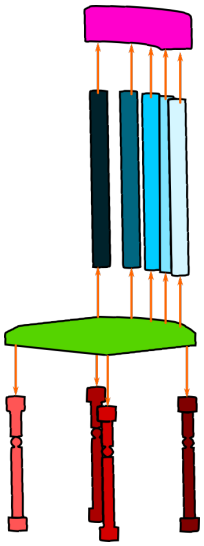
Exploration of Exploded View Diagrams for Schematic 3D analysis: a study on PartNet Dataset



Marzia Favaro

Master Thesis
May 2022

Under the supervision of:
Byungsoo Kim
CGL Simulation Animation Group
Prof. Dr. Markus Gross

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

cgl
computer graphics laboratory

POLITECNICO MILANO

Abstract

Low-quality pictures from the past are going to be lost if not digitalized and restored.
This work focuses on the restoration of exploded view diagrams, which represent a particular subset of those images. The peculiarity of these images is their simplicity, which is both an advantage and a disadvantage since the little information they provide is easier to handle but harder to understand.

The final goal of the study is to both reconstruct their individual components and rebuild the composite object. To do this, we need to generate a dataset of diagrams and their relative information that will be used for the application of reverse-engineering models which can restore the images in 2D or 3D.

Most of the studies on the depth estimation of a scene have been developed on real pictures or realistic simulations of scenes which include non-uniform lighting. The understanding of the depth in an image or a video, in those cases, heavily relies on the different shades that the light creates when hitting objects and shadows. However, we totally miss this information in the exploded views, and this opens a new type of challenge.

From segmented 3D models, we reconstruct their components' connections and explode them, proposing an algorithm for the generation of the diagrams. We study the possibilities and choose them with an eye on scalability and generalization, trying to adapt existing studies to ours.

In conclusion, we recognize how a set of ad-hoc techniques may outperform a general universal algorithm.

Introduction

Exploded view diagrams are a practical representation of the components of a 3D object which must also be easily understandable to people without expertise. They allow visualizing the otherwise hidden parts by decomposing the object into its components and showing them as separate entities. With respect to a classical composite view, exploded views not only remove the occlusions between components but also convey the information about their relationships through the way they have been exploded. In fact, the displacement of the components has to respect the hierarchy of dependencies they had in the original object, that is, it must be easily comprehensible where each part was connected. On the other hand, because it's often sufficient to provide a single 2D view to explain the object, the depth component is lost.

Reconstructing the relationships and estimating the depth of the components is an intrinsically ambiguous problem when dealing with single 2D views, as it is not possible to find a unique valid solution for the problem; however, not all the valid possibilities are equally likely. In the reconstruction phase is then fundamental to take into consideration the prior knowledge about the diagrams and the 3D objects behind those, in order to find the reconstruction which maximizes the posterior likelihood between all the possible choices. The rules that lead to a solution may come from direct human observations or via supervised learning techniques.

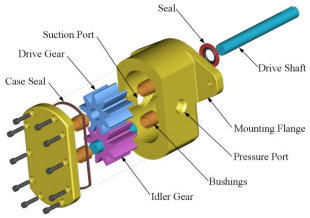
Because of the lack of existing annotated datasets of exploded diagrams, we build one by automatically generating it from the PartNet dataset. By studying the semantics that it provides, we define some heuristics that allow us to reconstruct the connections between components and explode them accordingly. We also propose a pipeline for information extraction and reconstruction of the diagrams.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics laboratory

Master Thesis

Segmentation of Exploded-View Drawing for Schematic 3D Reconstruction



Student Name: Marzia Favarro
Start Date: 01/11/2021
End Date: 29/04/2022

Introduction

We would like to work on the segmentation of parts of an object in an exploded-view drawing. Many exploded diagrams are drawn by hand and thus stored in a rasterized image format, which is limited in its magnification. In this project, we aim to separate all components of an assembly for the schematically meaningful 3D reconstruction of each component. Previous works introduced the first semantic segmentation of line drawings, and the goal of this thesis is to develop both non-ML / ML approach.

Task Description

We will first go into the details of previous and the state-of-the-art works first. We will mainly focus on synthetic generation of exploded-view drawing, non-ML based segmentation of components and finally ML-based approach to overcome issues from the counter part.

Skills

- Computer Graphics or Vision / Deep Learning
- Hands-on experience with open-source deep-learning frameworks such as PyTorch and TensorFlow
- Experience with Python and C/C++

Remarks

A written report and an oral presentation conclude the thesis. The thesis will be overseen by Prof. Markus Gross and supervised by Dr. Byungsoo Kim and Simulation and Animation Group at Computer Graphics Lab. Please contact Dr. Byungsoo Kim (kimby@inf.ethz.ch) for more information.

Contact

For further information, please contact the thesis coordinator (cgl-thesis@inf.ethz.ch)

Acknowledgment

To my supervisor Byungsoo Kim, thank you for your support and advice, your hints have been essential to overcoming the problems encountered.
Many thanks also to the whole Computer Graphics Laboratory of ETH, Prof. Dr. Markus Gross and Politecnico di Milano for having made this experience possible.

To all my friends, thank you for supporting me during the hardest times, for laughing and crying together. Special thanks to Nicoló, for your continuous encouragement, help and support, without which I wouldn't have been able to reach this important milestone.
To my siblings, thanks for the fun we had together. To my parents and grandparents, thank you for having always respected my decisions even when you couldn't understand them, and for always being supportive of what I do.

Contents

List of Figures

Background

1.1 Related work

1.1.1 Exploded view diagrams

The automatic generation of exploded view illustrations has been a subject of interest since the ’90s, when the process was starting to be computer-aided.
The solution proposed by [DC]:

- Requires human intervention to build the tree of connections between the pieces;
- From the tree of its components explodes an object by iteratively distancing its components;
- Is too manual to build a dataset efficiently.

A more complete automation tool has been studied by Li in [LAS], which:

- Analyzes the blocking constraints between parts (which components can not be moved before others do);
- Builds the explosion graph of the object from the blocking constraints;
- Works well in presence of many interlocking pieces, but for our applications this study is not enough because, as we will see, the data we use to develop our dataset lacks many intersections between components that in real life would exist.

Dataset

Because we are compacting the information about a 3D object into a single 2D image, the problem of its reconstruction is intrinsically ambiguous, and we need to be able to choose the most likely solution between the infinitely many compatible ones. For this reason, we need to have some prior knowledge to get the common sense of what is the most likely interpretation of the image, and this can be provided by using a dataset of diagrams whose meaning is known.
Such a dataset is not yet available, so we are going to build one ourselves. Because manually annotating existing illustrations is extremely expensive, we will build an automatically generated one.

- We took into consideration building objects from mechanical parts, for example from ABC dataset [KMJ+], and we observed that placing arbitrary objects on a randomly generated explosion graph is not a good choice, as the lack of semantic meaning impacted too strongly the quality of the diagram;
- We then decided to start from PartNet dataset [YLZ+], which provides a set of 3D segmented objects, and allows us to use semantically coherent models. However, PartNet requires extrapolating the connections and the explosion directions of each part, because it does not provide this information.

1.1.2 Scene graphs

Behind an exploded diagram there exists a graph of connections between its constituent pieces. The first step in the reconstruction of the object is to learn the relationships between its components. Many works focus on finding the spatial and logical relationships between subjects in an image, mostly working on photographs, and this leads to:

- The need to include some prior information about the world to include "commonsense" [ZWYC];
- Increasing the importance of the semantic (labels) of the entities in consideration

This is justified by the nature of their images, which need to rely mostly on those to compensate for the lack of depth information. The importance and efficacy of the objects' labels are shown in the benchmarks by [YRD], as the language-only model they use is not too far from the accuracy of more complex ones. Many different solutions for the scene graph reconstruction problem have been proposed, as shown in [CRX+], however, we will try to find a solution that best fits our diagrams, which ideally will need to rely more on the spatial information rather than the classification of its components.

1.2 Exploded view diagrams

An exploded-view diagram is a single 2D representation of an object that allows the understanding of its composition in terms of its components and their connections. The goal is to provide an understanding of the underlying structure of the subject by moving apart the components and

thus providing a more clear view of its design.

The relative positions of the pieces must be kept when they are moved. This may be achieved in different ways, but the key is the readability for a human, who must be capable of virtually visualizing the object in 3D and rebuilding it. An example is provided in Figure 1.1.

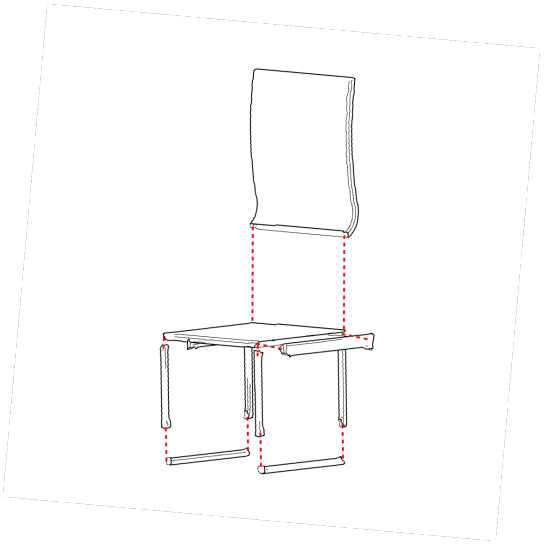


Figure 1.1: *An example of an exploded diagram of a chair showing the connections of the pieces with dotted lines. Even if the components are distanced we can infer how to reconstruct the object*

1.2.1 Explosion conventions

The key of a good diagram is to respect the rules meant to keep it simple and clear. From [LACS]:

- The arrangement of parts helps the viewer understand the blocking constraints (which pieces can not be moved before others) and the relative positions of parts;
- The offsets between parts are chosen such that all the parts of interest are visible;
- Exploded views often minimize the distance of the final position with respect to the parts' original location to make it easier for the viewer to mentally reconstruct the model;
- In most exploded views, parts are exploded only along the canonical axes: restricting the number of explosion directions makes it easier for the viewer to interpret how each part in the exploded view has moved from its original position;
- Parts can be grouped into sub-assemblies. To emphasize how the parts are grouped illustrators often separate higher-level sub-assemblies from each other before exploding them independently.

Other rules can be stated about other details such as cutaways and labels, which however are out of the scope of interest and are variable depending on the application field.

1.2.2 Scene graphs

In a more abstract and implementation-oriented way, an exploded diagram can be represented via a graph [DC, LAS]. In many cases, the graph can be seen as a tree, by not considering the single objects but grouping them in sub-assemblies (see Figure 1.2). Depending on the object, we can observe that the graph can have very different shapes, which can for example represent a stack of components (e.g., the stages of a rocket are one on top of the other) or enclosing shells around the origin (e.g., a matryoshka).

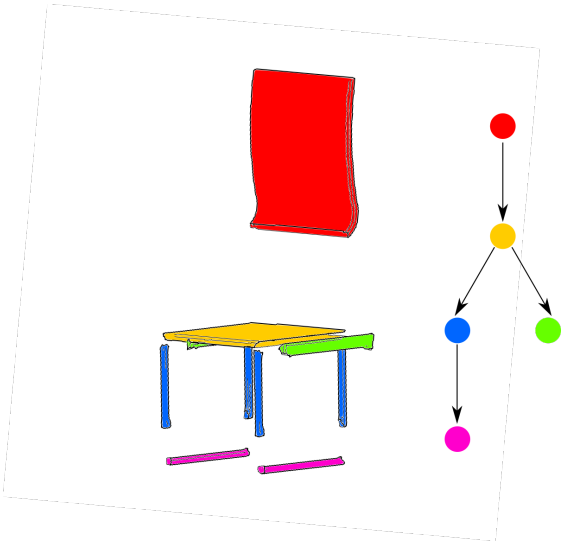


Figure 1.2: *On the left, the colours indicate a possible logical grouping of the pieces. On the right, each group is represented as a node with the corresponding colour and we can see that the connections create a tree.*

A generic scene graph G is characterized mainly by:

- A set S of elementary components that can be found on it
- A set of nodes N
- An association $A_{N,S}$ between one element of S and one node:

  $$A_{N,S} \subseteq N \times S$$

  $$\forall a_{n,s} \in A_{N,S}, \;\; \exists n \in N, s \in S \;\; s.t.\; a_{n,s} = \{n, s\} \in A_{N,S}$$

- A set of node labels L_N
- An association A_{N,L_N} between one node and one label:

  $$A_{N,L_N} \subseteq N \times L_N$$

  $$\forall a_{l,n} \in A_{L_N,S}, \;\; \exists n \in N, l \in L_N \;\; s.t.\; a_{n,l} = \{n, l\}$$

- A set of binary relations (arcs) between nodes:

  $$R_{N,N} \subseteq N \times N$$

  $$\forall r_{n,m} \in R_{N,N}, \;\; \exists n, m \in N \wedge n \neq m \;\; s.t.\; r_{n,m} = \{n, m\}$$

- A set of arc labels L_R
- An association $A_{R_{N,N},L_R}$ between an arc and a label:

$$A_{R_{N,N},L_R} \subseteq R_{N,N} \times L_R$$
$$\forall r_{r,l} \in A_{R_{N,N},L_R},\ \exists n \in R_{N,N}, l \in L_R \quad s.t.\ a_{r,l} = \{r, l\}$$

While the object detection phase (producing labels for the image as bounding boxes or segmentation masks) is the first fundamental step to understanding the scene, it is not enough to represent the object, as it is missing the spatial and logical relationships between the parts. To represent the hierarchy of the pieces, we can use a scene graph in the same way that many other works did [HRC$^+$, XZCFF, ZZJ$^+$], representing each object component's features with a node and the pairwise relationships between components with edges and their associated labels.

Adaptation of scene graphs to explosion graphs

With respect to the existing works, we need to introduce some small modifications that are needed to understand where the components are exploded from.

A label L_r on the arc $R_{n,m}$, represented by the association $A_{R_{n,m},L_r}$, will represent the displacement of node m with respect to node n. For example, in a bottle the cap is screwed in the liquid-containing body, therefore we could say $m = cap$, $n = body$ and to show this relationship we could lift the cap $R_{n,m} = upwards\ of\ 5\ centimeters$. We must notice that this relationship is not the only one that we can see, as we could reverse it ($m = body$, $n = cap$, $R_{n,m} = downwards\ of\ 5\ centimeters$), however, the former arc-label assignment is more natural to see, as we identify the body as the main component of the bottle. For this reason, we can simplify our graph by making it a directed graph and keep at most one arc connecting two nodes.

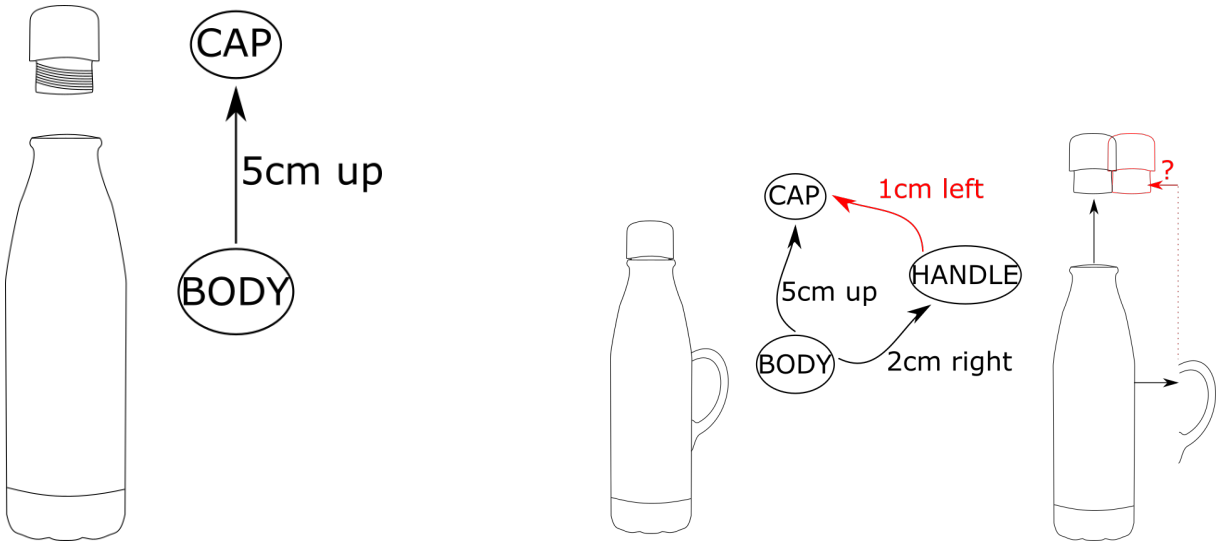
Always considering this type of labelling, we must notice that we can not have cycles in the graph: if we had those, we could get stuck in a loop in which the parts keep distancing from the others, or get physically impossible situations.

In conclusion, we are not going to use a generic graph, but a directed acyclic one. Because multiple interpretations of an object are possible (e.g., cap screwed to bottle or vice versa), we can have multiple valid graphs. Nonetheless, some are more likely than others, so we are going to look for the most meaningful one.

Groups-focused graphs

Another possible way to visualize an object is to highlight its possible component groupings and create a hierarchy of those. This is the interpretation of the objects that is provided by the PartNet dataset (that we will discuss more in detail later), and an example is shown in Figure 1.4. We can define a tree with two types of nodes: the leaves of the tree will be associated with physical components, while the internal ones are abstract and represent groups of components.

This type of structure is useful to hold the information about the semantic meaning of the object's component (each vertical or horizontal bar is a part of the legs), but is in general not



(a) An example of a good graph to represent a bottle. There is only one connection for each node

(b) A bad example of graph. In this case, it is geometrically impossible to represent this object respecting all the constraints on the labels.

Figure 1.3: *Examples from Section 1.2.2.*

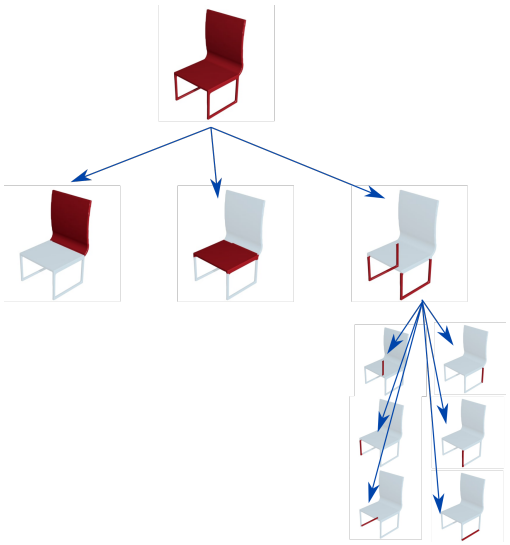


Figure 1.4: *The visualization of a tree based on the grouping of its components. The internal nodes represent a group of objects rather than a component, which can be found only on the leaves.*

ideal to store the physical connections; this can be explained by Figure 1.4, where the deepest leaves do not explain that the horizontal components are attached to the vertical ones and it is not explicit that the legs are attached to the seat, but not to the back.

1.3 Deep neural networks fundamentals

In the chapter about the applications of this work, we are going to talk about how we can solve some common problems relative to the dataset we are going to build. To do this, we will talk about solutions which use neural networks and, for this reason, we are now going to explain their fundamentals.

1.3.1 Supervised learning in a nutshell

In a supervised learning problem, we are given a series of labelled inputs (which means pairs (x, y) of an input object and its corresponding desired output) and the task to solve is to infer the function $f(\cdot)$ (model) that is able to map the inputs $x \in \mathbb{X}$ to their corresponding output $y \in \mathbb{Y}$.

We are now going to describe the general technique in a simplified way, to then deepen the explanations of the topics of our interest.

Training set

Learning the function requires the collection of a training set, which is an ensemble of known (x, y) pairs, where all the $x \in X \subseteq \mathbb{X}$ and $y \in Y \subseteq \mathbb{Y}$ belong to a homogeneous domain respectively (e.g., $\mathbb{X}$ represents images and $\mathbb{Y}$ a boolean value). How our training set is built is going to be explained in the dataset chapter.

The training set is going to guide us towards the model we seek and, for this reason, it should be rich enough to represent a significant portion of all data in $\mathbb{X}$, because our goal is not only to be able to reproduce the data in X (of which we have been explicitly told the solution), but especially predict the output for $x \in \mathbb{X} - X$. For example, if we wish to learn a periodic function, it won't be enough to provide samples from only one period to understand that the function is indeed periodic.

Our model may be more or less complex, and with the increase of complexity we will also need to increase the number of samples required to learn it. A trivial example is when we learn polynomial functions: if we know that our $\mathbb{Y}$ is the set of linear functions, it will be enough to know two samples to learn the parameters of the model; on the contrary, if $\mathbb{Y}$ is a polynomial of 5^{th} grade, having 3 samples is not enough.

Model structure

Every supervised task is going to have a different resolution strategy. There exists no unique technique to solve the problem, so we need to use the knowledge we have about it to choose the learning algorithm and can not feed our data to a magic black box which works every time. In general, the model will be a function characterized by a fixed structure and some parameters to be tuned according to the data. In machine learning there are many known resolution techniques, but we will focus on deep neural networks because they fit best our needs.

Learning

Once we have a training set and a model, our goal is to optimize the parameters such that we obtain a function that is able to predict the best the desired output from a given input in $\mathbb{X}$. When our model is a neural network, tuning such parameters proceeds in an iterative way:

1. An input is fed to the model
2. The result is calculated and compared to the expected output via a *loss* function, that produces an estimate of the quality of the response
3. A feedback system evaluates the loss and adjusts the parameters of the model, according to a learning rule

Overfitting

One of the main problems that can occur when learning (also said training in neural networks) is that our model tends to perform very well on the training samples in X, but generalizes poorly to the remaining $\mathbb{X} - X$. This is called overfitting, and it can be fought by introducing another independent data set, called validation set. When using the validation set, we can calculate the loss on its samples to have a better estimate of the behaviour of the model on new inputs and adjust the parameters taking this into account rather than checking only the training loss.

1.3.2 Artificial neural networks

Neurons and layers

An artificial neural network (NN or ANN) is a model formed by interconnected neurons. A neuron is an entity which processes some input signals (passed by other neurons) with a (non-linear) *activation function* and produces an output which is passed to other neurons. The different inputs x_i are combined with a weighted sum $\sum_i a_i x_i$, where the weights a_i need to be tuned according to the problem, and then fed to the activation function $f(\cdot)$ of a neuron, which itself depends on a bias parameter b.

Typically, the network is structured in an ordered manner by collecting the neurons in layers. A simple network is the feed-forward NN, which is composed of an input layer l_0, which gets the inputs directly from the data, succeeded by a set of internal layers $l_i, i \in 1..n-1$ and an output layer l_n. In this architecture, each layer l_j gets the inputs from a set of neurons belonging to l_{j-1}, and thus we can see this as a direct acyclic graph of neurons. In general, however, NNs can be more complex and have multiple input and output layers and non-contiguous layers may communicate with any other one.

The operation of one layer can be represented in the compact matrix form $f(WX + b)$, where

- $f(\cdot)$ is the activation function of the layer l_i
- X is the vector containing all the inputs

- W is a matrix containing the $n_{l_i} \times n_{l_{i-1}}$ weights, where n_{l_j} is the number of neurons on the j-th layer
- b is the vector of biases of the neurons of layer l_i

Layers can be of different types, which are defined by the activation function of its neurons and how many connections the neuron has with others.

Fully connected layers

A fully connected layer is of the most general type, as it exploits potentially all the inputs from all the connected neurons. While this allows collecting most of the information, it comes at a cost because, remembering that the number of weights relative to a layer is $n_{l_i} \times n_{l_{i-1}} + b$, the number of weights would increase too much to be handled, as the computation time would become unbearable. For this reason, other layers are often employed to reduce the size of its input vector to a value that can be handled.

2D convolution layer

A classic input example of a network is an image, which (considering it as a 1 channel image for simplicity) we can see as a bidimensional $N \times M$ matrix of pixel intensities. Handling the image directly with a fully connected layer is not a good idea for two reasons: the number of parameters and the loss of locality. Indeed, one pixel alone does not carry enough information but necessitates the context of the pixels on its surroundings. To combine the information of a pixel with its neighbourhood we can apply the convolution operation.
2D convolution employs a filter that consists of a $k \times k$ matrix W which is applied to one pixel $p[i, j]$ (the pixel in coordinates (i, j)) and its surroundings and is calculated as:

$$* \sum_{m=1..k} \sum_{n=1..k} W[m, n]p[i - m, j - n] \tag{1.1}$$

Applying the same convolution filter to all the pixel locations does not only mix the local pixel data but also greatly reduces the number of parameters, as the weights relative to the filter are negligible, as its size is typically $k = 3$ or $k = 5$.

Other layers

After using a convolution layer, the dimension of its output remains (approximately) the same, but in many cases we wish to compact its representation and keep only the most useful information. Similarly to the convolution layer, the max-pooling layer uses a filter which in this case is used to calculate the maximum of the region beneath it and is run across the input matrix.

More practical tools rather than logical ones, the flatten layer reduces all the dimensions of a tensor into one, thus unrolling it into a vector and the concatenation layer concatenates two tensors along one axis. The latter can be useful when our networks have merging points between two branches.

1.3.3 Back-propagation

To train our network (tune its weights/parameters) the backpropagation technique will be used to provide feedback of the quality of the predictions and update the weights to improve future predictions. It consists of calculating the gradient of the loss function with respect to the weights one layer at a time, with the goal of minimizing the loss function of the future predictions. The backpropagation algorithm is repeated multiple times to adjust the NN's weights progressively, and the calculation of the gradient is not computed analytically but rather with approximation techniques.

Dataset Generation

2.1 Data generation

While it is possible to find plenty of diagrams, it is hard to find these together with the information about their explicit underlying structure and their corresponding 3D models. However, because we are going to need to extract (most of) this information using supervised learning techniques, these are fundamental data to have. One solution could be to take the existing diagrams and manually annotate them, but this would require a lot of time and effort. The most viable solution is to produce a synthetic dataset starting from the knowledge of a set of objects' 3D structures.

2.1.1 Limitations induced by a synthetic dataset

Real diagrams can vary a lot in their details (e.g., dashed lines, labels) which can be very useful (if not fundamental) for understanding. However, we chose not going to represent these, as they do not respect strict rules. This does not mean that the work based on this dataset will not be applicable to more complex existing images, indeed it is the most general as possible to cover the broad spectrum of possibilities.
It is likely that the algorithms built for such artificially generated images won't immediately work if given in input real images with annotations and, more in general, noise. To adapt the work to real-world images (in which annotations are useful but not fundamental) we can perform a clean-up step to remove information that we are not used to working with. Nevertheless, for very complex diagrams the annotations are fundamental and ignoring them leads to wrong interpretations. Learning these is not feasible without including the required annotations to the dataset images, and therefore they require a more specific dataset. Such complex diagrams are however out of the scope of this work, as they are application-specific.

2.1.2 Restrictive hypothesis

For simplicity, we are also going to add some restrictions to the generated images. We will see that for our applications these are not going to be too strict, but will help a lot with the decision of the explosion directions and reduce significantly the computation time.

We are going to assume that the explosions occur always along one of three orthogonal canonical axes, so a component can never be exploded with respect to the piece it is attached to in a direction that is not canonical. To choose such axes, we are going to select those which are reciprocally orthogonal and parallel to most of the straight directions. Because in our case we will notice that the axes that respect these conditions are the x, y, z axes in the model space, we don't need to do this computation. Nonetheless, we are going to briefly present a technique that could be used when we are not given hints about the axes.

Computation of the axes

Given a 3D model, we need to fix it into a Cartesian coordinate system, which we will use as our explosion directions. Not every 3 orthogonal vectors constitute a good base for the explosion axes, as we wish to find a reference system in which the pieces can be moved in meaningful directions. Statistically, in mechanical objects we can identify them by looking at the direction of the edges of their components, which are usually aligned parallel to the normals of the planes of connection between pieces.

The first step is to detect the straight edges of the object; in case we don't have enough, we may try to detect symmetry axes to compensate. We can then detect 2 or 3 principal orthogonal directions of the lines by looking at the histogram of the distribution of such directions' angles and taking the 2 or 3 highest peaks distant 90 degrees. Ideally, we get three directions, though two suffice, as the third can be found as the cross product of the first two. An idea of part of this procedure is shown in Figure 2.1, where the same concept is applied to a 2D image.

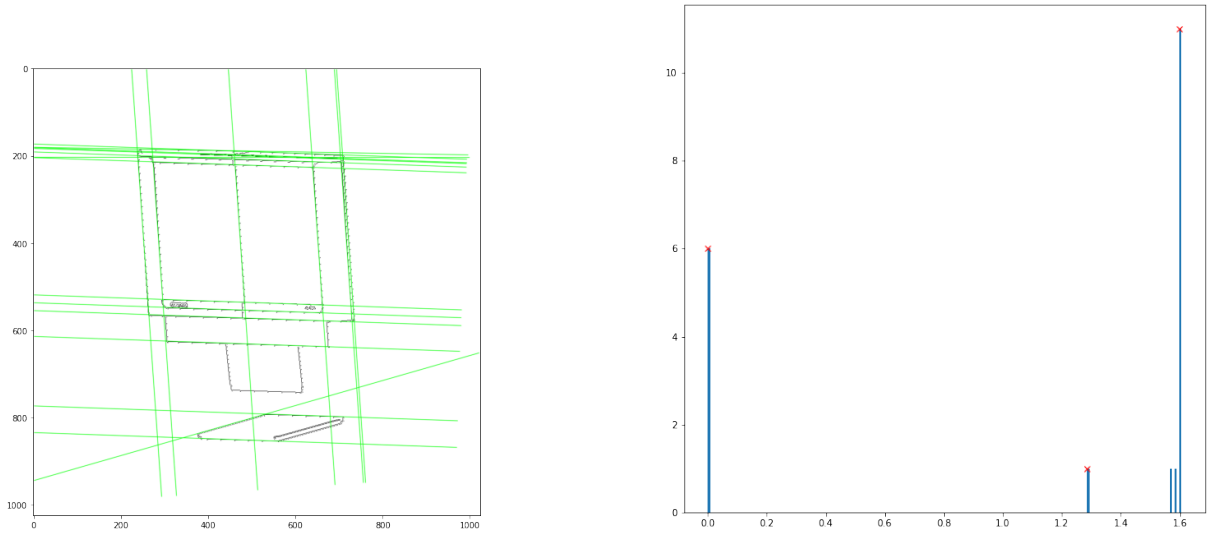
2.1.3 Scene generation from scratch

To create a meaningful image, we need to create or extract the relationships between the constituents of an object, in other words, create its scene graph. In the most trivial approach, we are going to build a randomized tree and place random components on it.

Degrees of randomization

The degrees of randomization of the tree and the objects chosen can influence a lot the learning process, and we need to find the most desirable fidelity-generalization trade-off. While leaving complete freedom to the graph structure would ideally help to approach the most variety of scene graphs, using structures that resemble real-life objects can be more helpful in most cases and allow to learn faster the object's structure.

The main issue, however, is not in the graph creation (it is fairly easy to identify the most



(a) Straight lines detected on a pc exploded diagram

(b) Distribution of the angles of the lines

Figure 2.1: *An example of axes directions detected with the Hough transform on a 2D image. On the histogram, the red crosses identify the three main peaks.*

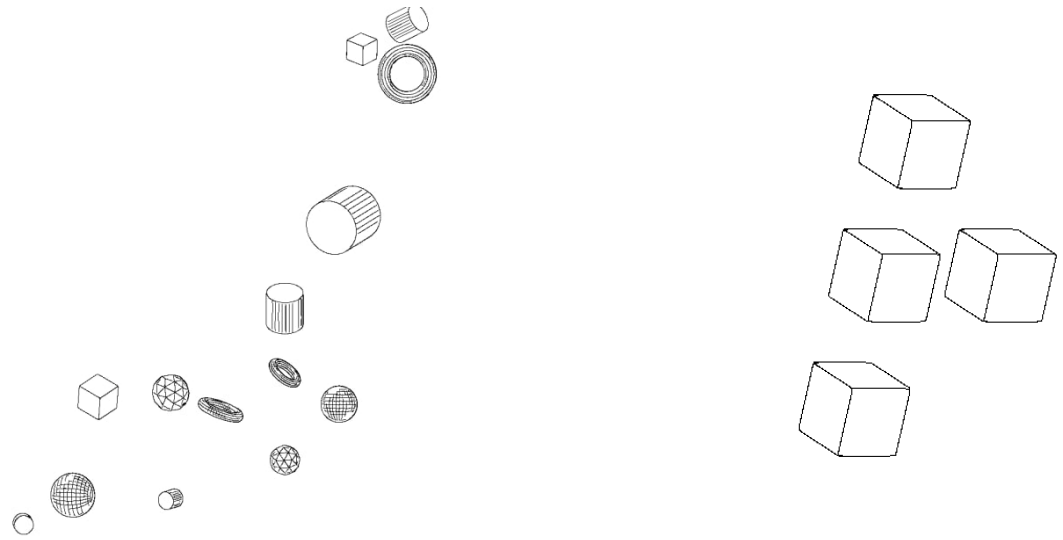
common shapes it can assume, such as stacks and stars), but rather in the choice and placement of the components on the graph. Indeed, selecting compatible objects (such as screws and nuts, or gears and wheels) from a set with a uniform probability distribution allows the greatest generalization, but setting constraints on couples of objects that need to be put close improves the semantic meaning. While datasets of single mechanical components are available, these are hard to exploit because they would require the integration of prior knowledge by hand (for example, defining which components need to interlock and in which direction, or which have no meaningful relation).

We can see in Figure 2.2 two examples that show the trade-off between simplicity and generalization, and why this approach is not good.

Conclusions

While ideally a dataset of images built in this way would be the most general possible, the semantic loss is too high (unless a relevant human contribution is introduced), and would lead to the necessity of introducing important constraints on the process of learning the underlying structure.

In 2.2 we are therefore going to build our images by exploiting a more direct source of prior knowledge, reducing the randomization factor but increasing the semantic meaning.



(a) Placing random objects along an L-shaped tree and rotating the components allows a lot of generalization, but there is no meaning and therefore even the tree shape is hard to understand

(b) Reducing the number of objects involved and their variability (transformations) allows to see the underlying structure clearly but does not resemble a plausible object

Figure 2.2: *Two examples of why placing random objects fails in generating good diagrams*

2.2 PartNet dataset explosion

The most common scene graph comprehension problems are based on natural images (see, for example, [CRX+, ZZJ+]), so their dataset is based on annotated photographs. More close to our work is [LAS], which uses pictures of mechanical objects and explodes them in 2D. This is closer to our needs, because of the subjects of the pictures, however, we wish to work on 3D objects and not on coloured images.
PartNet [YLZ+] dataset provides data in a similar fashion with respect to our needs, offering the segmentation of tridimensional models' parts for several common artificial objects and creating a logical tree of connections. In the next section, we are going to present it and adapt it to our needs.

2.2.1 PartNet

PartNet dataset represents a collection of 3D models designed for the segmentation of everyday objects. It is designed to be used with different levels of granularity thanks to its underlying tree structure that relates the object's components.

For each object, we are given the set of meshes (models) that compose it and their relationships, grouping semantically similar parts in a hierarchical fashion and creating a tree of connections between groups of objects (Figure 1.4). The tree is built such that:

1. A leaf must correspond to an individual component. It is labelled according to the class

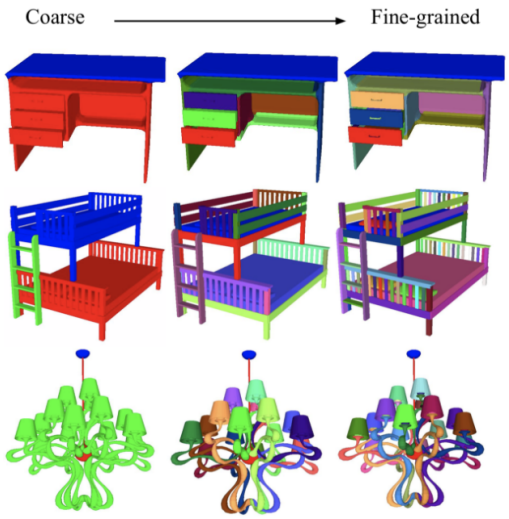


Figure 2.3: *An example of the different classifications of the parts of some objects in PartNet, from coarser to more refined. Image from [YLZ+]*

of objects it represents;

2. Grouping siblings provides a less fine segmentation for those, by treating them as on the same class. As a consequence,
3. an internal node represents the collection of all its children, plus possibly new individual components.

This structure allows to segment with different levels of granularity, going from a single class with all the parts in the root, to each individual component on the leaves.

2.2.2 Scene graph generation techniques and observations

As discussed previously, the segmentation of the model is necessary but not sufficient for our needs. The structure that we are given provides hints for how to build the scene graph but is not the scene graph itself, indeed it is represented by a structure to the one presented in Section 1.2.2. The main difference between these two forms of representation lies in the internal nodes: PartNet hierarchy tree can tell us that the legs and the seat of a chair build its bottom part, but cannot explicitly say that the legs are directly connected to the seat.

Therefore, we need to build the relationships $r_{n,m} \in R_{N,N}$ between pairs of parts $s \in S$ which are physically and semantically directly connected. This does not mean to represent all the possible contact points between objects, but to establish a more meaningful relationship of parenthood between a sustaining and a sustained node, which is more external to the object and requires its parent to be connected to the main body. An example of this hierarchy can be seen in a bike, where a wheel is connected to the main body through the fork, so in the graph we will encounter the main body first (closest to the root), the fork second and the wheel at last.

No-prior scene graph reconstruction method

One approach that could be used is similar to [LACS] or [LAS], who approached the same problem of exploding objects by creating a hierarchical explosion graph (Figure 2.4). This is a progressive unlocking approach based on iteratively removing the pieces that are free to be moved towards the outside (not blocked by others) and creating a decomposition that starts from the outer shell and reaches the core. The reciprocal unlocking dependencies naturally create a graph based on the relations between close objects and suggests also how to unlock and make the parts visible.

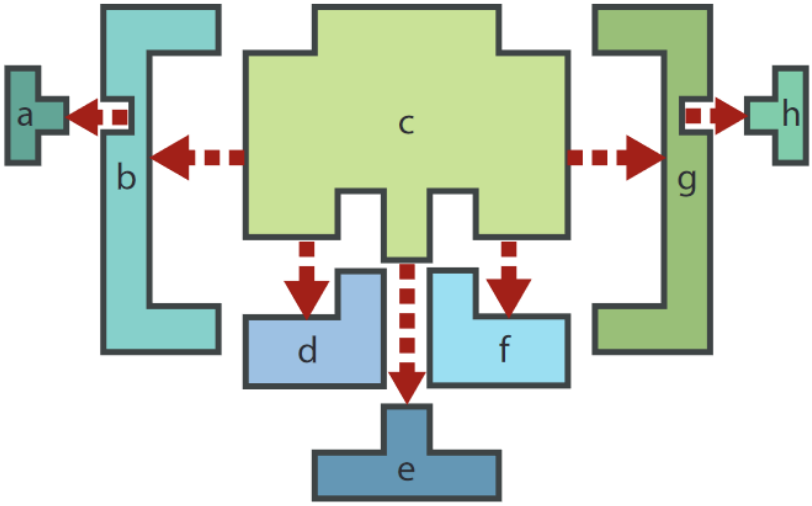


Figure 2.4: *The interlocking dependencies discussed by [LACS]. These create naturally an explosion graph*

The main advantage of this technique is that it requires no knowledge of the meaning of the object because it is only based on the position and the shape of its components, which makes it more versatile. Nonetheless, this can also be a disadvantage, because it leads to not considering any prior information about the type of object represented.

Another big issue with this method is that it is not enough to explode many objects in PartNet. This method is extremely useful in the case of objects interlocking with others (e.g., a screw and a nut), however, most objects in PartNet have parts that do not intersect. The non-interlock is a property generally shared by components that lay on each other (two flat faces touching), but in PartNet it is caused also by the fact that some models have originally been created as a unique piece (a mug is the fusion of the cup and the handle, which are not separate entities) and have been later cut in pieces with a plane, leaving two hollow meshes (the mug and the handle) that do not intersect (Figure 2.5). The lack of intersections between objects is problematic not only because the connections become less obvious, but also because we don't have a more or less obvious explosion direction imposed by the direction in which we would extract an object which is stuck into another.

Figure 2.5: *A model from PartNet. It is visible how in this case the pieces do not intersect as they would in real life, but have been cut from a single mesh*

Scene graph reconstruction by using the PartNet graph knowledge

Having to deal with this dataset in particular, we can rely on the knowledge of its underlying graph for classification. This does not provide necessarily obvious physical dependencies but integrates semantic meaning to the models. Combining it with the knowledge of the possible classes of objects in the dataset, we can express quite solid knowledge about the relationships between the components. For instance, in the PartNet tree, we will find all the legs of a chair in the same leaf with the same label and the combination of the seat and the legs will be found as its ancestor, as they compose the bottom part of the chair. While this kind of information is not bound to necessarily give information about the connections between objects (for example, one could label the break of a bike as an item attached to the handle but without telling us that it is connected to the wheel), it gives us an important hint on the semantic meaning of the components, telling us that even if the wheels of a car are far apart, they are all identical.

The most straightforward conclusions we can get from the PartNet tree come from the leaves. Parts in the same node that share the same label express that not only that they represent like objects, but can also hint at symmetries that we would like to keep during the explosion phase (for example, exploding all the legs of a chair along the vertical axis instead of using a different one for each). Less obviously, and with less certainty, we can observe that objects with the same class are often close by in space, hence we can consider the depth of two nodes and the depth of their lowest common ancestor to get an intuition on how far they may be.

2.2.3 Hybrid solution

To obtain the best results, we need to combine the strategies shown before. We are going to list heuristics that help us reconstruct the object by keeping semantic and positional information, and applying them while prioritizing the most informative. The construction of the tree requires observations directly from the models but also indirectly from the PartNet tree, and requires

comparisons between all the 3D components/PartNet nodes. From the pairwise relationships between objects in direct connection, we can iteratively build the tree.

Proximity detection and validation

Having two objects touching is for sure very meaningful, but it is not enough to state that they are indeed connected. For instance, the break touches the wheel of the bike, but it is not screwed into it. Another example is shown in Figure 2.6.

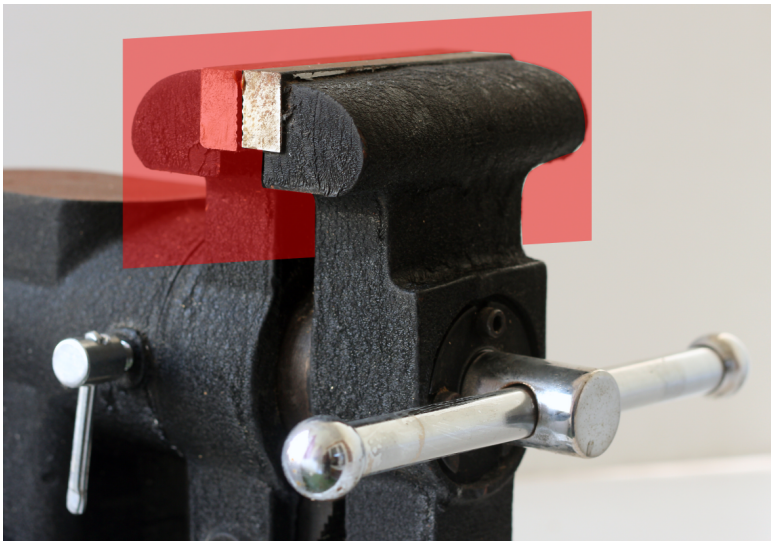


Figure 2.6: *The two parts of the vise separated by the plane do touch each other, but are not connected*

First of all, we are going to build a bidirectional forest graph by detecting which objects are in contact. In this graph, each object's component is represented by a node and the arcs connect parts that touch each other. Because we can't exclude that some pieces do not touch (it could happen for example in presence of repulsing magnets, as an example), we could find ourselves with an unconnected graph. To solve this, we can identify the N isolated trees T_i, $i \in \{1..N\}$ and select the main tree T_k such that it has the highest number of nodes. We are then going to connect each tree T_j, $j \neq k$ with T_k by connecting the closest nodes $n_k \in T_k$ to $n_j \in T_j$, obtaining a single tree. Notice that the choice of connecting everything to the biggest tree is not necessarily the best one, but simple enough and it fits well on the dataset we are operating on. In general, we may want to consider each tree as a group and link each group to the other closest one.

At this point, we need to integrate also the semantic information from the PartNet tree keeping only the most meaningful connections. To do this, we need to first decorate the graph's arcs with weights that correspond to the semantic distance $d_{i,j}$ of the nodes n_i, n_j; each node has a corresponding one in the PartNet tree, so we can define the semantic distance between two nodes as their distance in the PartNet tree (see Figure 2.7). Now, we can find the minimum spanning tree to trim away the redundant connections and keep only the direct parent-child relationships that should be associated with real attachment points.

To help us keep the information about the parts with the same class in the same PartNet node,

in this stage we can treat them as a single object. We are then keeping multiple parts on a node, and assign it a cardinality which corresponds to the number of objects contained. For the sake of simplicity, we are going to refer to multiple meshes on a node as a single component.

Up to now we have a bidirectional tree and haven't identified a root. The choice of selecting a node as root can be based on:

- The size of the components
- The distance from the most external components
- The cardinality: a big single object is more likely to be the central piece with respect to a node with many small parts

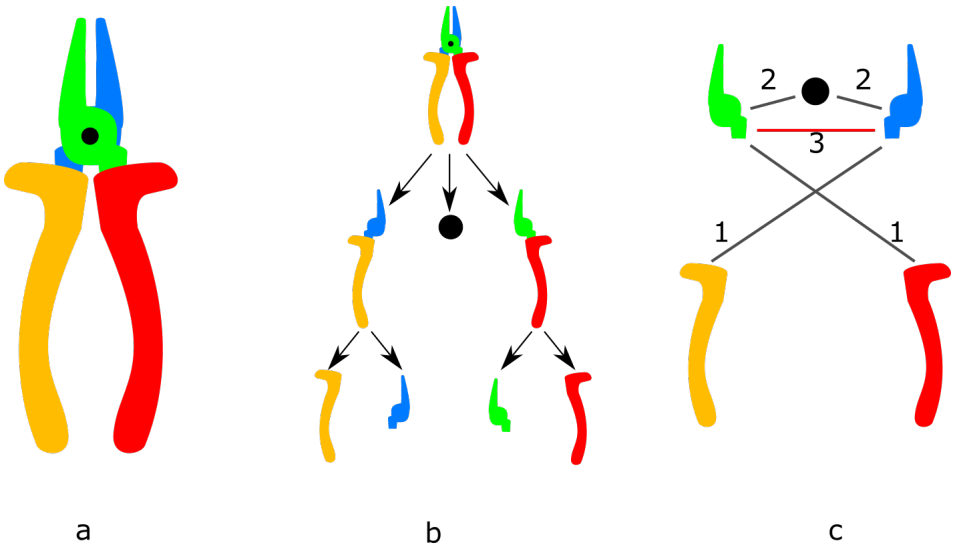


Figure 2.7: *The illustration of the logic behind the construction of the graph of connections.*
- a shows some pliers and their pieces
- b represents a semantically meaningful graph which highlights the symmetries
- c shows the connections and the distances of the touching components on the semantic tree. The connection between the metallic pliers (green and blue) will be removed, as it has a greater distance, thus maintaining the connection of the pliers with the screw but not between themselves directly

2.2.4 Graph explosion

Now that we know to whom each piece is attached, we need to understand which is the smartest direction to move it away from the parent. We will first discuss how to extract one component from the other in case of interlocks, and then define a new procedure for the other cases.

Interlocks detection

When two components are one inside the other, we get the clearest connections. We can call the enclosing object parent and the enclosed one its children. To detect this situation and understand

how to free the lock, we are going to check the collisions against the parent object's mesh and its convex hull. Because of the nature of PartNet, we are going to analyse a simplified case with respect to Li et al., excluding the case in which freeing one object implies breaking another.

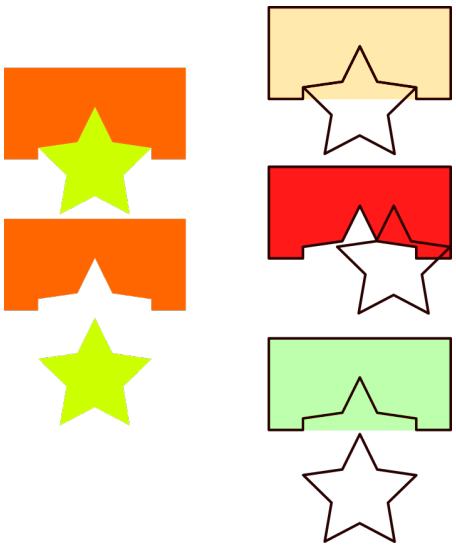


Figure 2.8: *On the left, the initial situation and the parts after unlocking them. On the top-right, we see that the star is blocked because it intersects the convex hull of the parent. On the centre-right, the star can't move to the right, because it would collide with the other component. On the bottom-right, the star is unlocked because it moved outside the convex hull without colliding.*

When the child collides with the convex hull of the parent, we know that in at least one direction it is blocked by it. Furthermore, we know that the meshes of the two can not intersect, because this would lead to the fusion of the two, which we assume to be physically impossible (we deal with solid objects). We can therefore try to move the children apart from the parent with the goal of reaching a position in which the child does not collide anymore with the parent's convex hull and which requires the child to never intersect the parent to get from its original position to the unlocked one. This is shown in Figure 2.8

Versus of explosion

Independently on the direction of the explosion, according to the rules of explosion, we must never let the children cross their parents during the explosion (we are not considering only the endpoint of the displacement, but also all the intermediate points between the start and end positions). We can make sure of this by checking that the pieces position is flipped during the explosion.
When the two objects' bounding boxes do not intersect, it suffices to check the relative positioning of their centroids (otherwise, we can treat this similarly to the interlocked case). We are going to take the vector d from the centroid of the parent to the centroid of the child and project it into the explosion axes x, y, z, obtaining d_x, d_y, d_z and their corresponding versors v_x, v_y, v_z. If the final position of the pieces is such that the new centroid-to-centroid vector d' has the same projections' versors, we have the guarantee that the rule is respected.

Direction of explosion

While for the versus we had constraints dictated by the rules of explosion, the direction is trickier, as it must be understood case by case. This is a fundamental step, as exploding in the wrong direction may confuse the observer. We are going to examine some common cases to extrapolate the rules to get the best axes.

One case consists of one piece lying on top of another so that they share a contact surface (see Figure 2.9.b). From the observation of the dataset, we can see that it is more likely that the child piece needs to be lifted orthogonally to the shared face (a pillow on the seat of a chair, the top of a table on the legs...). However, we must also take into consideration that the normal n to the surface of contact may be not aligned to the explosion axes; in this case, we are going to take the principal component of the projection of the normal on the axes.
Another possible heuristic does not rely on the contact of the parts, but checks only the relative position of the objects (see Figure 2.9.c). In this case we are going to rely again on the centroid-to-centroid vector d and its projections d_x, d_y, d_z. We are going to define a threshold $0 < \tau \leq 1$ and if $\exists d_j \; s.t. \; \frac{d_j}{d_k+d_l} > \tau$ where j, k, l represent one axis each, then we define the two components as aligned along the axis j, which is going to be the axis of explosion.

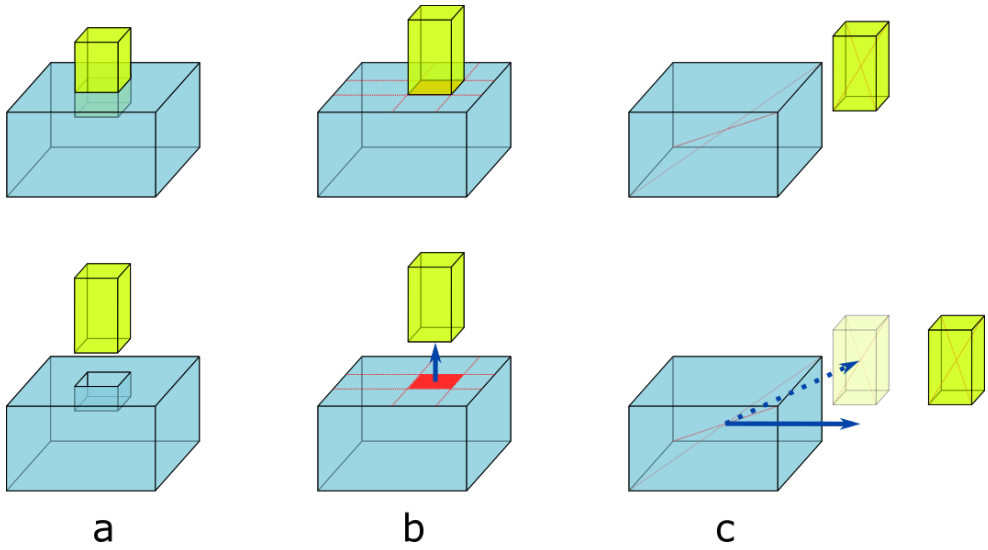


Figure 2.9: *Some possible cases for the explosion of the small parallelepiped (child) with respect to the big one (parent). The top row represents the object before the explosion, and the bottom row the object above after the explosion.*
- a represents the case in which the child is interlocked with the parent
- b represents the case in which parent and child share a face
- c represents the most generic case, in which we consider only the position of the centroids

Modulus of explosion

Regarding the distance of the pieces, there is no strict rule, and the only constraint we need to take into account is that we must avoid collisions and not cross the paths of other blocks. An example is on Figure 2.10. In this work, we have chosen to use a distance that is proportional to the size of the objects involved.

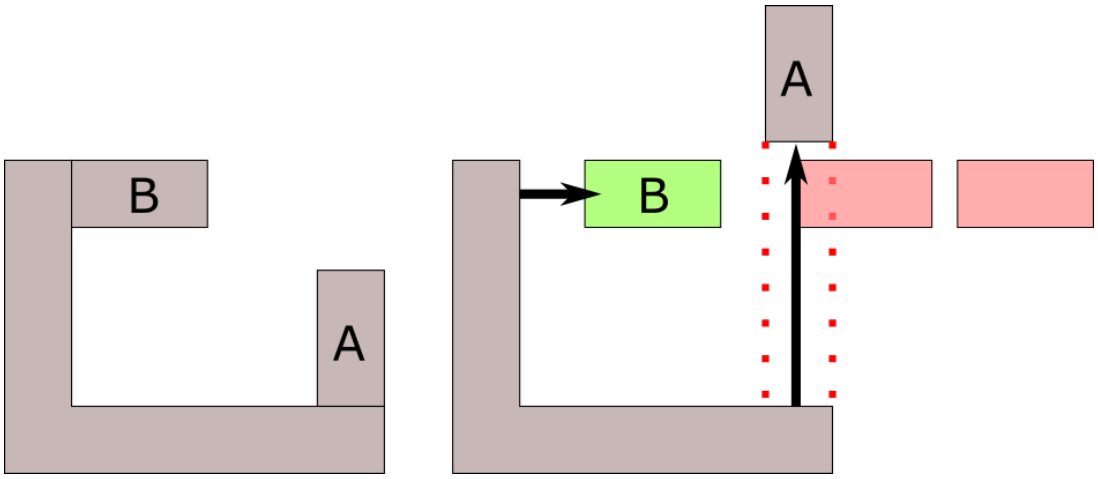


Figure 2.10: *On the left, the situation before the explosion, on the right A explodes first, B second. In green is represented a valid position for B, in red alternative not-valid positions for it*

2.2.5 Rendering

Once we have created and exploded our object in 3D, we must project it on an image.

Camera settings

While our eyes are most used to viewing in perspective, in the context of mechanical objects it is more helpful to use an orthographic view, as it allows us to notice more easily the regularities and symmetries of human artefacts and to understand their real reciprocal proportions.

Because the goal of an exploded-view diagram is to show the structure of the object, we need to let most of the components visible, or at least comprehensible thanks to symmetries. One first trivial constraint is then to fit all the objects in the viewing area of the camera, and then maximize the visibility of the components. In general, we would be choosing an area where to focus our attention based on its relevance, and position the camera such that its visibility is maximized. When trying to automate this, we could base our choice on some heuristics (e.g., the most central piece, the biggest) or learn it. However, learning the salient parts requires the dataset to be annotated with this information; nonetheless, we could simplify this labelling process by grouping the objects by categories and noticing, for example, that the seat is more likely to be the focus of our attention rather than one leg, and applying these observations uniformly for each category. Once we have an area to focus on, we can position it in the centre of the image and maximize its visibility. We can observe that, because our objects are aligned to the axes of explosion, it is unlikely that the camera will be aligned to those as well, as angling it allows a better view.

Edges

Up to now, we have been ignoring the fact that our final image should resemble a drawn schema. In the 3D meshes we are given we don't have the information about the edges which will be

visible in the final image and we are also totally missing some contours: for example, theoretically, a sphere has no edges, however, it will appear as a circle. Another factor we need to take into consideration is the absence of shading, which needs to be compensated to hint curve surfaces (Figure 2.11, Figure 2.12).

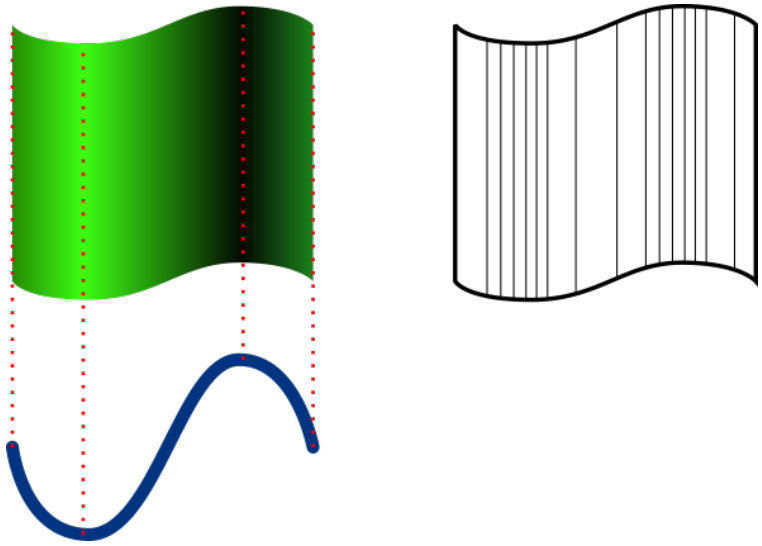


Figure 2.11: *It is necessary to hint at the shades even if they do not represent real edges to distinguish a flat surface from a curved one*

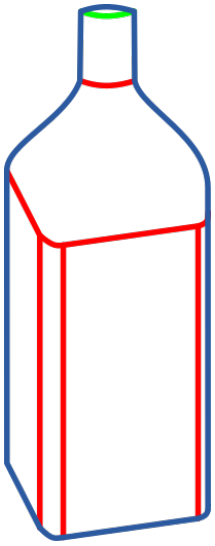


Figure 2.12: *The figure shows three types of lines that need to be represented:*
- Blue (external lines): contours. They may coincide with edges, or be fictitious lines that appear when projecting the object on a plane
- Red (internal lines in correspondence of smoothed edges): curves
- Green (bottleneck rim): real edges

We are then going to output two images: the first one containing the segmentation map of the components and the second one with the information about the face normals to consider respectively the external contours and the edges and curves of the components (Figure 2.13).

(a) The absolute value of the normals of the faces. Each component v_x, v_y, v_z of the vectors is represented respectively by the red, green and blue colour components

(b) A simplified segmentation map for the components of the chair

Figure 2.13: *The two renders with the normals and segmentation information, preliminary step to the final image*

We then merge the information provided by those images to generate the diagram in its final drawing-like shape. To generate the edges and hint at the curvatures, we are going to analyze the colour gradient of the normal map: the idea is to define a threshold on the colour gradient, above which we draw a line; in practice, we are going to apply a Canny edge detector. The contours of the image are straightforward to obtain as they are the contours of each class in the segmentation map, and the final image is obtained as the superposition of the two.

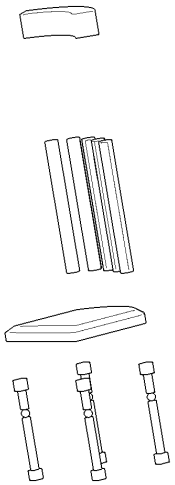


Figure 2.14: *The final exploded-view diagram of the chair in Figure 2.13*

2.3 Implementation

The following section will show the practical implementation of the aforementioned rules and techniques, describing how they are applied to the dataset and explaining the challenges encountered and how to overcome them.

2.3.1 Symmetries and similarities

In this first step, we are going to use the information about the single nodes of the PartNet tree to group components according to their similarities.

Each PartNet tree's node can represent a group of components, single parts or a mix of both. In this step we focus on the spare components and group them according to their class; we will consider each group as a new individual component. There are multiple reasons to do so:

- These items are similar in nature, as they belong to the same class;
- Because of the (often) symmetric nature of the objects treated, it is likely that these parts are symmetric along one axes or plane;
- From a practical point of view, we need to keep together some parts that are semantically separate but represent a single component.

Creating these groups will ease our work later, as it allows us to maintain the symmetries and keep together parts that must stay together.

This procedure eases the future steps and can be done in linear time with respect to the number of components. However, it is not perfect: these heuristics are indirect properties of the tree based on observation and are not necessarily correct in every case. A more accurate strategy would consist in detecting symmetries and similarities directly from the observation of the object, however, this requires increasing the complexity, as symmetries require comparing the meshes of each couple of components. Because the accuracy of the observations on PartNet tree is quite high and because of the complexity of the alternative approach, we chose not to proceed with this more accurate technique.

At the end of this process, we have a set of disconnected nodes that represent one or more entities.

2.3.2 Connection tree

The most obvious hints on how to connect the nodes are given by the points of contact, but, as discussed in Section 2.2.3, these are not enough. As already explained, we are first going to connect the meshes touching each other, and later cut the branches which correspond to points without direct connection.

After the first part of the procedure, we will find ourselves with a (possibly) cyclic graph. While this may represent a generic scene graph, we already observed that this can not be used for our objective, otherwise it would create impossible situations when exploding the object. We said

how in general we can use a DAG (directed acyclic graph), but looking at our objects we can state that we can reduce it further to a tree (in our case the minimum spanning one). We must remember that the nodes we are considering at this point are, in general, ensembles of meshes, so our parenthood relationships represent the point of contact of groups, and not the single components, meaning that at least one mesh of a child is connected to at least one mesh of the parent, as we will see later.

Root

Our tree, as an abstract structure, does not explicit a root, as the graph it is generated from is bidirectional. Nonetheless, we need to define one and this could change the final result because in the object explosion phase we will use the parent-child relation to define the explosion vectors and the explosion type (that we explain in the next section). We can see this in Figure 2.15.

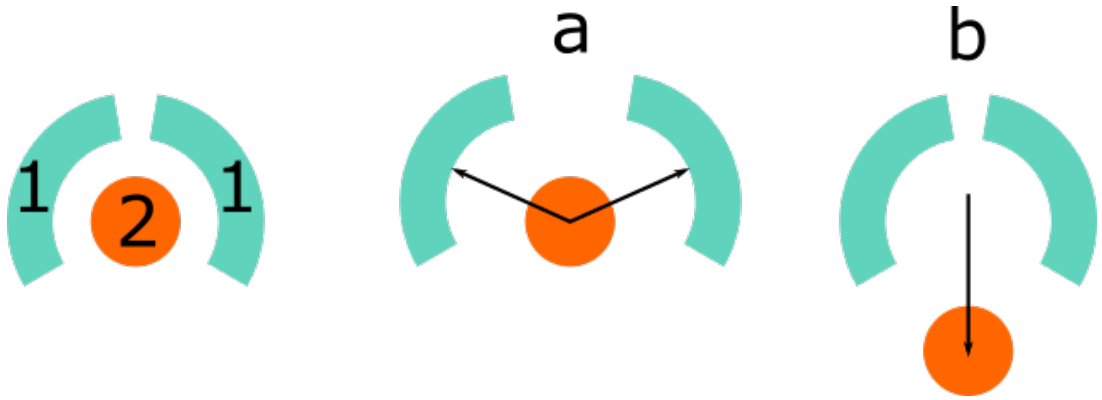


Figure 2.15: *The importance of defining the correct orientation on the tree. On the left, the original object is divided into two nodes (1,2). In case a, the circle is the parent, hence the objects of class 1 are exploded radially. In case b, the objects of class 1 are the parent, the circle is exploded by a translation and 1s don't move*

The root of the graph will be the starting point of the explosion, and it will not move. It may represent the centre of the object (the fulcrum screw on a pair of scissors), the most important part (the bike frame of a bike), or the biggest component.
We can observe that in all the above cases the root is characterized by a single piece, so this is a desirable characteristic in our root node.
It is less likely that a piece which has only one connection with the rest will be the centre rather than one which is connected to many, so we are going to prefer the latter. Extending this reasoning, we can start from the leaves of the tree (which are known, as they have only one connection) and follow them towards the internal ones to find the node with the greatest distance from the leaves, as shown in Figure 2.16.

These heuristics are not necessarily the best ones. In alternative, we could give priority to the biggest pieces, or choose the root that balances the tree.

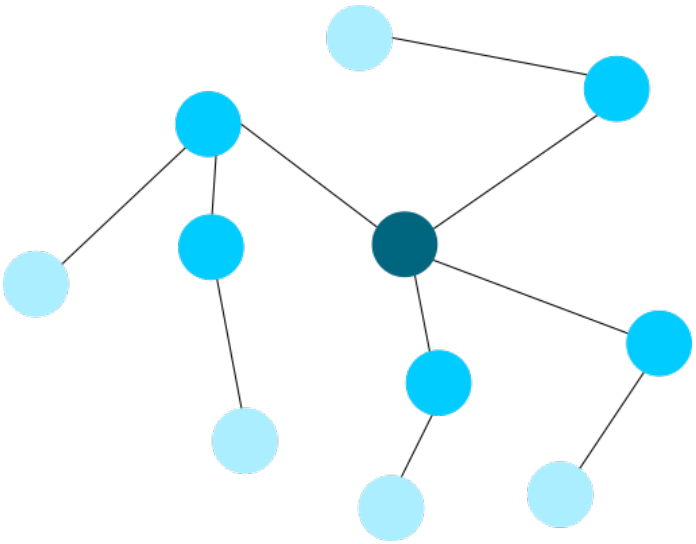


Figure 2.16: *The visualization of the minimum distance of a node from the leaves. The gradient goes from the most external nodes to the root*

2.3.3 Explosion criteria - linear and central

In Section 2.2.4 we have presented different criteria for exploding one node with respect to the parent. However, we were not considering nodes that are composed of multiple parts. Because of this choice, we introduce another type of explosion.

When we are dealing with only a pair parent-child of objects, as we discussed, we are going to use a displacement vector aligned to one axis of explosion, which is decided based on the geometries of the parts. In the scene graph that we are using in this step, however, our nodes represent multiple objects, that we treat as one component that is their union. If we apply the same techniques as before, we can find ourselves in situations like the one represented in Figure 2.17, where some parts are correctly exploded but others in the same node have indirectly been given less priority and are in the wrong final position. This occurs when the children are placed around another component (note that not necessarily there are interlocks between single parts), and it is generally solved by introducing a radial explosion.

We are going to use two types of explosion: linear and central (radial). By linear explosion we mean a translation of all the pieces on one node by the same vector (which is what we discussed up to now), while a central one implies that the parts in a node are exploded differently but relative to a common centre. In the traditional way of representing a radial explosion, the explosion vector has the direction and versus of the vector connecting the two parts that we are considering, however, to stick to the axes of explosion that we have defined we decided to take the principal component of the decomposition of such vector on the axes. This choice has been made because linear and central types of explosion are clearly understandable when used separately, but mixing these techniques may create confusion if we are not inserting lines to indicate the connections.

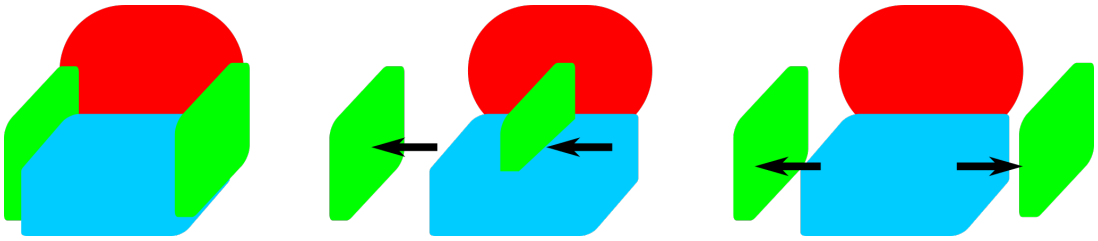


Figure 2.17: *On the left, the model before the explosion. On the centre, the armrests have been exploded with the same explosion vector and while one armrest is in the right position, the other is not and is also colliding with the other components. On the right, each armrest has been exploded around a common centre.*

2.3.4 Separation of groups

Up to now, we have been working with nodes representing groups of objects, but what we wish to obtain is a graph in which each individual part is connected to at least one other. To do this, we are going to start from the tree that we used to learn the explosion directions (let's call it *explosion tree*) and refine the connections (obtaining the *real connection graph*).

Let us consider the parent node p and its child c in the explosion tree; each is composed of $m_i^p, \quad i \in \{1..n\}$ and $m_i^c, \quad i \in \{1..l\}$ parts respectively. To connect them, we are going to proceed in an iterative fashion.

Graph constraints

Working with a tree, by definition, guarantees working with an acyclic graph and hence guarantees that no impossible dependencies like Figure 1.3 occur. However, converting our tree to a generic graph, we need to make sure that this property holds, but checking that the graph is not acyclic is not enough, as it can be seen in Figure 2.18.

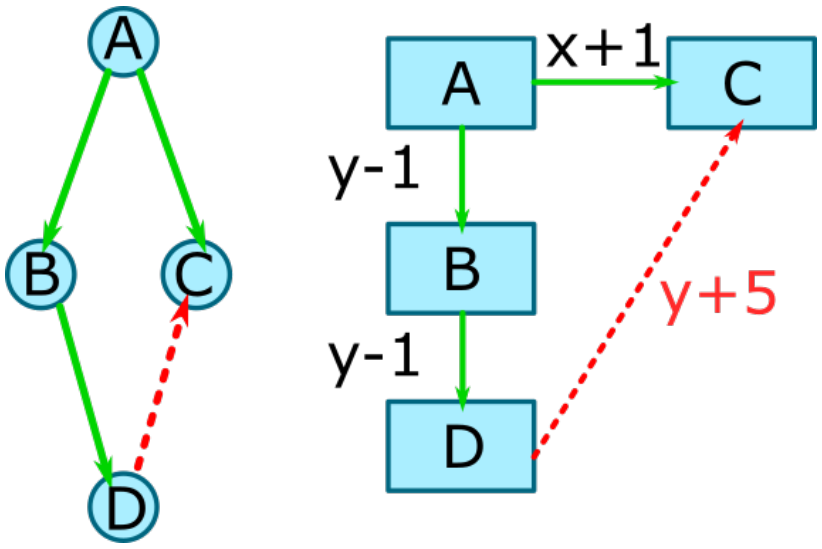


Figure 2.18: *A graph that is not acyclic, but still not acceptable*

In this step, we are going to attach one node at a time, and we need to verify that the graph is at least acyclic before the new node is inserted. To do this, we can use a label for each node to store its (temporary) depth (distance from the root). Every time there is the intention to connect a node c to a node p, we need to check their labels (l_c and l_p respectively) and whether they belong to another sub-graph or not. When c is the root of another sub-graph (or a disconnected node) or $l_c = l_p - 1$, it is accepted (and the labels of its sub-graph updated), otherwise rejected (Figure 2.19). In Appendix .1 is presented the demonstration that this procedure reduces the possibility of inconsistencies and shows its limitations.

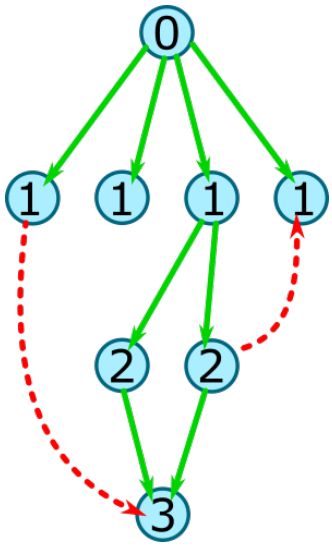


Figure 2.19: *In solid green arrows, some valid arcs, in dashed red some arcs that are discarded*

Algorithm

We are now going to show the procedure we use to connect each component to the others. The idea behind the procedure is that, in general, the objects in the child node (m_i^c) will be attached to some m_j^p; however this is not always the case, as there may be some m_i^c connected to some other m_k^c, when the node is not (only) collecting symmetric parts, but representing different parts of a bigger component. An example from PartNet is provided by Figure 2.20, where the leg of the chair (our child node c) is composed of multiple parts, but only one m_i^c is connected to a m_j^p, and the others are connected with other parts in the same node.
We are going to prefer external connections ($m_i^c - m_j^p$) to the internal ones ($m_i^c - m_j^c$), however, if two internal components touch and are not in contact with any part of the parent node, an internal connection is established. To avoid loops or disconnected parts, we are going to attach the nodes in an iterative way and every time we pick a node to be connected and attach it to another which is already part of the graph (has a parent).

In practice, this is implemented as:

1. Define the set $C = \{m_i^p \ : i \in \{1..n\}\}$ which will contain a subset of the nodes that are already attached to the connection graph (the last ones that have been connected);
2. Define the set $D = \{m_i^c : i \in \{1..l\}\}$ of the disconnected nodes to connect to the

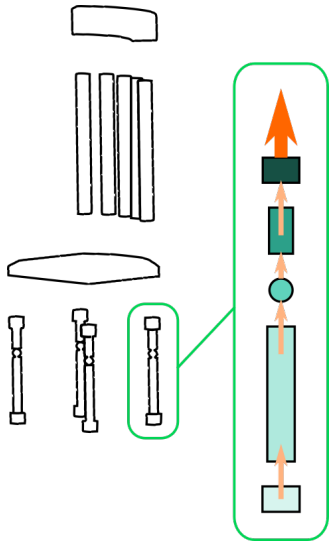


Figure 2.20: *In PartNet, the leg of this chair is composed of several pieces. Only one of them is connected to the seat, the rest are connected between themselves*

connection graph;

3. Until C and D are both not empty, loop on the following steps:
   a) Define the set $T = \emptyset$, which will contain the newly connected nodes;
   b) For each pair $d \in D$, $c \in C$, if they are in touch, and are compatible with the graph constraints, set c as parent of d and insert c in T;
   c) Set $C = T$;
   d) Define $D' = d \in C \wedge d \notin D$, the set of all the nodes still disconnected. Then update $D = D'$;

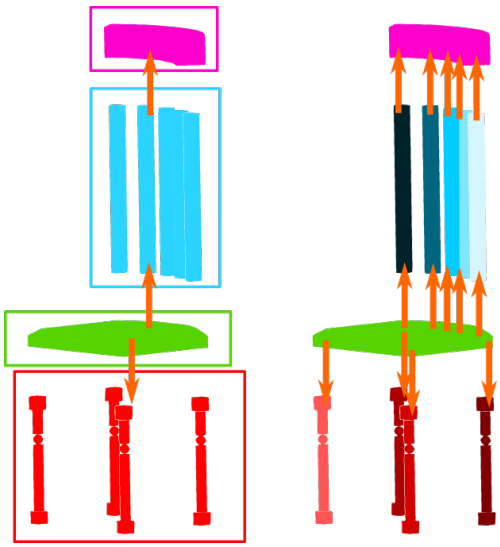


Figure 2.21: *On the left, a tree where the nodes are made by several components, on the right the tree has been extended to a graph in which the parts are connected in 1:1 relationships*

2.3.5 Explosion propagation

Up to now, we have discussed the explosion of one node of the explosion tree with respect to another. We must also notice that each part will have not only an explosion vector relative to the parent, v_p, but also an inherited displacement t_h, which represents the translation of its parent, so that its translation is $t = v_p + t_h$. If we treated only linear explosions, we could calculate the translations directly on the explosion tree, but, because there may be central explosions, we must do this on the final graph.

As the last step, we are then going to perform a BFS on the graph and assign the total translation of each node by inheriting the total translation of its parent.

2.4 Ground truth

Other than producing the images, we must also provide their respective data in order to be able to feed our dataset to a supervised learning model. A lot could be said about each diagram, but we will focus on the task of reconstructing the connection graph.
The dataset has been created with a certain structure which may need adaptations according to its application field, so we are going to discuss what is currently implemented but also what may be included. For the same reason, the hints on the structure of the data do not precisely reflect the practical implementation but represent a more readable simplification.

2.4.1 Components identification

In Section 2.2.5 we explained how distinguishing each part from the others is needed to produce the exploded diagram in its classic drawn-like style. The image we used encodes a label for each pixel, corresponding to a distinct colour for each part (or to the background) and, for convenience, it is stored as a colour image in which one colour channel holds the data. We are going to refer to this as *instances image*.

Components isolation

When we are going to need some semantic information about the diagram, we first need to identify and isolate each component. To do this, it is sufficient to store a map with the association of the information of a node and its correspondent colour on the instances image.
We can use this image also to generate other images, for example by treating it as a mask for the diagram.

```
[
    {id: 0, color: 1},
    {id: 1, color: 4},
    ...
]
```

Segmentation-oriented information

For the segmentation task, typically we are provided with a map containing pixels labelled according to the class they represent. Since we already have the instances image, we are not going to create an ad-hoc one, but rather specify the class for each instance and locate it thanks to the mapping specified above.

If we are not interested in isolating the components precisely, we may prefer to discard the instances image and keep only the bounding boxes of the object. A rough estimate of those may be extracted directly from the instances image, although it does not show the overlaps. We could more rapidly project the bounds of the 3D model and project it in the 2D space using the same camera matrix used to get the diagram, but it may also be more convenient to render all the objects separately so that we can use these images to study the edges hidden by other components.

```
[
    {id: 0,
    color: 1,
    class: 'leg',
    bounds: [301, 523, 324, 579]},
    ...
]
```

2.4.2 Relationships

To understand how the components are connected to form an object, we need to have a ground truth for the explosion directions and the parent-child relationships.

One approach to represent all the data is to collect all the data in a nested fashion:

```
{
    id: 0,
    <properties...>,
    children: [
                {
                    id: 3,
                    <properties...>,
                    children: [...]
                },
                {...}
              ]
}
```

To ease the access to the data we chose a different approach, which consists in splitting the information about the individual components and their relationships into two separate chunks. In the chunk of data relative to the individual pieces we are going to list all the parts and

their properties, while to show the connections we list the pairs of connected nodes and their relationship:

```
{
   parent: 3,
   children: 6,
   explosion_axis: 'x',
   versus: 'negative',
   explosion_distance: 4.5
}
```

2.4.3 3D reconstruction

To reconstruct the image in 3D we are going to use the information relative to the camera used, the translation applied to the components and their 3D model. We are also including the information about the axes of explosion, as they may be useful in the reconstruction.

```
scene:{
    camera_matrix = [[1, 0, 0, -1],
                     [0, 1, 0, 4],
                     [0, 0, 1, -5],
                     [0, 0, 0, 1]],
    explosion_axes = [[1, 0, 0], [0, 1, 0], [0, 0, 1]],
}
components: {
    1: {
        translation: [-4, 0, 0],
        model: 'models/leg_5.obj'
    },
    ...
}
```

According to the strategy adopted, it may be very useful to include also the face normals and depth with respect to the camera, which can be provided through two additional render images.

Dataset Results

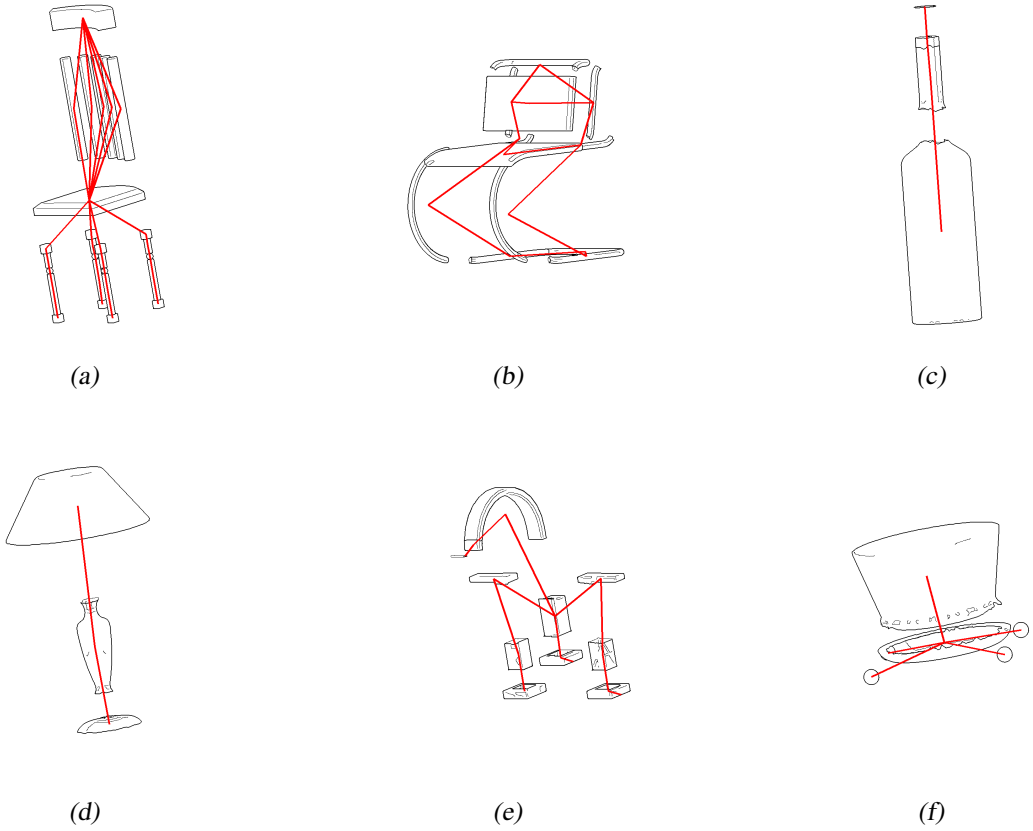
In this chapter, we are going to discuss the points of strength and the weaknesses of the dataset generated.

3.1 Connections

To visualize the connections between the components of a diagram, we have drawn some lines that connect the pieces according to the graph of connections. Some examples are shown in Figure 3.1, and in most of the cases the results we obtain are coherent with what we could expect from reality.

Strengths

The strength of the algorithm lies in the preliminary collection of symmetric or alike components, which are considered as separate entities only in the end. By creating first a graph where the nodes are constituted of groups of objects and then expanding it into one which distinguishes each individual part, we create two different refinement layers which increase the precision in the solution and the symmetry of both the objects and their connections.
This technique allows also to solve more easily the problems of possible loops, which could be potentially very expensive to solve in highly connected objects (depending on the technique used). A simplified example is provided by the chair of Figure 3.1.a, where from the top we see a horizontal bar, several vertical bars, a seat and the legs. In this case, the vertical bars vb_i are grouped together, which assigns them all the horizontal bar hb as a common parent. Yet, if they weren't, there could be some loops that we would need to solve: defining the symbol $p \rightarrow c$ as p is parent of c, we could have **seat** $\rightarrow vb_1 \rightarrow hb \rightarrow vb_2 \rightarrow$ **seat**. To solve this loop we could



(a) *(b)* *(c)*

(d) *(e)* *(f)*

Figure 3.1: *Some examples to show how the pieces have been connected. A red line connects the centres of two objects which are connected.*

break a link, for example $vb_2 \rightarrow seat$, but this would be semantically wrong, as such a connection does actually exist. Another option would be to try flipping the arc ($seat \rightarrow vb_2$), but this could cause other loops that would need to be handled as well and would probably require anyway to break some arcs to break the loops. Grouping first of all simplifies the graph, thus reducing the possible loops, but most importantly is proven effective when we get to the point in which we must choose which arcs to cut, because removing a connection between groups affects the elements contained in them in the same way.

Weaknesses

However, we can also see how some non-detected symmetries, combined with some imprecision, can lead to wrong connections. For example, in Figure 3.2 we are presented with the diagram of an armchair whose armrests were not in the same node in the PartNet tree. While the connections $d-e$ and $b-c$ are correct and symmetrical, armrest b has an additional connection to a, which creates an asymmetry which we can consequently find also in the explosion phase. Other issues arose from specific classes of objects, such as the plants. In their case, we decided not to explode the leaves which, however, we are still treating as single objects anyway. Because many leaves are close to others, often in a chaotic fashion, it can happen that in some objects the reconstruction of the connections fails, such as in Figure .9.

3.1.1 Conclusions about the scene graph construction

The phase of connections detection is generally accurate but suffers from some problems which mainly arise when the PartNet tree relative to an object varies its shape with respect to the standards and when the objects are particularly complex and dense.

One solution could be to (at least partially) drop the reliability of the system on the grouping of objects provided by PartNet, which could be substituted by an iterative procedure of corrections to remove optimally the loops or by a real object symmetry detection. While these solutions would be more reliable than the one chosen, they come at the cost of computational complexity and have not been adopted for scalability reasons, for the goal is to generate many images.

3.2 Explosion

The explosion phase is primarily governed by the connections that have been detected, and the main issue of this step consists in identifying the explosion direction of a component with respect to its parent. Some examples from different object categories are shown in Section .2.

This section of the work has been proven to be the most critical part, because it is both fundamental and hard to generalize. With the key idea of preparing a generator the most general as possible, we worked trying not to focus too much on a single category of objects but focusing on an algorithm capable of working on all the subjects in PartNet and possibly extendable to other datasets. This impacted the quality of some simple objects, such as the knives, which in their simplicity still sometimes fail in exploding correctly.

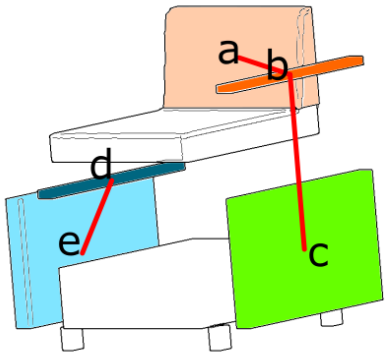
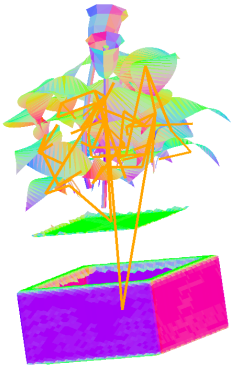


Figure 3.2: *An example of bad connections, which lead to asymmetries and a harder to understand diagram*

(a) A positive example of connections in a plant



(b) In some cases, plants suffer from important mistakes

Figure 3.3: *Two examples of connections from the plant class*

The PartNet dataset was the best fit for our needs at the time of writing but, not being made for our scope, it offers poor information about how some pieces lock into others and this forced us into looking for the new heuristics we have shown in the previous chapter. Generalizing these rules and keeping them efficient has been more challenging than expected, especially in presence of curved or branched shapes: we can see how these irregular shapes affect the results more than the almost rectangular ones (Figure .5.e, the filter of the faucet is exploded in the wrong direction).

The different heuristics of explosion have been assigned an importance factor which is independent of the class of object represented, and have been assigned the priority:

1. If the component is inside its parent, try escaping from it along the axis and directions that allow getting farther from the parent without crossing it;
2. If the two components appear to share a face, push the children away along the normal of such face (to be more precise, its principal component along the axes);
3. If the centroids of two components are enough aligned along one axis, use that for the explosion;
4. Otherwise, go away along the normal of the biggest face of the parent.

This sequence of rules has been established to balance the quality between all the classes of objects, but it could be modified to suit better a specific one: for example, the bottles (Figure 1.3) and knives (Figure .6) are more likely to have components exploded along the vertical direction, and we could first of all check whether this is likely to be a solution. Furthermore, due to the mesh complexity (Figure .9) and the problem of scalability to multiple models (to generate a big enough dataset), we chose to simplify some checks by sometimes approximating the objects, and this affects the quality as well.

3.2.1 Conclusions and future improvements regarding the explosion

Trying to generalize the algorithm to fit many classes of objects has opened the exploration of different techniques that are valid for multiple types of objects. However, it is hard to truly generalize without affecting specific classes and, for this reason, it would be better to balance the techniques giving priority to some over the others depending on the object represented, or considering the classification of each component.
If we want to consider the class of the whole object, we will look for which explosion criteria work best (e.g., explode a knife along the vertical axes most of the time). A more refined result can instead be obtained by studying the relationship between each possible pair of types of components and how a couple is likely to be exploded based on what they represent (e.g., the legs of a chair are likely to be exploded vertically with respect to the seat).

Applications and future work

In this chapter we are going to discuss some application fields of the dataset, proposing a pipeline of semi-independent blocks that could be of interest for future works.

4.1 Clean up

This work started from the necessity of recovering old exploded-view diagrams so, with this in mind, we can assume that we are not given a clean digital image, but rather a photograph of a piece of paper, possibly even hand-drawn.

4.1.1 Noise

Noise in pictures

As presented by [VA], when we deal with pictures obtained by a camera we can not avoid the presence of unwanted noises, which we can classify as:

- Impulse (or salt and pepper) noise: caused by sudden and sharp changes in the signal (for example, caused by dust). It appears as some isolated pixels of different colour intensity;
- Gaussian noise: adds a Gaussian distributed value to each pixel, independently from its intensity;
- Poisson: caused by the lack of lighting of the camera sensor, depends on the intensity of the pixels;
- Speckle (which is a major issue in other types of images, such as radar)

The presence of noise is going to affect the subsequent steps of our work if we did not take it into account when learning through synthetic data. To solve this problem we may choose to filter it out, as suggested by [VA], or to analyze it through deep learning methods as done by [TFZ+]. For the latter strategy, we have multiple possible solutions to choose from, but we must also consider that, because of the nature of our images, we won't need too complex models: indeed, we expect to find many contiguous high-signal (dark) pixels in the edges and isolated dark pixels are very likely noise.

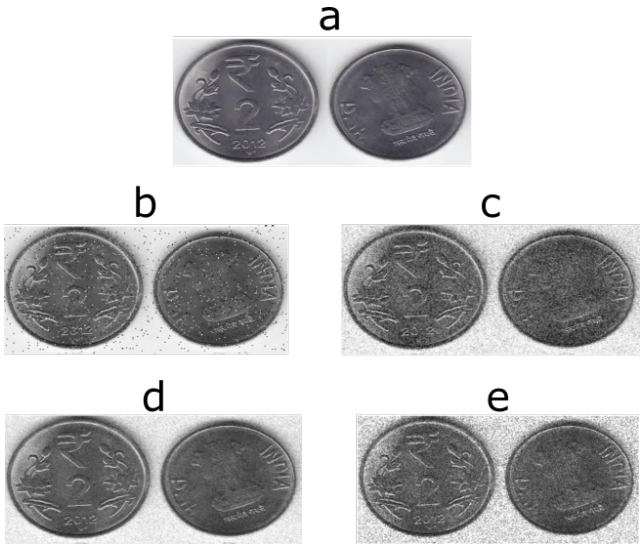


Figure 4.1: *The different types of noises, from [VA].*
a. Original
b. Salt and pepper
c. Gaussian
d. Poisson
e. Speckle

Testing on black and white images

We conducted a small test to show how the procedure of deletion of noise is simplified in the case of pictures of black and white images.
First, we took a grayscale picture in good lighting conditions of a printed diagram of a washing machine from a well-kept instruction book (Figure 4.2, left). It can be seen that everything is well visible and readable, but the lighting is not uniform and it would be helpful to restore the image in its black and white version. Because of the light not being homogeneous in the whole picture (see for example the upper-left corner), global thresholding is not recommended, instead, we use a *binarization* procedure which consists of stretching the histogram to increase the contrast and then thresholding. We can see that the obtained image (Figure 4.2, right) keeps the most important lines in the image and removed the paper texture and camera noise.
We can consider this picture a good representation of the noise that an image taken in recent years may suffer, however, we may want to study what would happen to images taken with much older cameras or taken on old, ruined paper. We preferred to not investigate too far on the restoration of these images, as tools have already been made available for this purpose, but we

will show anyway a trivial solution for the problem to show how we can solve it when having black contours.

To the same image, we applied additional pepper noise (affecting 10% of the image), obtaining the second picture of Figure 4.3. We proceeded to apply the same binarization operation as before but, as can be seen in the third picture of the same figure, it was not enough to clean all the noise. Therefore, we applied an additional step consisting of:

1. Application of a median filter: a non-linear filter whose response is the median of the pixels to which it is applied. This results in the reduction of noise, however, produces bolder lines than the initial ones, which can be a problem if we wish to distinguish an object's lines and guides;
2. Masking: to restore a more faithful image we can calculate the logical and between the binary image and the median filtered one; this will set the thin lines to their original thickness, but restore as well the holes in the lines caused by 0-valued noisy pixels;
3. Closure: the closure operation consists in dilating and then eroding the lines, and it can be useful to fill small holes. According to the size of the image, it may useful but it could also cause the merge of close parallel lines, so it may be skipped if not needed.

The result is in the last picture of Figure 4.3.

Following this procedure we can avoid studying separately the other types of noise, as after the binarization phase we find ourselves with the same problem we had with the pepper noise, confirming that ours is indeed a simplified case of denoising.

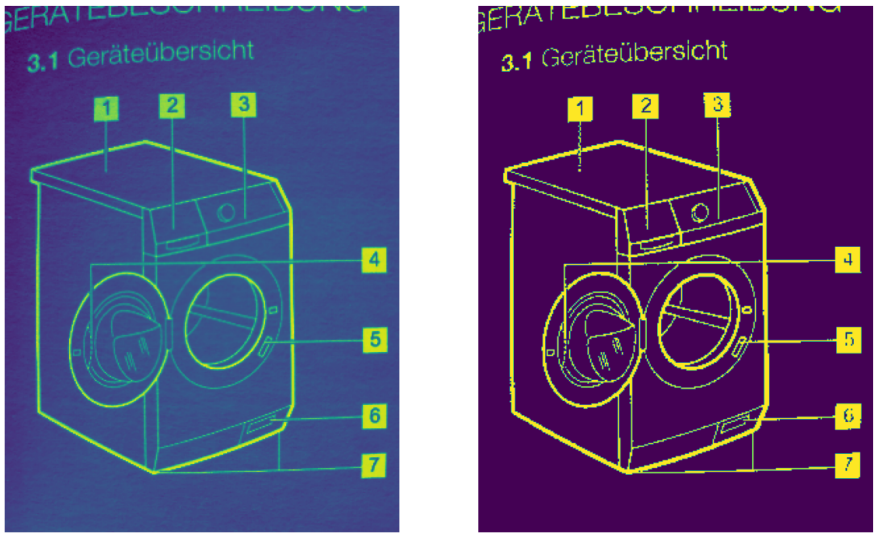


Figure 4.2: *On the left, is a grayscale picture of a diagram. On the right, its binary correspondent is obtained by increasing the contrast and hard-thresholding*

4.1.2 Notes and lines - Future work

Elements that we didn't include in our dataset are guidelines and notes that add information to the diagram or help in interpreting it correctly. We decided to not add this type of information

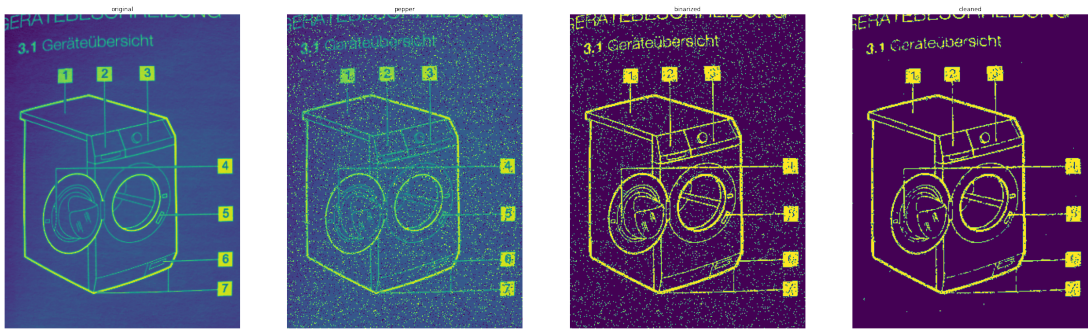


Figure 4.3: *From left to right, the original image, the image affected by pepper noise, its binarization and its cleaned version*

to avoid polluting the image with lines which do not belong to the object, but they could be added in the future to improve the dataset. However, even in that case, it is necessary to mask them away if we want to reconstruct the object's components.

This problem can be reconducted to a binary segmentation in which one class is representative of the useful information and the other of the notes we need to remove. One simple possible solution is to implement a U-Net [RFB], a network composed of an encoding and a decoding module which will classify each pixel as either useful or not. We also need to consider the imbalance problem of the classes, because, depending on how the background is classified, one class will be dominant over the other and if this is not taken into account it would lead to disregarding the dominated one.

4.2 Super-resolution and vectorization - Future work

Always keeping in mind that our goal is to be able to apply our algorithms to old images, it is likely that these will have a low resolution. Having as a goal to be able to handle these images, the dataset we created is already not in high resolution, to be more faithful to the images we will likely have to deal with.
Increasing the quality of the image could both be our goal or a tool to increase the precision of the following steps, but, it could as well be replaced by the vectorization of the diagram. We will now briefly illustrate the advantages and disadvantages of the two methods, to help decide which is the best solution relative to the desired application.

Raster vs. vectorial

In the raster representation of an image, it is shown as composed of a matrix of pixels which represent the colour and optionally the transparency. This is the traditional way in which photographs are represented. The quality of the image is dependent on its size, hence on its space occupancy, and is reduced when zooming in.
Producing a high-resolution raster image of a diagram can be helpful to refine its details, however, we must consider that this comes to a cost for possible future steps: if we are using super-resolution as an intermediate tool, we must take into account that increasing the size of

the image affects negatively the training times of a learning process. On the other hand, raster images are better than vector ones to understand local information, as access to neighbouring pixels is a cheap operation.

The vector representation of an image is better suited to digital images, as it is based on the decomposition of an image into basic shapes and curved lines. In images like ours, this type of representation is a very good fit, as it allows us to reduce a lot of space occupancy and has the great advantage of being scale independent: we can zoom on the image without loss of quality. The images produced by the dataset, in line with what we expect to have in practice, are however raster, so we would need to convert them. This representation is optimal if we want to use this as our final output, but it is less practical when we want to use it as an intermediate step.

4.3 Object detection and segmentation - Future work

It is fundamental for the reconstruction of the object represented by the diagram to first be able to distinguish one piece from the others. According to the application, we may choose to identify them with two levels of granularity, by distinguishing each part from the others or by limiting to recognizing which class each component belongs to.

The segmentation task consists of classifying each pixel as belonging to one of the possible classes (e.g., leg, seat and back for a chair), and does not distinguish between individual objects. While in general this is not sufficient to be able to reconstruct the object in its compact form, it is an important step to understanding its semantics. This step could be carried out in parallel with the individual instances recognition to proceed with the following tasks having both this information.

4.4 Connections

To understand how to compose the object from its parts, we need to learn how they have been exploded. The first step in this regard is to learn which components are connected, and along which axes they are exploded one from the other.

In this section, we are going to show the approach we used to develop this type of problem using a deep neural network.

4.4.1 Rationale and reasoning on the prerequisites

The task the network needs to solve is to identify the relationship between two components of the diagram which may be connected or not and, when they are, which is the direction of the explosion. Analyzing one pair at a time loosens some constraints on the graph we are going to reconstruct (we know that all the components must be attached to others but, in this way, we can't enforce this rule), but it is easier to learn the relationship between only two elements

rather than reconstructing the whole graph in one pass.

It is fundamental to, first of all, be able to identify the two parts of interest on the image and extract their features, and we could use a more rough estimate and identify the area of interest with a bounding box or be more precise and use a pixel-wise classification of which one belongs to the component or not.
The usage of bounding boxes is a natural choice if we think of the other works about scene graphs which use object recognition as the first step [CRX+] [ZWYC] [HRC+], as we could build a network similar to theirs and feed it with the input diagram and the bounds of the components. However, identifying an object up to the pixel level is very helpful when we deal with objects very close one to the other due to the perspective (which could share a big portion of their bounding boxes but have different connections), and it is also a natural consequence of a refinement on the segmentation step. For these reasons, we are going to use pixel-level component detection.

To simplify our work and reduce the amplification of errors due to the pipelining of multiple steps, for testing purposes we are going to use the instances image from the ground truth in order to get a mask for each component.

4.4.2 Input

To decide what we necessitate in input, we need to make some observations on what is necessary to understand if and how two parts are connected. We may wonder if isolating the two components (by masking the original diagram) is enough to understand the scene, and we notice that whether two pieces are connected or not depends on:

- if they could be aligned along one of the explosion axes;
- their distance;
- if there are other parts between them that imply they may be indirectly connected through those.

None of these properties can be checked without the knowledge of the surroundings, so we need to keep the information about the whole image in parallel to the one relative to the single components.

Also looking at the next encoding step, we are going to use three different inputs for our network, instead of using only one with the information about both the components, so that we can always encode them homogeneously and independently. We use two inputs consisting of the masked version of the diagram, keeping only one component each, and the whole diagram as the third input (Figure 4.4).

4.4.3 Encoder

Before identifying if the components are connected, we need to extract and compact their properties, which will be stored in a feature vector. As we are dealing with images, it is important to

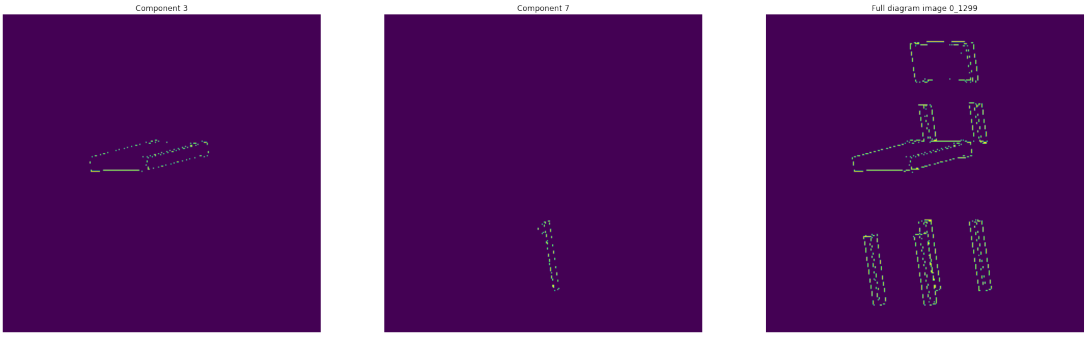


Figure 4.4: *One example of input of the connection classifier. On the left are the two components to compare, on the right the original diagram (at low resolution)*

keep in mind the surroundings of each pixel, as one is meaningful only together with its context; this is the reason why we are going to use a convolutional neural network (CNN) [AMAZ].

The encoder is built of repeating blocks constituted by:

1. 2D convolution: allows mixing the information of a pixel with its surroundings. The neurons are activated by a ReLU (Rectified Linear Unit) function.
2. Max pooling: reduces the dimensionality of the input

ReLU is an activation function defined as $f(x) = max(0, x)$. The choice of *ReLU* over other functions, such as sigmoid or hyperbolic tangent, is a popular one for its simplicity (ease of derivation) and its zeroing of the gradient which can help the training in deeper networks.

The full encoder structure is represented in Figure 4.5

4.4.4 Merge of branches

As the inputs are in the same shape and represent the same type of information, we are going to use the same encoder for them all. However, we need to reason on all the inputs together to predict the connection type and for this reason, after the encoder part, we add a concatenation layer to combine the three encodings for the images.

4.4.5 Classifier

A flatten layer prepares the concatenated features for two dense layers, which are used for predicting between 4 one-hot encoded classes: *no, x, y, z*, which represent no connection between the two pieces, or connected and exploded along axes x, y or z.
One-hot encoding means associating an index to each class and assigning to a vector the value 1 to express belonging to it or zero otherwise. In our case, the vector $[1, 0, 0, 0]$ means that the components are not directly connected and $[0, 0, 1, 0]$ that they are exploded along the y-axes one with respect to the other.

While the first dense layer uses, again, ReLU, the second and last one uses *softmax* activation

function:

$$\sigma(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1..k} e^{x_i}}$$

where **x** is a vector of size k.
This choice is fundamental to get a meaningful prediction, as softmax takes as input a vector **x** and outputs a vector $\mathbf{y} = \sigma(\mathbf{x})$ where each element x_i of **y** belongs to the range $(0, 1)$ and $\sum x_i = 1$. These properties allow to interpret the values in **y** as the probability of belonging to a class (e.g., [0.1, 0.8, 0.1] means that the probability of belonging to class 1 is 80%, while it's 10% for classes 0 and 2). If we are not interested in the probabilities, but rather wish to get a single value (the most likely class), we can extract it with $argmax(\mathbf{y})$.

The whole network structure (with the encoder compacted in one block), is shown in Figure 4.6

4.4.6 Loss

In multi-class classification problems, such as ours, the loss function used is *categorical cross-entropy*. Applied to the machine learning context, it compares individually each true expected value $\hat{y}_i$ for the class i with its predicted value y_i and gives a measure of their dissimilarity:

$$L = -\sum_i \hat{y}_i \cdot \log y_i$$

Classes imbalances

When calculating our loss metric, however, we must consider how the probability of belonging to each class is distributed on our dataset. The distribution of being exploded along each one of the possible axes depends on the object represented in the diagram (e.g., a chair has most of the components stacked vertically and will be exploded mainly along the y-axes), but choosing a uniform dataset balances all the three possibilities quite uniformly. However, the more components we have in a diagram, the fewer pairs are connected with respect to all the possible combinations, which leads to a great imbalance in the distribution of the class 'not connected' with respect to the others.
To handle this, we need to modify our loss in order to not focus only on succeeding in predicting the 'no connection' class, but give more importance to the other events, which are more important but also rarer. This is done by checking the frequency of each class in the dataset and assigning a weight which is dependent on its frequency: $weight_c = \frac{\#pairs\ of\ class\ c}{\#all\ pairs}$.

Test

We trained this network with a small set of chairs as proof of concept. The network is given different points of view of different chairs and adds a small random rotation and a horizontal flip for data augmentation.

The results can be seen in Figure 4.7 and Figure 4.8. Although the results are not perfect, due to a short train on a small dataset, we proved how this task can be performed also by a very basic model.

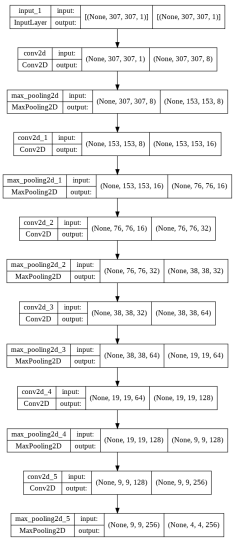


Figure 4.5: *The encoder network, whose task is to extract the features from the images*

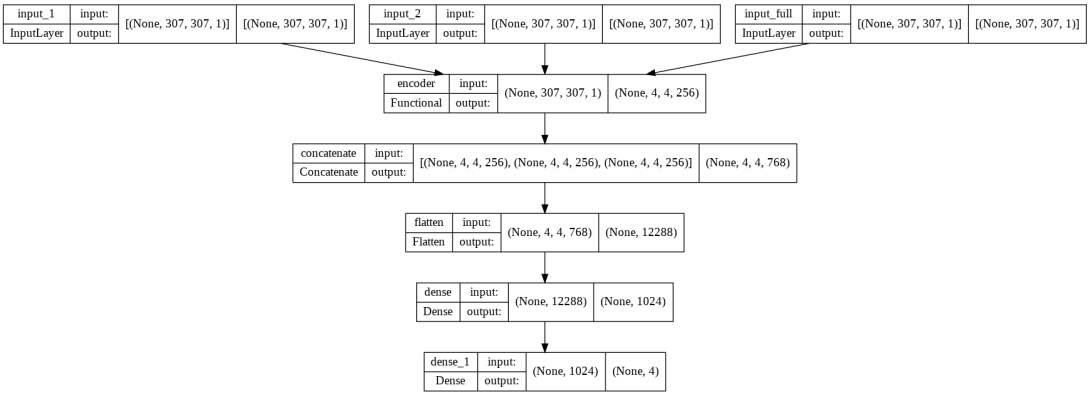


Figure 4.6: *The whole network, which includes the encoder one shown as a single node and the classifier*

4.5 Reconstruction of hidden parts -Future work

It can happen that in the diagram we find some objects occluded behind others, and this is an obstacle for the reconstruction, both 2D and 3D. To reconstruct the hidden parts, we can first identify the occluded parts and their occluders ([KTT] provides a comparison between different possible solutions to the problem), then we may complete them by using the occlusion masks to trace the areas where we expect the continuation of the object to be (similarly to [ZCC]).
An alternative approach could be based on the work of [LLP+], which proposes a solution to change the point of view of a 3D object having a single picture of it. This would not the best approach if our goal is the 3D reconstruction, but it is useful if we want to change the focus of the diagram to a different component and we don't need to go through the whole 3D reconstruction.

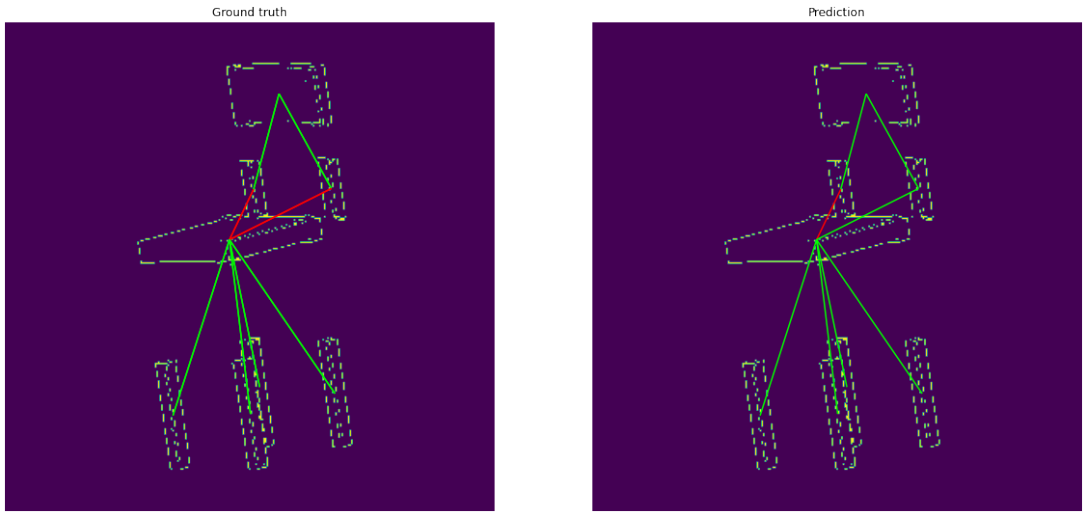


Figure 4.7: *On the left the ground truth of the connections, on the left the prediction. Green lines means that the components have been exploded along y axis, red along x*

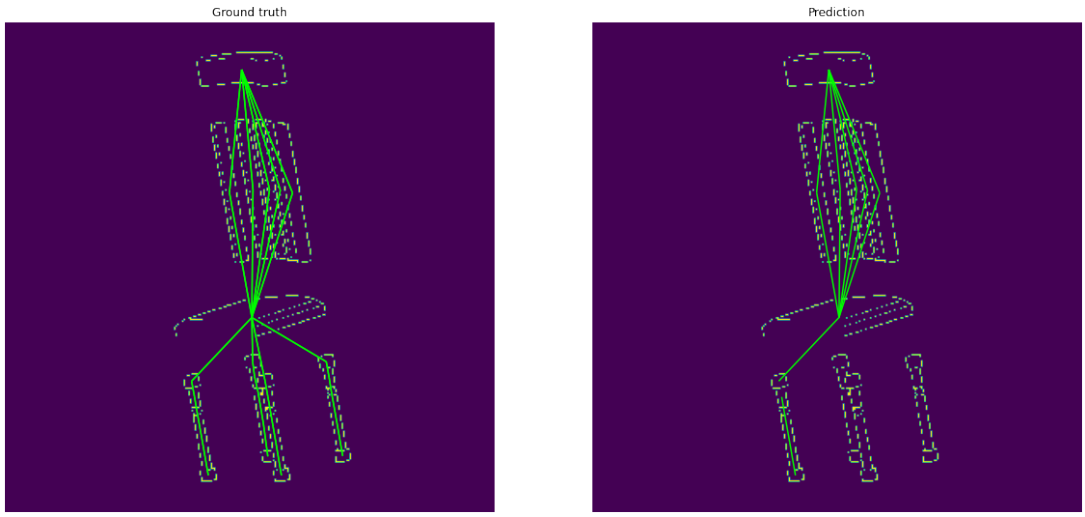


Figure 4.8: *Another example like Figure 4.7*

4.6 3D reconstruction of edges and surfaces - Future work

As for images we can either work with a raster or vector representation, the equivalent can be said for the 3D models, in which we may work with meshes or point clouds. It is desirable to work with meshes, as they are more scalable and easier to modify, and it would be even better if we could trace back our shapes to elementary ones, to make the model parametric and adjustable. For both possibilities [ZGZS] presents and compares different solutions that we can easily adapt to our images, as the networks they present have been trained on sketches of the same models of ours.
To improve our work we can detect and exploit the symmetries of the object but especially identify identical components and merge the information given by them all to improve accuracy

and solve the problem of hidden parts.

Conclusion and Outlook

In this work, we set the basis for the automatic generation of exploded diagrams and information extraction from those.

We started by observing the importance of having a well-structured object, which must consist of logically coherent pieces and coherent connections between them. For this reason, we discarded the possibility of generating images from arbitrary objects by using randomly generated scene graphs and chose to start from existing composite 3D models. This led to the choice of using PartNet as the basis of our work, and exploiting its segmentation of the objects to both use its components but also to add semantic value to our diagrams.
Afterwards, we found the connections between components by looking both at physically connected nodes and embedding the semantic relationships represented by the PartNet trees. By grouping similar objects we built a tree that we use to learn the explosion vectors.
We then defined different heuristics to learn how one component can be exploded with respect to the others, and we studied how to balance them to obtain an algorithm capable of dealing with different types of objects.
Finally, we proposed a pipeline for the reconstruction of the objects represented in the diagrams.

In this study, we learned how challenging it can be to define an algorithm capable of working on a very heterogeneous set of objects in an efficient way. To improve this work, before refining the analysis of the single meshes (which is too time-consuming), we suggest incorporating more information prior to the explosion to choose the heuristics that are expected to work best. Nonetheless, we have also seen how the reconstruction of the object can be performed by simple algorithms and deep neural networks, which hints at a promising future development in the field.

Appendix

.1 Demonstration of limitation of inconsistent dependencies in graphs with arcs bound by depth constraints

Let us define:

- G: a directed graph composed of the set of nodes N and the set of arcs A. To each arc a is associated a translation vector t_a. We restrict the possible graphs to the case in which they have only one root (our case study) but it can be extended to multiple.
- $T_P = \sum_{a \in P} t_a$: the composition of the translations along a path $P \subseteq A$
- *Inconsistency*: G presents a (potential) inconsistency when there exist two paths p, q from the root to a node n such that $T_p \neq T_q$.
- *Depth constraint*: a node n can accept m as its child only if m is the root of a sub-graph which without n would be disconnected to the rest of G, or if their depth respects $d_n = d_m - 1$. Remember also that in the first case the labels of the sub-graph are updated to update their depth accordingly to the graph they are connected to ($d_n = d_m - 1$ and the same repeats in ms children).

We observe that:

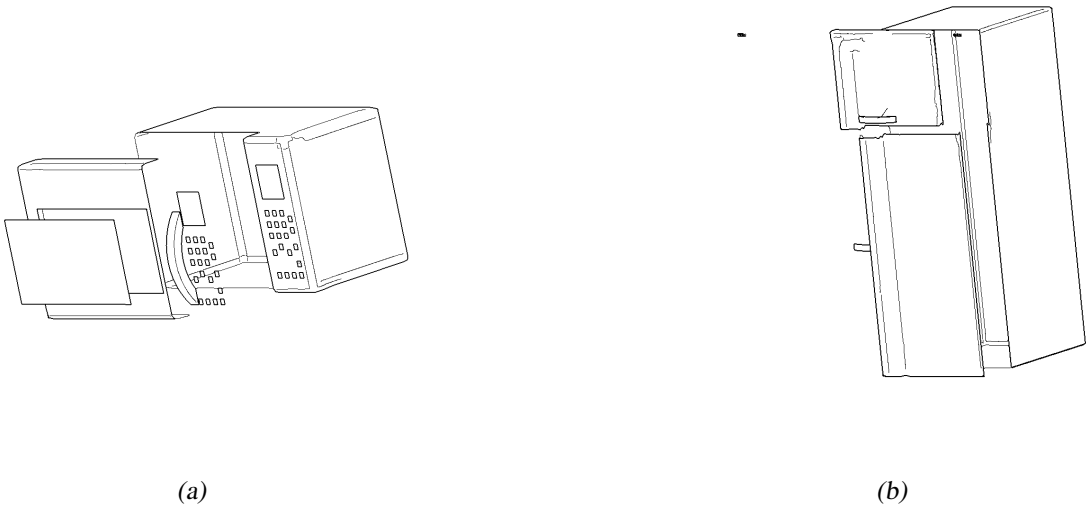
1. A tree has no inconsistencies, since there exists one and only one path from the root to any node.
2. When the depth constraint is respected, there can't exist a path both from m to n and viceversa.

3. If all the connections from m to n respect $d_n = d_m - 1$, then there can't exist loops, which would require the existence of a path $p = (n_0, n_1, ..., n_k, n_0)$ in which $d_{n_0} = d_{n_1} - 1 = d_{n_2} - 2 = ... = d_{n_k} - k$ but also $d_{n_k} = d_{n_0} - (k + 1)$, which is impossible

Our graph comes originally from a tree G' which sets some constraints on G: there can exists an edge $a = (n, m)$ in G only if n and m belonged to the same node of G' or n was in the parent node of the one containing m. Since there are no inconsistencies in G', they can originate in G if there exist $n', m' \in G'$, and $n, m, l \in G$ with n and l associated with n' and m to m', such that there exists both the arcs (n, l) and (n, m).
Because of this, the check of absence of inconsistencies becomes a local property (dependent on the strict neighbours of a node) instead of being global and requiring the visit of all the graph.

.2 PartNet explosion results

Figure .1, Figure .2, Figure .3, Figure .4, Figure .5, Figure .6, Figure .7, Figure .8 and Figure .9 show some examples of generated exploded diagrams.

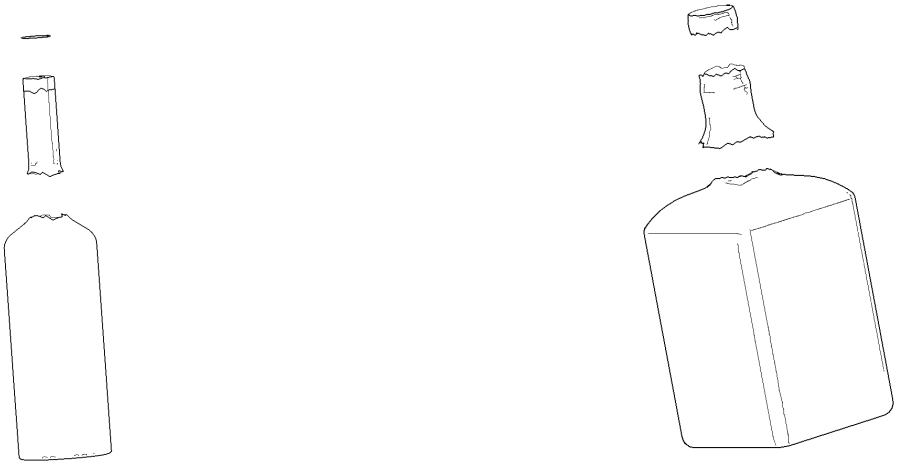


(a) *(b)*

Figure .1: *Electrical appliances*

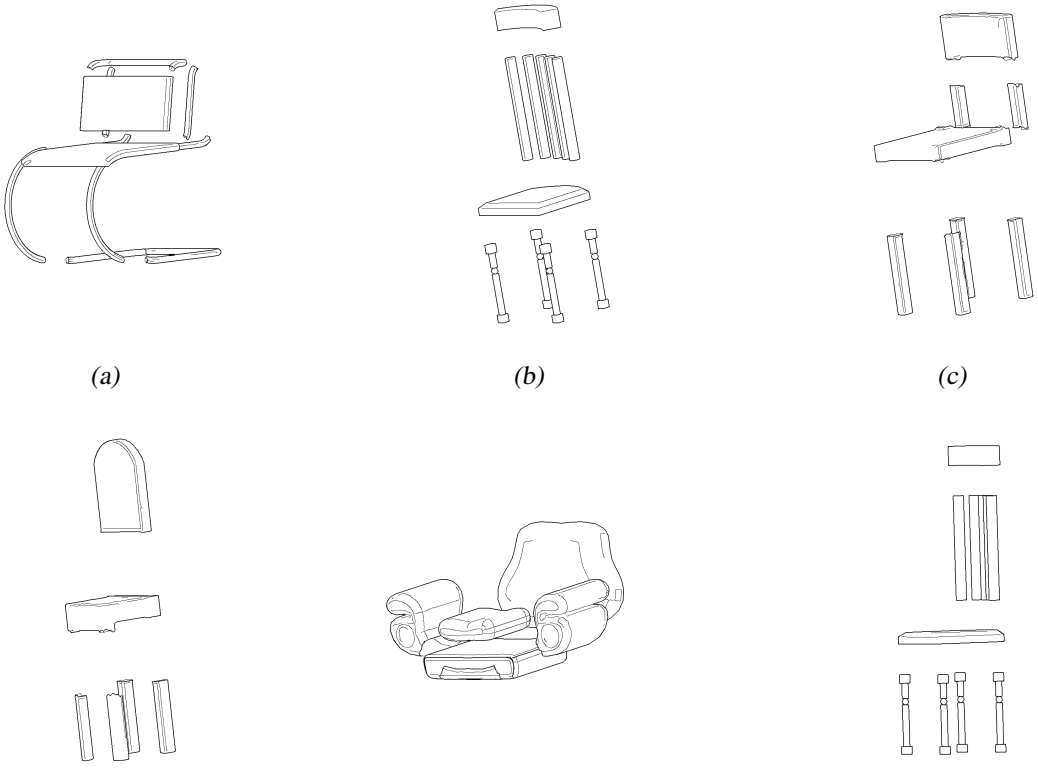
.3 Software used

The generation of the diagrams has been written in python with the fundamental help of the libraries trimesh [ea] and pyrender [noa]. The deep learning implementations have been written in python with tensorflow and keras.



(a) (b)

Figure .2: *Bottles*



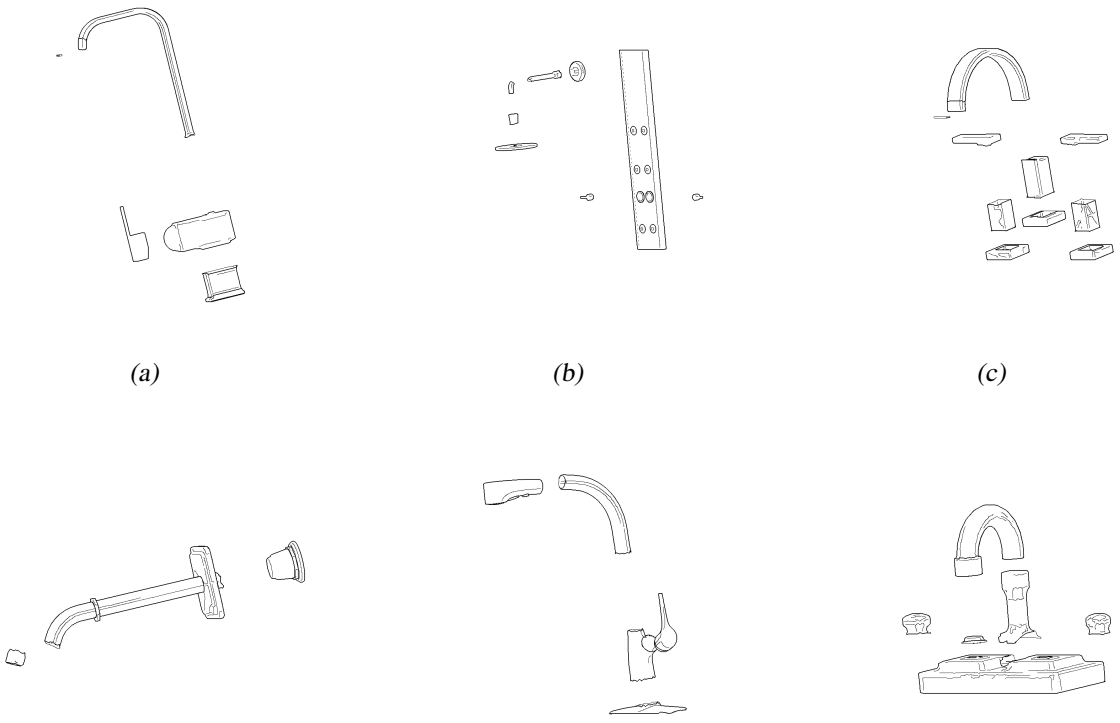
(a) (b) (c)

(d) (e) (f)

Figure .3: *Chairs*

(a) *(b)*

Figure .4: *Mugs*



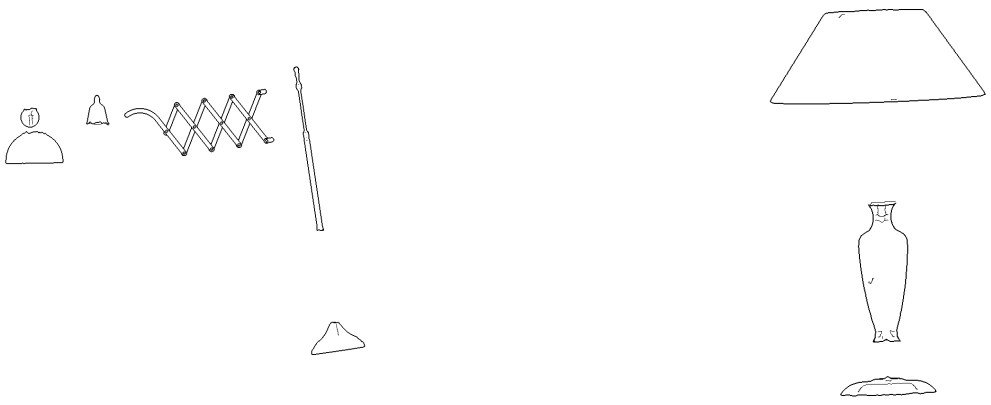
(a) *(b)* *(c)*

(d) *(e)* *(f)*

Figure .5: *Faucets*

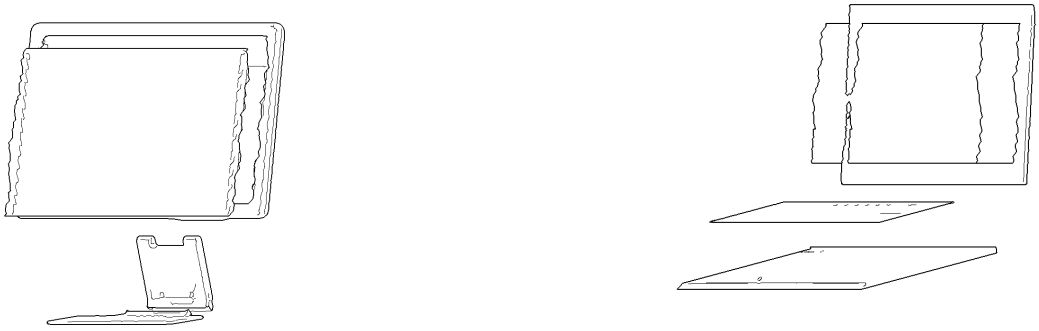
(a) (b)

Figure .6: *Knives*



(a) (b)

Figure .7: *Lamps*



(a) *(b)*

Figure .8: *Computers*

(a) *(b)*

Figure .9: *Plants*

Bibliography

[AMAZ] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE.

[CRX+] Xiaojun Chang, Pengzhen Ren, Pengfei Xu, Zhihui Li, Xiaojiang Chen, and Alex Hauptmann. A comprehensive survey of scene graphs: Generation and application. pages 1–1.

[DC] Elena Driskill and Elaine Cohen. Interactive design, analysis, and illustration of assemblies. In *Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 27–34.

[ea] Dawson-Haggerty et al. trimesh https://trimsh.org/.

[HRC+] Roei Herzig, Moshiko Raboh, Gal Chechik, Jonathan Berant, and Amir Globerson. Mapping images to scene graphs with permutation-invariant structured prediction.

[KMJ+] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. ABC: A big CAD model dataset for geometric deep learning.

[KTT] Lei Ke, Yu-Wing Tai, and Chi-Keung Tang. Deep occlusion-aware instance segmentation with overlapping BiLayers. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4018–4027. IEEE.

[LACS] Wilmot Li, Maneesh Agrawala, Brian Curless, and David Salesin. Automated generation of interactive 3d exploded view diagrams. 27(3):1–7.

[LAS] Wilmot Li, Maneesh Agrawala, and David Salesin. Interactive image-based exploded view diagrams. page 10.

[LLP+] Bingzheng Liu, Jianjun Lei, Bo Peng, Chuanbo Yu, Wanqing Li, and Nam Ling.

Novel view synthesis from a single image via unsupervised learning.

[noa] pyrender https://github.com/mmatl/pyrender.

[RFB] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation.

[TFZ+] Chunwei Tian, Lunke Fei, Wenxian Zheng, Yong Xu, Wangmeng Zuo, and Chia-Wen Lin. Deep learning on image denoising: An overview. 131:251–275.

[VA] Rhoit Verma and Jahid Ali. A comparative study of various types of image noise and efficient noise removal techniques.

[XZCFF] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene graph generation by iterative message passing.

[YLZ+] Fenggen Yu, Kun Liu, Yan Zhang, Chenyang Zhu, and Kai Xu. PartNet: A recursive part decomposition network for fine-grained and hierarchical shape segmentation.

[YRD] Kaiyu Yang, Olga Russakovsky, and Jia Deng. SpatialSense: An adversarially crowdsourced benchmark for spatial relation recognition.

[ZCC] Chuanxia Zheng, Tat-Jen Cham, and Jianfei Cai. Pluralistic image completion. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1438–1447. IEEE.

[ZGZS] Yue Zhong, Yulia Gryaditskaya, Honggang Zhang, and Yi-Zhe Song. Deep sketch-based modeling: Tips and tricks. pages 543–552.

[ZWYC] Alireza Zareian, Zhecan Wang, Haoxuan You, and Shih-Fu Chang. Learning visual commonsense for robust scene graph generation. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, volume 12368, pages 642–657. Springer International Publishing. Series Title: Lecture Notes in Computer Science.

[ZZJ+] Guangming Zhu, Liang Zhang, Youliang Jiang, Yixuan Dang, Haoran Hou, Peiyi Shen, Mingtao Feng, Xia Zhao, Qiguang Miao, Syed Afaq Ali Shah, and Mohammed Bennamoun. Scene graph generation: A comprehensive survey.