Relating images and 3D models with convolutional neural networks

Francisco Vitor Suzano Massa

École Doctorale Paris-Est
Mathématiques & Sciences et Technologies
de l'Information et de la Communication

Thèse de doctorat
de l'Université Paris-Est
Domaine : Traitement du Signal et des Images

Présentée par

Francisco Vitor Suzano Massa

pour obtenir le grade de

Docteur de l'Université Paris-Est

Relating images and 3D models with convolutional neural networks

---

Soutenue publiquement le 9 février 2017 devant le jury composé de :

Mathieu Aubry	École des Ponts ParisTech	Co-directeur de thèse
Renaud Marlet	École des Ponts ParisTech	Directeur de thèse
Patrick Perez	Technicolor	Rapporteur
Florent Perronnin	Xerox	Rapporteur
Bryan Russell	Adobe	Examinateur
Josef Sivic	Inria	Examinateur
Laurent Najman	ESIEE	Examinateur
Hugues Talbot	ESIEE	Examinateur

École des Ponts ParisTech
LIGM-IMAGINE
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77455 Marne-la-Vallée cedex 2
France

Université Paris-Est Marne-la-Vallée
École Doctorale Paris-Est MSTIC
Département Études Doctorales
6, Av Blaise Pascal - Cité Descartes
Champs-sur-Marne
77454 Marne-la-Vallée cedex 2
France

Abstract

The recent availability of large catalogs of 3D models enables new possibilities for a 3D reasoning from photographs. This thesis investigates the use of convolutional neural networks (CNNs) for relating 3D objects to 2D images.

We first introduce two preliminary studies that are used throughout this thesis: an automatic memory reduction method for deep CNNs, and a study of CNN features for cross-domain matching. In the first one, we develop a library built on top of Torch7 which automatically reduces up to 91% of the memory requirements for deploying a deep CNN. In the second one, we study the effectiveness of various CNN features extracted from a pre-trained network for retrieving images from different modalities (real or synthetic images). We show that despite the large cross-domain difference between rendered views and photographs, it is possible to use CNN features for instance retrieval. We also present a multi-view extension and demonstrate an application to image-based rendering.

We then present a framework to perform 3D instance detection in images: given a 3D model (or a set of 3D models) and an image we locate and align the model in the image. We show that simply using CNN features is not enough for this task, and we propose to learn a transformation that takes the features from the real images close to the features from the rendered views. We evaluate our approach both qualitatively and quantitatively on two standard datasets: the IKEAobject dataset, and a subset of the Pascal VOC 2012 dataset of the "chair" category, and we show state-of-the-art results on both of them.

Finally, we move away from instances and attempt to extract 3D information for a full object category. There has been several recent uses of CNNs for the task of object viewpoint estimation, sometimes with very different design choices. We present these approaches in an unified framework and we analyse the key factors that affect performance. We propose a joint training method that combines both detection and viewpoint estimation and performs better than any existing approach. We also study the impact of the formulation of viewpoint estimation either as a discrete or a continuous task, we quantify the benefits of deeper architectures and we demonstrate that using synthetic data is beneficial. With all these elements combined,

we improve over previous state-of-the-art results on the Pascal3D+ dataset by a approximately 5% of mean average viewpoint precision.

Résumé

La récente mise à disposition de grandes bases de données de modèles 3D permet de nouvelles possibilités pour un raisonnement à un niveau 3D à partir des photographies. Cette thèse étudie l'utilisation des réseaux de neurones convolutifs (CNN) pour mettre en relation les modèles 3D et les images.

Nous présentons tout d'abord deux études préliminaires qui sont utilisées tout au long de cette thèse : une méthode pour la réduction automatique de la mémoire pour les CNN profonds, et une étude des représentations internes apprises par les CNN pour la mise en correspondance d'images appartenant à des domaines différents. Dans un premier temps, nous présentons une bibliothèque basée sur Torch7 qui réduit automatiquement jusqu'à 91% des besoins en mémoire pour déployer un CNN profond. Dans un second temps, nous étudions l'efficacité des représentations internes des CNN extraites d'un réseau pré-entraîné lorsqu'il est appliqué à l'identification d'images provenant de modalités différentes (réelles ou synthétiques). Nous montrons que malgré la grande différence entre les images synthétiques et les images naturelles, il est possible d'utiliser des représentations des CNN pour l'identification du modèle de l'objet. Nous présentons aussi aussi une extension multi-vue que nous illustrons avec une application pour le rendu basé sur l'image.

Nous présentons ensuite une méthode pour la détection d'instances 3D sur les images : à partir d'un modèle 3D (ou un ensemble de modèles 3D) et d'une image, le modèle est localisé et aligné sur l'image. Nous montrons que l'application directe des représentations obtenues par un CNN ne suffit pas, et nous proposons d'apprendre une transformation qui rapproche les représentations internes des images réelles des représentations des images synthétiques. Nous évaluons notre approche à la fois qualitativement et quantitativement sur deux jeux de données standard: le jeu de données IKEAobject, et le sous-ensemble du jeu de données Pascal VOC 2012 contenant des instances de chaises, et nous montrons des améliorations sur chacun des deux.

Enfin, nous nous éloignons des instances et nous essayons d'extraire des informations 3D pour les catégories entières d'objets. Récemment, les CNN ont été utilisés pour l'estimation de point de vue des objets dans les images,

parfois avec des choix de modélisation très différents. Nous présentons ces approches dans un cadre unifié et nous analysons les facteurs clés qui ont une influence sur la performance. Nous proposons une méthode d'apprentissage jointe qui combine à la fois la détection et l'estimation du point de vue, et qui fonctionne mieux que toutes les approches existantes. Nous étudions également l'impact de la formulation de l'estimation du point de vue comme une tâche discrète ou continue, nous quantifions les avantages des architectures de CNN plus profondes et nous montrons que l'utilisation des données synthétiques est bénéfique. Avec tous ces éléments combinés, nous améliorons l'état de l'art d'environ 5% pour la précision de point de vue moyenne sur l'ensemble des données Pascal3D+.

Contents

Chapter 1

Introduction

1.1 Objectives

An application that automatically retrieves a specific piece of furniture in large online catalogs from only a single picture; a robot able to manipulate new objects without human supervision; augmented-reality systems that can change the arrangement or types of objects in a real scene: these are a few examples of applications that could benefit from the work presented in this dissertation. The goal of this thesis is to develop models and techniques to obtain important pieces of information to perform these tasks efficiently: to relate three-dimensional (3D) information of the objects in a scene with single images.

This is a very generic problem which can be approached in several ways. In this thesis, we focus on exploring the representational power of Deep Convolutional Neural Networks (CNNs) to achieve this goal. CNNs are a family of computational models that are able to extract high-level information from data (in our case, images) in a hierarchical manner. Initial levels in this hierarchy extracts low-level information, such as oriented edges, while intermediate levels in the hierarchy hold more complex information, inferred from the previous levels, such as structured patterns and textures. Going higher in the hierarchy, the combination of these mid-level features allows to represent highly-semantic information in a compact manner. CNNs are trained end to end, and were recently shown to perform extremely well in a wide variety of tasks. We are interested in exploring if the mid- and high-level information contained in deeper levels of this CNN hierarchy are suited for three-dimensional reasoning in images.

The three-dimensional informations that are the most meaningful to be extracted from images are most certainly application-specific. In this work, we focus our attention on inferring object properties that do not require reasoning at a whole-scene level and that only requires limited context. We

are interested in predicting the orientation and instance type of rigid objects that can be defined by a three-dimensional model. More precisely, we are addressing here the following tasks, which are illustrated in Figure 1.1:

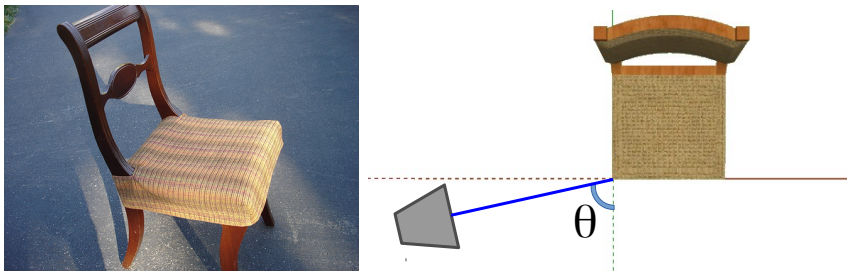
Retrieving a 3D model from images (from images to 3D models). The recent availability of large collections of 3D models [17, 119] allows to explore new possibilities for a 3D understanding of images. Given an image containing a single object and a large catalog of 3D models, we are interested in finding the 3D model that is the most similar to the object depicted in the query image. We restrict ourselves to rigid-body objects categories; no mesh deformations are thus taken into account.

Detecting a 3D model in an image (from 3D models to images). We are also interested in approaching the retrieval task the other way around. Instead of providing an image where we know there is an object and asking which object is pictured, we start from a 3D model and ask the opposite question: *is an object corresponding to the 3D model in this image, and if so, where?* This corresponds to a detection task, where we know in advance the object we are looking for, but we only have its 3D model and no real image of this object to help guide the detection. This is more complex than the retrieval task, as we also need to estimate a detection probability and not just a ranking between rendered views.

Object pose estimation. Estimating the pose and location of an object given a single image can be seen as a first step towards a three-dimensional understanding. While traditional approaches often focused on estimating the pose for a specific object instance, such as a particular type of IKEA chair, in this work we are interested in predicting orientation for whole categories of objects. This is an inherently more difficult task to solve, as in addition to the visual diversity that comes from different lighting conditions, occlusions and camera noise, we also need to cope with the intra-class variability of shapes and textures.

(a) **Instance retrieval.** Given a query image (left), find the 3D model and orientation that is the closest to the object depicted in the query image.

(b) **3D model detection.** Given a 3D model (or a collection of 3D models), find and align in a photograph all possible instances of the model.



(c) **Object pose estimation.** Estimate in an image the pose of any object of a set of pre-defined categories.

Figure 1.1: The three main tasks addressed in this thesis.

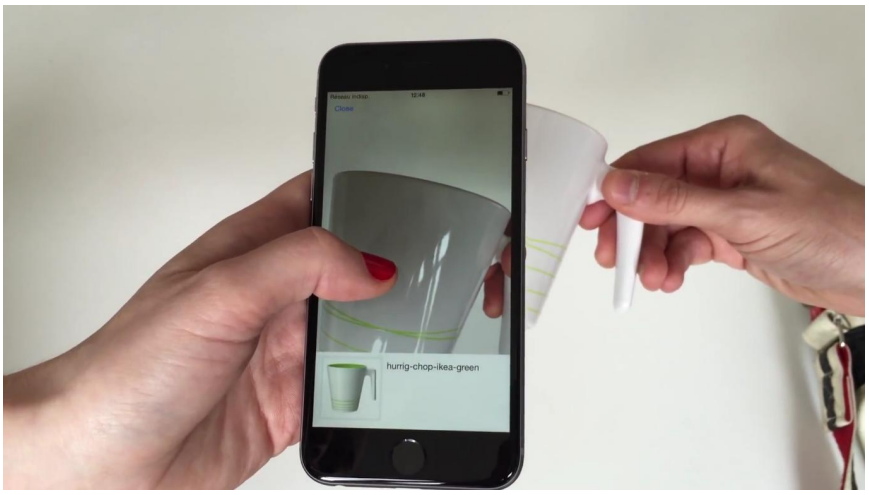
1.2 Motivation

With the advent of large 3D model repositories such as Shapenet [17] and Trimble 3D Warehouse [1], new exciting possibilities to reason about 3D objects in 2D images have appeared, at both research and industrial level. If we were able to automatically align 3D models in 2D images, we could directly transfer all the rich information from the CAD models to the images themselves, such as 3D normals, relative depth, part segmentation, grasping usage instructions, and any other information available in the model. Some examples of applications that leverages this 3D information are illustrated in Figure 1.2 and include:

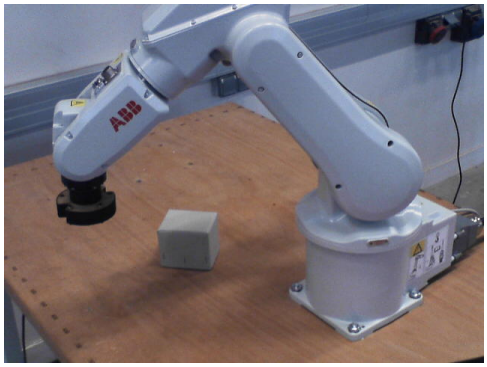
Image-based rendering. Traditional approaches for image-based rendering usually performs poorly in highly-specular surfaces, like the metallic surfaces of cars, or when transparency is present, as in windows, because the 3D reconstruction is not reliable in these cases. By automatically detecting, retrieving and aligning similar 3D models to the objects present in the scene, such as cars, we can improve the quality of the rendering on those surfaces, as we can directly leverage the 3D information present in the models. This enables more realistic augmented-reality experiences, without requiring any manual annotation. An example of such application is presented in Chapter 3.

Automatic 3D model retrieval. Nowadays, almost every available product on the market can be bought online. With current product indexes, it is straightforward to retrieve a product given its name. On the other hand, in the absence of a model name or brand, the task of retrieving a specific object becomes much more involved, usually requiring to browse over a catalog of elements from the desired class. It would be much simpler if we could provide an image of the desired product, and automatically retrieve its name and vendors. For rigid objects with little texture, such as industrial pieces, there might not be enough product images available to perform image retrieval using standard approaches without a severe decrease in retrieval accuracy. But for such products, 3D models could be readily available, as prototypes are first modeled in CAD softwares before going to production. In such context, leveraging these 3D models can lead to improved retrieval performances, and thus better customer experiences.

(a) Virtual reality systems which automatically uses 3D models aligned to the scene when rendering the environment, leading to an improved user experience and allowing a greater user interaction. Image from http://www.gputechconf.com/virtual-reality-track



(b) Product retrieval on large object catalogs. From a single picture, the system is able to retrieve the product depicted in the photograph. Image by Moodstocks.



(c) Automatic robotic manipulation.

Figure 1.2: Possible applications of the work developed in this thesis.

Figure 1.3: Small subset of the 3D models used in this work for the "chair" category. On devices with limited memory, leveraging large amounts of 3D models becomes challenging.

Robotic manipulation. Robotic interaction in complex dynamic environments is a very challenging task. In this scenario, the precise position of each object is not known in advance and can change over time. In such an environment, grasping an object is a very complex procedure. Not only the robot needs to identify and localize the object, it also needs to understand the underlying 3D shape associated with the object. An algorithm leveraging 3D CAD models of the object that the robot seeks to manipulate can greatly help attacking this grasping problem. By automatically aligning the 3D model in the scene, we directly get the 3D geometry of the object. More interestingly, if grasping annotations are available in the 3D model, they can be directly transferred to the image, allowing the robot to predict more precise movements and potentially leading to less mishandling and a more effective system overall.

1.3 Challenges

The problems we address in this thesis raise the following issues:

(a) Real image (b) Rendered image

Figure 1.4: For the same object in the same viewpoint, the visual appearance can drastically change between a photograph and a rendered 3D model.

1.3.1 Computational challenges

Using large amounts of CAD models brings the possibility of performing exemplar-based instance retrieval relying solely on synthetic data. Figure 1.3 shows a small subset of the 3D models that will we used in this thesis. But such a large amount of data also brings several computational difficulties. Another difficulty appears when one attempts to compare large numbers of elements together. For example, detecting and localizing in an image the presence of a specific instance of an object requires comparing this instance to many potential candidate regions in the image. When the number of instances to be detected becomes large, this comparison can quickly become very difficult to be performed in a reasonable time. Graphics processing units (GPUs) greatly speed up several tasks that are amenable to parallelization, such as convolutions, which are at the core of modern deep-learning architectures. But GPUs have much less available memory compared to CPUs, so dealing with large amounts of data efficiently on the GPU is challenging. Furthermore, deep CNNs usually have high memory requirements, making it necessary to optimize the way the memory is used in order to maximize GPU utilization when dealing with large amounts of data.

1.3.2 Domain gap between synthetic and real images

Leveraging 3D models enables retrieval on real images for which the query object is very specific and annotated photographs are not available or not easily available. It also enables artificially extending an existing dataset, for example to obtain a diverse dataset with balanced orientations for each class. There is however a considerable visual difference between the synthetically generated images and natural images, as the former usually lacks texture and context, whereas the latter is usually visually very rich in details. Figure 1.4 illustrates such a difference. One way to overcome this problem would be to create realistic 3D scenes for each object, as an attempt to reduce these differences. Such 3D scene creation would be very time consuming, as it usually requires not only good quality textures, but also a full scene model and a realistic lightning model. CNN features are able to extract both low-level and high-level information from images, but it is unclear whether they can directly be used in such disparate domains, or if substantial modifications to these features are needed.

1.3.3 Handling diversity and ambiguity

Predicting the orientation of a whole object category in real images is a difficult task for a variety of reasons:

1. It requires a varying level of invariance for different properties. On one hand, it involves being invariant to illumination, texture and intra-class variability. On the other hand, it requires being discriminative enough to identify small angle perturbations, which don't change much the image, as can be seen in Figure 1.5.

2. The pose of a rigid object instance or category, while well-defined for completely asymmetric classes, is usually ill defined when symmetries are involved. On may think about a square table for example: turning it by 90 degrees does not affect its geometry. As the orientation of an object is a continuous quantity, it is natural to express the pose estimation as a regression problem. There is a fundamental difficulty with this formulation though, as it cannot represent well ambiguities in the prediction.

(a) Visual diversity inside a category (all the chairs have the same viewpoint).

(b) Same instance of a chair at different orientations.

Figure 1.5: Challenges for pose prediction. (a) For the same category, visual appearance of different instances can greatly vary. (b) On the other hand, small angle differences may not significantly change the appearance of an object.

1.4 Contributions

This thesis focuses on relating 3D information of objects in natural images. We start by presenting preliminary studies on the following points:

- memory optimization for neural networks in Torch7. We developed a library that allows to train deep CNNs that otherwise would not fit in memory. It was also useful to enable large image batches in a CNN while computing predictions, enabling faster execution times as larger batch sizes better exploit GPU parallelism.
- study of the effectiveness of CNN features for retrieving CAD rendered views from natural images containing only one centered object.
- multi-view extension of this simple 3D model retrieval approach, which uses information from several images to find the single best 3D model depicted in the views, and application to image-based rendering.

We then present the two main contributions of this thesis:

- a new framework for exemplar-based CAD model detection on real images, which learns a mapping from the CNN features of natural images

such that they better align with the features of CAD models. Our exemplar-based detection framework enables detection given only a single 3D model of an object, and outputs both its location and orientation. When compared against previous approaches for exemplar-based detection, our technique gives state-of-the-art detection results on both the IKEAobject dataset [76] and the chair subset of Pascal VOC2012 validation set [31] when solely synthetic data is used for training.

- an extensive study of different ways of formulating pose estimation with Convolutional Neural Networks. We show that: (a) learning a multi-task classifier to perform both the detection as well as a discrete pose estimation performs best, and (b) leveraging synthetic data for increasing the amount of training data helps both detection and pose estimation. By combining both, we improve the state-of-the-art results on the challenging Pascal3D+ dataset by a considerable margin for all of the proposed metrics.

1.4.1 Publications

The work done during this PhD lead to the following publications:

Peer-reviewed conferences

- *Deep Exemplar 2D-3D Detection by Adapting from Real to Rendered Views*, Francisco Massa, Bryan Russell and Mathieu Aubry, at Conference on Computer Vision and Pattern Recognition (CVPR), 2016 [83]
- *Crafting a multi-task CNN for viewpoint estimation*, Francisco Massa, Renaud Marlet and Mathieu Aubry, at British Machine Vision Conference (BMVC), 2016 [82]
- *Automatic 3D Car Model Alignment for Mixed Image-Based Rendering*, Rodrigo Ortiz-Cayon, Abdelaziz Djelouah, Francisco Massa, Mathieu Aubry and George Drettakis, at International Conference on 3D Vision (3DV), 2016 [85]

Technical reports

- *Convolutional Neural Networks for Joint Object Detection and Pose Estimation : A Comparative Study*, Francisco Massa, Mathieu Aubry

and Renaud Marlet, arXiv preprint arXiv :1412.7190, 2014 [81]

1.4.2 Software contributions

During the course of this PhD, several contributions to open-source projects have been made. In particular, I'm one of the main contributors of Torch7 neural networks package. The following generic libraries were released during my work for this PhD:

Object detection

Generic framework for object detection developed in Torch7, which allows to seamlessly switch between different object detection algorithms, such as R-CNN, SPPnet and Fast R-CNN, and has around 75 weekly unique visitors. This library was used as the back-end for most of the experiments in this thesis.

https://github.com/fmassa/object-detection.torch

Optimize-Net

Generic memory optimizer for Torch7 neural networks. With the advent of very deep neural networks, it became more and more difficult to experiment with the latest models, as they usually require more memory than what is currently available in most GPUs. This library was born from the need of experimenting deep models in constrained environments, allowing to save up to 91% of memory at test time, and 39% at training time. As of December 2016, this library has around 175 weekly unique downloads, showcasing the importance of saving memory in current deep learning frameworks.

https://github.com/fmassa/optimize-net

Code for the projects

In addition to the generic libraries aforementioned, we also released the code corresponding to the papers:

- *Deep Exemplar 2D-3D Detection by Adapting from Real to Rendered Views* at imagine.enpc.fr/~suzano-f/exemplar-cnn/
- *Crafting a multi-task CNN for viewpoint estimation* at imagine.enpc.fr/~suzano-f/bmvc2016-pose/.

1.5 Thesis outline

This thesis is organized as follows: Chapter 2 presents an overview of the related work, Chapter 3 presents an algorithm for automatically reducing the memory requirements for deep convolutional neural networks, as well as a study of which CNN features are better adapted for relating 3D models to 2D images, and we evaluate this study for the task of 3D model retrival in 2D images. Chapter 4 presents our object detection pipeline solely based on CAD models. Chapter 5 presents our pose estimation study. Finally, Chapter 6 concludes this work, presenting possible avenues for future work.

Chapter 2

Background

In this chapter, we give an overview of the concepts and methods that are the most relevant to this dissertation.

We start by a brief presentation of the field of Machine Learning, and more specifically Supervised Learning. After a quick introduction to Machine Learning, which is the necessary foundation for presenting Artificial Neural Networks, we provide an overview of Neural Networks, starting from its origins until the recent breakthroughs in the field, which improved the quality of the results on many Computer Vision tasks by a large factor.

We then present an overview of the object detection task. We review both the classical methods that were employed to address this task, as well as more recent approaches that leverages CNNs and were shown to perform extremely well compared to more traditional approaches.

Finally, we present prior work on estimating rigid-object pose information from 2D images. We subdivide this part in three: first we present work on contour-based alignment, followed by part-based alignment techniques and then methods for general category pose estimation.

2.1 Machine Learning Framework and Notations

In this section, we give some foundations of the Machine Learning framework, where we focus on the Supervised Learning case.

2.1.1 Machine Learning

Machine Learning is the field of Computer Science that studies systems that learn from examples without being explicitly programmed. Such algorithms build models from example inputs, and use them to perform predictions, rather than by following hand-designed rules.

These models can be either parametric or non-parametric. A simple example of a non-parametric approach is the k-nearest neighbors algorithm.

In such a model, decisions are taken by considering the properties of the k nearest neighbors to an example. To define a neighborhood, we need to define a distance measure between examples. For images, one possible approach is to consider the RGB values for all the pixels in the image as the feature representation in a vector space, and use the euclidean distance to compute the neighborhood relationship between images. An example of parametric model is the linear regression. In this model, the relationship between the input examples, which belongs to a vector space, and the desired scalar-valued targets is approximated by a linear function. The parameters of the linear mapping can be adjusted to better explain the training examples.

Machine Learning is usually divided in three subfields:

Supervised Learning: given input examples with their corresponding labels, the goal of supervised learning is to learn a function which maps the input to the labels, such that predictions can be made on unseen data;

Unsupervised Learning: in the unsupervised learning setting, only the input examples are given, and the algorithm tries to discover structure or patterns in the data. Clustering is an example of an unsupervised learning algorithm;

Reinforcement Learning: the algorithm (or agent) interacts with a dynamic environment aiming at performing a specific task. It receives feedback for each decision it takes. The agent then adapts its strategy in order to maximize an objective function which measures how well the task was performed.

In the following section, we expand on Supervised Learning, as it is the framework used in this dissertation.

2.1.2 Supervised Learning framework

In the supervised setting, we suppose we have a dataset $\mathcal{D}$ with N_s examples. The dataset consists of data observations $x^i \in \mathcal{X}$, which represents the input that is fed to the system, and the targets $t^i \in \mathcal{T}$, which correspond to the desired output of the model. More formally, we define the dataset as follows:

$$\mathcal{D} = \left\{(x^i, t^i), i = 1, \ldots, N_s\right\}. \tag{2.1}$$

In what follows, we will restrict ourselves to the parametric case. Let $f^{\mathbf{w}} : \mathcal{X} \to \mathcal{Y}$ be the *decision function*, parametrized by $\mathbf{w}$. The output space $\mathcal{Y}$ can be different from the target space $\mathcal{T}$, in which case a pre-defined function $g : \mathcal{Y} \to \mathcal{T}$ is used to obtain the final prediction of the system. We will explain in more details the role of g later in this section, when introducing the classification case.

Let $y^i = f^{\mathbf{w}}(x^i)$ be the output of the decision function $f^{\mathbf{w}}$ on example x^i. To measure how different the output y^i is from the true target t^i, we define a loss function $\ell(y^i, t^i) : \mathcal{Y} \times \mathcal{T} \to \mathbb{R}^+$ which assesses the quality of the estimation. We seek to approximate the predictions $g(y^i)$ as much as possible to the targets t^i. In order to quantify how far off are the predictions from the targets for a given training dataset $\mathcal{D}$ and decision function $f^{\mathbf{w}}$, we define the empirical risk as the average of the losses over the training set:

$$R_{\text{emp}}(f^{\mathbf{w}}) = \frac{1}{N_s} \sum_{i=1}^{N_s} \ell(f^{\mathbf{w}}(x^i), t^i) \tag{2.2}$$

High values for the empirical risk means that $f^{\mathbf{w}}$ does not approximate well the training data, while a risk of zero indicates that the model perfectly describes the relationship between the input examples and the output targets. In order to correctly model the dependencies between the data and the targets, we look for the parameters $\mathbf{w}$ such that the empirical risk over the training data $\mathcal{D}$ is minimized. We call the function that we want to minimize the *objective function*.

Minimizing the objective function does not guarantee that $f^{\mathbf{w}}$ will perform well in unseen data. For example, whenever $f^{\mathbf{w}}$ has sufficient capacity, it is possible that the model exactly memorizes the training examples, which are often noisy, and possibly performs poorly in unseen examples, because it starts to model the underlying noise, as illustrated in Figure 2.1.

In such a scenario, the model is said to *overfit* the training data. This behaviour is not desirable, as it indicates that the model is unable to generalize well to new examples. One effective way to fight overfitting is to enforce some regularization $\mathcal{R}(f^{\mathbf{w}})$ on the objective function:

$$L(f^{\mathbf{w}}) = \frac{1}{N_s} \sum_{i=1}^{N_s} \ell(f^{\mathbf{w}}(x^i), t^i) + \lambda \mathcal{R}(f^{\mathbf{w}}) \tag{2.3}$$

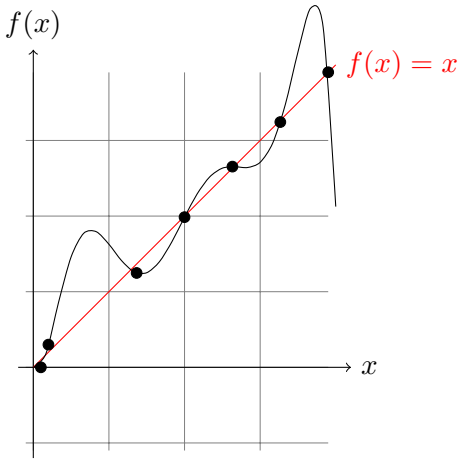


Figure 2.1: Illustration of overfit. Given a subset of noisy points following an affine distribution, merely fitting a polynomial on those points can lead to bad predictions.

with λ a scaling factor which defines a trade-off on the importance of the regularization on the objective to be minimized. The regularization will control the model complexity, for example by enforcing the norm of the parameters $\mathbf{w}$ to be small, for which a common example is the weight decay, given by $\mathcal{R}(f^{\mathbf{w}}) = \|\mathbf{w}\|_2^2$. This helps prevent the model from merely memorizing the training examples.

Another way to reduce the risk of overfitting is to enforce the model to also be able to perform another task. For a fixed budget of parameters, enforcing that the model learns shared representations for different tasks can lead to more informative representations for each task, because it reduces the effect of peculiarities of the data distribution and it can help overcome limitations on the amount of training data for each task.

We now subdivide the supervised setting in two branches: classification and regression. Both are relevant for this work, and will be developed in the following sections.

Classification

In the classification setting, we suppose that each data observation belongs to a discrete number of classes, and the goal of the model is to be able to predict in which class the observation belongs to. A typical example of a classification problem is to assign an e-mail to one of two classes: spam or

non-spam. Let N_c be the number of classes that each observation can take, and N_s the number of training examples in the training dataset $\mathcal{D}$. We define $\mathcal{T} = \{1, \ldots, N_c\}$ as the space of possible labels. Let the space of possible inputs $\mathcal{X} \subset \mathbb{R}^{N_D}$ be a subset of the N_D-dimensional euclidean space, with N_D the dimensionality of the input space, and let $\mathcal{Y} \subset \mathbb{R}^{N_c}$ be the output space of $f^{\mathbf{w}}$. As before, we consider $y^i = f^{\mathbf{w}}(x^i)$ to be the output of the decision function for input x^i. In our notation, we use subscripts to define the individual elements of a vector. In other words, we define the v_i subscript as the i-th coordinate of a vector v. One loss commonly used in CNNs for training classification models is the cross-entropy loss, which is defined as follows:

$$\ell^{\mathrm{CE}}(y^i, t^i) = -\log\left(\mathrm{softmax}(y^i)_{t^i}\right) \tag{2.4}$$

with the softmax function defined as

$$\mathrm{softmax}(x)_j = \frac{\exp(x_j)}{\sum_{c=1}^{N_c} \exp(x_c)}. \tag{2.5}$$

The output of the softmax function can be seen as converting the input vector x such that it is interpretable as a probability distribution, as all the entries are positive (because of the exponential) and sum to 1.

The classification model assigns a score $f^{\mathbf{w}}(x^i)_c$, with $c = 1, \ldots, N_c$, to each of the classes in $\mathcal{T}$ for each observation x^i. To obtain the predicted class from the scores given by $f^{\mathbf{w}}$, we define the conversion function $g : \mathcal{Y} \to \mathcal{T}$ as $g : x \mapsto \arg\max_c x_c$. Thus, the predicted class $\hat{t}^i$ is the one with the highest score, and can be obtained via:

$$\hat{t}^i = \underset{c \in \{1,\ldots,N_c\}}{\arg\max}\ f^{\mathbf{w}}(x^i)_c. \tag{2.6}$$

Regression

In the regression setting, we want to estimate the values of the continuous target variable $t \in \mathcal{T} = \mathbb{R}^{N_c}$ given the values of the input variable $x \in \mathbb{R}^{N_D}$, where N_c is the dimensionality of the target variables, and N_D is the dimensionality of the input variables. The mapping function $g : \mathcal{Y} \to \mathcal{T}$ can be the identity mapping $g : y \mapsto y$, so predictions are performed by evaluating the model $f^{\mathbf{w}}$ parametrized by $\mathbf{w}$. Regression models are widely used for prediction. One example of a regression model is the linear

regression. In such a model, the dependency between the inputs and the targets is approximated by a linear function. Let $\mathbf{w} \in \mathbb{R}^{N_c \times (N_D+1)}$ be the parameters of the model. The parameters include both a multiplicative factor of dimension $\mathbb{R}^{N_c \times N_D}$ as well as an additive bias of dimension $\mathbb{R}^{N_c}$. The linear regression model writes:

$$f^{\mathbf{w}}(x^i) = \mathbf{w} \begin{bmatrix} 1 & x^i \end{bmatrix}, \tag{2.7}$$

with $\begin{bmatrix} 1 & x^i \end{bmatrix}$ the concatenation of a 1 in the beginning of x^i, and $\mathbf{w} \begin{bmatrix} 1 & x^i \end{bmatrix}$ the matrix-vector multiplication between $\mathbf{w}$ and $\begin{bmatrix} 1 & x^i \end{bmatrix}$. In what follows, we simplify the notation by implicitly appending a 1 in the beginning of x^i to take into account the bias term in the model.

Different loss functions can be employed when modeling a regression problem. A common choice is the squared loss, given by Eq. (2.8).

$$\ell(u, v) = \|u - v\|_2^2 \tag{2.8}$$

One drawback with the squared loss is that it is not robust to outliers, so that if two elements are far apart, due for example to noise in the observations or rare events, the loss will be very affected. One way to avoid this problem is to consider the absolute loss, presented in Eq. (2.9), which does not suffer from the quadratic explosion on outliers.

$$\ell(u, v) = \|u - v\|_1 \tag{2.9}$$

The issue with the absolute loss is that its gradient equally penalizes elements that are nearby and elements that are far away, making learning via gradient descent suboptimal whenever the predictions are close to the ground-truth. In such situation, the Huber loss (2.11) can be used. For a pair of real-valued numbers $u_c \in \mathbb{R}$ and $v_c \in \mathbb{R}$, the Huber function is defined by:

$$H(u_c, v_c) = \begin{cases} 0.5(u_c - v_c)^2 & \text{if } |u_c - v_c| < 1 \\ |u_c - v_c| - 0.5 & \text{otherwise} \end{cases} \tag{2.10}$$

and the loss for two vectors $u \in \mathbb{R}^{N_c}$, $v \in \mathbb{R}^{N_c}$ is given by:

$$l(u, v) = \sum_{c=1}^{N_c} H(u_c, v_c) \tag{2.11}$$

where, once again the subscript corresponds to taking the elements of the vector. The Huber loss, which is used in robust regression, combines both the robustness of the absolute loss with respect to outliers as well as the sensitivity of the squared loss.

2.2 Artificial Neural Networks

2.2.1 Origins of Artificial Neural Networks

Artificial Neural Networks are a family of parametric models that have a specific hierarchical structure. The structure is a combination of linear functions followed by non-linearities, which allows the model to learn complex non-linear functions in a compact manner. In this section, we give a brief overview of the mathematical models that originated artificial neural networks.

The perceptron

The origins of Artificial Neural Networks dates back to the end of the 1950's, with the development of the perceptron by Rosenblatt [95]. The term *neural network* has its origins in attempts to find mathematical representations of information processing in biological systems [15]. In what follows, we will give a brief definition of the perceptron model.

Let's consider an observation $x^i \in \mathbb{R}^{N_D}$. For ease of notation, we append a 1 into x^i to take into account the bias, making it a $\mathbb{R}^{N_D+1}$ vector. The perceptron maps the input to a binary output $f^{\mathbf{w}}(x) \in \{0, 1\}$ by considering:

$$f^{\mathbf{w}}(x) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot x > 0 \\ 0 & \text{otherwise} \end{cases}, \tag{2.12}$$

where $\mathbf{w} \in \mathbb{R}^{N_D+1}$ is a vector of real-valued weights. Note that this is equivalent to a linear function $x \mapsto \mathbf{w} \cdot x$ followed by a non-linear *activation*

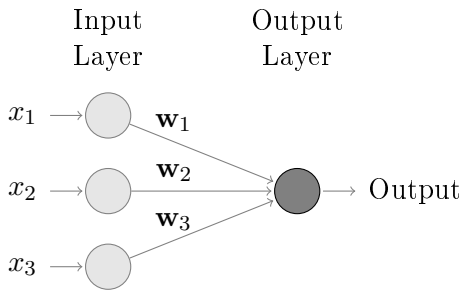


Figure 2.2: Illustration of the perceptron, for an input x of dimension 3 and a single output node.

function $\psi(x)$, and can be equivalently written

$$f^{\mathbf{w}}(x^i) = \psi(\mathbf{w} \cdot x^i), \tag{2.13}$$

where $\psi(x)$ in here is the *Heaviside step function*, defined by $\psi(x) = 1$ if $x > 0$ and 0 otherwise. An illustration of the perceptron is presented in Figure 2.2.

A crucial question is how to select the parameter $\mathbf{w}$ so that the perceptron defined by $f^{\mathbf{w}}(x)$ can perform a specific task. We consider the supervised setting where we have a dataset $\mathcal{D} = \{(x^i, t^i)_{i \in \{1,\dots,N_s\}}\}$, with, for all i, $x^i \in \mathbb{R}^{N_D+1}$ and $t^i \in \{0, 1\}$.

In the original perceptron algorithm, the parameters are updated iteratively by re-evaluating the predictions at each parameter update, and modifying the parameters that yield incorrect predictions.

More formally, let $y^i = \psi(\mathbf{w} \cdot x^i) \in \{0, 1\}$ be the output of the perceptron model. In order to find the set of parameters $\mathbf{w}$ that best explains the dataset $\mathcal{D}$, we perform stochastic parameter updates for every training pair (x^i, t^i) in $\mathcal{D}$ following:

$$\mathbf{w} \leftarrow \mathbf{w} + (t^i - y^i)x^i, \tag{2.14}$$

and the updates from (2.14) are performed either for a predetermined number of iterations, or when the iteration error $\frac{1}{N_s}\sum_{i=1}^{N_s}|t^i - y^i|$ is less than a predefined threshold.

Differentiable activation functions and the Delta Rule

The perceptron contains a discontinuous (and thus non-differentiable) activation function $\psi(x)$. If we replace the activation function by a differentiable one, we can derive a more generic learning rule, called the *delta rule*. Using the same notation as in the previous section, the delta rule, which updates the weights stochastically for every training example, can be stated as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(t^i - y^i)\psi'(\mathbf{w} \cdot x^i)x^i, \tag{2.15}$$

where $\psi'(x)$ is the derivative of $\psi(x)$ with respect to x, and η is the *learning rate*, a real value that controls how fast the updates to the weights are made. The learning rate is a very important hyper-parameter of the learning; too big values makes the learning unstable as the parameters oscillate around the desired solution or might even diverge, whereas too small values leads to a slow training and are more prone to get stuck in a poor local minima.

The delta rule can be derived by minimizing the loss in the output of the neural network for each example in the training dataset via stochastic gradient descent, using a squared distance loss. Gradient descent uses the gradient of the loss function with respect to the weights of the model $\mathbf{w}$ to perform the updates of the weights in a direction that will decrease the loss. For linear activation functions $\psi(x) = x$, the delta rule can be simplified as follows:

$$\mathbf{w} \leftarrow \mathbf{w} + \eta(t^i - y^i)x^i, \tag{2.16}$$

which is very similar to the update rule from the perceptron in (2.14), even though their derivations are different as the Heaviside function is non-differentiable.

2.2.2 Multi-layer neural networks

Despite the initial success of the perceptron in identifying digits in small images, its representational power is very limited. Indeed, it can only learn predictions which are linearly separable in the input space, which is rarely the case. Several extensions were proposed in order to overcome such limitations. In particular, having networks that contain internal representations (also called hidden layers) which are non-linear with respect to the input data allows for more expressive power. Unfortunately, the delta rule explained

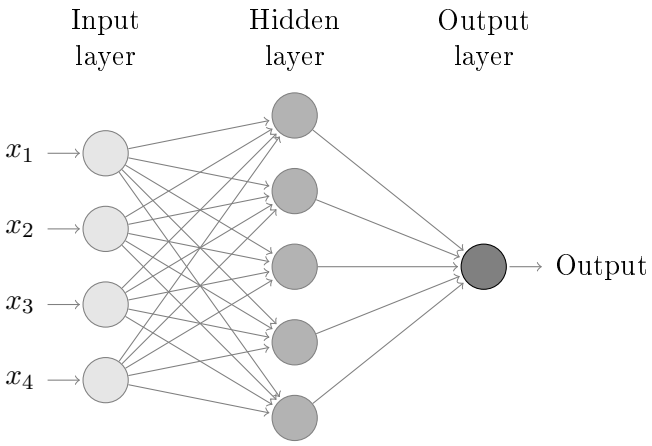


Figure 2.3: Illustration of a feed-forward neural network with one hidden layer.

before does not apply in such situations, as it was specifically tailored for the case where there is no hidden layers, so other learning techniques are needed. One early example is the Neocognitron [38], which stacked together several layers of linear functions followed by non-linearities, and used an unsupervised learning approach based on self-similarity between the input elements and the weights of the model to perform learning. Although such a learning approach allows to learn networks with hidden layers, there is no explicit constraint that ensures that the hidden layers learn an appropriate mapping. As we will see later in this section, it is possible to extend the delta rule to work for such multi-layer neural networks [97, 70]. Before that, we first introduce a sub-category of the multi-layer networks called feed-forward neural network, which is a commonly employed architecture for several tasks.

Feed-forward neural networks

In a feed-forward neural network, the output of each layer is passed as an input to the next layer. Each layer consists of a number of units (or neurons) that computes the weighted linear combination of the layer input, followed by an element-wise non-linearity. Figure 2.3 illustrates a feed-forward neural network with one hidden layer. Let N be the number of hidden layers. Denoting by $\mathbf{o}_n$ the output of layer n with weights $\mathbf{w}_n$, with as before the bias appended for sake of notation, the feed-forward procedure can be written

as follows:

$$\mathbf{o}_n = \psi_n(\mathbf{w}_n \mathbf{o}_{n-1}) \tag{2.17}$$

where $\psi_n(\cdot)$ is a sub-differentiable non-linearity function. Common choices for the non-linearity includes rectified linearities as ReLU, defined by $\psi(x) = \max(0, x)$, the sigmoid function $\psi(x) = (1+e^{-x})^{-1}$ or the hyperbolic tangent $\psi(x) = \tanh(x)$.

Backpropagation

The set of parameters $\mathbf{w} = \{\mathbf{w}_i\}_{i=1,\dots,N}$ are optimized to minimize the objective function $L(f^{\mathbf{w}})$ over the training set $\mathcal{D}$. As we mentioned before, the delta rule is not adapted for multi-layer networks as its formulation only considers the case without hidden layers. To obtain an optimization procedure for the multi-layer case, let's start with a formulation similar to the one used to derive the delta rule. We consider the loss $\ell(y^i, t^i)$ computed for each element in the training set $\mathcal{D}$, which we want to minimize. As for the delta rule, we use gradient descent to perform the optimization, which writes:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{\partial \ell(f^{\mathbf{w}}(x^i), t^i)}{\partial \mathbf{w}}, \tag{2.18}$$

where η is the *learning rate*, which controls the size of the update steps. As is, we note that this is a generalization of the delta rule, for the case where we consider a squared distance as the loss function and where the gradient is computed over the whole training dataset, and not only element by element. To perform gradient descent to find the parameters $\mathbf{w}$ that minimize the loss, we need a way of computing the derivatives of the loss function with respect to the parameters in an efficient manner.

The answer to this question, which is a generalization of the delta rule, was given in [97, 70], and is traditionally called backpropagation. Backpropagation consists of computing, in a recursive manner, the gradients of a module in function of the gradients of the modules that come after it. The derivation of backpropagation is obtained by recursively applying the chain rule of the derivatives in the loss function that we seek to minimize. This makes it possible to construct arbitrarily complex functions by combining a number of smaller blocks, for which the derivative is known, and the gradient of the whole complicated function can be readily computed.

Adding hidden layers to a network potentially increases the capacity of the network to model complex functions. In the early 1990's, Hornik [59] showed that a feed-forward network with a single hidden layer containing a finite number of neurons was capable of approximating any continuous function defined on compact subsets of $\mathbb{R}^n$. But as discussed in [13], an important result in favour of deeper networks is that functions that can be compactly represented by a depth k network might require an exponential number of parameters with respect to the input size to be represented by a depth $k-1$ network.

The objective function optimized during backpropagation of the multi-layer network is not convex with respect to the weights, due to the several layers of non-linearities that are present. Gradient descent can only find a local minima, and for such non-convex functions, it is natural to wonder about the quality of the local minima found. Since the middle of the 1980's, there were already evidences that different local minima in multi-layer networks performed similarly well for a number of tasks [97]. Recently, [66] has provided a mathematical proof for the fact that all local minima in deep neural networks are actually global minimum, given some reasonable assumptions.

2.2.3 Convolutional Neural Networks

Convolutional neural networks (CNNs) [71] are a sub-class of neural networks with constrained connectivity patterns in the linear mapping $\mathbf{w}x$. A principle which has proven very effective in natural images is to hypothesize that the feature representation of an image should be approximately translation covariant. In other words, for an image x with a feature representation $f(x)$, if a translation τ is applied to x, then the feature representation of the translated image should approximately correspond to $f(x)$ translated by τ. This covariance can be imposed by constraining the linear mapping $\mathbf{w}x$ to be a convolution. Enforcing that the linear mapping is a convolution brings the additional benefit that larger images can be used without a huge increase in the amount of parameters of the model. Each unit becomes responsible for detecting a particular pattern in the image, for example an oriented edge in an image. With convolutions, the output of a layer is translated by the same amount as the translation of the input. In order to make the output of the network invariant by small translations and deformations, a Max Pooling operation was introduced. Max Pooling is a form of non-linear

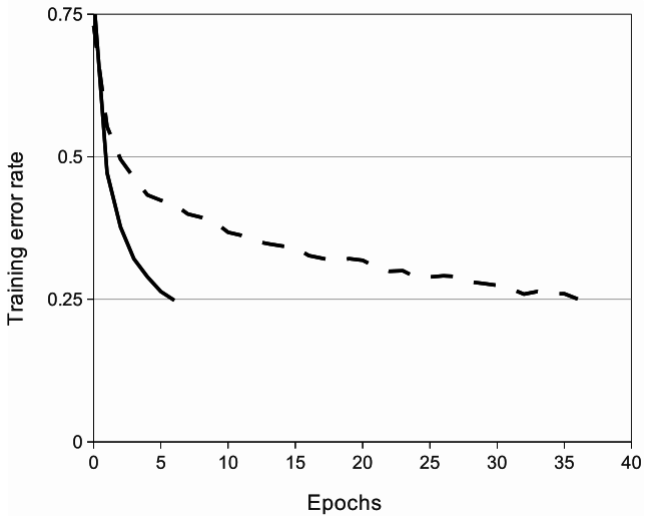


Figure 2.4: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). Figure from [68].

down-sampling, which uses the maximum operation in a local neighborhood to aggregate the feature representation.

CNNs and the ILSVRC competition

Even though most of the necessary foundations of CNNs have been established since [71], there was only a handful of tasks for which CNNs excelled most traditional approaches based on hand-engineered features. It was only after the seminal work of Krizhevsky *et al.* [68] on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [98] in 2012 that CNNs started to attract general attention. This breakthrough was due mainly to the following reasons:

GPU: Convolutions, pointwise non-linearities and matrix multiplications, all of which composes the basic building blocks of traditional Convolutional Neural Networks, are naturally amenable to parallelization. With the advent of the CUDA programming language, whose syntax resembles the syntax of C++, implementing programs that parallelize over hundreds or thousands of cores became much more accessible to the machine learning community. This parallelization brings a crucial

speed-up both at training and testing times, allowing for bigger models to be trained within reasonable time.

ReLU: One of the biggest problems with deep CNNs prior to 2012 was that they were very hard to train. Deep networks suffered from a *vanishing gradient* problem, where the gradients in the initial layers became increasingly small, harming training. One common way to address this problem was to initialize the weights of the network via a layer-wise unsupervised pre-training. This solution was not optimal for a few reasons: layer-wise pre-training was time consuming, as each layer had to be trained separately prior to the supervised training, and unsupervised learning of convolutional filters was difficult to optimize. Rectified Linear Units (ReLUs) [47] helped address these difficulties. Contrary to standard saturating non-linearities like sigmoid or hyperbolic tangent, ReLU does not suffer from vanishing gradients when the activations become larger. This accelerates training time considerably, allowing deeper networks to be trained in reasonable time, as can be seen in Figure 2.4.

Dropout: Dropout [105] is a regularization technique that was found crucial to combat overfitting. During training, dropout randomly sets half of the outputs of a specific layer to 0, which means that those zeroed elements won't participate in the backpropagation. This prevents complex co-adaptations between features, as each neuron won't be able to rely on the output of another neuron. At test time, all of the neurons are left active, but the output is multiplied by 0.5 to compensate that twice as many neurons are active. Dropout can also be interpreted as an efficient way of performing model ensembling, which only costs about a factor of two during training, and does not introduce any cost at test time.

Big model: The architecture proposed by Krizhevsky *et al.* contains 60 million parameters, and 8 layers of non-linearities, and is illustrated in Figure 2.5. Using a network with such a size was without precedents, but was found necessary to model such a complex task as the one required to classify between the 1000 classes of ImageNet. Indeed, [68] mentions that by removing any of the intermediate layers results in a

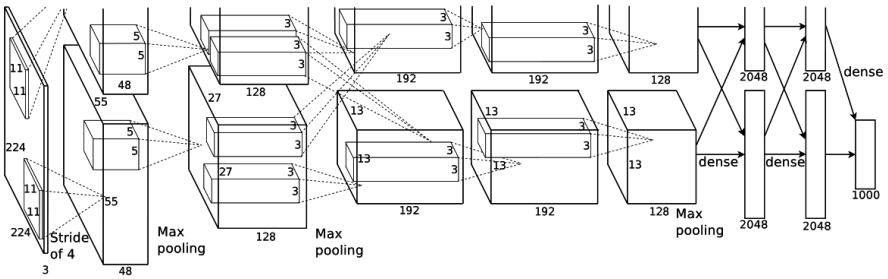


Figure 2.5: An illustration of the network architecture presented in [68], also called AlexNet in the literature. It consists of 5 convolutional layers, followed by 3 fully-connected layers. Figure from [68].

loss of about 2% for the top-1 performance of the network, indicating that the depth was fundamental to their good results.

Big dataset + dataset augmentation: Training such a huge model in a supervised manner is only possible if there exists enough training data to allow the model to generalize and not overfit. For very large networks, even the 1.2 million images from ImageNet may not be sufficient, so several techniques for artificially increasing the size of the dataset were employed, such as random scalings and croppings, horizontal flipping of the images and small color deformations.

Most of those contributions were already individually presented before [68], but it required ground-breaking results in a challenging competition such as ILSVRC to attract the attention of the computer vision community back to CNNs. Since then, several major improvements in the way of training and factorizing CNNs were made, improving state-of-art performance on many tasks [124, 126, 41, 30].

Visualizing the internal representations

What makes deep CNNs perform so well? Given the impressive results obtained by deep CNNs for vision tasks, it is natural to wonder what is internally learned by the network. By inspecting the internal feature activations of the network, Zeiler and Fergus [128] showed that earlier layers of the network are responsible for detecting oriented edges and colors, while later layers learn more complex patterns, such as grids, circles or even faces, as

can be seen in Figure 2.6.

Training larger and deeper networks

Simonyan *et al.* [103] proposed a deep CNN architecture which replaces large convolutional filters present in the original architecture from [68] by a series of 3×3 filters, with ReLU non-linearities in between. For example, by replacing one 5×5 filter by two 3×3 filters, the effective receptive field remains the same, meaning that the same region of the image is covered by the convolutions. This factorization increases the number of non-linearities present in the network and additionally decreases the amount of parameters. This was shown very beneficial, and greatly improves the representational power of the model, leading to an important improvement in terms of classification accuracy.

Shortly after, Ioffe and Szegedy proposed Batch Normalization [62], a simple technique that removes the covariate shift from the feature representations by normalizing the feature maps over each mini-batch. This also has the positive advantage that the outputs of each layer are in the same range. Batch Normalization allows for a faster training of the networks, and eliminates the vanishing gradient problem.

Given such improvements, it was natural to wonder if we only needed more powerful machines and bigger models to achieve better results. In [55], He *et al.* showed that simply increasing the depth in feed-forward CNNs doesn't necessarily improve classification accuracy, but by learning residual functions $h(x) = x + f(x)$, it is possible to train much deeper networks with increasing accuracy.

While increasing depth was shown very beneficial for learning more complex functions, it also increases by a large factor the amount of memory required by the CNN. In Section 3.1, we will present our automatic memory optimization framework, which allows to train deeper models for the same amount of available memory.

2.3 Object detection

Extracting high-level information from images is one of the utmost objectives of Computer Vision. Object detection can be described as the field that aims at providing tools to answer the question *what objects are where?*

Figure 2.6: Visualization of what activates the most each neuron in a CNN trained for classification on ImageNet 2012 training set, for different layers. Each image on the left contains reconstructed patterns from the validation set of ImageNet 2012 that cause high activations in a given feature map. On the right, the corresponding image patches for each feature map are shown. Figure by Zeiler and Fergus [128].

We follow the standard formulation of *category-level object detection* discussed by Girshick in his PhD thesis [42], where the goal is to retrieve and localize objects of predefined categories in still images. This localization is usually expressed in terms of a tight bounding box which delimits the visible part of the object.

This is a very challenging task for several reasons. Firstly, still images correspond to noisy two-dimensional projections of a three-dimensional scene. For the same object from two different viewpoints, the image representation can greatly differ. For smooth objects and small viewpoint changes, the difference in the image can be approximated by affine transformations. But this does not hold anymore as soon as a previously occluded part of the object becomes visible, or when a previously visible part of the object becomes occluded. Secondly, for the same scene, differences in illumination can dramatically affect the image representation. Furthermore, for the same object, truncations and surrounding clutter also modify the image representation in a complex manner. Thirdly, the definition of category is usually specified in terms of usage, and not in terms of visual appearance of the objects. This can entail a large intra-class variation that makes the task of object classification more challenging. To give an example, imagine a "chair" and all the possible variations that can be present, some of which are illustrated in Figure 1.5. It can have a back or not, its shape can be rectangular or ellipsoidal, with or without arm-chairs, with four legs or not. While all of the aforementioned properties can help differentiate a chair from a dog, some of them could also apply to tables.

Standard datasets have helped compare computer vision algorithms in the same setup. Pascal VOC Challenge [31] has been one of the most influential datasets for computer vision. To give some context, in the Pascal VOC 2007 Challenge, the top performing method achieved a mean Average Precision of 21% [35], by using a multi-scale deformable parts model based on HOG features. By 2016, performance has surpassed 85% [55] by using very deep networks trained end-to-end for the detection task. More recently, a more challenging dataset named Common Objects in Context [77] (also known as COCO) was released. With 80 classes and more than 200k images and 500k object instances segmented, it provides a new extremely challenging setup for object detection and instance segmentation.

2.3.1 Classical view

Initial work on category-level object detection focused on specific classes such as person or car, due to the lack of richer datasets for the task of object detection. One example of a successful category object detector is the face detector from Viola and Jones [118] in 2001. It uses Haar-like features and a cascade of classifiers to perform detection. Weak classifiers are used first, in order to remove most of the false positives, and more complex classifiers are applied to the selected regions of the previous classifiers. The Viola-Jones detector performes fairly well on frontal faces, but struggles whenever rotations are present.

Another line of research has developed methods that describe objects in terms of its parts. In [2], Agarwal and Roth propose a category detector which learns a vocabulary of object parts, together with the relationship between the parts and represent that relationship inside the feature representation. In [36], Fergus *et al.* propose a constellation model where parts are constrained to be in a sparse set of locations given by an interest point detector, and their geometric arrangement is determined by a Gaussian distribution. Leibe *et al.* [72] propose to learn a codebook of local appearance descriptors, and an implicit shape model that specifies where on the object each codebook entry may occur. In their approach, local patches are extracted around interest points and compared to the codebook. Matched patches vote for the position of the object in the image, leading to object hypotheses.

An important improvement in the field happened with the development of Histogram of Oriented Gradients (HOG) [25] features. The original article focused on pedestrian detection and showed significant performance improvement compared to previous approaches. HOG computes local histograms of image gradient orientations similar to SIFT [79] but in a dense grid, and were shown to be robust to illumination and small deformation changes. In order to identify the pedestrians in an image, a SVM classifier was trained on HOG features extracted from positive and negative patches.

Most datasets until 2005 consisted of objects in very restricted orientations and positions. For example, most of the images in Caltech-101 dataset [33] were centered and aligned in a stereotypical pose. In INRIA-Person dataset [25], only upright people were annotated, and there was very little variation in poses. The first version of the Pascal VOC challenge hap-

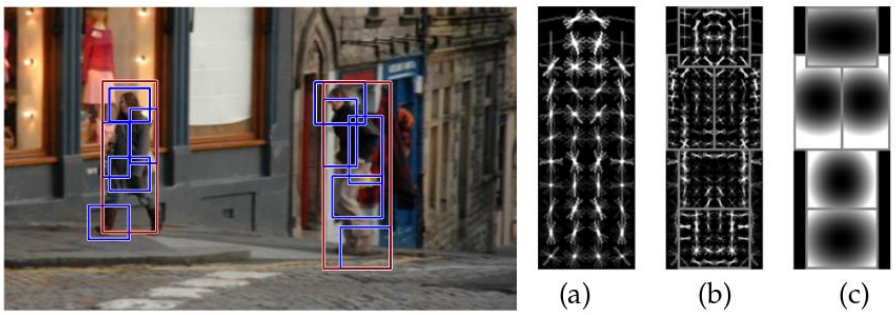


Figure 2.7: DPM [34] object detection framework. (a) coarse root filter, (b) high resolution parts filters and (c) spatial model for the location of each part. Figure from [34].

pened in 2005 [32]. It featured more complex images compared to previous datasets and showcased the limitations of existing approaches when faced with natural images in less controlled environments. One of such limitations was the use of a fixed template HOG per category, which limits the variability of orientations and deformations that the detector can identify. To address this limitation, deformable models based on HOG elements were proposed [35]. Instead of having a single fixed template to model the whole object, several templates are used to model different parts of an object. The relations between parts are usually expressed as a graph, where the nodes correspond to the parts and the edges to the relation between parts. The structure of the graph can be arbitrary, but to have fast and exact inference, tree or star structures are preferred.

In [35, 34], Felzenszwalb *et al.* proposed a discriminatively trained, multi-scale deformable parts model (DPM) that greatly improved detection performance on Pascal VOC 2006. Their system, illustrated in Figure 2.7, uses a low resolution root filter, combined with high-resolution part filters. The object parts are treated as latent variables and are learned together with the classifiers.

2.3.2 CNN-based object detection

CNNs have been used for object detection since the early 1990's [115], but with limited success and leading approaches relied on hand-crafted features as presented in Section 2.3.1. After the impressive classification results ob-

tained by Krizhevsky *et al.* on the ImageNet Large Scale Visual Recognition Challenge in 2012 [68], there was a vigorous discussion about the significance of those results to the computer vision community[1]. The main concern was to know to what extent the CNN classification results from a CNN trained on ImageNet could generalize to object detection. This question was answered concurrently by Sermanet *et al.* [100] and Girshick *et al.* [44]. In [44], Girshick *et al.* show that the features learned from a network trained on ImageNet transfer very well to object detection, and obtain an improvement of more than 30% relative to the best previous results on Pascal VOC 2012 detection challenge. The improvement in detection performance is mainly due to the better features that are learned end-to-end by the CNNs, which are able to capture more than simply the contour of objects, as in more classical object detectors based on HOG features. Furthermore, CNNs can be seen as a generalization of DPMs for object detection [45], where the feature representation is learned and not hand-crafted using HOG features. Since then, the paradigm of *supervised pre-training* followed by *task-specific fine-tuning* became widely adopted, leading to new state-of-art results in a wide range of computer vision tasks, such as edge detection [124], semantic segmentation [126], bounding box proposals [41], object-viewpoint estimation [112] and depth estimation [30].

Using regions for selective classification

How to adapt a classification network for a detection task? In [44], Girshick *et al.* propose Region-CNN (R-CNN), where bounding box proposals from Selective Search [114] are used to restrain the search space for the possible locations of the objects, and the last classification layer of the CNN is replaced by a randomly-initialized layer. The R-CNN detection system is illustrated in Figure 2.8. Each region is fed to the CNN and classified between background or one of the target classes (20 in the case of Pascal VOC 2012). The CNN is then fine-tuned with a small learning rate to avoid modifying too much the original network parameters, and the other hyper-parameters are kept the same.

One important aspect of their system is how to select the patches which are used for training. The number of ground-truth objects in Pascal VOC

[1] https://plus.google.com/+YannLeCunPhD/posts/JBBFfv2XgWM

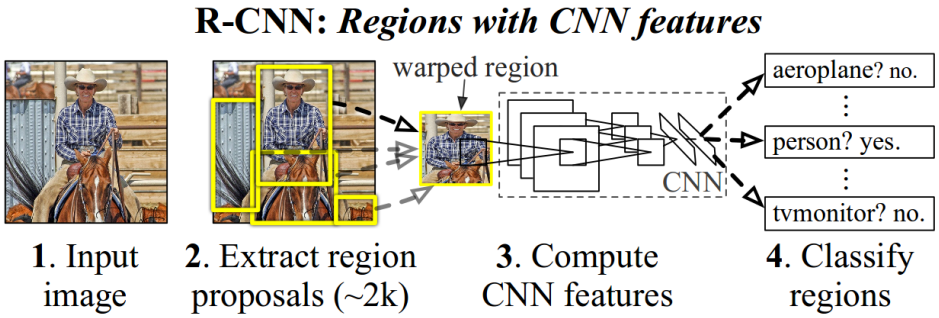


Figure 2.8: R-CNN object detection framework. Figure from [44].

2012 *trainval* is of 27450 for 11530 images, much smaller than the 2M regions provided by Selective Search. To cope with this high imbalance, every box proposal with an intersection-over-union overlap greater than 0.5 with a ground-truth is considered as a positive element during training, effectively increasing the positive training set by a large factor. Additionally, as there are many more background patches in the images than object patches, they balance every mini-batch such that 75% of the patches comes from the background, and the remaining ones are randomly sampled from the positive patches.

Improving computational efficiency by removing redundancy

A drawback of R-CNN is that each region is treated independently from the CNN point of view. Thanks to the Max Pooling present in most CNNs, the features are invariant with respect to small translations in the input image, meaning that two regions in the image with high overlap should have very similar convolutional features. This motivates reusing the feature maps instead of recomputing them for every region. In this line of thought, Sermanet *et al.* [100] apply a CNN densely over the image, predicting both the classes and the bounding boxes, instead of relying on external region proposals. Even though more run-time efficient compared to R-CNN, the results are less accurate due to more false positives because of their dense sliding-window approach.

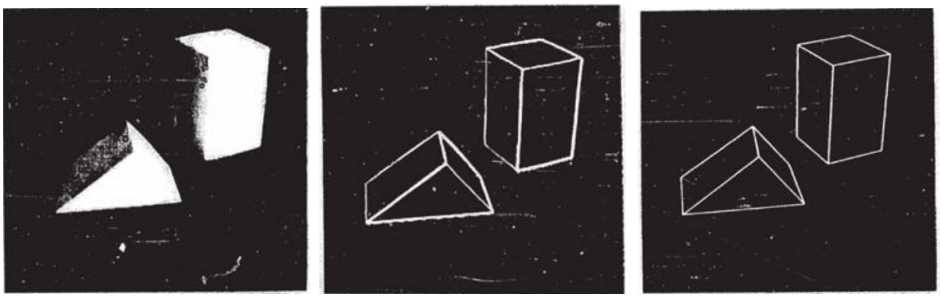
In [54], Kaiming *et al.* propose SPPnet, which leverages region proposals from [116] as in R-CNN, but instead of passing every image region inde-

pendently to a CNN, SPPnet computes the convolutional feature maps for the entire image, and fine-tunes only the fully-connected layers, effectively sharing computations among regions. The regions in the image space are projected into the feature space, and an Adaptive Max Pooling is performed in each region to output a fixed-size feature which is fed to the classifier. A limitation of SPPnet is that the convolutional filters are fixed and are not fine-tuned for detection. To circumvent this problem, Girshick propose Fast R-CNN [43], a method similar to SPPnet but which backpropagates through the Adaptive Max Pooling layers as well. In order to make training fast, he also proposes a different way of sampling training regions. Instead of selecting random patches from the whole pool of regions (ROI-Centric Sampling), he proposes to first sample a set of images, and then to sample patches from those images (Image-Centric Sampling). While this sampling could potentially reduce the diversity of patches that are presented to the network at each mini-batch, Girshick shows that even by sampling as little as two images in a mini-batch is enough to provide diverse enough regions for the network to be well optimized. Furthermore, it is very efficient in re-using the convolutional features.

Most experiments presented in this dissertation build upon object detection techniques. As a pre-requisite for the work we will present in Chapter 4 and Chapter 5, we developed a generic object detection framework on Torch7 that supports both R-CNN, SPPnet and Fast R-CNN, and which was also used as a starting point for further research using object detection [127, 102].

2.4 Pose estimation

The task of estimating the pose of objects in images has been studied since the mid 1960's [93], and different ways of approaching the problem have been proposed. In this section, we discuss three directions that have been explored in the literature. In Section 2.4.1, we present an overview of approaches that aligns known rigid 3D models to 2D images by matching 3D edges with image contours. In Section 2.4.2, we discuss prior work on alignment approaches that decomposes the object into parts and then retrieves the orientation given the arrangement of the parts. Finally, in Section 2.4.3 we present approaches that estimate the pose information for whole object categories, and not only single instances.



(a) Input image (b) differential image (c) reconstructed 3D solids

Figure 2.9: Object instance-level alignment by Lawrence Roberts [93].

2.4.1 Contour-based alignment

Since the early ages of Computer Vision, there has been an interest in aligning 3D models to images. Roberts in the abstract of his PhD thesis [93] explains that his ultimate goal is "to make it possible for a computer to reconstruct and display a three-dimensional array of solid objects from a single photograph". This is an ambitious goal, so in his work he restricted himself to the case where the objects have a known three-dimensional shape, thus being the first to consider the 2D-3D instance alignment problem. His work, as most of works until the 1990's [84], relies on object contours. The main idea of such approaches consists on using contours as the common representation between the 2D image and the 3D model, and the information from several edges are combined in order to align the 3D model to the image. The 3D edges from a 3D model are projected in the 2D image, which makes it possible to compare the 3D edges to the contours obtained from the image. Using contours bypasses the visual differences that exist between both representations, and also makes the 2D-3D correspondence more robust to small illumination and color changes. Several methods have been developed to aggregate the information from different edges. Roberts [93] uses the hypothesis of a block world to recover polygons from sets of lines. In [61], Huttenlocher and Ullman use an hypothesis-test paradigm where, given keypoints obtained by edges corners and inflexions, correspondences between the image and the model are used to hypothesize a pose, and the pose is kept if the rendered model in the proposed pose is coherent with the image. In [78], Lowe uses the idea of line grouping to hypothesize a smaller

number of possible correspondences between the image and the model.

More recently, a number of approaches leveraging contour information for instance alignment have been developed. In [5], Arandjelović and Zisserman retrieves sculptures using HOG descriptors on edge maps. In order to reliably obtain the edges from photographs, the authors present a solution which trains a classifier on super-pixels to distinguish them as either sculpture or not-sculpture. In [76], Lim *et al.* uses hand-made descriptors based on the contour of the objects to perform model retrieval and initial alignment, and the contours are once again used for refining the pose estimation of non-textured objects.

Contour detection

The success of those approaches depends on reliably finding the contours of the objects in the image. As such, a number of techniques have been developed for computing the edges on images. A classical example is the Canny edge detector [16]. Since then, a number of techniques have been proposed to improve edge detection results, making it more robust to textures and repetitive patterns, either by using image statistics [6, 64] or machine learning techniques [27, 28, 124].

2.4.2 Part-based alignment

Rigid object viewpoint estimation was first approached in the case of object instances with known 3D models, together with their detection, as presented in Section 2.4.1. These approaches were extended to whole object categories by leveraging techniques from object detection presented in Section 2.3. In [46], Glasner *et al.* propose an approach that integrates 3D reasoning with an appearance-based voting scheme, which relies on a non-parametric representation of the object class. Hejrati and Ramanan [56] present a method for detecting and analyzing the 3D configuration of rigid objects that consists of two steps. In the first step, a variant of DPM [34] is used to propose an initial detection and 2D estimates of shape via a number of detected keypoints. In the second step, the estimated detection and shape are refined by using an explicit 3D model of shape and viewpoint. In [90], Pepik *et al.* extend DPM to include both estimates of viewpoint and 3D parts that are consistent across viewpoints.

Another line of research consists of using parametric models for performing the pose estimation. In [122], Xiang and Savarese propose the Aspect Layout Model (ALM), which first constructs a parametric model of an object category via a collection of 3D models by decomposing the objects in parts, and then perform the detection and alignment using a Conditional Random Fields (CRF) [69] formulation.

More recently, there has been an increased interest in techniques leveraging large collection of 3D models, thanks to the availability of datasets such as ShapeNet [17] and ModelNet [119]. Aubry *et al.* [7] propose a technique for detecting and retrieving the most similar 3D model and orientation in a image. It consists of three steps: (i) representing each model as a collection of view-dependent mid-level visual elements learned from rendered views, (ii) a calibration of the different visual elements and (iii) the matching of the visual elements on the test images, which allows small deformations but preserves the viewpoint and style constraints. While their approach detects and aligns instances, as the techniques presented in Section 2.4.1, because they leverage a large number of instances for the same category, detection and pose estimation for whole categories is possible.

2.4.3 Category pose estimation

As with object detection presented in Section 2.3.1, initial work on category pose estimation focused either on faces or cars. One of the first works on category pose estimation which was able to reliably detect objects in a wide range of orientations is the paper of Schneiderman and Kanade [99]. They start by discretizing the possible orientations of each category, and learn separate object detectors for different views.

Most datasets for category pose estimation until the 2010's were restricted to single categories, approximate orientations, or were not publicly available. In order to have a standard and challenging dataset for category-viewpoint estimation comprising several categories, Xiang *et al.* propose the Pascal3D+ dataset [121], which extends the Pascal VOC dataset [31] by aligning a set of 3D CAD models for 12 rigid object classes, which enables the use of learning-based approaches leveraging real images, similarly to what was done for detection. Together with the Pascal3D+ dataset, Xiang *et al.* [121] proposed an extension of the method of [90] which also predicts the viewpoint of the object. CNN-based approaches, which were

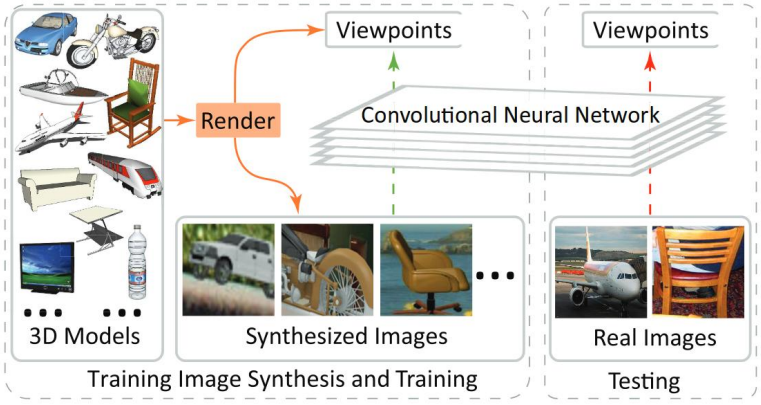


Figure 2.10: Render for CNN object category viewpoint estimation framework. Figure from [107].

until the availability of the Pascal3D+ dataset limited to special cases such as faces [86] and small datasets [87], also began to be applied to this problem at a larger scale. In [112], Tulsiani and Malik used a simple classification approach with the VGG16 network [103] and annotations for ImageNet objects to obtain very good pose estimation results on Pascal3D+. Su *et al.* [107] introduce a discrete but fine-grained formulation of the pose estimation which takes into account the geometry of the pose space, and demonstrate using AlexNet [68] that adding rendered CAD models could improve the results over using Pascal VOC data alone. Their framework is illustrated in Figure 2.10. Recently, ObjectNet3D [120], a new large-scale dataset for category-viewpoint estimation was made available. ObjectNet3D contains 100 categories and 90k images, and provides a new challenging setup for category-viewpoint estimation.

Chapter 3

Preliminary studies

In this chapter, we present three contributions that are going to be used through this dissertation. We first approach the problem of reducing the memory requirements of using deep CNNs, which is an important problem as deep CNNs have recently shown very good results, as discussed in Section 2.2.3. Then, we explore the use of CNN features for performing 3D instance retrieval in 2D images. We suppose we have available a potentially large database of 3D models. Our goal in this task is actually to study which CNN features are better adapted to relating natural images of objects with synthetic rendered views of 3D models representing these objects. Then, we extend the 3D model retrieval to use information from multiple images, and apply it to the image-based rendering task.

We place ourselves in the context of the instance-based alignment discussed in Section 2.4.1 and Section 2.4.2. We tackle the 2D-3D retrieval task by considering the 3D model as a set of 2D rendered views, and by matching the query image with the set of 2D rendered views for each model. By focusing only on 2D images, we can use the representational power of off-the-shelf CNNs that were pre-trained on large annotated datasets such as ILSVRC [98]. For a fixed number of 3D models, the amount of different 2D rendered views that can be generated is potentially unlimited. We target databases of around 1000 models, with 100 rendered views per model, making up to 100k different rendered images to compare against.

The contributions of this chapter are three-fold:

1. In Section 3.1 we propose an automatic memory optimization algorithm for Torch7 neural networks [24]. It facilitates the retrieval task when thousands of models are available and more generally it is extremely useful when experimenting with deep CNNs. Our algorithm works by reusing memory that is not needed anymore by the network to perform its computations. Without affecting runtime performance,

it is able to save as much as 91% of the memory usually required by the default Torch7 neural networks package.

2. In Section 3.2, we present a systematic study of approaches based on nearest-neighbor matching of CNN features for instance retrieval from CAD models. We show that, despite the appearance gap from rendered views and real images, it is possible to use off-the-shelf pre-trained neural network models to perform instance retrieval.

3. Finally, in Section 3.3, we propose a simple extension of the technique presented in Section 3.2 for the case where the query consists of a set of images of the same object, instead of a single image. By leveraging several images from different viewpoints, the retrieval can be made more robust as it can automatically correct ambiguities present in the single image case. Applications include image-based rendering.

3.1 Optimizing memory use in CNNs

In this section, we present our approach to automatically reduce the memory requirements of neural networks on Torch7 [24].

3.1.1 Overview

Most of current deep learning frameworks allocate the memory required for computing a prediction either during network initialization or during the computation of the first prediction. For modularity, a network is expressed as a sequence of individual modules (or layers), and each layer holds all necessary buffers or network states. Examples of such buffers or network states includes the output of the layer or the gradients with respect to the inputs, as well as any storage required for intermediate computations.

Having all the intermediate buffers already pre-allocated allows faster execution, as it avoids expensive memory allocations and deallocations, which are specially costly in the GPUs because they enforce synchronization points. But pre-allocating all the necessary buffers for each module comes with a price: the amount of memory required grows linearly with the depth of the network. This means that deep networks, such as the 152-layer ResNet, requires an enormous amount of memory in most deep learning frameworks, even during inference, where we are interested in the output of the network

after a forward pass. If the tensors were lazily allocated whenever they are needed and freed as soon as they go out of use, meaning that they are not necessary anymore for further computations, memory requirements would be greatly reduced, but runtime performance would suffer on the GPU due to the aforementioned problems.

Related work

There has been a variety of works proposing to reduce the memory requirements of deep learning models. Some of them focus on network pruning, such as [53, 52], and aim at reducing the model size by removing small weights and employing sparse data structures. Such works are orthogonal to the technique we present in this section, as they concern solely the network size, and not the total memory requirements for running the model.

More related to our problem of reducing the memory consumption on deep networks running on the GPU, the cudnn library [21] was proposed, containing primitives for deep learning, such as convolutions and max pooling. It integrates fast convolution routines that does not require internal buffers, such as the *unfolded* input image usually used to perform convolutions as a matrix multiplication.

Concurrently to our work, Rhu *et al.* [92] propose a *virtualized cudnn* approach, which is similar in spirit to what we present in this section. Their approach saves memory by offloading the intermediate buffers to the CPU, allowing for important memory savings, at the cost of some speed penalty. On the contrary, we propose to reuse the intermediate buffers whenever they go out of use, which does not affect runtime speed.

A number of approaches have recently been implemented for reducing memory usage in deep learning frameworks [19, 111], and share a number of similarities with our approach. The main difference with ours is that these approaches directly start with the computation graph representation of the network as input to their memory optimization system, but such computation graph representation is not directly available in Torch7 neural network library [24].

Approach: high-level overview

In order to be able to experiment with deeper models with large batch sizes, we decided to develop a non-intrusive library on top of Torch7 [24] which we call *optnet*, aiming at reducing the memory requirements of using deep networks in an automated manner.

If the buffers were allocated on demand and deallocated whenever they go out of use, the memory requirements would be kept close to minimal. Even though this would require less memory, it would involve expensive memory allocations and deallocations, which are specially expensive when performing GPU computations as they enforce synchronization points, harming parallelization. Instead, we would like to automatically identify whenever a tensor is not used anymore, and instead of freeing it, reusing the same tensor storage on further computations.

Overview of Torch7 neural networks framework

Torch7 neural network package defines the network architecture by means of computation *modules* and a set of *container* modules. A *module* defines 2 operations:

- a *forward* operation, which produces an output given a number of inputs;
- a *backward* operation, which computes the derivatives with respect to the inputs to the module, as well as with respect to its learnable parameters (if any).

Containers are a special type of modules that have no learnable parameters, but which have a number of child modules. Each container specifies how the computation of its child modules are linked together. The simplest example of a container is a *Sequential* container, which connects each of its child modules such that the output of child node i is fed as an input to child node $i + 1$. By mixing different containers, it is possible to construct arbitrarily complex network architectures for which the computation graph can be represented as a Directed Acyclic Graph (DAG).

We emphasize that the network structure is entirely defined during the construction of the network via containers and modules, and it doesn't change during runtime according to the inputs that are fed to it.

Two computation modes are present in Torch7 neural network package: *inference* mode and *training* mode. Those modes are only relevant for modules that behave differently during training or test time, such as dropout or batch normalization. Thus, Torch7 does not treat differently networks that are used for evaluation to networks that are used for training. It allocates all the necessary elements needed to perform training, such as the gradients with respect to the parameters, even if they are not needed as the network is used for evaluation.

An important consideration is that a *module* is completely indifferent to its neighboring *modules*. There is no global reasoning on the whole network level, only on individual *modules* and *containers*.

A note on Tensor implementation In Torch7, *Tensors* are wrappers around *Storages*. A storage is a structure representing a block of contiguous memory. Tensors are implemented as a structure with a number of fields. The most relevant ones for this section are the following:

dimension: number of dimensions of the tensor;

sizes: sizes for each dimension of the tensor;

strides: step in each dimension required to access element $i+1$ from element i. This allows some operations to be performed very fast, as tensor elements do not need to be contiguous in memory. As an example, permuting dimensions in a tensor does not require memory copy and only swapping the strides is needed.

storage: a *Storage* structure, which contains the pointer to the allocated memory;

By decoupling the tensor representation and the storage representation, it is possible for different tensors to share the same storage in a simple manner.

3.1.2 Computation graph construction from containers

In what follows, we restrict ourselves to the inference mode, which is the most relevant for this work. A similar reasoning was also applied for training mode, and is omitted here for brevity.

Defining a computation graph from containers

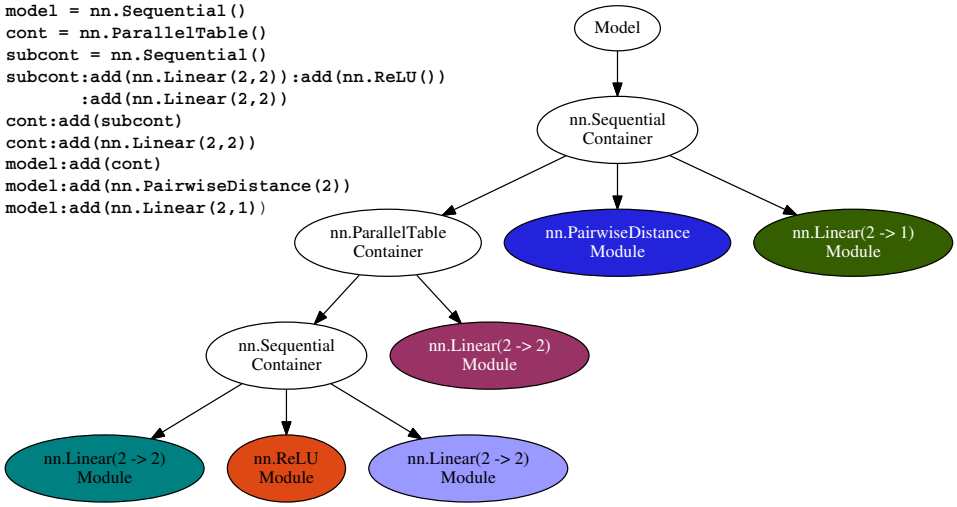
One of the drawbacks of the *Container* representation is that the network structure is implicitly defined. If we need to reason on the whole network level, for example to decide when a given tensor defined in a specific module can be reused, this *container* representation is not adapted. Instead, it would be better to reason using the computation graph defined by the network. The computation graph is a DAG that contains all the modules of the network, and the edges correspond to the data flow from one module to the other. Figure 3.1 illustrates the difference between a representation based on containers and the corresponding computation graph of the network. To compute the output using a container representation, a depth-first traversal of the tree representation defined by the containers is performed, visiting the nodes from left to right.

In order to facilitate the task of reasoning on the network structure, we convert the container representation, which is the default in Torch7, to its corresponding computation graph representation. The lack of constraints imposed by Torch7 neural network library when developing new layers makes a reliable reasoning on the graph structure harder. The only constraints imposed by the Torch7 neural network library when a module is to be implemented are the following:

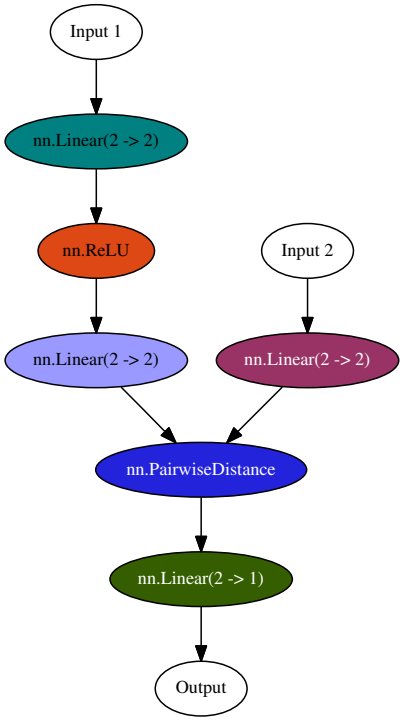
- during forward pass, the result of the computation should be stored internally in the module, and should be the returned argument;
- during backward pass, the forward pass has to be called beforehand with the same input data, and the result of the gradient computation with respect to the input should be stored in the module, and should be returned.

If only these constraints were to be used, specifying how the different elements of the computation graph are linked together would require rewriting dedicated code for each network container and module. As such, this solution would not be sustainable, as every new module added to the main library would require additional changes to the code which generates the computation graph to correctly handle this new module. Instead, we rely on a conceptually simpler, but more reliable and generic approach: to let the computation graph be constructed during a forward propagation of the

```
model = nn.Sequential()
cont = nn.ParallelTable()
subcont = nn.Sequential()
subcont:add(nn.Linear(2,2)):add(nn.ReLU())
       :add(nn.Linear(2,2))
cont:add(subcont)
cont:add(nn.Linear(2,2))
model:add(cont)
model:add(nn.PairwiseDistance(2))
model:add(nn.Linear(2,1))
```



(a) Example of a model representation via containers, with the corresponding code. A *ParallelTable* container passes each element of its input to a different sub-network. The model is evaluated in a depth-first manner, from left to right.



(b) The corresponding computation graph. The same colors were used to represent corresponding modules between the container representation and the computational graph representation.

Figure 3.1: Different model representations.

network. By letting the graph be constructed during the evaluation of the network, we effectively ensure that the computation graph will be representative of the flow of information inside the network, without requiring to implement specific code for each module or container.

In order to implement such a solution, we need to perform standard forward pass computation on the network, but keeping track of the inputs and outputs of each module at each time. The overall idea for constructing the computation graph is the following:

- for each module in the container representation, we keep a list of the input tensors that it uses, as well as the output tensors that it returns. In order to keep this list of input/output tensors, we overwrite the generic forward function such that it stores the input and output tensors of each module. To avoid having to change the function signature and introduce unwanted behaviour, we encapsulate the original function inside another function, that records the inputs and outputs of the module, all the while computing and returning the result. This recording is done via upvalues, which are variables that are accessed by the encapsulating function but whose scope is external to the function;

- every non-container module that performs some computation constitutes a node in the computation graph. This excludes graph-constructor modules like *nn.Identity* or *nn.SelectTable* that only exist because of the container representation, and are needed to guide the flow of information, but that do not perform any operation;

- perform a forward pass over the network, which will populate the list of input/output tensors for each module;

- the edges between modules are given by the list of input/output tensors of each module. We do not add edges linking the input tensor to the output tensor for containers that do not perform any operation except from connecting its child modules (like *nn.Sequential* or *nn.ConcatTable*), as it would add an unwanted edge on the graph. Instead, only *operative* containers such as *nn.Concat* or *nn.Parallel* contribute to new edges in the graph, as they perform some computation. To illustrate this point, we remark that a *nn.Concat* is equivalent to a *nn.ConcatTable* (which only distributes the inputs) followed by a

nn.JoinTable (which concatenates a table of tensors into a tensor in a specified dimension).

The computation graph representation allows for an easier reasoning on the network structure, as well as the dependencies of each node. We will use this representation in the next section to decide when each buffer is not needed anymore.

3.1.3 Selecting reusable buffers

In what follows, we restrict ourselves to the buffers corresponding to the outputs of each module of the network. A similar reasoning can be performed for the gradients with respect to the output of each layer.

Finding the moment in time where every buffer is not used anymore can be performed by applying a liveness analysis algorithm [4] on the computation graph of the network. Such algorithms are usually used by compilers to calculate at which point in a computer program a memory location can potentially still be used in future computations or if it cannot and can thus be freed or reused.

We implemented a liveness analysis based on the implementation from [111], with some additional improvements. The outline of the implemented algorithm is as follows:

1. Define an "analysis" as a data structure that contains two fields per element: the information of the first time a tensor is created, and the last time it is used.

2. Walk over the computation graph in the same order as the execution order. For each node in the graph (which corresponds to a module), identify which are the incoming tensors and the outcoming tensors. Insert in the "analysis" the aforementioned tensors, keeping track of the first time and last time a specific tensor is used.

3. Initialize an empty buffer pool.

4. Sort "analysis" by last time each tensor was used.

5. Iterate over the "analysis" in order.

   (a) For each "analysis" element, check if there is a tensor available in the buffer pool that does not overlap with the living time of the element of the analysis.

   (b) If there are available tensors in the buffer pool, take greedily the one for which the storage is the most similar in size than the storage of the tensor in the current element of the analysis; if there is no available tensor, create a new tensor.

6. Change the original storages of the tensors in the network to use the buffers that were created in the previous step.

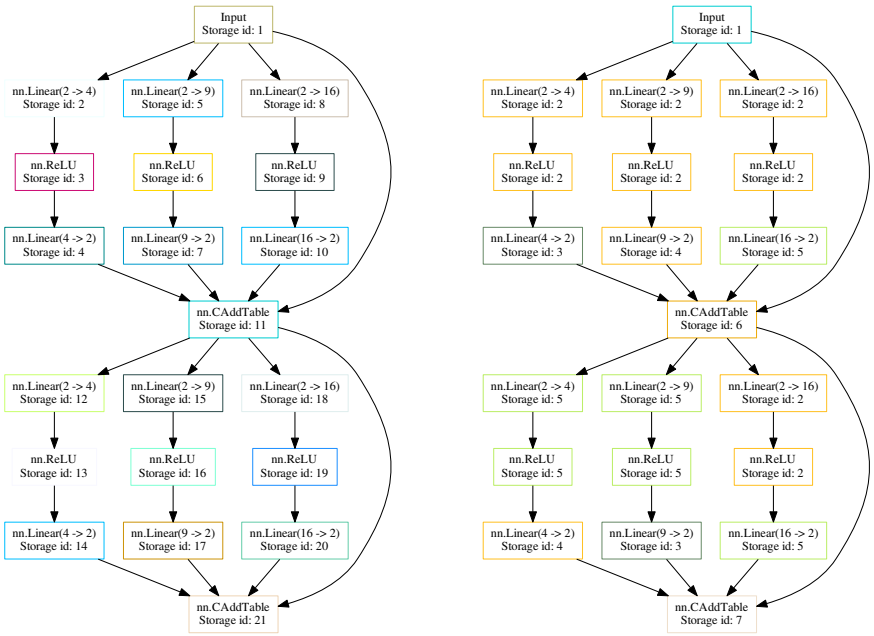
An example of memory optimization for a forward pass on a complex graph can be seen in Figure 3.2. In this example, instead of requiring 21 internal buffers for the outputs, our algorithm optimizes the assignment such that it only requires 7 of them.

Other savings

In addition to the outputs and gradients of the network, other temporary buffers that are specific to each module can be reused. The strategy that we employ is as follows: share any temporary buffers of a module between all instances in the network of the same module type. For a number of commonly used modules, such as convolutions and max-poolings, we keep a list of buffers that can be reused in inference mode as well as in training mode. While suboptimal, this simple strategy already allows for important memory savings.

We also employ another basic strategy for inference mode. Torch7 by default keeps in memory both the parameters of the network and the gradients with respect to the parameters, even when we only want to compute predictions. Thus, during inference mode optimization, we remove the tensors corresponding to the gradients with respect to the parameters of the module, and replace them by some meta-information containing the sizes, strides and storage reference used by the tensor. With this meta-information, we are able to exactly reconstruct the gradients again by creating a new tensor with the same sizes, strides and storage reference[1], which is necessary if

[1]The storage reference is important because different tensors might share the same storage, for example in siamese networks. Note that only the storage reference is used, so that it is possible to identify storages that are reused by different tensors.



(a) Without optimization (b) With optimization

Figure 3.2: Illustration of memory optimization on the forward pass. Same color corresponds to the same storage. For ease of visualization, we show the storage id for every module. Instead of allocating 21 different buffers for the output tensors, our optimization only requires 7 of them.

we want to reuse a previously optimized network for training. This easily saves 50% of the memory required for the parameters compared to standard Torch7 networks, without side effects during evaluation mode.

3.1.4 Results

We applied the optimization schemes presented in the previous sections to several standard CNN architectures. In what follows, we analyse the savings for both inference and training mode.

Inference mode

In inference mode, we are only interested in the output of the network, and not in computing any gradients. For this reason, we can save more memory as we can reuse the output of each module as soon as it gets out of

Network	before (MB)				after (MB)				Relative (%)			
	TOT	OUT	BUF	PAR	TOT	OUT	BUF	PAR	TOT	OUT	BUF	PAR
alexnet	486	3	21	462	236	1	4	231	51	75	80	50
vgg	1431	58	318	1056	666	25	113	528	53	57	64	50
googlenet	253	34	72	146	96	15	8	73	62	56	89	50
resnet18	166	20	57	89	60	8	8	45	64	61	86	50
resnet152	817	173	186	459	251	14	8	229	69	92	96	50

Table 3.1: Summary of optimization results for a batch size of 1 in inference mode, without using cudnn. Description for each column is present in the main text.

scope. Table 3.1 summarizes the savings for different network architectures when the batch size is 1, without using NVidia cudnn library. Each column contains a legend that corresponds to a specific element of the network that is evaluated, which is summarized as follow:

TOT: total memory used by the network;

OUT: memory used by the network for containing the outputs of each module;

BUF: intermediate buffers of each module, such as the *unfolded* image representation used for representing convolutions as matrix multiplications, or the indices containing the maximal elements in *Max Pooling*;

PAR: memory used for the parameters and gradients with respect to the parameters of the network.

Because the outputs can be directly reused in inference mode, the amount of memory required is not proportional to the depth of the network. As such, we can see larger gains for deeper networks. For example, with ResNet-152, we get 92% savings for the total memory used by intermediate outputs, going from 173MB to 14MB for a batch size of 1. We also notice that reusing internal temporary buffers saves up to 96% of memory for ResNet-152.

To facilitate the comparison with next experiments using cudnn, we also present in Table 3.2 the results for a batch size of 1, but this time using cudnn. We note that cudnn does not require temporary internal buffers for any of its modules, bringing already considerable savings for the baseline network. Looking closely, we see that, even though the relative savings for the output buffers can get as high as 92% for ResNet-152, for a batch size of

Network	before (MB)			after (MB)			Relative (%)		
	TOT	OUT	PAR	TOT	OUT	PAR	TOT	OUT	PAR
alexnet	465	3	462	232	1	231	50	75	50
vgg	1114	58	1056	553	25	528	50	57	50
googlenet	180	34	146	88	15	73	51	56	50
resnet18	109	20	89	53	8	45	51	61	50
resnet152	632	173	459	243	14	229	61	92	50

Table 3.2: Summary of optimization results for a batch size of 1 in inference mode, using cudnn. Description for each column is present in the main text.

Network	before (MB)			after (MB)			Relative (%)		
	TOT	OUT	PAR	TOT	OUT	PAR	TOT	OUT	PAR
alexnet	854	392	462	327	96	231	62	75	50
vgg	8428	7372	1056	3664	3136	528	57	57	50
googlenet	4538	4392	146	1993	1920	73	56	56	50
resnet18	2613	2524	89	1025	980	45	61	61	50
resnet152	22583	22124	459	1994	1764	230	91	92	50

Table 3.3: Summary of optimization results for a batch size of 128 in inference mode, using cudnn. Description for each column is present in the main text.

1 the size of the model parameters outweights the size of the output buffers by a considerable margin. We note though that the relative savings for the outputs are constant with respect to the batch size, so bigger batch sizes will benefit more from the savings. This is illustrated in Table 3.3, which in turn presents results when using cudnn with a batch size of 128. We note that the total required memory is dominated by the intermediate buffers holding the outputs of each module, and for the ResNet-152 network, we save up to 91% of the total memory which would usually be required, reducing the requirements from 22.6GB to only 2GB.

Training mode

In training mode, we cannot release the outputs in the same way as during inference mode, because the outputs are required for computing the gradients. Instead, we can reuse the gradients with respect to the outputs of each module during the backward pass. This means that the total amount of memory required after optimization is still dependent on the depth of the network, contrarily to the inference case. Table 3.4 presents the optimization results for a batch size of 128, using cudnn. The labels from each column correspond to the following:

Network	before (MB)		after (MB)		Relative (%)	
	TOT	GRAD	TOT	GRAD	TOT	GRAD
alexnet	1462	608	1182	328	19	46
vgg	15880	7748	11640	3208	27	57
googlenet	11226	6688	8578	4040	24	40
resnet18	5753	3136	4297	1680	25	46
resnet152	51229	28712	31483	8896	39	69

Table 3.4: Summary of optimization results for a batch size of 128 in training mode, using cudnn. Description for each column is present in the main text.

TOT: total memory used by the network;

GRAD: total memory used for the gradients with respect to the inputs of each layer.

We note that, as expected, the relative savings are smaller than in inference mode. Still, we are able to obtain 19% total savings for AlexNet, the shallowest among all tested networks, while for deeper networks such as ResNet-152 we obtain 39% total savings. As before, the relative savings for the gradients with respect to the outputs remains constant, so the absolute savings will be more important for larger batch sizes. Interestingly, one would expect that the relative savings with respect to the gradients in training mode to be similar to the savings of the outputs in inference mode. This is not the case here, where the relative savings are smaller. This is due to the fact that our backward computation graph construction is not optimal, and contains a number of spurious edges that limit optimizations. Improving the backward graph construction requires handling some special cases in a few containers *Containers* and is left for future work.

3.1.5 Conclusion

In this section, we have presented *optnet*, a library for Torch7 that automatically optimizes the memory use for neural networks. Several advantages come with a reduced memory requirement: we can train deeper models that would not normally fit in memory; we can use much larger batch sizes during inference mode, which can translate to faster runtimes on GPUs due to increased parallelism; and we can more easily deploy deep networks on memory-limited devices, such as mobiles. *optnet* was used in a number of experiments presented in this dissertation.

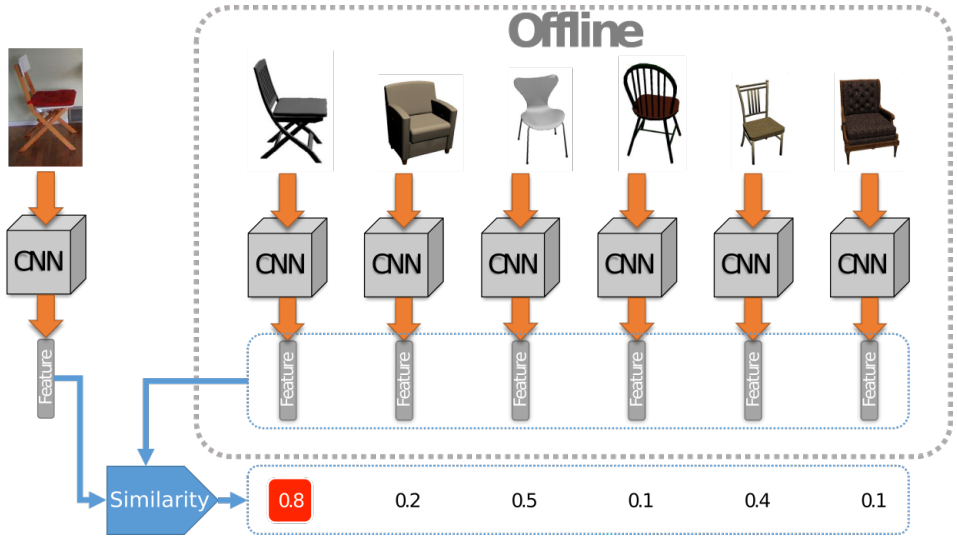


Figure 3.3: CNNs for exemplar-based retrieval. We consider each 3D model as a number of 2D rendered views, and we use a pre-trained CNN to compute a feature representation for each rendered view. For a given query image, we compute its feature representation using the same CNN, and compare it to the features from the rendered views using a similarity metric. The rendered view with the highest similarity corresponds to the retrieved model.

3.2 CNN features for relating 3D objects and images and application to object retrieval

This section describes our study of CNN features for relating 3D objects and images via a retrieval task: finding in a database the most similar 3D model given a real photograph containing a single object centered in the image. We obtain the centered objects by using 2D bounding boxes that indicates the location of the object in the image, and are used to crop the original image in an object-centric manner. Those bounding boxes can be either provided by the user (or dataset), or obtained via an object detector, such as [40]. We suppose we have available a large database of 3D models. We are interested in the task of finding in the 3D database the model which is the most similar to an object in a given image.

We pose the problem of 3D model retrieval as an image matching task. An overview of our system is illustrated in Figure 3.3. Given the difference between 3D meshes and 2D images, we consider each 3D model as a set of 2D rendered views and use them instead. We compute a matching score from the

input image to each rendered view of each 3D model in the database. This matching score can then be used to sort the 3D models by similarity with respect to the input image. The question is then which image representation should be used, and how we should compute the similarity between these representations.

Recently, several works have investigated the use of CNN-based representations for image retrieval. Most of these works treat the activations from certain layers directly as descriptors, either by concatenanting the representations [11, 101] or by pooling them [10]. Contrary our work presented in this section, these works focus on same-domain image retrieval.

In the sections that follows, we present our study of the efficiency of using the output of a fixed pre-trained CNN layer as the feature representation for 3D model retrieval between real and rendered images.

3.2.1 Similarity measure

In this work, we consider three standard metrics for computing the similarities between feature representations: a similarity based on the L_2 distance in Eq. (3.1), the cosine similarity in Eq. (3.2) and the dot-product similarity in Eq. (3.3).

$$S_{L_2}(a,b) = -\left\|a-b\right\|_2^2 \tag{3.1}$$

$$S_{\cos}(a,b) = \cos(a,b) \tag{3.2}$$

$$S_{\text{dot}}(a,b) = a \cdot b \tag{3.3}$$

In Section 3.2.4 we evaluate how the different metrics perform for a number of feature representations based on CNNs.

3.2.2 Feature representation

The visual difference between 3D models and real images is very important, specially when the 3D model does not have texture, realistic lighting or background. To perform instance-based retrieval on such disparate domains, it is crucial to consider a feature representation which is invariant to this

cross-domain variability, but discriminative enough to capture differences in models.

Deep CNN features were shown to perform extremely well on a wide range of tasks, as discussed in Chapter 2. While early layers of the network were shown to extract lower-level information such as edges and textures, deeper layers extract more and more semantic information of the image. If we were interested in only retrieving objects of the same category, it would be natural to consider deeper layers, which contains more semantic information. But retrieving specific object instances requires less invariance to shape deformations, all the while being robust to illumination changes, differences in textures and noise.

We perform a comparative study of different network architectures, for different layers, to identify which is the best suited for such a cross-domain retrieval. The network architectures that we consider are CaffeNet [65], which is very similar to the AlexNet architecture [68], and VGG-16 [103]. Both networks were trained on ILSVRC [98], and we also study if fine-tuning a CaffeNet network for object detection in Pascal VOC improves the quality of the retrievals.

In order to avoid memory issues with earlier layers, which have a high feature dimensionality and are spartially large, and to speed up nearest neighbor matching, we apply a Max Pooling operation with stride of 2 on the features from *conv3* and *conv4*, and we call them *pool3* and *pool4* respectively. This reduces the feature size by a factor of 4, and allows our experiments to be performed with less resources.

3.2.3 Aspect ratio filtering

To extract CNN features from images, the images are first resized to 224x224 pixels, such that they have the same resolution as the images used to train the original CNNs. For this reason, the original aspect ratio of the image is lost before it reaches the network. Given that the query objects are centered in the image, we can enforce an additional constraint that the aspect ratio of the retrieved 2D rendered view should be similar to the aspect ratio of the query image.

Let $AR(x) = \frac{\text{height}(x)}{\text{width}(x)}$ be the function that computes the aspect ratio of image x. For a pair of images q and r, and a tolerance parameter τ, we

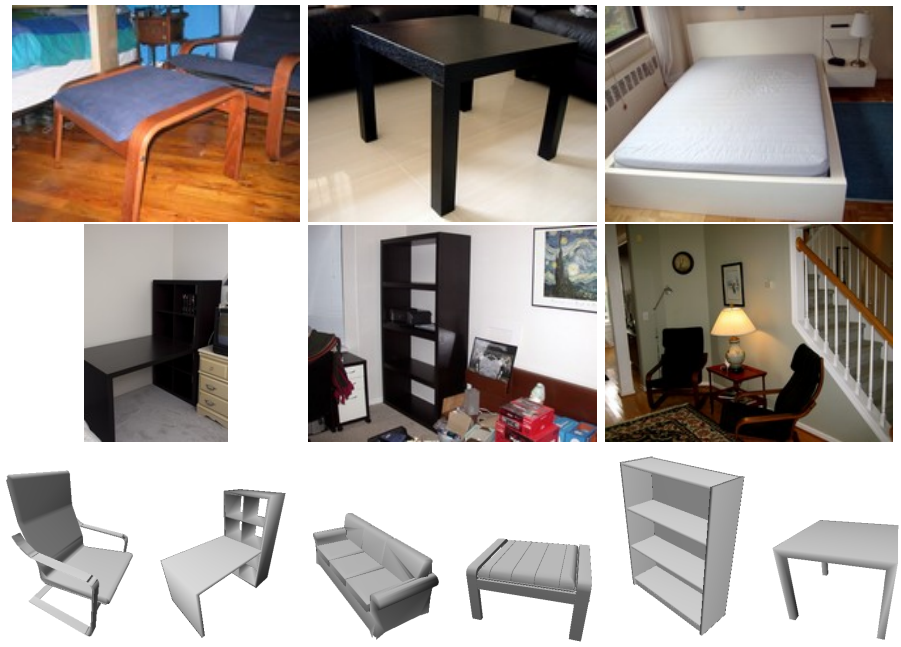


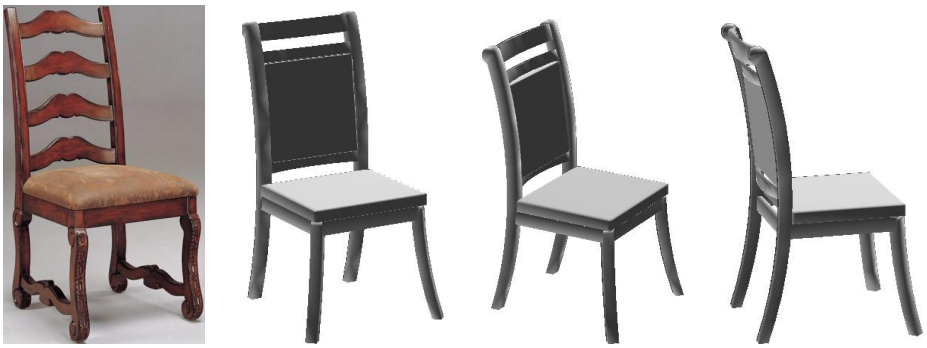
Figure 3.4: Illustrative examples from the IKEAobject dataset. In the first two rows, we show original images from the dataset, before cropping in an object-centric manner single. In the bottom row, we show rendered views of the 3D models available with the dataset.

define the aspect ratio compatibility condition $c_\tau(q, r)$ as follows:

$$c_\tau(q, r) = \tau < \frac{AR(q)}{AR(r)} < \frac{1}{\tau}. \tag{3.4}$$

The aspect ratio compatibility condition is true whenever both the ratio of the aspect ratios and its inverse are greater than the tolerance τ, indicating that both aspect ratios are similar, up to a factor of τ. During retrieval, by keeping only the rendered views for which the aspect ratio compatibility condition is true, we greatly reduce the number of false positives. Furthermore, the computational efficiency of the whole system can be improved if we restrict the computation of the similarity function only for pairs of images that are considered compatible.

One drawback of this method is that it is not robust to truncations of the query image, and could possibly reject good matches. This is not a problem in the current setting, as we assume that the query object is fully contained in the image.



(a) reference (b) pool5 (c) fc6 (d) fc7

Figure 3.5: Top nearest neighbor retrieval using features from different layers from the CaffeNet network fine-tuned for detection using R-CNN framework, using cosine similarity and the same fixed model. Retrieval results from convolutional features provides better orientation than features from fully-connected layers.

3.2.4 Results

In this section, we discuss the results obtained by the method proposed in Section 3.2 for 3D model retrieval from real images via rendered views. We consider the IKEAobject dataset of Lim *et al.* [76], which has textureless CAD models of IKEA object instances manually aligned to their location in images depicting cluttered scenes. The retrieval task is difficult as there is a variety of object poses and perspective effects in the IKEAobject dataset. To handle the variation in object pose and perspective effects, we rendered 36 azimuth and 7 elevation angles at 3 different distances for each object. Note that the rendered views cover many possible viewpoints and perspective effects, but it does not cover all cases. To reduce the bias due to the lack of color and texture in the 3D models, we used grayscale images both for the query images as well as for the 3D renders. Examples of images and 3D models from the IKEAobject dataset can be found in Figure 3.4. There is a single object present in each cropped image used for the retrieval.

We performed retrieval using features extracted from *pool3, pool4, pool5, fc6* and *fc7* layers of 3 different networks: CaffeNet, CaffeNet fine-tuned for detection, and the corresponding layers of VGG-16. All features were extracted after a ReLU layer. In all our experiments, the tolerance parameter τ for the aspect ratio compatibility function is set to $\tau = 0.9$. Table 3.5 presents the instance retrieval accuracy results on IKEAobject dataset. The

CaffeNet

	Dimensionality	L_2 similarity	Dot product	Cosine similarity
Pool3	13824	57.7	46.0	61.3
Pool4	13824	57.7	47.5	65.0
Pool5	9216	38.7	54.7	60.6
fc6	4096	38.7	59.9	59.9
fc7	4096	48.2	61.3	52.6

CaffeNet fine-tuned for detection using R-CNN

	Dimensionality	L_2 similarity	Dot product	Cosine similarity
Pool3	13824	53.3	46.7	62.0
Pool4	13824	56.9	48.2	65.0
Pool5	9216	37.2	52.6	58.4
fc6	4096	43.1	42.3	49.6
fc7	4096	50.4	48.9	48.2

VGG-16

	Dimensionality	L_2 similarity	Dot product	Cosine similarity
Pool3	12544	43.1	34.3	46.7
Pool4	25088	48.2	44.5	**69.3**
Pool5	25088	29.9	56.2	60.6
fc6	4096	40.2	56.9	58.4
fc7	4096	50.4	51.8	56.9

Table 3.5: Instance retrieval accuracy on the IKEAobject dataset [76] for different networks, different layers (rows) and different similarity measures (cols). In the second column, we indicate the dimensionality of the features extracted in each layer.

accuracy is computed as the percentage of the number of good retrievals divided by the total number of cropped images. Best results are consistently obtained by considering the combination of cosine similarity with the *pool4* features for all networks, and the quality of the retrieval decreases with higher layers using cosine similarity. Moreover, *conv4* features are known to be relatively generic features [3, 125] and make little use of the network knowledge gained on specific objects, such as chairs, sofas, and beds, in ImageNet classification. Interestingly, both L_2 similarity and dot product similarity metric perform poorly with VGG-16 features compared to CaffeNet or CaffeNet fine-tuned for object detection using R-CNN framework. On the other hand, cosine distance is significantly better with VGG-16 features. We also noticed that the retrievals using features from fully-connected

layers have a worse orientation quality compared to convolutional features. To illustrate this observation, Figure 3.5 shows the best match for a fixed 3D model and query image using cosine similarity, for different layers.

3.2.5 Conclusion

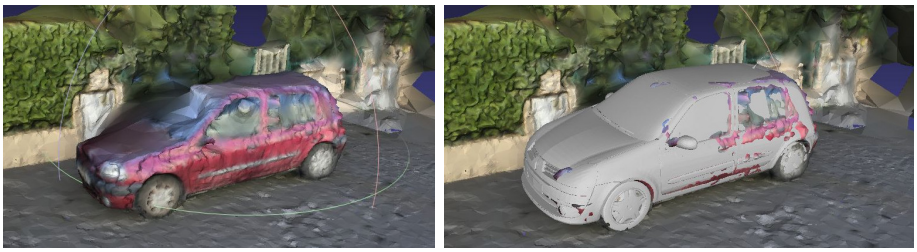
In this section, we have presented a study of the CNN features for relating 3D objects and 2D images. To evaluate this study, we have considered the task of 3D model retrieval in images based on computing a matching score of the image features to the features from rendered views of the 3D models, and we have showed that a cosine similarity function together with *pool4* features works best.

3.3 Multi-view extension for applications to Image-Based Rendering

In this section, we extend the 3D exemplar retrieval technique presented in the previous section to use information coming from several images. This is usually the case in robotic applications, when the system can combine informations from different viewpoints before making a decision, or with videos, where an object is usually visible from slightly different viewpoints in a sequence of frames, or in Image-Based Rendering.

Image-Based Rendering (IBR) is an approach for free-viewpoint navigation in captured environments, which may rely on 3D reconstruction from 2D images. Traditional approaches for IBR do not work well in the case of transparent surfaces or reflective objects. A typical example of commonly found objects on outdoor scenes that fall into this case are cars.

For the car class, a large number of 3D models from different brands and types are available in 3D repositories such as ShapeNet [17]. By manually aligning a 3D model to a 3D scene reconstructed from a number of photographs from the same scene, it is possible to improve the quality of the rendering in highly-specular surfaces, as can be seen in Figure 3.6. In this section, we apply the technique introduced in Section 3.2, which automatically retrieves and aligns a 3D model to a single photograph, in order to improve the quality of the rendering. To this end, we extend the method from Section 3.2 to use the information coming from several photographs to



(a) (b)

(c)

(d)

Figure 3.6: Benefits of using 3D models for image-based rendering (IBR). By aligning a 3D model from a car to the scene, the quality of the rendering can be greatly improved. (a) initial scene reconstruction from multiple images, used as a guide for IBR, (b) 3D model of a car aligned to the reconstructed scene, (c) rendering result of a standard IBR method, (d) rendering result using the aligned 3D model to the scene.

output the single best matching model.

3.3.1 Method

We build our method for the multi-view 3D model retrieval on top of the technique introduced in the previous section. We use *pool4* features from the VGG-16 network with cosine similarity, which was shown to perform best in our retrieval task. We suppose we have N input images, each of which contains different views of the same object centered in the image. As before, we obtain bounding boxes for the objects in the image by applying the object detector from [40] to each image and the detected bounding boxes are used to crop the original images into object-centric images. We suppose that there is only one object in each non-cropped image. This assumption is not a limitation of our method, but it simplifies the task of tracking the same objects in different images, and keeps the notations simpler.

For each cropped image I_n, with $n \in \{1, \dots, N\}$, we compute a matching score $s_n^{m,v}$ for every 3D model $m \in \mathcal{M}$ and viewpoint $v \in \mathcal{V}$ following the technique introduced in Section 3.2. Given those matching scores, we look for a single 3D model which best aligns with all the images.

Let us consider the matching score of each model m for each image n as follows:

$$s_n^m = \max_{v \in \mathcal{V}} s_n^{m,v}. \tag{3.5}$$

The score for a model m in image n is the maximum score over all the possible viewpoints v. We interpret the scores per model s_n^m as the log of the conditional probability $P(m \mid I_n)$ of model m given that the image is I_n, up to a normalization factor. We have:

$$P(m \mid I_n) = \frac{\exp(s_n^m)}{Z(I_n)} \tag{3.6}$$

with $Z(I_n) = \sum_{i \in \mathcal{M}} \exp(s_n^i)$. We suppose that the information given by each image is independent from each other, so we have:

$$P(m \mid I_1, I_2, \dots, I_N) = \prod_{n=1}^{N} P(m \mid I_n) \tag{3.7}$$

We look for the model $\hat{m}$ which has the highest probability given all the

images.

$$\begin{aligned}\hat{m} &= \arg\max_{m\in\mathcal{M}} P(m \mid I_1, I_2, \ldots, I_N) \\ &= \arg\max_{m\in\mathcal{M}} \prod_{n=1}^{N} P(m \mid I_n) \\ &= \arg\max_{m\in\mathcal{M}} \prod_{n=1}^{N} \frac{\exp(s_n^m)}{Z(I_n)} \\ &= \arg\max_{m\in\mathcal{M}} \frac{1}{Z} \prod_{n=1}^{N} \exp(s_n^m) \end{aligned} \tag{3.8}$$

where the normalizing constant $Z = \prod_{n=1}^{N} Z(I_n)$ is independent from m and can be removed from the maximization. By replacing (3.5) in (3.8), we have:

$$\hat{m} = \arg\max_{m\in\mathcal{M}} \sum_{n=1}^{N} \max_{v\in\mathcal{V}} s_n^{m,v}. \tag{3.9}$$

Thus, once we have computed the scores $s_n^{m,v}$ following the technique from Section 3.2, we use equation (3.9) to obtain the model that best aligns with the set of images.

Viewpoint estimation

Once the model $\hat{m}$ is selected, we are interested in obtaining the orientation of the model for each image I_n. For that, we rely solely on the matching scores $s_n^{\hat{m},v}$. We suppose that the viewpoint $\hat{v}_n$ for a given image I_n is the one with maximal score under the constraint that the model is $\hat{m}$, which is given by

$$\hat{v}_n = \arg\max_{v\in\mathcal{V}} s_n^{\hat{m},v}. \tag{3.10}$$

Handling truncations

The retrieval technique from Section 3.2 doesn't handle explicitly the case of truncated objects. In the case where the object present in the image is truncated, the matches with the 2D rendered views will be highly unreliable. For this reason, we only consider the subset of images $\mathcal{I} \subset \{I_1, \ldots, I_N\}$ for which the depicted object is not truncated. Estimating if an object in an image is truncated is a difficult problem by itself. Instead, we leverage the

fact that every query image is actually a cropped version of the original image by using the bounding boxes obtained from the object detector from [40], and we make the following simplifying assumption: whenever the bounding box corresponding to the object touches the boundary of the image, the object is assumed to be truncated. This first approximation for removing truncated instances, even though simple, already allows to filter a number of candidates that could potentially spoil the retrieval results.

3.3.2 Qualitative results

We tested our multi-view extension of the single-view instance retrieval algorithm on a dataset of street images containing cars [18, 85]. This dataset is used for assessing the quality of Image-Based Rendering approaches. Due to the lack of an annotated dataset, we do not present quantitative results and we restrict ourselves to qualitative results.

We obtained the 3D models from ShapeNet database [17]. ShapeNet contains a rich collection of the class "car", which we use to validate our approach. We downloaded 5k car models from this database, and for each 3D model we rendered the object from 108 viewpoints uniformly sampled from the viewing sphere, with azimuth and elevation increasing 10 degrees in the range of $[0, 360)$ and $[0, 30)$ respectively. This constitutes our database of 5k car models, each associated to 108 views of the object.

Some representative results of retrieval with multi-view model consistency can be found in Figure 3.7. As in the previous section, we use grayscale images both for the query images as well as for the 2D rendered views. We can see that the retrievals are generally visually similar to the cars depicted in the query image, up to a color difference. We also observed that the azimuthal orientations are generally correct within 20 degrees.

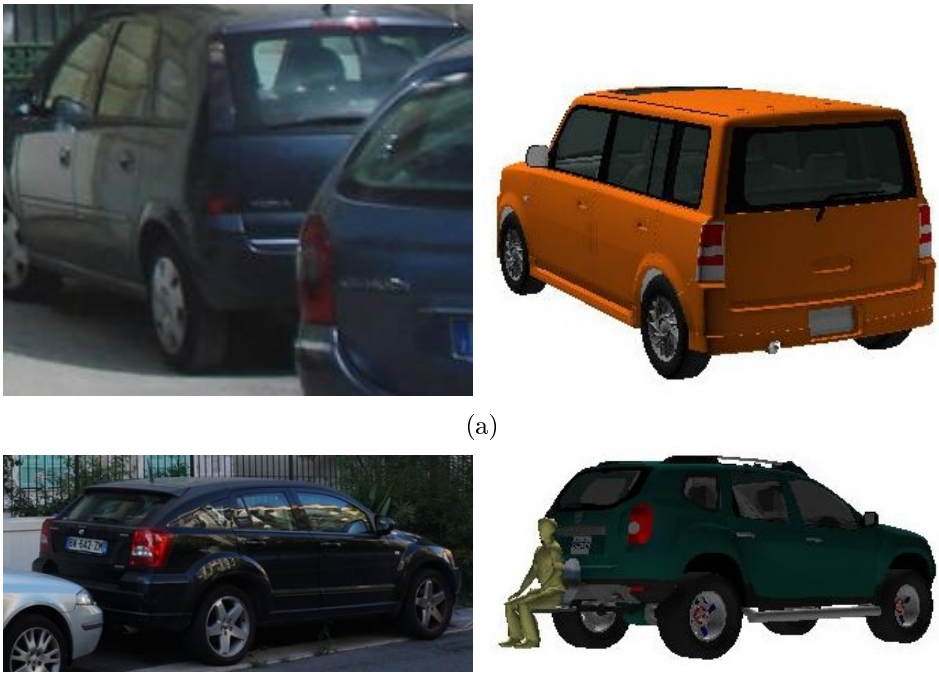
In Figure 3.8, we present qualitative results for both the single-view instance retrieval from Section 3.2 as well as the multi-view extension. The multi-view consistency constraint helps correct ambiguous matches, and usually improves the quality of the retrieved model.

Although our method prunes query images that are potentially truncated, we are still severely affected whenever the query object is occluded. Figure 3.9 illustrates this limitation of our method.

Figure 3.7: Representative multi-view retrieval examples for cars.

(a) (b) (c)

Figure 3.8: Example of the effect of the multi-view consistency constraint, with images coming from the same scene. (a) reference image, (b) without multi-view consistency between models, (c) with multi-view consistency between models.



(a)

(b)

Figure 3.9: Failure cases when occlusions are present.

3.4 Conclusion

In this chapter, we have presented preliminary studies in three subjects:

- We have developed a library on top of Torch7 that automatically reduces the memory requirements of running standard networks. For deep networks, it is able to save up to 91% of the total memory required without optimization. This library was extensively used for the experiments presented in this dissertation, some of which wouldn't be possible to be performed due to lack of available GPU memory.

- We have studied the use of a fixed CNN for the task of 2D-3D instance retrieval. By reasoning in 2D instead of 3D, we were able to use CNNs that were pre-trained on large annotated datasets. We showed that, despite the visual differences between synthetic renders and real images, the CNN features are robust enough such that a retrieval approach based on nearest neighbor matching can be successfully performed.

- We have proposed a simple extension of the 2D-3D instance retrieval method to exploit the information present in multiple query images. As an application of our method, our multi-view instance retrieval extension was combined with image-based rendering techniques in [85] and improved the quality of urban scene rendering results where cars are present.

In Chapter 4, we will extend the 2D-3D instance retrieval presented in this chapter, which tries to answer the question *which object is this?*, to perform instance detection, where we will instead look into answering the question *where is this object?*

Chapter 4

Detection

In Chapter 3, we tackled the problem of retrieving the most similar 3D model from an image containing a single object placed in the center of the image. In this chapter, we present an end-to-end convolutional neural network (CNN) for 2D-3D exemplar detection. We demonstrate that the ability to adapt the features of natural images to better align with those of CAD rendered views is critical to the success of our technique. We show that the adaptation can be learned by compositing rendered views of textured object models on natural images. Our approach can be naturally incorporated into a CNN detection pipeline and extends the accuracy and speed benefits from recent advances in deep learning to 2D-3D exemplar detection. We applied our method to two tasks: instance detection, where we evaluated on the IKEAobject dataset [76], and object category detection, where we outperform Aubry *et al.* [7] for "chair" detection on a subset of the Pascal VOC dataset.

4.1 Introduction

Recently, Aubry *et al.* [7] performed object category detection by exemplar alignment with a large library of 3D object models. The aligned models often approximately matched the style of the depicted objects and allowed 3D information, such as hidden object surfaces and object pose, to be propagated to the 2D images. Such a result is useful for 3D scene reasoning and may potentially be used in applications such as object manipulation in robotics and model-based object image editing in computer graphics [67].

Despite recent progress on 2D-3D matching and retrieval [60, 75, 106], detection by 2D-3D alignment lags behind state-of-the-art object detection systems based on annotated images, e.g., R-CNN [44], in terms of accuracy and speed. We see two primary reasons for this gap in performance: (i) there is a large appearance gap between views rendered from CAD models and real images; and (ii) 2D-based object detection has benefited from re-

cent successes of convolutional neural networks (CNNs) [68, 71]. This work addresses both issues.

The appearance gap across two different domains encountered in 2D-3D alignment is not unique to our problem and can be found in other tasks, e.g., when learning on one dataset and testing on another [110]. To bridge such appearance gaps, a number of cross-domain adaptation algorithms have been developed, some of which are presented in Section 4.1.1. Building on the success of these methods, we present an approach that learns to adapt natural image features for the task of 2D-3D exemplar detection. We hypothesize that, given the features of a natural image depicting an object, it is possible to infer the features of a corresponding rendered view of an object CAD model with similar style and pose. Note that a similar reasoning has been explored in a recent work to predict CAD object features for a different view [108].

To achieve our adaptation learning goal, we need a large training set of pairs made of a natural image and an aligned rendered view depicting a similar object. While there are existing datasets with aligned pairs, e.g., IKEA [76] and Pascal3D [121], such datasets are either relatively small or have aligned models that only coarsely approximates the object style. To overcome these challenges, we make use of the ability to render views from CAD models and composite with natural images, which allows us to create a large training set. The composite image and rendered view pairs form training data with which to learn the feature adaptation, and have been similarly employed in prior work to train 2D object detectors over CAD renders [88, 89] and predict object pose [107].

In learning the adaptation, we adopt a formulation similar to Lenc and Vedaldi [73], which studied the equivariance of image features under geometric deformations of the image. Our work can be seen as an extension of their approach beyond geometric transformations. We show that the adaptation can be incorporated as a module in a CNN-based object detection pipeline. Furthermore, we show that pre-computed features of the rendered views can be added as a fully-connected layer in a CNN, which brings the benefits of accuracy and speed from recent advances in deep learning to 2D-3D exemplar detection.

Contributions. Our contributions are twofold:

- We introduce a cross-domain adaptation approach for 2D-3D exemplar detection using generated pairs of rendered views of CAD models and composite views with natural background. Our adaptation routine adapts features of natural images depicting objects to more closely match features of CAD model rendered views.

- We show how our adaptation routine can be incorporated into a CNN-based detection pipeline, which leads to an increase in accuracy and speed for 2D-3D exemplar detection.

We evaluated our method on the tasks of CAD instance retrieval on the IKEA dataset [76] and on 2D-3D object class detection on the Pascal VOC subset used in Aubry *et al.* [7]. We show state-of-the-art exemplar detection performance on IKEA instances and out-perform the discriminative element approach of Aubry *et al.* [7] both in terms of accuracy and speed. The extended annotations for the IKEA object dataset, a new diverse dataset of textured and non-textured rendered views of CAD models we used to learn the adaptation, and our full code are available at `http://imagine.enpc.fr/~suzano-f/exemplar-cnn/`.

4.1.1 Related Work

A 3D understanding of 2D natural images has been a problem of interest in computer vision since its very beginning [93]. Our work is in line with traditional geometry-centric approaches for object recognition based on alignment [84]. There has been a number of successful approaches for instance-level recognition, e.g., [23, 74, 96], typically based on SIFT matching [79] with geometric constraints. More recent approaches have leveraged contour-based representation to align skylines [9] and statues [5]. Furthermore, simplified or parametric geometric models have been used for category recognition/detection [37, 49, 56, 90, 123, 129]. We will focus our discussion in this section on prior work using CAD models for category recognition and 2D-3D alignment.

Rendered views from CAD models have been used as input for training an object class detector [88, 89, 109] or for viewpoint prediction [107]. Most similar to us are approaches that align models directly to images. Examples include alignment of IKEA furniture models to images [76], exemplar-based object detection [80] by matching discriminative elements [7, 22], and using

hand-crafted features for retrieving CAD models for depth prediction [106] and compositing from multiple models [60]. Also related are approaches for CAD retrieval given RGB-D images (e.g., from Kinect scans) [50, 104]. More recently there has been work to enrich the feature representation for matching and alignment using CNNs, which include CAD retrieval based on CNN responses (e.g., AlexNet [68] "pool5" features) [8], learning a transformation from CNN features to light-field descriptors for 3D shapes [75], and training a Siamese network for style retrieval [12]. Building on efficient CNN-based object class detection, e.g., R-CNN [44], our approach extends the above CNN-based approaches for efficient CAD-exemplar detection.

Bridging two very different image modalities is a classic problem for alignment [63]. Past approaches have addressed this problem using two main strategies. A first line of work has used manually-designed feature detectors and adapted them, for example by adding a mask, so that they focus on the information available in both CAD models and real images [7, 22, 117]. Another line of work has focused on increasing the realism of rendered views, e.g., by extracting likely textures and background from annotated images [88, 89, 107, 109]. Domain adaptation approaches have been formulated for CNNs [14, 57, 94, 39], most recently for object detection [58], fine tuning across tasks [113], and, in a contemporary work, transfer learning from RGB to optical flow and depth [51]. Most similar to our approach is domain adaptation with CAD [109], which adapted hand-crafted features (HOG [25]) for object detection. We formulate a generic domain adaptation approach over image features, which can be applied to hand-crafted features, e.g., HOG [25] or CNN responses.

4.1.2 Overview

Figure 4.1 shows our 2D-3D exemplar detection pipeline. We start by computing CNN features for an image corresponding to a selective search window, along with CNN features for rendered views of CAD models. Due to the large appearance gap across the two domains, we learn how to adapt features of natural images to better match features for rendered views (Section 4.2). We then compare the adapted features with calibrated rendered view features to obtain matching scores for each rendered view (Section 4.3). Note that our detection pipeline can be implemented as a CNN. An evaluation of our approach is in Section 4.4.

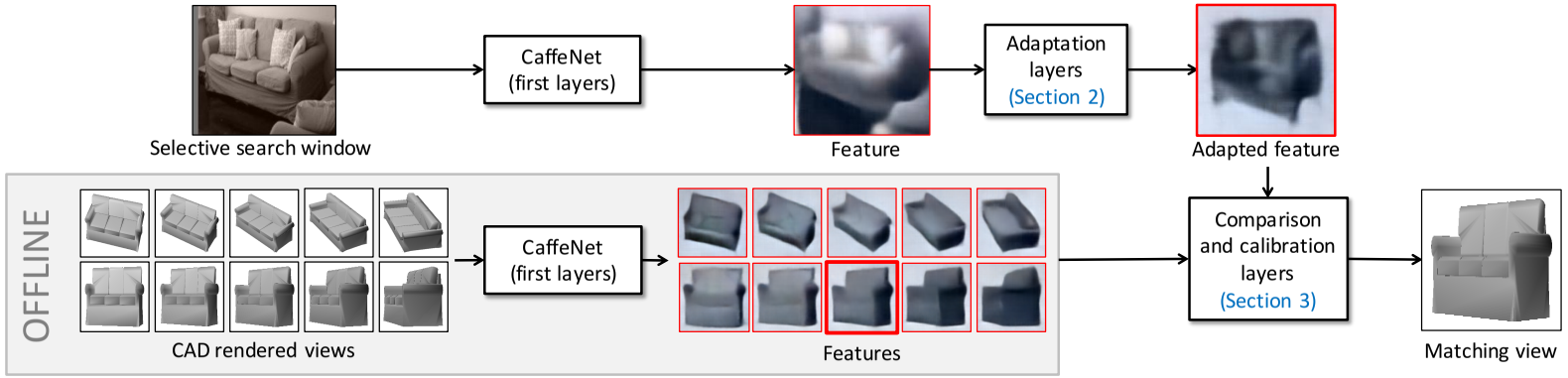


Figure 4.1: **System overview.** Our system takes as input individual 2D image object proposal windows (top-left) generated by the selective search algorithm [114]. The image window is passed through the initial layers of a pre-trained CaffeNet model [65] to generate a feature vector (top-middle). Here, we visualize CNN features using the inversion network of [29] (outlined in red), which infers the original image given a CNN layer's response. In an offline step (bottom-left), we similarly pass rendered views of a library of 3D object CAD models through the initial layers of CaffeNet and record their responses. As there is a domain gap between the appearance of natural images and rendered views of CAD models, we learn to adapt the features for a natural image to better align to those of CAD models (top-right). We compare the features and return the view that best matches the style and pose of the input image (bottom-right).

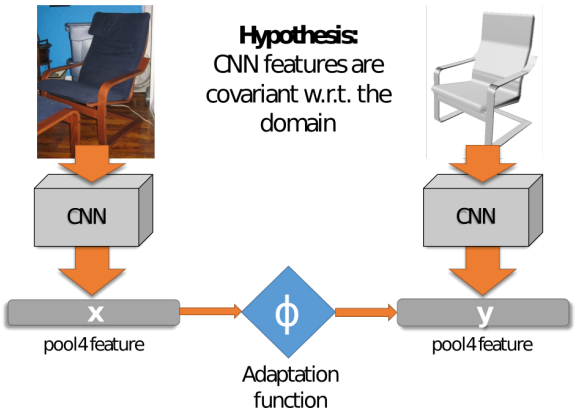


Figure 4.2: Adapting real images to rendered views. A transformation ϕ is learned such that it brings features from the real images closer to the features from the CAD rendered views.

4.2 Adapting from real to rendered views

In this section we describe our approach for adapting features extracted from real images to better correspond to features extracted from rendered views of CAD models. Our approach is general and can be applied to any image feature set, e.g., HOG [25] and CNN-based features [71]. We adapt from real images to rendered views (and not from rendered to real) since it is likely more difficult to hallucinate features corresponding to missing image details, such as the surrounding context of an object and its texture, than to remove them.

Formally, we seek to learn a transformation ϕ over the features of real images. Intuitively ϕ is a projection of the real image feature space to the space of features from CAD rendered views. Ideally, ϕ has the property of mapping a given real image feature depicting an object of interest to features of rendered views of CAD object models with the same geometry, style, and pose. Figure 4.2 illustrates our adaptation system.

Suppose we have as input a set of N pairs of features $\{(x_i, y_i)\}_{i=1}^{N}$ corresponding to examples of real images and rendered views of well-aligned CAD models, respectively. We seek to minimize the following cost over ϕ:

$$L(\phi) = -\sum_{i=1}^{N} S\left(\phi\left(x_i\right), y_i\right) + R(\phi), \tag{4.1}$$

where S denotes a similarity between the two features $\phi(x_i)$ and y_i, and R

is a regularization function over ϕ. Note that in the case where ϕ is an affine transformation, our formulation is similar to the one of Lenc and Vedaldi [73] where a mapping was learned given image pairs to analyze the equivariance of CNN features under geometric transformations.

4.2.1 Adaptation

While the simplest choice for ϕ is an affine transformation, which we use as a reference in our experiments, we also tested more constrained and complex transformations. We focused on transformations that could be formulated as CNN layers, and in particular successions of convolutional and ReLU layers. Note that considering more complex transformations also increases the risk of overfitting. Similar to Lenc and Vedaldi [73] we attempted to constrain the structure of the transformation and its sparsity. This is easily done in a CNN by replacing a fully-connected layer by a convolutional layer with limited support, which implies translation invariance in the adaptation. We found that the best-performing transformation was only a slight modification of the affine transformation:

$$\phi(x) = ReLU(Ax + b), \tag{4.2}$$

where $ReLU(x) = \max(0, x)$ is the element-wise maximum over zero. We observed that applying the $ReLU$ function consistently improved results, and is in agreement with state-of-the-art CNN architecture design choices for object recognition.

4.2.2 Similarity

We tried both L_2 and squared-cosine similarity to measure the similarity in Equation (4.1). We found that the squared-cosine similarity $S(a, b) = -\left(1 - \frac{a^T b}{\|a\|\|b\|}\right)^2$ leads to better results. This is expected, since cosine similarity is known to work better when comparing CNN features [8], but also because we later used the cosine distance to compare real and synthetic features (c.f. Section 4.4). This result is also consistent with the observation of the importance of task-specific similarities in Lenc and Vedaldi [73].

Figure 4.3: Examples of image pairs used for learning the adaptation.

4.2.3 Training data details.

Our adaptation formulation requires a large training set of well-aligned pairs of images and rendered views of CAD models matching the style and pose of depicted objects. Such a dataset is difficult to acquire. While existing datasets have object CAD models aligned to images closely matching the depicted object pose [121, 48], the models are often not similar in style.

Recent work on accurate alignment to 3D models by composition [60] and semi-automatic 3-sweep modeling [20] are promising approaches for obtaining accurate image-model alignments, but no large-scale results are yet available.

Instead, we build on recent approaches for effective training from rendered views [88, 107] to render views of CAD models and composite on natural image backgrounds. This gives us access to virtually unlimited training data. The backgrounds provide "natural-looking" surrounding context and encourages the transformation ϕ to learn to subtract away the background context. To avoid color artifacts in the composite images, we used gray-scale image pairs and also used gray-scale images at test time. Note that contrary to prior approaches using manually-annotated scenes to increase the realism of the composite [88, 89], we do not directly use any object annotation in our background selection process. Figure 4.3 shows four representative image pairs from our adaptation data (top – object rendered views; bottom – rendered views composited with natural image backgrounds).

For the 3D models, we found that using a diverse database comprising several object categories produced better results than focusing on a target

set of 3D models we aim to detect. We used as reference in all our experiments the textureless rendered views from Aubry and Russell [8] to train the adaptation.

4.2.4 Implementation details.

We used a small L_2 regularization R in all our experiments and found that it improved our results despite our very large training sets. We trained ϕ using stochastic gradient descent within the Torch7 framework [24]. We used a weight decay of 5e-4, corresponding to the L_2 regularization, a momentum 0.9, and mini-batch size of 128. We started with a learning rate of 1 and reduced it every 15 epochs by a factor of 10 until convergence.

4.3 Exemplar detection with CNNs

In this section we show how the adaptation procedure in Section 4.2, together with feature computation and exemplar-based retrieval, can be incorporated into an efficient CNN-based detection routine, similar to R-CNN [44], for 2D-3D exemplar detection. For a given input image, we seek to detect the bounding box location of an object in the image and return a corresponding CAD model having similar style, along with the pose of the depicted object.

4.3.1 Exemplar-detection pipeline.

Following the initial part of the R-CNN object detection pipeline [44], we first extract a set of selective search windows [114] and compute CNN responses x at an intermediate layer (e.g., CaffeNet *pool5* layer) for each window. We then apply our adaptation ϕ to these features and compare the results $\phi(x)$ to the features of different CAD model rendered views. Let $s_i(x) = S(\phi(x), y_i)$ be the similarity between $\phi(x)$ and the features y_i of the ith rendered view.

As shown in Aubry *et al.* [7], calibration is an important step for comparing similarity across different views and CAD models. Starting from the initial similarity score $s_i(x)$, we apply their affine calibration routine to compute a new calibrated similarity $s'_i(x) = c_i s_i(x) + d_i$. The scalar parameters c_i and d_i are selected using a large set of random patches such that $s'_i(x_0) = -1$ and $s'_i(x_1) = 0$, where x_0 and x_1 correspond to random patch features with

mean and 99.99-percentile similarity scores, respectively. This calibration leads to an expected false positive rate of 0.01% when $s'_i(x) = 0$.

We take advantage of the fact that in an exemplar-based detection setup the expected aspect ratio of the alignments are known. We remove candidate rendered-view alignments when the aspect ratio has a difference of more than 10% between the selective search window and rendered view. Finally, we rank the remaining alignments by their score $s'_i(x)$ and perform non-maximum suppression to obtain the final detections.

4.3.2 CNN implementation.

Figure 4.1 shows our CNN for 2D-3D exemplar detection. Our network starts with layers corresponding to a CNN trained on a different task (e.g., CaffeNet [65] trained for ImageNet classification in our experiments) until an intermediate layer (e.g., "pool5"). Next, the resulting features pass through the adaptation layers corresponding to ϕ, implemented as a fully-connected layer followed by a ReLU.

The resulting adapted features are compared to the exemplar rendered-view features. Several standard similarity functions, such as dot product and cosine similarity, can be implemented as CNN layers. For example, cosine similarity can be implemented by a feature-normalization layer followed by a fully-connected layer. The weights of the fully-connected layer correspond to a matrix Y of stacked unit-normalized features for the exemplar rendered views, computed in an offline stage. While the affine calibration could be implemented as an independent layer, we incorporated it directly into the fully-connected layer by replacing the matrix rows by $Y_i \leftarrow c_i Y_i$ and adding a bias d_i corresponding to each row i. The final exemplar rendered-view scores is $Y\phi(x) + d$ given image features x, and can be computed by a single forward pass in a CNN.

4.4 Experiments

In this section we qualitatively and quantitatively evaluate our method and analyze different design choices. Based on the results presented in Section 3.2, we use cosine distance over *pool4* features in all our experiments. First, we present our main results on object-instance and object-class detection by aligning to CAD rendered views, comparing against existing base-

lines (Section 4.4.1). Then, we perform an analysis of our algorithm (Section 4.4.2), study the quality of the retrieved pose (Section 4.4.3) and report computational running time (Section 4.4.4).

4.4.1 Detection

In this section, we demonstrate our feature-adaptation algorithm for 2D-3D detection. We consider two tasks: object-instance and object-category detection by 2D-3D alignment. For object-instance detection, we evaluated on the IKEA dataset [76]. For object-category detection, we evaluated on the subset of Pascal VOC containing "chairs" used in Aubry *et al.* [7]. We show qualitative and quantitative results on both benchmarks and compare against prior work.

Object-instance detection by 2D-3D alignment

For object-instance detection by 2D-3D alignment, we evaluated our approach on the IKEA dataset and followed the detection protocol outlined in Lim *et al.* [76]. We report average precision detection performance in Table 4.1(top), along with baselines for this task. It can be seen that we clearly improve over the baselines for several well-represented classes. However, our mAP is smaller than the baselines. We will show that this is due to two main effects: a chance factor for classes where very few objects were annotated or had missing annotations, and a failure of our algorithm on "bookcases", which we analyze in detail.

Dataset and additional annotations. Two important issues when using the IKEA object dataset for evaluating instance detection are (i) its relatively small size (we report the number of annotated instances in the first line of table 4.1), and (ii) the partial annotations made available, with a maximum of one object per image when several are often present. To partly address these issues, we annotated all instances in the 288 test images for the classes that included more than three instances in the original dataset (except for "Billy3", where the detections reported in [76] appear to correspond to a different model). This increases the number of annotated objects of the selected classes from 129 to 223. We report our results on our new extended annotation set in Table 4.1(bottom). With these extended annotations our

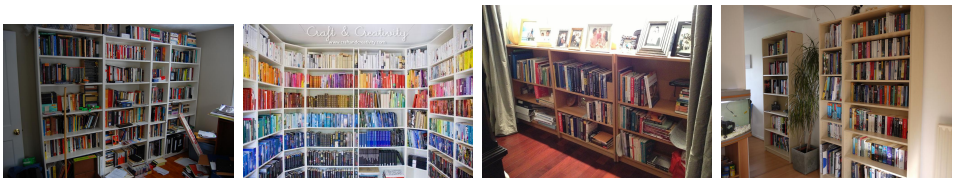


Figure 4.4: Example images of the bookcases missed by our algorithm, most of which are filled with books.

mAP is similar to [76], but with strong differences in the performance for the different objects. We have similar results or clear improvements over [76] (shown in blue in Table 4.1) for most classes, but much lower performance for bookcases (shown in red in Table 4.1).

Failures on bookcases. Here we analyze our failures for bookcases, which are very poor in contrast to other categories where they matched or exceeded the baselines. Inspecting the bookcases missed by our algorithm, some of which are present in Figure 4.4, almost all of them consist of highly cluttered examples, e.g., bookcases filled with books of different colors. We verified that for our extended annotations, only 14% of *billy1* bookcases are empty, whereas *billy2* and *billy4* do not have any non-cluttered examples in the dataset. Looking at our top false positives in Figure 4.5 confirms this, since we find many parts of empty bookcases or bookcases from other categories. The rest of the negatives is explained by the fact that the back- and side-views from the bookcases CAD models, i.e. half of the views we use, have almost no discriminative features, and thus, in the absence of hard negative mining, generate many false positive.

Object-category detection by 2D-3D alignment

For object-category detection by 2D-3D alignment, we evaluated our approach on the subset of the Pascal VOC dataset containing images of non-difficult, non-occluded, and non-truncated "chairs" used in Aubry *et al.* [7], and aligned to their chair rendered views. We followed their detection protocol and report average precision for the detection task. We compare our performance against the baseline of Aubry *et al.* [7], which also performs detection by 2D-3D alignment. We also report performance of DPM [34] and R-CNN [44] with and without SVM, both without bounding box regression,

class	chair *poang*	bookcase *billy1*	sofa *ektorp*	table *lack*	bookcase *billy2*	desk *expedit*	bookcase *billy4*	bed *malm2*	stool *poang*	mAP
Original annotations of [76]										
Number of instances	40	18	13	20	10	8	6	7	7	
Lim *et al.* [76]	27.0	**24.3**	7.3	14.0	**26.6**	18.8	32.6	**22.6**	14.8	**20.89**
DPM [34]	27.5	**24.3**	**12.1**	10.8	13.5	**46.1**	0.1	1.0	0.5	15.10
Ours without adaptation	15.9	1.3	8.4	25.7	0.1	25.5	0.2	0.9	18.3	10.69
Ours with *fc* adaptation	31.1	1.2	9.1	27.0	0.0	13.8	0.4	1.1	23.7	11.94
Ours with *fc+ReLU* adaptation	**33.1**	1.1	9.5	**27.1**	0.1	14.5	0.4	1.2	**24.4**	12.38
New annotations (see Section 4.4.1)										
Number of instances	56	35	15	34	17	7	17	24 *	18	
Lim *et al.* [76]	19.9	**14.8**	**6.4**	9.4	**15.7**	**15.4**	**13.4**	**7.6**	6.4	12.11
Ours with *fc+ReLU* adaptation	**35.4**	6.2	**8.2**	**21.4**	0.1	**16.5**	0.4	**10.4**	**28.4**	**14.11**

Table 4.1: Instance detection performance on the IKEA object dataset [76]. We report average precision using a bounding box overlap threshold of 0.5. Note that some categories reported in [76] have very few annotated examples. We report results for classes that include more than 3 annotated instances. The top part of the table presents results with the original annotation of [76] and the bottom part with our extended annotations. We evaluated the detection outputs provided from [76] using these extended annotations. * The dataset includes three different but similar sizes of the same bed. Since we were not able to differentiate visually between these three kind of beds, all were annotated with the same label.

Figure 4.5: Top 10 detections for the *billy1* IKEA model. Note that the first good detection is counted as negative with the original annotation because it was not annotated in the dataset, but is counted as positive with our extended annotations. Most of our other detections are different bookcases or parts of bookcases.

Training with real data			
DPM [34]	41.0		
R-CNN [44]	44.8		
R-CNN + SVM [44]	**54.5**		
Training with CAD data			
Aubry *et al.* [7]	33.9		
Peng *et al.* [88] (W-UG)	29.6		
	Adaptation	No Adaptation	
		Compos.	White bg.
Logistic *pool4*	12.9	3.7	1.4
Logistic *fc7*	26.6	9.2	14.0
Ours, no calibration	5.6	6.0	3.2
Ours with calibration	**52.3**	36.4	17.9

Table 4.2: Average precision for chair detection on the Pascal VOC subset of non-difficult chairs [7]. Our best method outperforms the baselines of [7] by 18%. "White bg" column corresponds to synthetic images on white background. "Compos" column corresponds to synthetic images composited on real-image backgrounds.

which were trained on natural images for 2D object detection. As another baseline, we report the performance of a logistic regression classifier trained using synthetic images (with and without adaptation), which is similar in spirit to recent approaches that trains a 2D object detector using synthetic training images [88, 89]. In order to better situate our work with respect to approaches that train a classifier using synthetic images with composite backgrounds [88, 89], we also report results for the following baselines using synthetic images composited with natural-image background as positives, and without adaptation: (a) logistic regression classifier, (b) our exemplar detector. Finally, we report results for the best performing method of Peng *et al.* [88], corresponding to their W-UG synthetic images.

We report our results in Table 4.2. With our adaptation, our method outperforms all baselines except R-CNN + SVM. We obtain an average precision of 52.3% compared to 41% for DPM, 33.9% for Aubry *et al.* [7] and 29.6% for Peng *et al.* [88]. Besides, we also tried using the method of [88] with the chairs from [7], which resulted in an average precision of 9.0%. This difference in performance is likely due to their manual selection of realistic viewpoints and models in the W-UG set.

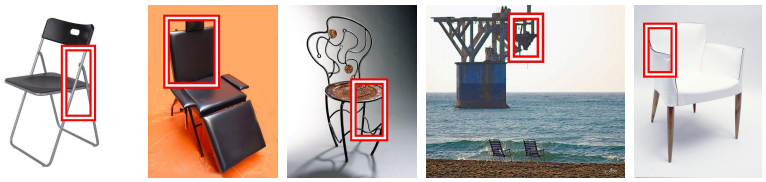
A more detailed analysis reveals the importance of the adaptation for all the methods based only on CNN features from CAD models. Note that the benefit of using the adaptation is less important when using the *fc7* layer for logistic regression. This shows that unsurprisingly *fc7* is less sensitive to the type of representation than *conv4*, and may explain the good results obtained by [88, 89] using the *fc7* layers directly. An interesting question is whether the adaptation could be replaced by synthetic images composited with natural-image backgrounds. As can be seen from Table 4.2, even though the composites help in some cases (notably in our exemplar detector), its performance still lags behind the performance obtained using the adaptation. Note that we used a single background per exemplar view. While one could include more composites per exemplar, this would excessively increase the memory requirements as one would need to store all of the additional exemplars.

4.4.2 Algorithm analysis

In this section we perform a study of different design choices of our approach.

(a) Without adaptation (b) With adaptation

Figure 4.6: Top detections without and with adaptation on the Pascal VOC chair subset [7]. Notice that while the alignments are good with and without adaptation, detection without adaptation returns dark chairs having "CAD-like" white backgrounds. Detections with adaptation include brighter objects and cluttered backgrounds.



(a) Top false positives without adaptation

(b) Top false positives with adaptation

Figure 4.7: Top-ranked false positives without and with adaptation on the Pascal VOC chair subset of [7]. Since there were several false positives per image without adaptation, we only show the best ranked for each image. The false positives without adaptation occur on uniform background patches. With adaptation, this effect largely disappears and the false positives correspond to patches that look like chairs or chair parts.

Influence of adaptation on alignment.

In Figure 4.6, we show the top detections with and without adaptation. Notice that while the non-adapted features have higher detection scores for "CAD-like" images of darker chairs on mostly white background (Fig. 4.6a), the adaptation allows us to detect chairs of all colors in natural cluttered scenes (Fig. 4.6b). Similarly, we show the top false positives in Figure 4.7. Notice that without adaptation the top false positives correspond to regions with uniform background (Fig. 4.7a), while adaptation yields chair-shaped false positives similar to an object detector trained on natural images only (Fig. 4.7b).

Adaptation design.

As discussed in Section 4.3, the adaptation ϕ in Equation (4.2) can be implemented in a CNN as a fully-connected layer, followed by a ReLU nonlinearity. We seek to study variants of ϕ. Since the *pool4* CaffeNet features maintain spatial bin structure, we consider adaptations with limited spatial support via convolution with 1×1 and 3×3 kernels. We also consider whether to use the ReLU nonlinearity or not, and whether to consider multiple convolutional layers in the adaptation.

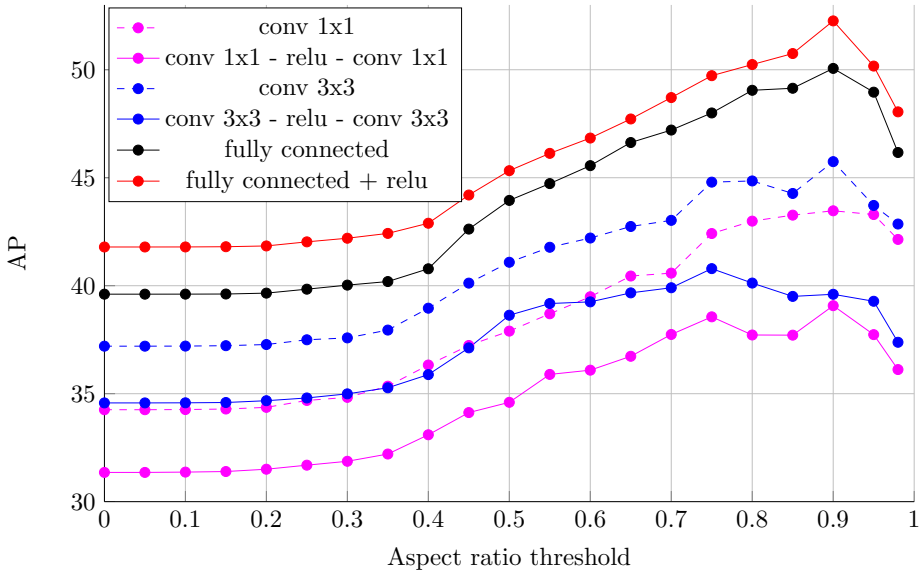


Figure 4.8: Average precision for different adaptation functions ϕ as a function of the aspect ratio threshold.

Figure 4.8 shows the average precision for different variants of ϕ as a function of the aspect ratio threshold. Notice that all of the adaptation variants that we tried performed better than without adaptation (17.9% AP from Table 4.2). Imposing adaptations with limited spatial support (*conv*) performed worse than a fully-connected layer. This can be understood by considering that the effect of the projection depends on the interpretation of the image as foreground object and background as clutter, a task that can be better performed globally. Using two layers for the adaptation degraded performance. Note that we observed the validation loss was better optimized using two layers. We believe this effect is due to the synthetic nature of our training data, which only approximates the relation between real and synthetic images. Finally, we found that adding a ReLU after the convolutional layer consistently increased the performance. The use of a single fully-connected layer followed by a ReLU produced the best performance.

Aspect ratio threshold.

Figure 4.8 shows the evolution of the average precision as a function of aspect ratio threshold for different adaptations on the Pascal VOC subset detection

(a) Number of rendered views						
# rend.	200	500	1k	2k	10k	86k
AP	33.3	37.6	41.3	44.8	45.7	50.0
(b) Number of CAD models						
# CAD	5	10	20	40	160	1393
# rend.	310	620	1240	2480	10k	86k
AP	21.7	26.6	29.8	33.9	44.6	50.0

Table 4.3: Detection AP in the subset of Pascal VOC chair subset [7] for the fully-connected projection as a function of (a) the number of CAD rendered views and (b) the number of unique CAD models used, where we also show the number of rendered views to facilitate the comparison.

experiment. As expected, increasing the threshold first improves the results because it removes many false positives.The results are then relatively stable between 0.75 and 0.9 since both positives and negatives are discarded. Finally, the performance drops for higher thresholds as more true positives get discarded. In all our experiments, we used an aspect-ratio threshold of 0.9.

Number of rendered views.

We studied the relative importance of the CAD model dataset size on the final detection performance by conducting experiments over the set of 86K renders from Aubry *et al.* [7]. We randomly selected increasing subsets of all rendered views (Table 4.3(a)), and randomly selected increasing numbers of CAD models and used all their 62 rendered views (Table 4.3(b)). Notice that performance increases with the number of CAD renders, as expected. Interestingly, the diversity of the CAD models plays an important role in the final detection score. For roughly the same number of rendered views, 5 CAD models (for a total of 310 views) performs considerably worse than 200 random views.

4.4.3 Evaluation of the retrieved pose.

We conducted the same experiment as in Aubry *et al.* [7] to evaluate the quality of the retrieved poses. For the ground truth, we used the pose annotations from Pascal3D [121]. Figure 4.9 shows a histogram of azimuth angle errors at 25% recall (similar to Fig. 6 in Aubry *et al.* [7]). Our algo-

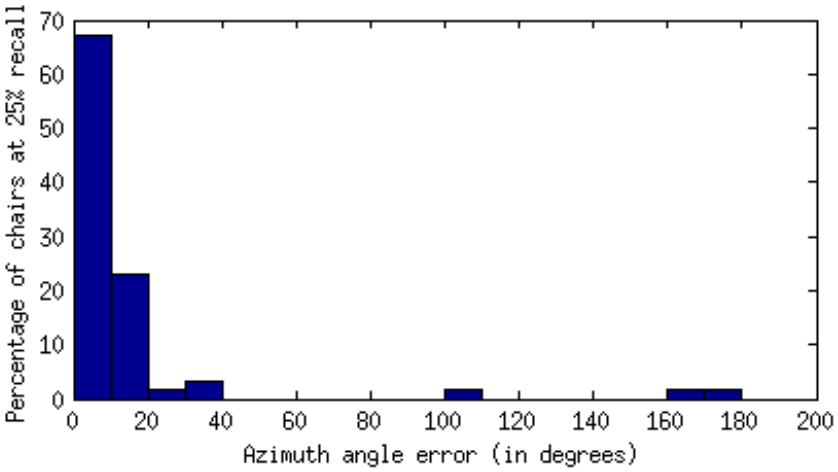


Figure 4.9: Azimuth angle error for correct detections at 25% recall.

rithm returns an azimuth angle within 20° of the ground truth for 90% of the examples, compared with 87% for Aubry *et al.* [7].

4.4.4 Computational run time

Our system runs in computational time similar to R-CNN [44] if all the CAD rendered views fit into GPU memory. Excluding the time to compute bounding box proposals, we can align a test image to 2k rendered views in approximately 9.5 seconds on a GeForce GTX980 graphics card. We can align to more views at the expense of copying pre-computed rendered view features to the GPU memory. This can be overcome with larger-memory graphics cards, by running on multiple cards in parallel or by using *optnet*, presented in Section 3.1, which allows to fit larger amounts of CAD rendered views into GPU memory. For 80k rendered views, our approach currently takes around 52 seconds. Similar to recent fast CNN detection pipelines [54, 43], our timings could be further optimized by reusing the convolutional features for each bounding box, which could potentially reduce the computational time to a fraction of a second. Filtering by aspect ratio before comparing the features could also reduce the number of tests to perform, especially in the case of a very large number of 3D views. Note that even without these improvements, our computational run times are already much faster than those presented in Aubry *et al.* [7].

4.5 Conclusion

We demonstrated an end-to-end CNN for 3D CAD model detection in 2D images. We showed that an adaptation of image features to closely match features of rendered views of CAD models is essential to its success. Our adaptation approach is agnostic to the feature set and could potentially benefit other 2D-3D detection methods.

Chapter 5

Pose Estimation

In this chapter, we present our study of the task of object category viewpoint estimation using Convolutional Neural Networks (CNNs). As discussed in Section 2.4.3, different ways of formulating this problem have been proposed and the competing approaches have been explored with very different design choices. This chapter presents a comparison of these approaches in a unified setting as well as a detailed analysis of the key factors that impact performance. Followingly, we present a new joint training method with the detection task and demonstrate its benefit. We also highlight the superiority of classification approaches over regression approaches, quantify the benefits of deeper architectures and extended training data, and demonstrate that synthetic data is beneficial even when using ImageNet training data. By combining all these elements, we demonstrate an improvement of approximately 5% mAVP over previous state-of-the-art results on the Pascal3D+ dataset [121]. In particular, for their most challenging 24-view classification task, we improve the results from 31.1% to 36.1% mAVP.

5.1 Introduction

Joint object detection and viewpoint estimation is a long-standing problem in computer vision. While it was initially tackled for single objects with known 3D models [93, 78, 61], it was progressively investigated for complete object categories. The interest in this problem has recently increased both by the availability of the Pascal3D+ dataset [121], which provides a standard way to compare algorithms on diverse classes, and by the improved performance of object detection, which encouraged researchers to focus on extracting more complex information from the images than the position of objects.

Convolutional Neural Networks were recently applied successfully to this task of object category pose estimation [107, 112], leading to large improvements of state-of-the-art results on the Pascal3D+ benchmark. However

many elements play an important role in the quality of these results, which have not yet been fully analyzed. In particular, several approaches have been proposed, such as a regression approach with joint training for detection [86, 87], a direct viewpoint classification [112], and a geometric structure aware fine-grained viewpoint classification [107], where the authors modify the classification objective to take into account the uncertainty of the annotations and encode implicitly the topology of the pose space. These papers however differ in a number of other ways, such as the training data or the network architecture they use, making it difficult to compare performances. We explore systematically the essential design choices for a CNN-based approach to pose estimation and we demonstrate that a number of elements influence the performance of the final algorithm in an important way.

Contributions

In this chapter, we study several factors that affect performance for the task of joint object detection and pose estimation with CNNs. Using the best design options, we rationally define an effective method to integrate detection and viewpoint estimation, quantify its benefits, as well as the boost given by deeper networks and more training data, including data from ImageNet and synthetic data. We demonstrate that the combination of all these elements leads to an important improvement over state-of-the-art results on Pascal3D+, from 31.1% to 36.1% AVP in the case of the most challenging 24 viewpoints classification. While several of the elements that we employ have been used in previous work [87, 107, 112], we know of no systematic study of their respective and combined effect, resulting in an absence of clear good practices for viewpoint estimation and sub-optimal performances.

Related work

Most of the related work for this chapter is covered in Chapter 2. Here we review some relevant work for this chapter.

Convolutional Neural Networks. While convolutional neural network have a long history in computer vision (e.g. [71]), their use has been generalized only in 2012 after the demonstration of their benefits by Krizhevsky *et al.* [68] on the ImageNet large-scale visual recognition challenge [26]. Since

then, they have been used to increase performances on many vision tasks.

This has been true in particular for object detection, where the R-CNN technique of Girshick *et al.* [44] provided an important improvement over previous methods on the Pascal VOC dataset [31]. Relying on an independent method to provide bounding box proposals for the objects in the image, R-CNN fine-tunes a network pre-trained on ImageNet to classify these proposal as objects or background. This method has then been improved in several ways, in particular using better network architectures [55], better bounding box proposals [91] and a better sharing of the computations inside an image [54, 43].

Viewpoint estimation. Rigid object viewpoint estimation was first tackled in the case of object instances with known 3D models, together with their detection [93, 78, 61, 5, 74, 76]. These approaches were extended to object categories detection using either extensions of Deformable Part Models (DPM) [34, 46, 56, 90], parametric models [129, 122] or large 3D instances collections [7, 107].

With the advent of Pascal3D+ dataset [121], which extends Pascal VOC dataset [31] by aligning a set of 3D CAD models for 12 rigid object classes, learning-based approaches using only on example images became possible and proved their superior performance. For example, Xiang *et al.* [121] extended the method of [90], which uses an adaptation of DPM with 3D constraints to estimate the pose. CNN-based approaches, which were until the availability of the Pascal3D+ data limited to special cases such as faces [86] and small datasets [87], also began to be applied to this problem at a larger scale. In [81], we explored different pose representations and showed the interest of joint training using AlexNet [68] and Pascal VOC [31] data. Tulsiani and Malik [112] used a simple classification approach with the VGG16 network [103] and annotations for ImageNet objects and established the current state-of-the-art on Pascal3D+. Su *et al.* [107] introduced a discrete but fine-grained formulation of the pose estimation which takes into account the geometry of the pose space, and demonstrate using AlexNet that adding rendered CAD models could improve the results over using Pascal VOC data alone.

5.2 Overview

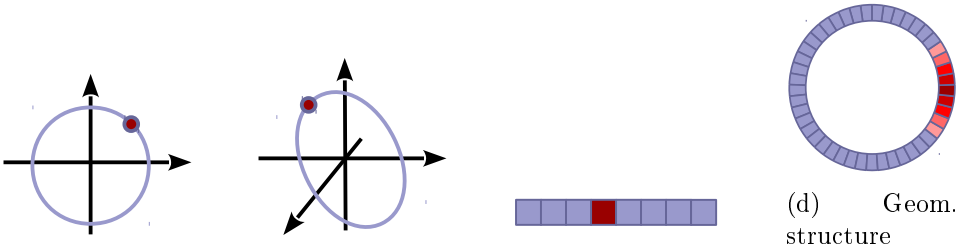
We focus on the problem of detecting and estimating the pose of objects in images, as defined by the Pascal3D+ challenge Average Viewpoint Precision (AVP) metric. In particular, we focus on the estimation of the azimuthal angle. For object detection, we use the standard Fast R-CNN framework [43], which relies on region proposal but is significantly faster than the original R-CNN [44]. In addition, we associate a viewpoint to each bounding box and for each object class. Indeed, since viewpoint conventions may not be coherent for the different classes, we learn a different estimator for each class. However, to avoid having to learn one network per class, we share all but the last layer of the network between the different classes.

In Section 5.3, we first discuss different approaches to viewpoint prediction with CNNs and in particular the differences between regression and classification approaches. Then in Section 5.4, we introduce different ways to integrate the viewpoint estimation and the detection problem. Finally, in Section 5.5 we present the results of the different methods as well as a detailed analysis of different factors that impact performance.

Notations. We call N_s be the number of training samples and N_c the number of object classes. For $i \in \{1, ..., N_s\}$ we associate to the i-th training sample x^i its azimuthal angle $\theta^i \in [0, 2\pi[$, its class $c^i \in \{1, ..., N_c\}$ and the output of the network with parameters $\mathbf{w}$, $f^{\mathbf{w}}(x^i)$. The viewpoints are often discretized and we call N_v the number of bins, and $\tilde{\theta}^i \in \{1, ..., N_v\}$ the bin that includes θ^i. We use subscripts to denote the elements of a tensor; for example, $f^{\mathbf{w}}(x^i)_{k,l}$ is the element (k, l) from tensor $f^{\mathbf{w}}(x^i)$. To make the notation simpler, whenever we mention a loss, we omit the weight decay regularization factor $\mathcal{R}(f^{\mathbf{w}}) = \|\mathbf{w}\|_2^2$ as it will be used in all our equations.

5.3 Approaches for viewpoint estimation

In this section, we assume the bounding box and the class of the objects are known and we focus on the different approaches to estimate their pose. Section 5.3.1 first discusses the design of regression approaches. Section 5.3.2 then presents two variants of classification approaches. The intuition behind these different approaches are visualized on Figure 5.1.



(a) 2D regression (b) 3D regression (c) Direct classification (d) Geom. structure aware classification

Figure 5.1: Different approaches to orientation prediction discussed in this chapter. The target for each approach is visualized in red. For the regression approaches, the possible values of the targets lie on a line. For the classification approaches, the predictions correspond to probability distributions on a discrete set.

5.3.1 Viewpoint estimation as regression

The azimuth angle of a viewpoint being a continuous quantity, it is natural to tackle pose estimation as a regression problem. The choice of the pose representation $F(\theta)$ of an azimuthal angle θ is of course crucial for the effectiveness of this regression. Indeed, if we simply consider $F(\theta) = \theta$, the periodicity of the pose is not taken into account. Thus, as highlighted in [86], a good pose representation $F(\theta)$ satisfies the following properties: (a) it is invariant to the periodicity of the angle θ, and (b) it is analytically invertible.

We explore two representations which satisfy both properties:

(i) $F(\theta) = \Big(\cos(\theta),\ \sin(\theta)\Big)$, probably the simplest way to represent orientations, used for example in [87];

(ii) $F(\theta) = \Big(\cos\left(\theta - \frac{\pi}{3}\right),\ \cos\left(\theta\right),\ \cos\left(\theta + \frac{\pi}{3}\right)\Big)$, a formulation which was presented in [86], and that has a higher dimensionality than the previous one, allowing more flexibility for the network to better capture the pose information.

These representations have different output dimensionality N_d, respectively 2 and 3, and we designate the associated regressions by *regression 2D* and *regression 3D* respectively. Since we treat the regression independently for each class, the outputs $f^{\mathbf{w}}(x)$ of the network that we train for pose estimation have values in $\mathbb{R}^{N_c \times N_d}$ and we designate by $f^{\mathbf{w}}(x)_{c,k}$ the angular element k

of the output for class c.

For training the regression with these representations, we used the Huber loss (also known as Smooth L1) on each component of the pose representation $F(\theta)$. It is known to be more robust to outliers than the Euclidean loss and provides much better results in our experiments. Our regression loss can then be written:

$$L^{\text{reg}}(\mathbf{w}) = \sum_{i=1}^{N_s} \sum_{k=1}^{N_d} H(f^{\mathbf{w}}(x^i)_{c^i,k} - F(\theta^i)_k) \tag{5.1}$$

with H the Huber loss, defined by:

$$H(z) = \begin{cases} 0.5z^2 & \text{if } |z| < 1 \\ |z| - 0.5 & \text{otherwise} \end{cases} \tag{5.2}$$

Given the output $f^{\mathbf{w}}(x)_{c,\bullet}$ of the network for a sample x of class c, we can estimate its pose simply by computing the pose of the closest point on the curve described by F (cf. Figure 5.1). Other regression approaches and loss are discussed in [81] but lead to lower performances.

5.3.2 Viewpoint estimation as classification

As pointed out by [107], the main limitation of a regression approach to viewpoint estimation is that it cannot represent well the ambiguities that may exist between different viewpoints. Indeed, objects such as a table have symmetries or near symmetries that make the viewpoint estimation problem intrinsically ambiguous, and this ambiguity is not well handled by the representations discussed in the previous paragraph. One solution to this problem is to discretize the pose space and predict a probability for each orientation bin, thus formulating the problem as one of classification. Note that a similar difficulty is found in the problem of keypoint prediction, for which the similar solution of predicting a heat map for each keypoint instead of predicting directly its position has proven successful [112].

In the case of a classification approach, the output of the network belongs to $\mathbb{R}^{N_c \times N_v}$ and each value can be interpreted as a log probability. We write $f^{\mathbf{w}}(x)_{c,v}$ the value corresponding to the orientation bin v for an input x of class c.

Direct classification

The approach successfully applied in [112] is to simply predict, for each class independently, the bin in which the orientation of the object falls. This classification problem can be addressed for each object class with the standard cross-entropy loss:

$$L^{\text{classif}}(\mathbf{w}) = -\sum_{i=1}^{N_s} \log\left(\frac{\exp(f^{\mathbf{w}}(x^i)_{c^i,\tilde{\theta}^i})}{\sum_{v=1}^{N_v}\exp(f^{\mathbf{w}}(x^i)_{c^i,v})}\right) \tag{5.3}$$

At test time, the predicted angular bin $\hat{\theta}(x,c)$ for an input x of class c is given by

$$\hat{\theta}(x,c) = \underset{v\in\{1,\ldots,N_v\}}{\arg\max}\, f^{\mathbf{w}}(x)_{c,v} \tag{5.4}$$

Geometric structure aware classification

The drawback of the previous classification approach is that it learns to predict the poses without using explicitly the continuity between close viewpoints. Two neighboring bins have indeed a lot in common. This geometrical information may be especially important for fine-grained orientation prediction, where only few examples per bin are available.

A solution to this problem was proposed in [107]. The authors finely discretize the orientations in $N_v = 360$ bins and consider the angle estimation as a classification problem, but adapt the loss to include a structured relation between neighboring bins and penalize less angle errors that are smaller:

$$L^{\text{geom}}(\mathbf{w}) = -\sum_{i=1}^{N_s}\sum_{v=1}^{N_v} \exp\left(\frac{-d(v,\tilde{\theta}^i)}{\sigma}\right)\log\left(\frac{\exp(f^{\mathbf{w}}(x^i)_{c^i,v})}{\sum_{v=1}^{N_v}\exp(f^{\mathbf{w}}(x^i)_{c^i,v})}\right) \tag{5.5}$$

where $d(v,\tilde{\theta}^i)$ is the distance between the centers of the two bins v and $\tilde{\theta}^i$, and σ is a parameter controlling how much similarity is enforced between neighboring bins. Following [107], we use $\sigma = 3$ for $N_v = 360$. The inference is done as in Equation (5.4).

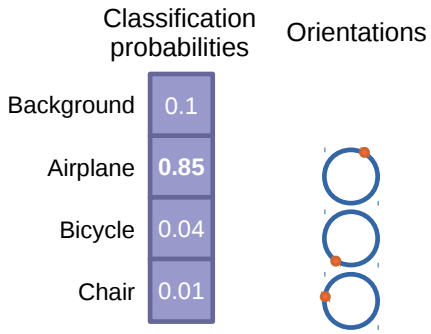


Figure 5.2: **Joint model with regression.** For each category, a continuous viewpoint prediction is performed. The detection and the viewpoint estimation are jointly trained, and the category with the highest detection score determines the orientation.

5.4 Joint detection and pose estimation

The methods presented in the previous section assume that the object detector is already trained and kept independent from the pose estimator. Since object detection and pose estimation relies on related information, we expect a benefit from training them jointly. We thus present extensions of the methods from Section 5.3 to perform this joint training.

5.4.1 Joint model with regression

Two main approaches can be considered to extend the regression approach of Section 5.3.1 to jointly perform detection. The first one, described in [86] is to encode respectively the presence or absence of an object by a point close or far from the regression line described by F in the space where the regression is performed. An alternative approach, discussed in [87] and illustrated in Figure 5.2, is to add an output to the regression network specifically dedicated to detection. The loss used to train the network can then be decomposed into two terms: a classification loss $L^{\text{det}}(\mathbf{w})$, which is independent on the pose, and a regression loss $L^{\text{reg}}(\mathbf{w})$ which takes into account only the pose estimation. Since state-of-the-art performance for detection are obtained using a classification loss, we selected the second option in the following.

Our network thus has two outputs: $f^{\mathbf{w},\text{det}}(x) \in \mathbb{R}^{N_c+1}$ for the detection part (predicting probabilities for each of the N_c classes and the background

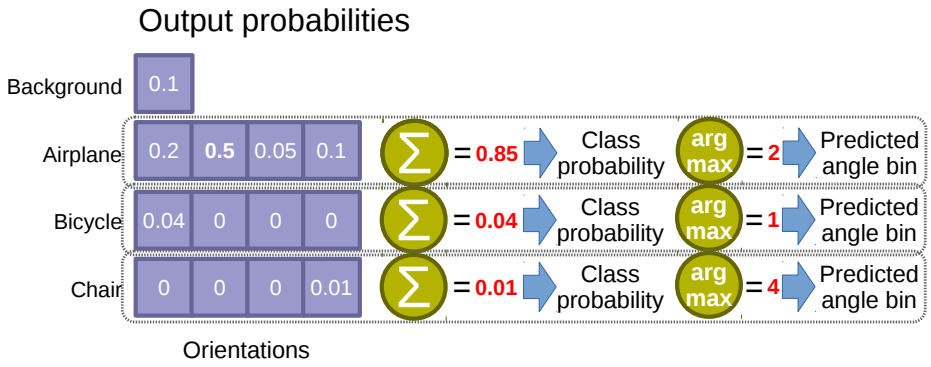


Figure 5.3: **Joint model with classification.** The orientations are discretized into bins, and the detection and viewpoint estimation are jointly trained using cross-entropy loss. During test time, the detection score for a category is the sum of the detection scores for that category over all orientations.

class), and $f^{\mathbf{w},\text{pose}}(x) \in \mathbb{R}^{N_c \times N_d}$ for the pose estimation part. The multi-task loss for joint classification and regression-based pose estimation writes as follows:

$$L^{\text{j-reg}}(\mathbf{w}) = L^{\text{det}}(\mathbf{w}) + \lambda L^{\text{reg}}(\mathbf{w}) \tag{5.6}$$

We define L^{reg} exactly as in Equation (5.1), using the pose estimation output of the network $f^{\mathbf{w},\text{pose}}(x)$. The detection loss L^{det} is the standard cross-entropy loss for detection, using the detection part of the network output $f^{\mathbf{w},\text{det}}(x)$. We set the balancing parameter $\lambda = 1$ in our experiments.

Also, we share the weights of the detection and pose estimation network only up to the *pool5* layer. This is essential to obtain a good performance, as the regression and classification losses are different enough that sharing more weights leads to much worse results.

5.4.2 Joint model with classification

A similar approach, separating two branches of the network, can be applied for classification. However, we introduce a new simpler and parameter-free way to perform jointly detection and pose estimation in a classification setup, which is illustrated in Figure 5.3. Indeed, one can simply add a component, associated to the background patches, to the output vector of the pose estimation setup of Section 5.3.2 and normalize it globally, rather than for each

class independently as in Equation (5.3). Each value is then interpreted as a log probability of the object being of one class and in a given orientation bin, rather than the conditional probability of the object being in a given orientation bin knowing its class. To obtain the probability of the object to belong to one class, one can simply sum the probabilities corresponding to all the bins for this class.

Similar to Section 5.3.2, we write $f^{\mathbf{w},\text{obj}}(x)_{c,v} \in \mathbb{R}^{N_c \times N_v}$ the value of the network output corresponding to the orientation bin v for an input x of class c. We additionally write $f^{\mathbf{w},\text{bg}}(x) \in \mathbb{R}$ its value corresponding to the background and associate a class $c^i = 0$ to the elements x^i in the background.

The loss, which derives from the cross-entropy, writes:

$$\begin{aligned} L^{\text{j-classif}}(\mathbf{w}) = \\ -\sum_{i=1}^{N_s} \mathbb{1}_{c^i=0} \log \left(\frac{\exp(f^{\mathbf{w},\text{bg}}(x^i))}{\exp(f^{\mathbf{w},\text{bg}}(x^i)) + \sum_{c=1}^{N_c} \sum_{v=1}^{N_v} \exp(f^{\mathbf{w},\text{obj}}(x^i)_{c,v})} \right) \\ -\sum_{i=1}^{N_s} \mathbb{1}_{c^i \neq 0} \log \left(\frac{\exp(f^{\mathbf{w},\text{obj}}(x^i)_{c^i,\tilde{\theta}^i})}{\exp(f^{\mathbf{w},\text{bg}}(x^i)) + \sum_{c=1}^{N_c} \sum_{v=1}^{N_v} \exp(f^{\mathbf{w},\text{obj}}(x^i)_{c,v})} \right) \end{aligned} \tag{5.7}$$

At inference, the score associated to the detection of an object x for class c is

$$S(x,c) = \frac{\sum_{v=1}^{N_v} \exp(f^{\mathbf{w},\text{obj}}(x)_{c,v})}{\exp(f^{\mathbf{w},\text{bg}}(x)) + \sum_{c'=1}^{N_c} \sum_{v=1}^{N_v} \exp(f^{\mathbf{w},\text{obj}}(x)_{c',v})} \tag{5.8}$$

5.5 Experiments

We now present experiments comparing the different approaches for pose estimation which were presented in the previous sections. Our experiments are based on the Fast R-CNN object detection framework [43], with Deep Mask [91] bounding boxes proposals.

We trained and evaluated our models using the Pascal3D+ dataset [121], which contains pose annotations for the training and validation images from Pascal VOC 2012 [31] for 12 rigid classes, as well as for a subset of ImageNet [26]. We also extended the training data by adding the synthetic images from [107]. The evaluation metric we used is the *Average Viewpoint Precision* (AVP) associated to Pascal3D, which is very similar to the standard Average Precision (AP) metric used in detection tasks, but which considers

as positive only the detections for which the viewpoint estimate is correct. More precisely, the viewpoints are discretized into K bins and the viewpoint estimate is considered correct if it falls in the same bin as the ground-truth annotation. We focus on the AVP24 metric, which discretizes the orientation into $K = 24$ bins and is the most fine-grained of the Pascal3D+ challenge [121]. We also consider the mean AP (mAP) and mean AVP (mAVP) over all classes.

5.5.1 Training details

We fine-tune our networks, starting from a network trained for ImageNet classification, using Stochastic Gradient Descent with a momentum of 0.9 and a weight decay of 0.0005. We augment all datasets with the horizontally-flipped versions of each image, flipping the target orientations accordingly. During the training of the joint detection and pose estimation models, 25% of the mini-batches consist of positive examples. Our mini-batches are of size 128 except when using synthetic images. When using synthetic images, we randomly create montages with the rendered views from [107], each montage containing 9 objects, for a total mini-batch size of 137 (96 backgrounds and 32 positive patches from real images and 9 positive synthetic objects). This allows for an efficient training in the setup of Fast R-CNN.

We initialize the learning rate at 0.001, and divide it by 10 after convergence of the training error. The number of iterations depends of the amount of training data: when using only Pascal VOC data, we decrease the learning rate after 30K iterations and continue to train until 40K; when adding ImageNet data we decrease the learning rate after 45K iterations and continue to train until 100K; and finally, when adding synthetic data, we decrease the learning rate after 100K iterations and continue to train until 300K.

All experiments were conducted using the Torch7 framework [24] and the full code can be found at `imagine.enpc.fr/~suzano-f/bmvc2016-pose/`.

5.5.2 Results

Comparison of the different approaches for pose estimation

We first compare the different approaches for pose estimation from Section 5.3. We use a fixed object detector based on the AlexNet architecture, trained for detection on Pascal VOC 2012 training set and we report the

Method	mAP	mAVP24
Regression 2D	51.6	13.9
Regression 3D	51.6	15.7
Direct classification	51.6	**19.3**
Geometric structure aware classification	51.6	18.4

Table 5.1: Different approaches for pose estimation with AlexNet architecture, Pascal VOC 2012 data, and using a fixed detector.

results in Table 5.1. We can first observe that for regression, a pose representation with a higher dimensionality (3D) performs better than when using a smaller dimensionality (2D). We believe the redundancy in the representation helps to better handle ambiguities in the estimation. The classification approach however significantly outperforms both regressions (19.3% AVP compared to 13.9% and 15.7%). Interestingly, the simplest classification approach from Section 5.3.2 performs slightly better than the geometry-aware method. We think the main reason for this difference is that the simple classification optimize exactly for the objective evaluated by the AVP, and thus this result can be seen as an artefact of the evaluation. Note that the results could be different for even more fine-grained estimation where less examples per class are available. Nevertheless, since the more complex geometric structure aware approach performed worse than the direct classification baseline, we focus in the rest of this chapter on the simplest direct classification approach.

Benefits of joint training for detection and pose estimation

We evaluate the benefits of jointly training a model to detect the objects and predict their orientation. These benefits can be of two kinds. First, the order of the detections candidates given by the new detector may favor the confident orientations and thus increase the AVP. Second, the pose estimates can be better for a given object. To evaluate both effects independently, we report in Table 5.2 the results using both the order given by the detector used in the previous section and the order given by the new joint classifier. All experiments were performed as above, with the AlexNet architecture and the Pascal VOC training data.

Comparing Table 5.2 to Table 5.1 shows two main effects. First, the mAVP is improved even when using the same classifier, demonstrating im-

Method	Joint detector mAP	Joint detector mAVP24	Independant detector mAP	Independant detector mAVP24
Joint Regression 2D	49.2	15.7	51.6	16.4
Joint Regression 3D	49.6	17.1	51.6	17.4
Joint classification	48.6	**21.1**	51.6	**20.5**

Table 5.2: Jointly training for detection and pose estimation with AlexNet architecture and Pascal VOC 2012 data.

proved viewpoint estimation with joint training. Second, the mAP is decreased, showing that the detection performs worse when trained jointly. However, one can also notice that the best mAVP is still obtained with the joint classifier. This shows that the pose estimation is better in the joint model, and also that for the case of classification the order learned when training jointly the detector favours confident poses. This is not the case for the regression approaches for which the best results are obtained using the independent detector and the jointly-learned pose estimation.

Influence of network architectures and training data

In this section, we consider our joint classification approach, which performs best in the evaluations of the previous section, and study how its performance varies when using different architectures and more training data.

The comparison of the left and right columns of Table 5.3 shows that unsurprisingly the use of the VGG16 network instead of AlexNet consistently improves performances. This improvement is slightly less for the mAVP than for the mAP, hinting that the mAVP boost is mainly due to improved detection performances.

For the training data, we first progressively add training images from ImageNet to the training images from Pascal VOC. The full subset of the ImageNet dataset annotated in Pascal3D+ contains in average approximately 1900 more images per class, but is strongly unbalanced between the different classes. The analysis of these results shows consistent improvements when the training set includes more data. Interestingly, the mAVP is improved more than the mAP, showing that the additional data is more useful for pose estimation than for detection. The addition of synthetic data (2.4M positive examples) improves the results even more, demonstrating that the amount of training data is still a limiting factor even if one uses an AlexNet

Training data	AlexNet mAP	AlexNet mAVP24	VGG16 mAP	VGG16 mAVP24
Pascal VOC2012 train	48.6	21.1	56.9	27.3
+ 250 per class	51.6	25.0	58.0	30.0
+ 500 per class	53.8	26.5	59.0	31.6
+ 1000 per class	53.6	28.3	60.0	32.9
+ full ImageNet	52.8	28.4	59.9	34.4
+ synthetic data	55.9	**31.5**	61.6	**36.1**

Table 5.3: Influence of the amount of training data and network architecture on our joint classification approach.

architecture and includes the ImageNet images, a fact that was not demonstrated in [107]. Note that our joint approach significantly outperforms the state-of-the-art results [112] (currently 31.1% mAVP, based on VGG16 and ImageNet annotations) both without using synthetic data with VGG16, and with synthetic data and AlexNet architecture.

Comparison to the state of the art

Table 5.4 provides the details of the AVP24 performance improvements over all classes as well as a comparison with three baselines: DPM-VOC+VP [90], which uses a modified version of DPM to also predict poses, Render for CNN [107] which uses real images from Pascal VOC as well as CAD renders for training a CNN based on AlexNet, and [112] which uses a VGG16 architecture and ImageNet data to classify orientations for each object category. It can be seen that we improve consistently on all baselines except for the chair class. A more detailed analysis shows that this exception is related to the difference between the ImageNet and Pascal chairs. Indeed, when adding the ImageNet data to the Pascal data, the detection performance for chairs drops from 34.5% AP to 19.23% AP. Similarly, the difference between the very different appearance of the rendered 3D models and real images is responsible for the fact that synthetic training data decreases performance on boats, motorbikes and trains. In average, we still found that synthetic images boost the results by 1.7% mAVP.

Finally, Table 5.5 provides the comparison between our full pipeline and the baselines for the 4, 8 and 16 viewpoint classification tasks, showing that our improvement of the state of the art is consistently high.

5.6 Conclusion

Combining our joint classification approach to the improvements provided by a deep architecture and additional training data, we increase state-of-the-art performance of pose estimation by 5% mAVP. We think that highlighting the different factors of this improvement and setting a new baseline will help and stimulate further work on viewpoint estimation.

Method	aero	bike	boat	bus	car	chair	table	mbike	sofa	train	tv	mAVP24
DPM-VOC+VP [90]	9.7	16.7	2.2	42.1	24.6	4.2	2.1	10.5	4.1	20.7	12.9	13.6
Render For CNN [107]	21.5	22.0	4.1	38.6	25.5	7.4	11.0	24.4	15.0	28.0	19.8	19.8
Viewpoints & Keypoints [112]	37.0	33.4	10.0	54.1	40.0	**17.5**	19.9	34.3	28.9	43.9	22.7	31.1
Classif. approach & AlexNet	21.6	15.4	5.6	41.2	26.4	7.3	9.3	15.3	13.5	32.9	24.3	19.3
+ our joint training	24.4	16.2	4.7	49.2	25.1	7.7	10.3	17.7	14.8	36.6	25.6	21.1
+ VGG16 instead of AlexNet	26.3	29.0	8.2	56.4	36.3	13.9	14.9	27.7	20.2	41.5	26.2	27.3
+ ImageNet data	42.4	37.0	**18.0**	59.6	43.3	7.6	25.1	**39.3**	29.4	**48.1**	28.4	34.4
+ synthetic data	**43.2**	**39.4**	16.8	**61.0**	**44.2**	13.5	**29.4**	37.5	**33.5**	46.6	**32.5**	**36.1**

Table 5.4: Summary of results and comparison with baselines using AVP24.

Method	measure	aero	bike	boat	bus	car	chair	table	mbike	sofa	train	tv	mAVP
[112]	AVP4	63.1	59.4	23.0	69.8	55.2	**25.1**	24.3	61.1	43.8	59.4	55.4	49.1
Ours	AVP4	**70.3**	**67.0**	**36.7**	**75.4**	**58.3**	21.4	**34.5**	**71.5**	**46.0**	**64.3**	**63.4**	**55.4**
[112]	AVP8	57.5	54.8	18.9	59.4	51.5	**24.7**	20.5	59.5	43.7	53.3	45.6	44.5
Ours	AVP8	**66.0**	**62.5**	**31.2**	**68.7**	**55.7**	19.2	**31.9**	**64.0**	**44.7**	**61.8**	**58.0**	**51.3**
[112]	AVP16	46.6	42.0	12.7	64.6	42.7	**20.8**	18.5	38.8	33.5	42.5	32.9	36.0
Ours	AVP16	**51.4**	**43.0**	**23.6**	**68.9**	**46.3**	15.2	**29.3**	**49.4**	**35.6**	**47.0**	**37.3**	**40.6**

Table 5.5: Comparison with state of the art using AVP4, AVP8 and AVP16.

Chapter 6

Discussion

This chapter presents a summary of the contributions presented in this thesis, as well as possible directions for future work.

6.1 Contributions

In this thesis, we explored the possible use of deep CNNs to relate three-dimensional information to photographs of objects. In Chapter 3, we have conducted preliminary studies that were used for the remaining of this dissertation, which can be summarized as follows:

- In Section 3.1, we have presented the core concepts of *optnet*, a library built on top of Torch7 that automatically optimizes the memory usage in neural networks. By exploiting the optimizations implemented by *optnet*, it is possible to use deeper architectures in limited resources environments. *optnet* was extensively used in the experiments presented in this dissertation, some of which wouldn't be possible to be performed due to lack of GPU memory.

- In Section 3.2, we have presented a study of the efficiency of off-the-shelf pre-trained CNNs for the task of 3D model retrieval from real photographs. We have showed that the CNN features are robust enough so that it is possible to use them to perform 3D model retrieval, despite the large appearance gap between real images and rendered views.

- In Section 3.3, we have proposed a multi-view extension of the approach presented in Section 3.2, which exploits the information of an ensemble of query images to retrieve the best matching 3D model. We have applied this technique in the pipeline of an image-based rendering algorithm, considerably improving the quality of the rendering on reflective surfaces of cars.

These studies were the basis for the two main contributions of this thesis which can be summarized as follows:

- In Chapter 4, we have presented our 2D-3D exemplar-based detection, which uses 3D models and CNNs to perform instance detection on real images. Without using any real annotated image, our technique outperforms previous approaches based on exemplar detection, and performs almost on par with R-CNN technique when evaluated on the subset of Pascal VOC 2012 validation containing only non-occluded and non-truncated instances of chairs.

- In Chapter 5, we have studied different ways of formulating the viewpoint estimation problem using a CNN architecture. We have showed that a formulation that jointly optimizes over the detection and the viewpoint estimation is beneficial, and that an approach that discretizes the orientations performs best. With the combination of synthetic renders and our formulation for viewpoint estimation, we have improved over the previous state-of-art on Pascal3D+ dataset [121] by 5 mAVP over all the viewpoint metrics, setting a new baseline for the viewpoint estimation task.

6.2 Perspectives

6.2.1 Improving memory optimization in training mode

In Section 3.1, we have presented a library that automatically optimizes the memory use in Torch7 neural networks. Even though the initial goal was to use it in inference mode, it has found a lot of interest in the Torch7 community thanks to the training mode as well. Our current backward computation graph creation is sub-optimal, as it does not remove a number of spurious edges that appear due to the container representation. Properly handling the computation graph would allow for better optimizations, allowing to fit larger models in memory.

6.2.2 Object compositing

In Section 3.2, we have presented a method that can retrieve a 3D model similar to a given picture by matching the object to many 3D models and

selecting the most confident one. When the number of models increases, this requires comparing a large number of templates to the image. We would like to explore the fact that objects can naturally be decomposed into parts, and those parts can be shared among different 3D models. By combining parts from different models, we can expect to predict better 3D information from an image, as there might not be a single 3D model that explains the object.

6.2.3 Retrieval with millions of objects

Scaling our 3D model retrieval pipeline to millions of images requires a number of optimizations. One possibility is to study more compact feature representations, which would allow for smaller memory footprint and faster runtime for performing nearest neighbor. Another possibility, orthogonal to the first one, would be to have a more efficient nearest neighbor operation, which does not require to compute the comparison of the query image to all objects of the database.

6.2.4 Additional constraints in multi-view instance retrieval

In Section 3.3, we have presented an approach for multi-view instance retrieval using CNNs. Additional constraints could be exploited to improve the retrieval. For example, by exploiting the calibration (camera pose estimation) between different images, it would be possible to enforce a 3D model viewpoint consistency between images. Another possibility would be to use an out-of-the-box object viewpoint estimator, like the one presented in Chapter 5, to re-weight the matching scores taking the viewpoint probabilities into account. Besides, a current limitation of our method appears when the query object is occluded. Modeling the occlusion inside the method would allow to handle such cases, improving retrieval accuracy in cluttered environments.

6.2.5 Metric learning for domain adaptation

In Chapter 4, we have presented our framework for detecting 3D models in 2D images. The core of our method is based on learning an adaptation that makes the features from the real images similar to the features from the rendered views. Instead of defining by hand a similarity function to compare

features from different domains, we might instead learn how we should compare the images from different domains. From our experience when designing the adaptation, we saw that adding more layers to the adaptation reduced the detection accuracy, probably because the training data that we used for learning the adaptation is only an approximation of the task we want to address. In order to learn more complex representations, instead of using a synthetic dataset for learning the adaptation, we might need large amounts of pairs of real images and aligned 3D models, which is however difficult to acquire.

Bibliography

[1] Trimble 3D warehouse. https://3dwarehouse.sketchup.com/. Accessed: 2016-11-21. 4

[2] S. Agarwal and D. Roth. Learning a sparse representation for object detection. In *European Conference on Computer Vision (ECCV)*, pages 113–127. Springer, 2002. 31

[3] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. In *European Conference on Computer Vision (ECCV)*, pages 329–344. Springer, 2014. 60

[4] A. V. Aho, R. Sethi, and J. D. Ullman. *Compilers, Principles, Techniques*. Addison Wesley, 1986. 49

[5] R. Arandjelović and A. Zisserman. Smooth object retrieval using a bag of boundaries. In *International Conference on Computer Vision (ICCV)*, 2011. 37, 71, 93

[6] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. Contour detection and hierarchical image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 33(5):898–916, May 2011. 37

[7] M. Aubry, D. Maturana, A. A. Efros, B. C. Russell, and J. Sivic. Seeing 3D chairs: exemplar part-based 2D-3D alignment using a large dataset of CAD models. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 38, 69, 71, 72, 77, 79, 80, 82, 83, 84, 85, 87, 88, 93

[8] M. Aubry and B. C. Russell. Understanding deep features with computer-generated imagery. In *International Conference on Computer Vision (ICCV)*, 2015. 72, 75, 77

[9] G. Baatz, O. Saurer, K. Köser, and M. Pollefeys. Large scale visual geo-localization of images in mountainous terrain. In *European Conference on Computer Vision (ECCV)*, 2012. 71

[10] A. Babenko and V. Lempitsky. Aggregating local deep features for image retrieval. In *International Conference on Computer Vision (ICCV)*, pages 1269–1277, 2015. 56

[11] A. Babenko, A. Slesarev, A. Chigorin, and V. Lempitsky. Neural codes for image retrieval. In *European Conference on Computer Vision (ECCV)*, pages 584–599. Springer, 2014. 56

[12] S. Bell and K. Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (Proceeding of SIGGRAPH)*, 2015. 72

[13] Y. Bengio. Learning deep architectures for AI. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009. 24

[14] Y. Bengio. Deep learning of representations for unsupervised and transfer learning. In *JMLR Workshop on Unsupervised and Transfer Learning*, 2012. 72

[15] C. M. Bishop. *Pattern recognition*. Springer, 2006. 19

[16] J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, (6):679–698, 1986. 37

[17] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: an information-rich 3D model repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University, Princeton University, Toyota Technological Institute at Chicago, 2015. 2, 4, 38, 61, 65

[18] G. Chaurasia, S. Duchene, O. Sorkine-Hornung, and G. Drettakis. Depth synthesis and local warps for plausible image-based navigation. *ACM Transactions on Graphics (TOG)*, 32(3):30, 2013. 65

[19] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 43

[20] T. Chen, Z. Zhu, A. Shamir, S.-M. Hu, and D. Cohen-Or. 3-sweep: extracting editable objects from a single photo. *ACM Transactions on Graphics (TOG)*, 32(6):195, 2013. 76

[21] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 43

[22] C. B. Choy, M. Stark, S. Corbett-Davies, and S. Savarese. Object detection with 2D-3D registration and continuous viewpoint estimation. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 71, 72

[23] O. Chum, J. Philbin, J. Sivic, M. Isard, and A. Zisserman. Total Recall: Automatic query expansion with a generative feature model for object retrieval. In *International Conference on Computer Vision (ICCV)*, 2007. 71

[24] R. Collobert, K. Kavukcuoglu, and C. Farabet. Torch7: A Matlab-like environment for machine learning. In *BigLearn, NIPS Workshop*, 2011. torch.ch. 41, 42, 43, 44, 77, 101

[25] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 31, 72, 74

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 92, 100

[27] P. Dollár, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2006. 37

[28] P. Dollár and C. L. Zitnick. Structured forests for fast edge detection. In *International Conference on Computer Vision (ICCV)*, 2013. 37

[29] A. Dosovitskiy and T. Brox. Inverting visual representations with convolutional networks. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 73

[30] D. Eigen, C. Puhrsch, and R. Fergus. Depth map prediction from a single image using a multi-scale deep network. In *Neural Information Processing Systems (NIPS)*, pages 2366–2374, 2014. 27, 33

[31] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal visual object classes (VOC) challenge. *IJCV*, 88(2):303–338, 2010. 10, 30, 38, 93, 100

[32] M. Everingham, A. Zisserman, C. K. Williams, L. Van Gool, M. Allan, C. M. Bishop, O. Chapelle, N. Dalal, T. Deselaers, G. Dorkó, et al. The 2005 pascal visual object classes challenge. In *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 117–176. Springer, 2006. 32

[33] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding (CVIU)*, 106(1):59–70, 2007. 31

[34] P. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 32(9), 2010. 32, 37, 80, 81, 82, 93

[35] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–8. IEEE, 2008. 30, 32

[36] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 2, pages II–264. IEEE, 2003. 31

[37] S. Fidler, S. Dickinson, and R. Urtasun. 3D object detection and viewpoint estimation with a deformable 3D cuboid model. In *Neural Information Processing Systems (NIPS)*, 2012. 71

[38] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980. 22

[39] Y. Ganin and V. Lempitsky. Unsupervised domain adaptation by backpropagation. In *Proceedings of The 32nd International Conference on Machine Learning*, pages 1180–1189, 2015. 72

[40] S. Gidaris and N. Komodakis. Object detection via a multi-region and semantic segmentation-aware CNN model. In *International Conference on Computer Vision (ICCV)*, pages 1134–1142, 2015. 55, 63, 65

[41] S. Gidaris and N. Komodakis. Attend refine repeat: Active box proposal generation via in-out localization. *British Machine Vision Conference (BMVC)*, 2016. 27, 33

[42] R. Girshick. *From Rigid Templates to Grammars: Object Detection with Structured Models*. PhD thesis, University of Chicago, 2012. 30

[43] R. Girshick. Fast R-CNN. In *International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015. 35, 88, 93, 94, 100

[44] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 33, 34, 69, 72, 77, 80, 82, 88, 93, 94

[45] R. Girshick, F. Iandola, T. Darrell, and J. Malik. Deformable part models are convolutional neural networks. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 437–446, 2015. 33

[46] D. Glasner, M. Galun, S. Alpert, R. Basri, and G. Shakhnarovich. Viewpoint-aware object detection and pose estimation. In *International Conference on Computer Vision (ICCV)*, 2011. 37, 93

[47] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *JMLR W&CP: Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2011)*, Apr. 2011. 26

[48] R. Guo and D. Hoiem. Beyond the line of sight: labeling the underlying surfaces. In *European Conference on Computer Vision (ECCV)*, pages 761–774. Springer, 2012. 76

[49] A. Gupta, A. A. Efros, and M. Hebert. Blocks world revisited: Image understanding using qualitative geometry and mechanics. In *European Conference on Computer Vision (ECCV)*, 2010. 71

[50] S. Gupta, P. A. Arbeláez, R. B. Girshick, and J. Malik. Aligning 3D models to RGB-D images of cluttered scenes. In *CVPR*, 2015. 72

[51] S. Gupta, J. Hoffman, and J. Malik. Cross modal distillation for supervision transfer. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 72

[52] S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015. 43

[53] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Neural Information Processing Systems (NIPS)*, pages 1135–1143, 2015. 43

[54] K. He, X. Zhang, S. Ren, and J. Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *European Conference on Computer Vision (ECCV)*, pages 346–361, 2014. 34, 88, 93

[55] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 28, 30, 93

[56] M. Hejrati and D. Ramanan. Analyzing 3D objects in cluttered images. In *Neural Information Processing Systems (NIPS)*, 2012. 37, 71, 93

[57] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. *NIPS Deep Learning Workshop*, 2014. 72

[58] J. Hoffman, S. Guadarrama, E. Tzeng, R. Hu, J. Donahue, R. Girshick, T. Darrell, and K. Saenko. LSDA: Large scale detection through adaptation. In *Neural Information Processing Systems (NIPS)*, 2014. 72

[59] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural networks*, 4(2):251–257, 1991. 24

[60] Q. Huang, H. Wang, and V. Koltun. Single-view reconstruction via joint analysis of image and shape collections. *ACM Transactions on Graphics (Proceeding of SIGGRAPH)*, 34(4), 2015. 69, 72, 76

[61] D. P. Huttenlocher and S. Ullman. Object recognition using alignment. In *International Conference on Computer Vision (ICCV)*, 1987. 36, 91, 93

[62] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015. 28

[63] M. Irani and P. Anandan. Robust multi-sensor image alignment. In *International Conference on Computer Vision (ICCV)*, 1998. 72

[64] P. Isola, D. Zoran, D. Krishnan, and E. H. Adelson. Crisp boundary detection using pointwise mutual information. In *European Conference on Computer Vision (ECCV)*, 2014. 37

[65] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia*, pages 675–678. ACM, 2014. 57, 73, 78

[66] K. Kawaguchi. Deep learning without poor local minima. *Neural Information Processing Systems (NIPS)*, 2016. 24

[67] N. Kholgade, T. Simon, A. Efros, and Y. Sheikh. 3D object manipulation in a single photograph using stock 3D models. *ACM Transactions on Graphics (TOG)*, 33(4):127, 2014. 69

[68] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 25, 26, 27, 28, 33, 39, 57, 70, 72, 92, 93

[69] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *18th*

International Conference on Machine Learning (ICML), volume 1, pages 282–289. 38

[70] Y. LeCun. Une procedure d'apprentissage pour réseau a seuil asymetrique (a learning scheme for asymmetric threshold networks). *Cognitiva 85, Paris-Est Ed 599-604*, 1985. 22, 23

[71] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. 24, 25, 70, 74, 92

[72] B. Leibe, A. Leonardis, and B. Schiele. Combined object categorization and segmentation with an implicit shape model. In *Workshop on statistical learning in computer vision, ECCV*, 2004. 31

[73] K. Lenc and A. Vedaldi. Understanding image representations by measuring their equivariance and equivalence. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 70, 75

[74] Y. Li, N. Snavely, D. Huttenlocher, and P. Fua. Worldwide pose estimation using 3D point clouds. In *European Conference on Computer Vision (ECCV)*, 2012. 71, 93

[75] Y. Li, H. Su, C. R. Qi, N. Fish, D. Cohen-Or, and L. J. Guibas. Joint embeddings of shapes and images via CNN image purification. *ACM Transactions on Graphics (Proceeding of SIGGRAPH Asia)*, 2015. 69, 72

[76] J. J. Lim, H. Pirsiavash, and A. Torralba. Parsing IKEA objects: Fine pose estimation. In *International Conference on Computer Vision (ICCV)*, 2013. 10, 37, 59, 60, 69, 70, 71, 79, 80, 81, 93

[77] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014. 30

[78] D. Lowe. The viewpoint consistency constraint. *International Journal of Computer Vision (IJCV)*, 1(1):57–72, 1987. 36, 91, 93

[79] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision (IJCV)*, 60(2):91–110, 2004. 31, 71

[80] T. Malisiewicz, A. Gupta, and A. A. Efros. Ensemble of exemplar-svms for object detection and beyond. In *International Conference on Computer Vision (ICCV)*, 2011. 71

[81] F. Massa, M. Aubry, and R. Marlet. Convolutional neural networks for joint object detection and pose estimation: A comparative study. *arXiv preprint arXiv:1412.7190*, 2014. 11, 93, 96

[82] F. Massa, R. Marlet, and M. Aubry. Crafting a multi-task CNN for viewpoint estimation. In *British Machine Vision Conference (BMVC)*, 2016. 10

[83] F. Massa, B. Russell, and M. Aubry. Deep exemplar 2D-3D detection by adapting from real to rendered views. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 10

[84] J. L. Mundy. Object recognition in the geometric era: A retrospective. In *Toward Category-Level Object Recognition, volume 4170 of Lecture Notes in Computer Science*, pages 3–29. Springer, 2006. 36, 71

[85] R. Ortiz-Cayon, A. Djelouah, F. Massa, M. Aubry, and G. Drettakis. Automatic 3d car model alignment for mixed image-based rendering. In *2016 International Conference on 3D Vision (3DV)*, Stanford, United States, Oct. 2016. 10, 65, 68

[86] M. Osadchy, Y. LeCun, and M. L. Miller. Synergistic face detection and pose estimation with energy-based models. *The Journal of Machine Learning Research*, 8:1197–1215, 2007. 39, 92, 93, 95, 98

[87] H. Penedones, R. Collobert, F. Fleuret, and D. Grangier. Improving object classification using pose information. Technical report, Idiap Research Institute, 2011. 39, 92, 93, 95, 98

[88] X. Peng, K. Saenko, B. Sun, and K. Ali. Learning deep object detectors from 3D models. In *International Conference on Computer Vision (ICCV)*, 2015. 70, 71, 72, 76, 82, 83

[89] B. Pepik, R. Benenson, T. Ritschel, and B. Schiele. What is holding back convnets for detection? In *Pattern Recognition*, pages 517–528. Springer, 2015. 70, 71, 72, 76, 83

[90] B. Pepik, M. Stark, P. Gehler, and B. Schiele. Teaching 3D geometry to deformable part models. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 37, 38, 71, 93, 104, 106

[91] P. O. Pinheiro, R. Collobert, and P. Dollar. Learning to segment object candidates. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Neural Information Processing Systems (NIPS)*, pages 1990–1998. Curran Associates, Inc., 2015. 93, 100

[92] M. Rhu, N. Gimelshein, J. Clemons, A. Zulfiqar, and S. W. Keckler. Virtualizing deep neural networks for memory-efficient neural network design. *arXiv preprint arXiv:1602.08124*, 2016. 43

[93] L. Roberts. Machine perception of 3-D solids. In *PhD. Thesis*, 1965. 35, 36, 71, 91, 93

[94] A. Romero, N. Ballas, S. E. Kahou, A. Chassang, C. Gatta, and Y. Bengio. Fitnets: Hints for thin deep nets. *arXiv preprint arXiv:1412.6550*, 2014. 72

[95] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958. 19

[96] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce. 3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints. *International Journal of Computer Vision (IJCV)*, 66(3):231–259, 2006. 71

[97] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985. 22, 23, 24

[98] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and

L. Fei-Fei. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 25, 41, 57

[99] H. Schneiderman and T. Kanade. A statistical method for 3D object detection applied to faces and cars. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages 746–751. IEEE, 2000. 38

[100] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. *preprint arXiv:1312.6229*, 2013. 33, 34

[101] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 806–813, 2014. 56

[102] M. Simonovsky and N. Komodakis. Onionnet: Sharing features in cascaded deep classifiers. In *British Machine Vision Conference (BMVC)*, 2016. 35

[103] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. 28, 39, 57, 93

[104] S. Song and J. Xiao. Sliding shapes for 3d object detection in depth images. In *ECCV*, 2014. 72

[105] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014. 26

[106] H. Su, Q. Huang, N. Mitra, Y. Li, and L. Guibas. Estimating image depth using shape collections. *ACM Transactions on Graphics (Proceeding of SIGGRAPH)*, 33(4), 2014. 69, 72

[107] H. Su, C. Qi, Y. Li, and L. Guibas. Render for CNN: Viewpoint estimation in images using CNNs trained with rendered 3D model views.

In *International Conference on Computer Vision (ICCV)*, 2015. 39, 70, 71, 72, 76, 91, 92, 93, 96, 97, 100, 101, 104, 106

[108] H. Su, F. Wang, E. Yi, and L. J. Guibas. 3D-assisted feature synthesis for novel views of an object. In *International Conference on Computer Vision (ICCV)*, pages 2677–2685, 2015. 70

[109] B. Sun and K. Saenko. From virtual to reality: Fast adaptation of virtual object detectors to real domains. In *British Machine Vision Conference (BMVC)*, 2014. 71, 72

[110] A. Torralba and A. A. Efros. Unbiased look at dataset bias. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 70

[111] A. Tulloch. fb-caffe-exts. https://github.com/facebook/fb-caffe-exts, 2015. 43, 49

[112] S. Tulsiani and J. Malik. Viewpoints and keypoints. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*. 33, 39, 91, 92, 93, 96, 97, 104, 106

[113] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko. Simultaneous deep transfer across domains and tasks. In *International Conference on Computer Vision (ICCV)*, 2015. 72

[114] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *International Journal of Computer Vision (IJCV)*, 2013. 33, 73, 77

[115] R. Vaillant, C. Monrocq, and Y. LeCun. Original approach for the localisation of objects in images. *IEE Proc on Vision, Image, and Signal Processing*, 141(4):245–250, August 1994. 32

[116] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders. Segmentation as selective search for object recognition. In *International Conference on Computer Vision (ICCV)*, pages 1879–1886. IEEE, 2011. 34

[117] D. Vazquez, A. M. Lopez, J. Marin, D. Ponsa, and D. Geronimo. Virtual and real world adaptation for pedestrian detection. *IEEE*

Transactions on Pattern Analysis and Machine Intelligence (PAMI), 36(4):797–809, 2014. 72

[118] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 1, pages I–511. IEEE, 2001. 31

[119] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shape modeling. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2, 38

[120] Y. Xiang, W. Kim, W. Chen, J. Ji, C. Choy, H. Su, R. Mottaghi, L. Guibas, and S. Savarese. Objectnet3d: A large scale database for 3d object recognition. In *European Conference on Computer Vision (ECCV)*, 2016. 39

[121] Y. Xiang, R. Mottaghi, and S. Savarese. Beyond PASCAL: A benchmark for 3D object detection in the wild. In *Winter Conference on Applications of Computer Vision (WACV)*, 2014. 38, 70, 76, 87, 91, 93, 100, 101, 108

[122] Y. Xiang and S. Savarese. Estimating the aspect layout of object categories. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 38, 93

[123] J. Xiao, B. Russell, and A. Torralba. Localizing 3D cuboids in single-view images. In *Neural Information Processing Systems (NIPS)*, 2012. 71

[124] S. Xie and Z. Tu. Holistically-nested edge detection. In *International Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1395–1403, 2015. 27, 33, 37

[125] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Neural Information Processing Systems (NIPS)*, pages 3320–3328. Curran Associates, Inc., 2014. 60

[126] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. In *ICLR*, 2016. 27, 33

[127] S. Zagoruyko, A. Lerer, T.-Y. Lin, P. O. Pinheiro, S. Gross, S. Chintala, and P. Dollár. A multipath network for object detection. In *British Machine Vision Conference (BMVC)*, 2016. 35

[128] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833. Springer, 2014. 27, 29

[129] M. Zia, M. Stark, B. Schiele, and K. Schindler. Detailed 3D representations for object recognition and modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, 2013. 71, 93