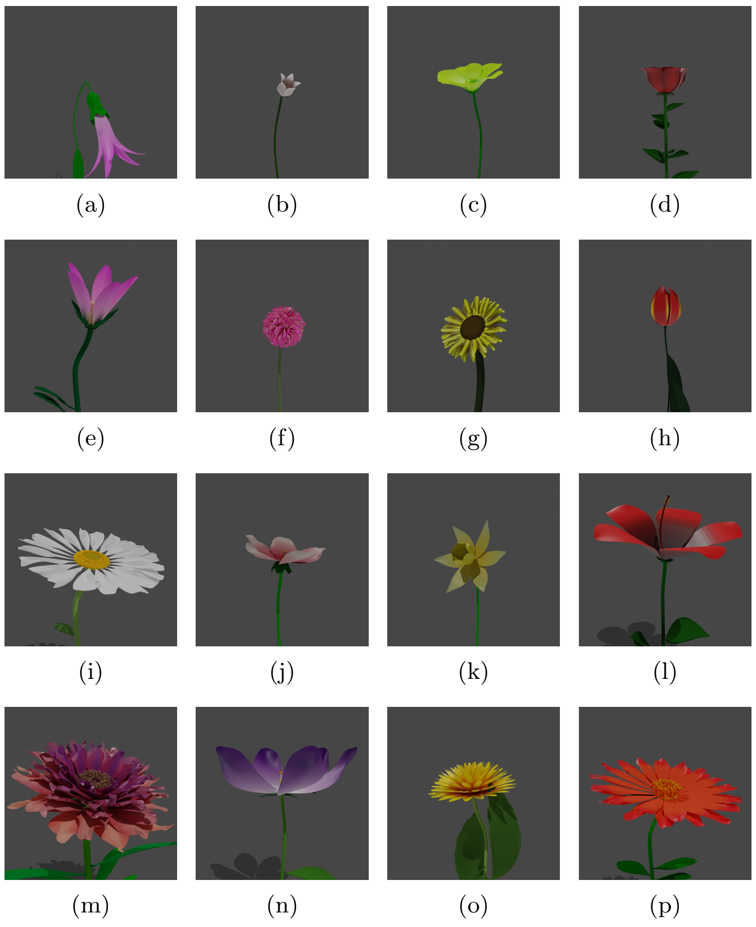


Figure 3.2: Initiatory Models

With such transformations, we added 11 secondary models, for a total number of 27 models. Since our idea is to train the network through our rendered models we need to make some our renderings realistic. In the next section we explain how we tackle the problem.

3.2.2 High Dynamic Range Imaging and Image Based Lighting

Images with standard dynamic range (SDR) store 8 bits of data for each of the red, green, and blue channels for each pixel. An example of SDR image is a JPG file. SDR image is limited to a relatively small range of luminance values from 0 to 255, which is insufficient for the illumination in 3D.

HDR (high dynamic range) is an established family of standards that enables monitors to show brighter, more vivid images with increased color impact. This results in more realistic-looking content. Details are more

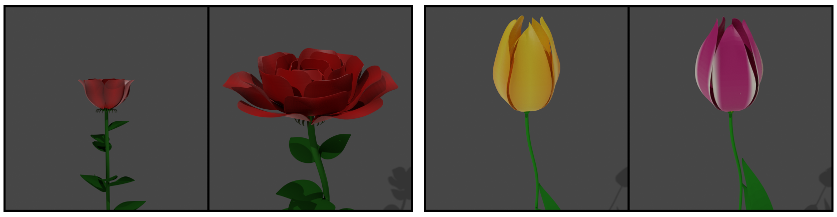


Figure 3.3: Opening of petals on the left, different pigmentation on the right

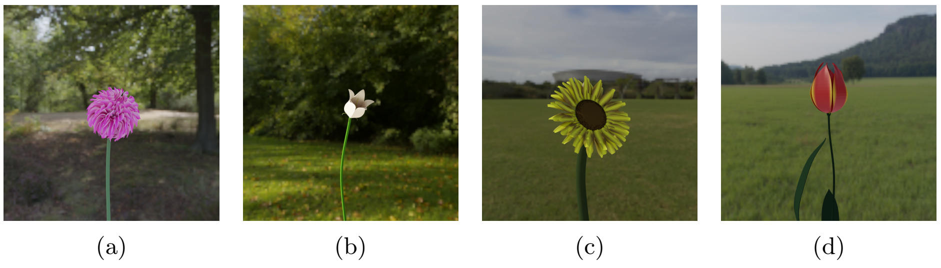


Figure 3.4: HDRI and IBL renderings

refined and images appear more accurate than with standard dynamic range. HDR images usage in computer graphics is referred as HDR imaging (HDRI). HDRI is useful for recording many real-world scenes containing very bright, direct sunlight to extreme shade, or very faint nebulae. HDR images are often created by capturing and then combining several different, narrower range, exposures of the same subject matter.

This is the reason why we used HDRI in our work. We utilize 15 natural HDR images as skyboxs, i.e. a visual background for our environment. Every render, one of the HDR images is chosen randomly, and it constitutes our scene setting.

To better exploit this technique we combine it with image-based lighting. Image-based lighting (IBL) is a 3D rendering technique which involves capturing an omnidirectional representation of real-world light information as an image. This image is then projected onto a dome or sphere analogously to environment mapping, and this is used to simulate the lighting for the objects in the scene. This allows highly detailed real-world lighting to be used to light a scene, instead of trying to accurately model illumination using an existing rendering technique. Also, IBL allows us to add a shadow to our model, to be projected to the base plan.

As can be seen from figure 3.4, coupling HDRI and IBL, we can obtain more realistic images that vary in background and lighting.

3.2.3 Sibling Addition and Camera Randomization

Flowers in our world are often found in groups. Giving just images of single flowers to our network would not allow us to teach it how to recognize flower when there are more then one in the image. For this reason, we rendered our models adding randomly generated siblings of the original model. We name sibling an instance that is almost the same of the original flower, though every numerical parameter is slightly different, as its value is obtained randomly obtained from a gaussian distribution, whose mean is the value of the original flower parameter. Moreover, we maintain only our original in focus and put all the sibling out of focus. The more distance from the original they have the more out of focus they result.

Also, at each rendering, we randomize our camera coordinates. Images can have close or far visual and top, lateral or bottom view. We rendered 50 images for each of our initiatory and secondary models. Every 5 images we randomly added a pair of siblings to the scene. Table 3.5 present some examples. Removing some unlucky rendered images (where some siblings were too close to the camera), we were able to collect about 1300 images. We are aware that this quantity may not be enough to train a network from scratch for such a complex task. We tackle this problem in the next section, alongside the actual implementation of our network.

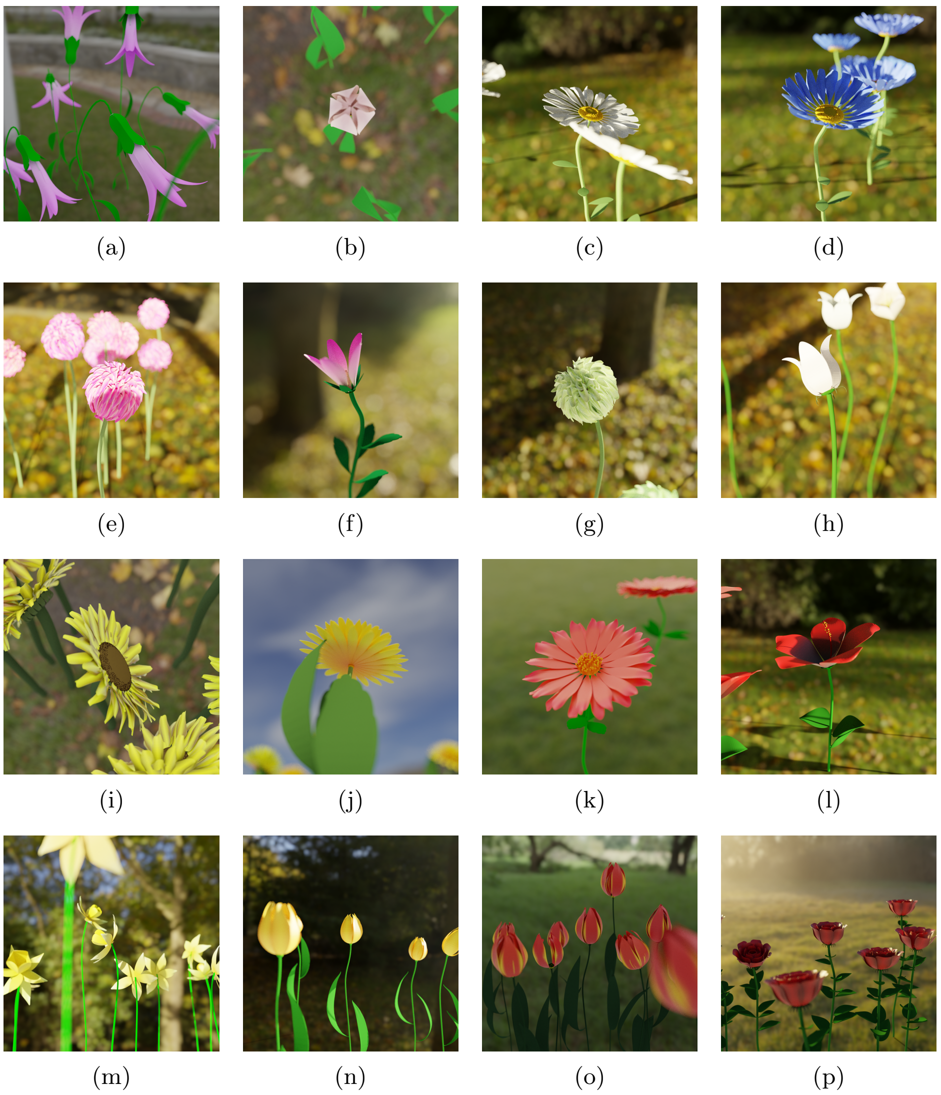


Figure 3.5: Final flower renderings

Chapter 4

Network Implementation

In this chapter, we are going to describe the data pre-processing procedure we adopted for our input data, comprehensive of data augmentation. Consequently, our network architecture is presented: the global average pooling of the VGG16 output (only convolutional modules) is taken as input by our two prediction branches, the first designed for the classification parameters and the second for the regression parameters. Eventually, we reason about the optimizer and the metrics chosen for training.

4.1 Data pre-processing

4.1.1 Image pre-processing

Before applying convolutional neural networks, our images should be prepared for training. Pre-processing is a common name for operations with images at the lowest level of abstraction. Data pre-processing is useful to improve model performance. Since our input images are obtained as software renderings, most of the pre-processing techniques which are usually performed over the dataset are not necessary. Anyway, before using our images, we apply a couple of operations to them:

- Resize image from high resolution (920x920) to low resolution (250x250). The most common networks take inputs of this size, and anyway performances are not usually significantly compromised. The main reason of resizing images is resource saving.

- Pixel value normalization between 0 and 1. The reason we normalize input data is to grant a uniform scale among all the quantities present in our network. If some operations result in a different scale there is a risk for vanishing gradients and slow learning, as the model essentially has to learn to normalize the values to some degree itself, before it actually starts to learn. Features can obviously operate on very disparate scales, but their scale does not necessarily have anything to do with how important they will be to the model.

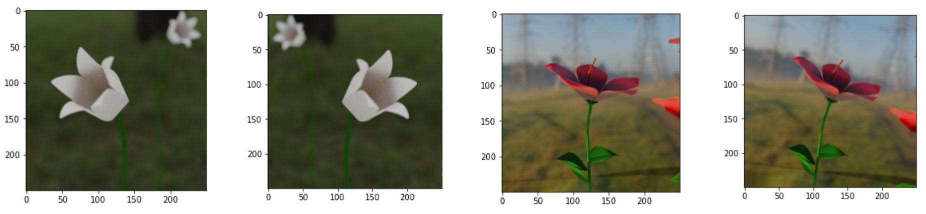
Besides data pre-processing, image data augmentation can be used to improve model performance and reduce generalization error. We already did some kind of static data augmentation technique during our data collection process. Real-time data augmentation instead is a technique used to generalize the inputs seen by the network. Without real-time data augmentation, at each epoch the network analyses always the same samples. Instead, applying real-time data augmentation, every new epoch new images are used to train the network. These new images are obtained randomly as variations of the original samples, i.e. with random rotation, flipping, zooming, shearing, etc. Not all the operations are meaningful for every CNN input. Here we describe the two operations we chose to randomly variate the input image seen by the network at each epoch:

- Random Horizontal Flipping. Since our flowers preserve some sort of symmetry over the vertical axis, flipping the images horizontally generates another possible input for our network, figure 4.1 (a), (b).
- Random Rotation, from -20 to 20 degrees. Flower stems can be bent from the wind or by the weight of the flower head. A rotation of few degrees, therefore, serves another realistic input for our network, figure 4.1 (c), (d). Only 90 degrees multiple rotations preserve image data perfectly. Other types of rotations generate a loss in the images, that has somehow to be filled. The information on the angles is not essential for our network, so it's a trade-off we can afford. We decided to fill the space with "nearest" fashion, i.e. the empty pixels are replaced by their neighbour pixels.

4.1.2 Output parameters pre-processing

Also our output values should be prepared before we use them for training.

First of all we need to translate every not-numerical value to a numerical feature. For this reason we performed One Hot Encoding (OHE) on every categorical parameter. This process takes every categorical parameter field, counts all its possible values and add a binary field for each of them. In this way all the categorical fields are translated into binary fields. This could



(a) Original Image (b) Flipped Image (c) Original Image (d) Rotated Image

Figure 4.1: Application of augmentation techniques

sometimes result in a label explosion, but in our case we have less then ten categorical features, and each of them have four or less possible values, so this technique is practicable in our case. Also, we have to divide every vector parameter into singular element parameters.

Once our parameters are represented as numerical features we need to normalize them to [0,1] range, as we did with our input. We apply this normalization to the output data for the same reason we applied it to the input: to grant a uniform scale among all the quantities present in our network.

Eventually, as we explained in chapter 3, some of our parameters values are meaningless if their corresponding enabler is set to "False". We want to be sure that our network will not try to minimize some meaningless values, therefore, we are going to iterate over the whole samples and set to 0 every parameter if its correspondent enabler is set to "False".

4.1.3 Train, Validation and Test Split

In order to evaluate the performance of our network, we divided our dataset into two subsets. The first subset is used to fit the model and is referred to as the training set. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second subset is referred to as the test set. The objective is to estimate the performance of the machine learning model on new data: data not used to train the model.

We divided the samples as follows: for each flower species, we took 90% of its renderings for the training set and left the other 10% to the test set.

As our network will also use validation to avoid over-fitting, we further divided our training set into two subsets. 15% of the training set constitute the validation set, the rest is the actual training set. Also this subdivision was made granting equal subdivision of flower species among the parties.

4.2 Network architecture

Building a custom output branch on top of a pre-trained network designed for similar tasks is a common pattern in deep learning. As we anticipated in chapter 2, we rely on some known architecture for the first image understanding phase. For our work, we take into account the VGG-16 model and use the weights pre-trained on the well known ImageNet dataset. Our task anyway is different from the original task of VGG network, therefore we remove the last layers used for classification and substitute them with an average pooling layer in order to collect all the spatial information in this last layer.

Our goal is to predict all the flower parameters described in chapter 3. Although, some of our output parameters are preferable to be predicted as classes, i.e. our boolean and categorical parameters; instead, the rest of the parameters have to be predicted with regression, i.e. integer, floating point and vector floating point parameters. Consequently, we design two branches: a classification branch and a regression branch.

Chen et al.[1] and Torgo et al.[22] present approaches where regression network are supported by a preparatory classification phase. We take inspiration from their works and exploit the variety of our output parameters for our purpose. In fact, we decided that our approach will be to first predict the classification parameters and then, combine them with the image data to predict the regression parameters.

We designed the two branches as so:

- **Classification branch**. A fully connected layer of 1024 neurons with RELU activation function takes the Global Average Pooling (GAP) output from the VGG network. A dropout of 0.2 is applied and a fully connected layer for the output follows. We used sigmoid function for this last layer, since our outputs have to be in range [0,1].

- **Regression branch**: A fully connected layer of 1024 neurons with RELU activation function takes the Global Average Pooling (GAP) output from the VGG network along with the Classification branch output. Another fully connected layer of 512 neurons with RELU activation is applied. Since the our regression parameters are more complex then our classification parameters, no dropout is applied. At last, a fully connected layer for the output is added. Also here we use a sigmoid function as activation function for the last layer, since even these outputs have to be in range [0,1].

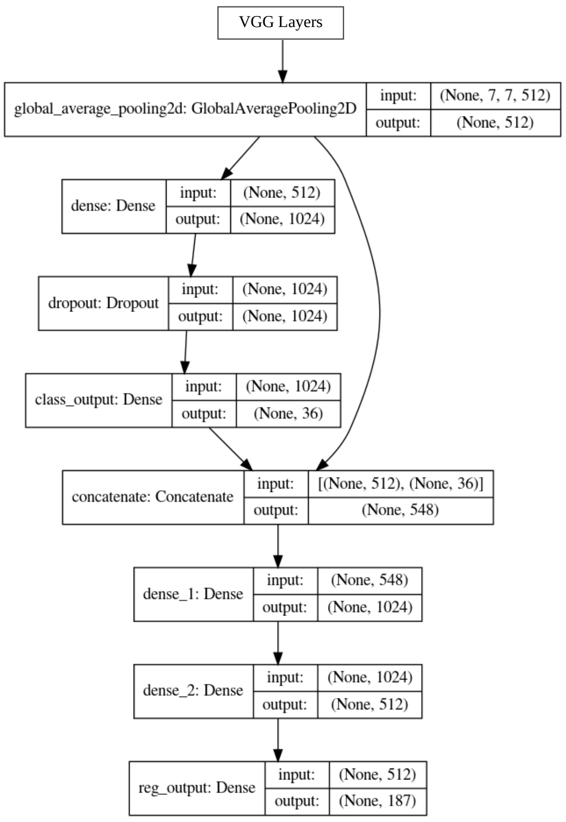


Figure 4.2: Scheme of our Multi-classification and Multi-regression Network

Figure 4.2 shows a scheme of the complete network.

Taking into account the importance of some particular parameter, i.e. the ones describing main corolla colour and shape, our actual implementation of the network will further divide the two main branches output layers into two, one for the key parameters and one for the other parameters. The loss over the key parameters will have more weight than the loss over the other parameters.

4.2.1 Optimizer

For this network we decided to adopt the well known Adam optimizer algorithm. Here we explain the main reasons of our choice.

Adam is an adaptive learning rate optimization algorithm that has been designed specifically for training deep neural networks. In the original paper [13], Adam was demonstrated empirically to show that convergence meets the expectations of the theoretical analysis. Adam was applied to the logistic regression algorithm on the MNIST digit recognition and IMDB sentiment analysis datasets, a Multilayer Perceptron algorithm on the MNIST dataset and Convolutional Neural Networks on the CIFAR-10 image recognition dataset.

In the Stanford course on deep learning for computer vision developed by Karpathy et al., the Adam algorithm is suggested as the default optimization method for deep learning applications.

Adam is being adapted for benchmarks in deep learning papers. For example, it was used by Xu et al.[23] on attention in image captioning and by Gregor et al.[6] on image generation.

4.2.2 Losses and Measures

As our branches predict categorical and regression values we need two different loss functions to train our model:

- **Binary cross entropy (BCE)**, also called Sigmoid Cross-Entropy loss. Unlike the well known Softmax loss, it is independent for each vector component (class), meaning that the loss computed for every CNN output vector component is not affected by other component values. That is why it is used for multi-label classification, where the insight of an element belonging to a certain class should not influence the decision for another class. It is called Binary Cross-Entropy Loss because it sets up a binary classification problem between two values for every class.

$$BCE = -\frac{1}{M}\sum_{i=1}^{M}(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$$

- **Mean Squared Error (MSE)**, which is the average of the squared difference between each predicted point and the actual point. MSE is a risk function, corresponding to the expected value of the squared error loss.

$$MSE = \frac{1}{M}\sum_{i=1}^{M}(y_i - \hat{y}_i)^2$$

We adopted BCE to train our classification branch whilst we used MSE to minimize the regression branch error.

We also involve another couple of measures, to better understand our results:

- **Classification Error Rate (CER)**, proportion of samples misclassified over the whole set of samples.

$$CER = \frac{\#\, misclassified}{\#\, total\ samples}$$

- **Mean Absolute Error (MAE)**, average of the absolute difference between each predicted value and the actual value. MAE corresponds to the expected value of the absolute error loss.

$$MSE = \frac{1}{M}\sum_{i=1}^{M} |y_i - \hat{y}_i|$$

We are going to adopt BCE and CER to measure classification error over the test set, whilst MSE and MAE will be useful for the regression evaluations.

Chapter 5

Experimental Results

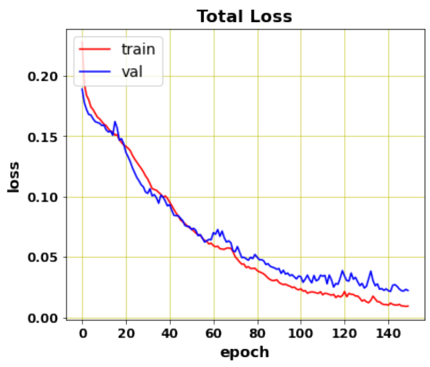
In this chapter we will report the main experimental results of our work. We start analysing the results obtained on the test set, the portion of original renderings designed to test the performance of our network. After that, we select some real flower images. First, we take some pictures similar to the images the network took as input for training, then, we pick other flower pictures and compare the obtained results. Eventually, we propose some possible exploitation for this work.

5.1 Test Set Results

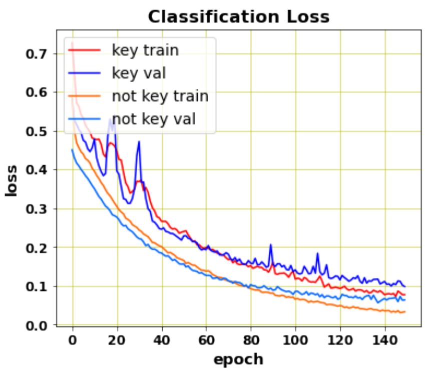
Before testing our network with real flowers images, we first take a look at the training and testing scores. We train our network for 150 epochs, with a learning rate of 0.01, and batches of 20 images. We use respectively BCE and MSE as loss functions for our classification and regression branches. We define a total loss function as the some of the former loss function with coefficient 0.1 and the latter with full weight:

$$Total\ Loss = 0.1BCE + MSE$$

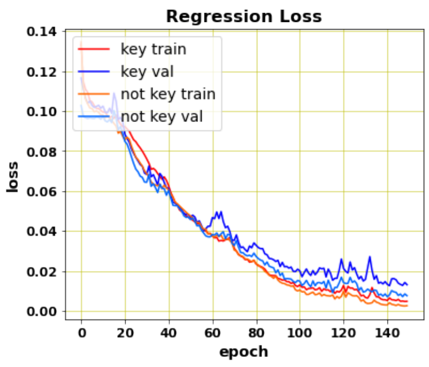
The reason for this choice derives from our experimental trainings: we have seen that BCE is always about ten times MSE. Therefore, since we want to give same importance to classification and regression losses, we decided to apply this reduction over BCE loss. As we anticipated in chapter 3, we are going to treat some parameters with a special regard. Therefore, BCE and MSE losses will be constituted of two parts each: key parameters loss and not-key parameters loss. We decided to give not-key parameters a tenth of the key parameters importance.



(a) Total Loss



(b) Classification Loss (BCE)



(c) Regression Loss (MSE)

Figure 5.1: Plots of Loss functions over the epochs of training

Consequently, our definitive total loss will look as follows:

$$Total\ Loss = 0.1(BCE_K + 0.1BCE_{NK}) + MSE_K + 0.1MSE_{NK}$$

The results of the training are visible in figure 5.1a. In this plot, the loss function sharply decreases over the first 60 epochs; then, the error still diminishes, but the curve is smoother. We can also notice that in the first part of the plot the validation loss is sometimes equal or lower than the training loss. This is a possible phenomenon when real-time data augmentation is applied. However, in the last parts of the plot, validation loss overcome training loss, which is an expected behaviour.

In figure 5.1b and 5.1c we report the plots of the independent losses. These plots present a similar trend w.r.t. the Total Loss function plot.

As for the test set evaluation, here we report the values of the selected measures evaluated for every parameter. Table 5.1 shows the values of the BCE and CER for the classification parameters. The biggest error rate belongs to the sepalsTopForm_SHARP parameter: on the test set, our

	BCE	CER		BCE	CER
petalBaseForm_CURVE	0.7043	2.2900 %	petalBaseForm_SHARP	0.7050	3.0534 %
petalTopForm_CURVE	0.3518	0.7633 %	petalTopForm_DOUBLE	0.2354	1.5267 %
petalTopForm_SHARP	0	0. %	petalColour Type_INNER_OUTER	0	0. %
petalColourType_RADIAL	0	0. %	petalColourType_SIMPLE	0.1177	0.7633 %
sepalsBaseForm_CURVE	0.7043	2.2900 %	sepalsBaseForm_SHARP	1.0555	2.2900 %
sepalsTopForm_CURVE	1.2910	3.8167 %	sepalsTopForm_DOUBLE	0.2340	0. %
sepalsTopForm_SHARP	1.9989	9.9236 %	ovaryColourType_SIMPLE	0	0. %
stamenBaseColour Type_BOTTOM_TOP	1.1746	4.5801 %	stamenBaseColour Type_SIMPLE	1.0576	4.5801 %
stamenType_DOUBLE	1.2931	6.1068 %	stamenType_MONO	1.8776	5.3435 %
leavesBaseForm_CURVE	0.3532	2.2900 %	leavesBaseForm_SHARP	0.1177	0.7633 %
leavesTopForm_CURVE	0.5880	3.0534 %	leavesTopForm_CUT	0.2354	1.5267 %
leavesTopForm_DOUBLE	0	0. %	leavesTopForm_SHARP	0.8235	4.5801 %
petalOffsetAlternate	1.2938	6.8702 %	petalWidthLimit	0.3518	0.7633 %
petalWidthLimitRotate	1.2917	4.5801 %	petalUnify	0.5880	3.0534 %
petalUnifyOnlyFirst	0.2340	0. %	sepalsWidthLimit	0.1170	0. %
sepalsUnify	0.7029	0.7633 %	stamenOnPistil	0.5859	0.7633 %
includeSepals	0.3518	0.7633 %	includePistil	0.8235	4.5801 %
includeStamens	0.2340	0. %	includeLeaves	0.4702	2.2900 %

Table 5.1: Classification Parameters Measures Evaluation

network mispredicts whether sepals have a sharp top about 10% of the times. Other parameters such as petalTopForm_SHARP or petalColour-Type_INNER_OUTER are predicted, over the test set, always correctly.

Parameters like includeSepals, includePistil, includeStamens, include-Leaves are fundamental for an accurate representation of the flower. As we can appreciate by looking at values in the table, their CER is not grater than 5%. Also petalUnify and petalUnifyOnlyFirst parameters have to be taken particularly care of, their CER is not greater than 4%.

Tables 5.2 and 5.3 present the values of the MSE and MAE for the regression parameters.

In order to correctly read these values we are going to specify that, while MSE is evaluated over the normalized predicted values, MAE is instead calculated on the original domain values of the parameters (i.e. MAE is computed after transforming both expected value and predicted value to the original domain of the parameter). Therefore, MAE is specific to every parameter domain.

Let us now have a closer look to the MAE value of some of the key parameters: petalRingsNumber, petalNumberMin, petalNumberMax, petal-ColourMin, petalColourMax (from the first table), petalLengthMin, petal-LengthMax, petalWidthMin, petalWidthMax (from the second table). Petal-RingsNumber, which indicates the number of rings constituting the corolla, has a MAE of 0.22, which means that about one flower out of five has its petalRingsNumber value mispredicted by one ring (or that one flower out of ten has its value mispredicted by two rings, etc). PetalNumberMin and

	MSE	MAE		MSE	MAE
petalRingsNumber	0.0049	0.2290	petalNumberMin	0.0029	0.3435
petalNumberMax	0.0030	0.5114	sepalsNumber	0.0025	0.4580
pistilTopNumber	0.0378	0.1145	stamenNumber	0.0031	75.2671
leavesRingsNumber	0.0029	0.1755	petalColourMin_0	0.0154	0.0715
petalColourMin_1	0.0317	0.1122	petalColourMin_2	0.0388	0.1200
petalColourMin_3	5.6515e-06.	0	petalColourMax_0	0.0133	0.0584
petalColourMax_1	0.0364	0.1210	petalColourMax_2	0.0303	0.0923
petalColourMax_3	4.9809e-06.	0	petalColour2_0	0.0104	0.0351
petalColour2_1	0.0155	0.0699	petalColour2_2	0.0215	0.0788
petalColour2_3	5.7813e-06.	0	petalColour2_0	0.0091	0.0307
petalColour2_1	0.0177	0.0860	petalColour2_2	0.0259	0.0827
petalColour2_3	3.9284e-06.	0	sepalsColour_0	0.0001	0.0063
sepalsColour_1	0.0006	0.0108	sepalsColour_2	0.0001	0.0065
sepalsColour_3	0.0094	0	ovaryColour_0	0.0099	0.0666
ovaryColour_1	0.0049	0.0451	ovaryColour_2	0.0039	0.0324
ovaryColour_3	4.1132e-06.	0	ovaryColour2_0	0.0275	0.0436
ovaryColour2_1	0.0086	0.0388	ovaryColour2_2	0.0138	0.0473
ovaryColour2_3	4.4857e-06.	0	stemColour_0	0.0095	0.0141
stemColour_1	0.0144	0.0276	stemColour_2	0.0062	0.0036
stemColour_3	5.4630e-06.	0	pistilBaseColour_0	0.0093	0.0125
pistilBaseColour_1	0.0075	0.0264	pistilBaseColour_2	0.0063	0.0193
pistilBaseColour_3	0.0295	0	pistilTopColour_0	0.0100	0.0277
pistilTopColour_1	0.0029	0.0208	pistilTopColour_2	0.0006	0.0076
pistilTopColour_3	0.0297	0	stamenBaseColour_0	0.0087	0.0493
stamenBaseColour_1	0.0089	0.0452	stamenBaseColour_2	0.0077	0.0200
stamenBaseColour_3	0.0032	0	stamenBaseColour2_0	0.0221	0.0321
stamenBaseColour2_1	0.0223	0.0216	stamenBaseColour2_2	0.0039	0.0136
stamenBaseColour2_3	0.0031	0	stamenTopColour_0	0.0100	0.0485
stamenTopColour_1	0.0059	0.0432	stamenTopColour_2	0.0128	0.0139
stamenTopColour_3	0.0031	0	leavesColourMin_0	0.0051	0.0045
leavesColourMin_1	0.0168	0.0140	leavesColourMin_2	0.0031	0.0022
leavesColourMin_3	0.0153	0	leavesColourMax_0	0.0049	0.0043
leavesColourMax_1	0.0147	0.0149	leavesColourMax_2	0.0025	0.0019
leavesColourMax_3	0.0153	0			

Table 5.2: Regression Parameters Measures Evaluation (integer and vector parameters)

petalNumberMax have their MAE values near 0.3 and 0.5, it means that about one flower out of two predicted has one more or one less petal per ring w.r.t. the original flower. PetalColourMin and petalColourMax are fundamental for the representation of the flower. The MAE of each component of these vectors (RGBA, which varies in range [0,1]) is around 0.1. Finally, petalLengthMin and petalLengthMax, which describe the initial and the final petal length, and petalWidthMin and petalWidthMax, which describe the initial and final petal width, have MAE values: 0.28 and 0.30, 0.21 and 0.22. These correspond to the mean errors in centimeters.

The overall evaluations over the test set offer us a good base to further test the network on real images.

	MSE	MAE		MSE	MAE
flowerTopInclination	0.0141	14.2553	petalHeightInitial	0.0104	0.0428
petalHeightFinal	0.0036	0.0292	petalOffset	0.0098	6.2253
petalLengthMin	0.0027	0.2845	petalLengthMax	0.0024	0.3066
petalWidthMin	0.0059	0.2183	petalWidthMax	0.0060	0.2254
petalUnifyHeight	0.0164	0.0648	petalRotationMin	0.0098	0.6122
petalRotationMax	0.0099	0.6140	petalCenterHeightMin	0.0175	0.0647
petalCenterHeightMax	0.0173	0.0646	petalCenterScaleMin	0.0077	0.1357
petalCenterScaleMax	0.0081	0.1378	petalBaseSharpMin	0.0066	0.0418
petalBaseSharpMax	0.0064	0.0405	petalTopSharpMin	0.0065	0.0448
petalTopSharpMax	0.0066	0.0439	petalTopFuzziness	0.0213	0.0441
petalBendHorizontal	0.0191	6.7550	petalBendHorizontal2	0.0078	6.7633
petalBendHorizontalStep	0.0082	1.9497	petalBendHorizontalStep2	0.0207	0.7800
petalBendHorizontalHeight	0.0153	0.0176	petalBendHorizontalHeight2	0.0277	0.0177
petalBendLateral	0.0031	6.1575	petalBendLateralHeight	0.0183	0.0436
petalBendLateralStep	0.0080	0.2087	petalColourHeight	0.0119	0.0538
petalColourHeight2	0.0087	0.0442	sepalsOffset	0.0209	0.0786
sepalsLength	0.0030	0.1323	sepalsWidth	0.0012	0.0921
sepalsUnifyHeight	0.0153	0.0155	sepalsCenterHeight	0.0149	0.0296
sepalsCenterScale	0.0283	0.0725	sepalsBaseSharp	0.0297	0.0272
sepalsTopSharp	0.0105	0.0480	sepalsTopFuzziness	0.0261	0.0267
sepalsBendHorizontal	0.0070	7.9537	sepalsBendHorizontal2	0.0073	6.3064
sepalsBendHorizontalHeight	0.0327	0.0163	sepalsBendHorizontalHeight2	0.0228	0.0119
sepalsBendLateral	0.0161	1.0554	ovaryHeight	0.0113	0.1094
ovaryRadius	0.0036	0.0806	ovaryBaseHeight	0.0044	0.0220
stemHeight	0.0150	1.4553	stemThickness	0.0061	0.0440
stemInclinationFactor	0.0016	0.0180	stemBendHeight	0.0269	0.0026
stemBendLateral	0.0109	0.0747	stemBendLateral2	0.0091	0.0541
stemBendForward	0.0073	0.0066	stemBendForward2	0.0094	0.0060
stemBendHeight2	0.0086	0.0017	pistilBaseHeight	0.0091	0.1467
pistilBaseThicknessTop	0.0202	0.0039	pistilBaseThicknessBottom	0.0293	0.0044
pistilBaseBendHeight	0.0173	0.0098	pistilBaseBendLateral	0.0078	0.0148
pistilBaseBendForward	0.0178	0.0590	pistilBaseFuzziness	0.0129	0.0141
pistilTopHeight	0.0318	0.0033	pistilTopLength	0.0370	0.0027
pistilTopWidth	0.0324	0.0031	stamenHeightInitial	0.0080	0.0408
stamenHeightFinal	0.0157	0.0294	stamenBaseHeight	0.0008	0.0930
stamenBaseThickness	0.0116	0.0056	stamenBaseBendHeight	0.0161	0.0408
stamenBaseBendLateral	0.0224	0.0648	stamenBaseBendForward	0.0179	0.0531
stamenBaseFuzziness	0.0142	0.0475	stamenTopHeight	0.0145	0.0038
stamenTopLength	0.0180	0.0039	stamenTopWidth	0.0124	0.0033
leavesHeightInitial	0.0136	0.0320	leavesHeightFinal	0.0042	0.0209
leavesFuzziness	0.0078	0.0241	leavesLengthMin	0.0091	0.3452
leavesLengthMax	0.0088	0.3536	leavesWidthMin	0.0055	0.1491
leavesWidthMax	0.0066	0.1749	leavesCenterHeightMin	0.0118	0.0434
leavesCenterHeightMax	0.0117	0.0436	leavesCenterScaleMin	0.0189	0.0419
leavesCenterScaleMax	0.0192	0.0418	leavesBaseSharpMin	0.0055	0.0011
leavesBaseSharpMax	0.0193	0.0063	leavesTopSharpMin	6.1138e-05.	0.0004
leavesTopSharpMax	0.0009	0.0033	leavesTopFuzziness	0.0067	0.0190
leavesBendHorizontal	0.0110	5.4676	leavesBendHorizontal2	0.0102	7.3589
leavesBendHorizontalStep	0.0145	0.4024	leavesBendHorizontalStep2	0.0144	0.4045
leavesBendHorizontalHeight	0.0174	0.0129	leavesBendHorizontalHeight2	0.0057	0.0014
leavesBendLateral	0.0044	2.8794	leavesBendLateralStep	0.0023	0.3521

Table 5.3: Regression Parameters Measures Evaluation (floating point parameters)

5.2 Real Images Results

After the preparatory results over the test set, we test our network on real flower images. The results of the network over these images cannot be evaluated as we have done formerly since real pictures lack of labeling.

We are going to compare the original image and a rendering of the predicted 3D model obtained from the parametric representation retrieved by the network. With the aim of providing a more accurate measure than just looking at the images and sentence whether or not the rendering of the predicted model looks like the original flower, we designed the following model based on qualitative questions.

We first developed a set of qualitative comparing questions, which can be easily answered by a human and whose answers can only be yes or no, and give a weight to each of them depending on how much the characteristic investigated by the question matters in the final realization of the flower. For each predicted image, we respond to every question in the set. Consequently, we use the sum of the weighted positives as a measure to evaluate the accuracy of the predicted model.

We distinguish four scopes of investigation: singular petals shape and colour, corolla form, top features (ovary, stamens, pistil) conformation, base features (stem, sepals, leaves) conformation. For each of them we have developed three questions:

- PETAL SHAPE AND COLOUR:
  - Is the overall petal colour similar to the petal colouration in the original image? *(Weight: 2)*
  - Are the overall petals proportions respected w.r.t. the original petals proportions? *(Weight: 2)*
  - Is the top shape of the petals (sharp, smooth, fuzzy) similar to the top shape of the original petals? *(Weight: 2)*
- COROLLA FORM:
  - Is the overall number of rings of the corolla respected w.r.t. the original number of rings? *(Weight: 2)*
  - Is the overall number of petals for every ring of the corolla respected w.r.t. the original number of petals? *(Weight: 2)*
  - Is the overall corolla shape (close, open) similar to the corolla shape of the original image? *(Weight: 2)*

- TOP FEATURES CONFORMATION:
  - Does the ovary resemble the original ovary in shape and colour? *(Weight: 1)*
  - Is stamens presence respected and, if there are stamens, are their overall shape and colour respected too? *(Weight: 1)*
  - Is pistil presence respected and, if there is pistil, is its overall shape and colour respected too? *(Weight: 1)*
- BASE FEATURES CONFORMATION:
  - Does the stem resemble the original stem in shape and colour? *(Weight: 1)*
  - Is sepals presence respected and, if there are sepals, are their overall shape and colour respected too? *(Weight: 1)*
  - Is leaves presence respected and, if there are leaves, are their overall shape and colour respected too? *(Weight: 1)*

We decided to give a weight of 2 to every question about petals and corolla, whilst we give a weight of 1 to every question about the other flower characteristic components, i.e. ovary, stamens, pistil, stem, sepals and leaves. Hence the possible score for each evaluation varies in range [0,18].

5.2.1 Similar Real Images

In the first instance, we try to predict images similar to the images provided for training. We test the network on 50 real images similar to the models chosen for training. Table 5.4 shows some of the results.

We apply our questioning model to every image. The mean total score is: 11.54/18. The means for each of the question scopes, i.e. petals shape and colour, corolla form, top features conformation and base features conformation, are respectively 2.9/6, 3.9/6, 2.4/3, 2.34/3.

We notice that the network is pretty good at predicting flowers like daisies, sunflowers, tulips, alliums and daffodils; whilst it has more problems in recognising flowers similar to roses, yellow clovers, dandelions and hibiscuses. Those are instead often confused with other types of flowers that the network already knows, e.g. alliums or sunflowers. We think that the rose-like and dandelion-like flowers misprediction fashion is referable to the lack of top perspective pictures of those flowers in our dataset, perspective that is instead common to all the real images chosen for testing. As for the yellow clovers and hibiscuses, we reckon that a possible issue that brought

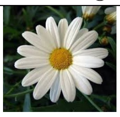
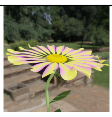
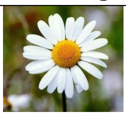
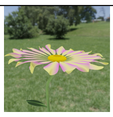
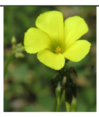
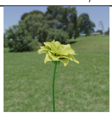
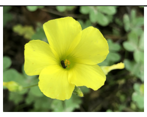
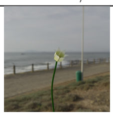
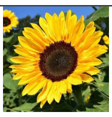
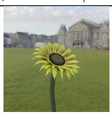
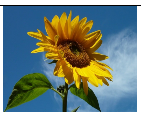
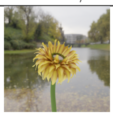
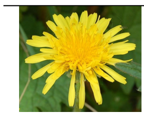
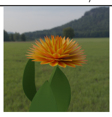
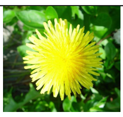
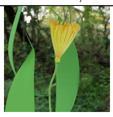
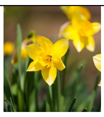
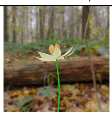
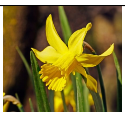
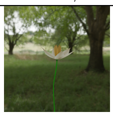
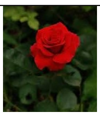
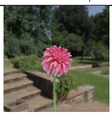
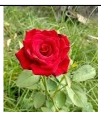
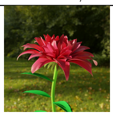
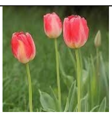
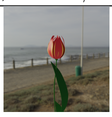
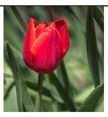
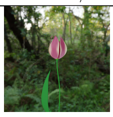
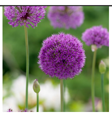
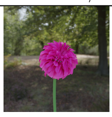
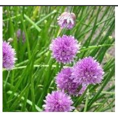
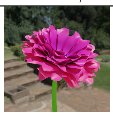
Real Image	Rendered Model	Real Image	Rendered Model
			
P:2 C:6 T:3 B:3, TOT:14/18		P:2 C:6 T:3 B:3, TOT:14/18	
			
P:4 C:0 T:3 B:2, TOT:9/18		P:0 C:4 T:3 B:3, TOT:10/18	
			
P:6 C:6 T:3 B:3, TOT:18/18		P:6 C:4 T:2 B:2, TOT:14/18	
			
P:4 C:4 T:2 B:3, TOT:13/18		P:4 C:0 T:2 B:3, TOT:9/18	
			
P:4 C:6 T:3 B:3, TOT:16/18		P:4 C:6 T:3 B:2, TOT:15/18	
			
P:0 C:0 T:3 B:1, TOT:4/18		P:2 C:4 T:3 B:2, TOT:11/18	
			
P:6 C:6 T:3 B:3, TOT:18/18		P:4 C:4 T:3 B:3, TOT:14/18	
			
P:4 C:6 T:2 B:3, TOT:15/18		P:4 C:4 T:2 B:3, TOT:13/18	

Table 5.4: Comparison between real images similar to the training models and renderings of their corresponding 3D model retrieved by the network. Score: weighted sum of positive answers to the questioning model. Scopes legend: P $\rightarrow$ Petals, C $\rightarrow$ Corolla, T $\rightarrow$ Top, B $\rightarrow$ Base.

to a common misprediction of these kind of flowers is their characteristic size. Indeed, yellow clover images used for training set often present an unbalanced distribution of flower and environmental space: the most of the picture is occupied by the environment and only a small portion of the image is occupied by the actual pixels of the flower. On the other hand, hibiscus training images present the opposite problem, i.e. they space is mostly occupied by the target flower. In the selected real images instead, subjects occupy harmonically the scene and result all in medium zoomed visuals. Consequently we think that a first possible solution to this problem could be improve zooming balance of rendered images during dataset collection. Also, adding random zooming as a real-time data augmentation pre-process could help.

We are going to further discuss on these result in chapter 6. Now we show the general flowers real images results.

5.2.2 General Real Images

After testing our network on images similar to the images provided for training, we test it with general flower images, i.e. pictures whose flowers are not similar to the initiatory flower models used for training. We test the network on 50 general flower images. In table 5.5 some of the results are shown.

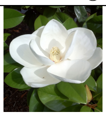
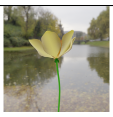
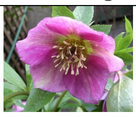
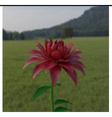
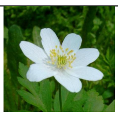
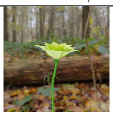
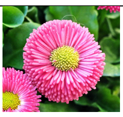
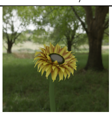
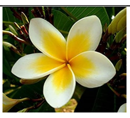
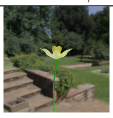
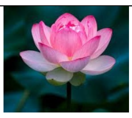
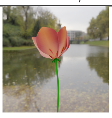
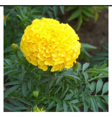
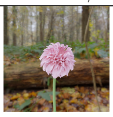
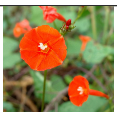
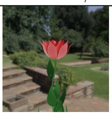
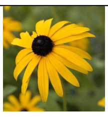
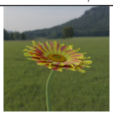
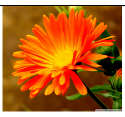
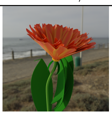
Real Image	Rendered Model	Real Image	Rendered Model
			
P:4 C:6 T:3 B:2, TOT:15/18		P:0 C:0 T:1 B:3, TOT:4/18	
			
P:4 C:4 T:3 B:2, TOT:13/18		P:0 C:4 T:1 B:1, TOT:6/18	
			
P:6 C:4 T:2 B:2, TOT:14/18		P:6 C:2 T:3 B:2, TOT:13/18	
			
P:2 C:6 T:2 B:2, TOT:12/18		P:2 C:0 T:1 B:1, TOT:4/18	
			
P:2 C:0 T:1 B:3, TOT:6/18		P:4 C:6 T:3 B:2, TOT:15/18	

Table 5.5: Comparison between general real images (not similar to the training models) and renderings of their corresponding 3D model retrieved by the network. Score: weighted sum of positive answers to the questioning model. Scopes legend: P → Petals, C → Corolla, T → Top, B → Base.

Applying our questioning model to the images, the mean total score is: 9.02/18. The means for each of the question scopes, petals shape and colour, corolla form, top features conformation and base features conformation, are respectively 2.5/6, 2.62/6, 1.99/3, 1.91/3.

The evaluation means are lower than the means of the similar flower real images. With general flowers real images we can see that our network struggles to identify the right colours and shape in most of the cases, though sometimes grasps the overall structure. One of the best results is showed in the last row at the second column of table 5.5, here we can see that the network have almost completely understood the colour and the structure of the flower, even if no daisy-like flower used in input had this orange as petal colour or these open corolla connotation. Anyway, more than one daisy-like model was used to train the network (normal daisy, bumped blue daisy, yellow fuzzy daisy). Therefore we think that more variation over the initiatory models and more sample could significantly improve the performances. Further proposals and comments are made in chapter 6.

5.3 Exploitation

In this section we show some possible exploitation for our work.

First of all, having a tool that can generate a modifiable model of a flower from an image can be useful in entertainment industry (games, films) and in architectural renderings. Such a tool can be integrated in softwares like SketchUp, Revit, Lumion, Unity and Unreal engines (figure 5.2) as a package for fast modeling of environment flowers. Since the model is firstly given with a parametric representation, it is possible to change the flower characteristics and obtain a variate flower collection, adding variety to the scene in a fast and simple way.

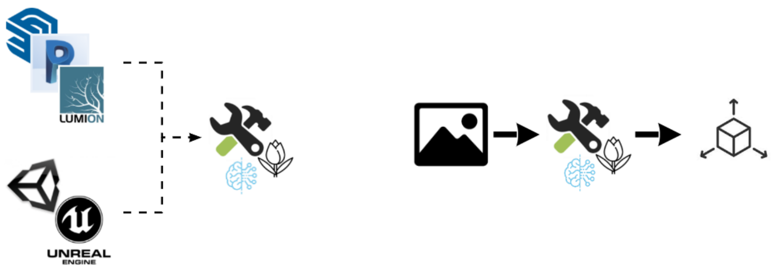
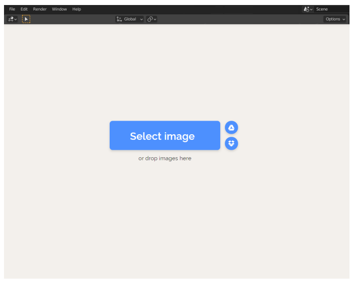
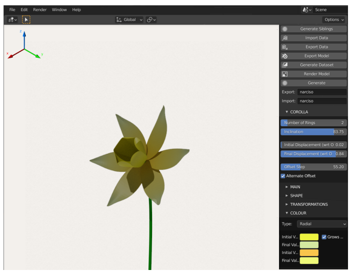


Figure 5.2: Flower generator as a package for architectural and entertainment rendering softwares.

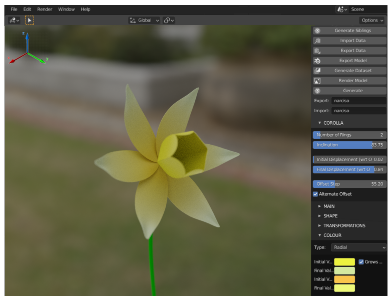
We can also think about a stand-alone application. Such a software can be used by 3D artists for modeling sake, and it could also be used in educational field for floral characteristics studies. From a photo of a flower, we rebuild a similar 3D model and change its parameters to show its variation. Figure 5.3 shows some mock-ups for the application. First the real picture is selected and loaded (figure 5.3a), then the 3D model is retrieved and showed on screen in a rendering window (figure 5.3b). From a menu, the users can interact with the parameters and change the values to variate the model. There should also be the possibility to add a 3D background, i.e. a HDR image, to better visualize the model in a realistic environment (figure 5.3c).



(a) Selecting the image



(b) Building the 3D model and playing with parameters



(c) Adding the flower in a 3D environment, such a HDR image

Figure 5.3: Mock-ups for stand-alone application

Augmented reality has been given new life in the past few years as an innovative technology that could transform social media as we know it. Snapchat was the first social media platform to incorporate AR with its fun and interactive lenses, and other social networks are now scrambling to follow suit. The latest of these is Instagram, which has just jumped onto the augmented reality bandwagon with face filters.

According to these new trends, another possible application of our work could be a social flower filter: users would be able to use their smartphone to take a picture at some real flower, modify their features as desired and add the on-the-fly 3D flower model to their pictures and stories.

Chapter 6

Conclusions

This thesis aimed to generate a modifiable 3D model from a single image of a flower, exploiting an ad hoc PMT and recent deep learning technologies, in order to serve an accessible way to obtain a variety of similar flower models from the original flower image.

In the first stage of the thesis, the flower parametric representation has been designed, the ad hoc modeling tool has been realized as an add-on of the well known software Blender. This tool had two purposes in our project. The first was to reconstruct the 3D models of the flowers once our parameters had been recognized by the deep learning network, the second was to gather the data to train our network. Some initiatory models have been chosen. Some variations in shape and colour have broad to an extension of the model base. Accordingly, various techniques to augment data were taken: the addition of HDRI and consequent IBL, the addition of shadows and out of focus similar siblings, the randomization of the camera. Theoretically speaking, with these techniques, we were able to realize an infinite number of realistic flower images, all good for training. We collected about 1300 realistic images that constituted our dataset. Along with them, we stored the information of the parameters, so that we were ready to train our network.

In the second stage of the thesis we designed a network to be trained on the data collected. Our goal was to teach it how to recognise the main characteristics of a flower from an image. We relied on transfer learning, and exploited the most known VGG model for the first module of the network, dedicated to image understanding. Classification and regression branches were built on top of it, in order to predict the parameters for the flower representation. Data was pre-processed, in order to improve the network performances. Input was resized and normalized, and real time data augmentation was adopted to generalize our input. We also prepared and normalized the parameters to be predicted. The dataset was split into train, validation and test set. We used ADAM optimizer to minimize the total loss of our network, expressed as a combination of BCE (for the classification parameters) and MSE (for the regression parameters). The network thus obtained was tested on the test set and on real images, as described in chapter 5.

The network has learnt how to recognize the images in the dataset. Indeed, the metrics evaluated over the test set, the a small portion of our original dataset chosen to test the network once trained, are generally fulfilling the expected results. Regarding real images results: we have made a distinction between similar images prediction and general images prediction. Even if the input image is a real picture of a flower, when our network try to predict a flower which is similar to a model we used for our training, the overall results are still generally good. On general real flower images our network performances should be improved. We try here to explore some possible ways to increase general real images results:

- A first proposal is to increment our dataset size and assortment. In fact, a main issue that can be inferred by looking at the experimental results is that our network mostly "classify" inputs instead of finding regression values of the parameters: the models retrieved are sometimes similar to the ones used in training phase, even when the real flower images are much different from them. Therefore, a possible approach could be to spend more time on data collection, adding more variation to the initiatory models.

- Given the Rose issue, i.e. the struggle of the network to recognize roses because most of the input data had a lateral perspective whilst all the test data had a top perspective, a better balancing of perspectives in data collection phase could surely improve performances.

- Introducing some other real time data augmentation techniques, such as random zooming, could improve network performances. Indeed, our network has some problems on recognition of both small and large sized flowers. Maybe, adding such random zooming, understanding of small subjects and full space occupying subject could be enhanced.

- Variations could be conducted on the main features of our network: from the main training parameters (i.e. learning step, batch size, number of epochs) to the optimizer algorithm and the loss functions. Sometimes, changing these high level parameters, better results can be achieved.

- Other CNN models in place of VGG network could be taken into account for the image understanding part. Resnet [10] and Densenets [11] obtained optimal results over the state-of-the-art on the highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet).

- Further improvements could be achieved changing the latter modules of the network, i.e. the branches designed for the output prediction.

Eventually, we are going to present a set of possible future works that may start from our project:

- We might think of a first extension of our work as the objective passes from one modifiable 3D flower model to multiple models retrieval from the same image. If images present more types of flowers, i.e. flower fields or flower bouquet, a possible proposal would be to develop our project to recognize every flower present in the image. In this way, multiple modifiable 3D models could be retrieved at once.

- Another interesting work could consist in retrieval of 3D model representations that not only are bring mutability in shape and colour, but that also support the basis for animations of blooming, withering, bending to wind, growing and so on. In this way, flower behaviours could be analysed by a single image of it. This kind of work would surely constitute an interesting feature for education. Moreover, entertainment field would largely benefit from such a project.

- Finally, we imagine expanding our approach to many other useful categories for 3D scene environments, like trees, plants, buildings, etc. Indeed, having a full tool box for retrieval of modifiable environmental 3D objects from single images could spare a lot of time and resources in recreating 3D scenes, with the benefit of maintaining variety and complexity.

Bibliography

[1] Jing Chen, Long Cheng, Xi Yang, Jun Liang, Bing Quan, and Shoushan Li. Joint learning with both classification and regression models for age prediction. *Journal of Physics: Conference Series*, 1168:032016, feb 2019.

[2] F. Chi, W. Kurth, and K. Streit. Generating 3d models from a single 2d digitized photo using gis and groimp. In *2016 IEEE International Conference on Functional-Structural Plant Growth Modeling, Simulation, Visualization and Applications (FSPMA)*, pages 22–27, 2016.

[3] Musa CIBUK, Umit Budak, Yanhui Guo, M. Ince, and Abdulkadir Sengur. Efficient deep features selections and classification for flower species recognition. *Measurement*, 137, 04 2019.

[4] P.N. Druzhkov and V.D. Kustikova. A survey of deep learning methods and software tools for image classification and object detection. Technical Report 1, 2016. cited By 115.

[5] I. Gogul and Sathiesh Kumar. Flower species recognition system using convolution neural networks and transfer learning. pages 1–6, 03 2017.

[6] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation, 2015.

[7] Jiuxiang Gu, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu, Xingxing Wang, Gang Wang, Jianfei Cai, and Tsuhan Chen. Recent advances in convolutional neural networks. *Pattern Recognition*, 77:354–377, 2018.

[8] Jérôme Guénard, Géraldine Morin, Frédéric Boudon, and Vincent Charvillat. Reconstructing Plants in 3D from a Single Image Using Analysis-by-Synthesis. In Bebis, George, Boyle, Victor, Klosowski,

James, Coquillart, Sabine, Luo, Xun, Chen, Min, Gotz, David, Richard, Parvin, Bahram, Koracin, Darko, Li, Baoxin, Porikli, Fatih, and Zordan, editors, *Advances in Visual Computing*, volume 8033 of *Lecture Notes in Computer Science*, pages 322–332. Springer Berlin Heidelberg, 2013.

[9] Xianfeng Han, Hamid Laga, and Mohammed Bennamoun. Image-based 3d object reconstruction: State-of-the-art and trends in the deep learning era. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2019.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[11] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks, 2018.

[12] Takashi Ijiri, Shigeru Owada, Makoto Okabe, and Takeo Igarashi. Floral diagrams and inflorescences: Interactive flower modeling using botanical structural constraints. *ACM Transactions on Graphics*, 24(3):720–726, July 2005. ACM SIGGRAPH 2005 ; Conference date: 31-07-2005 Through 04-08-2005.

[13] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[14] Xiaoxue Li, Rongxin Lv, Yanzhen Yin, Kangkang Xin, Zeyuan Liu, and Zhongzhi Li. Flower image classification based on generative adversarial network and transfer learning. *IOP Conference Series: Earth and Environmental Science*, 647:012180, 01 2021.

[15] Leo Lou. Fine-grained flower classification. 2016.

[16] Usha Mittal, Sonal Srivastava, and Priyanka Chawla. Review of different techniques for object detection using deep learning. In *Proceedings of the Third International Conference on Advanced Informatics for Computing Research*, ICAICR '19, New York, NY, USA, 2019. Association for Computing Machinery.

[17] M. Nilsback and A. Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, pages 722–729, 2008.

[18] Yuji Roh, Geon Heo, and Steven Euijong Whang. A survey on data collection for machine learning: a big data – ai integration perspective, 2019.

[19] Ilya Shlyakhter, M. Rozenoer, Julie Dorsey, and S. Teller. Reconstructing 3d tree model from instrumented photograph. *Computer Graphics and Applications, IEEE*, 21:53–61, 06 2001.

[20] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[21] Ping Tan, Tian Fang, Jianxiong Xiao, Peng Zhao, and Long Quan. Single image tree modeling. In *ACM SIGGRAPH Asia 2008 Papers*, SIGGRAPH Asia '08, New York, NY, USA, 2008. Association for Computing Machinery.

[22] Luís Torgo and João Gama. Regression by classification. In Díbio L. Borges and Celso A. A. Kaestner, editors, *Advances in Artificial Intelligence*, pages 51–60, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg.

[23] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention, 2016.

[24] Feilong Yan, Minglun Gong, Daniel Cohen-Or, Oliver Deussen, and Baoquan Chen. Flower reconstruction from a single photo. *Computer Graphics Forum*, 33(2):439–447, 2014.

[25] Gang Zeng, Jingdong Wang, Sing Bing Kang, and Long Quan. Image-based tree modeling. *ACM Trans. Graph.*, 26:87, 07 2007.

[26] J. Zeng, Y. Zhang, and S. Zhan. 3d tree models reconstruction from a single image. In *Sixth International Conference on Intelligent Systems Design and Applications*, volume 2, pages 445–450, 2006.

[27] Bo Zhao, Jiashi Feng, Xiao Wu, and Shuicheng Yan. A survey on deep learning-based fine-grained object classification and semantic segmentation. JOUR, 2017.

Appendix A

Flower Characteristic Parameters Detailes

- petalRingsNumber : Number of rings of petals.
- petalHeightInitial : Starting position of first (or unique) ring of petals w.r.t. the height of the ovary (starting from the base of ovary); domain [0, 1].
- petalHeightFinal : Starting position of last ring of petals w.r.t. the height of the ovary (starting from the base of ovary); domain [0, 1].
- petalOffset : Rotation step in degrees of any ring of petals with the underlying ring; domain [0, 180].
- petalOffsetAlternate : Switch on/off to alternate rotation of rings (i.e. 1st clockwise, 2nd counterclockwise, etc).
- petalNumberMin : Number of petals of first (or unique) ring.
- petalNumberMax : Number of petals of last ring.
- petalLengthMin : Length of petals of first (or unique) ring.
- petalLengthMax : Length of petals of last ring.
- petalWidthMin : Width of petals of first (or unique) ring.
- petalWidthMax : Width of petals of last ring.
- petalWidthLimit : Switch on/off to limit petal width of each ring to avoid overlaps in the ring.

- petalWidthLimitRotate : Switch on/off to rotate automatically petals of each ring to avoid overlaps in the ring.
- petalUnify : Switch on/off to unify contiguous petals in the rings (starting from the base of petals).
- petalUnifyOnlyFirst : Switch on/off to unify contiguous petals only in the first ring.
- petalUnifyHeight : Point where the contiguous petals unification should stop. It is expressed as a percentage w.r.t. petal length. Domain [0, 1].
- petalRotationMin : Determines petals rotation, w.r.t. longitudinal axis (exiting from the ovary), of petals of first (or unique) ring. It is expressed in degrees. Domain [-45, 45].
- petalRotationMax : Determines petals rotation, w.r.t. longitudinal axis (exiting from the ovary), of petals of last ring. Domain [-45, 45].
- petalCenterHeightMin : Determines the point where the petals first (or unique) ring should have their width scaled (the basic shape is a circle). It is expressed as a percentage of the total petal Length. The scaling value is described by petalCenterScaleMin.
- petalCenterHeightMax : Determines the point where the petals of last ring should have their width scaled (the basic shape is a circle). It is expressed as a percentage of the total petal Length. The scaling value is described by petalCenterScaleMax.
- petalCenterScaleMin : Scaling factor to be applied at the point expressed by petalCenterHeightMin. Refers to the petals of the first (or unique) ring. Domain [0.50, 3.00].
- petalCenterScaleMax : Scaling factor to be applied at the point expressed by petalCenterHeightMax. Refers to the petals of the last ring. Domain [0.50, 3.00].
- petalBaseForm : Form of the base of petal; domain ["Curve", "Sharp"].
- petalBaseSharpMin : Level of sharpness (active only if petalBaseForm : Sharp) of the base of petals of first (of unique) ring; domain [0, 1].
- petalBaseSharpMax : Level of sharpness (active only if petalBaseForm : Sharp) of the base of petals of last ring; domain [0, 1].

- petalTopForm : Form of the top of petal; domain ["Curve", "Sharp", "Double"].
- petalTopSharpMin : Level of sharpness (active only if petalTopForm : Sharp) of the top of petals of first (of unique) ring; domain [0, 1].
- petalTopSharpMax : Level of sharpness (active only if petalTopForm : Sharp) of the top of petals of last ring; domain [0, 1].
- petalTopFuzziness : Level of fuzziness of the wavy edge of the top of petals; domain [0, 1].
- petalBendHorizontalHeight : Position of first point of bending of the petal. It is expressed as a percentage of the length of the petal, starting from the base of petal. Domain [0.00, 0.25].
- petalBendHorizontalHeight2 : Position of second point of bending - if any -. It is expressed as a percentage of the length of the petal, starting from the base of petal. Domain [0.75, 1.00].
- petalBendHorizontal : Value in degrees of the first bending (concave or convex) of petals of the first (or unique) ring; domain [-90, 90].
- petalBendHorizontal2 : Value in degrees of the second bending - if any - (concave or convex) of petals of the first (or unique) ring; domain [-90, 90].
- petalBendHorizontalStep : Step value in degrees to add to the first petal bending value of the petals of each other ring; domain [0, 50].
- petalBendHorizontalStep2 : Step value in degrees to add to the second petal bending value of the petals of each other rings; domain [0, 50].
- petalBendLateral : Value in degrees of bending (concave or convex) of the two symmetrical sides of the petals of the first (or unique) ring towards their symmetrical axis; domain [-180, 180].
- petalBendLateralHeight : Position of starting of lateral bending w.r.t. the width of the petal; domain [0, 1].
- petalBendLateralStep : Step value in degrees to add to the lateral petal bending value of the petals of each other ring; domain [0, 20].
- petalColourType : Kind of deployment of the colour on the petals; domain ["Simple", "Radial" (from the base to the top), "Inner Outer" (from vertical axis to lateral edges)].

- petalColourHeight : Point of starting of interpolation between the initial colour and the final one w.r.t. the length of petal/ the width of the petal (depending on petalColourType parameter); domain [0, 1].

- petalColourHeight2 : Point of ending of interpolation between the initial colour and the final one w.r.t. the length of petal/ the width of the petal (depending on petalColourType parameter); domain [0, 1].

- petalColourMin : Colour [RGBA] of the whole petal/of base of petal/of vertical axis of petal (depending on petalColourType parameter) of the petals of first (or unique) ring.

- petalColourMax : Colour [RGBA] of the whole petal/of base of petal/of vertical axis of petal (depending on petalColourType parameter) of the petals of last ring.

- petalColour2Min : Colour [RGBA] of top of petal/of lateral edges of petal (depending on petalColourType parameter) of the petals of first (or unique) ring.

- petalColour2Max : Colour [RGBA] of top of petal/of lateral edges of petal (depending on petalColourType parameter) of the petals of last ring

- flowerTopInclination : Inclination (in degrees) of the whole flower w.r.t. the stem; domain [-135, 135]

- includeSepals : Switch on/off to determine the presence of the sepals.

- sepalsNumber : Number of sepals.

- sepalsLength : Length of sepals.

- sepalsWidth : Width of sepals.

- sepalsWidthLimit : Switch on/off to automatically reduce sepal width to avoid overlaps of contiguous sepals.

- sepalsUnify : Switch on/off to unify contiguous sepals (starting from the base of sepal).

- sepalsUnifyHeight : Point where the contiguous sepals unification should stop. It is expressed as a percentage w.r.t. petal length. Domain [0, 1].

- sepalsOffset : Rotation value of the ring of sepals w.r.t. the first petal ring standing above it; domain [0, 1].
- sepalsCenterHeight : determines the point where the sepals should have their width scaled (the basic shape is a circle). It is expressed as a percentage of the total sepal Length. The scaling value is described by sepalCenterScale.
- sepalsCenterScale : scaling factor to be applied at the point expressed by sepalCenterHeight. Domain [0.50, 2.00].
- sepalsBaseForm : Form of the base of sepal; domain ["Curve", "Sharp"].
- sepalsBaseSharp : level of sharpness (active only if sepalsBaseForm : Sharp) of the base of sepals; domain [0, 1].
- sepalsTopForm : Form of the top of sepal; domain ["Curve", "Sharp", "Double"].
- sepalsTopSharp : level of sharpness (active only if sepalsTopForm : Sharp) of the top of sepals; domain [0, 1].
- sepalsTopFuzziness : Level of fuzziness of the wavy edge of the top of sepals; domain [0, 1].
- sepalsBendHorizontalHeight : Position of first point of bending. It is expressed as a percentage of the length of the sepal, starting from the base of sepal. Domain [0.00, 0.25].
- sepalsBendHorizontalHeight2 : Position of second point of bending. It is expressed as a percentage of the length of the sepal, starting from the base of sepal. Domain [0.75, 1.00].
- sepalsBendHorizontal : Value in degrees of the first bending (concave or convex) of sepals (starting from the base of sepal); domain [-90, 90].
- sepalsBendHorizontal2 : Value in degrees of the second bending (concave or convex) of sepals (starting from the base of sepal); domain [-90, 90].
- sepalsBendLateral : Value in degrees of bending (concave or convex) of the two symmetrical sides of the sepals towards their symmetrical axis; domain [-180, 180].
- sepalsColour : Colour [RGBA] of the whole sepal.

- ovaryHeight : Heigth of the ovary.
- ovaryRadius : Radius of the ovary.
- ovaryBaseHeight : Point of the ovary where stem colour stops and actual ovary colour starts. It is expressed as a percentage of the total ovary Height. Domani [0, 0.5].
- ovaryColourType : Kind of deployment of colour in the ovary; domain ["Simple", "Bottom-Top"].
- ovaryColour : Colour [RGBA] of the whole ovary/ of the base of ovary (depending on ovaryColourType parameter).
- ovaryColour2 : Colour [RGBA] of the top of the ovary (active only if ovaryColourType : Bottom Top).
- stemHeight : Height of the stem.
- stemThickness : Width of the stem w.r.t the radius of the ovary; it is expressed as a percentage. Domain [0, 1].
- stemInclinationFactor : Extent of arch of the stem w.r.t. the inclination of the head of the flower. Domain [0, 1].
- stemBendHeight : Position of first bending of the stem. It is expressed as a percentage of the stem height, starting from the base of stem; Domain [0.10, 0.40].
- stemBendHeight2 : Position of second bending of the stem. It is expressed as a percentage of the stem height, starting from the base of stem; Domain [0.60, 0.90].
- stemBendLateral : Lateral bending (right/left direction) of the stem applied on first bending point; it is proportional to the stem total height. Domain [-1 , 1].
- stemBendLateral2 : Lateral bending (right/left direction) of the stem applied on second bending point; it is proportional to the stem total height. Domain [-1 , 1].
- stemBendForward : Frontal bending (forward/reverse direction) of the stem applied on first bending point; it is proportional to the stem total height. Domain [-1 , 1].

- stemBendForward2 : Frontal bending (forward/reverse direction) of the stem applied on second bending point; it is proportional to the stem total height. Domain [-1 , 1].
- stemColour : Colour [RGBA] of the whole stem.
- includePistil : Switch on/off to determine the presence of the pistil.
- pistilBaseHeight : Height of the pistil.
- pistilBaseThicknessTop : Radius of the top of the pistil.
- pistilBaseThicknessBottom : Radius of the base of the pistil.
- pistilBaseBendHeight : Point of bending of the pistil. It is expressed as a percentage of the height of the pistil, starting from the base of pistil; domain [0.10, 0.90].
- pistilBaseBendLateral : Lateral bending (right/left direction) of the pistil applied on the bending point; it is proportional to the pistil height. Domain [-1 , 1].
- pistilBaseBendForward : Frontal bending (forward/reverse direction) of the pistil applied on the bending point; it is proportional to the pistil height. Domain [-1 , 1].
- pistilBaseFuzziness : Level of fuzziness of the top of the pistil; domain [0, 1].
- pistilBaseColour : Colour [RGBA] of the pistil.
- pistilTopNumber : Number of elements making up the stigma.
- pistilTopHeight : Height of each element of the stigma.
- pistilTopLength : Length of each element of the stigma.
- pistilTopWidth : Width of each element of the stigma.
- pistilTopColour : Colour [RGBA] of the elements of the stigma.
- includeStamens : Switch on/off to determine the presence of the stamens.
- stamenOnPistil : Switch on/off to determine if the stamens are placed on the pistil and not on the ovary.

- stamenHeightInitial : Initial position of the placement of the stamens on top of the ovary/ pistil depending on stamenOnPistil parameter. It is expressed as a percentage of the height of the ovary/pistil, starting from the base; domain [0, 1].
- stamenHeightFinal : Final position of the placement of the stamens on the height of the ovary/ pistil depending on stamenOnPistil parameter. It is expressed as a percentage of the height of the ovary/pistil, starting from the base; domain [0, 1].
- stamenNumber : Number of stamens.
- stamenBaseHeight : Hight of each stamen.
- stamenBaseThickness : Radius of each stamen.
- stamenBaseBendHeight : Point of bending of the stamens. It is expressed as a percentage of the height of the stamens, starting from the base of stamens; domain [0.10, 0.90].
- stamenBaseBendLateral : Lateral bending (right/left direction) of the stamens applied on the bending point; it is proportional to the stamens height. Domain [-1 , 1].
- stamenBaseBendForward : Frontal bending (forward/reverse direction) of the stamens applied on the bending point; it is proportional to the stamens height. Domain [-1 , 1].
- stamenBaseFuzziness : Level of fuzziness of the stamens on the ovary/pistil, depending on stamenOnPistil parameter; domain [0, 1].
- stamenBaseColourType : Kind of deployment of colour on the stamen; domain ["Simple", "Bottom-Top"].
- stamenBaseColour : Colour [RGBA] of each stamen in its entirety/ of the base of each stamen (depending on stamenBaseColourType parameter).
- stamenBaseColour2 : Colour [RGBA] of the top of each stamen (active only if stamenBaseColourType : Bottom Top).
- stamenType : Kind of the anthers; domain ["Mono" (piece), "Double" (pieces)]
- stamenTopHeight : Height of each piece of the anthers.

- stamenTopLength : Length of each piece of the anthers.
- stamenTopWidth : Width of each piece of the anthers.
- stamenTopColour : Colour [RGBA] of each piece of the anthers.
- includeLeaves : Switch on/off to determine the presence of the leaves on the stem.
- leavesRingsNumber : Number of petioles.
- leavesHeightInitial : Position of the placement of the first petiole. It is expressed as a percentage of the height of the stem, starting from the base of the stem; domain [0, 1].
- leavesHeightFinal : Position of the placement of the last petiole. It is expressed as a percentage of the height of the stem, starting from the base of the stem; domain [0, 1].
- leavesLengthMin : Length of the first leaf.
- leavesLengthMax : Length of the last leaf.
- leavesWidthMin : Width of the first leaf.
- leavesWidthMax : Width of the last leaf.
- leavesFuzziness : Level of fuzziness of the entire leaves; domain [0, 1].
- leavesCenterHeightMin : Determines the point where the first leaf should have its width scaled (the basic shape is a circle). It is expressed as a percentage of the total leaf Length. The scaling value is described by leafCenterScaleMin.
- leavesCenterHeightMax : Determines the point where the last leaf should have its width scaled (the basic shape is a circle). It is expressed as a percentage of the total leaf Length. The scaling value is described by petalCenterScaleMax.
- leavesCenterScaleMin : Scaling factor to be applied at the point expressed by leavesCenterHeightMin. Refers to the first leaf. Domain [0.50, 3.00].
- leavesCenterScaleMax : Scaling factor to be applied at the point expressed by leavesCenterHeightMax. Refers to the last leaf. Domain [0.50, 3.00].

- leavesBaseForm : Form of the base of the leaves; domain ["Curve", "Sharp"]
- leavesBaseSharpMin : Level of sharpness (active only if leavesBaseForm : Sharp) of the base of the first leaf; domain [0, 1].
- leavesBaseSharpMax : Level of sharpness (active only if leavesBaseForm : Sharp) of the base of the last leaf; domain [0, 1].
- leavesTopForm : Form of the base of the leaves; domain ["Curve", "Sharp", "Double", "Cut"].
- leavesTopSharpMin : Level of sharpness (active only if leavesTopForm : Sharp) of the top of the first leaf, domain [0, 1].
- leavesTopSharpMax : Level of sharpness (active only if leavesTopForm : Sharp) of the top of the last leaf, domain [0, 1].
- leavesTopFuzziness : Level of fuzziness of the top of the leaves; domain [0, 1].
- leavesBendHorizontalHeight : Position of first point of bending. It is expressed as a percentage of the length of the leaf, starting from the base of leaf. Domain [0.00, 0.25].
- leavesBendHorizontalHeight2 : Position of second point of bending. It is expressed as a percentage of the length of the leaf, starting from the base of leaf. Domain [0.75, 1.00].
- leavesBendHorizontal : Value in degrees of the first bending (concave or convex) of the first leaf; domain [-90, 90].
- leavesBendHorizontal2 : Value in degrees of the second bending (concave or convex) of the first leaf; domain [-90, 90].
- leavesBendHorizontalStep : Step value in degrees to add to the first leaf bending value of the subsequent leaves, domain [0, 20].
- leavesBendHorizontalStep2 : Step value in degrees to add to the second leaf bending value of the subsequent leaves, domain [0, 20].
- leavesBendLateral : Value in degrees of bending (concave or convex) of the two symmetrical sides of the leaves towards their symmetrical axis; domain [-180, 180].

- leavesBendLateralStep : Step value in degrees to add to the lateral petal bending value of the subsequent leaves, domain [0, 20].
- leavesColourMin : Colour [RGBA] of the first leaf.
- leavesColourMax : Colour [RGBA] of the last leaf.