THE UNIVERSITY
of EDINBURGH

Advances in Scene Understanding: Object Detection, Reconstruction, Layouts, and Inference

Paul Henderson

Doctor of Philosophy

Institute of Perception, Action and Behaviour
School of Informatics
University of Edinburgh

2019

Abstract

The goal of *scene understanding* is to capture the full content of an image in a human-interpretable representation. This must describe the different objects present, including their attributes such as class, shape, and pose, as well as the relations between objects. Moreover, the representation should be globally-consistent across the entire image. In this thesis, we consider four sub-tasks within scene understanding, and make contributions to each.

When describing the content of an image, it is natural to start by detecting all the objects that are present—that is, localising and classifying them. Our first contribution is to show how to train a neural-network-based object class detector end-to-end in a principled fashion, using the evaluation metric as the training loss, and using the same pipeline at both training and test time. This is simpler and more elegant than the traditional approach of using a surrogate loss, yet we show it achieves comparable performance.

Once the location and class of an object are known, we can estimate its shape and pose in 3D space. Our second contribution is a new approach to these tasks, which supports training purely from 2D images—without 3D supervision, multiple views, or annotations such as pose or keypoints. Moreover, this model is generative, and so allows sampling new object shapes *a priori*.

To produce a globally-consistent description of a scene, it is important to reason over all objects simultaneously, rather than considering each individually. Our third contribution is a probabilistic generative model over complete indoor scene layouts. It models complex arrangements in 3D space, including high-order spatial relations among furniture and other objects.

One common approach to generating predictions that are consistent over all objects in a scene, or pixels in an image, is to formulate and solve a discrete energy minimisation problem. The energy is defined as a sum over factors, and the factor structure greatly affects what minimisation algorithms work well. Our fourth contribution is a method that automatically selects a suitable algorithm to solve a given energy minimisation problem. To do so, it learns to predict the best algorithm based on characteristics of the problem instance.

Acknowledgements

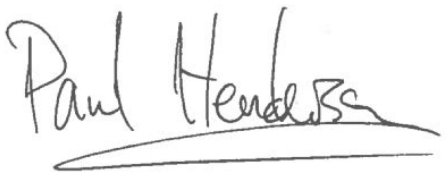
First and foremost, I would like to thank my supervisor Vittorio Ferrari for his guidance and insights over the past four years. Vitto's passion and strive for perfection have had a strong and lasting effect on my perspectives as a researcher. I am also grateful to my committee of examiners, Ram Ramamoorthy and Bastian Leibe, for taking the time to review this thesis, and to the School of Informatics for supporting my research.

I thank all the members of the CALVIN group, particularly Holger Caesar and Buyu Liu, for making my time here enjoyable. The same goes for my office-mates Boyan Gao, Ozzy Cebe, and Daniel Thul, for engaging conversations on vision, programming, and many other things.

Finally, I give my deepest thanks to my fiancée Lauriane, for her support and encouragement over the past four years. *Miaou et coin-coin!*

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.



Paul Henderson, 26th January 2019

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

The human visual system has evolved to understand scenes precisely and effortlessly. We can identify objects, perceive their shape, reason about relations among multiple objects, and imagine the position and even appearance of objects that are hidden from view. All this happens unconsciously, in a fraction of a second (Kersten and Yuille, 2014; DiCarlo et al., 2012). However, computer vision systems still struggle with this task of *scene understanding*. This is in spite of substantial progress over decades of research, and an appreciation that human-like scene understanding is a vital goal in computer vision, and artificial intelligence more generally.

For the purposes of this thesis, we consider the task of scene understanding as taking an image as input, and producing as output a human-interpretable representation that describes the different objects it shows, including their identities, shapes, and spatial relations. This representation should be globally consistent across the entire image; ideally it should also explicitly represent uncertainties due to ambiguous input, *e.g.* occlusions.

Scene understanding has myriad applications; we now give a brief flavour of some. A robot able to form the type of representation we describe becomes aware of positions and shapes of objects; thus, it can navigate without colliding with objects, and can plan grasping movements to pick them up. Self-driving cars must be able to identify and localise obstructions, pedestrians, and other road users, and to predict their future behaviour; this requires a global representation capturing their spatial relations and intents. Image search engines typically rely on simple queries for object or scene types, but could be greatly enhanced by allowing users to search for images satisfying semantically-complex queries incorporating relations among objects. Automated surveillance systems similarly benefit from a richer understanding of images,

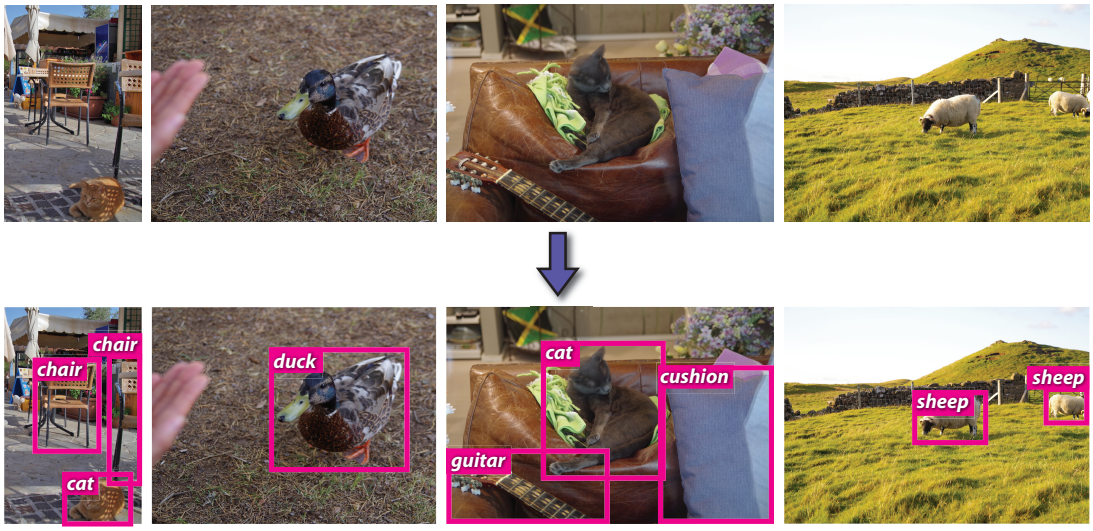


Figure 1.1: Object detection aims to recognise and localise objects in an image, by predicting a class and tight bounding-box for each instance of certain classes.

beyond simply identifying individual objects—thus allowing more accurate detection of crimes or hazardous situations. Finally, assistive technologies that describe images for the visually-impaired become more compelling when the system itself has a deep understanding of what is visible, and can pass on this understanding in its description.

The overall task of scene understanding is rather broad, and may be considered as the combination of many narrower sub-tasks. Research typically considers one sub-task at a time; that is also the approach we follow in this thesis. In particular, we do not attempt to build a system incorporating all our contributions, although we do present suggestions on how they can be combined in future work. The sub-tasks that we consider, and contributions made to each, are described in the next section.

1.1 Tasks and contributions

This thesis presents four main contributions, touching on four different, yet related, aspects of scene understanding; we now briefly describe each.

Object detection. In order to reason about objects and their relations, it is natural to first detect them. That is, given an image, we aim to recognise and localise all instances of objects of certain *classes* we are interested in, such as *chair*, *sheep*, or *cat* (Figure 1.1). Typically, this localisation is in terms of axis-aligned 2D bounding-boxes; the goal of the task is then that every object in the image is tightly enclosed by exactly one such bounding-box, correctly labelled with its class. By convention, object

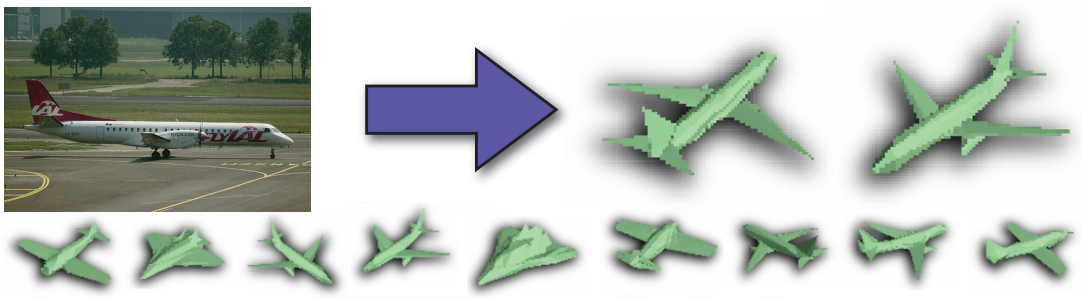


Figure 1.2: Single-image 3D reconstruction (top) aims to produce a complete 3D representation of an object, given just a single image of it. This is an ambiguous task, which must utilise prior information to mitigate depth ambiguities and reconstruct occluded regions. Our model for reconstruction is also generative: it allows sampling new 3D shapes for a given class *a priori*, without conditioning on an image (bottom).

detection is evaluated using a metric known as mean average precision (mAP). Our first contribution is a new method for training the object detector Fast R-CNN (Girshick, 2015), that directly optimises it for mAP. We train the final model end-to-end rather than using a different pipeline for training and testing (as in prior works), and we do not require a heuristically-chosen surrogate loss. This more-principled training method could also be adapted to other object detectors based on neural networks (*e.g.* Liu et al., 2016). This contribution was presented at **ACCV 2016** (Henderson and Ferrari, 2016b).

3D shape reconstruction and generation. As well as the class and 2D bounding-box produced by an object detector, objects are characterised by their 3D shape and pose. Reconstructing the full 3D shape of an object from a single image is a challenging and inherently ambiguous task, which involves reasoning over likely shapes given its class. Nonetheless, it is vital for applications such as grasp planning for robotic arms, and novel-view synthesis in computer graphics. Reconstruction is naturally cast as inference in a probabilistic generative model over object shapes for a given class. Our second contribution is such a model, that is able to reconstruct 3D shapes from a single image, to estimate their pose, and to generate new shape samples for a given object class (Figure 1.2). Unlike previous works, our method supports training from unannotated 2D images, without expensive 2D or 3D supervision. It outputs meshes, instead of voxels as used in most prior works, which allows us to exploit highly-informative shading cues that earlier methods cannot. This work was presented at **BMVC 2018** (Henderson and Ferrari, 2018).

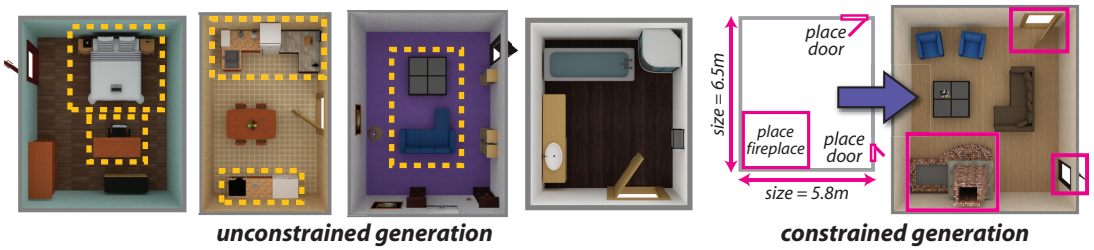


Figure 1.3: We design a generative model over room layouts, which allows sampling new layouts, and models high-order relations among objects (yellow dashed boxes). It supports user-specified constraints such as object placement (right; pink boxes indicate doors and fireplace placed by the user).

Room layout generation. To produce a globally-consistent representation of a scene, it is important to reason over all objects simultaneously, as opposed to just considering each independently. One context in which this is particularly relevant is indoor scenes, which tend to have a high degree of regularity, such as functional groupings of objects. Our third contribution is a probabilistic generative model over complete room layouts, capturing object classes, shapes, and positions (Figure 1.3). This models complex arrangements in 3D space, including spatial relations among large groups of furniture and other objects. As well as being relevant to holistic scene understanding, our model has applications in computer graphics, where it enables automated authoring of scenes. In particular, sampling from the model generates novel, realistic indoor scenes, and we also allow user-specification of constraints, which is important to artists who wish to control particular aspects of a scene.

Inference algorithm selection. Scene understanding tasks are often formulated as structured prediction problems; again, this allows for a globally-consistent result to be produced. The most common approach here is to pose the task as inference in a discrete energy-based model defined over factors. Our fourth contribution eases inference in such models: we develop a technique that can automatically predict which inference algorithm is best to use for a given problem instance, by learning from a large dataset of problems (Figure 1.4). This saves vision practitioners from having to be experts on inference algorithms too, as it provides a turn-key solution for inference in a broad range of models. This work was presented at **ECCV 2016** (Henderson and Ferrari, 2016a)

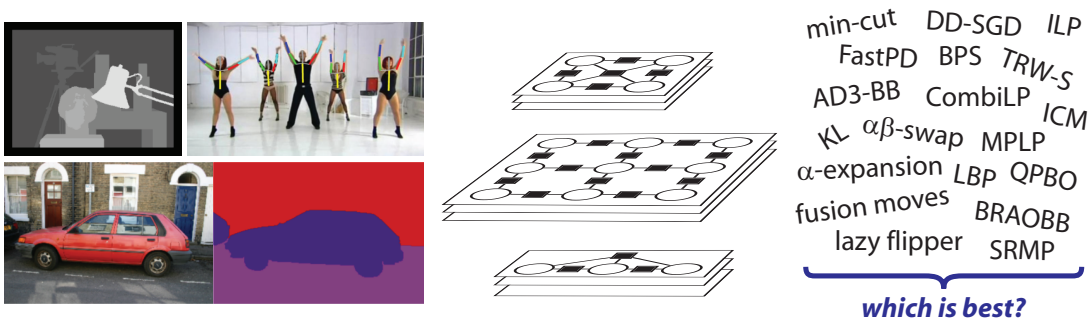


Figure 1.4: Vision problems producing high-dimensional outputs that must be globally-consistent (left) are commonly expressed using discrete energy-based models over factors (centre). Many algorithms exist to solve these problems (right); we automatically select the best algorithm for a given problem instance.

1.2 Organisation

We devote one chapter to describing each of our main contributions:

- **Chapter 2** describes our proposed method for training object detectors end-to-end for mean average precision;
- **Chapter 3** describes our model for generating and reconstructing 3D meshes, without 3D supervision;
- **Chapter 4** describes our probabilistic generative model of room layouts;
- **Chapter 5** describes our system for automatically selecting inference algorithms to solve discrete energy minimisation problems.

As the topics covered are quite heterogeneous, we present background and prior work on each problem separately in the corresponding chapter. Then, we describe our proposed method, present experimental results, and discuss and analyse our method. Each chapter concludes with a discussion of various extensions that we believe are promising for future research. Finally, in **Chapter 6**, we summarise our findings, and give a more general discussion of how the different elements described in the earlier chapters might be combined together as parts of a unified system for scene understanding.

Chapter 2

End-to-end training of object class detectors for mean average precision

Object class detection is the task of localising all instances of a given set of object classes in an image. It is a natural first stage in a scene understanding pipeline, which may then perform tasks such as 3D shape reconstruction on each object separately. Given an input image, a detection algorithm outputs a set of bounding-boxes, each labelled with a class; the goal is that every object in the scene is tightly enclosed by exactly one bounding-box, correctly labelled with its class (as in Figure 1.1).

Aside from being an important part of a holistic scene understanding pipeline, object detection is also valuable in its own right. Self-driving cars must be able to detect other vehicles and pedestrians; assistive robots must be able to detect household objects. Image search engines must be able to identify what objects are present in an image, as must systems tailored to more sophisticated tasks such as visual question answering. However, object detection in the general setting is rather challenging. There is great intra-class variation in object shape and appearance, and moreover, objects may appear in different poses, under different illumination, and partially occluded. Detection models must therefore be trained to be robust to such variations, and to extract the salient features defining instances of their target classes.

Many modern techniques for object detection (*e.g.* Girshick, 2015; Zhang et al., 2015; Sermanet et al., 2014; Girshick et al., 2014) use a convolutional neural network (CNN) classifier (Krizhevsky et al., 2012; Simonyan and Zisserman, 2015), operating on object proposal windows (Alexe et al., 2010; Uijlings et al., 2013; Zitnick and Dollár, 2014); see Section 2.1.1 for more detailed background. Given an image, they first generate a set of windows (*i.e.* bounding-boxes) likely to include all objects, then

apply a CNN classifier to each window independently. The CNN is trained to output one score for each possible object class on each window, and an additional one for 'background' or 'no object'. Such models are trained for window classification accuracy: the loss attempts to maximise the number of training windows for which the CNN gives the highest score to the correct class. At test time, the CNN is applied to every window in a test image, followed by a non-maximum suppression processing stage (NMS). This eliminates windows that are not locally the highest-scored for a class, yielding the output set of detections. Typically, the performance of the detector is evaluated using mean average precision (mAP) over classes (see Section 2.1.3), which is based on the ranking of detection scores for each class (Everingham et al., 2010).

Thus, the traditional approach is to train object detectors with one measure, classification accuracy over all windows, but test with another, mAP over locally-highest-scoring windows. While the training loss correlates somewhat with the test-time evaluation metric, they are not the same, and furthermore, training ignores the effects of NMS. As such, the traditional approach is not true end-to-end training for the final *detection* task, but for the surrogate task of *window classification*.

In this chapter, we present a method for training object detectors directly using mAP computed after NMS as the loss. This is in accordance with the machine learning dictum that the loss we minimise at training time should correspond as closely as possible to the evaluation metric used at test time. It also fits with the recent trend towards training models end-to-end for their ultimate task, in vision (Long et al., 2015; Vinyals et al., 2015; Pfister et al., 2015) and other areas (Sutskever et al., 2014; Levine et al., 2016), rather than training individual components for engineered sub-tasks, and combining them by hand.

Directly optimising for mAP following NMS is very challenging for two main reasons: (i) mAP depends on the global ordering of class scores for all windows across all images, and as such is piecewise constant with respect to the scores; and, (ii) NMS has highly non-local effects within an image, as changing one window score can have a cascading effect on the retention of many other windows. In short, we have a structured loss over many thousands of windows, that is non-convex, discontinuous, and piecewise constant with respect to its inputs. Our main contribution is to overcome these difficulties by proposing new gradient-like quantities for piecewise constant functions, and showing how these can be computed efficiently for mAP following NMS. This allows us to train a detector based on Fast R-CNN (Girshick, 2015) in a truly end-to-end

fashion using stochastic gradient descent, but with NMS included at training time, and mAP as the loss.

Experiments on the PASCAL VOC 2007 and 2012 detection datasets (Everingham et al., 2015) show that end-to-end training directly for mAP with NMS reaches equivalent performance to the traditional way of training for window classification accuracy and without NMS. It achieves this while being conceptually simpler and more appealing from a machine learning perspective, as exactly the same model and loss are used at both training and test time. Furthermore, our method is widely applicable on two levels: firstly, our loss is a simple drop-in layer that can be directly used in existing frameworks and models; secondly, our approach to defining gradient-like quantities of piecewise-constant functions is general and can be applied to other piecewise-constant losses and even internal layers. For example, using our method can enable training directly for other rank-based metrics used in information retrieval, such as discounted cumulative gain (Järvelin and Kekäläinen, 2000). Moreover, we do not require a potentially expensive max-oracle to find the most-violating inputs with respect to the model and loss, as required by prior works such as Yue et al. (2007) and Song et al. (2016).

Chapter overview. In Section 2.1, we give background on object detection models and evaluation, and prior works that train with structured losses or with NMS. Section 2.2 describes our proposed method. We present experimental results in Section 2.3, then conclude in Section 2.4.

The work described in this chapter was presented at ACCV 2016 (Henderson and Ferrari, 2016b).

2.1 Background and related work

In this section, we first give background on methods for object detection (Section 2.1.1). Then, we describe one particular detection method, Fast R-CNN (Girshick, 2015), in more detail (Section 2.1.2), as it forms the basis for our proposed method. We also describe the evaluation metric that is used for object detectors (Section 2.1.3), and discuss prior works that train with similar losses (Section 2.1.4).

2.1.1 Object class detection

Object class detection first emerged as a vision task in the late 1990s; prior to that, only the related task of detecting a specific object *instance* had been considered (*e.g.*

by Lowe (1987) and Rothwell et al. (1992)). Note that the literature on object class detection is vast, and here we only highlight some of the most prominent approaches. For a more comprehensive survey, see for example Zhang et al. (2013) and Wang (2016).

The most prominent early work on object class detection is that of Viola and Jones (2001); they presented a method for detecting faces, by extracting a large number of windows (*i.e.* rectangular regions) from an image, and classifying each. The classifier is based on AdaBoost (Freund and Schapire, 1997), which uses multiple fast, weak classifiers, applied in turn such that windows not containing faces are discarded without evaluating every classifier. In conjunction with features that are fast to extract, this produces a very efficient detector.

The same *sliding window* approach, applying a classifier to a grid of windows of different positions and sizes, dominated object detection for the next ten years. Dalal and Triggs (2005) presented a highly successful model for detecting humans, that extracts *histogram of oriented gradients* (HOG) features from each window, and classifies these using a support vector machine (SVM). Both the features and the classifier are more powerful than those used by Viola and Jones (2001); however, the detector has limited flexibility with respect to different poses and aspects. Felzenszwalb et al. (2010) mitigated these problems by treating objects as groups of parts, allowed to shift relative to each other—hence, their method is known as the *deformable part model*. Each part is still detected using an SVM over HOG features, but now an additional cost term evaluates whether the relative locations of the parts are plausible. For efficiency, only windows which score highly according to an initial, fast, single-part model are processed by the full model. For flexibility, the entire model is replicated in a mixture, to allow learning different aspects or styles for each class.

Another approach to window classification uses *bag-of-words* models. These discard spatial information inside the window, and instead classify occurrence histograms for a large set of quantised local features (Sivic et al., 2005; Harzallah et al., 2009; Prest et al., 2012; Uijlings et al., 2013). More precisely, local feature descriptors such as SIFT (Lowe, 2004a) are computed at numerous locations inside each window. These locations may be densely sampled on a grid (often at multiple scales), or may be produced by a keypoint detector (*e.g.* Harris and Stephens, 1988; Mikolajczyk and Schmid, 2001; Lowe, 2004a). Features sampled from the training set are vector-quantised, using k-means or more sophisticated techniques such as random forests, to produce a set of representative *visual words*. For each window, the features are mapped

to their nearest visual words, and an occurrence histogram computed; this histogram retains information on how often each visual word occurs in the window, but discards information about where. This provides invariance to deformations, at the expense of not being able to reason about the spatial or part structure of the object. The histogram of visual words for a window is then classified using an SVM, typically with a histogram-intersection kernel; some methods also add first- and second-order statistics to the representation (Perronnin et al., 2010; Jégou et al., 2010).

Sliding window methods must classify hundreds of thousands of windows, which are sampled on a regular grid over the entire image, and at different scales and aspect ratios. This in turn means that the model used to classify each window must be inexpensive, to make this many evaluations practical. To mitigate this, more recent methods evaluate only a subset of windows, produced by *object proposal* algorithms, such as those of Alexe et al. (2010), Uijlings et al. (2013), and Zitnick and Dollár (2014); see Hosang et al. (2016a) for a detailed comparison of different techniques. These proposal algorithms are designed to quickly produce a set of windows that is likely to contain all objects in the image, based on carefully-engineered features that can be evaluated efficiently. This enables more powerful window classifiers to be used; one successful example is the bag-of-words model of Uijlings et al. (2013), which uses features with over 36000 dimensions quantised to a codebook of over 4000 visual words.

In the last six years, following the breakthrough work of Krizhevsky et al. (2012) on ImageNet classification (Russakovsky et al., 2015), convolutional neural networks (CNNs) have been applied to numerous vision tasks, in many cases improving the state-of-the-art significantly (Girshick et al., 2014; Long et al., 2015; Vinyals et al., 2015; Toshev and Szegedy, 2014; Dong et al., 2014). Instead of operating on hand-engineered features such as HOG, these models directly learn layers of multi-channel linear filters from training data; these filters are interleaved with fixed non-linear functions to increase the representative power of the model (LeCun et al., 1998).

Girshick et al. (2014) and Sermanet et al. (2014) were the first to successfully apply CNNs to object detection. Sermanet et al. (2014) use a sliding-window approach, while Girshick et al. (2014) achieve better performance by adding the CNN to a more modern detection pipeline. They use the CNN to extract features from object proposal windows, replacing the hand-engineered features used in earlier methods. These features are then classified using a linear SVM, and an additional bounding-box regression model predicts an offset to be added to each proposal to align it better with the detected object instance. Subsequently, numerous works have been published enhancing these

methods, *e.g.* those of Girshick (2015), Ren et al. (2015), and He et al. (2016). We give a detailed description of the method proposed by Girshick (2015) in Section 2.1.2.

Regardless of whether CNNs or hand-engineered features are used, window classification generally results in multiple detections per object. To avoid this, a process of *non-maximum suppression* is applied to the classifier outputs; we describe this in more detail below, as it is central to our own work.

Finally, we note that in the years since our work was published, several powerful (and very fast) object detectors have been proposed that do not use window classification as their basis (Redmon et al., 2016; Liu et al., 2016; Redmon and Farhadi, 2017). Instead, they directly regress both the class and location of the object, typically relative to a small set of anchor boxes spaced regularly across the image. In Section 2.4.2, we discuss the possibility of applying the techniques developed in this chapter to these newer detection models.

Non-maximum suppression (NMS). Given a set of windows in an image, with scores for some object class, NMS removes those windows which are not locally the highest-scored, to yield a final set of detections. This is a vital stage in detectors based on window classification, as it avoids multiple detections of the same object instance when several windows overlap that object.

Specifically, all the windows are marked as retained or suppressed by the following procedure: first, the highest-scored window is marked as retained, and all those overlapping with it by more than some threshold intersection-over-union (IoU) are marked as suppressed; typical values for this threshold are 30% (*e.g.* Girshick, 2015) or 50% (*e.g.* Felzenszwalb et al., 2010). Then, the highest-scored window neither retained nor suppressed is marked as retained, and again all others sufficiently-overlapping are marked as suppressed. This process is repeated until all windows are marked as either retained or suppressed. The retained windows then constitute the final set of detections.

It should be emphasised that NMS is a fixed (and somewhat heuristic) processing stage, used at test time only, to give good results under the commonly-used evaluation metric of mAP (Section 2.1.3). Only a handful of works account for the effects of NMS during the training process; we discuss these in Section 2.1.4. Interestingly, Parikh and Zitnick (2011) show that replacing the NMS stage of a detection pipeline by a human can increase the overall accuracy of the detector. This is corroborated by Hosang et al. (2016a), who show that a perfect oracle-like NMS function would achieve significantly higher mAP on a standard dataset (PASCAL VOC 2007; see below) than the usual

heuristic does. Thus, there is scope for improving NMS in state-of-the-art detection pipelines.

Datasets. Object detectors have most often been evaluated on the PASCAL VOC datasets (Everingham et al., 2015), typically the 2007 and 2012 versions. These cover 20 commonly-occurring object classes, and include hundreds of images per class, separated into training and test sets. The 2007 version has a total of 9963 annotated images; the 2012 version has 11540, plus the test set which is not publicly available. There is a standard evaluation protocol, which uses mean average precision (mAP) as the metric (Section 2.1.3).

Russakovsky et al. (2015) present the larger ILSVRC detection dataset. This includes around 200 object classes, with a total of 80 000 fully-annotated images available, plus 40 000 more for testing. The images are of a similar complexity to PASCAL VOC, in terms of the number of object instances and degree of occlusion.

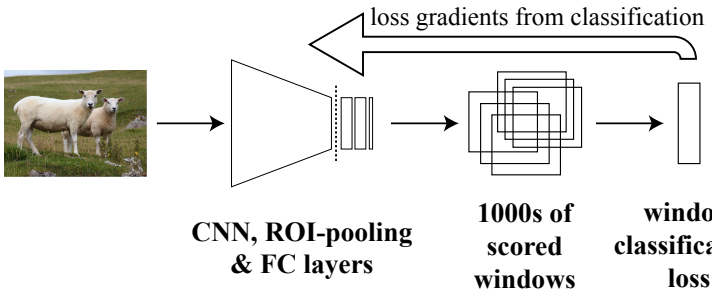
The COCO dataset of Lin et al. (2014) has become the most prominent benchmark for object detection in the last few years. This includes 80 object classes, with over 120 000 fully-annotated images and 40 000 more for testing, with a higher degree of occlusion and clutter than PASCAL VOC. Lin et al. (2014) also define several new evaluation metrics, most based on mAP, designed to evaluate different aspects of detection at a finer level than the single metric of PASCAL VOC.

2.1.2 Fast R-CNN

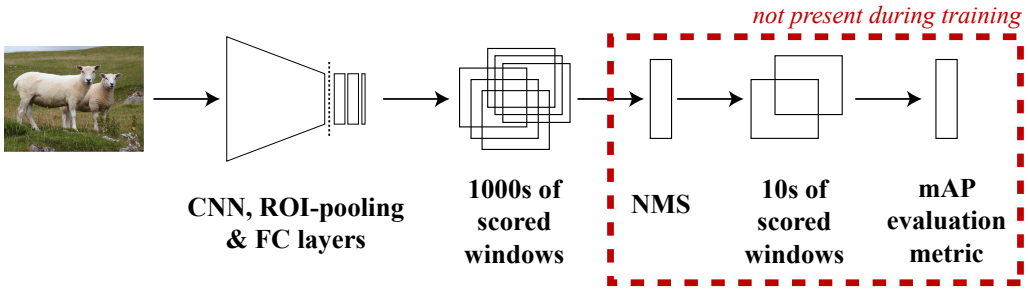
In this section, we describe one particular object detection method, Fast R-CNN (Girshick, 2015), in more detail. This was the most commonly used detector when the work in this chapter was published, and it forms the basis for our own model.

Model. The architecture of Fast R-CNN is illustrated in Figures 2.1a and 2.1b [1]. The model operates by using a CNN to classify proposal windows of an image, as belonging to one of a set of object classes, or as 'background'. Whole images are processed by a sequence of convolutional layers; then, for each window, convolutional features with spatial support corresponding to that window are extracted and resampled to fixed dimension, before being passed through three fully-connected layers, the last of which yields a score for each object class and 'background'. The class scores for each window are then passed through a softmax function, to yield a distribution over classes.

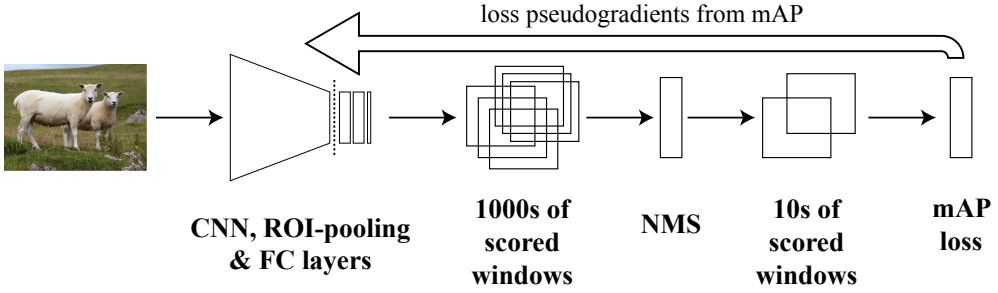
[1] note that we do not use bounding-box regression, described in Girshick (2015), in this chapter



(a) Fast R-CNN, training: scores passed directly to window classification loss



(b) Fast R-CNN, testing: NMS applied, and detections evaluated with mAP



(c) Our method, both training and testing: exactly the same operations occur at train and test time, with identical model structure and the training loss matching the test-time evaluation metric

Figure 2.1: Fast R-CNN (Girshick, 2015) architecture during training (a) and testing (b) phases, and our architecture (c), which is the same in both phases.

Training. The CNN is trained with a window classification loss. If a window overlaps a ground-truth object with IoU > 0.5, its true class is defined as being that object class; otherwise, its true class is 'background'. For each window, the network outputs softmax probabilities for each class, and the negative log likelihood (NLL) of the true class is used as the loss for that window; the total loss over a minibatch is simply a sum of the losses over all windows in it. The network is trained by stochastic gradient

descent (SGD) with momentum, operating on minibatches of two images at a time, with 64 proposal windows sampled from each.

Testing. At test time, all proposal windows for each image are scored by passing them forwards through the network, and recording the final softmax probabilities for each class. Then, for each class and image independently, NMS is applied to the scored windows (Section 2.1.1). Note that this NMS stage is not present at training time. Finally, the detections are evaluated using mAP over the full test set.

2.1.3 Mean Average Precision (mAP)

Object detectors are typically evaluated according to mean average precision, or mAP. The mAP for a set of detections is the mean over classes, of the interpolated AP (Salton and McGill, 1986) for each class (Everingham et al., 2015; Russakovsky et al., 2015; Everingham et al., 2010). This per-class AP is given by the area under the precision/recall (PR) curve for the detections (Figure 2.2).

The PR curve is constructed by first mapping each detection to its most-overlapping ground-truth object instance, if any overlaps sufficiently—for PASCAL VOC, this is defined as overlapping with $> 50\%$ IoU (Everingham et al., 2015). Then, the highest-scored detection mapped to each ground-truth instance is counted as a true-positive, and all other detections as false-positives. Next, we compute recall and precision values for increasingly large subsets of detections, starting with the highest-scored detection and adding the remainder in decreasing order of their score. Recall is defined as the ratio of true-positive detections to ground-truth instances, and precision as the ratio of true-positive detections to all detections. The PR curve is then given by plotting these recall-and-precision pairs as progressively lower-scored detections are included. Finally, dips in the curve are filled in (interpolated) by replacing each precision with the maximum of itself and all precisions occurring at higher recall levels (pink shading in Figure 2.2) (Everingham et al., 2010; Salton and McGill, 1986).

The area under the interpolated PR curve is the AP value for the class. For the PASCAL VOC 2007 dataset, this area is calculated by a rough quadrature approximation sampling at 11 uniformly spaced values of recall (Everingham et al., 2010); for the VOC 2012 dataset it is the true area under the curve (Everingham et al., 2015).

Note that the official evaluation protocol for the COCO dataset (Lin et al., 2014) is slightly different to that for PASCAL VOC. Instead of evaluating with a single, fixed IoU threshold of 50%, an average is taken over 10 different values, equally spaced from

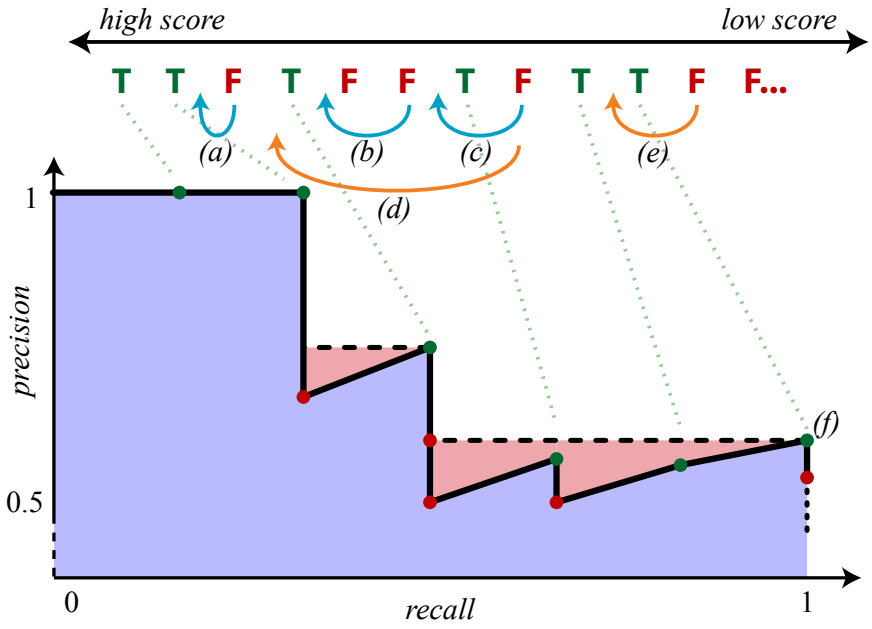


Figure 2.2: Precision/recall curve (bottom) for a sequence of true-positive (TP) and false-positive (FP) detections ordered by score (top) for some object class with six ground-truth instances. Plotting the sequence of precision and recall values yields the black curve. The pink area shows the result of replacing each precision with the maximum at same or higher recall. AP is the total area of the pink and blue regions. The arrows (a-e) show the effect of positive perturbations to scores of FP detections. Blue arrows (a-c) show perturbations with no effect on AP: (a) the order of detections does not change; (b) the detection swaps places with another FP; (c) the detection swaps places with a TP, but a higher-recall TP (f) has higher precision so there is no change to area under the filled-in curve (pink shading). Orange arrows (d-e) show perturbations that do affect AP: (d) the same FP as (c) is moved beyond a TP that does appear on (hence affect) the filled in curve; (e) the FP moves past a single TP, altering the filled-in curve as far away as 0.5 recall.

50% to 95%. For each IoU threshold, the calculation is the same as described above. The effect of this change is that methods are penalised more for localising instances poorly, as detections must be closer to the ground-truth box to count as correct at the higher IoU thresholds.

2.1.4 Training for structured losses or with NMS

Nearly all works on object class detection train a window classifier, and ignore NMS and mAP at training time. We now review works in other domains that train for similar rank-based losses, and the few on object detection that are similar in spirit to our own.

Optimising for rank-based losses. Several works under the umbrella of *listwise learning to rank algorithms*, such as those of Cao et al. (2007) and Xia et al. (2008), train for global rank-based losses. However, these methods are concerned with optimising the ranking itself, rather than a particular evaluation metric such as AP. One notable example is the work of Taylor et al. (2008), which presents an approach that is similar in spirit to our own: they discuss a formulation for gradient-descent optimisation of such rank-based metrics (though not AP specifically). To achieve this, they define a smooth proxy loss for the non-differentiable, piecewise constant ranking metric. They treat the predicted score of each training point as a Gaussian random variable centered on the actual value, and hence compute the distribution of ranks for each score, by pairwise comparisons to all other scores. This distribution is used in place of the usual hard ranks when evaluating the loss, and the resulting quantity is differentiable with respect to the original scores. This method has cubic complexity in the number of training samples, making it intractable when there are tens of classes and thousands of windows (as in all standard object detection datasets). Xu and Li (2007) also present a method that aims to train for rank-based metrics; they do consider AP, among others. Their method is inspired by AdaBoost (Freund and Schapire, 1997), and iteratively builds an ensemble of simpler ('weak') ranking models. At every iteration, a weak ranking model is added to the ensemble, chosen such that it reduces training error on the overall ranking problem as much as possible. Each of these weak models can be trained efficiently as it has only a very small number of parameters.

Other works train specifically for AP, but for binary classification problems, rather than for multi-class object detection with NMS. Yue et al. (2007) optimise AP in the structured SVM framework (Tsochantaridis et al., 2005); this uses a linear model, trained with a hinge loss weighted according to AP. This requires solving a loss-

augmented inference problem, *i.e.* finding the scores that maximise the sum of AP and the output of the current model. They present a dynamic programming algorithm to solve this, which has quadratic complexity in the number of training points.

Applications to object detection. Blaschko and Lampert (2008) formulate object detection as a structured prediction problem, producing a binary indicator for object presence and a set of bounding-box coordinates. This is trained using a structured SVM (Tsochantaridis et al., 2005), with a task loss that aims for correct classification and maximal IoU of predicted and ground-truth boxes in images containing the target class. Like our method, this is a structured loss involving IoU of detections and ground-truth objects; however, it does not correspond to maximising AP, and only a single detection is returned in each image, so there is no NMS. More recently, Zhang et al. (2015) use the same structured SVM loss, but with a CNN in place of the kernelised linear model over SURF features used by Blaschko and Lampert (2008). This work directly optimises the structured SVM loss via gradient descent, allowing backpropagation to update the nonlinear CNN layers.

Mohapatra and Jawahar (2014) extend the work of Yue et al. (2007), and apply it to object detection. They present various techniques that when combined, yield a significantly faster algorithm than that of Yue et al. (2007), still solving the same structured SVM problem with AP weighting the hinge loss. Their method is demonstrated for object detection, but just using single-class AP, and without NMS at training time. Further extending both these works, Song et al. (2016) generalise this technique to allow training *nonlinear* structured models directly for non-differentiable losses, again assuming that loss-augmented inference can be performed efficiently. Using a similar dynamic-programming approach to Yue et al. (2007) and Mohapatra and Jawahar (2014), they again apply it to object detection, using a model based on R-CNN (Girshick et al., 2014). As with the preceding works, this only trains for single-class AP, without NMS at training time.

Unlike most other approaches to object detection, Wan et al. (2015) include NMS at training time as well as test time. They use a deformable parts model over CNN features, that outputs scored windows derived from a continuous response map; this contrasts with the more common approach of feeding fixed proposal windows through a CNN (*e.g.* Girshick, 2015). The scored windows are passed through a non-standard variant of NMS, specifically designed to allow straightforward application during training. Instead of training for mAP or window classification accuracy, the authors then introduce a new structured loss. This includes terms for detections retained by NMS,

but also for suppressed windows, in a fashion requiring knowledge of which detection suppressed them. As such, it is deeply tied to the NMS implementation at training time, rather than being a generally-applicable loss such as mAP.

Works published after our own. Hosang et al. (2016b) again include a non-standard version of NMS at training time. This is a CNN-based model that locally combines the results of evaluating the standard NMS function at different IoU thresholds. They train this using a loss designed to encourage there to be exactly one detection produced per ground-truth object. Hosang et al. (2017) train for the same loss, but using a more flexible CNN for the NMS stage, that can rescore windows arbitrarily based on the scores of neighbouring windows.

Stewart et al. (2016) train a model that uses a recurrent neural network (RNN) for its final output stage, allowing the number of boxes to vary depending on the image content. It is trained to produce exactly one detection per ground-truth object, so NMS is not required. The loss function for a given set of bounding boxes is defined by finding an optimal mapping from these to their nearest ground-truth boxes, accounting for class, location and score ranking; in this respect, it is similar to mAP. However, mAP uses greedy (not optimal) matching of predicted and ground-truth boxes, and Stewart et al. (2016) use a combination of softmax classification and box-location losses on the matched boxes, rather than average precision.

Rao et al. (2018) use a method inspired by the policy-gradient estimator (Williams, 1992) to fine-tune a detection model for mAP with NMS present. They retain the normal window classification loss, and after training with this, fine-tune the model using a weighted combination of cross-entropy and mAP gradients. The detection problem is cast in the policy-gradient framework by treating mAP as the reward, the CNN as the policy, and the class scores as actions. This approach improves mAP on the COCO dataset by around 2%.

2.2 Proposed Method

We now describe our proposed method, which is illustrated in Figure 2.1c. We discuss how our model differs from Fast R-CNN (Section 2.2.1) and why it is challenging to train (Section 2.2.2). Then we introduce our general method for defining gradients of piecewise-constant functions (Section 2.2.3) and describe how we apply it to train our model (Section 2.2.4).

2.2.1 Detection Framework

Model. Our model is identical to Fast R-CNN (Girshick, 2015), up to the softmax layer: windows are still scored by passing through a sequence of convolutional and fully-connected layers. As in Girshick (2015), we can use different convolutional network architectures pretrained for ILSVRC 2012 (Russakovsky et al., 2015) classification, such as AlexNet (Krizhevsky et al., 2012) or VGG16 (Simonyan and Zisserman, 2015). We omit the softmax layer, using the activations of the last fully-connected layer directly as window scores. In our experiment we found that the softmax has little effect on the final performance, but its tendency to saturate causes problems with propagating the loss gradients back through it. In contrast to Fast R-CNN, our model also includes an NMS layer immediately after the last fully-connected layer, which performs the same operation as used at test time for Fast R-CNN. We regard the NMS layer as part of the model itself, present at both training and test time.

Training. During training, we add a loss layer that computes mAP over the minibatch, after NMS. Thus, at training time, minibatches undergo exactly the same sequence of operations as at test time, and the training loss matches the test-time evaluation metric. The network is still trained using SGD with momentum. Section 2.2.2 describes how to define derivatives of the mAP and NMS layers, while Section 2.2.5 discusses some minor changes we make to the training process.

Testing. During testing, our method is identical to Fast R-CNN, except that the softmax layer is omitted.

2.2.2 Gradients of mAP and NMS Layers

In order to minimise our loss by gradient descent, we need to propagate derivatives back to the fully-connected layers of the CNN and beyond. However, mAP is a piecewise constant function of the detection scores, as it depends only on their ordering—each score can be perturbed slightly without changing the loss. The partial derivatives of such a loss function do not convey useful information for gradient descent (Figure 2.3a) as they are almost everywhere zero (in the constant regions), and otherwise undefined (at the steps). The subgradient is also undefined, as the function is non-convex.

Furthermore, even if we could compute the derivatives of mAP with respect to the class scores, they still need to be propagated back through the NMS layer. This

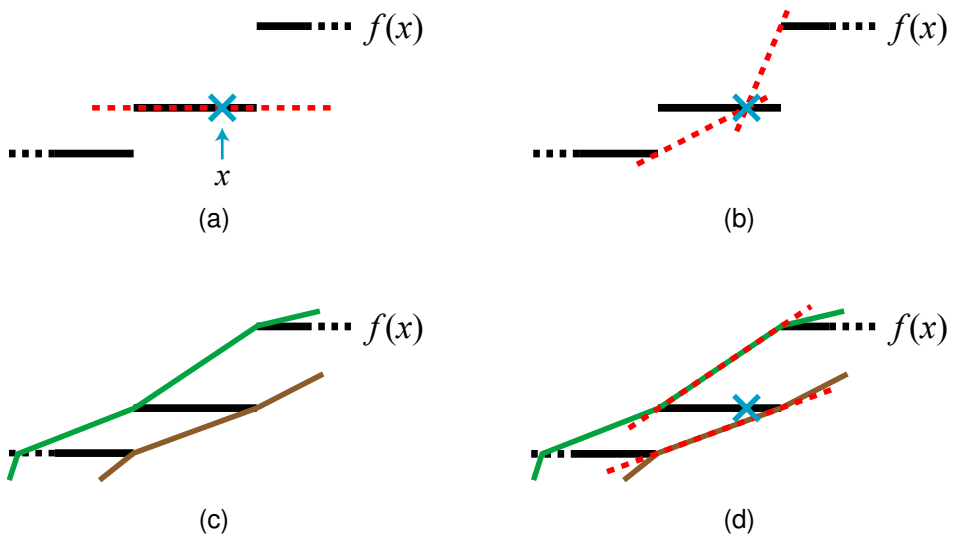


Figure 2.3: A piecewise constant function $f(x)$ with steps at two points, and various definitions for gradients. (a) Conventional partial derivative (red dashed) at x, equal to zero, does not convey useful information for gradient descent. (b) Gradients at x given by positive-perturbing and negative-perturbing finite difference estimators. (c) Piecewise-linear upper (green) and lower (brown) envelopes of $f(x)$. (d) Gradients at x given by slope of upper/lower envelopes. When applied to our model, $f(x)$ is mAP, and the horizontal axis corresponds to the score of a single window with respect to which the partial derivative is being computed.

requires a definition of the Jacobian of NMS, which is again non-trivial. Note that max-pooling layers are similarly non-differentiable, but good results are achieved by simply propagating the gradient back to the maximal input only. We could do similar for NMS: allow only the locally-maximal windows propagate gradients back; however, this loses valuable information. For example, if all detections overlapping some ground-truth object are suppressed, then there should be a gradient signal favouring increasing the score of those windows (or decreasing that of their suppressors). This does not occur if we naïvely copy gradients back through to maximal windows. In contrast, we require a Jacobian-like quantity for NMS that does capture this information.

We therefore develop general definitions for gradient-like quantities of piecewise-constant functions in Section 2.2.3, and then describe how to apply them efficiently to NMS and mAP in Section 2.2.4.

2.2.3 Pseudogradients of General Piecewise-Constant Functions

We consider how to define a general *pseudo partial derivative* (PPD) operation for piecewise-constant functions, that can be used to define quantities analogous to the gradient and the Jacobian. For any piecewise-constant function $f(\mathbf{x})$ with countably many discontinuities (steps), we denote the PPD with respect to x_i by $\widetilde{\partial}_{x_i} f$. Even when the PPD is non-zero we need to move some non-infinitesimal distance before any change in the function occurs (unlike a conventional partial derivative). However when there is a change, it will be in the direction indicated by the PPD, and in magnitude corresponding to the PPD (this is made more precise below). We then use our PPD to define an analogue to the gradient by $\widetilde{\nabla} f = (\widetilde{\partial}_{x_1} f, \ldots, \widetilde{\partial}_{x_N} f)$. Intuitively, this tells us locally what direction to move so that the function will decrease, if we move some non-infinitesimal distance in this direction. Similarly, for the Jacobian of vector-valued $\mathbf{f}$, we have $\widetilde{J}_{ij} = \widetilde{\partial}_{x_j} f_i$.

We now discuss two possible definitions for the PPD; these and the regular partial derivative are illustrated in Figure 2.3 for a one-dimensional function, at a point lying in a constant region between two steps.

Finite difference estimators. Most simply, we can apply a traditional single-sided finite difference estimator, as used for computing numerical gradients of a differentiable function. Here, a small, fixed perturbation δx is added to x, the function evaluated at this point, and the resulting slope used to approximate the gradient, by $\widetilde{\partial}_x f = \frac{f(x+\delta x) - f(x)}{\delta x}$. The piecewise-constant functions we are interested in have finitely many steps, and so the probability of f being undefined at the perturbed point is zero. However, the constant regions of our function vary in size by several orders of magnitude, and so it is impossible to pre-select a suitable value for δx. Instead, we use an adaptive approach: given x, set δx to the smallest value such that $f(x+\delta x) \neq f(x)$, then compute $\widetilde{\partial}_x f$ as above (Figure 2.3b). Note that this method is single-sided: it only takes account of the change due to perturbing x in one direction or the other. This is undesirable, as in general, it delivers different results for each direction, perhaps yielding complementary information. We address this issue by performing the same calculation independently with positive then negative perturbations δx^+ and δx^-, and taking a mean of the resulting pseudogradients. We refer to this mean pseudogradient as SDE, for symmetric difference estimator. This approach has the disadvantage that the magnitude of the gradient is sensitive to the exact location of x: if it is nearer to a step, the gradient will be larger, yet a correspondingly larger change to the network

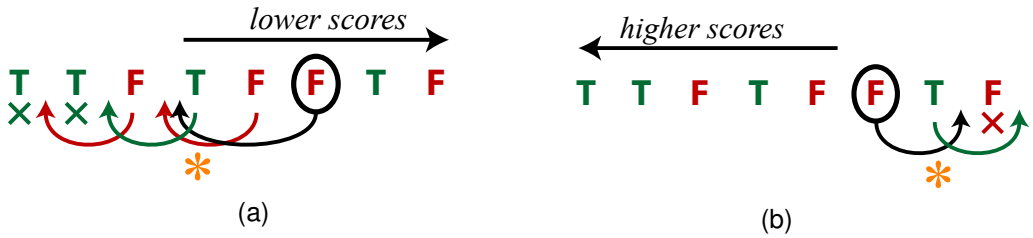
parameters may be undesirable.

Linear envelope estimators. An alternative approach to defining the PPD is to fit a piecewise-linear upper or lower envelope to the steps of the piecewise-constant function (Figure 2.3c). The PPD $\widetilde{\partial}_x f$ is then given by the slope of the envelope segment at the point x (Figure 2.3d). In practice, we take the average of the gradients of the upper and lower envelopes. Unlike SDE, this estimator does not become arbitrarily large as x approaches a step. If the values of x corresponding to steps are bounded, then for all points before the first step and after the last, both linear envelopes have zero gradient; we find that better results are achieved by using SDE in these regions, but with an empirically-tuned lower-bound on δx. We refer to this pseudogradient as MEE, for mean envelope estimator.

2.2.4 Application to mAP and NMS

To apply the above methods to mAP, we must compute the PPD of each class' AP with respect to each window score independently, holding the other scores constant. This raises two questions: (i) how to efficiently find the locations of the nearest step before and after a point, and (ii) how to efficiently evaluate the loss around those locations. We solve these problems by noting that changes to AP only occur when two scores change their relative ordering, and even then, only in certain cases. Specifically, AP changes value only when a window counted as a true-positive changes place with one counted as a false-positive. Also, the effective precision at a given recall is the maximum precision at that or any higher recall (Section 2.1.3 and Figure 2.2). So we have further conditions, *e.g.* decreasing the score of a false-positive only affects AP when it drops below that of a true-positive at which precision is higher than any with even lower score. This effect and other examples of perturbations are illustrated in Figure 2.2 (blue and orange arrows).

Thus, for each class, we can find the nearest step before and after each point by making two linear passes over the detections, in descending then ascending order of score (Figure 2.4). Assuming we have computed AP as described in Section 2.1.3, we know whether each detection is a true- or false-positive, and can keep track of the last-seen detection of each kind. In the descending pass, for each detection, we find the smallest increase to its score that would result in a change to AP, thus giving the location of the nearest step on the positive side. This score increase is that which moves it an infinitesimal amount higher than the score of the last-seen window of the other



(a) (b)

Figure 2.4: Efficient calculation of smallest perturbations to detection scores to cause a step in AP. In each case the circled FP is currently being considered. (a) Iterating detections in decreasing order of score, finding the smallest increase to each score that causes a change in AP (higher for TPs, lower for FPs). Detections already considered have an arrow showing where they are perturbed to; a cross indicates no increase to that score affects AP. When considering the circled FP, the last-seen TP is shown by the orange asterisk; perturbing the score of the circled detection just beyond (left) of this is the minimal change to affect AP. (b) Similar but iterating in increasing order of score, and hence calculating minimal decreases in score to affect AP.

kind (true-positive vs. false-positive), subject to the additional conditions mentioned above. Similarly, in the ascending pass, we can find the required decreases in scores that would cause a change in AP. Once the step locations have been found, the new AP values resulting from perturbing the scores accordingly can be calculated by updating the relevant part of the PR curve, and then computing its area as normal. Given the step locations and AP values, it is then straightforward to use the methods of Section 2.2.3 to compute the SDE or MEE.

Incorporating NMS. We must also account for NMS when propagating gradients back. The PPDs of NMS can be used to define a Jacobian as described in Section 2.2.3, which may then be composed with the pseudogradient of mAP to define the gradient of mAP with respect to the pre-NMS scores. However, subject to a small approximation, it is both easier and more efficient to consider NMS simultaneously with AP when determining step locations and the resultant changes to the loss. Specifically, we introduce two transitivity approximations (Figure 2.5): (i) we do not attempt to model cascaded long-distance interactions between windows through multiple steps of NMS; (ii) we assume in certain cases that windows suppressed by some detection overlap exactly the same ground-truth instances as the detection itself. Under these approximations, it is possible to compute the PPDs with respect to pre-NMS scores in linear time in the number of windows. This is achieved by: (i) adding gradient contributions due

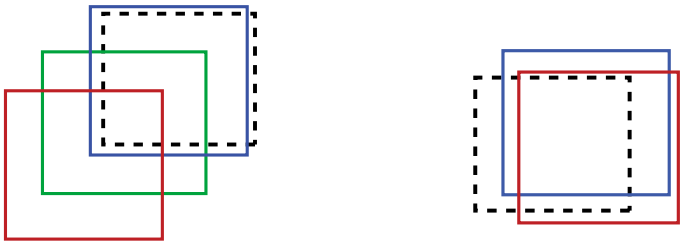


Figure 2.5: Transitivity approximations for NMS. Dashed black box is a ground-truth object; coloured boxes are scored windows, red $>$ green $>$ blue. (a) Red overlaps green sufficiently for NMS inhibition, and green overlaps blue similarly, but red does not overlap enough with blue. However, whether red is retained indirectly affects whether blue is retained, as if red suppresses green, then green does not suppress blue. In our approximation, this long-distance interaction between red and blue is ignored; however the two local interactions (red-green and green-blue) are included. (b) Red and blue overlap each other sufficiently for NMS inhibition; given that red suppresses blue, our approximation assumes that blue overlaps the same ground-truth instance as red (if any).

to windows suppressed by a true-positive or false-positive detection at the same time as that detection, as these suppressed windows need to have their scores perturbed to the same point as their suppressor did to cause a change in AP; (ii) including a third pass that adds gradient contributions from suppressed windows overlapping ground-truth instances that were missed entirely (*i.e.* no detection covers them); (iii) also adding gradient contributions from the detections that caused the suppressed-but-overlapping windows of (ii) to be suppressed.

2.2.5 Training Protocol

In order to train our model successfully, we make various changes to the training protocol used by Girshick (2015) for Fast R-CNN. The impact of each of these changes is given in Section 2.3.

Minibatch composition. We use larger minibatches than Girshick (2015), as (i) object detection mAP has a much higher batch-to-batch variance than simple window classification accuracy, and (ii) including more windows increases the density of the gradient signal, as there are likely to be more false positives which score higher than

some true positive (and vice versa). We also find that performance is improved by using proportionally fewer foreground windows (those overlapping a ground-truth instance as opposed to background) in each training minibatch. While Fast R-CNN uses 25% foreground windows, we use 5%, which roughly corresponds to the distribution of windows seen at test time, when 5% of all selective search proposals overlap a ground-truth instance.

Regularisation. Using our method, we found empirically that scores are prone to grow very large after several hundred iterations of training. This is effectively mitigated by introducing a regulariser on the window scores. We find that an L4 regulariser with very small weight performs best, as it gives greater freedom to smaller-magnitude scores while imposing a relatively hard constraint on magnitude, compared to the more common L1 or L2 regularisation.

Log-space. We find it is beneficial to follow gradients of $\log(\text{mAP} + \varepsilon)$ instead of mAP itself, for some small, fixed constant ε. Early in training when mAP is low, scores of true-positive windows are uniformly distributed amongst those of false-positive windows, and so an increase in the score of a true-positive often yields only a very small gain in mAP. Using $\log(\text{mAP} + \varepsilon)$ instead amplifies the effect of these changes, so training quickly escapes from the initial very low mAP.

Gradient clipping. We find that numerical behaviour is improved (particularly at high learning rates) by clipping elements of the gradient to a fixed threshold.

2.3 Experiments

We now evaluate the performance of our approach on two datasets: PASCAL VOC 2007 and 2012 (Everingham et al., 2015). Both datasets have 20 object classes; for VOC 2007, we train on the trainval subset (5011 images) and test on the test subset (4952 images); for VOC 2012, we train on the train subset (5717 images) and test on the validation subset (5823 images). We also give results training on the union of VOC 2007 trainval and VOC 2012 trainval (total 16551 images), and testing on VOC 2007 test.

We compare our method to two others: (i) Fast R-CNN trained with the standard NLL loss for window classification, as described by Girshick (2015) (bounding box regression is disabled, to give a fair comparison with our method); and (ii) that of Song et al. (2016), which also trains an R-CNN-like model for AP, but with a separate

trained on...	2007 only		2007 + 2012	
	AlexNet	*VGG16*	*AlexNet*	*VGG16*
Ours, MEE	51.6	58.9	54.9	62.5
Ours, SDE	51.3	60.7	54.8	62.3
Fast R-CNN	52.0	62.4	53.8	63.5

Table 2.1: Performance of our method measured by mAP on VOC 2007 test set, with different pseudogradients (MEE vs SDE), network architectures (AlexNet vs VGG16), and training sets (VOC 2007 trainval vs union of VOC 2007 trainval and VOC 2012 trainval). We also give results for Fast R-CNN trained using a traditional softmax loss, without bounding box regression.

model for each class, no NMS at training time, and with a different way to compute parameter gradients. This is the closest work in spirit to ours.

Settings. We use Fast R-CNN as described by Girshick (2015), with AlexNet (Krizhevsky et al., 2012) or VGG16 (Simonyan and Zisserman, 2015) as the underlying CNN. The network weights are initialised by pretraining on ILSVRC 2012 classification (Russakovsky et al., 2015). We remove the softmax layers at both training and test time, as described in Section 2.2.1, and replace the training loss layer with our NMS layer and mAP loss.

Incorporating the techniques described in Section 2.2.5, the overall loss we minimise by SGD is

$$L = -\log\left\{\sum_c \mathrm{AP}\left(\mathrm{NMS}(\mathbf{s}_c)\right)/K + \varepsilon\right\} + \lambda \sum_{c,b} |s_c^b|^4 \tag{2.1}$$

where $\mathbf{s}_c$ are the window scores for class c, K is the total number of classes, and b indexes over windows.

The AP calculation during training is always matched to that used for evaluation. When testing on VOC 2007, we train using the VOC 2007 approximation to AP (Section 2.1.3); when testing on VOC 2012, we train using the true AP. In order to compute pseudogradients for training, we try both SDE and MEE and compare their performance (Section 2.2.3). As our method works best with large minibatches, for the VGG16 experiments, we clamp the maximum image dimension to 600 pixels, to conserve GPU memory (this does not have a significant impact on the baseline performance).

Main results on VOC 2007. Table 2.1 shows how our methods compare with Fast R-CNN, testing on the PASCAL VOC 2007 dataset. Overall, our method achieves comparable performance to Fast R-CNN. The results also show that using a larger training set (union of VOC 2007 and 2012 trainval subsets) increases performance by up to 3.6% mAP, compared to training from VOC 2007 trainval alone. This effect is significantly stronger for our method than for Fast R-CNN: for AlexNet, we gain 3.3% mAP compared with 1.8% for Fast R-CNN; for VGG16, we gain 3.6% compared with 1.1% for Fast R-CNN. This indicates that our approach particularly benefits from more training data, possibly because optimising for mAP implies many comparisons between windows. Of our two pseudogradient estimators, MEE slightly outperforms SDE, in all cases apart from VGG16 training on VOC 2007 trainval only. This is likely because MEE is insensitive to the distances from points to nearest steps, in contrast to SDE (Section 2.2.3); hence, MEE is a more robust estimator of the impact of a score change, whereas SDE may introduce very large derivatives for a particular window. In all cases, VGG16 significantly outperforms AlexNet, confirming previous studies (Simonyan and Zisserman, 2015; Girshick, 2015).

Ablation study. In Section 2.2.5, we noted that certain modifications to the original training procedure of Fast R-CNN were necessary to achieve these results. Ablating away these modifications reduces our mAP, as follows (all using AlexNet on VOC 2007 with the MEE gradient estimator):

1. minibatch composition: increasing foreground fraction to 25% (as used in Fast R-CNN): -6.1 mAP

2. minibatch size: halving batch size but doubling iteration count (so the same amount of data is seen): -0.8 mAP

3. score regularisation: with L2 regularisation instead of L4 and the constant adjusted appropriately: -1.0 mAP. With no regularisation, training fails after fewer than 100 iterations as the magnitudes of the classification scores explode.

4. gradient clipping: with this disabled, training fails after fewer than 100 iterations due to numerical issues caused by large gradients.

Comparison to Song et al. (2016) on VOC 2012. The only previous work that attempts to train a CNN-based object detector directly for AP is Song et al. (2016). Table 2.2 compares their method to ours; we use the PASCAL VOC 2012 dataset

Ours, MEE	Ours, SDE	Song et al. (2016)
48.2	48.0	48.5

Table 2.2: Performance of our method compared with Song et al. (2016) (who train for single-class AP, with a technique very different from ours). All models were trained on VOC 2012 train subset, tested on VOC 2012 validation subset, and use AlexNet. Bounding box regression was not used in any of the models.

(testing on the validation subset) as this is what Song et al. (2016) report results on. Our method achieves comparable performance to theirs, with the MEE estimator again being slightly better than SDE. Note that unlike us, Song et al. (2016) train a separate model for each class; their dynamic-programming solution to the loss-augmented inference problem is for single-class AP only (not mAP over all classes).

2.4 Discussion and conclusions

In this section, we discuss various aspects of our method, then present some options for future research.

2.4.1 Analysis and discussion of our method

Benefits of end-to-end training. We have noted that our approach of training end-to-end with the same model and metric as used at test time is more principled and theoretically sound than using a surrogate loss. One practical advantage of this is that it relieves designers of detection models from having to consider the effect of modifications to the model on the surrogate loss, as well as on the final metric. Indeed, fewer engineering decisions are required overall, than if a surrogate loss and suitable training protocol for this must be designed.

Performance vs. Fast R-CNN. As noted in Section 2.3, our methods do not significantly out-perform Fast R-CNN overall. We hypothesise that this is due to three reasons:

1. Our gradients are sparser than those of a softmax loss: not every window propagates information back for every class, as changing scores of certain windows has no effect on mAP (*e.g.* low-scored background windows suppressed by

NMS). For example, for VOC 2007, around 20% of scores have a non-zero gradient — compared with 100% when using a softmax loss.

2. mAP is a more rapidly changing function than the softmax loss: an estimate over a minibatch is a much higher-variance estimator of loss over the full set.

3. It can be shown numerically that mAP over a minibatch of images is a biased estimator of mAP over the population of images from which that minibatch was drawn.

The real advantage of our method over the standard training procedure of Fast R-CNN is not performance, but being more principled by respecting the theoretical need for having the same evaluation during training and testing. It would be possible to combine the two paradigms, *i.e.* use both window-classification and mAP losses, as in the very recent work of Rao et al. (2018). This would require some care, as the mAP loss behaves better with a different strategy for sampling minibatches (Section 2.2.5), but this could likely be mitigated by re-weighting the window-classification loss.

Simplicity vs. Song et al. (2016). The method of Song et al. (2016) requires substantial modifications to the optimisation procedure itself, while our method does not. Instead, we simply define a new loss layer that can be easily dropped into existing frameworks, and do not require solving a loss-augmented inference problem. Furthermore, our approach can incorporate NMS and train simultaneously for multiple classes. Thus, while Song et al. (2016) train for AP over binary window classification scores, we train directly for mAP over object detections.

Applicability to other domains. Our approach to defining gradient-like quantities of piecewise-constant functions is not specific to mAP or object detection, and opens up the possibility of training for other piecewise-constant losses. For example, ranking-based metrics are common in information retrieval, including simple AP on document scores, and discounted cumulative gain (Järvelin and Kekäläinen, 2000). Our method enables training neural networks directly for these metrics. Moreover, we do not require a potentially expensive max-oracle to find the most-violating inputs with respect to the model and loss, in contrast to the method of Song et al. (2016) and the structured SVM methods of Yue et al. (2007) and Zhang et al. (2015).

In fact, our approach could also be applied to piecewise-constant internal layers of a network, not just to losses. It allows back-propagation of approximate gradients through such layers, by defining a pseudo-Jacobian.

2.4.2 Future work

Extension to support bounding-box regression. As noted in Section 2.1.2, we do not use bounding-box regression in any of our experiments. This is an additional output of Fast R-CNN, described by Girshick (2015), that predicts an offset to be applied to the edges of each proposal window to make them better match the detected object. To incorporate this in our approach, it would be necessary to calculate the pseudogradient of mAP with respect to the box locations, not just the scores as we currently do.

However, this is more challenging than for scores. If moving a box is to cause a change to mAP, that movement must change which ground-truth objects it overlaps; this means that the transitivity assumptions of Section 2.2.4 are no longer valid. Moreover, to find steps in mAP with respect to position, we have to consider what changes to the edges of a bounding-box actually change the assignment of detections to ground-truth windows.

One possible approach would be to evaluate *all* choices of perturbation to the score and position of every box. This is a finite set, as we only consider values which cause a step in mAP: scores are perturbed to just beyond every other score, and bounding-box-edges are perturbed to just beyond every point at which they cause an IoU to become greater or lesser than the threshold. While computationally expensive, rough calculations indicate that this is tractable, if the NMS and mAP calculations are run on the GPU, rather than the CPU as currently.

Application to other detection models and datasets. Our loss is a simple drop-in layer that can be directly used in other frameworks and models. Hence, it should be possible to apply it to recent object detectors other than Fast R-CNN, such as its direct extensions (Ren et al., 2015; He et al., 2016, 2017), and single-shot methods (Redmon et al., 2016; Redmon and Farhadi, 2017; Liu et al., 2016). In both cases, similar adaptations are required to those noted above for bounding-box regression. This is because these methods do not use pre-computed object proposals, but instead predict bounding-box locations within the model itself by regressing offsets relative to a fixed set of anchors.

Recently, the COCO dataset (Lin et al., 2014) has become the standard benchmark for object detection, replacing PASCAL VOC (Everingham et al., 2015). This uses a slightly different evaluation metric: rather than simply evaluating mAP at a fixed IoU threshold of 0.5, an average is taken over multiple different IoU thresholds. It would be straightforward to adapt our loss layer to incorporate this, by recomputing

the assignments of detections to ground-truth windows for each IoU threshold.

Application to semantic segmentation. Semantic segmentation is the task of labelling each pixel in an image with a semantic class such as 'road' or 'sky'. One common evaluation metric for semantic segmentation is mean intersection-over-union (mIoU); this is computed per class, and depends on the discrete class label assigned to each pixel (Long et al., 2015; Caesar et al., 2018). As such, it is non-differentiable, hence challenging to train for. Surrogate losses of pixel- or region-classification are typically used instead (Pohlen et al., 2017; Caesar et al., 2016), analogously to window-classification in object detection. Several recent methods have attempted to replace these surrogate losses with mIoU (Rahman and Wang, 2016; Máttyus et al., 2017; Berman et al., 2018); our pseudogradient approach would be a possible alternative to these.

2.4.3 Concluding remarks

We have presented two definitions of pseudo partial derivatives of piecewise-constant functions. Using these, we have trained a Fast R-CNN detector directly using mAP as the loss, with identical model structure at training and test time, including NMS during training. This ensures that training is truly end-to-end for the final detection task, as opposed to window classification. Our method achieves equivalent performance to Fast R-CNN. It is easily integrated with standard frameworks for SGD, such as Caffe (Jia, 2013), as our NMS and mAP loss layers can be dropped in without affecting the minimisation algorithm or other elements of the model. We have suggested how in future research, this method could be applied to more modern object detectors, or to entirely different domains.

Chapter 3

Learning to generate and reconstruct 3D meshes with only 2D supervision

In this chapter, we present a unified framework tackling two problems in 3D scene understanding: class-specific 3D reconstruction from a single image, and generation of new 3D shape samples. Single-image reconstruction is the task of predicting the full 3D shape of an object instance, given just one image of it; shape generation is the related task of probabilistically sampling a 3D shape for a given object class.

Single-image reconstruction is an important problem, with various applications in robotics and computer graphics. In robotics, it allows planning of grasping actions and motion, which naturally require knowing the 3D structure of the scene to avoid collisions. In computer graphics, it allows synthesising novel views of scenes, either for interactive content-creation, or for applications such as converting 2D images to 3D for stereo or virtual-reality displays. In both cases, being able to reconstruct scenes using only single 2D images greatly increases applicability, as it avoids the need for expensive depth sensors, multiple cameras, or capturing video data. However, single-image reconstruction is challenging, as it is inherently an ambiguous task. There are infinitely many 3D scenes that project to give the same 2D image, and a reconstruction algorithm must be able to make a physically-plausible choice, by exploiting prior knowledge.

Learning generative models of 3D shapes has traditionally received less attention than reconstruction. In computer vision, a generative model can act as a prior for other tasks, guiding them towards reasonable solutions; we apply this principle in this chapter, where a generative model guides reconstruction. Moreover, and perhaps more importantly, humans are able to learn the space of valid shapes for different types of

object just by observing them; as such, showing that this can be learnt by a machine is scientifically valuable. Finally, in computer graphics, generative models allow synthesis of diverse scenes without user interaction (Chapter 4 discusses this further in a slightly different setting).

Single-image reconstruction and generation have received considerable attention recently, leveraging new techniques based on deep learning; we review these in Section 3.1. Most learning-based methods for both tasks rely on strong supervision. For generation, this means large collections of manually constructed 3D shapes (see Section 3.1.5). For reconstruction, it means 3D shapes manually aligned with images, keypoint or pose annotations on images, and/or training with multiple views of each object instance (see Section 3.1.4). In contrast, our model does not require any of these strong forms of supervision: we focus on the more challenging setting where we only have access to unannotated 2D images for training. This was considered in just one earlier work (Gadelha et al., 2017).

It is well known that *shading* provides an important cue for 3D understanding (Horn, 1975). It allows determination of surface orientations, if the lighting and material characteristics are known; this has been explored in numerous works on shape-from-shading over the years (see Section 3.1.3). However, existing learning-based generation and reconstruction methods do not leverage shading information in their loss. Indeed, the vast majority use voxel grids as an output representation; these can only model axis-aligned surfaces, limiting the usefulness of shading cues. To exploit shading information in a learning-based approach, we therefore need to move beyond voxels; a natural choice of representation is then meshes. As we discuss in Section 3.1.6, these have desirable properties for our tasks: they can represent surfaces of arbitrary orientation and dimensions at fixed cost, and are able to capture fine details. Thus, they avoid the visually-displeasing 'blocky' reconstructions that result from voxels.

In this chapter, we present a single model that supports both reconstruction and generation of 3D shapes, that is trained with only 2D supervision, and models 3D meshes rather than voxels. Our framework is very general, and can be trained in similar settings to earlier models (Tulsiani et al., 2017b; Yan et al., 2016; Wiles and Zisserman, 2017), while also supporting weaker supervision scenarios. It allows:

- use of different **mesh parameterisations**, which lets us incorporate useful modeling priors such as smoothness or composition from primitives

- exploitation of **shading cues** due to monochromatic or coloured directional light-

ing, letting us discover concave structures that silhouette-based methods (Gadelha et al., 2017; Tulsiani et al., 2017b; Yan et al., 2016) cannot

- training with **varying degrees of supervision**: single or multiple views per instance, with or without ground-truth pose annotations.

To achieve this, we design a probabilistic generative model that captures the full image formation process, whereby the shape and pose of a 3D mesh are first sampled independently, then a 2D rendering is produced from these (see Section 3.2). We use stochastic gradient variational Bayes (Kingma and Welling, 2014; Rezende et al., 2014) for training (see Section 3.3). This involves learning an *inference network* that can predict 3D shape and pose from a single image, with the shape placed in a canonical frame of reference, *i.e.* disentangled from the pose. Thus, as a side-effect of learning reconstruction and generation, our model also learns to perform object pose-estimation (Xiang et al., 2014).

We evaluate our model on synthetic data in various settings, showing that (i) it learns to predict both 3D shape and pose, and to correctly disentangle them; (ii) unlike prior works, it can exploit shading cues; (iii) it is comparable or superior to state-of-the-art methods on quantitative metrics, while producing results that are visually more pleasing; (iv) it still performs well when given supervision weaker than supported by prior works.

Chapter overview. In Section 3.1, we give background on the reconstruction and generation tasks, and on methods used to solve them. Section 3.2 and Section 3.3 describe our model and how it is trained. We present results from our method in Section 3.5, and conclude in Section 3.6.

The work described in this chapter was presented at BMVC 2018 (Henderson and Ferrari, 2018).

3.1 Background and related work

This section presents background on the reconstruction and generation problems. First, we describe commonly-used datasets and evaluation methodologies. Then, for each task, we describe classical techniques, and how new learning-based methods have become prevalent. Finally, we give more details on the various representations of 3D shapes used by these techniques.

3.1.1 Datasets

In order to train and evaluate learning-based methods for reconstruction and generation, datasets of 3D shapes are required. To evaluate single-image reconstruction, these must also be paired with corresponding images. We now describe the most common datasets, and how methods are evaluated on them.

The ShapeNet Core dataset (Chang et al., 2015) consists of around 51000 CAD models grouped in 55 object classes. The ModelNet dataset (Wu et al., 2015b) consists of 151 000 CAD models grouped in 660 object classes; however 40-class and 10-class subsets of covering common categories are typically used for evaluation.

The above datasets contain only 3D CAD models, though these may be rendered to produce matching images. Two other recent datasets go further, by pairing real photographs with corresponding CAD models, thus allowing evaluation of single-image 3D reconstruction methods on natural images. The IKEA dataset (Lim et al., 2013) contains 219 CAD models of IKEA furniture; for 90 of these, there are also a total of 759 photographs, each containing the corresponding item of furniture, annotated with its precise pose. Very recently, Sun et al. (2018) released the similar but larger Pix3D dataset, which contains 395 CAD models across nine object categories, and a total of 10069 photographs, each showing one of the CAD models with pose annotation.

The Pascal3D+ (Xiang et al., 2014) and ObjectNet3D (Xiang et al., 2016) datasets are designed to evaluate pose-estimation. Similar to IKEA and Pix3D, these contain many natural images with CAD models aligned to them. However, the CAD models are few in number, and only broadly similar in style to the objects in the images. This is not an issue for the pose-estimation task, as the goal here is simply to predict the correct position and rotation of the 3D shape, not to reconstruct it in detail. However, some works do in fact evaluate reconstruction on Pascal3D+; these results should be viewed with caution, due to the very small number of CAD models (maximum of ten per class) encouraging over-fitting.

3.1.2 Evaluation

Reconstruction. Given a dataset of images paired with ground-truth 3D shapes, it is conceptually simple to evaluate reconstruction methods: we run the method on each image, and check whether the result matches the ground-truth shape. In practice, however, we need to define a metric for shape similarity; the most common choice is intersection-over-union (IoU), taking the mean over a large test set (Choy et al., 2016;

Fan et al., 2017; Yan et al., 2016; Tulsiani et al., 2017b). IoU is calculated on voxel grids, and is given by

$$\frac{\sum_i x_i y_i}{\sum_i (x_i + y_i - x_i y_i)} \tag{3.1}$$

where i indexes voxels, and $x_i, y_i \in \{0, 1\}$ are the i^{th} predicted and ground-truth voxels respectively. Other representations, such as meshes, are converted to voxels by sampling on a fixed grid, usually 32^3. Note that this metric assumes that the predicted and ground-truth shapes are in the same frame of reference, *i.e.* aligned with each other. In the majority of works, a canonical object-centric frame is used, common to all instances of the same class.

Generation. Evaluating generation is less straightforward, as we must measure how close the distribution of shapes drawn from a model is to that of the ground-truth data. Different modes of failure are possible: for example, the model may produce realistic shapes, but with these always being identical to shapes in the training set (*i.e.* the model has memorised the training set); or, the shapes may be different from training samples, but centred around only one mode of the training distribution (*e.g.* sports cars or sedans). Many works resort to displaying a large number of examples, in order to convince the reader that the model has indeed learnt the true distribution (*e.g.* Wu et al., 2015a; Soltani et al., 2017; Zou et al., 2017; Li et al., 2017). Some also display nearest neighbours in the training set for each sample, to demonstrate that the model is not simply memorising the training data (*e.g.* Wu et al., 2016). For models with a continuous latent space, some works also display results of interpolating in that space, to show all latent points yield realistic samples (*e.g.* Achlioptas et al., 2018; Li et al., 2017); this provides further evidence that a model has learnt the true data manifold, instead of memorising its training data as isolated points.

To measure the quality of generative models over 2D images, two metrics are commonly used: inception score (Salimans et al., 2016) and Fréchet inception distance (Heusel et al., 2017). These are both based on matching the responses of pre-trained neural networks to those expected from ground-truth data. Unfortunately however, neither of these metrics generalises to the case of 3D shapes.

3.1.3 Classical 3D reconstruction

The goal of *3D reconstruction* is to infer the 3D shape of an object or scene, given one or more images of that object/scene. Solving this problem is a long-standing research area in computer vision. Traditional methods can be grouped into two very

broad categories: *stereo*, and *shape-from-X*. Stereo methods are typically split into binocular and multi-view techniques, depending on the camera configuration they are specialised for. They make only weak assumptions about the scene, but rely on multiple images taken from different viewpoints (see the surveys by Scharstein and Szeliski (2002), Seitz et al. (2006), and more recently Furukawa and Hernández (2015)). In contrast, shape-from-X methods can operate with only a single image, but they make stronger assumptions about the scene and are typically more brittle. In this section, we give a brief overview of these classical methods, while Section 3.1.4 describes modern learning-based methods.

Binocular stereo. 3D reconstruction based on a stereo pair of images, *i.e.* taken near-simultaneously from slightly differing locations, is a classical problem dating back to the seminal work of Marr and Poggio (1979). Binocular stereo methods are based on finding matching points in the two images along epipolar lines, *i.e.* horizontal lines in rectified images (Hartley and Zisserman, 2004). The difference in horizontal coordinates between a pair of matching points is known as the disparity, and is inversely proportional to the physical depth. The results of this process are typically noisy, for example due to untextured or occluded regions. Smoothing is therefore performed, using various discrete energy-minimisation techniques (Boykov et al., 2001a; Felzenszwalb and Huttenlocher, 2004; Szeliski et al., 2008); see also Chapter 5 where we discuss how to select suitable minimisation algorithms automatically. Once the disparity at each point has been estimated, it is straightforward to construct a 3D point-cloud representation of the scene if the camera parameters are known. A comprehensive survey and evaluation of stereo matching methods is given by Scharstein and Szeliski (2002), while more recent discrete optimisation methods applicable to smoothing are surveyed by Szeliski et al. (2008).

Multi-view stereo. In contrast to binocular stereo, multi-view stereo (MVS) assumes access to more than two images of a scene, typically with larger displacements between them. This changes the character of the problem significantly, and leads to very different approaches (Furukawa and Hernández, 2015; Seitz et al., 2006). Modern pipelines begin with a structure-from-motion (SfM) stage, which estimates the camera parameters (relative locations, orientations, focal lengths) by matching sparse keypoints across images (Harris and Stephens, 1988; Lowe, 2004b; Bay et al., 2008), and inferring their locations in 3D space (Hartley and Zisserman, 2004; Pollefeys et al., 1998; Agarwal et al., 2009; Frahm et al., 2010). The results of SfM are refined using

bundle-adjustment, which directly minimises the squared error in projected locations of all keypoints extracted in the SfM stage, across all images (Triggs et al., 2000; Hartley and Zisserman, 2004). As in binocular stereo, once the camera parameters are known, it is necessary to estimate point correspondences between images, and then the detailed 3D geometry of the scene. One approach is to cluster the images according to their estimated camera pose, then to directly estimate depth-maps using binocular stereo techniques within each cluster (Goesele et al., 2007; Furukawa and Hernández, 2015). Many such depth-maps may be fused to yield a global point-cloud reconstruction (Curless and Levoy, 1996; Gallup et al., 2007; Goesele et al., 2007), or a voxel reconstruction (Lempitsky and Boykov, 2007; Zach et al., 2007). Alternatively, a point-cloud may be estimated directly, by progressively building and refining an initial estimate during the reconstruction process (Furukawa and Ponce, 2010; Habbecke and Kobbelt, 2007).

Shape-from-X. There exist a variety of techniques that aim to extract 3D information from a single image (or multiple images taken from the same location) based on different physical principles, broadly grouped as shape-from-X. Among such methods, the most extensively-studied is shape-from-shading. This exploits the fact that the brightness of a surface under diffuse illumination depends on its normal direction. If the lighting and material characteristics are known, it is possible to formulate an optimisation problem to recover the shape of the surface, given just the pixel intensities (Horn, 1975; Zhang et al., 1999; Prados and Faugeras, 2006). Unfortunately, this problem is ill-posed, having an infinite number of solutions, even under the idealised assumption of perfectly Lambertian reflectance (Prados and Faugeras, 2006). Achieving good results therefore requires strong priors (Barron and Malik, 2015). Other shape-from-X methods have $\mathrm{X} \in \{\text{focus, defocus, texture, specularities, shadows}\}$. Each of these techniques is applicable only to rather specific circumstances, for example objects with regular textures (Loh and Hartley, 2005), or when several images are taken with different focal settings but identical pose (Ens and Lawrence, 1993).

3.1.4 Learning-based 3D reconstruction

Humans can mentally form accurate reconstructions of 3D shapes in spite of the intrinsic ambiguity of the problem, even in challenging situations such as partial occlusion. This implies they employ learnt priors on expected shapes, and motivates the use of *learning-based* techniques for reconstruction. In the last three years, there has been a

surge of interest in such methods; this has been enabled both by the growing maturity of deep learning techniques, and by the recent release of large datasets of 3D shapes (Section 3.1.1). These learning-based methods can be divided into two broad groups: (i) those based on a classical MVS workflow, requiring multiple views at test time, but imposing weak assumptions on scene structure (*e.g.* Kar et al., 2017; Hartmann et al., 2017); (ii) those requiring only a single view at test time, but assuming it shows an object of a class seen during training (*e.g.* Choy et al., 2016; Fan et al., 2017; Tulsiani et al., 2017b). Among the latter, we also differentiate between methods that require full 3D supervision (*i.e.* 3D shapes paired with images), and those that need only weaker 2D supervision (*e.g.* pose annotations). Note that we give more details on the different 3D shape representations produced by these methods, such as voxels and point-clouds, in Section 3.1.6.

MVS with learning. In the past year, five works have introduced learning-based reconstruction methods that broadly follow the framework of classical MVS techniques. These assume that the camera parameters have been estimated beforehand using SfM, then apply learning in various different ways to improve the reconstruction process.

Kar et al. (2017) present a model trained end-to-end for reconstruction, that replaces each stage of a classical MVS pipeline with a learnt function. Each image is processed by a fully-convolutional network, and the resulting features unprojected along rays into a 3D voxel grid. Features from different images are aggregated into a single grid using an RNN; this grid is then smoothed by an encoder/decoder model, and a final convolutional stage yields voxel occupancies. They evaluate the method on the ShapeNet dataset (Chang et al., 2015), and show good performance with respect to baselines, although there is no direct comparison to an equivalent non-learnt MVS pipeline. Similarly, Ji et al. (2017) learn to predict voxel occupancies given posed images. Instead of applying a 2D CNN then unprojecting the features, they directly unproject the pixel values. Then, instead of aggregating features from different images using an RNN, they process pairs of images, concatenating their channels before applying an encoder/decoder CNN. Multiple views are handled by applying the model to similarly-posed pairs of images, and combining the results from different pairs, using a learnt weighting. They evaluate the method on the DTU dataset (Aanæs et al., 2016) of table-top images of objects, and show performance comparable to classical MVS methods.

In contrast, Hartmann et al. (2017) use a traditional (non-learnt) MVS pipeline, but learn the similarity metric used to estimate correspondences between images. This

uses a Siamese network (Bromley et al., 1993) applied to several patches at once, which outputs a similarity score for each. They show that this yields more accurate reconstructions than traditional, non-learnt similarity metrics. Huang et al. (2018) learn to estimate disparities directly, in a slightly different pipeline. They construct plane-sweep volumes (Furukawa and Hernández, 2015) for several neighbours of each of selected reference volumes, and apply a Siamese architecture to these volumes, followed by an encoder/decoder network for smoothing. Their method learns to aggregate information from several neighbouring images when estimating disparities for each reference image. It achieves comparable performance on the ETH3D dataset (Schöps et al., 2017) to classical MVS methods. Paschalidou et al. (2018) again learn the disparity estimation stage in an otherwise fixed pipeline. However, instead of optimising for disparities *per se*, they train end-to-end for reconstruction. To achieve this, they back-propagate through the remainder of the pipeline, which is an MRF over voxel occupancies, with potentials enforcing agreement with the estimated disparities for each image. This method again achieves comparable performance to classical MVS methods, on the datasets of Aanæs et al. (2016) and Restrepo et al. (2014).

Overall, it is clear that these methods do not yet provide a great benefit over classical MVS approaches based solely on geometry. However, in the case of single-view reconstruction, learning becomes vital, as there is no longer any 3D geometric information to exploit.

Optimisation-based single-view methods. A handful of works perform single-image reconstruction by directly optimising parameters of a shape model to match a given image. This contrasts with later approaches that learn to regress the shape directly from an image, without an optimisation stage at test time. Prisacariu et al. (2011) and Prisacariu et al. (2012) present a method for single-image reconstruction of cars only, based on learning a low-dimensional generative model of car shapes, and fitting this to single images by optimising an energy functional. It is demonstrated on natural images, but assumes that a segmentation of the car can be obtained.

Vicente et al. (2014) aim to reconstruct object instances in the PASCAL VOC 2012 dataset (Everingham et al., 2015), based on keypoint annotations and ground-truth segmentations. Rather than adopting a learning-based approach, they consider all instances of a given class jointly. The keypoints allow estimation of camera parameters via SfM, with the segmentation imposing further constraints on valid solutions. For every instance, they then search for other instances viewed from different directions; a voxel-carving method is used to combine these into a consistent reconstruction. This

yields many candidate reconstructions per instance, and the most similar to the average shape for the class is selected. Kar et al. (2015) tackle essentially the same problem, with the same inputs during training, but use a CNN to predict the segmentation and approximate viewpoint for unseen images at test time. They model shape for each object class by a mixture of linearly-deformable point-clouds. To train this model, they use non-rigid SfM to estimate camera parameters, and then optimise the shape model for consistency with the ground-truth segmentations. At test time, the linear deformation parameters are directly optimised for consistency with the predicted segmentation.

3D-supervised deep single-view methods. Learning-based single-view reconstruction techniques do away with the classical MVS pipeline, and instead directly predict 3D shapes from images using more conventional CNN architectures. Such methods pose 3D reconstruction as a regression problem, taking an image as input, and yielding the reconstruction as output. They combine the benefits of classical MVS and shape-from-X techniques, as they require only a single image, but make weaker assumptions on the scene geometry. Instead, they assume that the input image shows an object of a class seen during training, and rely on this semantic knowledge to avoid the ambiguity caused by having only a single image at test time. Most such methods are trained with full 3D supervision—*i.e.* they are provided with a large dataset of images, each paired with an aligned 3D shape, and trained to regress the latter from the former. This implies a high annotation cost where the method is applied to natural images, as 3D models must be precisely aligned to each. Hence, some methods train and test on renderings from ShapeNet, which allows ground-truth annotations to be obtained for free, but is an easier task than using images from the wild.

Choy et al. (2016) presented the first deep-learning-based reconstruction pipeline. Their method first applies a 2D CNN to the input image; if multiple views are available, they are processed separately by the CNN, then the results integrated by an RNN. The resulting features are passed to a 3D deconvolutional network, that maps from the feature space to occupancies of a 32^3 voxel grid. This is trained with a standard cross-entropy loss, maximising the likelihood of the ground-truth voxelised shape for the input image. The method is evaluated on renderings of the ShapeNet dataset, as well as on Pascal3D+. Concurrently, Girdhar et al. (2016) proposed a similar model, but supporting only a single view at test time. They train an autoencoder on 3D shapes (again represented by voxels), then separately train a 2D CNN to regress the latent embeddings corresponding to the encoded voxels for its input image; lastly, the entire pipeline is fine-tuned together. This method is evaluated on the IKEA and Pascal3D+

datasets. Wu et al. (2016) proceed similarly, but use a GAN (see Section 3.1.5) in place of the autoencoder, and achieve slightly better results on the IKEA dataset.

Instead of voxels, Soltani et al. (2017) output multiple depth-maps and/or silhouettes, from known (fixed) viewpoints; these are subsequently fused if a voxel reconstruction is required. Their model is a VAE (Kingma and Welling, 2014) that is trained to reconstruct these inputs; as such, it supports generation as well as reconstruction, but does not allow reconstruction from natural images. They show it can be trained with different numbers of inputs to reconstruct from, including a single view.

The current state-of-the-art for reconstruction on ShapeNet is the method of Richter and Roth (2018), which outputs voxels, but in a framework that requires less memory—hence allows higher resolutions—than prior techniques. Instead of predicting voxels directly, they predict six 2D depth maps, facing inwards along each coordinate axis; the volume is given by fusing these depth maps. As this cannot represent occluded regions, the model also outputs a second set of depth maps, which define a volume to be subtracted from the first, and then another set of depth maps, defining a volume to be added, and so on.

Fan et al. (2017) generate a point cloud as the output. As noted in Section 3.1.6, this is memory-efficient, but requires carefully-designed neural network architectures to ensure invariance under reordering of the points (Qi et al., 2017). In order to calculate the reconstruction error differentiably, the authors propose to use a weighted combination of the chamfer and earth-mover's distances, with the ground-truth models also converted to point clouds before training. When reconstructing rendered ShapeNet models, this method achieves higher performance for all classes than that of Choy et al. (2016). Indeed, this method was state-of-the-art for 3D-supervised single-view reconstruction when the work we describe in this chapter was published. Mandikal et al. (2018) again reconstruct point-clouds, but train their model in two stages. They first train an autoencoder for point clouds, based on the architecture of Qi et al. (2017), and using chamfer distance as the reconstruction loss, Then, they separately train an encoder that maps an image to the latent space of the autoencoder, keeping the decoder fixed. This latter stage is trained using either an L1/L2 loss directly on the latent space, or a chamfer loss on the final point-cloud; the former is shown to give better results. Results from applying this method to ShapeNet renderings are slightly better than those of Fan et al. (2017), but a different evaluation metric is used. They also show results on the Pix3D dataset.

Tulsiani et al. (2017a) focus on mapping voxelised shapes to cuboidal primitives

fitting their parts, but also briefly discuss inferring this representation from single images. Niu et al. (2018) again reconstruct varying-cardinality sets of cuboidal primitives, by mapping images to the latent space of the generative model of Li et al. (2017), which we describe further in Section 3.1.5. As that model explicitly represents symmetries, this method can also be used to refine voxel reconstructions predicted by another method, by ensuring that the expected symmetries are satisfied.

Finally, Gwak et al. (2017) and Zhu et al. (2017) present methods that operate on natural images, but have slightly weaker requirements on ground-truth data than the previous methods. As for the previous works, they require a dataset of 3D shapes and images; however, these do not need to be paired with each other. Instead, the images are annotated only with silhouettes. Gwak et al. (2017) train their model to map a given image to a 3D shape that reprojects to match its ground-truth silhouette. An additional adversarial loss is imposed, ensuring that shapes produced by the model be indistinguishable from those in the unpaired training set of ground-truth 3D shapes. Zhu et al. (2017) first train a model mapping ShapeNet renderings to their corresponding voxels, then fine-tune on natural images to minimise the reprojection error to the ground-truth silhouette. Note that this latter method, similarly to our own but in contrast to the others here, explicitly models the pose of the object, in addition to its shape.

2D-supervised deep single-view methods. In contrast to those described so far, a few recent learning-based reconstruction techniques do not rely on 3D ground-truth. These methods are the closest in spirit to our own, which relies on neither 3D ground-truth, nor 2D annotations. They typically work by passing input images through a CNN, which predicts a 3D representation, which is then rendered to form a reconstructed 2D image; the loss is defined to minimise the difference between the reconstructed and original images. These methods vary in how they implement the rendering operation (which must be differentiable to train the system end-to-end), and in the type of supervision required. In all cases, they assume that ground-truth silhouettes are available, or equivalently that the input is rendered on a plain background. They all evaluate quantitative performance using renderings of models from the ShapeNet dataset, but unfortunately use different evaluation protocols, making direct comparison impossible.

Yan et al. (2016) present a method that takes single image as input, and yields a voxel reconstruction. It is trained in two stages. First, multiple views of the same object instance at different poses are passed to the network, which is trained to directly regress an image showing one pose to an image showing another. Importantly, the network has a bottleneck layer, that must encode all the 3D structure of the object

independent of the input and output poses, before being upsampled to a new image given an output pose. In the second training stage, the 2D upsampling network is replaced by a new decoder network that maps from the bottleneck representation to a voxel grid. The voxels are rendered into 2D pixels by computing a max operation along rays cast from each pixel into the voxel grid, at poses matching the input images. The training objective is then to maximise the IOU between these renderings and the silhouettes of the original images.

Rezende et al. (2016) briefly discuss single-image reconstruction, but do not provide detailed results. They define a conditional generative model over meshes, where the conditioning is on a 2D view, and the output mesh is a 162-vertex sphere parameterised by radial offsets to its vertices. The mesh is rendered with a standard OpenGL pipeline; to circumvent non-differentiability of this rendering operation, they use a score-function gradient estimator (Williams, 1992; Mnih and Rezende, 2016). The model is trained in a variational framework to maximise the reconstructed pixel likelihood. This is shown to successfully reconstruct simple shapes such as cubes and cylinders.

Wiles and Zisserman (2017) propose a method that takes silhouette images as input, and produces rotated silhouettes as output; the input and output poses are provided. They consider two methods; the first directly predicts a 2D silhouette, akin to the first training stage of Yan et al. (2016). The second produces voxels in 3D space, and then projects these by computing a max operation along rays cast from each pixel into the voxel grid. The model consists of a simple CNN encoder applied to each input image, with the poses injected at the final layer, and the resulting embedding max-pooled over the different images. This is then concatenated with the required output pose, and passed through an upsampling decoder network, which yields the 2D or 3D output prediction.

Tulsiani et al. (2017b) solve essentially the same problem as Yan et al. (2016), but using a different methodology. They still use an encoder/decoder CNN to regress a voxel grid from a single image. However, the values in this voxel grid are treated as occupancy probabilities, which allows use of a novel differentiable rendering operation. This traces a ray from each pixel, but instead of stopping at the first occupied voxel, calculates the probability that it will terminate at each of the voxels it passes through, not having terminated at any nearer ones. In order to enforce consistency with a silhouette, for pixels inside the silhouette, the probability of rays terminating somewhere along their length is maximised, while for pixels outside the silhouette,

it is minimised. This is straightforwardly extended to the case of depth supervision, where the rays are instead encouraged to terminate at a specific voxel, and not nearer or farther.

In contrast to the previous methods, Novotny et al. (2017) focus on natural images. Specifically, they train a model to perform single-image pose estimation and point-cloud reconstruction, using videos as supervision. Each such video is filmed by a camera moving around an instance of the target class; this allows classical SfM and multi-view stereo methods (see above) to be applied. The obtained reconstructions are then used as ground-truth to train CNNs for single-image reconstruction and pose-estimation.

2D-supervised methods published after ours. Three further works that train for reconstruction with only 2D supervision have been published since ours was submitted. Kato et al. (2018) present a similar method to Yan et al. (2016), but using meshes instead of voxels as the output representation. Specifically, their network outputs vertex offsets to be applied to a 642-vertex icosphere. It is trained using the silhouette IOU as the loss, but also adds a smoothness regularisation term, penalising sharply creased edges. They introduce a new differentiable mesh renderer, that is used to propagate loss gradients back from the silhouette image to the mesh vertices.

Tulsiani et al. (2018) extend their previous work (Tulsiani et al., 2017b) to support the case where the object pose is not given at training time—this is the only work aside from ours which supports training in this setting. To achieve disentanglement of shape and pose (and hence predict pose at test time), they require multiple views of each object instance to be presented together during training; the model is then trained to reconstruct the silhouette for each view using its predicted pose, but the shape predicted from some other view. As in its predecessor work (Tulsiani et al., 2017b), voxels are used as the output representation, with 2D silhouettes rendered using probabilistic ray termination. Here, however, the model also transforms the voxel grid (or equivalently the camera) according to the predicted pose, using trilinear interpolation.

Finally, Genova et al. (2018) consider reconstructing human faces instead of rigid objects, using only unannotated face images for training. They parameterise the output shape using a standard morphable face model, and render the resulting mesh using another new differentiable mesh renderer. This method uses multiple loss terms: (i) one encourages the network to produce output images that have the same identity features as the input image, even under differing illumination (*i.e.* it ensures the input and output show the same person, according to an automated criterion); (ii) one encourages

the network to produce output images that, when fed back into the network, yield the same parameters as the original input image; (iii) one is used on synthetic inputs generated directly from the morphable model, and encourages their (known) parameters to be reconstructed; (iv) one encourages the parameters predicted for each batch to have moments matching those expected by the morphable model.

Depth completion. We note for completeness that there also exist methods that take a single RGB-D image as input, *i.e.* they have access to depth information at test time (Song et al., 2017; Zhang and Funkhouser, 2018; Wu et al., 2015b; Yang et al., 2017; Zou et al., 2017). Instead of reconstruction, the problem then becomes one of completion: the depth image can be directly converted to a sparse point-cloud, but this will be incomplete due to depth-sensing issues, occlusions, and out-of-frame regions. These methods then aim to fill in missing parts of the point-cloud in a semantically-meaningful fashion, typically by using an encoder-decoder network architecture. Similarly, Engelmann et al. (2017) make use of a generative model of shape to regularise the fusion of RGB-D sequences showing cars.

3.1.5 Generation

Here we consider the task of *generating* new 3D shapes for a given class. We focus on probabilistic models, which cast generation as learning a probability distribution over shapes, and then drawing samples from this distribution.

Both generation and reconstruction methods may learn a low-dimensional model of shape. The crucial difference between generation and reconstruction is that generative methods allow sampling *without* conditioning on an image or any other input. Given a reconstruction method that is based on learning a low-dimensional parametric deformable model (*e.g.* Vicente et al., 2014; Kar et al., 2015), this can in theory be adapted to perform generation by learning a prior distribution over the model parameters. However, in general, this distribution will not have a tractable form, as sampling the various deformation parameters independently will not lead to realistic shape samples (Huang et al., 2015). Modern learning-based methods solve this problem by directly optimising the model to ensure that the model parameters required to reconstruct a training set do indeed follow a simple, factored distribution.

As with reconstruction, deep learning techniques have been successfully applied to generation in recent years, leveraging large datasets of 3D CAD models. We begin by discussing the few generative models of shape that preceded the deep-learning era, and

then discuss various recent approaches using deep learning.

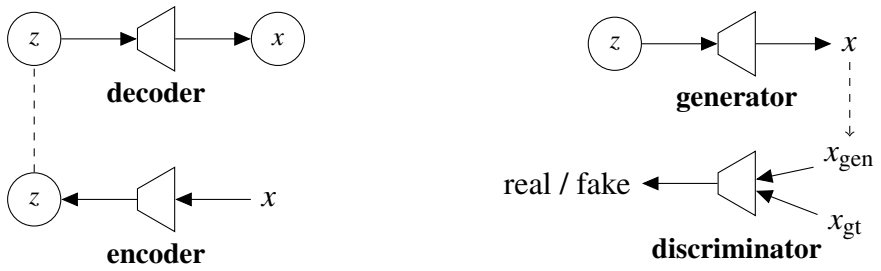
Non-deep methods. Blanz and Vetter (1999) learn a generative model of human faces, from 200 aligned 3D scans. Their model is parameterised as a linear combination of a subset of these scans, defining the position and colour of each vertex. This is extended to allow a different set of coefficients for different areas of the face (*e.g.* mouth, nose, eyes). The distribution of coefficients is learnt from the full dataset of scans, by fitting a multivariate Gaussian; this allows sampling new faces from the model. Allen et al. (2003) apply a very similar technique to a dataset of full human body scans.

Kalogerakis et al. (2012) present a method for generating complex shapes such as aeroplanes and tractors, by learning to combine pre-segmented parts of instances in a training dataset. Their model learns to produce reasonable samples by combining only appropriately-matched parts, based on size, curvature, and other features.

Finally, Prisacariu et al. (2012) focus on single-image reconstruction and pose estimation for cars, but their method involves learning a Gaussian process latent-variable model over 3D car shapes, which can itself be sampled.

3D-supervised deep energy-based models. Huang et al. (2015) present a deep Boltzmann machine (DBM) over 3D shapes, learnt from a dataset of CAD models. They require part annotations on a small, representative subset of the CAD models; based on these, part- and point-correspondences are estimated for the full dataset. Then, a DBM is trained to model the locations of surface points of all shapes in the dataset, including relations within and between parts. This DBM has a somewhat restricted connectivity structure, reflecting the decomposition of the shape into parts. To produce realistic results, point-sets sampled from the DBM are post-processed, by replacing each part with the corresponding part from the dataset that most-closely matches the sampled points. The selected part is weakly deformed to even better match the target points. Wu et al. (2015b) train a convolutional deep belief network (another variant of the Boltzmann machine) over voxel grids. They show that this can produce samples from a wide variety of object classes, as well as supporting inference tasks such as shape completion. Xie et al. (2018) present an alternative convolutional energy-based model over voxel grids, which they find gives improved quantitative results over Wu et al. (2015b).

3D-supervised deep directed models. In 2014, two new classes of deep generative model were developed. These radically improved the capabilities of generative



(a) Variational autoencoder (Kingma and Welling, 2014)

(b) Generative adversarial network (Goodfellow et al., 2014)

Figure 3.1: Deep generative models; see text for details of each. The trapezoids represent neural networks. Variables in a circle are Gaussian random variables; uncircled variables are deterministic. The dashed line in (a) represents the fact that samples from the approximate posterior distribution of z (bottom) replace the prior (top) during training; the dashed line in (b) represents the fact that generated samples are passed to the discriminator, and gradients flow back to train the generator.

models of 2D images. Since then, several methods have emerged applying these two techniques to generative modelling of 3D shapes.

Kingma and Welling (2014) proposed the *variational autoencoder* (VAE). The generative process for this model is to draw a sample z from a latent low-dimensional Gaussian prior distribution, then to transform it via a learnt neural network (the *decoder*) into the output space (typically higher-dimensional); the final sample x is then drawn from an isotropic Gaussian centred at this point (Figure 3.1a). In order to train this network, amortised variational inference (Rezende et al., 2014; Jordan et al., 1999) is used. This uses a second neural network (the *encoder*) to estimate the posterior distribution of the latent variable z given a sample x in the output space, under the assumption that this posterior is also approximately Gaussian. The two neural networks are trained together end-to-end to reconstruct points x, via latent values z drawn from the approximate posterior. The resulting model therefore resembles an autoencoder, albeit with a latent space that is encouraged to follow a Gaussian distribution.

Goodfellow et al. (2014) proposed the *generative adversarial network* (GAN). This has a similar generative process to the VAE: a value z drawn from a low-dimensional Gaussian prior is passed through a neural network, to produce the final sample x (there is no stochasticity at the output stage, unlike in the VAE). However, the training objective is very different. A second neural network is introduced, that is trained to discriminate between 'real' samples drawn from the training set, and 'fake' samples

generated by the model (Figure 3.1b). The generative model is simultaneously trained to minimise the accuracy of this discriminator, and thus learns to generate samples resembling those in the training set.

We now describe several generative models for 3D shape based on VAEs and GANs. Wu et al. (2016) present a GAN-based method, that directly models voxels. This is based closely on a similar model for images (Radford et al., 2016), but uses 3D convolutions instead of 2D. They also present a second model, that extends this by adding an encoder network that takes 2D images as input, and is trained to predict the latent representation of the corresponding 3D shape. Samples from this model are shown to be significantly higher-quality than those from Wu et al. (2015b); however, it cannot perform other inference tasks such as shape completion. Rezende et al. (2016) focus on learning and reconstructing from images, but also describe a VAE-based method over voxels. This uses a recurrent neural network for both the decoder and encoder, which allows the shape to be generated and refined progressively, rather than in a single feed-forward pass through a decoder network as with other VAE-based models. Rather than directly modelling voxels, Soltani et al. (2017) present a VAE over multiple depth-maps and/or silhouettes, from known (fixed) viewpoints; these can be straightforwardly converted to/from a full voxel grid if needed, but impose lower computational costs than a full 3D model, allowing more powerful neural networks to be used.

Nash and Williams (2017) model surface points of shapes divided into parts, similarly to Huang et al. (2015); however, instead of a DBM, they use a VAE formulation. They explicitly model the presence or absence of different parts by binary variables; these are provided to the model both during training and at test time. They are combined with a latent embedding z, before being mapped to one embedding per part, and then to point locations and normals, by learnt neural networks in both cases. Instead of a VAE, Achlioptas et al. (2018) present a model that first trains an autoencoder for dimensionality reduction of point-clouds, then a GAN on its embeddings. They use the network architectures of Qi et al. (2017) to process point-clouds in an order-invariant fashion, and show high-quality samples when trained on ShapeNet. Interestingly, their baseline, which uses a Gaussian mixture model on the embedding space in place of the GAN, also produces plausible samples.

Li et al. (2017) model shapes hierarchically: coarsely, as assembled from cuboidal primitives, and finely, by voxels within each primitive. Moreover, they explicitly model symmetry and part re-use, through a tree-structured shape representation. This requires

explicit annotation of training instances with parts, but the tree structure is discovered automatically. They use a recursive neural network to learn this, trained for autoencoding, and then separately train a GAN over the latent space to allow sampling. Zou et al. (2017) present a simpler model based on a recurrent neural network (RNN). This again models shapes as a set of cuboidal primitives, which the RNN is trained to output sequentially. Unlike for Li et al. (2017), the primitives constitute the final shape—there is no separate stage to add detail, and so the samples are less realistic. To train this model, the input shapes are first converted to primitives by an automatic optimisation process; these are then passed to the RNN sorted according to their vertical location. This method is shown to generate reasonable samples; however, they are limited in detail as the model uses relatively few primitives per instance (up to ten).

Tan et al. (2018) present a VAE generating meshes (this work was published after ours). They fix the mesh topology and choose a particular parameterisation; the VAE then directly models these parameters. Calculating the actual vertex locations requires a further energy-based optimisation given the mesh parameters. This method is demonstrated on human bodies, and yields high-quality samples; however, it is not directly applicable to the more common ShapeNet or ModelNet datasets, as these have varying mesh topology across instances.

2D-supervised deep directed models. The only prior work that learns a generative model of 3D shapes given just *2D* images is that of Gadelha et al. (2017); as such, this is the most similar in spirit to our own. They use a GAN over voxels; these are projected to images by a simple max operation along rays, to give silhouettes. The discriminator network is trained to distinguish projections of samples from projections of ground-truth data; the generator accordingly learns to produce plausible voxelised shapes. This method assumes that the object poses are restricted to a small set (just eight angles), and cannot learn concave shapes due to its reliance on silhouettes.

3.1.6 3D shape representations

We now give some more details on the different 3D shape representations used by the learning-based methods for reconstruction and generation described in the previous sections.

The most common shape representation in learning-based methods is a grid of **voxels**, as it is straightforward to extend tools such as CNNs from 2D pixels to 3D voxels (*e.g.* Choy et al., 2016; Tulsiani et al., 2017b). Moreover, voxels provide a very

flexible representation, allowing arbitrary surface topologies. However, they cannot faithfully represent surfaces that are curved, or otherwise non-axis-aligned. Moreover, dense voxel grids are memory-intensive. To mitigate this, both Tatarchenko et al. (2017) and Häne et al. (2017) aim to reduce the memory requirement by sparsening the voxel grid. They use an octree representation, which adapts according to the geometry so fine details can be modelled where needed, but large areas of free/occupied space are handled efficiently. An alternative approach for reducing the memory required for voxels, is to instead model multiple depth-maps, taken from different, fixed viewpoints, *e.g.* along each axis (Richter and Roth, 2018; Soltani et al., 2017). These depth-maps are lightweight to model as they are only 2D, but they may be fused into (or generated from) a voxel grid as required.

A representation with lower memory requirements than voxels is unstructured **point clouds**. Here, 3D shapes are represented by a fixed number of points, and parameterised by the locations of those points. This technique is both efficient and effective (Fan et al., 2017). Like voxels, it can approximate shapes with arbitrary topology. However, it does not explicitly define the space occupied by the object, as the extent of the points is effectively zero, and surface normals are similarly not well-defined. Moreover, it requires non-standard neural network architectures to account for the fact that points may be reordered without changing the resulting shape (Qi et al., 2017; Ben-Shabat et al., 2017). As a special case of point clouds, some models represent just the surface points of an object, not the interior (Nash and Williams, 2017; Huang et al., 2015), thus reducing memory requirements further. In this case, it is possible to also model the surface normal direction at every point (Nash and Williams, 2017), which allows for higher-fidelity rendering.

In computer graphics, shapes are most commonly represented by **meshes**. These are defined by a set of vertex locations, and a set of faces, typically triangular, that connect the vertices. Unlike voxels and point-clouds, a fixed-size mesh representation can model surfaces of arbitrary spatial extent and orientation. In general, meshes can also represent arbitrary topologies, by having differing numbers of vertices and choices of faces. Meshes have not been widely used in learning-based reconstruction methods: for 2D-supervised techniques, they imply the need for a differentiable renderer; for 3D-supervised techniques, the differing mesh topologies of ground-truth models must be reconciled. An exception is the very recent work of Kato et al. (2018), which does use meshes as an output representation, keeping the topology fixed.

At a higher level, objects may be assembled from **primitives**, such as blocks (Tul-

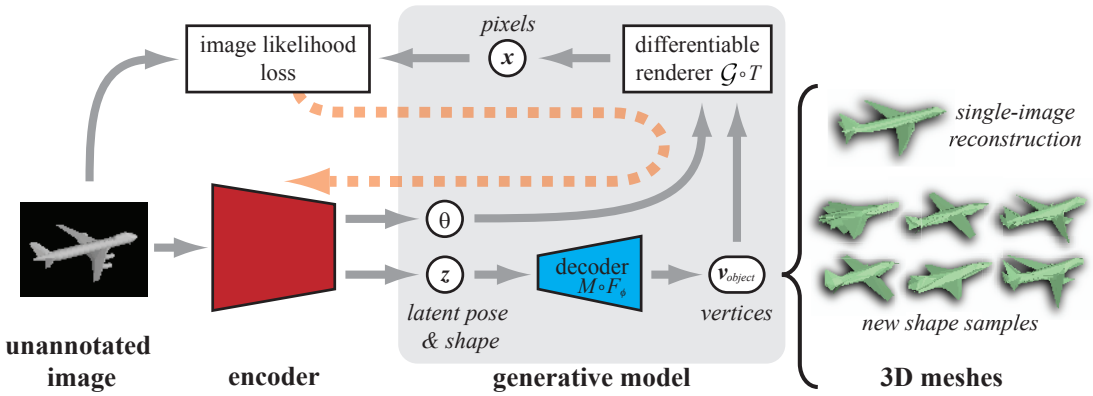


Figure 3.2: Given only unannotated 2D images as training data, our model learns (1) to reconstruct and predict the pose of 3D meshes from a single test image, and (2) to generate new 3D mesh samples. It is trained end-to-end (orange dashed arrow) to reconstruct input images, via a differentiable renderer that produces lit, shaded RGB images, allowing us to exploit shading cues in the loss.

siani et al., 2017a; Li et al., 2017; Niu et al., 2018); this is a very old approach in computer vision, dating back to the 1960s (Binford, 1971; Roberts, 1965). Primitive-based representations can be even more memory-efficient than point-clouds, as a single primitive may be scaled to represent an arbitrarily large volume of the object. However, the expressiveness of this representation is strongly dependent on the choice of primitives and allowed transformations. Recent approaches have used only cuboids, parameterised by a translation, rotation, and scale (Tulsiani et al., 2017a; Li et al., 2017; Niu et al., 2018).

3.2 Generative Model

Recall that our goal in this chapter is to build a model that supports both single-image 3D reconstruction and generation of new shape samples, and that can be trained with unannotated 2D images. To achieve this, we construct a probabilistic generative model, that first samples the shape and pose of a 3D mesh, then produces a 2D rendering from these. Incorporating the full image-formation process in the graphical model allows it to be trained purely from 2D images, while explicitly modelling the latent 3D shape allows us to perform the reconstruction and generation tasks.

We assume that the content of an image can be explained by two independent latent components—the shape of the mesh, and its pose relative to the camera. These

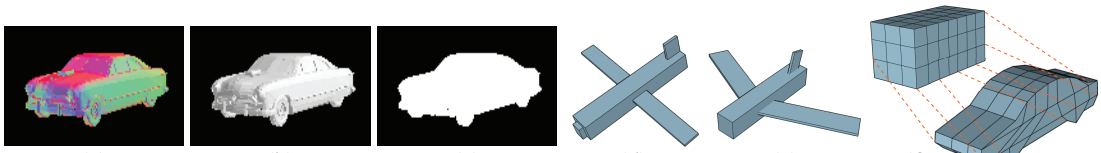


Figure 3.3: **Lighting**: Coloured directional lighting (a) provides strong cues for surface orientation; white light (b) provides less information; silhouettes (c) provide none at all. Our model is able to exploit the shading information from coloured or white lighting. **Mesh parameterisations**: **ortho-block** & **full-block** (assembly from cuboidal primitives, of fixed or varying orientation) are suited to objects consisting of compact parts (d-e); **subdivision** (per-vertex deformation of a subdivided cube) is suited to complex continuous surfaces (f).

are modelled by two low-dimensional random variables, $\mathbf{z}$ and θ respectively. The joint distribution over these and the resulting pixels $\mathbf{x}$ factorises as $P(\mathbf{x},\mathbf{z},\theta) = P(\theta)P(\mathbf{z})P(\mathbf{x}\,|\,\mathbf{z},\theta)$.

Following Gadelha et al. (2017), Yan et al. (2016), Tulsiani et al. (2017b), and Wiles and Zisserman (2017), we assume that the pose θ is parameterised by just the azimuth angle, with $\theta \sim \text{Uniform}(-\pi,\pi)$. The camera is then placed at fixed distance and elevation relative to the object. Following recent works on deep latent variable models (Kingma and Welling, 2014; Goodfellow et al., 2014), we assume that $\mathbf{z}$ is drawn from a standard isotropic Gaussian, and then transformed by a deterministic *decoder network*, F_ϕ, parameterised by weights ϕ which are to be learnt. This produces the *mesh parameters* $\Pi = F_\phi(\mathbf{z})$. Intuitively, the decoder network F_ϕ transforms and entangles the dimensions of $\mathbf{z}$ such that all values in the latent space map to plausible values for Π, even if these lie on a highly nonlinear manifold. Note that our approach contrasts with previous models that directly output pixels (Kingma and Welling, 2014; Goodfellow et al., 2014) or voxels (Wu et al., 2016; Gadelha et al., 2017) from a decoder network.

We use Π as inputs to a fixed mesh parameterisation function $M(\Pi)$, which yields vertices $\mathbf{v}_{\text{object}}$ of triangles defining the shape of the object in 3D space, in a canonical pose (different options for M are described below). The vertices $\mathbf{v}_{\text{object}}$ are transformed into camera space according to the pose θ, by a fixed function T: $\mathbf{v}_{\text{camera}} = T(\mathbf{v}_{\text{object}}, \theta)$. They are then rendered into an RGB image $I_0 = \mathcal{G}(\mathbf{v}_{\text{camera}})$ by a rasteriser $\mathcal{G}$ with Gouraud shading (Gouraud, 1971) and Lambertian directional lighting (Lambert, 1760); we give more details on this rendering operation in Section 3.4. We are free to choose the lighting parameters: our experiments include tri-directional coloured

lighting, and white directional lighting with an ambient component.

The final observed pixel values $\mathbf{x}$ are modelled as independent Gaussian random variables, with means equal to the values in an L-level Gaussian pyramid (Burt and Adelson, 1983), whose base level equals I_0, and whose L^{th} level has smallest dimension equal to one:

$$P_\phi(\mathbf{x}\,|\,\mathbf{z},\theta) = \prod_l P_\phi(\mathbf{x}_l\,|\,\mathbf{z},\theta) \tag{3.2}$$

$$\mathbf{x}_l \sim \text{Normal}\left(I_l, \tfrac{\varepsilon}{2^l}\right) \tag{3.3}$$

$$I_0 = \mathcal{G}(T(M(F_\phi(\mathbf{z})),\theta)) \tag{3.4}$$

$$I_{l+1} = I_l * k_G \tag{3.5}$$

where k_G is a small Gaussian kernel, ε is the noise magnitude at the base scale, and $*$ denotes convolution with stride two. We use a multi-scale pyramid instead of just the raw pixel values to ensure that, during training, there will be gradient forces over long distances in the image, thus avoiding bad local minima where the reconstruction is far from the input.

Mesh parameterisations. After the decoder network has transformed the latent embedding $\mathbf{z}$ into the mesh parameters Π, these are converted to actual 3D vertices using a simple, non-learnt mesh-parameterisation function M. One possible choice for M is the identity function, in which case the decoder network directly outputs vertex locations. However, initial experiments showed that this does not work well: it produces very irregular meshes with large numbers of intersecting triangles. Conversely, using a more sophisticated form for M enforces regularity of the mesh. We use three different parameterisations in our experiments.

In our first parameterisation, Π specifies the locations and scales of a fixed number of axis-aligned cuboidal *primitives* (Figure 3.3d), from which the mesh is assembled (similar to Zou et al. (2017) and Tulsiani et al. (2017a)). Changing Π can produce configurations with different topologies, depending which blocks touch or overlap, but all surfaces will necessarily be axis-aligned. In our experiments we call this **ortho-block**.

Our second parameterisation is strictly more powerful than the first: we still assemble the mesh from cuboidal primitives, but now parameterise each with a 3D rotation, in addition to its location and scale. In our experiments we call this **full-block** (Figure 3.3e).

The above parameterisations are naturally suited to objects composed of compact

parts, but cannot represent complex continuous surfaces. For these, we define a third parameterisation, **subdivision** (Figure 3.3f). This parameterisation is based on a single unit cube, centred at the origin; the edges and faces of the cube are subdivided several times along each axis. Then, Π specifies a list of displacements, one per vertex, which deform the subdivided cube into the required shape.

3.3 Variational Training

We wish to learn the parameters of our model from a training set of 2D images of objects of a single class. More precisely, we assume access to a set of images $\{\mathbf{x}^{(i)}\}$, each showing an unknown object instance at unknown pose. Note that we do *not* require that there are multiple views of each object (in contrast with Yan et al., 2016), nor that the object poses are given as supervision (in contrast with Yan et al., 2016; Tulsiani et al., 2017b; Wiles and Zisserman, 2017).

We seek to maximise the marginal log-likelihood of the training set, which is given by $\sum_i \log P_\phi(\mathbf{x}^{(i)})$, with respect to ϕ. For each image, we have

$$\log P_\phi(\mathbf{x}^{(i)}) = \log \int_{\mathbf{z},\theta} P_\phi(\mathbf{x}^{(i)} \,|\, \mathbf{z}, \theta) P(\mathbf{z}) P(\theta) d\mathbf{z}\, d\theta. \tag{3.6}$$

Unfortunately this is intractable, due to the integral over the latent space $\mathbf{z}, \theta$. Hence, we use amortised variational inference, in the form of stochastic gradient variational Bayes (Kingma and Welling, 2014; Rezende et al., 2014). This introduces an approximate posterior $Q_\omega(\mathbf{z}, \theta \,|\, \mathbf{x})$, parameterised by some ω that we learn jointly with the model parameters ϕ. Intuitively, Q maps an image $\mathbf{x}$ directly to a distribution over likely values of the latent variables $\mathbf{z}$ and θ. Instead of the log-likelihood (3.6), we then maximise the *evidence lower bound* (ELBO):

$$\mathop{\mathbb{E}}_{\mathbf{z},\theta \sim Q_\omega(\mathbf{z},\theta \,|\, \mathbf{x}^{(i)})} \Big[\log P_\phi(\mathbf{x}^{(i)} \,|\, \mathbf{z}, \theta) \Big] - KL \Big[Q_\omega(\mathbf{z}, \theta \,|\, \mathbf{x}^{(i)}) \,\Big|\Big|\, P(\mathbf{z}) P(\theta) \Big] \leq \log P_\phi(\mathbf{x}^{(i)}). \tag{3.7}$$

This lower-bound on the log-likelihood can be evaluated efficiently, as the necessary integration is now with respect to Q, for which we are free to choose a tractable form. The expectation can then be approximated using a single sample.

We let Q be a mean-field approximation, factorised as $Q_\omega(\mathbf{z}, \theta \,|\, \mathbf{x}) = Q_\omega(\mathbf{z} \,|\, \mathbf{x}) Q_\omega(\theta \,|\, \mathbf{x})$. $Q_\omega(\mathbf{z} \,|\, \mathbf{x})$ is a multivariate Gaussian with diagonal covariance. The mean and variance of each latent dimension are given by an *encoder network*, $\text{enc}_\omega(\mathbf{x})$, which takes the image $\mathbf{x}$ as input. For this encoder network we use a CNN with architecture similar to

that of Wiles and Zisserman (2017). When training with multiple views per instance, we apply the encoder network to each image separately, then calculate the final shape embedding $\mathbf{z}$ by max-pooling each dimension over all views.

For the pose θ, we could similarly use a Gaussian posterior. However, many objects are roughly symmetric with respect to rotation, and so the true posterior is typically multi-modal. We capture this multi-modality by decomposing the rotation into coarse and fine parts (Mousavian et al., 2017): an integer random variable θ_{coarse} that chooses from R rotation bins, and a small Gaussian offset θ_{fine} relative to this. We apply this transformation in both the generative $P(\theta)$ and variational $Q_\omega(\theta)$, giving

$$\theta = -\pi + \theta_{\text{coarse}} \frac{2\pi}{R} + \theta_{\text{fine}} \tag{3.8}$$

$$P(\theta_{\text{coarse}} = r) = 1/R, \qquad P(\theta_{\text{fine}}) = \text{Normal}(\theta_{\text{fine}} \,|\, 0, \pi/R) \tag{3.9}$$

$$Q_\omega\left(\theta_{\text{coarse}} = r \,\middle|\, \mathbf{x}^{(i)}\right) = \rho_r\left(\mathbf{x}^{(i)}\right), \qquad Q_\omega(\theta_{\text{fine}}) = \text{Normal}\left(\theta_{\text{fine}} \,\middle|\, \xi(\mathbf{x}^{(i)}), \zeta(\mathbf{x}^{(i)})\right) \tag{3.10}$$

where the variational parameters ρ_r, ξ, ζ for image $\mathbf{x}^{(i)}$ are again estimated by the encoder network $\text{enc}_\omega(\mathbf{x}^{(i)})$. Provided R is sufficiently small, we can integrate directly with respect to θ_{coarse} when evaluating (3.7), *i.e.* sum over all possible rotations. We found in initial experiments that this significantly improves performance.

Imposing a uniform pose prior. While the above allows our training process to reason over different poses, it is still prone to predicting the same pose θ for every image; clearly this does not correspond to the prior on θ given by (3.9). The model is therefore relying on the shape embedding $\mathbf{z}$ to model all variability, rather than disentangling shape and pose. The ELBO (3.7) does include a KL-divergence term that should encourage latent variables to match their prior. However, it does not have a useful effect for θ_{coarse}: minimising the KL divergence from a uniform distribution for each sample individually corresponds to independently minimising all the probabilities $Q_\omega(\theta_{\text{coarse}})$, which does not encourage uniformity of the full distribution. The effect we desire is to match the aggregated posterior distribution $\left\langle Q_\omega(\theta \,|\, \mathbf{x}^{(i)}) \right\rangle_i$ to the prior $P(\theta)$, where $\langle \cdot \rangle_i$ is the empirical mean over the training set. As θ_{coarse} follows a categorical distribution in both generative and variational models, we can directly minimise the L1 distance between the aggregated posterior and the prior:

$$\sum_r^R \left| \left\langle Q_\omega\left(\theta_{\text{coarse}} = r \,|\, \mathbf{x}^{(i)}\right) \right\rangle_i - P\left(\theta_{\text{coarse}} = r\right) \right| = \sum_r^R \left| \left\langle \rho_r(\mathbf{x}^{(i)}) \right\rangle_i - \frac{1}{R} \right|. \tag{3.11}$$

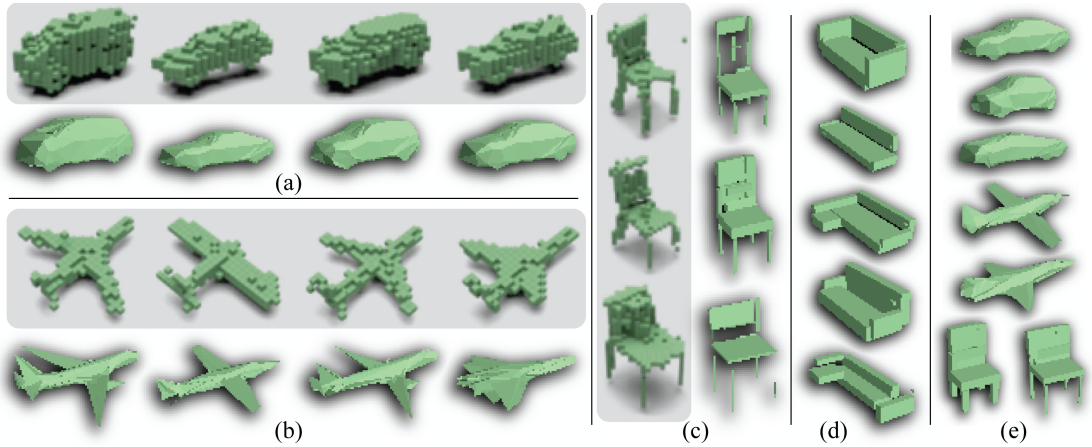


Figure 3.4: **(a-c)** Samples from Gadelha et al. (2017) (grey background), shown next to stylistically-similar samples from our model (white background). Both are trained with a single view per instance, and without ground-truth pose. However, our model outputs meshes, and uses shading in the loss. **(d)** For *sofa*, we only show samples from our model, as Gadelha et al. (2017) cannot handle sofas due to the concavities. **(e)** Additional samples from our model, showing their diversity and quality

We use this term in place of $KL\left[Q(\theta_{\text{coarse}}\,|\,\mathbf{x}^{(i)})\,\middle|\middle|\,P(\theta_{\text{coarse}})\right]$ in our loss, approximating the empirical mean with a single minibatch.

Loss. Our final loss function for a minibatch $\mathcal{B}$ is then given by

$$\sum_{r}^{R}\left\{-\left\langle \rho_r(\mathbf{x}^{(i)})\mathop{\mathbb{E}}_{\mathbf{z},\theta_{\text{fine}}\sim Q_\omega}\left[\log P_\phi\left(\mathbf{x}^{(i)}\,\middle|\,\mathbf{z},\theta_{\text{coarse}}=r,\theta_{\text{fine}}\right)\right]\right\rangle_{i\in\mathcal{B}}+\alpha\left|\left\langle\rho_r(\mathbf{x}^{(i)})\right\rangle_{i\in\mathcal{B}}-\frac{1}{R}\right|\right\}$$
$$+\beta\left\langle KL\left[Q_\omega\left(\mathbf{z},\theta_{\text{fine}}\,\middle|\,\mathbf{x}^{(i)}\right)\,\middle|\middle|\,P(\mathbf{z})P(\theta_{\text{fine}})\right]\right\rangle_{i\in\mathcal{B}} \quad (3.12)$$

where β increases the relative weight of the KL term as in Higgins et al. (2017), and α controls the strength of the pose prior matching. We minimise (3.12) with respect to ϕ and ω using ADAM (Kingma and Ba, 2015), applying the reparameterisation trick (Kingma and Welling, 2014; Rezende et al., 2014) to handle the Gaussian random variables. Section 3.4 describes how we back-propagate gradients from the loss through the rendering operation, to the vertices themselves.

3.4 Differentiable rendering

Optimising (3.12) by gradient descent requires differentiating through the rendering operation $\mathcal{G}$ used to calculate $P_\phi(\mathbf{x}\,|\,\mathbf{z},\theta)$, to find the derivative of the pixels with respect

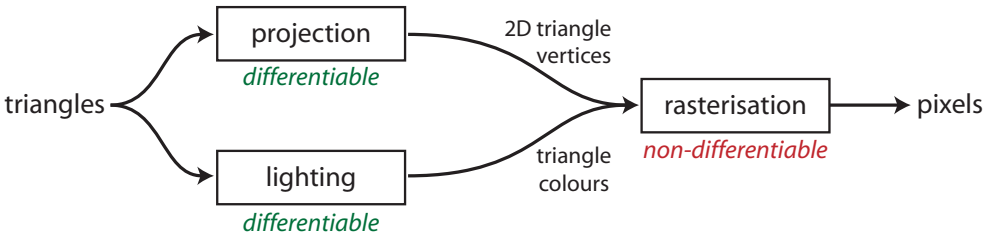


Figure 3.5: In order to render mesh, consisting of triangles in 3D space, we must calculate the light reflected by each face, and project the vertices to 2D. Then, we rasterise the resulting 2D triangles to give the final pixels. The first two stages are differentiable; the third is not, and we must define approximate derivatives.

to the vertex locations and colours. While computing exact derivatives of $\mathcal{G}$ is very expensive, Loper and Black (2014) describe an efficient approximation. We employ a similar technique here, and have made our TensorFlow implementation available to the community.[1] In the remainder of this section, we describe the forward rendering process in more detail, and then how we calculate approximate derivatives.

Forward rendering. The rendering function $\mathcal{G}$ takes triangles in 3D space as input, and produces an image as output (Figure 3.5). It first calculates the light reflected by each triangle; we use only direct illumination by directional and ambient light sources, so the reflected intensity is constant over the area of the triangle. The surface of each triangle is modelled as a purely-diffuse (Lambertian) white material (Lambert, 1760), and the light sources are as described in Section 3.2.

The renderer next maps the 3D vertices into 2D pixel space using a perspective projection, and draws, or *rasterises*, the resulting 2D triangles on the screen—each pixel receives the colour of the nearest triangle to the camera, that lies under the centre point of that pixel. Pixels that are not covered by any triangle are left coloured black. This rasterisation process corresponds directly to what is described by the OpenGL standard (Segal and Akeley, 2018), and is efficiently implemented on modern GPUs via depth-buffering.

Differentiating the rendering operation. The rendering function just described is non-differentiable, as the intensity of a pixel undergoes a step change when a triangle moves over its centre (Figure 3.6). We therefore need to define approximate derivatives of the image with respect to the projected vertex coordinates; this corresponds to differ-

[1] `https://github.com/pmh47/dirt`

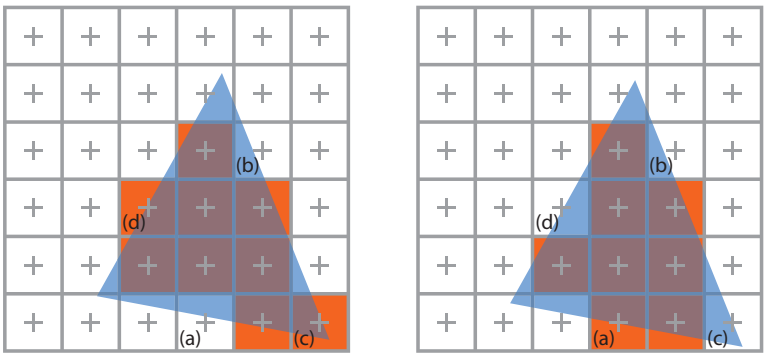


Figure 3.6: Rasterising a triangle, at two slightly different locations. **Left:** A projected triangular face (blue overlay) is rasterised by setting all pixels (grid cells) whose centres (grey crosses) underlie it to its colour (orange), and leaving other pixels unchanged (white) **Right:** This process is non-differentiable, as shifting the triangle down very slightly results in a different set of pixel centres touching it.

entiating through the 'rasterisation' box in Figure 3.5. The resulting gradient can then be propagated back through the projection and lighting calculations straightforwardly, as these operations are differentiable.

Loper and Black (2014) suggest an efficient approximation to calculate the necessary derivatives; our approach is inspired by theirs, but is slightly more efficient. Note first that the gradient of the pixels with respect to the projected vertex locations is non-zero only for pixels that lie immediately inside or immediately outside a triangle, *i.e.* at an edge. The remaining pixels either lie in the uniform black background, or in the interior of a uniformly-coloured triangle, so a small change to the vertex coordinates will not affect them. We therefore consider pixels at two kinds of edges: those where a triangle occludes the background, and those where a triangle meets another triangle (either abutting or occluding).

Edges separating a triangle and the background. In this case, a pixel just outside the triangle will become part of the triangle if the corresponding edge moves outwards (pixel (a) in Figure 3.6), hence its colour will change from that of the background to that of the triangle. If the edge moves inwards, there will be no change to its colour (*e.g.* pixel (b) in Figure 3.6). Conversely, a pixel just inside the triangle will undergo an intensity change (becoming equal to the background; pixels (c) and (d) in Figure 3.6) when the edge moves inwards, but not outwards. Changes in the position of an edge are caused by changes in the positions of the vertices at either end of it. At an arbitrarily small distance from one vertex, the motion of the edge matches that of the vertex;

elsewhere, the influence of the vertex decreases linearly with distance along the edge from it.

More formally, we consider the rendered image as a function $I(\mathbf{p}, \mathbf{e}_1, \ldots, \mathbf{e}_n)$ of the continuous location $\mathbf{p}$ in 2D pixel space, and the endpoints $\mathbf{e}_i$ of a set of triangle edges, each of which is defined by two vertex locations; when evaluated at integer $\mathbf{p}$, this function matches the relevant pixel (we omit the dependency of I on the lit triangle colours). Our goal is to approximate $\nabla_{\mathbf{e}_i} I$. For simplicity, we now describe the 1D case where both x and e_i are scalars; however, the following results extend straightforwardly to the 2D case.

The rendering operation has the property that in the vicinity of e_i and for small ε,

$$I(p, e_1, \ldots, e_i + \varepsilon, \ldots, e_n) = I(p - \varepsilon, e_1, \ldots, e_i, \ldots, e_n) \tag{3.13}$$

This can be seen intuitively, by noting that if we are near enough to the i^{th} edge that no other edges influence the colour there, then the effect of moving the edge is equivalent to moving the view (or camera) in the opposite direction.

In the vicinity of e_i we therefore have

$$\begin{aligned}
\frac{\partial I}{\partial e_i} &= \lim_{\varepsilon \to 0} \frac{I(p, e_1, \ldots, e_i + \varepsilon, \ldots, e_n) - I(p, e_1, \ldots, e_i, \ldots, e_n)}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{I(p - \varepsilon, e_1, \ldots, e_i, \ldots, e_n) - I(p, e_1, \ldots, e_i, \ldots, e_n)}{\varepsilon} \\
&= \lim_{\varepsilon' \to 0} \frac{I(p + \varepsilon', e_1, \ldots, e_i, \ldots, e_n) - I(p, e_1, \ldots, e_i, \ldots, e_n)}{-\varepsilon'} \\
&= -\frac{\partial I}{\partial p}
\end{aligned} \tag{3.14}$$

This shows that the gradient $\nabla_{\mathbf{e}_i} I$ of the image with respect to the edge locations can be written in terms of the spatial gradient of the image itself. For an image sampled at discrete pixels, we approximate latter using finite-difference (Sobel or Scharr) filters.

As noted above, the location of points on an edge varies linearly with the locations of the vertices at either end; we can therefore calculate the gradient of the pixels with respect to the locations of the 2D vertices themselves:

$$\nabla_{\mathbf{u}_i} I = \lambda \nabla_{\mathbf{e}_i} I \tag{3.15}$$

$$\nabla_{\mathbf{v}_i} I = (1 - \lambda) \nabla_{\mathbf{e}_i} I \tag{3.16}$$

where $\mathbf{u}_i$ and $\mathbf{v}_i$ are the endpoints of the i^{th} edge, and λ varies from zero to one as we move along the edge from $\mathbf{v}_i$ to $\mathbf{u}_i$.

Edges separating two triangles. In this case, we either have two triangles meeting at an edge, or one triangle occluding another. In practice, we can treat the former as a special case of the latter, with an infinitesimally small overlap. The edge of the occluding triangle (*i.e.* the one nearer the camera) is then analogous to an edge between a triangle and the background, and the derivatives of the pixels just inside and just outside can be calculated using the same approach as described above. In contrast, there is no gradient propagated to the vertices of the occluded triangle, as changing its location slightly under the occluder has no effect on the pixels.

Implementation. We have noted that the forward rendering operation can be implemented using OpenGL. For the backward (derivative) operation, we also use OpenGL, to rasterise pseudo-images showing the index of the triangle under each pixel, and the barycentric coordinate of the pixel with respect to the triangle (*i.e.* the normalised distance of the point from each vertex). We apply a finite-difference filter to the rendered image, and use (3.14) to transform this into derivatives with respect to the edge locations. The triangle-index and barycentric pseudo-images are then used to transform these derivatives into derivatives with respect to the vertices themselves, according to (3.15) and (3.16), summing contributions due to each edge.

Finally, note that the derivative of each pixel with respect to the lit triangle colours, necessary to exploit shading cues, is one for the nearest triangle overlapping the pixel, and zero otherwise. This is calculated straightforwardly given the triangle-index pseudo-image.

3.5 Experiments

We follow recent works (Gadelha et al., 2017; Yan et al., 2016; Tulsiani et al., 2017b; Fan et al., 2017) and evaluate our approach using the ShapeNet dataset (Chang et al., 2015). Using synthetic data has two advantages: it allows (i) controlled experiments modifying lighting and other parameters; (ii) benchmarking the performance of the reconstruction network against ground-truth 3D shapes. Our experiments focus on the four classes *aeroplane*, *car*, *chair*, and *sofa*. The first three are used in Gadelha et al. (2017), Tulsiani et al. (2017b), and Yan et al. (2016), while the fourth is an example of a highly concave class that is not easily handled by silhouette-based approaches.

To rigorously evaluate the performance of our model, we vary several factors:

- **Mesh parameterisations:** We evaluate the three parameterisations described in

Section 3.2.

- **Lighting:** Unlike previous works (Gadelha et al., 2017; Wiles and Zisserman, 2017; Yan et al., 2016; Tulsiani et al., 2017b), our method is able to exploit shading in the images. We test in two settings, illumination by (i) three coloured directional lights (**colour**), and (ii) one white directional light plus a white ambient component (**white**).

- **Reconstruction loss:** We typically calculate the reconstruction loss (pixel log-likelihood) over the RGB shaded image (**shading**), but for comparison with Yan et al. (2016), Tulsiani et al. (2017b), and Wiles and Zisserman (2017), we also experiment with using only the silhouette in the loss (**silhouette**), disregarding differences in shading between the input and reconstructed pixels. To do so, we differentiably binarise both our reconstructed pixels I_0 and the ground-truth pixels $\mathbf{x}^{(i)}$ prior to calculating the reconstruction loss. Specifically, we transform each pixel p into $p/(p+\eta)$, where η is a small constant; after this modification to I_0, the Gaussian pyramid and pixel likelihoods are calculated as usual, *i.e.* following (3.2), (3.3) and (3.5).

- **Pose supervision:** Previous works that train for 3D reconstruction with 2D supervision require the ground-truth pose of each training instance (Yan et al., 2016; Wiles and Zisserman, 2017; Tulsiani et al., 2017b). Although our method does not need this, we evaluate whether it can benefit from it.

- **Multiple views:** Yan et al. (2016) and Wiles and Zisserman (2017) require that multiple views of each instance are presented together in each training batch, and Tulsiani et al. (2017b) also focus on this setting. Our model does not require this, but for comparison we include results with four views per instance at training time, and either one or four at test time.

During training, we construct each minibatch by randomly sampling 128 meshes from the relevant ShapeNet class uniformly with replacement. For each selected mesh, we render a single image, using a pose sampled from Uniform$(-\pi, \pi)$. Only these images are used to train the model, not the meshes themselves. In experiments using multiple views, we instead sample 32 meshes and four poses per mesh, and correspondingly render four images.

Network architectures. The decoder network F_ϕ takes the 12-dimensional latent embedding $\mathbf{z}$ as input. This is passed through a fully-connected layer with 32 output

channels using ReLU activation. The resulting embedding is processed by a second fully-connected layer that outputs the mesh parameters: vertex offsets for subdivision parameterisation, and locations, scales and rotations for the primitive-based parameterisations. For the primitive scales, we use a softplus activation to ensure they are positive; for the other parameters, we do not use any activation function.

The encoder network $\text{enc}_{\omega}(\mathbf{x})$ is a CNN operating on RGB images of size 128×96 pixels; its architecture is based on that of Wiles and Zisserman (2017). Specifically, it has the following layers, each with batch normalisation and ReLU activation:

- 3×3 convolution, 32 channels, stride = 2
- 3×3 convolution, 64 channels, stride = 1
- 2×2 max-pooling, stride = 2
- 3×3 convolution, 96 channels, stride = 1
- 2×2 max-pooling, stride = 2
- 3×3 convolution, 128 channels, stride = 1
- 2×2 max-pooling, stride = 2
- 4×4 convolution, 128 channels, stride = 1
- fully-connected, 128 channels

This yields a 128-dimensional feature vector for the image. The parameters for each variational distribution are produced by a further fully-connected layer, each taking this feature vector as input. For the mean of $\mathbf{z}$, we do not use any activation function; for the mean of θ_{fine} we use tanh activation, scaled by π/R to ensure θ_{coarse} rather than θ_{fine} is used to model large rotations. For the standard deviations of $\mathbf{z}$ and θ_{fine}, we use softplus activation, to ensure they are positive. Finally, for θ_{coarse}, we use a softmax output giving the probabilities of the different coarse rotations.

3.5.1 Generation

Figure 3.4 shows examples of meshes sampled from our model, using the same setting as Gadelha et al. (2017) (*i.e.* single-view training without pose supervision). Theirs is the only prior work that learns a 3D generative model with just images as supervision. We manually selected samples from our model that are stylistically similar to those displayed by Gadelha et al. (2017) to allow side-by-side comparison.

We see that in all cases, generating meshes tends to give cleaner, more visually-pleasing samples than voxels (as used by Gadelha et al., 2017). For *chair*, our model is able to capture the very narrow legs; for *aeroplane*, it captures the diagonal edges of the

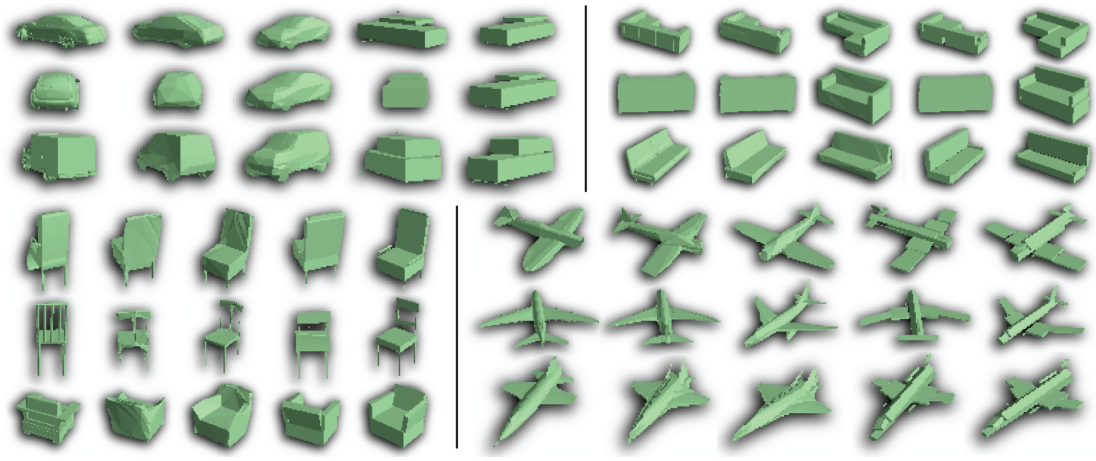


Figure 3.7: Qualitative examples of reconstructions. Each row of five images shows (i) ShapeNet ground-truth; (ii) our reconstruction with **subdivision** parameterisation; (iii) reconstruction aligned to canonical pose; (iv) our reconstruction with **blocks**; (v) aligned reconstruction. Experimental setting: single-view training, colour lighting, shading loss.

wings; for *car*, it captures the smoothly curved edges. We have also successfully learnt a model for the concave class *sofa*—which is impossible for Gadelha et al. (2017) as it does not consider shading. Finally, note that our samples are diverse: the model generates various different styles for each class.

3.5.2 Reconstruction

We now evaluate the performance of our model on 3D reconstruction from a single image. We benchmark on a held-out test set, following the protocol of Yan et al. (2016), where each object is presented at 24 different poses, and statistics are aggregated across objects and poses. We evaluate according to the following measures:

- *iou*: to measure the shape reconstruction error, we calculate the mean intersection-over-union between the predicted mesh and ground-truth (see Section 3.1.2); this follows recent works on reconstruction (Yan et al., 2016; Tulsiani et al., 2017b). To calculate this, we voxelise both meshes at a resolution of 32^3

- *err*: to measure the pose estimation error, we calculate the median error in degrees of predicted rotations

- *acc*: again to evaluate pose estimation, we measure the fraction of instances whose predicted rotation is within $\pi/6$ of the ground-truth rotation.

	car				**chair**			
	iou	*err*	*acc*	*iou* $\mid \theta$	*iou*	*err*	*acc*	*iou* $\mid \theta$
ortho-block	0.71	7.3	**0.84**	0.74	0.41	9.2	**0.69**	0.49
full-block	0.54	6.5	0.82	0.63	**0.46**	**4.6**	**0.69**	**0.51**
subdivision	**0.77**	**4.7**	**0.84**	**0.81**	0.39	7.9	0.65	**0.51**
	aeroplane				**sofa**			
	iou	*err*	*acc*	*iou* $\mid \theta$	*iou*	*err*	*acc*	*iou* $\mid \theta$
ortho-block	0.30	7.9	0.73	0.24	**0.59**	**7.3**	**0.94**	**0.69**
full-block	**0.51**	**4.4**	**0.89**	**0.57**	0.39	9.1	0.70	0.68
subdivision	0.49	6.7	0.64	**0.57**	0.39	14.7	0.52	0.59

Table 3.1: Reconstruction performance for four classes, with three different mesh parameterisations (Section 3.2). For each class, the first three columns are in the default setting of no pose supervision and correspond to the metrics in Section 3.5.2; iou $\mid \theta$ is the IOU when trained with pose supervision. Higher is better for *iou* and *acc*; lower is better for *err*. Experimental setting: single-view training, colour lighting, shading loss.

Note that the metrics *err* and *acc* are used to evaluate object pose estimation by Tulsiani and Malik (2015). These robust statistics are preferred to the mean error, as estimated rotations are frequently around 180° from the true rotation, due to the symmetry of many object classes

Object classes and mesh parameterisations. Table 3.1 shows the performance of our model on four different classes, comparing the three mesh parameterisations of Section 3.2. This focuses on our default setting of colour lighting, shading loss, single-view training without pose supervision (columns *iou, err, acc*); we also give *iou* when trained with pose supervision (column *iou* $\mid \theta$). We see that different parameterisations are better suited to different classes, in line with our expectations. Cars have smoothly curved edges, and are well-approximated by a single simply-connected surface; hence, **subdivision** performs well. Chairs vary in topology (*e.g.* the back may be solid or slatted) and sometimes have non-axis-aligned surfaces, so the flexible **full-block** parameterisation performs best. Aeroplanes have one dominant topology and include non-axis-aligned surfaces; both **full-block** and **subdivision** perform well here. Sofas often consist of axis-aligned blocks, so the **ortho-block** parameterisation is expressive

	car				chair			
	iou	*err*	*acc*	*iou*|θ	*iou*	*err*	*acc*	*iou*|θ
colour	**0.77**	**4.7**	**0.84**	**0.81**	**0.46**	**4.6**	**0.69**	**0.51**
white	0.58	13.8	0.82	0.81	0.25	33.6	0.49	0.42
colour + silhouette	0.46	65.2	0.29	0.64	0.28	51.7	0.35	0.48
	aeroplane				**sofa**			
	iou	*err*	*acc*	*iou*|θ	*iou*	*err*	*acc*	*iou*|θ
colour	**0.51**	**4.4**	**0.89**	**0.57**	**0.59**	**7.3**	**0.94**	0.69
white	0.42	7.7	0.85	0.54	0.51	56.1	0.49	**0.71**
colour + silhouette	0.20	17.8	0.57	0.47	0.27	89.8	0.15	0.57

Table 3.2: Reconstruction performance with different lighting and loss. *colour* indicates three coloured directional lights with shading loss; *white* indicates a single white directional light plus white ambient, with shading loss; *colour + silhouette* indicates coloured lighting with only the silhouette used in the loss. Our model can exploit the extra information gained by considering shading in the loss, and coloured directional lighting helps further. Experimental setting: single-view training, best mesh parameterisations from Table 3.1.

enough to model them. We hypothesise that it performs better than the other more flexible parameterisations as it is easier for training to find a good solution in a more restricted representation space. This is effectively a form of regularisation. Overall, the best reconstruction performance is achieved for cars, which accords with prior works (Tulsiani et al., 2017b; Yan et al., 2016; Fan et al., 2017).

The low values of *err* (and corresponding high values of *acc*) indicate that the model has indeed learnt to disentangle pose from shape. This is noteworthy given the model has seen only unannotated 2D images with arbitrary poses—disentanglement of these factors presumably arises because it is easier for the model to learn to reconstruct in a canonical frame, given that it is encouraged by our loss to predict diverse poses. However, providing the ground-truth poses as input improves reconstruction performance further in almost all cases (column *iou*|θ vs. *iou*).

Benefit of lighting. Table 3.2 shows how reconstruction performance varies with the different choices of lighting, **colour** and **white**, using **shading** loss. Coloured

	car				chair			
	iou	*err*	*acc*	*iou*|θ	*iou*	*err*	*acc*	*iou*|θ
single-view	0.77	4.7	0.84	0.81	0.46	4.6	0.69	0.51
4-view train, 4-view test	**0.83**	**2.6**	**0.94**	**0.86**	**0.51**	4.7	0.72	**0.55**
4-view train, 1-view test	0.81	5.1	0.93	0.83	0.46	**2.5**	**0.78**	0.50

Table 3.3: Reconstruction performance with multiple views at train/test time. Our model is able to exploit the extra information gained through multiple views, and can benefit even when testing with a single view. Experimental setting: best mesh parameterisations from Table 3.1, colour lighting, shading loss.

directional lighting provides more information during training than white lighting, and the results are correspondingly better. We also show performance with **silhouette** loss for coloured light. This performs significantly worse than with shading in the loss, in spite of the input images being identical. Thus, back-propagating information from shading through the renderer does indeed help with learning—it is not merely that colour images contain more information for the encoder network. As in the previous experiment, we see that pose supervision helps the model (column *iou*|θ vs. *iou*). In particular, only with pose supervision are silhouettes informative enough for the model to learn a canonical frame of reference reliably, as evidenced by the high median rotation errors without (column *err*).

Multi-view training/testing. Table 3.3 shows results when we provide four views of each object instance to the model. Using four views at both training and testing time improves results in all cases—the model has learnt to exploit the additional information about each instance. There is also a smaller performance improvement when we train with four views, but test with only one—although the network has not been optimised for the single-view task during training.

Comparison to previous works. Table 3.4 compares our results with previous works. Here, we conduct experiments in a setting matching Tulsiani et al. (2017b) and Yan et al. (2016): multiple views at training time, with ground-truth pose supervision. This shows that our results using meshes are roughly comparable with these previous works using voxels, even when only silhouette supervision is used (our results are worse on 'chair', better on 'sofa', and similar on the other classes). Furthermore, when we add shading information to the loss (which these previous works cannot), our results show

method	*lighting*	*loss*	**car**	**chair**	**aeroplane**	**sofa**
Tulsiani et al. (2017b)	*white*	*silhouette*	0.73	0.43	0.50	-
Tulsiani et al. (2017b)	*white*	*depth*	0.74	0.44	0.49	-
Yan et al. (2016)	*white*	*silhouette*	0.71	0.50	0.56	0.62
Yan et al. (2016), our images	*colour*	*silhouette*	0.66	0.22	0.42	0.46
ours	*white*	*silhouette*	0.71	0.25	0.53	0.68
ours	*white*	*shading*	0.79	0.44	0.54	**0.69**
ours	*colour*	*shading*	**0.83**	**0.51**	**0.57**	**0.69**
Fan et al. (2017)	*white*	*3D*	*0.83*	*0.54*	*0.60*	*0.71*

Table 3.4: Reconstruction performance (iou$|\theta$) in a setting matching Tulsiani et al. (2017b) and Yan et al. (2016) (multi-view training, and best parameterisations from Table 3.1) but with mesh output instead of voxels. *Yan et al. (2016), our images* is running the unmodified public code of Yan et al. (2016) with its normal silhouette loss, on our coloured images. The final row shows performance of a state-of-the-art method at time of publication (Fan et al., 2017) with full 3D supervision—note that our colour results are comparable with this, in spite of using only unannotated 2D images as supervision

a significant improvement; coloured lighting helps even further. We also show results for Yan et al. (2016) using our coloured lighting images as input, but their silhouette loss. This performs worse than our method on the same images, again showing that shading in the loss is useful—our colour images are not simply more informative to the encoder network than those of Yan et al. (2016). Interestingly, when trained with shading or colour, our method outperforms the method of Tulsiani et al. (2017b) even when the latter is trained with depth information. When trained with colour, our results are even close to those of Fan et al. (2017), which is trained with full 3D supervision, and was a state-of-the-art method when the work in this chapter was published.

3.6 Discussion and conclusions

In this section, we discuss various aspects of our model, then present some options for future research.

3.6.1 Analysis and discussion of our model

Together, our generative model plus its inference network resemble a variational autoencoder (Kingma and Welling, 2014) on pixels. It represents 3D shapes in a compact latent embedding space, and has extra layers in the decoder corresponding to the mesh representation and renderer. As we do not provide 3D supervision, the encoder and decoder must bootstrap and guide one another during training. The decoder learns the manifold of shapes, while at the same time the encoder learns to map images onto this. This learning process is driven purely by the objective of reconstructing the training images. While this is an ambiguous task and the model cannot guarantee to reconstruct the true shape of an object from a single image, its generative capability means that it always produces a plausible instance of the relevant class; the encoder ensures that this is consistent with the observed image. This works because the generative model must learn to produce shapes that reproject well over *all* training images, starting from low-dimensional latent representations. This creates an inductive bias towards regularity, which avoids degenerate solutions with unrealistic shapes that could, in isolation, explain each individual training image.

Analysis-by-synthesis. Our approach falls under the general umbrella of *analysis-by-synthesis* (Nair et al., 2008): we learn the discriminative task of reconstruction by starting with a generative model that can explain all available images in terms of some latent parameters, and we then learn to predict the latent parameters required to reconstruct a particular image. Other recent works following a similar route are those of Genova et al. (2018), which considers single-image reconstruction of faces, and Romaszko et al. (2017), which considers images of mugs. In our case, the latent parameters are the 3D shape and pose, disentangled from one another, with the shape placed in a canonical frame of reference. Unlike previous works, we also learn the parameters of the generative model itself, rather than merely learning to perform inference in it. In order for such approaches to be successful, it is necessary that the generative model is powerful enough to produce outputs matching the observed distribution. Our method works well on synthetic data where the ground-truth images are produced by a similar pipeline to the generative, but applying it to natural images is challenging; we discuss possible approaches to this below.

3.6.2 Future work

Mixed object classes. In the experiments of Section 3.5, we train one model for each class separately. However, it would be interesting to evaluate reconstruction performance when the model is trained on several classes jointly. This is a more challenging task, as there is more variability, and the model must avoid averaging or blending across the classes. In this case, it would also be reasonable to test reconstruction on objects of unseen classes; if the training classes are heterogeneous enough, the results may still be reasonable.

More degrees of freedom for object poses. In line with prior works on 2D-supervised reconstruction and generation (Tulsiani et al., 2017b; Yan et al., 2016; Wiles and Zisserman, 2017; Gadelha et al., 2017), we parameterise object poses with only the azimuth angle θ. The object is always assumed to be upright and centred in the image. However, it would be desirable to relax this restriction, as objects take much more diverse poses in reality. This involves a straightforward modification to the generative model: the additional parameters need to be incorporated in the object-to-camera transformation function T. Mathematically, it also only requires a small change during inference: the encoder should now output variational parameters for these additional degrees of freedom. However, we currently integrate over the pose space, to encourage the network to keep pose and shape disentangled. This quickly becomes intractable as more degrees of freedom are added, and so an alternative approach would be needed.

The very recent work of Tulsiani et al. (2018), published after ours was submitted, does investigate reconstruction and pose-estimation in roughly this setting. Specifically, they parameterise the object orientation with both azimuth and elevation. Moreover, in one experiment, they train with variable object positions, though evaluation is restricted to objects centred at the origin. Interestingly, they do not integrate over the entire pose space; instead they predict eight sets of pose parameters and a variational probability distribution over these. A similar approach could be applied in our method, to maintain tractability without losing pose diversity as the number of pose parameters increases.

It is worth emphasising that when translations are incorporated in the pose, our encoder network is effectively solving three problems jointly: object detection (to find the location), pose estimation (to find the orientation), and shape reconstruction. This suggests structuring the encoder in a progressive fashion, where the location is predicted first, and the rotation and shape conditioned on this. Thus, we have a *structured*

variational approximation, rather than mean-field.

Semi-supervised object poses. We currently assume that poses are either specified for all object instances or for none. It would be interesting to see whether annotating a very small number of instances with pose alleviates the additional inference complexity required in the unannotated case—in particular, to see whether the requirement for integrating over the space of poses could be dropped. This would be particularly beneficial if we add more degrees of freedom for pose, as we noted above that this makes the integration intractable. A few pose-annotated images would instead bootstrap the learning process, allowing it to form a canonical frame of reference without considering every possible pose for every image.

Support for natural images. In this chapter, we have focused on synthetic data, as this is easier to obtain and evaluate against. However, it would be interesting to apply a similar approach to natural images. Unfortunately, this is non-trivial with our current model. At a high level, the training objective of our method is to reconstruct *pixels*, albeit via a latent space that models 3D structure. In order to achieve good results on any given domain, it is therefore necessary that the generative has the modelling capacity to reproduce images in that domain. Moreover, it must do so while ensuring the latent features we care about (*e.g.* object shape) are reproduced correctly—without being entangled with confounding features. In the case of reconstruction from natural images, this means that lighting and surface properties such as albedo cannot be ignored, as they interact with the object shape in producing the final pixels, and a failure to accurately model them will cause the shape itself to be inferred wrongly. Indeed, unless segmentations are available, then the model must correctly account for the background itself, *i.e.* not attempt to model it as part of the mesh.

More concretely, a first step towards modelling natural images is to add colour to the model: the colour of each surface of the object should be sampled along with the shape, from a learnt prior distribution. For typical object classes, we expect the shape and colour not to be independent. Hence, we model both the shape *and* the colour as dependent on the latent embedding $\mathbf{z}$. Just as the mesh parameters are produced by a decoder network taking $\mathbf{z}$ as input, the face colours are produced by a second decoder, also taking $\mathbf{z}$ as input. Given the face colours, it is a straightforward extension to the rendering function $\mathcal{G}$ to generate the corresponding images.

We have conducted preliminary experiments applying this model to frames from traffic surveillance cameras (Sochor et al., 2018). This domain was chosen because

Figure 3.8: Examples of single-image reconstruction on natural images containing multiple objects. The top row shows original frames from Sochor et al. (2018); the bottom row shows their reconstructions using the adapted model described in Section 3.6.2. The model must learn to reconstruct both shape and colour, without supervision.

(i) plentiful images are available, showing objects belonging to just two classes (car and truck), (ii) camera calibration is straightforward, and (iii) the background is fixed. The latter is important, as otherwise it is very difficult for the model to learn whether background pixels near the object are in fact part of the object, but of a different colour. We make several other modifications to our model:

- object orientations are fixed based on the position (*i.e.* which lane the vehicle is in)
- object positions are sampled uniformly over the lane area; the variational distribution for positions is parameterised not by a learnt encoder, but by a deterministic function based on background-subtraction and blob-detection
- the renderer $\mathcal{G}$ is extended to support the soft shadows cast by vehicles onto the road
- the L2 reconstruction loss on pixels is replaced by the perceptual loss MS-SSIM (Wang et al., 2003), which we found to give better results

Figure 3.8 displays some preliminary reconstruction results using this method.

3.6.3 Concluding remarks

We have presented a framework for generation and reconstruction of 3D meshes. Our approach is flexible and supports many different supervision settings, including weaker

supervision than any prior works (*i.e.* a single view per training instance, and without pose annotations). Unlike prior works, we can exploit shading cues due to directional lighting; we have shown that this improves performance over silhouettes. Moreover, performance is higher than that of a method with depth supervision (Tulsiani et al., 2017b), and even close to the state-of-the-art results using full 3D supervision (Fan et al., 2017). Notably, ours is the first method that can learn a generative model of 3D meshes, trained with only 2D images. We have shown that use of meshes leads to more visually-pleasing results than prior voxel-based works (Gadelha et al., 2017). Finally, we have shown encouraging preliminary results on applying our framework to natural images.

Chapter 4

Generating constrained room layouts

Humans spend a large fraction of their time in indoor environments, which have therefore received significant attention in both computer vision and computer graphics over the years. In both fields, it is natural to consider probabilistic models over such environments—in this chapter, we specifically consider modelling 3D layouts of furniture and other objects in rooms of houses or apartments.

In computer vision, a strong probabilistic model of room layouts is valuable as a prior for indoor scene understanding tasks. Here the goal is to predict the classes, locations, and orientations of objects in 3D space, given an image of a room. By capturing global, high-order relations between object instances, a model of layouts can be used to guide prediction, by reasoning over all objects simultaneously to ensure a globally-consistent and semantically-plausible layout. Moreover, a model that produces realistic layouts may also be used to synthesise training data for discriminative methods that would otherwise require expensive manual annotation of images.

In computer graphics, manually authoring environments is expensive and time-consuming; hence, there is a need for algorithms that can automatically generate realistic scenes. One approach here is to draw samples from a probabilistic model over layouts. For such algorithms to be useful, it is important not only that they produce realistic layouts, but also that they can accommodate constraints stipulated by domain experts.

In this chapter, we propose a new data-driven, probabilistic generative model for 3D room layouts (Figure 4.1). Our model learns occurrence and placement statistics from a database containing over 250 000 human-designed rooms and 2500 CAD models (Song et al., 2017). Drawing a sample is extremely fast, and results in a complete room layout, including furniture (*e.g.* tables, chairs), smaller objects (*e.g.* books, lap-

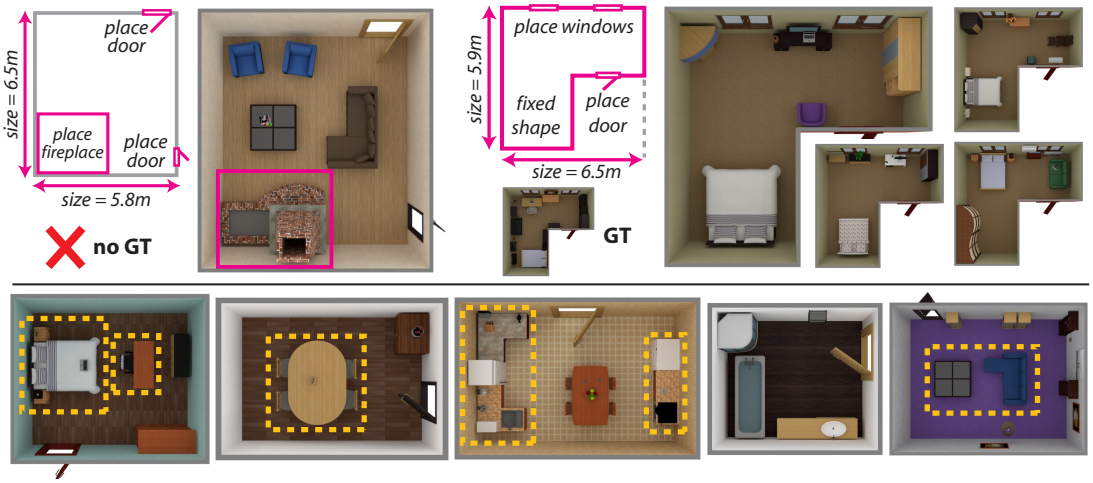


Figure 4.1: We present a generative model for room layouts, which reasons over high-order relations among objects. It also accommodates a variety of user-specifiable constraints such as room shape and size, and object placement. Our model is able to generate a valid layout in cases (top left) where there is no layout in the training set which satisfies the given size and placement constraints. We visualise four samples generated by our model for the L-shaped bedroom, along with the only human-designed room (GT) in the training set that satisfies the constraints. Our model supports efficient unconstrained sampling (0.04s per room).

tops), lamps, shelves, and other objects hung from the ceiling and walls. We show by extensive human evaluation that these sampled layouts are plausible, *i.e.* appear to be real apartment layouts, and are also novel, *i.e.* the model does not simply memorise its training set. Moreover, our approach is amenable to a variety of *a priori* constraints.

Learning from thousands of rooms arranged by humans allows us to distill their designers' knowledge into our model. By sampling from it, we can then generate novel layouts that retain the characteristics preferred by human designers. The model reasons over complex layouts in 3D space, incorporating high-order spatial relations between objects. For example, it captures the fact that chairs are often positioned around a table and facing it, or that books may be placed on a shelf or desk, but not on a sink.

Probabilistic modelling of room layouts is a challenging task. The output space is a high-dimensional combination of discrete and continuous variables, such as object classes and locations. There are strong couplings between many of these: for example, the size of a room, and the numbers, sizes, and locations of the objects in it, are linked through the requirement that all objects fit without intersection with each other or the

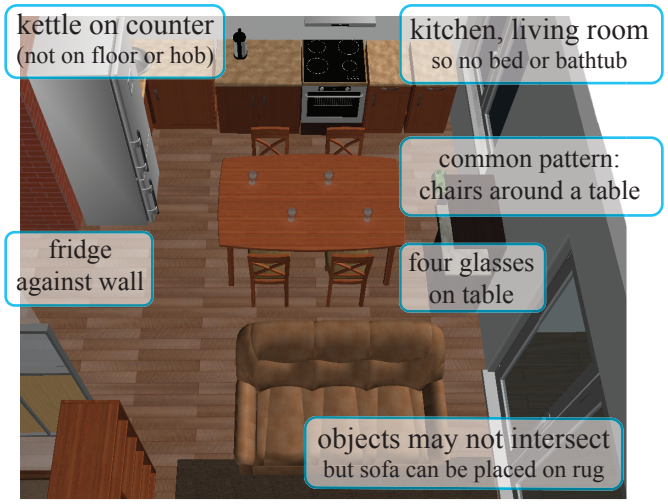


Figure 4.2: An example room, showing various constraints and relations that a model over layouts must capture. The layout and CAD models are from the SUNCG dataset (Song et al., 2017)

walls, and that certain object classes are more likely to appear in close proximity than others (Figure 4.2). To mitigate this, we adopt the framework of directed graphical models, as these admit efficient, ancestral sampling (Bishop, 2006). The result is that a single sample from our model can be computed in a fraction of a second, placing a large number of objects in a natural layout. This efficiency allows us to use rejection sampling to incorporate user-specified constraints, such as accessibility, sizes and shapes of rooms, locations of doors and windows, and constraints on the location of furniture items such as sofas and television screens.

Although several algorithms have been proposed to generate room layouts, none of them is fully automatic, considers inter-object relationships, and is amenable to a diverse range of user-specified constraints (Section 4.1.2). Most automatic methods are limited to simple constraints such as room type, size and the presence of a particular object class. Others are formulated as undirected graphical models, which are costly to sample from and therefore do not scale. Incorporating additional constraints in any of these models is complicated if not impossible.

Chapter overview. In Section 4.1, we give background on generating environments, with a focus on the task of modelling room layouts. Section 4.2 and Section 4.3 describe our new generative model, and how it is trained. Section 4.4 discusses the mechanism by which user-specified constraints may be imposed. We present results from our method in Section 4.5, and conclude in Section 4.6.

The work described in this chapter is new material, that is not yet published.

4.1 Background and related work

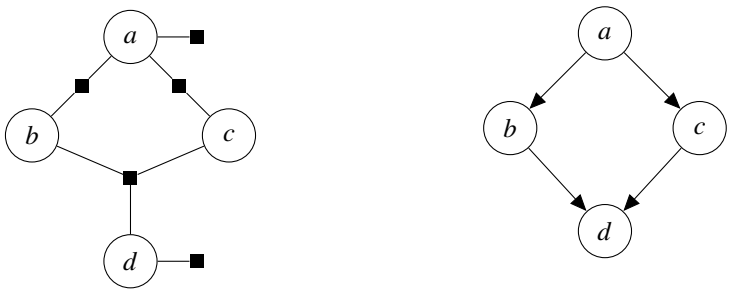
The computer graphics literature is rich with methods that enable 3D content creation, from landscapes (Cordonnier et al., 2017) and urban sprawls (Parish and Müller, 2001) to individual objects such as furniture (Li et al., 2017) and buildings (Müller et al., 2006; Nishida et al., 2016). These methods involve varying degrees of user interaction to achieve realism and/or aesthetic appeal. Procedural approaches rely on parametric controllability (*e.g.* Müller et al., 2006; Cordonnier et al., 2017) while methods that are posed as optimisation rely on constraint-driven controllability (*e.g.* Merrell et al., 2011; Yeh et al., 2012). A third class of methods adopts a *data-driven* approach to generate or edit scenes based on features learnt from training examples (*e.g.* Funkhouser et al., 2004; Emilien et al., 2015; Li et al., 2017). Our own work falls into this latter category, and adopts the framework of *probabilistic graphical models*, in line with several previous techniques modelling room layouts (see Section 4.1.2). There are also several works in computer vision that use similar techniques, not to produce realistic-looking scenes, but to capture the regularities of object placement expected in rooms, and use this to guide scene understanding tasks such as 3D layout estimation (*e.g.* Del Pero et al., 2012; Zhao and Zhu, 2013).

In this section we give background on probabilistic graphical models, focusing on their application to environment generation. Then, we provide details on prior works in vision and graphics that apply these techniques to modelling room layouts.

4.1.1 Probabilistic graphical models

A *probabilistic model* is simply a joint distribution over some set of random variables: $P(x_0, x_1, \ldots, x_n) = P(\mathbf{x})$. Probabilistic models are often represented by a graph having the random variables as nodes. This graphical model may be either *directed* or *undirected* (Bishop, 2006).

Undirected graphical models. These are sometimes known as energy-based models, and decompose the joint probability distribution into a set of *factors*. Each factor is a function of some subset of the variables, know as its *clique*. We can therefore write $P(\mathbf{x}) = \prod_s \phi_s(\mathbf{x}_s)$, where s indexes factors, and $\mathbf{x}_s$ is the subset of the variables $\mathbf{x}$ in clique s. The most natural graphical representation is then a *factor graph*, a bipartite



(a) (b)

Figure 4.3: Examples of simple undirected and directed graphical models. **(a)** Undirected graphical model, represented as a factor graph, for the distribution $P(a,b,c,d) = P(a)P(d)P(a,b)P(a,c)P(b,c,d)$. **(b)** Directed graphical model, or Bayesian network, representing the distribution $P(a,b,c,d) = P(a)P(b|a)P(c|a)P(d|b,c)$.

graph with one set of nodes for factors, and another for variables Figure 4.3a). Undirected edges connect each factor to the variables that it is a function of. Undirected models are challenging to sample, as the factors introduce complex interdependencies among variables. Hence, Markov chain Monte-Carlo (MCMC) approaches are typically used, such as the Metropolis-Hastings algorithm (Metropolis et al., 1953). Starting from an initial setting of the variables, these move progressively through the space of possible samples, following some proposal distribution over 'nearby' states, and choosing whether to accept each. They are guaranteed to provide an unbiased sample from the underlying distribution, but are computationally expensive. Constrained samples, where certain variables are fixed, may be generated using the same approach. Undirected models have been used in several prior works on room layout generation, which we describe in Section 4.1.2. In addition, they have been applied to generating golf courses (Yeh et al., 2012), and wall plans of houses (Merrell et al., 2010).

Directed graphical models. These are sometimes known as Bayesian networks, and decompose the joint probability distribution into a set of conditional distributions, of each variable given some others. We can therefore write $P(\mathbf{x}) = \prod_{i=1}^{n} P\left(x_i \,|\, \mathbf{x}_{Pa(i)}\right)$, where i indexes variables, and $Pa(i)$ indicates the 'parent' variables on which the i^{th} is conditioned. Again, the graph representation has one node per variable, but now with directed edges indicating the direction of the conditional dependency, forming a directed acyclic graph (Figure 4.3b). The conditional dependencies are often defined to reflect causal structure in the underlying problem. Directed models may be sampled

very efficiently by ancestral sampling, whereby each variable is sampled conditional on its parents in the graph. However, constrained samples, where certain variables are fixed, are harder to generate if the variables to be fixed have parents in the graph. In this case, we must infer the allowable states for their ancestor variables, which is challenging and often computationally expensive. Directed models have not been applied widely to environment generation, with two notable exceptions. Merrell et al. (2010) use one to generate architectural briefs, which are subsequently converted to wall plans via an undirected model. Fisher et al. (2012) use one to generate a support hierarchy and classes for objects on a table/desk; again, an undirected model is used afterwards to resolve the exact object positions.

Stochastic grammar models. These are a prominent approach in procedural generation, where structures are defined in terms of a set of production rules over symbols. Starting from an initial symbol, generation proceeds recursively; for each symbol, a production rule is selected at random, and the symbol is replaced by others specified in that production rule. Finally, the resulting symbols are directly translated into scene geometry. Such models bear a strong resemblance to directed graphical models; indeed, there is a direct correspondence when the set of structures that can be generated by the grammar is finite. However, works using stochastic grammars disregard the joint distribution they imply, and do not consider questions of inference (with the notable exception of Talton et al. (2011)). These models have been used to generate various structures, for example buildings (Wonka et al., 2003; Müller et al., 2006) and plants (Prusinkiewicz and Lindenmayer, 2004).

4.1.2 Models of room layouts

Existing research on models for room layouts can be split into the two domains of vision and graphics. In vision, the focus is on defining a prior for scene understanding, so while the model captures important features of a layout, its samples need not be fully realistic (*e.g.* using blocks in place of detailed CAD models). In graphics, the focus is on sampling layouts for use in visual effects, game level design, or user-assistance for interior design—this imposes stronger requirements for the results to be realistic.

Priors for understanding structure in indoor scenes. Several works aim to recover the layout of indoor scenes, typically including wall planes and geometry of large block-like elements, while incorporating probabilistic priors. These priors are hand-engineered, with just a few learnt parameters such as mean object sizes. Although these

models are designed for the discriminative task of layout estimation for a given image, it is theoretically possible to sample layouts unconditionally from the prior—however, none that we are aware of show examples of this.

Zhao and Zhu (2011, 2013) build a probabilistic grammar model over cube-based 3D structures constituting parts of rooms, including seven object classes. The grammar model defines a hierarchy of conditional variables, starting with the room type (*e.g.* living room), then functional areas (*e.g.* a sitting area), then furniture object classes (*e.g.* sofa), then parts (*e.g.* legs, back); this part of the model is hand-engineered rather than learnt. Then, an additional learnt layer models the distribution of locations and sizes of the object parts, treating these as cuboids. This model allows sampling, but yields coarse arrangements of blocks, not realistic scenes, and does not model entire rooms, only areas that would be visible in a single image. When inferring the most-probable parse graph for a given image, they first detect edges. Then, simulated annealing is used to find the parse graph that best explains the observed edges.

Similarly, Del Pero et al. (2012) propose a generative model over room size and layout, treating objects as cuboids. Their model learns per-class prior distributions for the cuboid dimensions, as well as the frequency with which instances of each class are placed against a wall. It does not capture inter-object relationships such as co-occurrence or relative locations. In order to infer the most likely configuration for a given image, edge and surface-orientation cues are used as input to an MCMC framework. Del Pero et al. (2013) extend this to incorporate manually-designed probabilistic models of object classes, again in terms of blocks, capturing their relative locations and sizes. They mention the value of inter-object relations, but do not incorporate them in the generative model. These models again support sampling from their priors, but this only yields arrangements of blocks, not realistic scenes.

Choi et al. (2013) perform joint scene classification, 3D object localisation and classification, and layout estimation. They use an energy-based formulation, with a prior that incorporates spatial relations between objects. This includes terms favouring frequently-occurring patterns of objects, which are learnt from a training set of 960 rooms. Again, this method does this not model entire rooms, only regions visible to a camera; objects take the form of 3D bounding boxes with an associated class. Very recently, Hueting et al. (2017) present a method specialised for detecting heavily-occluded chairs in indoor scenes. To do so, they define a prior on arrangements of chairs, that incorporates simple spatial co-occurrence relationships.

Generating realistic furniture layouts. Several works have investigated generating

	Wang (2018)	Qi (2018)	Fu (2017)	Liang (2017)	Sadeghipour (2016)	Handa (2016)	Yu (2011)	Merrell (2011)	ours
# room types	3	10	8	5	1	5	7	4	9
whole room	✓	✓	✓	✓	✓	✓	✓	✓	✓
inter-object relations	high-order	high-order	high-order	pairwise	high-order	pairwise	pairwise	pairwise	high-order
fully automatic	✓	✓	-	✓	✓	✓	-	-	✓
small objects	-	-	✓	-	-	✓	-	-	✓
model type	CNN	undirected	exemplar	hybrid	undirected	undirected	undirected	undirected	directed
room type	✓	✓	-	✓	✓	✓	-	-	✓
room size	✓	✓	✓	✓	✓	✓	✓	✓	✓
room shape	✓	-	-	-	-	-	-	-	✓
traversability	-	-	-	-	-	-	✓	✓	✓
object existence	-	-	✓	-	-	-	✓	✓	✓
object placement	-	-	-	-	-	-	-	✓	✓

Table 4.1: Comparison of works generating full room layouts. **Top:** Characteristics of different methods. 'Fully automatic' methods support generation without user specification of any furniture classes. 'Small objects' means these are also sampled, rather than just furniture'. 'Hybrid' for model means a directed model for object classes followed by undirected for positions; 'exemplar' means a non-parametric exemplar-based method for arrangement. **Bottom:** Constraints supported by each method.

realistic room layouts, either from scratch, or rearranging user-selected furniture items. Features of these methods are summarised in Table 4.1. Most of these works formulate an undirected graphical model (Merrell et al., 2011; Yu et al., 2011; Sadeghipour Kermani et al., 2016; Handa et al., 2016; Qi et al., 2018). This defines a joint distribution over variables that represent the scene—*e.g.* object locations and classes, room size, *etc.* Factors are added to impose constraints and preferences, such as (i) enforcing non-intersection of all objects; (ii) favouring certain objects being near others (*e.g.* a chair near a table); (iii) ensuring that a human can traverse the room. This leads to models with tight couplings between many variables, which makes them difficult to sample efficiently, limiting their scalability.

Handa et al. (2016) generate room layouts by optimising a pairwise energy using simulated annealing, starting from a random arrangement. This is not guaranteed to converge to a valid layout, particularly when there are many objects present, nor is it an unbiased sample from the Gibbs distribution induced by the model energy. Their model incorporates larger furniture items on the floor, but does not encompass smaller objects positioned on the furniture, nor objects mounted on the walls or ceiling (such as shelves or lamps). Sadeghipour Kermani et al. (2016) propose a similar method, but separate the sampling of classes and counts from the spatial arrangement. They learn several kinds of spatial relationships from an annotated database of RGB-D images (Song et al., 2015), including support, proximity, and relative pose. These are incorporated in a factor graph model that is manually-designed and specific to the one room type considered (bedrooms). Samples from this model are generated using a progressive (object-by-object) approach based on MCMC sampling, followed by a rejection step that discards implausible layouts (*e.g.* those with intersecting objects). Liang et al. (2017) also propose a two-step probabilistic generative model. The counts of different object classes are sampled first; they are modelled using LDA (Blei et al., 2003), with topics corresponding to the room type. Then, the selected instances are placed, by MCMC sampling of their locations from a pairwise undirected model. This captures simple inter-object relations such as relative location, but does not support higher-order relations. The model is trained using a proprietary dataset of around 1000 rooms, and is demonstrated on five room types.

Very recently, Fu et al. (2017) propose a different approach to the layout-generation problem. They require that the user specify one or more object classes that are present in the room; these are used to infer the likely room type. Then, more objects are automatically added, by considering typical human interactions with those already

present (*e.g.* presence of a coffee table implies a likely seating area, hence a sofa should be added). To position the objects, this method first finds similar rooms in a large database, seeking those with similar shape and door/window locations. Then, it uses a weighted combination of these layouts to define high-probability locations for large objects (*e.g.* beds), with smaller objects (*e.g.* nightstands) being deterministically placed adjacent to them. This method does not support fully-automatic generation without any user input, nor constraints such as traversability and placement of objects at a specific location.

Instead of generating whole room layouts, Fisher et al. (2012) propose a method that synthesises small areas of rooms by sampling a focal object (*e.g.* a table), then adding relevant objects around it (*e.g.* chairs) as well as on it (*e.g.* books). Similarly to that of Liang et al. (2017), this method operates in two stages. It first samples object classes in a support hierarchy, from a directed graphical model. Then, the locations of the objects are sampled using MCMC from an undirected model. The model is trained with only 130 layouts, and so a focus is on making optimal use of this data. Ma et al. (2016) also generate small areas of rooms by adding objects around a focal object, but they decide the classes and locations of objects to add by reasoning over human actions that could take place in the scene.

Instead of generating layouts from scratch, Merrell et al. (2011) and Yu et al. (2011) propose systems that suggest layout improvements, based on a user-provided set of objects and an initial layout. In both cases, they use undirected models, supporting pairwise relations between objects. They augment these relations with various manually-specified aesthetic and functional constraints, such as symmetry and clearance. Merrell et al. (2011) use a GPU-accelerated implementation of Metropolis-Hastings sampling (Bishop, 2006), while Yu et al. (2011) resort to simulated annealing (Kirkpatrick et al., 1983) starting from a user-specified arrangement. Merrell et al. (2011) consider only rooms in residences, while Yu et al. (2011) also model various commercial and industrial spaces. The evaluation of both of these works focuses on how easily they allow non-expert users to create high-quality layouts. Fisher et al. (2015) instead take a 3D scan of an existing room as input, and rather than rearranging the objects, suggest new layouts that support the same actions (*e.g.* working at a desk) at similar locations.

Concurrent works. Two new methods have been published while we were preparing the work presented in this chapter. Both propose data-driven strategies to generate indoor furniture layouts. Qi et al. (2018) represent indoor scenes using a probabilistic grammar, with a Gibbs distribution as prior on its parse graphs. Each parse graph rep-

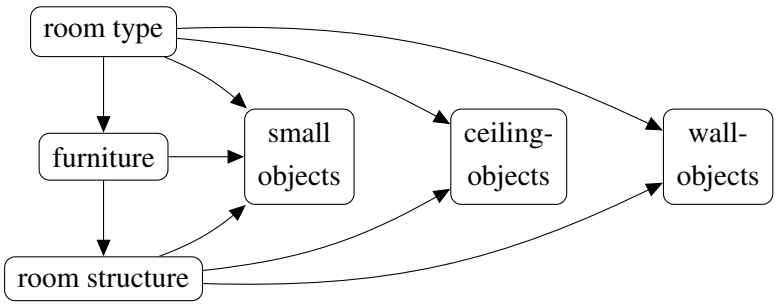


Figure 4.4: Elements of our generative model with arrows representing conditional dependencies. Different categories of object (middle row) are sampled conditional on the room type; the room structure (*i.e.* sizes of the cells) is defined by the furniture it contains.

resents a simple decomposition of a room into objects. The probability of a given parse graph depends on the probability of this hierarchical decomposition, and on additional potentials encoding inter-object relations (*e.g.* functional groups and support). This approach requires considerable manual modelling, including specification of potential functions and grouping relations between objects such as chairs and tables. Layouts are sampled using MCMC sampling, combined with simulated annealing. Jiang et al. (2018) demonstrate the application of this model to generating synthetic training data with pixel-perfect ground-truth for tasks such as depth and normal prediction.

Wang et al. (2018) exploit the power of convolutional neural networks (CNNs). This method generates layouts sequentially, starting from an empty room. At each step, three different CNNs decide (i) whether to add furniture, (ii) what furniture to add, and (iii) where to place it. The CNNs all take the current floor-plan as input, and are trained separately for their respective sub-tasks. This approach avoids a costly MCMC optimisation process, but still requires heavy computation, taking several minutes on a GPU to sample a single room, and several days to train the models. Both Qi et al. (2018) and Wang et al. (2018) train with the same dataset that we use in our work (Song et al., 2017). While both methods are fully automatic, neither of them supports constraints such as object class presence and placement.

4.2 Generative model

We now describe our probabilistic model for room layouts. In Section 4.3, we describe how it is trained, and in Section 4.4, how user-specified constraints may be imposed.

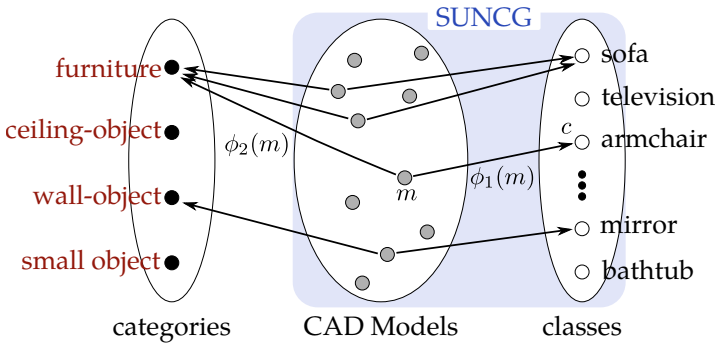


Figure 4.5: We introduce a set of *categories* (left) along with a manually-specified mapping ϕ_2 from models to categories. The SUNCG database already specifies a mapping ϕ_1 from models to object classes. We use both mappings.

Our generative process for sampling rooms is illustrated in Figure 4.4. We begin by sampling a *room type* (*e.g.* kitchen, living room), then sequentially sample furniture instances, conditioned on the room type and instances already sampled (Section 4.2.2). These define the sizes of *cells* they are placed in (Section 4.2.1), and hence the *room structure*. Then, in an *embellishment* step (Section 4.2.3), we sample instances of other classes, given the furniture items and their locations.

The parameters used for sampling the classes and positions of object instances are learnt from the dataset (Section 4.3). They define simple parametric models, which are human-interpretable and modifiable.

Classes and categorisation. The SUNCG dataset (Song et al., 2017) contains CAD models, where each model m is assigned an *object class* $\phi_1(m)$ such as 'television', 'bathtub', 'armchair', *etc.* There are a total of 2500 CAD models and 170 object classes in the collection. We introduce an additional set of labels called *categories*: furniture, small objects such as books and laptops, wall objects such as picture frames, and ceiling objects such as lamps. We manually specified a second mapping ϕ_2 from models to categories (Figure 4.5). Our sampling strategy for models differs between the different categories, as discussed in the next sections.

4.2.1 Cells

In order to sample layouts efficiently without introducing intersections, we divide rooms into *cells*; each item of furniture we place is assigned to one cell, *e.g.* 'next to the east wall'. Figure 4.6 shows the standard cell structure that we use for most rooms. This has nine cells: four corners, four edges, and one interior. Corner cells

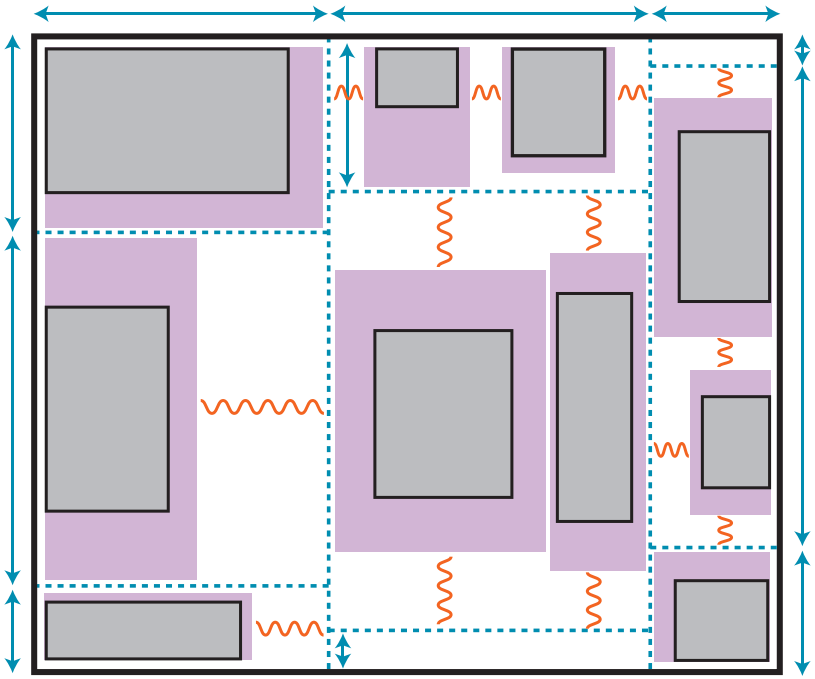


Figure 4.6: Cell structure for furniture layout. Thick black lines represent the walls enclosing the room, grey boxes represent furniture objects, dashed blue lines delimit cells and blue arrows indicate dimensions that expand/contract to fit their contents. There is one cell along each edge of the room, one at each corner, and one in the interior. Objects are padded with free space on each side (purple). Objects in cells that are larger than the sum of their contents, due to constraints due to neighbouring cells, are distributed evenly (orange springs), with the exception that objects in corner/edge cells always remain flush with walls.

are restricted to contain a maximum of one object each; edge and interior cells may contain multiple objects, and these are placed next to each other in a line. Cells are flexibly-sized regions, and expand to fit the objects that are assigned to them, plus sampled padding distances around each. When a cell is larger than needed to accommodate its contents, due to another cell which shares an edge being larger, the spare space is divided evenly between objects (orange springs in the Figure 4.6). For smaller rooms with all objects arranged around the walls (*e.g.* bathrooms), we omit the interior cell.

Note that the cell structure imposes a strong prior on the layouts we generate. For example, it captures the fact that objects are often placed along walls, without the model having to learn this itself—it does however learn *which* object classes are likely to be in edge cells, as opposed to corner or interior cells.

We can also adapt the cell structure to incorporate different prior knowledge. For

one room type, living rooms, we use slightly different cell structure, that better captures the functional layout of typical living rooms. Specifically, to form a natural area for conversation, we define cells for sofas, chairs and side-tables, around a focal item such as a coffee table or fireplace; the sampling process is restricted to only allow appropriate object classes in each cell.

4.2.2 Sampling furniture

We place furniture instances by sampling counts of *singletons* (*i.e.* individual objects), and of *motifs* and *abutments*, which are spatially-related groups learnt from the training data. For each instance, we sample an orientation and padding, and a cell in the layout it is to be assigned to. After all furniture counts and instance parameters have been sampled, we position the resulting objects deterministically such that all cells accommodate the objects and their paddings.

Singletons. We sample singletons using Algorithm 1, where the functions SampleNumInstances, SampleCell, SampleOrientation and SamplePadding sample from distributions whose parameters are learnt from training data. The arguments to these functions signify what the underlying distributions are conditional on.

Motifs. Motifs are groups of furniture items that are present together in many rooms in the dataset, such as a table with chairs around it. We sample counts and instance parameters for each motif following lines 5–10 of Algorithm 1, where m in this case represents a motif rather than a singleton CAD model. Then, we set the relative offsets and orientations between items within a motif as observed in a randomly selected instance of the motif in the training database. This non-parametric strategy for determining relative placement eliminates the need for testing if there are geometric collisions/intersections.

Abutments. Abutments are groups of items that appear in rows, abutting one another, with variations in their sequence, *e.g.* a row of cabinets in a kitchen interspersed with appliances such as a dishwasher, sink, or refrigerator. Again, we sample counts and instance parameters for each abutment using lines 5–10 of Algorithm 1, where m now represents a class of abutment. The furniture items within an abutment are modeled as a Markov chain with a terminal state; for each instance of the abutment, we sample from this Markov chain to obtain a specific sequence and number of CAD models. The transition probabilities of the Markov chains are learnt during training.

Algorithm 1 Sampling singleton furniture instances

Input: r is the room type

1: **function** SAMPLEFURNITURE(r)
2: **for each** object class c **do**
3: $n_c \leftarrow 0$
4: **for each** model m **with** $(\phi_1(m) = c) \wedge (\phi_2(m) = \textit{furniture})$ **do**
5: $n_m \leftarrow$ SampleNumInstances(m, n_c, r)
6: **for** $j \leftarrow 1$ to n_m **do** ▷ n_m instances of m
7: $k_j \leftarrow$ SampleCell(m)
8: $\theta_j \leftarrow$ SampleOrientation(m, k)
9: $p_j \leftarrow$ SamplePadding(m)
10: **end for**
11: $n_c \leftarrow n_c + n_m$ ▷ accumulate count
12: **end for**
13: **end for**
14: **end function**

Output: for each CAD model m:
(i) number of instances n_m that we place;
(ii) parameters $\{k_j, \theta_j, p_j\}_{j=1}^{n_m}$ of each instance of m.

4.2.3 Embellishment

Ceiling objects. Given the room type r, we draw a single CAD model m randomly according to a discrete probability mass function (PMF) $\theta_{c.o.}$ over models in this category. The number of instances of m is determined using SampleNumInstances$(m, 0)$. This number is rounded up so that it can be factorised into a product of integers, and the objects are positioned on a regular grid.

Wall objects. For each CAD model m that is a wall object, we determine the number of instances in similar fashion to furniture in lines 2–5 of Algorithm 1. Each instance is then assigned to a wall uniformly randomly and its position on the wall is a combination of a Normal distribution along the Y axis and Uniform distributions in X and Z. The parameters of the Normal distribution are learnt (conditioned on m). If this results in a geometric conflict (intersection with other wall objects, doors, *etc.*) we reject the sampled location and repeat the process until there are no conflicts.

Small objects. For each furniture instance with CAD model m, we sample small objects non-parametrically conditioned on m and r. We choose a random instance of m in a room of type r from the database, and replicate the configuration of small objects associated with that instance.

4.2.4 Algorithm summary

To summarise, we first randomly sample the type of room r from a discrete distribution over nine room types found in the database. The distribution (PMF) of r is learnt during training. Then, we sample furniture items conditioned on r: first singletons, then motifs and finally abutments. The counts and instances of each item are determined by parameters learnt during training. Once all furniture items are sampled, and assigned to cells, we use a deterministic placement algorithm that calculates their final positions in the room. Then, we sample ceiling objects and wall objects conditioned on r and the furniture placed. Finally, we sample small objects conditioned on the furniture and r.

4.3 Training

Dataset. We use a large dataset of ground-truth room layouts to learn the parameters of our model. SUNCG (Song et al., 2017) is a new dataset of 45000 house/apartment layouts, created by humans, and separately verified as reasonable by humans. Each layout has an average of 8.1 rooms; the majority are annotated with the room type. The layouts are designed with an online tool, and contain objects of 170 classes, represented by around 2500 CAD models. There are 4.5M object instances; each consists of a reference to a CAD model, and its location and orientation in the room.

Number of instances. We model the number of instances n_m of each CAD model as being conditional on the model m, the room type r, and the number n_c of furniture instances already sampled for the class $\phi_1(m)$. The distribution (PMF) $\theta_{n.m.}$ over count bins $\{0, 1, 2, 3, 4, > 4\}$ is calculated as a normalised histogram over all scenes in the database of type r. Further, a Poisson distribution is fitted to the observed n_m in all scenes with $n_m > 4$. SampleNumInstances (line 5 of Algorithm 1) is implemented in two steps. First, we draw an indicator variable according to $\theta_{n.m.}$. If this variable is less than or equal to four, then we return it as the number of instances. Otherwise, we return a sample from the Poisson distribution.

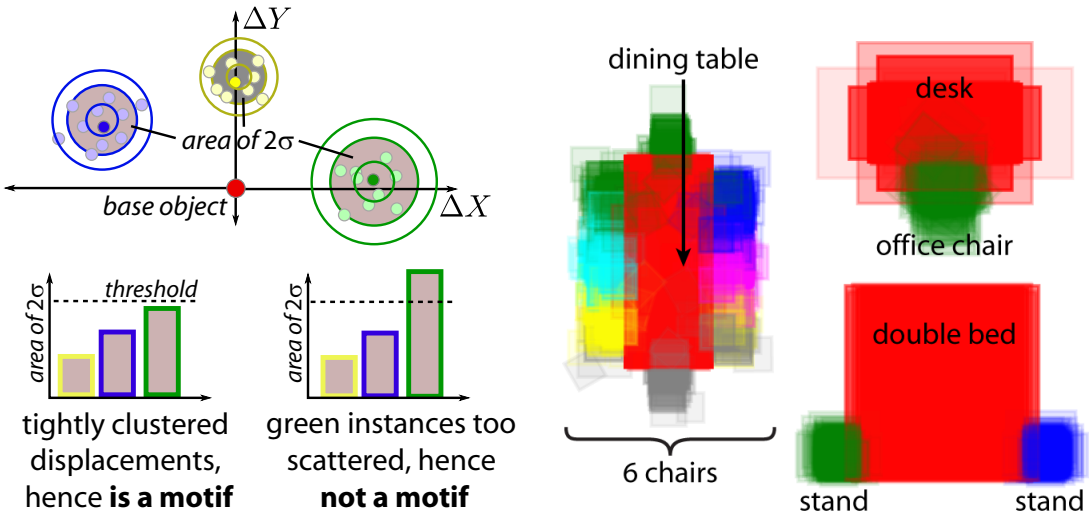


Figure 4.7: Motif discovery. **Left:** We use DPMM clustering for K-tuples of object occurrences, and identify motifs as those clusters for which the standard deviation of displacements from a base object in the tuple is within some threshold. **Right:** Examples of motifs that we automatically discover in SUNCG. Each colour corresponds to a different object in the pattern; we overlay 200 occurrences of each pattern to illustrate its variability. The red objects are the base objects of the patterns.

Instance attributes. For each model m, during training, we calculate a PMF over nine cells (four corners, four edges, and interior) by normalising the histogram of occurrences. We implement SampleCell (line 7 of Algorithm 1) by returning a cell according to this PMF. For models in interior cells, we count the number occurrences where they are aligned (positively or negatively) with respect to any axis and the number of 'non-aligned' instances, and use this to learn a PMF. We implement SampleOrientation (line 8 of Algorithm 1) by sampling an indicator variable according to this PMF for orientations. If this variable indicates non-alignment, we sample an orientation uniformly at random. Finally, we model padding around CAD models as a 4D diagonal-covariance Normal distribution conditioned on the CAD model m. The dimensions correspond to padding on each side of the object: in-front-of, behind, to-the-left-of and to-the-right-of. SamplePadding (line 9 of Algorithm 1) returns a sample from this 4D Normal distribution.

The knowledge learnt during training and captured in these PMFs is interpretable: Table 4.4 shows values from the PMFs, indicating typical placements of objects, while Figure 4.16 shows typical locations where we place various classes. In both cases, these agree well with human intuition on interior design.

Motif discovery. We search the training set for all joint occurrences of a given K-tuple of classes (*e.g.* table, chair, chair) within a room. For every occurrence of one of these classes—designated as the *base* object—we calculate displacements of the centres of the other objects relative to the base object. We model these displacements as points in a $2(K-1)$-dimensional space and cluster them with a Dirichlet process mixture model (DPMM) (Rasmussen, 2000), fitted by variational inference. We use Gaussian clusters with diagonal covariance and fit a DPMM per K-tuple of classes. We calculate the area inside the 2σ contour for the location of each element in the motif; if all of these are less than a threshold, then the cluster is accepted as a motif. We store the CAD models, relative locations, and orientations for every occurrence assigned to the cluster; one of these will be selected when instantiating the pattern. Some examples of motifs we discover are given in Figure 4.7.

Abutment discovery. We discover abutments in two stages. First, we gather sets $\mathcal{S}_i$ of sequences of CAD models, where each set will ultimately become an abutment pattern. Each sequence of CAD models represents an abutting series of instances in the training set. Then, for each $\mathcal{S}_i$, we calculate the transition probabilities for the corresponding Markov chain, as maximum-likelihood estimates given the CAD model sequences $\mathbf{s} \in \mathcal{S}_i$.

More precisely, we collect the sets $\mathcal{S}_i$ in a collection $\mathcal{T}$, initialising $\mathcal{T}$ to be empty. While doing so, we maintain the invariant that $\forall i \neq j$, $\mathcal{S}_i$ and $\mathcal{S}_j$ do not contain any sequences that share CAD models. For each room in the training set, we find all pairs of objects that abut, based on their rotated bounding-boxes touching at an edge. These pairs are combined transitively to form full sequences of objects $\mathbf{s}_j$, each being a row of abutting objects of some orientation. For each object-sequence $\mathbf{s}_j$, ignoring those with just two objects, we check if any of its CAD models appears in any sequence in a set $\mathcal{S}_i \in \mathcal{T}$ already created. If so, we add the object-sequence to $\mathcal{S}_i$; if not, we create a new one storing just $\mathbf{s}_j$, and add it to $\mathcal{T}$. In the first case, we also check that adding the sequence to $\mathcal{S}_i$ has not broken the invariant that sets do not share CAD models; if it has, we merge sets until the invariant holds again. At the end of the above process, each $\mathcal{S}_i \in \mathcal{T}$ contains many sequences of CAD models, each of which we will treat as a sample from the Markov chain M_i. It is then straightforward to learn the transition probabilities for M_i by maximising the likelihood of all the sequences $\mathbf{s}_j \in \mathcal{S}_i$. Some examples of abutments we discover are given in Figure 4.8.

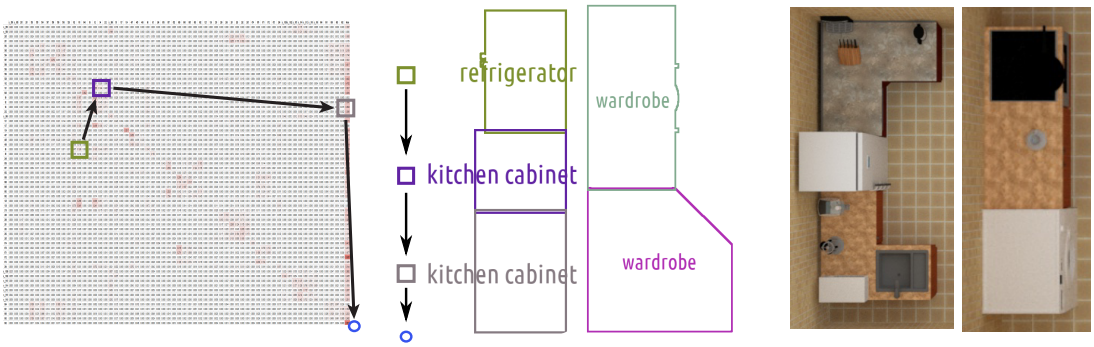


Figure 4.8: Statistics of abutments observed in the training database are recorded in a transition matrix, with rows and columns corresponding to CAD models, along with a terminal state (blue circle). We synthesise abutments by generating Markov chains using the learnt transition probabilities, and placing the selected CAD models next to each other.

4.4 Applying constraints

Our generative model accommodates diverse constraints using rejection sampling as a general mechanism, *i.e.* we sample layouts until we obtain one that satisfies the necessary constraints. We demonstrate the versatility of our generative model using some example constraints. Incorporating other constraints can be achieved similarly as long as a given layout can be verified to satisfy them. Since our sampling process is fast (tens of milliseconds per room), any inefficiency due to rejection sampling is outweighed by its ability to serve as a common mechanism to impose a wide range of constraints (Table 4.2). In some special cases, we can avoid rejection sampling by allowing users to explicitly manipulate parameters of distributions learnt (Figure 4.12).

Room type and size. As the room type r has no ancestors in our model, it can directly be assigned a constrained value, avoiding rejection sampling entirely. Since room size is a continuous value, and the probability of any sample satisfying this is zero, we allow a small tolerance on each dimension (2% in all our examples).

Traversability. A layout is traversable if there exists a path between all points in free space (regions with no furniture) and from all points in free space to all doors in the room. To verify this, we first rasterise an orthographic projection of the furniture onto the floor plane at a fixed resolution and identify free space as the complement of this footprint. We calculate P (areas where people can stand or pass) via morphological

erosion of the free space using a circular kernel of radius 0.25m and also add regions on doors to P. Similarly we calculate regions A that require access using morphological erosion, but with a larger kernel. Then we verify traversability by checking whether x is reachable from y, $\forall x, y \in A$ via some $\{z\} \subseteq P$.

Gap placement. Ensuring there is a gap at a particular location allows users to augment layouts with their own 3D models, that are not part of our system. In order to make rejection sampling efficient, rather than just discarding layouts until one that satisfies the constraint is found, we directly place a 'gap instance' in a suitable cell, ensuring that no object will occupy the relevant space. Note that some rejections will still occur, as cell locations are not known precisely until all furniture items are placed.

Object placement. We allow users to place instances of CAD models known to our system, at specific locations—*e.g.* a bed against a particular wall. Similarly to placing gaps, we ensure that a suitable instance is placed in the relevant cell, thereby greatly reducing the chances of rejection.

Doors and windows. We model door and window specification using a combination of gap-placement at edges of rooms, and traversability (the area just in front of each door is included in P).

Refurnishing. By simultaneously constraining room size, type, and door/window positions, we can *refurnish* existing rooms from SUNCG. This allows us to lift meshes and textures for the walls, floor, and ceiling, and also doors and windows, from a room in SUNCG—then populate it with an arbitrary number of different furniture layouts. This is valuable for generating large numbers of complete, realistic rooms without any user input. Another related use-case is in open-world game level construction. Here one can manually design a building mesh with walls, doors, and windows, then automatically populate different instances of that building with different furniture layouts.

4.5 Experiments

In this section, we present qualitative results to highlight the samples (with and without constraints) generated using our model, and quantitative results measuring performance. We also assess the quality of our generated layouts via a simple user study. All rendered images were obtained using path-tracing (Jakob, 2010). The execution times reported in this chapter were obtained using our unoptimised and sequential Python implementation, on an Intel Xeon E5-2620v3 processor, using less than 1GB of RAM.

Figure 4.9: Samples from our model, without constraints applied

Figure 4.10: Samples from our model, with constraints. The sizes of the rooms and the locations/classes of objects shown in pink boxes are constrained

4.5.1 Generating layouts

Unconstrained output. We show some output examples from our generative model, without any constraints specified, in Figure 4.9. Our model produces results without objects intersecting and mimics the diversity found in the training dataset—both in terms of the types of rooms as well as the objects in them. The co-occurrence and relative placements of objects are also realistic and natural. Unconstrained sample layouts are generated in 0.04s on average.

Examples with constraints. Figure 4.10 shows examples of layouts where the room size and placement of one object were specified by the user. Note that the other sampled objects in the room are automatically chosen, and placed harmoniously. For example, for the first image, the constraint was 'place a bed near the top right corner'. Our method automatically places nightstands on either side.

Figure 4.11 shows sample layouts where the shapes of the rooms and the locations of doors were specified as constraints. Note that doors are unobstructed.

Figure 4.12 shows examples of layouts where a user has specified particular clearances to be respected around specific object classes. The bar plots on the first column

Figure 4.11: Samples from our model with constraints. The sizes of the rooms, their shapes and door locations are constrained.

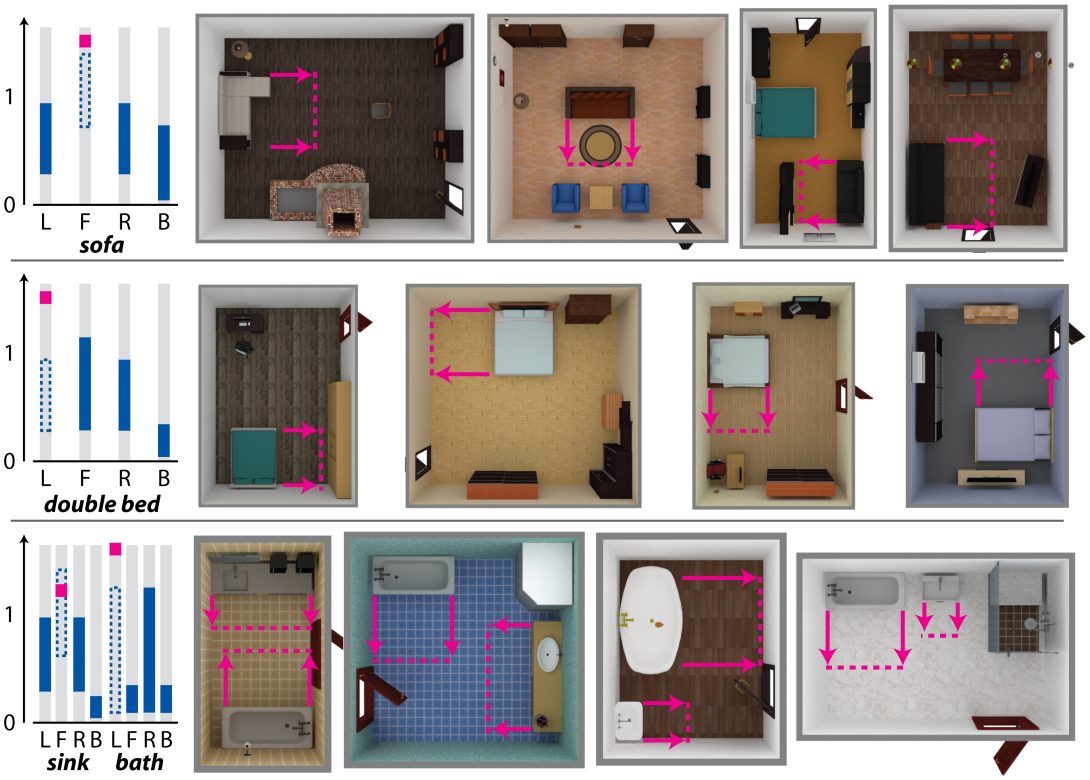


Figure 4.12: Samples from our model, with user-specified clearance constraints. The left column shows the default (blue) and user-specified (pink) padding ranges in meters for each side (left/front/right/back) of the indicated object; the remaining columns show samples drawn from our model with the constraint applied, with the specified padding regions indicated

(solid blue) show the ranges of clearances learnt during training on the left (L), front (F), right (R) and back (B) of the models sampled from four chosen classes (sofa, double bed, sink and bath). The pink squares on the bar plots depict user modifications of the learnt parameter (dashed blue rectangles). For each specified constraint (rows), four sample outputs are visualised (columns), and the impact of the user specification is shown using pink arrows as annotation. In this particular example, the constraints are imposed by directly editing learnt parameters rather than using rejection sampling, which leads to faster runtime.

Constraints producing uncharacteristic layouts. Large generative models run the risk of over-fitting their training set. However, a benefit of training a *constrainable* generative model is that we can generate arbitrary numbers of rooms fulfilling constraints, that are never (or very rarely) fulfilled in the training dataset. We demonstrate this using random sets of reasonable constraints and identify those sets of constraints which

Figure 4.13: Samples from our model, applying constraints that are not satisfied by any layout in the training set. In each case, we constrain the room type, size, and placement of one object (indicated by a pink box), choosing a combination of constraints that is not satisfied by any layout in SUNCG. Our model is able to sample rooms fulfilling the constraints, despite not having seen such examples at training time.

are not jointly satisfied by any room in the SUNCG dataset. Then, we use our model to sample a room that does satisfy the constraint. Examples are given in Figure 4.13.

Nearest neighbours in training set. As a further check that our model truly generates new rooms differing from those in the training set, we can search the layouts in SUNCG for the nearest neighbours of each of our samples. To find these, we first look for ground-truth rooms of the same room type as the sample, and similar size (differing by up to 50% along either dimension, allowing for rotations by $\pi/2$). Then, we filter these according to the counts of different object classes present, requiring that the total difference in counts over all classes is less than half the total number of instances in the sample. This yields a set of rooms with broadly the same shape and furniture items as the target sample; these are then ranked according to how similar their layout is to the sample. We divide each room into a grid, and at each point, note the class of the furniture item there (or 'none'); then, the similarity between ground-truth room and sample is the fraction of grid points with the same class. Examples of samples and their nearest neighbours are given in Figure 4.14; we see that they are typically quite

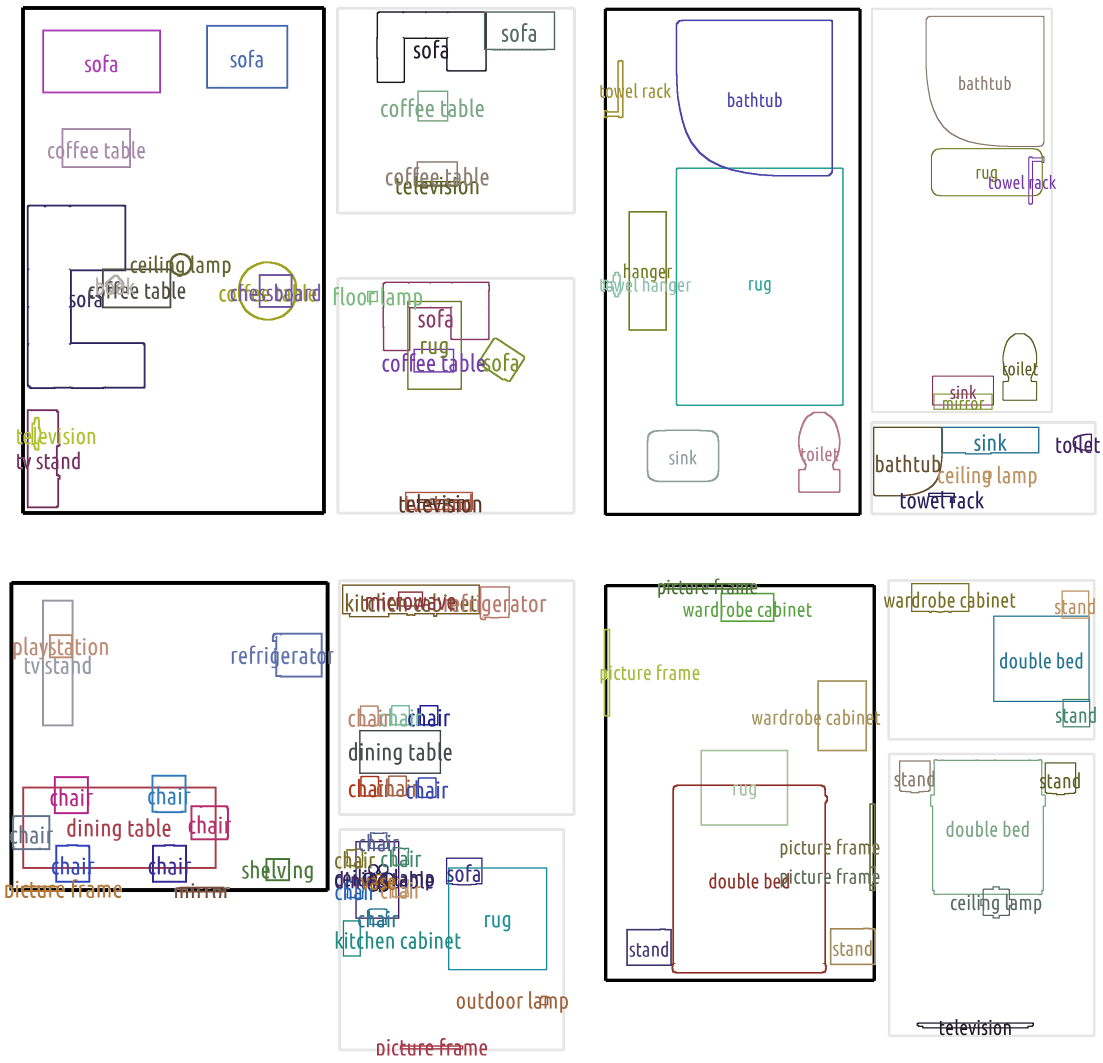


Figure 4.14: Samples from our model without constraints applied (black frames), displayed alongside their two nearest neighbours in the training set (grey frames)

different in arrangement and exact object counts, while remaining broadly similar in the classes of object present. Hence, our model is indeed generating novel layouts, rather than simply memorising the training set.

Runtime with constraints. We measured the average time taken by our method to sample layouts with different types of constraint applied. Results are shown in Table 4.2. Note that our unconstrained sampler implemented in Python takes just 0.04s to sample a complete layout, running on a single thread, and that with many classes of constraint, sampling is still fast compared with prior works.

Constraint	Time per sample /s
unconstrained	0.04
room type	0.04
direct parameter editing	0.04
object class exclusion	0.04
traversability	0.05
object placement	1.4
gap placement	1.8
room size	6.8
size + doors + windows	112

Table 4.2: Average time taken to sample a complete layout from our model, with different types of constraint applied. The timings are for an unoptimised Python implementation running on a single thread.

4.5.2 User study

We assessed the realism of layouts generated from our model by a user study comparing them to human-designed rooms from the SUNCG database, and layouts generated by a concurrent work (Wang et al., 2018). We recruited eight users, none of whom had any prior qualifications or professional experience in interior design. The users were shown pairs of images via a simple web interface (Figure 4.15), and asked to identify the image with a more realistic, or natural, layout of objects, disregarding other factors such as the realism of rendering. After the user made a choice by clicking on one of the images, their selection was recorded, and the next pair was shown. In each case, one image was a ground-truth (human-designed) layout from SUNCG, and the other was a sample from our model; the order of the two images was randomised for each pair.

In total, we conducted five experiments. For each experiment, we generated 128 samples from our model, and selected 128 layouts from SUNCG uniformly at random (for the experiment comparing with Wang et al. (2018), we instead used all 44 layouts included in their preprint). When presenting a pair of images to a user, we selected one image from each set uniformly at random. Each user was assigned randomly to two or three experiments, and shown 50–100 pairs of images per experiment. The images were rendered using the photo-realistic Mitsuba renderer (Jakob, 2010), at a resolution

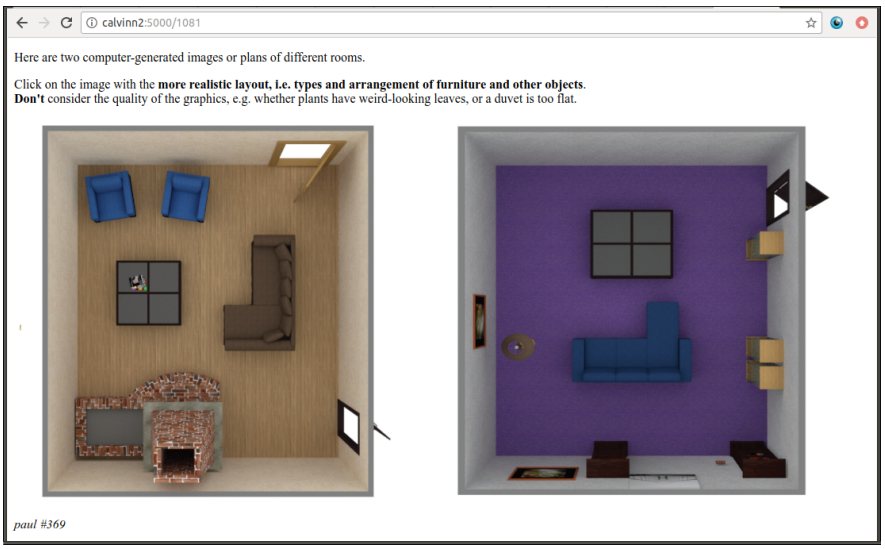


Figure 4.15: Screenshot of the web interface for user comparison of layouts. The user is presented with renderings of two layouts, and asked to select the more realistic by clicking on it; the result is then recorded and the next pair for comparison is displayed.

of 640×480 pixels. For most experiments we used overhead views, with the camera positioned looking down at the centre of the room, from high enough to ensure that the entire room was visible. For one experiment we used first-person views, with the camera parameters selected using the method of Song et al. (2017).

For each experiment, we merged the results from the different users. We then measured the overall fraction f_p of pairs for which the user preferred our layout to that from SUNCG. In order to quantify the reliability of these results, we also calculated 95% confidence intervals for each experiment, using the method of Efron and Tibshirani (1986). Given N user responses, this approximates the confidence interval by first sampling with replacement K new sets of N responses from the original N. For each of these resampled sets, the fraction of pairs where the user preferred our layout is calculated; the standard deviation σ over different sets is then used to approximate the 95% confidence interval as $f_p \pm 1.96\,\sigma$.

Unconstrained. Our first two experiments compared layouts from our model without constraints, to ground-truth layouts from SUNCG. For one experiment, the images presented were overhead renderings; for the other, they were first-person views from inside the room. The observed user preferences and confidence intervals are given in Table 4.3a. In first-person views, users preferred our layouts; in overhead views, our layouts are indistinguishable from ground-truth up to statistical significance.

Viewpoint	Ours preferred
overhead	$48.1 \pm 6.6\%$
1st person	$58.1 \pm 6.0\%$

(a)

Constraints	Ours preferred
size + object	$45.2 \pm 6.8\%$
size + door	$35.2 \pm 5.4\%$

(b)

Table 4.3: Percentage of image-pairs where non-expert users preferred (*i.e.* deemed more realistic) a layout sampled from our model, as opposed to a ground-truth layout from SUNCG. Higher is better, with 50% indicating that our samples are indistinguishable from ground-truth. Ranges are the 95% confidence interval, estimated by bootstrap (Efron and Tibshirani, 1986). **(a)** Unconstrained layouts; **(b)** Constrained layouts.

Constrained. We then assessed room layouts generated by our model with constraints, similarly to layouts without constraints, but using only overhead renderings. We conducted two experiments, corresponding to two representative settings for constrained generation: (i) fixing the room size and placement of one object; and (ii) fixing the room size and locations of doors and windows (implying gap placement and traversability constraints). For (i), we randomly generated arbitrary, but meaningful, pairs of constraints and sampled one layout fulfilling each. For (ii), we randomly selected rooms from SUNCG, and used their size and door/window locations as constraints for our model, again sampling one layout for each. In the second case, our model *refurnishes* existing rooms. In both cases, we compare our samples against arbitrary ground-truth rooms, which typically do not fulfill the same constraints, *i.e.* we test the realism of our samples and not whether constraints are fulfilled (which is guaranteed by rejection). Results are given in Table 4.3b. With room size and the placement of one object constrained, our layouts are indistinguishable from ground-truth up to statistical significance. With room size and the positions of doors constrained, users preferred human-designed layouts.

Comparison with Wang et al. (2018). Finally, we compared unconstrained samples from our model with images obtained from a preprint of concurrent work (Wang et al., 2018). We presented 4 users with a total of 192 pairs, each using one of the 44 images from their preprint and a randomly selected layout from our model. For $57.8 \pm 7.0\%$ of pairs presented, users preferred our image. Thus, our unconstrained layouts are competitive with state of the art in layout generation.

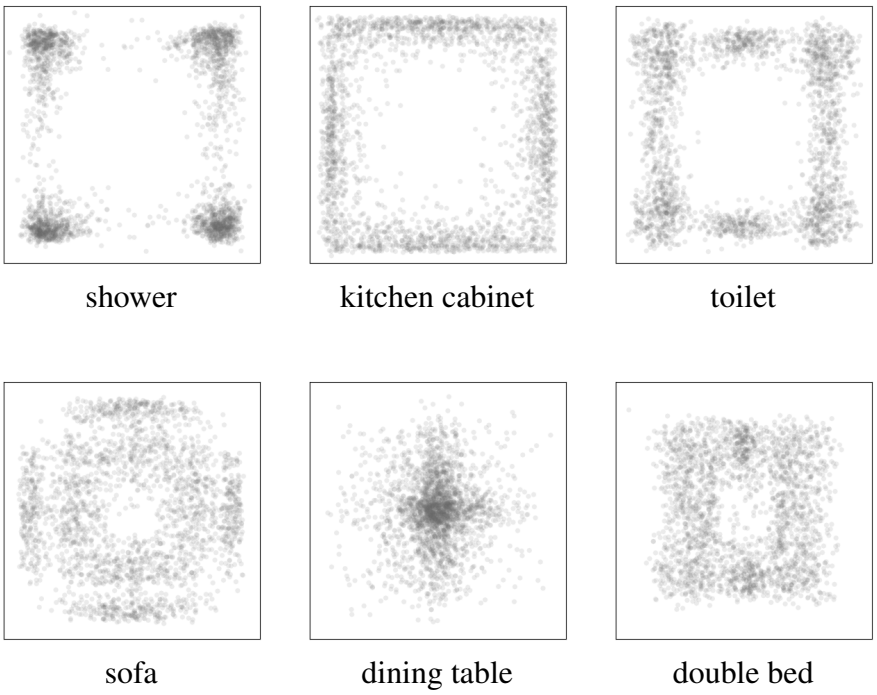


Figure 4.16: Heat-maps showing locations where our model places objects of different classes

4.6 Discussion and conclusions

In this section, we discuss various aspects of our model, then present some options for future research.

4.6.1 Analysis and discussion of our model

Comparison with related/concurrent work. Probabilistic generative methods for room layouts are challenging to sample from. Often the sampling is not guaranteed to converge to a valid layout, especially when many objects are present, *e.g.* the model proposed by Handa et al. (2016). That particular model also requires that the number of objects, and size of the room, be specified manually. Our model performs comparably with the concurrent work of Wang et al. (2018) that learns millions of parameters over days of training. In $57.8 \pm 7.0\%$ of pairs presented, users preferred our layouts to theirs. In addition to accommodating constraints easily, our model has another advantage in that the parameters learnt are over semantically meaningful concepts (categories such as furniture) allowing direct modulation of learnt parameters as shown in Figure 4.12. Although we manually specified padding constraints, they could be calculated from

furniture	p_{edge}
toilet	0.88
range oven	0.86
sink	0.83
chair	0.32
tripod	0.32
armchair	0.29

furniture	$p_{\pi/2}$
kitchen cabinet	0.99
dishwasher	0.99
single bed	0.99
office chair	0.74
armchair	0.65
tripod	0.49

Table 4.4: Many parameters that are learnt during training are human-interpretable. **(a)** furniture classes with highest (top) and lowest (bottom) probability p_{edge} of being at the edge of a room rather than the interior **(b)** furniture classes highest (top) and lowest (bottom) probability $p_{\pi/2}$ of being at an angle that is a multiple of $\pi/2$.

alternatives such as human-centric affordances (Qi et al., 2018).

Inter-object relationships. We explicitly discover and encode relationships across classes of objects using patterns such as motifs and abutments. These patterns capture higher order relationships (not just pairwise); in the case of abutments, they are able to model sequences of variable lengths which may not be present in the training database. Figure 4.17 shows unnatural layouts generated when inter-object relationships such as patterns and abutments are not modeled. Additionally, implicit relationships are captured between different CAD models of the same class in a given layout. The generative process does not favor a large item from a class if multiple small items from that class have been sampled.

Efficient implementation of constraints. For many of the constraints listed in Section 4.4, rejection sampling can be avoided using alternative implementations. For example, space constraints may be tailored at the class level by modifying the 4D Normal distribution learnt for padding. Figure 4.12 shows direct editability of learnt parameters. Example layouts produced by the modified distribution are shown on the right, along with the effects of the user manipulation on the corresponding objects. Another example of a constraint that may be implemented efficiently is the specification of certain object classes (or CAD models) as not desirable. Rather than relying on rejection sampling, these classes (or models) may be pre-emptively avoided during instantiation.

Figure 4.17: Samples from our model, but without patterns. Left: the kitchen cabinets and appliances are scattered, rather than placed adjacent to one another (as enabled by abutments). Centre: the chairs are scattered, rather than placed around the dining table (as enabled by motifs). Right: the two night-stands (lower left) are not at the expected location near the bed (as enabled by motifs)

Interpretability. Since our model learns parameters associated with semantically meaningful categories and positions, the learnt results can be interpreted and manipulated. Table 4.4 lists the highest and lowest probability entries for object positioning and orientation. We obtained these numbers by averaging over the corresponding probabilities for all CAD models in the stated classes. The numbers align with our expectation that chairs and tripods may not necessarily be placed along the edges of rooms, and that they may be less likely to be aligned with edges than beds, kitchen cabinets, or dishwashers. Figure 4.16 visualises heat-maps of where the model places a few chosen object classes. For each class, we sampled 2500 rooms, and plotted (black dots) where objects of the indicated class were placed (normalising the room bounds to a unit square). The model has learnt to place different classes meaningfully—for example, showers are almost always at the corner of a bathroom, dining tables are often at the center of a room, and toilets are always against a wall.

Room shape. In our implementation we decompose non-rectangular rooms into rectangular regions and apply our cell structure on each region. Other strategies to partition rooms into cells may be adopted as long as they are kept consistent across training and sample generation. However, the choice of partitioning strategy may impact the quality of results.

Multiple, simultaneous constraints. Another advantage of rejection sampling as a general mechanism to impose constraints is that support for multiple constraints is trivial to implement. However this flexibility is accompanied by a penalty in terms of runtime. The time taken to generate a sample that satisfies multiple constraints is on average the product of the times taken to support each constraint.

4.6.2 Future work

Improving sample quality. We have shown that our model produces samples that are often indistinguishable from those in the training set. However, there is still scope for improvement. We noted in Section 4.2.1 that it is beneficial to use a different cell structure for certain types of room. This allows us to introduce stronger priors on the layout, incorporating functional and other common-sense relations that are not otherwise captured. One room type to which this is particularly relevant is kitchens; these have stereotyped layouts with strong functional constraints, that are not readily modelled in our present system. In particular, while we model abutting lines of furniture such as cabinets and appliances, these are not necessarily placed spanning entire walls (as observed in reality), and may be functionally inappropriate (*e.g.* placing an oven near a fridge). Note that introducing such prior information can be seen as bridging the gap between classical procedural modelling, where the generative process is fully hand-engineered, and data-driven probabilistic modelling.

It may also be beneficial to prune the training data more aggressively. For example, some layouts in SUNCG contain implausibly few or many objects, or are very small or large. The very recent work of Wang et al. (2018) defines a set of criteria and removes all rooms that do not satisfy these; taking a similar approach may reduce noise in our training data, and hence improve sample quality.

***A posteriori* editing.** Our model is designed for fast generation of layouts with pre-specified constraints—however, our formulation does not facilitate editing an existing (previously sampled) layout. While we can handle *a priori* specification such as 'I want a new layout such that this television set is along the north-south wall', it cannot handle *a posteriori* editing such as 'in the previously generated layout, move the television to the north-south wall'. Typically, a user-specified edit (such as moving an item of furniture) requires the remainder of the layout to be updated in a semantically-consistent fashion. Any objects that the moved furniture now collides with should be moved out of the way; any objects used with it for actions (*e.g.* chairs at a table, or a television

facing a sofa) should have their positions adjusted suitably, but again with avoidance of collisions. The overall layout may also need adjusting for aesthetic reasons.

Mathematically, it suffices to draw a sample from the layout model conditioned on the edited item of furniture—this is equivalent to how we draw a constrained sample currently. Practically, however, this is not useful—the remainder of the layout will almost certainly change drastically. Instead, it is preferable for the layout to remain as close as possible to the current one, but yet to remain plausible under the model. One way to achieve this would be to define a similarity metric on rooms, then draw samples from our model that are minimally different from the current room according to this metric, and that also satisfy the constraint defined by the user edit. Such a metric could be based on the one we use to find nearest neighbours to our samples in the training set, as described in Section 4.5.1.

Density evaluation. Our layout generation framework is a directed graphical model; as such, it defines a joint probability density over all variables. Evaluating this joint density while sampling a layout is straightforward. However, some of the variables are latent—given an arbitrary layout, we must marginalise over them in order to calculate its likelihood under the model. Examples of latent variables in our model are (i) cell assignments, (ii) whether objects are attributable to patterns or singletons, and (iii) padding values. Unfortunately, marginalising over the large space of latent variables in our model is intractable. Hence, it would be valuable either to adapt the model to be more amenable to likelihood evaluation, or to develop an efficient (maybe approximate) algorithm for marginalisation.

Learning the cell structure. Our model places furniture items by assigning them to cells, and the topology of the cell structure is manually defined (Section 4.2.1). However, there exist realistic room layouts that do not conform to this cell structure—and so our model could never generate them. This suggests that it would be valuable to learn the cell structure itself from data. This is more challenging than learning any of the parameters currently in our model, as the training examples are not annotated with a cell structure, hence this must be discovered automatically. One possible approach would be to define a tree-structured model that recursively splits areas of the room into two sub-regions; the leaf nodes of this tree structure would then correspond to objects (or to empty space). The structure itself would need to be inferred for each room during training, and parameters governing the probability of splits learnt, as well as the probabilities for different object classes for leaf cells conditioned on the tree

structure. One powerful class of models able to learn this kind of structure is recursive neural networks (Li et al., 2017).

4.6.3 Concluding remarks

We have presented an efficient, probabilistic, data-driven, generative model for indoor furniture layouts. The algorithm used to generate layouts is simple and the parameters learnt from training data are human-interpretable. We demonstrated that our model is able to accommodate a variety of constraints using rejection sampling as well as editing of learnt parameters. We presented qualitative and quantitative results through rendered layouts, performance measurements and a user study. Finally, we have suggested several possible directions for future research.

Chapter 5

Automatically selecting inference algorithms for discrete energy minimisation

It is vital that scene understanding methods make predictions that are globally-consistent across all variables in a problem, *e.g.* pixels or object instances. We wish to avoid solutions are that locally good, but globally implausible. For example, in semantic segmentation, it is highly unlikely that pixels labelled 'sky' should appear below pixels labelled 'road'; while a local classifier may make such mispredictions, a global model should enforce consistency. In binocular stereo (Section 3.1.3), disparities are estimated by matching patches of images, which can result in fragmentary, discontinuous reconstructions; a global model ensures that the result is smooth and regular, while still remaining as close as possible to the locally-predicted disparities. Thus, we require global models, that reason over an entire image (or even multiple images) at once, and yield a single, consistent, structured prediction.

One natural way to formalise this requirement is to cast the problem as inference in a graphical model. More specifically, we consider the minimisation of discrete energies defined over factors (see Section 5.1.1). Each variable takes values in some discrete label-space, and each factor is a real-valued function on some subset of the variables, its clique, yielding an additive contribution to a global energy.

These minimisation problems arise from many different underlying *problem classes*—we have already mentioned semantic segmentation and stereo matching. In scene understanding, other examples are object localisation, instance segmentation, and plane-fitting; in lower-level vision, other examples are montaging and texture reconstruction.

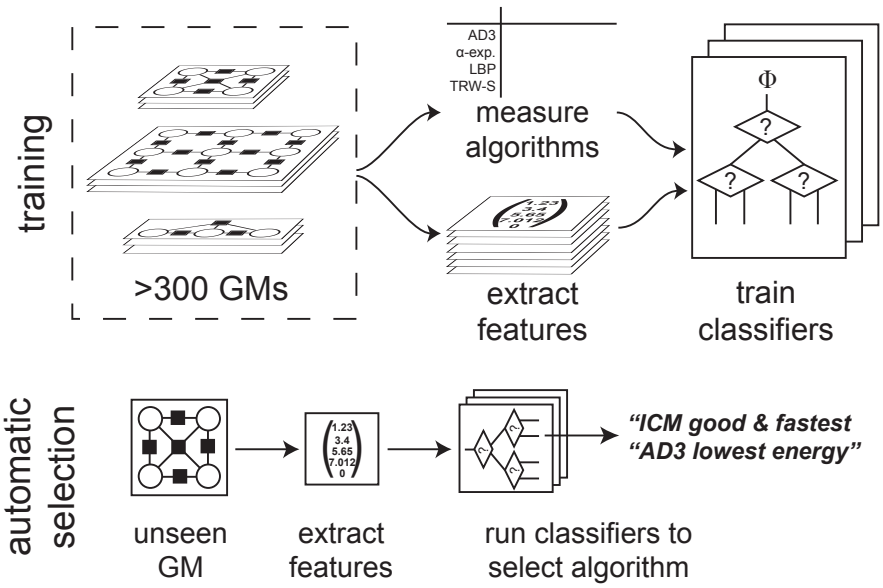


Figure 5.1: Our pipeline for automatic algorithm selection

Outside of vision, they also appear in domains such as bioinformatics.

Different problem classes give rise to problem instances with different characteristics, such as size of cliques and number of variables, affecting which inference algorithms are best suited to them. As such, the space of published inference algorithms is vast (Kappes et al., 2015), with methods ranging from highly specialised to very general (see Section 5.1.3). For example, message passing (Bishop, 2006) is widely applicable, but takes exponential time for large cliques, and may not converge. Variants such as TRW-S (Kolmogorov, 2006) do guarantee convergence, but not necessarily to a global optimum. α-expansion (Boykov et al., 2001b) and graph-cuts (Greig et al., 1989) are better suited to models with dense connectivity, but require factors to take certain restricted forms, while QPBO (Rother et al., 2007) only works for models with binary labels, and may leave some variables unlabelled. Other algorithms such as those of Kappes et al. (2012) and Martins et al. (2015) are applicable to models with arbitrary factors and labels, but run more slowly.

Thus, when developing a new model, it may be difficult to decide what algorithm to use for inference. Selecting a good algorithm for a given model requires extensive expertise about the landscape of existing algorithms and typically involves understanding the operational details of many of them. Moreover, even for an expert who can choose which algorithm is best overall on a particular problem class, it may not be clear which is best for a particular *instance*—certain problem classes are heterogeneous enough that different instances within them may be best solved by different algorithms (Sec-

tion 5.3). An alternative solution would be to run many algorithms on each input model and see which one performs best empirically. However, this would be computationally very expensive.

Recently, studies have appeared that evaluate a number of algorithms on various problems, comparing their performance (Kappes et al., 2015; Szeliski et al., 2008; Andres et al., 2010; Kolmogorov and Rother, 2006; Alahari et al., 2010). These are intended to provide a 'field guide' for the practitioner, suggesting which techniques are suited for which models. In this chapter, we take the next step forward and propose a technique to *automatically* select which inference algorithm to run on an input problem instance (Section 5.4). We do so without requiring the user to have any knowledge of the applicability of different inference methods, and without the computational expense of running many algorithms. Thus, our method is particularly suited for the vision practitioner with limited knowledge of inference, but who wishes to apply it to real-world problems.

Our method uses features extracted from the problem instance itself, to select inference algorithms according to two criteria relevant for the practitioner: (1) the fastest algorithm reaching the lowest energy for that instance; or (2) the fastest algorithm delivering a very similar labelling to the lowest energy one (Figure 5.1). The features are designed to capture characteristics of the instance that affect algorithm applicability or performance, such as the clique sizes and connectivity structure (Section 5.4.1). We train our selection models without human supervision, based on the results of running many algorithms over a large dataset of training problem instances.

We perform experiments on an extended version of the OpenGM2 benchmark (Kappes et al., 2015), containing 344 problem instances drawn from 32 diverse classes, and consider a pool of 15 inference algorithms drawn from the most prominent approaches. The results show that on 69% of problem instances our method selects the best algorithm. On average, the labels of 96% of variables match that returned by the algorithm achieving the lowest energy. Our automatic selector achieves these results over 88$\times$ faster than the obvious alternative of running all algorithms and retaining the best solution.

Chapter overview. In Section 5.1, we give background on the MAP inference problem, inference algorithms, applications in computer vision, and on methods for automatic algorithm selection in various domains. Section 5.2 and Section 5.3 describe our dataset of models, and the set of inference algorithms. Section 5.4 presents our method for automatic algorithm selection. We present results in Section 5.5, then analyse these

and conclude in Section 5.6.

The work described in this chapter was presented at ECCV 2016 (Henderson and Ferrari, 2016a).

5.1 Background and related work

In this section we give background on graphical models and the MAP inference problem, including some applications in computer vision. Then, we describe different types of MAP inference algorithm. Finally, we describe prior work on automatic algorithm selection, in machine learning and other domains.

5.1.1 MAP inference in factor graph models

Factor graph models. In this chapter, we consider inference in discrete factor graph models (GMs), which are defined as follows.

Let V be a set of variables (typically thousands of them), each taking values in some discrete label-space, which may or may not be shared among the different variables. Let F be a set of factors, each of which is a real-valued function ϕ_s on some subset $V_s \subseteq V$; this subset is known as the factor's *clique*. The *order* of a factor is the number of variables in its clique. The factors define a global energy function

$$E(V) = \sum_s \phi_s(V_s). \tag{5.1}$$

Equivalently, they may be seen as defining a joint probability density according to the Gibbs distribution

$$P(V) = \exp\{-E(V)\}/Z \tag{5.2}$$

$$= \frac{1}{Z}\prod_s e^{-\phi_s(V_s)} \tag{5.3}$$

where Z is the *partition function* which normalises the distribution.

This defines a rather general class of graphical model, that can represent both directed and undirected probabilistic models (Section 4.1.1) with discrete variables, while keeping the factorisation explicit.

MAP inference. For a given GM, the task of maximum *a posteriori* (MAP) inference

is to compute a joint labelling V^* of all the variables V such that

$$V^* = \arg\max_V P(V) \tag{5.4}$$

$$= \arg\min_V E(V) \tag{5.5}$$

$$= \arg\min_V \sum_s \phi_s(V_s) \tag{5.6}$$

V^* is then the minimum-energy joint-labelling, or the most-probable joint-labelling under the above Gibbs distribution.

In general, computing V^* is NP-hard. This can be seen by considering the case of a graph with a single factor, whose clique includes all variables. Assuming the function defined by the factor is unknown, we can only find V^* by checking all the exponentially-many possibilities for V. However, in typical problems, the decomposition of the energy into factors, whose cliques are typically only a small subset of V, allows more efficient inference algorithms to be devised. Different decompositions lead to radically different inference algorithms; our goal in this chapter is to select suitable algorithms for a given GM.

5.1.2 GMs in computer vision

GMs have been applied to a vast range of tasks in computer vision; here we describe some representative examples. Note that we mention only some applications, intended to give a flavour of possible uses; see Kohli and Rother (2012) and Kappes et al. (2015) for a more comprehensive survey.

Although these application domains are rather different at first sight, they all have in common the fact that they have a high-dimensional discrete output space, and predictions must jointly satisfy various dependencies among the output dimensions. Moreover, these dependencies can be expressed in terms of factors that operate only on small subsets of the variables—it is this that makes inference tractable. One recurring pattern is that the GM acts as a regulariser on the output of some local prediction model, with unary factors encouraging the variables to agree with that local model, and other factors ensuring that the prediction is globally consistent.

GMs were first used in the guise of Markov random fields (MRFs), for low-level image-processing tasks such as denoising—that is, reconstructing an original image given a noisy version as input (Geman and Geman, 1984; Besag, 1986). These models have one variable per pixel; each has a unary factor encouraging it to take a value close to that in the observed, noisy image. The simplest such models add only pairwise

factors connecting each variable to its four neighbours, encouraging smoothness of the result.

Another domain wherein GMs are used extensively is stereo reconstruction (see also Section 3.1.3). Here, there is again one variable per pixel, but the values now correspond to disparities, or inverse depths. Pairwise factors once again enforce smoothness, while unary factors favour the initial, noisy disparity values calculated by matching image patches. Inference in the GM therefore regularises or smoothes the disparity values, but in a globally-optimal fashion ensuring minimal deviation from the initial values (Boykov et al., 2001a; Felzenszwalb and Huttenlocher, 2004; Szeliski et al., 2008). Both denoising and stereo models can be extended to incorporate more factors. These may be longer-distance pairwise interactions, or more recently, higher-order interactions that capture statistics across groups of pixels (Boykov et al., 2001b; Kohli et al., 2007, 2008; Woodford et al., 2008). These higher-order factors typically take restricted forms, to ensure that inference is tractable.

GMs are also applied to higher-level vision tasks. Semantic segmentation aims to label each pixel of an image with a semantic class such as ‘sky’ or ‘building’. Given an imperfect classifier that operates per-pixel (*e.g.* a random forest or convolutional neural network), a GM (typically in the form of a conditional random field, or CRF) allows the predicted labels for different pixels to be reconciled with one another. As with denoising and stereo, simpler models use only pairwise factors between neighbouring pixels (Shotton et al., 2006); these favour adjacent pixels taking the same label if their colours are similar. Others capture higher-order relations, albeit still only locally, favouring all pixels in local but arbitrarily-shaped super-pixels taking the same label (Kohli et al., 2009). The same principle can be extended to video, by including pixels from different frames that unproject to the same physical location in one clique (Floros and Leibe, 2012). More recent models capture contextual relations through long-distance interactions across the entire image (Krähenbühl and Koltun, 2011; Guillaumin et al., 2013).

The variables need not correspond to individual pixels. Deselaers et al. (2010) present a model for weakly-supervised object localisation. Here, the variables correspond to images known to contain a certain object class; the labels taken by each variable correspond a set of bounding boxes within that image, that might contain an instance of the object class. Unary factors are given by the objectness probability of a bounding box (Alexe et al., 2010), while pairwise factors measure the appearance similarity between two bounding boxes in different images. Inference then selects one

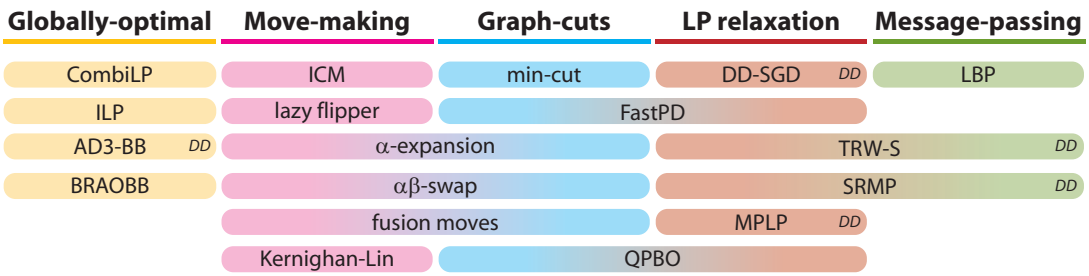


Figure 5.2: Categories of MAP inference algorithm, following the grouping of Kappes et al. (2015). 'DD' indicates methods based on dual-composition. Note that many methods span different categories, *e.g.* move-making methods use graph-cuts to solve sub-problems, and several LP-based methods are implemented in terms of message-passing.

bounding box per image, so that they are likely to contain objects and to be visually similar over the images.

5.1.3 Inference algorithms

MAP inference algorithms can be split into several broad categories (see Figure 5.2). We follow the grouping defined by Kappes et al. (2015); however, this is only one of several possible ways to categorise algorithms, and many algorithms span more than one category. For example, the widely-used TRW-S algorithm of Kolmogorov (2006) is implemented in terms of message-passing, but actually solves a dual problem to to a particular linear programming relaxation. In this section, we describe each of these categories of algorithm in more detail.

Graph-cuts. When all variables in the model are binary, and all factors are pairwise and submodular (a discrete analogue of convexity; see Lovász (1983)), the inference problem can be recast as that of finding the min-cut for a particular graph. An exact solution for this can be calculated efficiently (Greig et al., 1989). The applicability of graph-cuts has been extended in recent years. Ishikawa (2009, 2011) shows that high-order binary problems can be reduced to pairwise, by introducing auxiliary variables. This transformation preserves the optimal labelling, but may introduce non-submodular terms; Fix et al. (2011) present a version that produces fewer such terms. In the case of non-submodular pairwise models (still with binary labels), a solution cannot be found by graph-cuts; however, the related QPBO (Rother et al., 2007) method can be used to find a partial labelling. This marks some variables as

unknown, but the remainder are assigned the same label as they would take in the globally-optimal solution.

Move-making algorithms. When the problem has more than two labels per variable, graph-cuts is not directly applicable, but can be used indirectly through move-making methods. These no longer guarantee global optimality, but do guarantee convergence to a local minimum. They work by solving a sequence of binary sub-problems, exactly using graph-cuts or approximately using QPBO. Each sub-problem computes a refined solution to the global problem, with the guarantee that this does not increase the energy. In α-expansion (Boykov et al., 2001b), each sub-problem allows all variables to retain their current label or change to some other label α. In $\alpha\beta$-swap (Boykov et al., 2001b), each sub-problem allows all variables of label α or β to retain their current label or swap to β or α respectively. In fusion moves (Lempitsky et al., 2010), each sub-problem merges two other candidate solutions, yielding a new solution with energy lower than (or equal to) either candidate. The candidate solutions typically come from efficient but approximate or otherwise-restricted inference methods.

A much simpler move-making method is ICM (Besag, 1986); here, the sub-problems simply choose a new state for each individual variable, keeping the remainder fixed. This is a coordinate descent algorithm: each variable in turn is set to the lowest-energy state given all others. It provides a local-optimality guarantee in that two or more variables must be changed simultaneously to produce a lower energy than its result. However, this typically yields poor solutions to all but the simplest problems. A more powerful extension is the lazy flipper of Andres et al. (2012a), which uses a brute-force search, combined with efficient data structures. This provides the stronger local-optimality guarantee that at least k variables must be changed simultaneously to improve the energy. It is typically initialised with the output of another algorithm and used to fine-tune that solution, using a small value of k.

Linear programming relaxations. The MAP inference problem can be relaxed to a linear program (LP), by allowing the variables to take continuous values instead of their discrete labels. It is typically easier to solve the resulting LP than the original discrete problem. Unless the LP relaxation is *tight*, this yields a non-integer solution, which requires rounding to an integer labelling; note that naïvely rounding to the nearest such labelling often does not yield the global optimum of the original problem (even if the LP itself was minimised). While the LP can be solved directly using off-the-shelf solvers for small problems (Kappes et al., 2015), even this is not feasible for the

larger problems typically encountered in vision applications. Instead, methods solve a more-tractable dual to the LP. Kolmogorov (2006) presents TRW-S, an algorithm for general pairwise GMs that performs block coordinate ascent over a dual problem. The blocks are chain sub-graphs of the GM, constructed in a fashion that allows convergence to be guaranteed, and admits an efficient implementation. Schoenemann and Kolmogorov (2014) and Kolmogorov (2015) present two different ways of extending this to GMs with higher-order factors. Globerson and Jaakkola (2007) present MPLP, another dual-space coordinate ascent algorithm, which uses a different choice for the dual; Sontag et al. (2008, 2012) present extensions with improved relaxations. These methods based on dual coordinate ascent do not necessarily converge to an optimal solution of the LP, merely an arbitrary fixed point. They are however very efficient, and can be implemented in terms of message-passing updates (see below). Komodakis et al. (2008) present a method (FastPD) that maintains solutions to both the original and dual problems, but similarly to α-expansion, solves a graph-cut sub-problem at each iteration to update these. Maintaining the dual allows this method to solve the sub-problems more efficiently, hence it is typically faster than α-expansion. FastPD is guaranteed to converge, but once again to an arbitrary fixed point. Komodakis et al. (2007), Sontag et al. (2008), Kappes et al. (2012), and Martins et al. (2015) all solve the Wolfe dual (Guignard and Kim, 1987; Komodakis et al., 2007) to the LP, using a class of methods termed *dual decomposition*. These split the original energy into a sum of sub-problems over subsets of the variables (*e.g.* cliques or trees). The sub-problems are solved individually, and the results combined using different strategies. Unlike TRW-S, MPLP, and FastPD, these methods always reach a global optimum of the LP relaxation. Importantly, all methods that solve a dual problem yield a lower-bound on the true energy, given by the current dual solution. This is valuable in practice, as it can be used to check how far a method is from the global minimum, even while that minimum has not been discovered yet.

Globally-optimal ILP-based methods. The MAP inference problem can be restated as an integer linear program (ILP). As with LP relaxations, this can be solved directly using off-the-shelf solvers for small problems (Kappes et al., 2015). To handle larger problems, Savchynskyy and Schmidt (2013) use an ILP solver to refine the non-integer parts of a solution from an LP relaxation. Alternatively, branch-and-bound methods allow the space of possible solutions to be searched efficiently, by discounting many possible labellings. Bergtholdt et al. (2010) cast inference as finding a minimal-cost path over a graph whose vertices represent partial and complete labellings of the variables in

the original problem. They use the well-known A^* search algorithm (Hart et al., 1968) to accelerate this search; however, it still becomes too computationally-expensive for large problems. Martins et al. (2015) instead use upper and lower bounds computed by their dual-space LP solver AD^3 to guide a branch-and-bound algorithm. This gives a guarantee of global optimality which the LP solver alone does not; again, however, it does not scale to large problems. Otten and Dechter (2011) propose another pure branch-and-bound algorithm, which scales to somewhat larger problems, but is still much slower than approximate methods.

Message-passing. Message-passing algorithms (Bishop, 2006; Kschischang and Frey, 2001) have each variable/factor iteratively send to its neighbours messages encoding its current belief about each neighbour's min-marginals. Messages may be sent in parallel or sequentially, with different choices being better for different problem classes (Szeliski et al., 2008). These algorithms can be implemented straightforwardly, and parallelise easily. However, they do not guarantee convergence (indeed, they often exhibit oscillatory behaviour) and are out-performed on most problems by more modern techniques (Kappes et al., 2015). Tree-reweighted methods (Kolmogorov, 2006; Wainwright et al., 2005; Kolmogorov, 2015) use a message-passing formulation, but actually solve a Lagrangian dual of the LP relaxation (see above), and can provide a certificate of optimality where relevant. Similarly, the LP-based method of Globerson and Jaakkola (2007) is implemented in terms of message-passing.

Surveys on inference algorithms. The survey papers by Kolmogorov and Rother (2006), Szeliski et al. (2008), Andres et al. (2010), Alahari et al. (2010), and Kappes et al. (2015) each evaluate a number of algorithms on various problems, comparing their performance. Kolmogorov and Rother (2006) focus on stereo matching and consider highly-connected grid models defined on pixels with unary and pairwise factors only. They evaluate three inference algorithms (graph-cuts, TRW-S, and belief propagation). Szeliski et al. (2008) consider a wider selection of problems—stereo matching, image reconstruction, photomontaging, and binary segmentation—but with 4-connectivity only, and apply a wider range of algorithms, adding ICM and α-expansion to the above. Recently, Andres et al. (2010) and Kappes et al. (2015) substantially widened the scope of such analysis, by considering also models with higher-order factors, regular graphs with denser connectivity, models based on superpixels with smaller number of variables, and partitioning problems without unary terms. They compare the performance of many different types of algorithms on these models, including some

specialised to particular problem classes.

These surveys help to understand the space of existing algorithms and provide a guide to which algorithms are suited for which models. Our work takes a natural step forward, with a technique to automatically select the best algorithm to run on an input problem instance.

5.1.4 Inferning

Our work is a form of *inferning* (Doppa et al., 2013), as it considers interactions between inference and learning. A few such methods use learning to guide the inference process. Unlike the hard-wired inference algorithms described in Section 5.1.3, these approaches learn to adapt to the characteristics of a particular problem class. Some operate by pruning the model during inference, by learning classifiers to remove labels from some variables (Guillaumin et al., 2013; Conejo et al., 2014), or to remove certain factors from the model (Stoyanov and Eisner, 2012; Roig et al., 2013). Jiang et al. (2013) instead learn an optimal sequence of operations to perform during conventional message-passing inference.

Our work operates at a higher level than these approaches. Instead of incorporating learning into an algorithm to allow adaptation to a problem class, we instead learn to predict which of a fixed set of hard-wired algorithms is best to apply to a given problem instance. While this approach has not been applied to MAP inference previously, the next section describes other domains in which automatic algorithm selection has been explored.

5.1.5 Automatic algorithm selection

Algorithm selection as an abstract problem was first considered by Rice (1976), who analysed optimisation-based approaches to selecting algorithms for quadrature and process scheduling. He assumes that the performance characteristics of each algorithm are known as a function of its hyperparameters and features of the target problem instance. Given this, the selection problem is formulated directly as an optimisation problem, with no learning required. For quadrature problems, the features include function smoothness and presence of singularities; for process scheduling, they include the system load and various characteristics of jobs.

Applications to combinatorial search problems. Much research on automatic algorithm selection has focused on algorithms for various combinatorial search problems;

Kotthoff (2016) provides a comprehensive survey. While the first such systems were based on hand-specified rules, the majority of research (and that relevant to our approach) has been on systems that learn from a training set of problems. The actual selection methods in these systems vary, and include support vector machines, neural networks, decision trees, and various other standard machine-learning models (Kotthoff, 2016). In all cases, these take manually-designed features of the problem instances as input, and are trained using the measured performance of all available algorithms on all available problems. We now discuss some representative examples of such systems.

Xu et al. (2008) address the boolean satisfiability problem (SAT); their system includes seven algorithms, and was trained on 4811 problem instances. They extract 48 features, capturing the size and complexity of the problem, and various other measures known to correlate with algorithm performance. The selection model first predicts the runtime of each algorithm using ridge regression (Bishop, 2006), then chooses the one predicted to be fastest.

Pulina and Tacchella (2009) address the quantified boolean formula problem; their system includes up to 16 algorithms, but the best empirical results were achieved when using just two. They use a classification approach that directly selects the best algorithm given problem features, and experiment with various underlying predictors including decision trees and nearest-neighbour.

Kotthoff et al. (2015) address the travelling salesman problem; they select from four algorithms, and experiment with various different selection models—both direct classification to predict the best algorithm, and regression to predict the performance of each algorithm. Their best system uses regression splines to predict performance, taking 13 features as input. The rather small pool of algorithms and high cost of extracting features from problem instances in this domain result in only a small gain over simply running the best algorithm.

Finally, some works present more general selection frameworks that can be applied separately to several different problem domains, using different algorithms and features for each, but the same overall architecture. A recent example is that of Lindauer et al. (2015), which is applied to six problem domains, each having 300–4000 instances, 2–30 algorithms, and using 15–200 features. Moreover, this method also predicts optimal hyperparameters for the selected algorithm. Overall, this method achieves on average 3.9× better performance than simply using the single best solver for each domain.

Applications to machine learning problems. In machine learning, automated algorithm selection is typically referred to as meta-learning (Vilalta and Drissi, 2002; Smith-Miles, 2009). One of the earliest works in this area is that of Aha (1992), which automatically derives rules describing how problem characteristics affect the performance of three different classification algorithms. Brazdil and Henery (1994) and more recently Ali and Smith (2006) extend the same line of work, to include 100 classification problems, selecting from eight algorithms. As the selection model, they use a decision tree classifier; this takes 31 problem features as input, and directly outputs the choice of algorithm. Ali and Smith-Miles (2006) slightly extend the same dataset of problems, but rather than selecting an algorithm, they select which kernel to use in a support vector machine. Going beyond classification, other works take similar approaches to selecting algorithms for regression (Köpf et al., 2000), or time-series forecasting (Venkatachalam and Sohl, 1999). However, no prior works consider selecting algorithms for MAP inference.

A related but distinct line of research aims to find the optimal algorithm and/or parameters for a specific problem instance, by optimising for that individual instance. Instead of learning to generalise across instances, these methods directly run a subset of the possible choices of algorithm and hyperparameters. Thornton et al. (2013) present a system integrated with the well-known WEKA machine-learning toolkit (Hall et al., 2009), including 39 classification methods, and various combinations thereof. This jointly selects the best algorithm and its hyperparameters, and compares two different optimisation techniques (Hutter et al., 2011; Bergstra et al., 2011) which allow exploring promising regions of the hyperparameter space, without exhaustively evaluating every possible combination. For optimising hyperparameters of a single algorithm, Snoek et al. (2012) model performance according to a Gaussian process (Rasmussen and Williams, 2005). This represents the expectation and uncertainty in algorithm performance as a function of the hyperparameters. Again, this allows running experiments to explore promising areas of the search space; it leads to state-of-the-art performance on various datasets.

5.2 Dataset of models

In this section we describe the dataset of GMs that we use to train and evaluate our algorithm selection framework.

OpenGM2 (Kappes et al., 2015). The OpenGM2 dataset contains GMs drawn from 28 problem classes, including pairwise and higher-order models from computer vision and bioinformatics; it is the largest dataset of GMs currently available. We briefly summarise here the main kinds of problems; the reader may refer to Kappes et al. (2015) for additional details.

- low-level vision problems such as stereo matching (Szeliski et al., 2008), inpainting (Lellmann and Schnörr, 2011; Nowozin et al., 2011), and montaging (Szeliski et al., 2008). These are all locally-connected graphs with variables corresponding to pixels, and with pairwise factors only; label counts vary widely between classes, from 2–256.

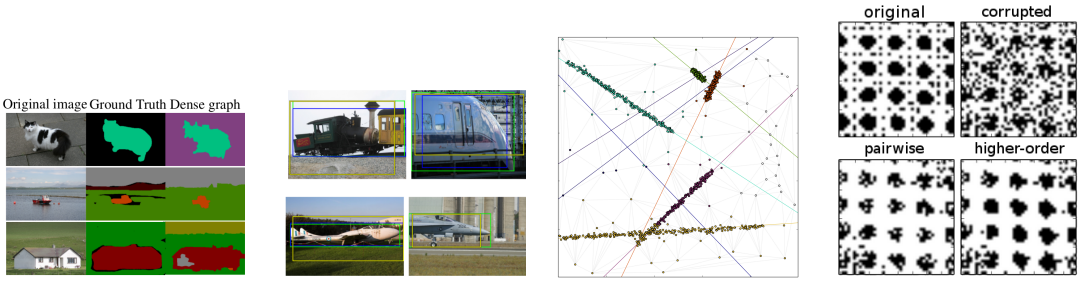
- small semantic segmentation problems with up to eight classes, with labels corresponding to surface types (Gould et al., 2009) and geometric descriptions (Hoiem et al., 2011). These are irregular, sparse graphs over superpixels; Gould et al. (2009) use pairwise factors only, while Hoiem et al. (2011) include general third-order terms.

- partitioning (unsupervised segmentation by clustering) based on patch similarity, operating on superpixels and with as many labels as variables, in both 2D (Kim et al., 2011; Andres et al., 2011; Brandes et al., 2008) and 3D (Andres et al., 2012b). Potts or generalised Potts factors are used in all cases; the graphs of Kim et al. (2011) use very large cliques with up to 300 variables, while the other classes are pairwise or third-order, with just one class having dense connectivity.

- two problem classes from bioinformatics: protein side-chain prediction (Jaimovich et al., 2006), and protein folding (Yanover et al., 2008); both are defined over irregular graphs, with Jaimovich et al. (2006) using only two labels but general third-order factors, and Yanover et al. (2008) using up to 503 labels and dense pairwise connectivity.

We complement the OpenGM2 dataset with four additional, interesting problem classes which arise in modern computer vision applications (Figure 5.3); these classes are described below.

Semantic segmentation with context (Guillaumin et al., 2013). Semantic segmentation on the MSRC-21 dataset (Shotton et al., 2009) with relative position factors. Each problem instance corresponds to a single image. Variables correspond to superpixels and labels to 21 object/background classes (*e.g.* car, road, sky). Unary factors



msrc-segmentation	**joint-localisation**	**line-fitting**	**texture-restoration**
$\|V\| = 275$	$\|V\| = 81$	$\|V\| = 661$	$\|V\| = 1900$
$L = 21$	$L = 100$	$L = 51$	$L = 2$
$O = 2$	$O = 2$	$O = 2$	$O = 49$

Figure 5.3: Our four added GM classes. Variable count $|V|$ is a mean over all instances; L is mean label count, and order O is largest factor clique size

are given by appearance classifiers on features of a superpixel, while pairwise factors encode relative location in the image, to favour labellings showing classes in the expected spatial relation to one another (*e.g.* sky above road). The model is fully connected, *i.e.* there is a pairwise factor between every two superpixels in the image.

Joint localisation (Guillaumin et al., 2013). Joint object localisation across images on the PASCAL VOC 2007 dataset (Everingham et al., 2007). Each problem instance corresponds to the set of images containing a certain object class. Variables correspond to images and labels to object proposals (Alexe et al., 2010) in the images. Unary factors are given by the objectness probability of a proposal (Alexe et al., 2010), while pairwise factors measure the appearance similarity between two proposals in different images. Inference on this model will select one proposal per image, so that they are likely to contain objects and to be visually similar over the images.

Line fitting (Isack and Boykov, 2012). Fitting of multiple lines to a set of points in $\mathbb{R}^2$. This is an alternative to RANSAC (Fischler and Bolles, 1981) for fitting an unknown number of geometric models to a dataset. Variables correspond to points and labels to candidate lines from a fixed pool (sampled from the point set in a preprocessing stage). Unary factors favour labelling a point with a nearby line, while pairwise factors promote local smoothness of the labelling (*i.e.* nearby points should take the same label).

Texture restoration (Rother et al., 2009). Binary texture restoration with pattern potentials. Given a binary image corrupted by noise, the task is to reconstruct the original

noise-free image, while preserving the underlying texture regularity. Variables correspond to pixels, and labels to 'on' or 'off'. Unary factors penalise deviations from the input noisy image, while pairwise factors prefer pixels at certain offsets taking certain pairs of values (learnt on a training image showing a noise-free texture). Higher-order factors reward image patches for taking joint labellings which occur frequently in the training image (patterns). The pairwise and higher-order factors capture low and high order texture properties, respectively.

Data diversity. From each problem class we take all instances up to a maximum of 20. This results in a diverse dataset of 344 problem instances drawn from the 32 classes; 224 of these instances are pairwise and 120 higher-order. 21 of the problem classes have small label-spaces (< 20 labels), while the remainder vary greatly up to a maximum of 17074. Variable counts similarly cover a wide range, from 19 to 2356620, with a median of 10148. Amongst the higher-order problems, 58% of instances have arbitrary dense factor tables, while the remainder have Potts potentials (Boykov et al., 2001b) or generalised versions thereof (Kohli et al., 2007, 2008). The problem classes also differ greatly in the degrees of homogeneity of their instances. For example, instances in the *line-fitting* class vary by an order of magnitude in variable and label counts, whereas all instances in the *inclusion* class have identical characteristics but for the factor energies themselves.

5.3 Inference algorithms and performance measures

In this section we describe the set of MAP inference algorithms that we incorporate in our algorithm selection framework.

Inference algorithms. As noted in Section 5.1.3, a vast number of MAP inference algorithms have been proposed in the literature, with differing approaches, degrees of generality, and performance characteristics. We selected 15 to use in our experiments (Table 5.1), including representative algorithms from most prominent approaches, *e.g.* move-making, message-passing, dual-decomposition, globally optimal, *etc.* This covers many of the most commonly used algorithms in computer vision, such as TRW-S (Kolmogorov, 2006), QPBO (Rother et al., 2007), and α-expansion (Boykov et al., 2001b). Note however that we do not aim to form an exhaustive pool of all good algorithms; our automated selection method is agnostic to the pool of algorithms it is trained to select from, and explicitly avoids making prior assumptions on their appli-

cability.

We also include a simple method, dubbed *unary-modes (UM)*, which labels each variable by minimising its unary factors only; this should perform poorly on genuinely hard structured prediction problems, where the non-unary factors have a decisive impact on the MAP labelling.

Protocol for inference. We used the original authors' implementation of each algorithm where available, and the implementations of Kappes et al. (2015) otherwise. Every algorithm was run on every problem instance in our dataset, with limits of 60 minutes CPU time and 4GB RAM imposed for inference on one instance. For each successful run, we recorded the MAP labelling and time taken.

Many of the algorithms have free parameters that must be defined by the user. While it was not practical to evaluate every possible combination of parameters, for several of the algorithms we included multiple parameterisations where this affects their results significantly. For example, we ran four versions of loopy belief propagation, with damping set to 0.0 and 0.75, and maximum iteration counts of 50 and 250. In such cases, the different parameterisations are combined to create a meta-algorithm, which simulates the user running every parameterisation, then taking the results from that yielding lowest energy on the problem instance.

Several incompatible combinations of algorithms and GMs were included. When possible, we still ran the algorithm to obtain an approximate solution:

- higher-order factors are omitted when passing GMs to pairwise algorithms. However, when evaluating the algorithm's performance, the energy of the output labelling is still computed on the full model including all factors.
- non-metric pairwise factors passed to α-expansion are handled as if they were metric, sacrificing the usual correctness and optimality guarantees (Boykov et al., 2001b).

When it was not possible to run the algorithm, we counted this as a failure:

- QPBO aborts when presented with a GM having non-binary variables.
- FastPD aborts when presented with a GM whose pairwise factors are not all proportional to some uniform distance function on labels.
- Kernighan-Lin aborts when presented with a GM having factors that are not pairwise Potts

alias	order	#p	name / description	reference
A*	*all*		implicitly convert to shortest-path problem and apply A*	Bergtholdt et al. (2010)
AD^3	*all*		alternating directions dual decomposition with branch and bound	Martins et al. (2015)
α-exp	*pw*		alpha-expansion	Boykov et al. (2001b)
BPS	*all*	4	sequential loopy belief propagation, implementation of Kappes et al. (2015)	Bishop (2006)
DDS	*all*	2	dual decomposition with subgradient descent	Kappes et al. (2012)
FPD	*pw*	3	fast primal/dual (FastPD)	Komodakis et al. (2008)
ICM	*all*		iterated conditional modes	Besag (1986)
ILP	*all*		solve as integer programming problem with Gurobi	Kappes et al. (2015)
KL	*pw*		Kernighan-Lin method for 2nd order partitioning problems	Kernighan and Lin (1970)
LBP	*all*	4	parallel loopy belief propagation, implementation of Kappes et al. (2015)	Bishop (2006)
LP	*all*		solve linear programming relaxation with Gurobi	Kappes et al. (2015)
MPLP	*all*		max-product linear programming with cutting plane relaxation tightening	Sontag et al. (2008, 2012)
QPBO	*pw*		quadratic pseudo-boolean optimisation	Rother et al. (2007)
TRW-S	*pw*	3	sequential tree-reweighted message-passing	Kolmogorov (2006)
UM	*all*		take lowest-energy label according to unary factors only	-

Table 5.1: Algorithms used in our selection framework, including the GM orders they are applicable to (pw = pairwise), number of parameter settings included if more than one (#p), full name or description, and reference to the original work. More details on the algorithms are given in Section 5.1.3.

	% instances for which...			mean time /s
	completes	*best-&-fastest*	*good-&-fastest*	
A*	4	0	0	0.1
AD^3	52	7	1	390.2
α-exp	98	5	7	23.4
BPS	72	4	2	158.3
DDS	80	0	0	296.6
FPD	31	9	22	7.2
ILP	48	1	0	96.3
LP	52	2	1	76.8
ICM	100	30	31	60.7
KL	12	10	10	142.2
LBP	73	6	4	193.5
MPLP	56	1	1	1116.3
QPBO	12	0	2	0.1
TRW-S	94	19	10	236.4
UM	100	0	3	0.1

Table 5.2: Aggregate performance of each inference algorithm on our dataset; mean time is over instances for which the algorithm successfully returns a result

Performance measures. We measured three aspects of the performance of each algorithm:

- *completes:* whether the algorithm runs to completion, *i.e.* returns a solution within 60 minutes, regardless of the energy of that solution.
- *best-and-fastest:* whether the algorithm reaches the lowest energy among all algorithms, faster than any other one that does so. This is relevant for a user requiring the solution with lowest possible energy, even at high computational cost.
- *good-and-fastest:* whether the algorithm is the fastest to reach a solution with 98% of variables matching the lowest energy labelling. This is highly relevant in practice, as minor deviations from that labelling may not matter to the user, while achieving it would require a significantly slower algorithm.

Table 5.2 shows the performance of the algorithms with respect to these measures.

Algorithm diversity. We see that the distributions of both best-and-fastest and good-and-fastest algorithms over instances have high entropy—many different algorithms are best-and-fastest or good-and-fastest for a significant fraction of GMs. 11 of the 15 algorithms are able to return a solution for at least one instance on more than half of the problem classes; the other four are particularly restricted, such as QPBO (which only operates on binary problems). All the algorithms other than A* and DDS are the best-and-fastest for at least one problem instance. TRW-S and FastPD perform particularly well on pairwise problems, with TRW-S generally reaching slightly lower energies, but FastPD being much quicker. Kernighan-Lin outperforms all algorithms on pairwise partitioning problems. AD^3 gives low energies for high-order problems, but often takes longer than other algorithms. Only ICM and unary-modes are able to return a solution for all problem instances. Although they are fast and widely-applicable, these naïve methods are unable to return the best solution in the majority of cases. All these observations show how our goal of learning to select the best inferencer is much harder than simply picking any algorithm that runs to completion.

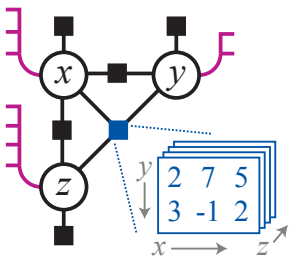
5.4 Learning to select an algorithm

We now consider how to automatically select the best MAP inference algorithm for an input problem instance. This is our main contribution in this chapter. We define two tasks: (1) predicting the best-and-fastest algorithm; and (2) predicting the good-and-fastest algorithm. To address these tasks, we design selection models that take a GM as input, and select an algorithm as output (Section 5.4.2). The selection models operate on features extracted from the GMs themselves (Section 5.4.1). This is different from the typical approach in computer vision of extracting features from images and using these to build a GM.

5.4.1 GM features

We extract the following three groups of features from each problem instance (Figure 5.4).

Instance size. The number of variables, $|V|$, and of factors, $|F|$, are used to indicate the overall size of the problem instance, hence whether slower algorithms are likely to



(a)

$|V| = 3$

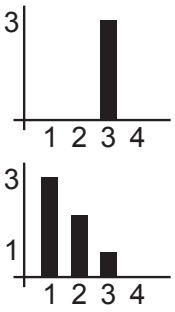
$|F| = 6$

$L_{min} = 2$

$L_{max} = 4$

$L_{mean} = 3$

(b) size



(c) variable (top) and factor (bottom) orders

order	density
2	$\frac{2}{\binom{3}{2}} = \frac{2}{3}$
3	$\frac{1}{\binom{3}{3}} = 1$
4	0

(d) factor densities

Figure 5.4: Example GM structure (a) and associated features (b-d). Circles correspond to variables, and squares to factors; the label space of each variable is shown as purple dashes. Part of the value table for the third-order factor (blue) is also shown

be applicable. We also include the minimum, maximum and mean label count over all variables. See Figure 5.4b.

Structural features. We extract more sophisticated features based on the model structure, *i.e.* which do not depend on the factor values themselves. Firstly, we take a histogram (Figure 5.4c) and statistics (minimum, maximum, mean) of both:

- variable orders, *i.e.* for each variable, the number of factors connected to it
- factor orders, *i.e.* for each factor, the number of variables in its clique

Secondly, we measure factor densities—for each factor order $M \geq 2$, the number of factors of order M divided by the binomial coefficient $\binom{|V|}{M} = \frac{|V|!}{M!(|V|-M)!}$. Intuitively, this is the number of possible M-cliques that actually have an associated factor. In Figure 5.4, this is 1 for third order, as there is only one possible triplet, but $2/3$ for second order, as only two of the possible three pairs of variables have a pairwise factor: (x,y) and (x,z) but not (z,y).

Energy features. Our final group of features depend on the values of the factors themselves, *e.g.* the blue values in Figure 5.4a. To determine the influence of different orders of factors, we compute means and deviations over values they take, defined as follows:

- for each factor $f \in F$, let μ_f be the mean and σ_f the standard deviation of all unique values taken by f

- then, for each factor order $M \geq 2$, compute for factors F_M of that order, the ratio of each of the following to the same quantity for $M = 1$:
  (i) $\sum_{f \in F_M} \mu_f$ (ii) $\sum_{f \in F_M} \mu_f / |F_M|$ (iii) $\sum_{f \in F_M} \sigma_f / |F_M|$.

Intuitively, these capture how much influence each order of factor has on the final energy, *i.e.* how much changing the labelling will change the values of factors of each order. On pairwise GMs, a large influence of the pairwise (as opposed to unary) factors makes inference harder; on higher-order GMs, pairwise algorithms should perform relatively well only when the influence of higher-order factors is small. We also count the fraction of pairwise factors f having each of the following characteristics for all labels a, b, c:

(i) $f(a,b) = 0$ iff $a = b$ (ii) $f(a,b) \geq 0$
(iii) $f(a,b) = f(b,a)$ (iv) $f(a,b) + f(b,c) \geq f(a,c)$.

Together, these are the conditions for a factor to be metric; without (iv), to be semi-metric—respectively requirements for the α-expansion and $\alpha\beta$-swap algorithms to fulfill their correctness guarantees (Boykov et al., 2001b). Finally, we measure the fraction of pairwise factors which are submodular; in general pairwise submodular problems are easier to solve by LP-based methods, as their LP relaxation is tight.

5.4.2 Algorithm selection models and their training

Selection models. We propose two algorithm selection models. Each is a 1-of-N classifier implemented as a random forest (Criminisi et al., 2011), taking the features described in Section 5.4.1 as input. Model *BF* is trained to predict the best-and-fastest algorithm; model *GF* is trained to predict the good-and-fastest algorithm. The random forests are trained recursively by selecting the best split from a randomly-generated pool at each step, using information gain (*i.e.* entropy decrease) as the criterion, and with outputs modelled by categorical distributions (Criminisi et al., 2011).

Data. We train the selection models on a subset of our dataset (Section 5.2). A training sample consists of features extracted from a problem instance and a target output label denoting which algorithm works best on it. It is important to note that these training labels are automatically generated by running all algorithms on the training instances, as in Section 5.3. No human annotation is required. At test time, we run the selection models on a separate subset of the dataset. The evaluation compares the algorithm selected by our model to the one known to perform best (Section 5.5).

Again, this test label is produced automatically.

5.5 Experiments and analysis

In this section, we describe the experimental setup used to evaluate our algorithm selection models, then present and analyse the results.

Tasks and baselines. We report results on the two tasks defined in Section 5.4: (i) predicting the good-and-fastest inference algorithm for an input GM; and (ii) predicting the best-and-fastest algorithm. Task (i) is addressed by the selection model GF, and task (ii) by model BF (Section 5.4.2). For both tasks, we also analyse the performance of two baseline methods that select an algorithm without looking at features of the input instance. The first baseline *NB* always selects the algorithm that is most often best over the full training set. This mimics the behaviour of a naïve user who simply chooses one commonly good algorithm to use. For the second, stronger baseline *SB*, we assign each of the problem classes to one of three superclasses:

1. pairwise—many algorithms are designed for pairwise problems only;
2. higher-order—there exist algorithms designed explicitly to handle higher-order factors, but which may be slow for pairwise instances;
3. partitioning—these are a special class which is hard for general algorithms (due to having a large label space, and being invariant to label permutations) but certain methods can exploit this structure to solve them efficiently; most partitioning problems in our dataset are pairwise, but some are third-order.

Then, at test time, each problem instance is assigned the algorithm that is most often best for training problems of its superclass. This strong baseline mimics the behaviour of a user with good working knowledge of inference—enough to recognise how her problem fits in these superclasses, and to know which algorithm will be best for each.

Training and evaluation protocols. We select half the problem instances at random to train on, and the remainder are used for testing. As discussed in Section 5.3, the ground-truth labels marking which inference algorithms perform best on a problem instance are obtained automatically by running all algorithms on all instances. No human annotation is necessary for training or testing.

When training and evaluating selection models, the underlying problem classes are always treated as unknown—they are not provided as input data. We want the

	good-and-fastest			**best-and-fastest**		
algorithm selected by...	*GF*	*NB*	*SB*	*BF*	*NB*	*SB*
% instances correctly classified	69	31	28	62	30	36
mean % matching variables	96.4	75.3	87.5	97.1	75.4	95.6

Table 5.3: Performance of our model GF and baselines NB and SB for selecting the good-and-fastest algorithm (first three columns), and performance of our model BF and baselines for selecting the best-and-fastest algorithm (last three columns).

selection models to freely learn the optimal association between GM features and good algorithms to run. The GM features we propose are designed to enable the selection models to reason upon various properties of GMs, which can be used to characterise problem classes (*e.g.* connectivity structure and distributions of energy values in the factors). So, we might expect the selection models to learn at least some of the problem class structures, given that this often correlates with the best algorithms (Section 5.3).

Evaluation measures. Our algorithm selection models are evaluated on the test set with the following measures:

- percentage of instances with the correct algorithm (best-and-fastest or good-and-fastest) selected. This is the measure for which we trained our selection models.
- mean (over instances) of fraction of variables matching the labelling returned by the best-and-fastest algorithm. This is particularly relevant in practice, as users typically care about the quality of the labelling output, by an algorithm, especially in terms of how close it is to the lowest-energy labelling that could have been returned.

Table 5.3 presents our results for the tasks and evaluation metrics described above.

Predicting the good-and-fastest algorithm. Model GF correctly chooses the good-and-fastest algorithm for 69% of instances, with 96.4% of variables taking the same label as in the true best labelling on average. This compares favourably to the naïve baseline NB, which correctly selects only on 31% of the instances and returns labellings that are considerably worse (75.3% correctly-labelled variables on average). Indeed, our model also substantially outperforms the strong baseline, which only achieves an average of 87.5% of correctly-labelled variables.

mean...	time /s	speed-up	matching variables
exhaustive	13046.8	1.0×	100%
good-and-fastest	221.3	88.1×	96.4%
best-and-fastest	312.5	46.8×	97.1%

Table 5.4: Mean times and speed-ups from using our method, versus exhaustively applying all algorithms. *matching variables* is fraction of variables whose labels match true best result; *speed-up* is ratio of time to that for exhaustive testing

These results show that our selection model successfully generalises to new problem instances not seen during training. It is able to select an algorithm much better than even the strong baseline of a user who knows which algorithm performs best for similar problems in the training set.

Predicting the best-and-fastest algorithm. Model BF correctly selects the best-and-fastest algorithm on 62% of instances, exceeding the naïve baseline (32%). This results in 97.1% of variables taking the same label as in the true lowest-energy solution, greatly exceeding the naïve baseline of 75.4%. Our model performs well against even the strong baseline, which only classifies 36% of instances correctly and has a slightly lower fraction of correct variables at 95.6%.

Efficiency. A simple alternative to our selection method would be to run every algorithm on the test problem instance, and select the lowest-energy solution. However, this is computationally very expensive. To evaluate the speed-up made by our method, for each problem instance we also measured (i) the total time to run every inference algorithm; (ii) the time to predict the best-and-fastest algorithm with model BF then run it; and, (iii) the time to predict the good-and-fastest algorithm with model GF then run it. As we see in Table 5.4, our method results in an average speed-up of 46.8× using model BF, and 88.1× using model GF, with 97.1% and 96.4% of variables correctly labelled respectively. Thus, automated selection achieves labellings very similar to running every algorithm, but at a small fraction the computational expense. Model GF yields a significantly faster-running algorithm on average than model BF, with only a small drop ($< 1\%$) in variables correctly labelled.

Algorithms selected by the strong baseline. As described in Section 5.5, our strong baseline chooses the algorithm that is most often best-and-fastest or good-and-fastest over the training set, for problems in the same superclass as the test instance. For pre-

	AD^3	α-exp	BPS	FPD	ICM	KL	LBP	QPBO	TRW-S	UM
AD^3	0	0	0	0	2	0	0	0	0	0
α-exp	0	5	1	1	0	0	0	0	2	0
BPS	0	0	1	0	2	0	1	0	1	0
FPD	0	0	1	19	0	0	0	0	0	0
ICM	0	0	0	0	16	1	0	0	3	0
KL	0	0	0	0	0	24	0	0	0	0
LBP	0	0	0	0	3	0	4	0	0	1
QPBO	0	0	3	0	0	0	0	3	0	0
TRW-S	0	1	0	6	1	0	1	0	6	0
UM	0	1	0	1	0	0	0	1	0	0

Table 5.5: Confusion matrix showing true (rows) and predicted (columns) good-and-fastest algorithms for pairwise problems. The table only includes those algorithms that are the true good-and-fastest for at least one problem instance.

dicting the best-and-fastest algorithm, Kernighan-Lin is selected for partitioning problems, TRW-S for other pairwise instances, and AD^3 for other (higher order) instances. However, for the good-and-fastest algorithm, FastPD is selected instead of TRW-S for pairwise instances, and ICM for higher order instances, indicating that these often label 98% or more of variables correctly, while being faster to run.

Algorithms selected by our method. At a coarse level, for the task of selecting the best-and-fastest algorithm, we find that our model BF most often chooses pairwise-specific algorithms for pairwise problems, and AD^3 for higher-order problems. This agrees with intuition—pairwise algorithms are specifically designed to be faster for pairwise instances, while AD^3 is a good general-purpose algorithm for higher-order instances. Interestingly, for the good-and-fastest task, model GF correctly learns to choose ICM or a good pairwise method for higher-order problems in place of AD^3—for many instances, these provide solutions close in labelling to the lowest-energy, and do so much faster than AD^3.

To explore whether our method can also draw more subtle distinctions, we now examine the distribution of algorithms it selects for pairwise problems. 10 of the algorithms we consider are useful for these, in the sense of being good-and-fastest for at least one instance. Table 5.5 shows the confusion matrix for true and predicted good-and-fastest algorithms amongst these 10. Certain groups of problems can be dis-

	good-and-fastest			best-and-fastest		
algorithm selected by...	*GF*	*NB*	*SB*	*BF*	*NB*	*SB*
% instances correctly classified	40	26	11	28	25	23
mean % matching variables	89.8	73.3	71.7	85.5	73.3	86.2

Table 5.6: Performance of algorithm selection methods selecting good-and-fastest and best-and-fastest algorithms in the LOCO regime; see Table 5.3 for details.

tinguished based on structural properties, such as partitioning problems to be solved with KL, or very large instances that only run to completion with ICM. Our model correctly makes these distinctions. Other distinctions are even more subtle—such as whether to use α-expansion, TRW-S, or FastPD for a pairwise problem of moderate size. Our model is able to select between these three algorithms, making the correct choice for 75% of instances.

LOCO regime. We also tested our models and baselines in an even harder 'leave one class out' (LOCO) regime, where for each problem class C in turn, we train on all instances from classes other than C, and test on those in C; the final performance is given by a weighted mean over classes. This tests generalisation to classes absent from the training set, which is relevant when the user does not wish to train our model on her classes. The results are presented in Table 5.6.

For selecting the good-and-fastest algorithm, model GF still performs well in LOCO regime, selecting algorithms labelling 89.8% of variables correctly, and exceeding both naïve and strong baselines by over 15%. Moreover, we correctly choose the good-and-fastest algorithm 14% more often than the baselines.

For the best-and-fastest task, model BF results in 85.5% of variables being correctly labelled, significantly exceeding the naïve baseline at 73.3% and comparable with the strong baseline at 86.2%. These results demonstrate that our selection models are strong enough to generalise *across* the hidden problem classes, going beyond discovering and recalling distinguishing features of these.

5.6 Discussion and conclusions

In this section, we present some options for future research and then conclude.

5.6.1 Future work

More algorithms. New inference algorithms are being published regularly; it would be interesting to incorporate some of these in our selection framework. This is in fact straightforward—it just requires running the algorithms over our training set of GMs, and retraining the classifier accordingly. For example, two generalisations of TRW-S to higher-order problems have recently been published (Kolmogorov, 2015; Schoenemann and Kolmogorov, 2014). These tend to yield similar energies, but for many problem classes, one is faster than the other. It would be interesting to see if our selection model is able to predict which of these to use for a given instance (or whether an older algorithm such as MPLP is sufficient).

More problem classes. The dataset of GMs we use here is the most diverse collection available; it extends the largest previously-published dataset OpenGM2 (Kappes et al., 2015) with four new classes. However, it is not exhaustive: there exist many other vision tasks that are solved with these models, and different formulations even for the same tasks. As such, it would be interesting to extend the dataset of models even further, both to learn from a wider range of problems, and to see if our selection method is still successful.

More features. Our features are designed to capture relevant information about the GM, while being fast to compute; however, there are still problem characteristics that they do not capture. For example, it would be interesting to add features that better represent whether the LP relaxation for a higher-order GM is tight, as this has a strong effect on the performance of LP-based methods, but isn't easily inferred from the present set of features.

Selecting hyperparameters. For five of the algorithms considered, we included multiple hyperparameter settings, but did not attempt to predict which is best. However, it would be interesting to do so, as all the algorithms have some hyperparameters, which may significantly affect performance. One straightforward approach would be to treat different hyperparameter settings as different algorithms. Unfortunately this would lead to impractically-many different choices to select from: most algorithms have several hyperparameters, which may take a wide range of values. This could be mitigated by a cascaded system, that first predicts the best algorithm (and maybe very coarse hyperparameter settings), and then uses a per-algorithm model to regress the optimal hyperparameters.

Selecting algorithms for sub-problems. Some algorithms, such as α-expansion and fusion moves, solve sub-problems that themselves require an inference algorithm. The choice of this inner algorithm is itself a hyperparameter, as are its own hyperparameter values. It would be possible to select the inner algorithm at the same moment as the outer, or alternatively, to select a different inner algorithm for each iteration of the outer. For the latter to be feasible, the selection method must be extremely quick to evaluate; while our random forest classifier is already very fast, the feature extraction stage may require speeding up. If the sub-problems have the same structure from iteration to iteration, it may be possible to amortise the feature-extraction cost, only updating those which actually change.

As a further extension, the lazy flipper (Andres et al., 2012a) may be used to 'fine-tune' a labelling, after another algorithm has been run. It would be interesting to predict whether this tuning stage is worthwhile, and if so, what depth of search the lazy flipper should perform. Given that it would be run after an initial labelling has been computed, one could also include features of this labelling, not just of the problem, in the classifier inputs. For example, one could add features representing the proportion of factors or tractable sub-graphs of different orders that are locally minimised by the candidate labelling.

Learning to sequence algorithms. All the inference algorithms we consider in this chapter are iterative, and can be stopped after a fixed time or iteration count, regardless of whether they have reached convergence. Hence, it would be possible to select one algorithm, run it for a short time only, and then make a decision on whether to continue running it, or to refine the solution with a different algorithm, or to start over with a different algorithm. However, in this case, the space of possible sequences of algorithms that could be run on each problem becomes too large to test exhaustively, as we currently do to obtain training data for our model. Instead, we could use reinforcement learning, as applied to neural network architecture search by Zoph and Le (2017); in our case, the problem features would be the state, actions would correspond to a choice of algorithm and hyperparameters, and the reward would be the final energy (or fraction of variables labelled correctly, if known) after running that algorithm for a fixed length of time.

5.6.2 Concluding remarks

We have presented a method to automatically choose the best inference algorithm to apply to an input problem instance. It selects an inference algorithm that labels 96% of variables the same as the best available algorithm for that instance. Our method is over 88$\times$ faster than exhaustively trying all algorithms. The experiments show that our automated selection methods successfully generalise across problem instances and importantly, even across problem classes. Finally, we have suggested several promising directions for future research.

Chapter 6

Conclusions

In this concluding chapter, we first summarise the contributions and insights of this thesis. Then, we discuss potential avenues for future work that incorporate ideas spanning several of the earlier chapters.

6.1 Summary of insights and contributions

Here we briefly recapitulate our key contributions and insights:

- We have demonstrated that it is possible to train a CNN-based object detector end-to-end for mean average precision—that is, in a principled fashion using the same model at training and test time, without a surrogate loss. Moreover, this results in a detector with comparable accuracy to the same model trained with the less-principled window-classification loss

- We have shown that it is possible to learn a generative model over 3D shapes, with an inference network supporting single-image reconstruction and pose-estimation, without relying on explicit annotations. Our results demonstrate that incorporating shading information in the loss significantly improves reconstruction performance, and allows us to learn concave object classes. Moreover, with minimal levels of supervision (just pose annotations), this method performs almost as well as a method using full 3D supervision that was state-of-the-art at the time

- We have presented a new probabilistic generative model for room layouts, which is considerably faster to sample than prior works. It allows generating con-

strained layouts, which is important to the computer graphics community, and may also be applied as a prior for scene understanding tasks

- We have shown that it is possible to automatically select the best inference algorithms to apply to discrete energy minimisation problems, by unsupervised learning of a classifier operating on features extracted from the problem instances

6.2 Future work: holistic perspectives

In the concluding sections of the preceding four chapters, we have discussed extensions and future work applicable to each. We now discuss some directions for future work that draw together ideas presented in the different chapters.

6.2.1 From unsupervised pose-estimation to unsupervised detection

In Chapter 3, we described how to train a model that can perform 3D reconstruction and pose-estimation, learning purely from 2D images, without supervision. We mentioned in Section 3.6.2 that it would be worthwhile to add more degrees of freedom for object pose. If the pose is extended to include *position* (*i.e.* the object is no longer assumed to be centred at the origin), and the model learns to predict this through its encoder network, it is performing the task of 3D object *localisation*. This differs from detection in that it is assumed that exactly one instance of the target class is present in the image. It is natural then to consider relaxing this assumption—we remain in the single-class setting, but allow the number of instances to vary from zero to arbitrarily many. In this setting, the inference network would truly be performing object detection.

If the images have a fixed, known background, then this can be addressed as described for natural images in Section 3.6.2, by simple background subtraction, without an inference network. However, this is brittle for many domains (*e.g.* due to objects that touch), and a learning-based solution has the potential to be more reliable. Moreover, a learning-based method remains applicable even in the case of an unknown background, provided this has some smoothness or regularity that can be captured in a prior.

If we allow the number of objects to vary, then we must model the position, orientation and shape of each, as well as the number of objects present. Several approaches are possible, such as a hierarchical model that has an integer object-count variable and

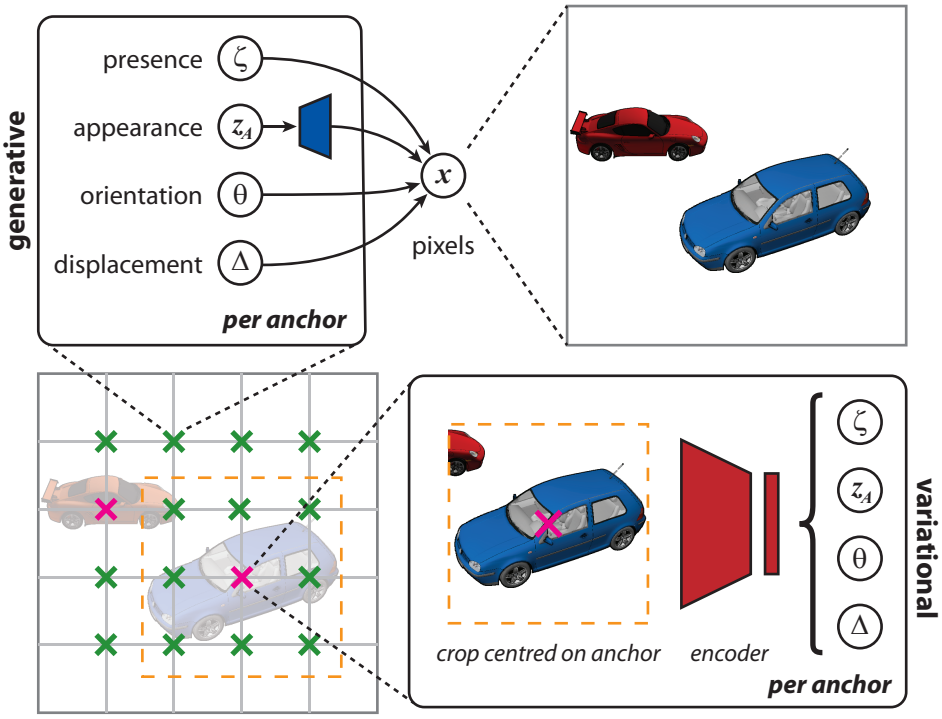


Figure 6.1: Extending the work of Chapter 3 to support varying numbers of objects at unknown locations. We define a grid of anchor points (crosses, bottom left) over the image area. The generative process for images, is that for each anchor point, we sample variables (top left) representing (i) whether there is an object present near the anchor (indicated here by the anchor being pink rather than green); (ii) its appearance embedding, decoded (blue) to give its vertex locations and colours; (iii) its orientation; (iv) its displacement, *i.e.* position relative to the anchor. All the objects marked as 'present' are rendered to produce the final image pixels (top right). To perform inference, for each anchor point, a crop is extracted from the image and passed to an encoder CNN (bottom right), which predicts the variables corresponding to that anchor. The model can then be trained with a variational loss for pixel reconstruction.

instantiates the location/*etc.* variables a corresponding number of times, or a model with a fixed maximum number of objects, and a binary presence indicator for each, in addition to their other parameters. One promising method, inspired by modern detection algorithms such as those of Ren et al. (2015) and Liu et al. (2016), would be to define a grid of *anchor* locations for the objects, each with a binary indicator for presence (Figure 6.1). Instead of modelling the position of an object in the global frame of reference, we can then simply represent the offset from the nearest anchor. This makes the inference problem spatially-local, as the encoder merely needs to predict for each

anchor, whether there is an object present nearby, and if so, what its parameters are.

As the existence of each object, whether represented by a count or binary indicators, is discrete, the reparameterisation trick (Kingma and Welling, 2014) can no longer be used directly to train the full model. One option to mitigate this would be to integrate over possible object counts; this is tractable if we use an explicit count variable and limit it to reasonable values, but exponentially-expensive if we have a large set of binary presence indicators. However, other gradient estimators could be used that are applicable to discrete variables, such as that of Mnih and Rezende (2016).

6.2.2 Indoor scene understanding with a prior on room layouts

Although Chapter 4 focused on the relevance of our probabilistic room layout model to the graphics community, it may also be applied as a prior in computer vision. Several downstream indoor scene understanding tasks could be considered, such as (i) semantic segmentation, (ii) 2D object detection, (iii) joint object detection and pose estimation (DPE), and (iv) 3D object detection. The latter two tasks extend 2D object detection, and are closely related to our work in Chapter 3. DPE produces 2D bounding boxes, but each is augmented with the object pose, typically the azimuth angle (Xiang et al., 2014; Su et al., 2015); 3D object detection instead produces bounding boxes in 3D space (Geiger et al., 2012).

One approach to leverage our layout model in these tasks would be to begin with a local (*i.e.* unstructured) predictor such as a CNN, that produces initial estimates of the output variables, *e.g.* scores, locations, and orientations of bounding-boxes for 3D object detection. These would be regularised by the layout model, with new, adjusted outputs computed, that increase the likelihood under the model, hopefully eliminating implausible arrangements while remaining close to the initial estimates. In the case of 3D object detection, a natural formulation would be an energy-based model over detection scores and locations that has unary factors favouring matching the output of the local predictor (CNN, *etc.*), and higher-order factors corresponding to the negative likelihood under the probabilistic layout model. Inference at test time would then yield the lowest-energy set of detections, balancing the unary and layout-regularisation energies. One difficulty here is that evaluating the likelihood of a layout under our model is currently intractable due to the large latent space, as noted in Section 4.6.2, so a solution to this would have to be found first.

In principle, a similar approach could be extended to the 2D tasks of semantic

segmentation and object detection. For example, for semantic segmentation, we could formulate an energy-based model over classes for each pixel, with unary factors favouring the output of the local predictor, and higher-order factors favouring likely layouts under the generative model. However, this is in fact more complex than the 3D case described above. In these 2D applications, it is actually a distribution over variables in *2D* space, such as detection windows or pixel classes, that is needed. This requires incorporating the camera parameters, which define the projection to pixels, as variables in the model, and marginalising over them, which is intractable with current techniques. One possibility would be to restrict the space of possible camera parameters by directly estimating them with a discriminative model, and either regularising towards these values or simply fixing them.

Training directly for 3D mAP-based losses. For both DPE and 3D object detection, the standard evaluation protocol uses a variant of mAP. This raises the possibility of using our methods from Chapter 2 to directly optimise for the test-time metric on detections, instead of a surrogate loss. Assuming the layout model were trained separately and kept fixed, the CNN parameters and relative weights of factors could then be optimised end-to-end for the final metric. For the methods of Chapter 2 to be applicable, this would require that gradients could be back-propagated through the test-time inference process, for example by using gradient descent for inference also (Domke, 2012; Maclaurin et al., 2015).

Selecting the inference algorithm. The energy-based approaches described above lead to rather complex, high-order models. In principle, our methods of Chapter 5 could be applied to select suitable inference algorithms for these. Different room types lead to differently-structured layouts (*e.g.* living rooms tend to have high-order groups of furniture, while toilets do not), and so it seems likely that different problem instances lead to different inference algorithms being best. However, our algorithm selection framework is currently focused on inference in discrete models. Object detection, both 2D and 3D, is naturally cast in terms of continuous-valued locations and orientations; as such, a different selection of algorithms would have to be used, or alternatively, the detection problem itself discretised somehow.

6.2.3 Unsupervised learning of joint layout and object models

We have discussed the possibility of learning object detection, reconstruction, and pose estimation without supervision (Section 6.2.1), and the possibility of incorporating

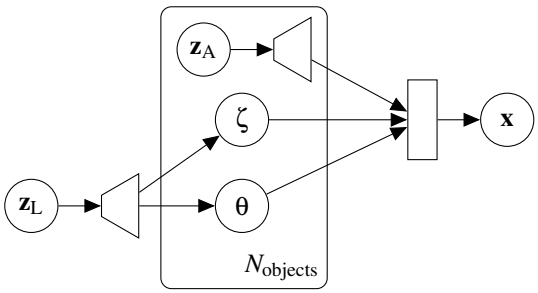


Figure 6.2: Plate diagram showing a possible generative model over appearance and layout of multiple objects. $\mathbf{z}_L$ is the Gaussian layout-embedding hyper-prior; $\mathbf{z}_A$ is the Gaussian appearance-embedding prior; ζ is the binary object presence indicator; θ is the object pose; $\mathbf{x}$ is the final pixels. The trapezoids represent deterministic decoder networks; the rectangle on the right represents the deterministic rendering operation

global layout priors into 3D object detection and similar tasks (Section 6.2.2). We now consider combining these two aspects. In particular, humans are able to simultaneously learn a *local* model of object appearance, composed with a *global* model of object layouts, with only very sparse supervision. In this final section, we discuss promising avenues for research in this direction.

Simple domain: traffic surveillance frames. In Section 3.6.2, we showed how our model for shape generation and reconstruction could be extended to a simple domain of natural images: frames from traffic surveillance cameras. There, we assumed that the object pose was fixed given the location, and treated the distribution of locations as uniform. However, in practice, locations are not uniform (cars are much more likely to appear centred in lanes), and the pose distribution could in theory be learnt instead of hand-specified. Moreover, the car locations are not independent, as two cars may not intersect one another in 3D space. Our task then is to learn the global distributions for location and pose, jointly with the local object appearance model. One approach is to place a learnt, joint prior on the locations and poses, itself having the same structure as the shape model—*i.e.* samples from Gaussian hyper-prior are decoded by a learnt network to yield a valid joint-sample of locations and poses (Figure 6.2). This allows joint training of the entire model using a variational objective derived from (3.7).

Extension to multiple classes. Our reconstruction and generation framework, and the extensions described so far, assume that all objects are of a single class. In order to support multiple classes, it is natural to replicate the shape model in a mixture, with the latent mixture component indicator corresponding to the object class. Training this in

an unsupervised fashion would be challenging, as the individual mixture components are very powerful models (due to the non-linear decoder network). One approach that may help is semi-supervised learning, where a small fraction of object instances are annotated with their class, in order to bootstrap the learning process.

Complex domain: indoor scenes. Given the extension to multiple classes just described, it should also be possible to include these in the global layout prior, with the discreteness handled similarly to object presence indicators. This results in a model with sufficient power to jointly model indoor scene layouts and object shapes. However, inference in that domain is significantly more challenging, as the background (*i.e.* walls and floor) vary in appearance, meaning that the model must disentangle this from the appearance of the foreground objects. It remains as future research to establish how this can be achieved.

Bibliography

Aanæs, H., Jensen, R. R., Vogiatzis, G., Tola, E., and Dahl, A. B. (2016). Large-scale data for multiple-view stereopsis. *International Journal of Computer Vision*, 120(2):153–168. 40, 41

Achlioptas, P., Diamanti, O., Mitliagkas, I., and Guibas, L. (2018). Learning representations and generative models for 3D point clouds. In *International Conference on Machine Learning*. 37, 50

Agarwal, S., Snavely, N., Simon, I., Seitz, S. M., and Szeliski, R. (2009). Building Rome in a day. In *Proceedings of the International Conference on Computer Vision*. 38

Aha, D. W. (1992). Generalizing from case studies: A case study. In *International Conference on Machine Learning*. 123

Alahari, K., Kohli, P., and Torr, P. (2010). Dynamic hybrid algorithms for discrete map mrf inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(10):1846–1857. 113, 120

Alexe, B., Deselaers, T., and Ferrari, V. (2010). What is an object? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7, 11, 116, 125

Ali, S. and Smith, K. A. (2006). On learning algorithm selection for classification. 6(2):119–138. 123

Ali, S. and Smith-Miles, K. A. (2006). A meta-learning approach to automatic kernel selection for support vector machines. 70(1–3):173–186. 123

Allen, B., Curless, B., and Popović, Z. (2003). The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Transactions on Graphics*, 22(3):587–594. 48

Andres, B., Kappes, J. H., Beier, T., Köthe, U., and Hamprecht, F. A. (2011). Probabilistic image segmentation with closedness constraints. In *Proceedings of the International Conference on Computer Vision*. 124

Andres, B., Kappes, J. H., Beier, T., Köthe, U., and Hamprecht, F. A. (2012a). The lazy flipper: Efficient depth-limited exhaustive search in discrete graphical models. In *Proceedings of the European Conference on Computer Vision*, pages 154–166. 118, 139

Andres, B., Kappes, J. H., Köthe, U., Schnörr, C., and Hamprecht, F. A. (2010). An empirical comparison of inference algorithms for graphical models with higher order factors using OpenGM. In *Proceedings of the Annual Symposium of the German Association for Pattern Recognition*, number 6376, pages 353–362. 113, 120

Andres, B., Kroeger, T., Briggman, K. L., Denk, W., Korogod, N., Knott, G., Koethe, U., and Hamprecht, F. A. (2012b). Globally optimal closed-surface segmentation for connectomics. In *Proceedings of the European Conference on Computer Vision*. 124

Barron, J. T. and Malik, J. (2015). Shape, illumination, and reflectance from shading. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(8):1670–1687. 39

Bay, H., Ess, A., Tuytelaars, T., and van Gool, L. (2008). SURF: Speeded up robust features. *Computer Vision and Image Understanding*. 38

Ben-Shabat, Y., Lindenbaum, M., and Fischer, A. (2017). 3D point cloud classification and segmentation using 3D modified fisher vector representation for convolutional neural networks. *arXiv preprint*, arXiv:1711.08241. 52

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, pages 2546–2554. 123

Bergtholdt, M., Kappes, J., Schmidt, S., and Schnörr, C. (2010). A study of parts-based object class detection using complete graphs. *International Journal of Computer Vision*, 87(1):93–117. 119, 128

Berman, M., Rannen Triki, A., and Blaschko, M. B. (2018). The Lovász-Softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in

neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 32

Besag, J. (1986). On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, 48(3):48–259. 115, 118, 128

Binford, T. O. (1971). Visual perception by computer. *IEEE Conf. on Systems and Control*. 53

Bishop, C. (2006). Pattern recognition and machine learning. *Springer*. 77, 78, 84, 112, 120, 122, 128

Blanz, V. and Vetter, T. (1999). A morphable model for the synthesis of 3D faces. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. 48

Blaschko, M. B. and Lampert, C. H. (2008). Learning to localize objects with structured output regression. In *Proceedings of the European Conference on Computer Vision*. 18

Blei, D., Ng, A., and Jordan, M. (2003). Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022. 83

Boykov, Y., Veksler, O., and Zabih, R. (2001a). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239. 38, 116

Boykov, Y., Veksler, O., and Zabih, R. (2001b). Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239. 112, 116, 118, 126, 127, 128, 132

Brandes, U., Delling, D., Gaertler, M., Görke, R., Hoefer, M., Nikoloski, Z., and Wagner, D. (2008). On modularity clustering. 2(2):172–188. 124

Brazdil, P. B. and Henery, R. J. (1994). Analysis of results. In *Machine Learning, Neural and Statistical Classification*, pages 175–212. Ellis Horwood. 123

Bromley, J., Guyon, I., LeCun, Y., Säckinger, E., and Shah, R. (1993). Signature verification using a ”siamese” time delay neural network. In *Advances in Neural Information Processing Systems*. 41

Burt, P. J. and Adelson, E. H. (1983). The laplacian pyramid as a compact image code. *IEEE Trans. on Communications*, COM-31(4):532–540. 55

Caesar, H., Uijlings, J., and Ferrari, V. (2016). Region-based semantic segmentation with end-to-end training. In *Proceedings of the European Conference on Computer Vision*. 32

Caesar, H., Uijlings, J., and Ferrari, V. (2018). COCO-Stuff: Thing and stuff classes in context. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 32

Cao, Z., Qin, T., Liu, T.-Y., Tsai, M.-F., and Li, H. (2007). Learning to rank: From pairwise approach to listwise approach. In *International Conference on Machine Learning*. 17

Chang, A. X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Su, H., Xiao, J., Yi, L., and Yu, F. (2015). ShapeNet: An Information-Rich 3D Model Repository. *arXiv preprint*, arXiv:1512:03012. 36, 40, 62

Choi, W., Chao, Y.-W., Pantofaru, C., and Savarese, S. (2013). Understanding indoor scenes using 3D geometric phrases. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 81

Choy, C. B., Xu, D., Gwak, J., Chen, K., and Savarese, S. (2016). 3D-R2N2: A unified approach for single and multi-view 3D object reconstruction. In *Proceedings of the European Conference on Computer Vision*. 36, 40, 42, 43, 51

Conejo, B., Komodakis, N., Leprince, S., and Avouac, J.-P. (2014). Inference by learning: Speeding-up graphical model optimization via a coarse-to-fine cascade of pruning classifiers. In *Advances in Neural Information Processing Systems*, pages 1–9. 121

Cordonnier, G., Galin, E., Gain, J., Benes, B., Guérin, E., Peytavie, A., and Cani, M.-P. (2017). Authoring landscapes by combining ecosystem and terrain erosion simulation. *ACM Transactions on Graphics*, 36(4). 78

Criminisi, A., Shotton, J., and Konukoglu, E. (2011). Decision forests for classification, regression, density estimation, manifold learning and semi-supervised learning. *Microsoft Research Cambridge, Tech. Rep. MSRTR-2011-114*. 132

Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. 39

Dalal, N. and Triggs, B. (2005). Histogram of Oriented Gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10

Del Pero, L., Bowdish, J., Fried, D., Kermgard, B., Hartley, E., and Barnard, K. (2012). Bayesian geometric modeling of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 78, 81

Del Pero, L., Bowdish, J., Fried, D., Kermgard, B., Hartley, E., and Barnard, K. (2013). Understanding bayesian rooms using composite 3D object models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 81

Deselaers, T., Alexe, B., and Ferrari, V. (2010). Localizing objects while learning their appearance. In *Proceedings of the European Conference on Computer Vision*. 116

DiCarlo, J. J., Zoccolan, D., and Rust, N. C. (2012). How does the brain solve visual object recognition? *Neuron*, 73(3):415–434. 1

Domke, J. (2012). Generic methods for optimization-based modeling. 145

Dong, C., Loy, C. C., He, K., and Tang, X. (2014). Learning a deep convolutional network for image super-resolution. In *Proceedings of the European Conference on Computer Vision*. 11

Doppa, J. R., Kumar, P., Wick, M., Singh, S., and Salakhutdinov, R. (2013). ICML 2013 workshop on inferning. `http://inferning.cs.umass.edu/`. 121

Efron, B. and Tibshirani, R. (1986). Bootstrap methods for standard errors, confidence intervals, and other measures of statistical accuracy. *Statistical science*, pages 54–75. 102, 103

Emilien, A., Vimont, U., Cani, M.-P., Poulin, P., and Benes, B. (2015). WorldBrush: Interactive example-based synthesis of procedural virtual worlds. *ACM Transactions on Graphics*, 34(4). 78

Engelmann, F., Stückler, J., and Leibe, B. (2017). SAMP: shape and motion priors for 4d vehicle reconstruction. In *Proceedings of the IEEE Workshop on Applications of Computer Vision*. 47

Ens, J. and Lawrence, P. (1993). An investigation of methods for determining depth from focus. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(2):97–108. 39

Everingham, M., Eslami, S., van Gool, L., Williams, C., Winn, J., and Zisserman, A. (2015). The PASCAL visual object classes challenge: A retrospective. *International Journal of Computer Vision*. 9, 13, 15, 26, 31, 41

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2007). The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results. http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html. 125

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010). The PASCAL Visual Object Classes (VOC) Challenge. *International Journal of Computer Vision*. 8, 15

Fan, H., Su, H., and Guibas, L. (2017). A point set generation network for 3D object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 37, 40, 43, 52, 62, 67, 69, 74

Felzenszwalb, P., Girshick, R., McAllester, D., and Ramanan, D. (2010). Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(9). 10, 12

Felzenszwalb, P. F. and Huttenlocher, D. P. (2004). Efficient belief propagation for early vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 261–268. 38, 116

Fischler, M. A. and Bolles, R. C. (1981). Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Comm. Assoc. Comp. Mach.*, 24(6):381–395. 125

Fisher, M., Ritchie, D., Savva, M., Funkhouser, T., and Hanrahan, P. (2012). Example-based synthesis of 3d object arrangements. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. 80, 84

Fisher, M., Savva, M., Li, Y., Hanrahan, P., and Nießner, M. (2015). Activity-centric scene synthesis for functional 3D scene modeling. *ACM Transactions on Graphics*, 34(6):179:1–179:13. 84

Fix, A., Gruber, A., Boros, E., and Zabih, R. (2011). A graph cut algorithm for higher-order markov random fields. In *Proceedings of the International Conference on Computer Vision*. 117

Floros, G. and Leibe, B. (2012). Joint 2D-3D temporally consistent semantic segmentation of street scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 116

Frahm, J.-M., Fite-Georgel, P., Gallup, D., Johnson, T., Raguram, R., Wu, C., Jen, Y.-H., Dunn, E., Clipp, B., Lazebnik, S., and Pollefeys, M. (2010). Building Rome on a cloudless day. In *Proceedings of the European Conference on Computer Vision*, pages 368–381. 38

Freund, Y. and Schapire, R. (1997). A decision theoretic generalisation of online learning. *Computer and System Sciences*, 55(1):119–139. 10, 17

Fu, Q., Chen, X., Wang, X., Wen, S., Zhou, B., and Fu, H. (2017). Adaptive synthesis of indoor scenes via activity-associated object relation graphs. *ACM Transactions on Graphics*, 36(6). 82, 83

Funkhouser, T., Kazhdan, M., Shilane, P., Min, P., Kiefer, W., Tal, A., Rusinkiewicz, S., and Dobkin, D. (2004). Modeling by example. *ACM Transactions on Graphics*, 23(3):652–663. 78

Furukawa, Y. and Hernández, C. (2015). Multi-view stereo: A tutorial. *Foundations and trends® in Computer Graphics and Vision*, 9(1–2):1–148. 38, 39, 41

Furukawa, Y. and Ponce, J. (2010). Accurate, dense, and robust multi-view stereopsis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1362–1376. 39

Gadelha, M., Maji, S., and Wang, R. (2017). 3D shape induction from 2D views of multiple objects. In *3DV*. xi, 34, 35, 51, 54, 58, 62, 63, 64, 65, 71, 74

Gallup, D., Frahm, J.-M., Mordohai, P., Yang, Q., and Pollefeys, M. (2007). Real-time plane-sweeping stereo with multiple sweeping directions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 39

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 144

Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741. 115

Genova, K., Cole, F., Maschinot, A., Sarna, A., Vlasic, D., and Freeman, W. T. (2018). Unsupervised training for 3D morphable model regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 46, 70

Girdhar, R., Fouhey, D., Rodriguez, M., and Gupta, A. (2016). Learning a predictable and generative vector representation for objects. In *Proceedings of the European Conference on Computer Vision*. 42

Girshick, R. (2015). Fast R-CNN. In *Proceedings of the International Conference on Computer Vision*. 3, 7, 8, 9, 12, 13, 14, 18, 20, 25, 26, 27, 28, 31

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7, 11, 18

Globerson, A. and Jaakkola, T. (2007). Fixing max-product: Convergent message passing algorithms for MAP LP-relaxations. In *Advances in Neural Information Processing Systems*. 119, 120

Goesele, M., Snavely, N., Curless, B., Hoppe, H., and Seitz, S. M. (2007). Multi-view stereo for community photo collections. In *Proceedings of the International Conference on Computer Vision*. 39

Goodfellow, I., Pouget-Abadle, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in Neural Information Processing Systems*. 49, 54

Gould, S., Fulton, R., and Koller, D. (2009). Decomposing a scene into geometric and semantically consistent regions. In *Proceedings of the International Conference on Computer Vision*. 124

Gouraud, H. (1971). Continuous shading of curved surfaces. *IEEE Trans. on Computers*, C-20(6):623–629. 54

Greig, D. M., Porteous, B. T., and Seheult, A. H. (1989). Exact maximum a posteriori estimation for binary images. *Journal of the Royal Statistical Society*, 51(2):271–279. 112, 117

Guignard, M. and Kim, S. (1987). Lagrangean decomposition: a model yielding stronger Lagrangean bounds. *Mathematical Programming*, pages 215–228. 119

Guillaumin, M., Van Gool, L., and Ferrari, V. (2013). Fast energy minimization using learned state filters. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 116, 121, 124, 125

Gwak, J., Choy, C. B., Chandraker, M., Garg, A., and Savarese, S. (2017). Weakly supervised 3D reconstruction with adversarial constraint. In *3DV*. 44

Habbecke, M. and Kobbelt, L. (2007). A surface-growing approach to multi-view stereo reconstruction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 39

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. (2009). The WEKA data mining software: an update. 11(1):10–18. 123

Handa, A., Patraucean, V., Stent, S., and Cipolla, R. (2016). SceneNet: An annotated model generator for indoor scene understanding. In *Proceedings of IEEE International Conference on Robotics and Automation*. 82, 83, 104

Häne, C., Tulsiani, S., and Malik, J. (2017). Hierarchical surface prediction for 3D object reconstruction. In *3DV*. 52

Harris, C. G. and Stephens, M. (1988). A combined corner and edge detector. In *Proceedings of the 4th Alvey Vision Conference, Manchester*, pages 147–151. 10, 38

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on SSC*, 4:100–107. 120

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition. 38, 39

Hartmann, W., Galliani, S., Havlena, M., Van Gool, L., and Schindler, K. (2017). Learned multi-patch similarity. In *Proceedings of the International Conference on Computer Vision*. 40

Harzallah, H., Jurie, F., and Schmid, C. (2009). Combining efficient object localization and image classification. In *Proceedings of the International Conference on Computer Vision*. 10

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask R-CNN. In *Proceedings of the International Conference on Computer Vision*, pages 2980–2988. 31

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 12, 31

Henderson, P. and Ferrari, V. (2016a). Automatically selecting inference algorithms for discrete energy minimisation. In *Proceedings of the European Conference on Computer Vision*, pages 235–252. 4, 114

Henderson, P. and Ferrari, V. (2016b). End-to-end training of object class detectors for mean average precision. In *Proceedings of the Asian Conference on Computer Vision*, pages 198–213. 3, 9

Henderson, P. and Ferrari, V. (2018). Learning to generate and reconstruct 3D meshes with only 2D supervision. In *Proceedings of the British Machine Vision Conference*. 3, 35

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. (2017). GANs trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*. 37

Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). β-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*. 58

Hoiem, D., Efros, A. A., and Hebert, M. (2011). Recovering occlusion boundaries from an image. *International Journal of Computer Vision*, 91(3):328–346. 124

Horn, B. (1975). Obtaining shape from shading information. In Winston, P. H., editor, *The Psychology of Computer Vision*. 34, 39

Hosang, J., Benenson, R., Dollár, P., and Schiele, B. (2016a). What makes for effective detection proposals? 38(4):814–830. 11, 12

Hosang, J., Benenson, R., and Schiele, B. (2016b). A convnet for non-maximum suppression. In *German Conference on Pattern Recognition*. 19

Hosang, J., Benenson, R., and Schiele, B. (2017). Learning non-maximum suppression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 19

Huang, H., Kalogerakis, E., and Marlin, B. (2015). Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces. *Computer Graphics Forum*, 34(5):25–38. 47, 48, 50, 52

Huang, P.-H., Matzen, K., Kopf, J., Ahuja, N., and Huang, J.-B. (2018). DeepMVS: Learning multi-view stereopsis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 41

Hueting, M., Reddy, P., Yumer, E., Kim, V. G., Carr, N., and Mitra, N. J. (2017). SeeThrough: Finding chairs in heavily occluded indoor scene images. *arXiv preprint*, arXiv:1711.08241. 81

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*, pages 507–523. 123

Isack, H. and Boykov, Y. (2012). Energy-based geometric multi-model fitting. *International Journal of Computer Vision*, 97(2):123–147. 125

Ishikawa, H. (2009). Higher-order clique reduction in binary graph cut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2993–3000. 117

Ishikawa, H. (2011). Transformation of general binary MRF minimization to the first order case. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1234–1249. 117

Jaimovich, A., Elidan, G., Margalit, H., and Friedman, N. (2006). Towards an integrated protein-protein interaction network: a relational markov network approach. *Journal of Computational Biology*, 13(2):145–164. 124

Jakob, W. (2010). Mitsuba renderer. `http://www.mitsuba-renderer.org`. 94, 101

Järvelin, K. and Kekäläinen, J. (2000). IR evaluation methods for retrieving highly relevant documents. In *Special Interest Group on Information Retrieval*. 9, 30

Jégou, H., Douze, M., Schmid, C., and Pérez, P. (2010). Aggregating local descriptors into a compact image representation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11

Ji, M., Gall, J., Zheng, H., Liu, Y., and Fang, L. (2017). SurfaceNet: An end-to-end 3D neural network for multiview stereopsis. In *Proceedings of the International Conference on Computer Vision*. 40

Jia, Y. (2013). Caffe: An open source convolutional architecture for fast feature embedding. `http://caffe.berkeleyvision.org/`. 32

Jiang, C., Qi, S., Zhu, Y., Huang, S., Lin, J., Yu, L.-F., Terzopoulos, D., and Zhu, S.-C. (2018). Configurable 3D scene synthesis and 2D image rendering with per-pixel ground truth using stochastic grammars. *International Journal of Computer Vision*, 126(9):920–941. 85

Jiang, J., Moon, T., Daumé III, H., and Eisner, J. (2013). Prioritized asynchronous belief propagation. In *ICML Workshop on Inferning*. 121

Jordan, M., Ghahramani, Z., Jaakkola, T., and Saul, L. (1999). An introduction to variational methods for graphical models. *Machine Learning*, 37:183–233. 49

Kalogerakis, E., Chaudhuri, S., Koller, D., and Koltun, V. (2012). A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics*, 31(4):55:1–55:11. 48

Kappes, J., Andres, B., Hamprecht, F., Schnörr, C., Nowozin, S., Batra, D., Kim, S., Kausler, B., Kröger, T., Lellmann, J., Komodakis, N., Savchynskyy, B., and Rother, C. (2015). A comparative study of modern inference techniques for structured discrete energy minimization problems. *International Journal of Computer Vision*, pages 1–30. 112, 113, 115, 117, 118, 119, 120, 124, 127, 128, 138

Kappes, J., Savchynskyy, B., and Schnörr, C. (2012). A bundle approach to efficient MAP-inference by lagrangian relaxation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 112, 119, 128

Kar, A., Häne, C., and Malik, J. (2017). Learning a multi-view stereo machine. In *Advances in Neural Information Processing Systems*. 40

Kar, A., Tulsiani, S., Carreira, J., and Malik, J. (2015). Category-specific object reconstruction from a single image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 42, 47

Kato, H., Ushiku, Y., and Harada, T. (2018). Neural 3D mesh renderer. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 46, 52

Kernighan, B. W. and Lin, S. (1970). An efficient heuristic procedure for partitioning graphs. *Bell Systems Technical Journal*, 49(2):291–307. 128

Kersten, D. and Yuille, A. (2014). Inferential models of the visual cortical hierarchy. In *The Cognitive Neurosciences*, chapter 34. MIT Press, 5th edition. 1

Kim, S., Nowozin, S., Kohli, P., and Yoo, C. D. (2011). Higher-order correlation clustering for image segmentation. In *Advances in Neural Information Processing Systems*. 124

Kingma, D. P. and Ba, J. L. (2015). Adam: A method for stochastic optimization. In *International Conference on Learning Representations*. 58

Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *International Conference on Learning Representations*. 35, 43, 49, 54, 56, 58, 70, 144

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *Science*, 220:671–680. 84

Kohli, P., Kumar, M., and Torr, P. (2007). P3 & beyond: Solving energies with higher order cliques. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 116, 126

Kohli, P., Ladicky, L., and Torr, P. (2008). Robust higher order potentials for enforcing label consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 116, 126

Kohli, P., Ladicky, L., and Torr, P. (2009). Robust higher order potentials for enforcing label consistency. *International Journal of Computer Vision*. 116

Kohli, P. and Rother, C. (2012). Higher-order models in computer vision. In *Image Processing and Analysing Graphs: Theory and Practice*. CRC Press. 115

Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(10):1568 – 1583. 112, 117, 119, 120, 126, 128

Kolmogorov, V. (2015). A new look at reweighted message passing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(5):919–930. 119, 120, 138

Kolmogorov, V. and Rother, C. (2006). Comparison of energy minimization algorithms for highly connected graphs. In *Proceedings of the European Conference on Computer Vision*, pages 1–15. 113, 120

Komodakis, N., Paragios, N., and Tziritas, G. (2007). Mrf optimization via dual decomposition: Message-passing revisited. In *Proceedings of the International Conference on Computer Vision*, pages 1–8. 119

Komodakis, N., Tziritas, G., and Paragios, N. (2008). Performance vs computational efficiency for optimizing single and dynamic MRFs: Setting the state of the art with primal-dual strategies. *Computer Vision and Image Understanding*, 112(1):14–29. 119, 128

Köpf, C., Taylor, C., and Keller, J. (2000). Meta-analysis: From data characterisation for meta-learning to meta-regression. In *Proceedings of the PKDD-00 Workshop on Data Mining, Decision Support,Meta-Learning and ILP*. 123

Kotthoff, L. (2016). Algorithm selection for combinatorial search problems: A survey. In *Data Mining and Constraint Programming: Foundations of a Cross-Disciplinary Approach*, pages 149–190. Springer International Publishing. 122

Kotthoff, L., Kerschke, P., Hoos, H. H., and Trautmann, H. (2015). Improving the state of the art in inexact TSP solving using per-instance algorithm selection. In *LION*. 122

Krähenbühl, P. and Koltun, V. (2011). Efficient inference in fully connected CRFs with gaussian edge potentials. In *Advances in Neural Information Processing Systems*, pages 109–117. 116

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 7, 11, 20, 27

Kschischang, F. R. and Frey, B. J. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519. 120

Lambert, J. H. (1760). *Photometria*. Eberhard Klett Verlag. 54, 59

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*. 11

Lellmann, J. and Schnörr, C. (2011). Continuous multiclass labeling approaches and algorithms. *SIAM Journal on Imaging Sciences*, 4(4):1049–1096. 124

Lempitsky, V. and Boykov, Y. (2007). Global optimization for shape fitting. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 39

Lempitsky, V. S., Rother, C., Roth, S., and Blake, A. (2010). Fusion moves for markov random field optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(8):1392–1405. 118

Levine, S., Finn, C., Darrell, T., and Abbeel, P. (2016). End-to-end training of deep visuomotor policies. *Journal of Machine Learning Research*, 17:1–40. 8

Li, J., Xu, K., Chaudhuri, S., Yumer, E., Zhang, H., and Guibas, L. (2017). GRASS: Generative recursive autoencoders for shape structures. *ACM Transactions on Graphics*, 36(4). 37, 44, 50, 51, 53, 78, 109

Liang, Y., Zhang, S.-H., and Martin, R. R. (2017). Automatic data-driven room design generation. In *Next Generation Computer Animation Techniques*. Springer. 82, 83, 84

Lim, J. J., Pirsiavash, H., and Torralba, A. (2013). Parsing IKEA Objects: Fine Pose Estimation. In *Proceedings of the International Conference on Computer Vision*. 36

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. (2014). Microsoft COCO: Common objects in context. In *Proceedings of the European Conference on Computer Vision*. 13, 15, 31

Lindauer, M., Hoos, H., Hutter, F., and Schaub, T. (2015). AutoFolio: An automatically configured algorithm selector. *Journal of Artificial Intelligence Research*, 53:745–778. 122

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2016). SSD: Single shot multibox detector. In *Proceedings of the European Conference on Computer Vision*. 3, 12, 31, 143

Loh, A. M. and Hartley, R. (2005). Shape from non-homogeneous, non-stationary, anisotropic, perspective texture. In *Proceedings of the British Machine Vision Conference*. 39

Long, J., Shelhamer, E., and Darrell, T. (2015). Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8, 11, 32

Loper, M. M. and Black, M. J. (2014). OpenDR: An approximate differentiable renderer. In *Proceedings of the European Conference on Computer Vision*, pages 154–169. 59, 60

Lovász, L. (1983). Submodular functions and convexity. In *Mathematical Programming: The State of the Art – Bonn 1982*, pages 235–257. Springer Berlin Heidelberg. 117

Lowe, D. (2004a). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110. 10

Lowe, D. (2004b). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110. 38

Lowe, D. G. (1987). Three-dimensional object recognition from single two-dimensional images. *Artificial Intelligence*, 31(3):355–395. 10

Ma, R., Li, H., Zou, C., Liao, Z., Tong, X., and Zhang, H. (2016). Action-driven 3D indoor scene evolution. *ACM Transactions on Graphics*, 35(6):173:1–173:13. 84

Maclaurin, D., Duvenaud, D., and Adams, R. P. (2015). Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*. 145

Mandikal, P., Murthy, N., Agarwal, M., and Babu, R. V. (2018). 3D-LMNet: Latent embedding matching for accurate and diverse 3D point cloud reconstruction from a single image. In *Proceedings of the British Machine Vision Conference*. 43

Marr, D. and Poggio, T. (1979). A computational theory of human stereo vision. *Philosophical Transactions of the Royal Society of London, Series A*, 204:301–328. 38

Martins, A. F. T., Figueiredo, M. A. T., Aguiar, P. M. Q., Smith, N. A., and Xing, E. P. (2015). AD3: Alternating directions dual decomposition for MAP inference in graphical models. *Journal of Machine Learning Research*, 16:495–545. 112, 119, 120, 128

Máttyus, G., Luo, W., and Urtasun, R. (2017). DeepRoadMapper: Extracting road topology from aerial images. In *Proceedings of the International Conference on Computer Vision*. 32

Merrell, P., Schkufza, E., and Koltun, V. (2010). Computer-generated residential building layouts. *ACM Transactions on Graphics*, 29(6):181:1–181:12. 79, 80

Merrell, P., Schkufza, E., Li, Z., Agrawala, M., and Koltun, V. (2011). Interactive furniture layout using interior design guidelines. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. 78, 82, 83, 84

Metropolis, N., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computer machines. *Journal of Chemical Physics*, 21:1087–1092. 79

Mikolajczyk, K. and Schmid, C. (2001). Indexing based on scale invariant interest points. In *Proceedings of the 8th International Conference on Computer Vision, Vancouver, Canada*. 10

Mnih, A. and Rezende, D. J. (2016). Variational inference for monte carlo objectives. In *International Conference on Machine Learning*. 45, 144

Mohapatra, P. and Jawahar, C. V. (2014). Efficient optimization for average precision svms. In *Advances in Neural Information Processing Systems*. 18

Mousavian, A., Anguelov, D., Flynn, J., and Kosecka, J. (2017). 3D bounding box estimation using deep learning and geometry. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 57

Müller, P., Wonka, P., Haegler, S., Ulmer, A., and Van Gool, L. (2006). Procedural modeling of buildings. *ACM Transactions on Graphics*, 25(3):614–623. 78, 80

Nair, V., Susskind, J., and Hinton, G. E. (2008). Analysis-by-synthesis by learning to invert generative black boxes. In *International Conference on Artificial Neural Networks*, pages 971–981. 70

Nash, C. and Williams, C. K. I. (2017). The shape variational autoencoder: A deep generative model of part-segmented 3D objects. *Computer Graphics Forum*, 36(5):1–12. 50, 52

Nishida, G., Garcia-Dorado, I., Aliaga, D. G., Benes, B., and Bousseau, A. (2016). Interactive sketching of urban procedural models. *ACM Transactions on Graphics*, 35(4). 78

Niu, C., Li, J., and Xu, K. (2018). Im2Struct: Recovering 3D shape structure from a single RGB image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 44, 53

Novotny, D., Larlus, D., and Vedaldi, A. (2017). Learning 3d object categories by looking around them. In *Proceedings of the International Conference on Computer Vision*, pages 5218–5227. 46

Nowozin, S., Rother, C., Bagon, S., Sharp, T., Yao, B., and Kohli, P. (2011). Decision tree fields. In *Proceedings of the International Conference on Computer Vision*. 124

Otten, L. and Dechter, R. (2011). Anytime AND/OR depth-first search for combinatorial optimization. In *Proceedings of the Annual Symposium on Combinatorial Search*. 120

Parikh, D. and Zitnick, C. L. (2011). Human-debugging of machines. In *Workshop at Advances in Neural Information Processing Systems*. 12

Parish, Y. I. H. and Müller, P. (2001). Procedural modeling of cities. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*, pages 301–308. ACM. 78

Paschalidou, D., Ulusoy, A. O., Schmitt, C., Van Gool, L., and Geiger, A. (2018). RayNet: Learning volumetric 3D reconstruction with ray potentials. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 41

Perronnin, F., Sánchez, J., and Mensink, T. (2010). Improving the fisher kernel for large-scale image classification. In *Proceedings of the European Conference on Computer Vision*, pages 143–156. 11

Pfister, T., Charles, J., and Zisserman, A. (2015). Flowing convnets for human pose estimation in videos. In *Proceedings of the International Conference on Computer Vision*. 8

Pohlen, T., Hermans, A., Mathias, M., and Leibe, B. (2017). Full-resolution residual networks for semantic segmentation in street scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 32

Pollefeys, M., Koch, R., and Van Gool, L. (1998). Self calibration and metric reconstruction in spite of varying and unknown internal camera parameters. In *Proceedings of the 6th International Conference on Computer Vision, Bombay, India*, pages 90–96. 38

Prados, E. and Faugeras, O. (2006). Shape from shading. In *Handbook of Mathematical Models in Computer Vision*, pages 375–388. Springer. 39

Prest, A., Leistner, C., Civera, J., Schmid, C., and Ferrari, V. (2012). Learning object class detectors from weakly annotated video. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 10

Prisacariu, V. A., , and Reid, I. (2011). Shared shape spaces. In *Proceedings of the International Conference on Computer Vision*. 41

Prisacariu, V. A., Segal, A. V., and Reid, I. (2012). Simultaneous monocular 2D segmentation, 3D pose recovery and 3D reconstruction. In *Proceedings of the Asian Conference on Computer Vision*. 41, 48

Prusinkiewicz, P. and Lindenmayer, A. (2004). *The Algorithmic Beauty of Plants*. Springer-Verlag. 80

Pulina, L. and Tacchella, A. (2009). A self-adaptive multi-engine solver for quantified Boolean formulas. 14:80–116. 122

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). PointNet: Deep learning on point sets for 3D classification and segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 43, 50, 52

Qi, S., Zhu, Y., Huang, S., Jiang, C., and Zhu, S.-C. (2018). Human-centric indoor scene synthesis using stochastic grammar. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 82, 83, 84, 85, 105

Radford, A., Metz, L., and Chintala, S. (2016). Unsupervised representation learning with deep convolutional generative adversarial networks. In *International Conference on Learning Representations*. 50

Rahman, M. A. and Wang, Y. (2016). Optimizing intersection-over-union in deep neural networks for image segmentation. In *International Symposium on Visual Computing*. Springer. 32

Rao, Y., Lin, D., Lu, J., and Zhou, J. (2018). Learning globally optimized object detector via policy gradient. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 19, 30

Rasmussen, C. E. (2000). The infinite gaussian mixture model. In *Advances in Neural Information Processing Systems*. 92

Rasmussen, C. E. and Williams, C. K. I. (2005). *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press. 123

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 12, 31

Redmon, J. and Farhadi, A. (2017). YOLO9000: better, faster, stronger. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 12, 31

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*. 12, 31, 143

Restrepo, M. I., Ulusoy, A. O., and Mundy, J. L. (2014). Evaluation of feature-based 3-D registration of probabilistic volumetric scenes. *ISPRS Journal of Photogrammetry and Remote Sensing*, 98:1–18. 41

Rezende, D. J., Eslami, S. M. A., Mohamed, S., Battaglia, P., Jaderberg, M., and Heess, N. (2016). Unsupervised learning of 3D structure from images. In *Advances in Neural Information Processing Systems*. 45, 50

Rezende, D. J., Mohamed, S., and Wierstra, D. (2014). Stochastic backpropagation and approximate inference in deep generative models. In *International Conference on Machine Learning*. 35, 49, 56, 58

Rice, J. R. (1976). The algorithm selection problem. *Advances in Computers*, 15:65–118. 121

Richter, S. R. and Roth, S. (2018). Matryoshka networks: Predicting 3D geometry via nested shape layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1936–1944. 43, 52

Roberts, L. G. (1965). Machine perception of three-dimensional solids. *Optical and electro-optical information processing, J. Tippet et al., Ed.* 53

Roig, G., Boix, X., De Nijs, R., Ramos, S., Kuhnlenz, K., and Van Gool, L. (2013). Active map inference in crfs for efficient semantic segmentation. In *Proceedings of the International Conference on Computer Vision*, pages 2312–2319. 121

Romaszko, L., Williams, C. K. I., Moreno, P., and Kohli, P. (2017). Vision-as-inverse-graphics: Obtaining a rich 3D explanation of a scene from a single image. In *Proceedings of the International Conference on Computer Vision Workshops*, pages 940–948. 70

Rother, C., Kohli, P., Feng, W., and Jia, J. (2009). Minimizing sparse higher order energy functions of discrete variables. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 125

Rother, C., Kolmogorov, V., Lempitsky, V., and Szummer, M. (2007). Optimizing binary MRFs via extended roof duality. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 112, 117, 126, 128

Rothwell, C., Zisserman, A., Mundy, J., and Forsyth, D. (1992). Efficient model library access by projectively invariant indexing functions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 109–114. 10

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A., and Fei-Fei, L. (2015). ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*. 11, 13, 15, 20, 27

Sadeghipour Kermani, Z., Liao, Z., Tan, P., and Zhang, H. (2016). Learning 3D scene synthesis from annotated RGB-D images. *Computer Graphics Forum*, 35(5):197–206. 82, 83

Salimans, T., Goodfellow, I., Zaremba, W., Cheung, V., Radford, A., Chen, X., and Chen, X. (2016). Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*. 37

Salton, G. and McGill, M. J. (1986). *Introduction to Modern Information Retrieval*. McGraw-Hill. 15

Savchynskyy, B. and Schmidt, S. (2013). Getting feasible variable estimates from infeasible ones: MRF local polytope study. In *Proceedings of the International Conference on Computer Vision Workshops*. 119

Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision*, 47(1):7–42. 38

Schoenemann, T. and Kolmogorov, V. (2014). Generalized sequential tree-reweighted message passing. In *Advanced Structured Prediction*, pages 75–102. MIT Press. 119, 138

Schöps, T., Schönberger, J. L., Galliani, S., Sattler, T., Schindler, K., Pollefeys, M., and Geiger, A. (2017). A multi-view stereo benchmark with high-resolution images and multi-camera videos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 41

Segal, M. and Akeley, K. (2018). *The OpenGL® Graphics System: A specification*. Khronos Group. 59

Seitz, S., Curless, B., Diebel, J., Scharstein, D., and Szeliski, R. (2006). A comparison and evaluation of multi-view stereo reconstruction algorithms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 38

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., and LeCun, Y. (2014). Overfeat: Integrated recognition, localization and detection using convolutional networks. In *International Conference on Learning Representations*. 7, 11

Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2006). *TextonBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation. In *Proceedings of the European Conference on Computer Vision*, pages 1–15. 116

Shotton, J., Winn, J., Rother, C., and Criminisi, A. (2009). TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling appearance, shape and context. *International Journal of Computer Vision*, 81(1):2–23. 124

Simonyan, K. and Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*. 7, 20, 27, 28

Sivic, J., Russell, B., Efros, A., Zisserman, A., and Freeman, W. (2005). Discovering object categories in image collections. Technical Report A. I. Memo 2005-005, Massachusetts Institute of Technology. 10

Smith-Miles, K. A. (2009). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys*, 41(1):6:1–6:25. 123

Snoek, J., Larochelle, H., and Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959. 123

Sochor, J., Juránek, R., Špaňhel, J., Maršík, L., Široký, A., Herout, A., and Zemčík, P. (2018). Comprehensive data set for automatic single camera visual speed measurement. *IEEE Transactions on Intelligent Transportation Systems*, pages 1–11. 72, 73

Soltani, A. A., Huang, H., Wu, J., Kulkarni, T. D., and Tenenbaum, J. B. (2017). Synthesizing 3d shapes via modeling multi-view depth maps and silhouettes with deep generative networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 37, 43, 50, 52

Song, S., Lichtenberg, S., and Xiao, J. (2015). SUN RGB-D: A RGB-D scene understanding benchmark suite. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 83

Song, S., Yu, F., Zeng, A., Chang, A. X., Savva, M., and Funkhouser, T. (2017). Semantic scene completion from a single depth image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 47, 75, 77, 85, 86, 90, 102

Song, Y., Schwing, A. G., Zemel, R. S., and Urtasun, R. (2016). Training deep neural networks via direct loss minimization. In *International Conference on Machine Learning*, pages 2169–2177. xiii, 9, 18, 26, 28, 29, 30

Sontag, D., Choe, D. K., and Li, Y. (2012). Efficiently searching for frustrated cycles in MAP inference. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 795–804. 119, 128

Sontag, D., Meltzer, T., Globerson, A., Weiss, Y., and Jaakkola, T. (2008). Tightening LP relaxations for MAP using message-passing. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 503–510. 119, 128

Stewart, R., Andriluka, M., and Ng, A. Y. (2016). End-to-end people detection in crowded scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 19

Stoyanov, V. and Eisner, J. (2012). Fast and accurate prediction via evidence-specific MRF structure. In *ICML Workshop on Inferning*. 121

Su, H., Qi, C. R., Li, Y., and Guibas, L. J. (2015). Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views. In *Proceedings of the International Conference on Computer Vision*, pages 2686–2694. 144

Sun, X., Wu, J., Zhang, X., Zhang, Z., Zhang, C., Xue, T., Tenenbaum, J. B., and Freeman, W. T. (2018). Pix3D: Dataset and methods for single-image 3D shape modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 36

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*. 8

Szeliski, R., Zabih, R., Scharstein, D., Veksler, O., Kolmogorov, V., Agarwala, A., Tappen, M., and Rother, C. (2008). A comparative study of energy minimization methods for markov random fields with smoothness-based priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(6):1068–1080. 38, 113, 116, 120, 124

Talton, J. O., Lou, Y., Lesser, S., Duke, J., Měch, R., and Koltun, V. (2011). Metropolis procedural modeling. *ACM Transactions on Graphics*, 30(2):11:1–11:14. 80

Tan, Q., Gao, L., and Yu-Kun Lai, S. X. (2018). Variational autoencoders for deforming 3d mesh models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 51

Tatarchenko, M., Dosovitskiy, A., and Brox, T. (2017). Octree generating networks: Efficient convolutional architectures for high-resolution 3D outputs. In *Proceedings of the International Conference on Computer Vision*. 52

Taylor, M., Guiver, J., Robertson, S., and Minka, T. (2008). SoftRank: Optimising non-smooth rank metrics. In *Proceedings of the ACM International Conference on Web Search and Data Mining*. 17

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of KDD-2013*. 123

Toshev, A. and Szegedy, C. (2014). Deeppose: Human pose estimation via deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660. 11

Triggs, W., McLauchlan, P., Hartley, R., and Fitzgibbon, A. (2000). Bundle adjustment: A modern synthesis. In Triggs, W., Zisserman, A., and Szeliski, R., editors, *Vision Algorithms: Theory and Practice*, LNCS, pages 298–375. Springer Verlag. 39

Tsochantaridis, I., Joachims, T., Hofmann, T., and Altun, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484. 17, 18

Tulsiani, S., Efros, A. A., and Malik, J. (2018). Multi-view consistency as supervisory signal for learning shape and pose prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 46, 71

Tulsiani, S. and Malik, J. (2015). Viewpoints and keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 66

Tulsiani, S., Su, H., Guibas, L. J., Efros, A. A., and Malik, J. (2017a). Learning shape abstractions by assembling volumetric primitives. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 43, 52, 53, 55

Tulsiani, S., Zhou, T., Efros, A. A., and Malik, J. (2017b). Multi-view supervision for single-view reconstruction via differentiable ray consistency. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. xiii, 34, 35, 37, 40, 45, 46, 51, 54, 56, 62, 63, 65, 67, 68, 69, 71, 74

Uijlings, J. R. R., van de Sande, K. E. A., Gevers, T., and Smeulders, A. W. M. (2013). Selective search for object recognition. *International Journal of Computer Vision*. 7, 10, 11

Venkatachalam, A. R. and Sohl, J. E. (1999). An intelligent model selection and forecasting system. 18(3):167–180. 123

Vicente, S., Carreira, J., Agapito, L., and Batista, J. (2014). Reconstructing PASCAL VOC. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 41, 47

Vilalta, R. and Drissi, Y. (2002). A perspective view and survey of meta-learning. 18(2):77–95. 123

Vinyals, O., Toshev, A., Bengio, S., and Erhan, D. (2015). Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 8, 11

Viola, P. and Jones, M. (2001). Robust real-time object detection. *International Journal of Computer Vision*. 10

Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2005). MAP estimation via agreement on (hyper)trees: Message-passing and linear-programming approaches. *IEEE Transactions on Information Theory*, 51(11):3697–3717. 120

Wan, L., Eigen, D., and Fergus, R. (2015). End-to-end integration of a convolution network, deformable parts model and non-maximum suppression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 18

Wang, K., Savva, M., Chang, A. X., and Ritchie, D. (2018). Deep convolutional priors for indoor scene synthesis. *ACM Transactions on Graphics*. 82, 85, 101, 103, 104, 107

Wang, X. (2016). Deep learning in object recognition, detection, and segmentation. 8(4):217–382. 10

Wang, Z., Simoncelli, E. P., and Bovik, A. C. (2003). Multiscale structural similarity for image quality assessment. In *The Thrity-Seventh Asilomar Conference on Signals, Systems Computers*, volume 2, pages 1398–1402. 73

Wiles, O. and Zisserman, A. (2017). SilNet: Single- and multi-view reconstruction by learning from silhouettes. In *Proceedings of the British Machine Vision Conference*. 34, 45, 54, 56, 57, 63, 64, 71

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256. 19, 45

Wonka, P., Wimmer, M., Sillion, F. X., and Ribarsky, W. (2003). Instant architecture. *ACM Transactions on Graphics*, 22(4):669–677. 80

Woodford, O. J., Torr, P. H. S., Reid, I. D., and Fitzgibbon, A. W. (2008). Global stereo reconstruction under second order smoothness priors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 116

Wu, J., Zhang, C., Xue, T., Freeman, W. T., and Tenenbaum, J. B. (2016). Learning a probabilistic latent space of object shapes via 3D generative-adversarial modeling. In *Advances in Neural Information Processing Systems*. 37, 43, 50, 54

Wu, Z., Li, F., Sukthankar, R., and Rehg, J. M. (2015a). Robust video segment proposals with painless occlusion handling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 37

Wu, Z., Song, S., Khosla, A., Yu, F., Zhang, L., Tang, X., and Xiao, J. (2015b). 3D ShapeNets: A deep representation for volumetric shape modeling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 36, 47, 48, 50

Xia, F., Liu, T.-Y., Wang, J., Zhang, W., and Li, H. (2008). Listwise approach to learning to rank: Theory and algorithm. In *International Conference on Machine Learning*. 17

Xiang, Y., Kim, W., Chen, W., Ji, J., Choy, C., Su, H., Mottaghi, R., Guibas, L., and Savarese, S. (2016). ObjectNet3D: A large scale database for 3D object recognition. In *Proceedings of the European Conference on Computer Vision*. 36

Xiang, Y., Mottaghi, R., and Savarese, S. (2014). Beyond PASCAL: A benchmark for 3D object detection in the wild. In *Proceedings of the IEEE Workshop on Applications of Computer Vision*. 35, 36, 144

Xie, J., Zheng, Z., Gao, R., Wang, W., Zhu, S.-C., and Wu, Y. N. (2018). Learning descriptor networks for 3d shape synthesis and analysis. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 48

Xu, J. and Li, H. (2007). AdaRank: A boosting algorithm for information retrieval. In *Special Interest Group on Information Retrieval*. 17

Xu, L., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2008). SATzilla: Portfolio-based algorithm selection for SAT. *Journal of Artificial Intelligence Research*, 32:565–606. 122

Yan, X., Yang, J., Yumer, E., Guo, Y., and Lee, H. (2016). Perspective transformer nets: Learning single-view 3D object reconstruction without 3D supervision. In *Advances in Neural Information Processing Systems*. xiii, 34, 35, 37, 44, 45, 46, 54, 56, 62, 63, 65, 67, 68, 69, 71

Yang, B., Wen, H., Wang, S., Clark, R., Markham, A., and Trigoni, N. (2017). 3D object reconstruction from a single depth view with adversarial learning. In *Proceedings of the International Conference on Computer Vision Workshops*. 47

Yanover, C., Schueler-Furman, O., and Weiss, Y. (2008). Minimizing and learning energy functions for side-chain prediction. *Journal of Computational Biology*, 15(7):899–911. 124

Yeh, Y.-T., Yang, L., Watson, M., Goodman, N. D., and Hanrahan, P. (2012). Synthesizing open worlds with constraints using locally annealed reversible jump MCMC. *ACM Transactions on Graphics*, 31(4):56:1–56:11. 78, 79

Yu, L.-F., Yeung, S.-K., Tang, C.-K., Terzopoulos, D., Chan, T. F., and Osher, S. J. (2011). Make it home: Automatic optimization of furniture arrangement. In *Proceedings of the ACM SIGGRAPH Conference on Computer Graphics*. 82, 83, 84

Yue, Y., Finley, T., Radlinski, F., and Joachims, T. (2007). A support vector method for optimizing average precision. In *Special Interest Group on Information Retrieval*. 9, 17, 18, 30

Zach, C., Pock, T., and Bischof, H. (2007). A globally optimal algorithm for robust TV-L1 range image integration. In *Proceedings of the International Conference on Computer Vision*. 39

Zhang, R., Tsai, P.-S., Cryer, J. E., and Shah, M. (1999). Shape-from-shading: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):690–706. 39

Zhang, X., Yang, Y.-H., Han, Z., Wang, H., and Gao, C. (2013). Object class detection: A survey. 46(1):10:1–10:53. 10

Zhang, Y. and Funkhouser, T. (2018). Deep depth completion of a single RGB-D image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 47

Zhang, Y., Sohn, K., Villegas, R., Pan, G., and Lee, H. (2015). Improving object detection with deep convolutional networks via Bayesian optimization and structured prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 7, 18, 30

Zhao, Y. and Zhu, S.-C. (2011). Image parsing with stochastic scene grammar. In *Advances in Neural Information Processing Systems*. 81

Zhao, Y. and Zhu, S.-C. (2013). Scene parsing by integrating function, geometry and appearance models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 78, 81

Zhu, R., Kiani Galoogahi, H., Wang, C., and Lucey, S. (2017). Rethinking reprojection: Closing the loop for pose-aware shape reconstruction from a single image. In *Proceedings of the International Conference on Computer Vision*. 44

Zitnick, C. L. and Dollár, P. (2014). Edge boxes: Locating object proposals from edges. In *Proceedings of the European Conference on Computer Vision*. 7, 11

Zoph, B. and Le, Q. V. (2017). Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*. 139

Zou, C., Yumer, E., Yang, J., Ceylan, D., and Hoiem, D. (2017). 3D-PRNN: Generating shape primitives with recurrent neural networks. In *Proceedings of the International Conference on Computer Vision*. 37, 47, 51, 55