Chapter 1
Introduction: The Data Problem

Machine learning has been growing in scale, breadth of applications, and the amounts of required data. This presents an important problem, as the requirements of state-of-the-art machine learning models, especially data-hungry deep neural networks, are pushing the boundaries of what is economically feasible and physically possible. In this introductory chapter, we show and illustrate this phenomenon, discuss several approaches to solving the data problem, introduce the main topic of this book, *synthetic data*, and outline a plan for the rest of the book.

1.1 Are Machine Learning Models Hitting a Wall?

Machine learning is hot, and it has been for quite some time. The field is growing exponentially fast, with new models and new papers appearing every week, if not every day. Since the deep learning revolution, for about a decade, deep learning has been far outpacing other fields of computer science and arguably even science in general. Analysis of the submissions from *arXiv*[1], the most popular preprint repository in mathematics, physics, computer science, and several other fields, shows how deep learning is taking up more and more of the papers. For example, the essay [117] cites statistics that show how:

- the percentage of papers that use deep learning has grown steadily over the last decade within most computer science categories on *arXiv*;
- and moreover, categories that feature high deep learning adoption are growing in popularity compared to other categories.

[1] https://arxiv.org/.

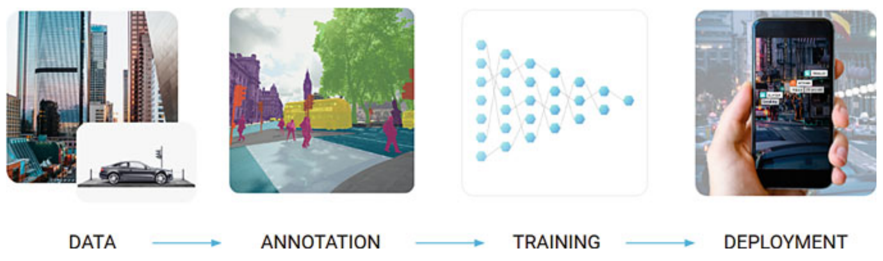

Fig. 1.1 The structure of a machine learning project.

The essay [117] was published in 2018, but the trend continues to this day: the number of papers on deep learning is growing exponentially, each individual subfield of deep learning is getting more and more attention, and all of this does not show any signs of stopping. We are living on the third hype wave of artificial intelligence, and nobody knows when and even if it is going to crash like the first two (we will briefly discuss them in Section 2.1).

Still, despite all of these advances, the basic pipeline of using machine learning for a given problem remains mostly the same, as shown in Figure 1.1:

- first, one has to collect raw *data* related to the specific problem and domain at hand;
- second, the data has to be *labeled* according to the problem setting;
- third, machine learning models *train* on the resulting labeled datasets (and often also *validate* their performance on subsets of the datasets set aside for testing);
- fourth, after one has trained the model, it needs to be *deployed* for inference in the real world; this part often involves deploying the models in low-resource environments or trying to minimize latency.

The vast majority of the thousands of papers published in machine learning deal with the "Training" phase: how can we change the network architecture to squeeze out better results on standard problems or solve completely new ones? Some deal with the "Deployment" phase, aiming to fit the model and run inference on smaller edge devices or monitor model performance in the wild.

Still, any machine learning practitioner will tell you that it is exactly the "Data" and (for some problems especially) "Annotation" phases that take upwards of 80% of any real data science project where standard open datasets are not enough. Will these 80% turn into 99% and become a real bottleneck? Or have they already done so? Let us find out.

For computer vision problems, the labeling required is often very labor-intensive. Suppose that you want to teach a model to count the cows grazing in a field, a natural and potentially lucrative idea for applying deep learning in agriculture. The basic computer vision problem here is either *object detection*, i.e., drawing bounding boxes around cows, or *instance segmentation*, i.e., distinguishing the silhouettes of cows. To train the model, you need a lot of photos with labeling like the one shown in Fig. 1.2 (segmentation on Fig. 1.2a; object detection on Fig. 1.2b).

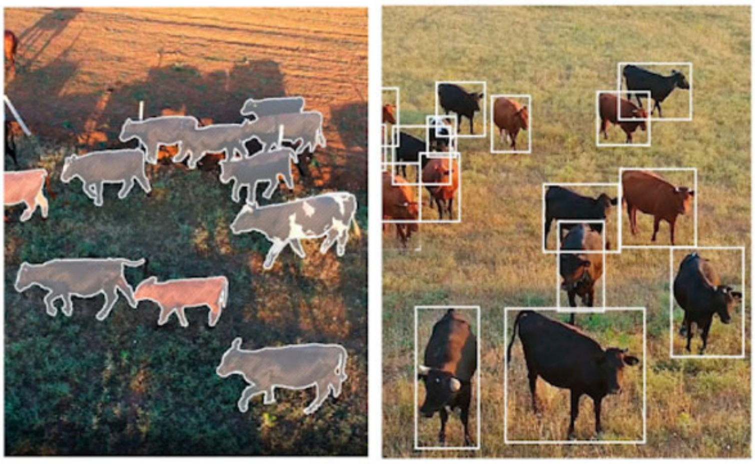

Fig. 1.2 Sample labeling for standard computer vision problems: (a) instance segmentation; (b) object detection.

Imagine how much work it is to label a photo like this by hand. Naturally, people have developed tools to help partially automate the process. For example, a state-of-the-art labeling tool (see, e.g., [829]) will suggest a segmentation candidate produced by some general-purpose model, and the human annotator is only supposed to fix the mistakes of this model by clicking on individual pixels that are segmented incorrectly. But it might still take minutes per photo, and the training set for a standard segmentation or object detection model should have thousands or even tens of thousands of such photos. This adds up to human-years and hundreds of thousands, if not millions, of dollars spent on labeling only.

There exist large open datasets for many different problems, segmentation and object detection included. But as soon as you need something beyond the classes, settings, and conditions that are already well covered in these datasets, you are out of luck; for instance, *ImageNet* does have cows, but not shot from above with a drone.

And even if the dataset appears to be tailor-made for the problem at hand, it may contain dangerous biases. For example, suppose that the basic problem is *face recognition*, a classic computer vision problem with large-scale datasets freely available [37, 38, 366, 542]; there also exist synthetic datasets of people, and we will discuss them in Section 6.6. But if you want to recognize faces "in the wild", you need a dataset that covers all sorts of rotations for the faces, while standard datasets mostly consist of frontal photos. For example, the work [1017] shows that the distribution of face rotations in IJB-A [460], a standard open large-scale face recognition dataset, is extremely unbalanced; the work discusses how to fill in the gaps in this distribution by producing synthetic images of faces from the IJB-A dataset (see Section 10.3, where we discuss this work in detail).

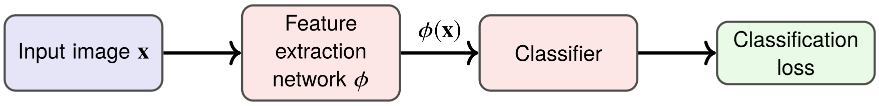

Fig. 1.3 General architecture of a simple classification network.

To sum up: current systems are data-intensive, data is expensive, and we are hitting the ceiling of where we can go with already available or easily collectible datasets, especially with complex labeling. So what's next? How can we solve the data problem? Is machine learning heading towards a brick wall? Hopefully not, but it will definitely take additional effort. Over the next sections, we discuss what can be done to alleviate the data problem. We will give a very high-level overview of several possible approaches currently explored by machine learning researchers and then make an introduction to the main topic of this book: synthetic data.

1.2 One-Shot Learning and Beyond: Less Data for More Classes

We have already mentioned face recognition systems and have just discussed that computer vision systems generally need a lot of labeled training samples to learn how to recognize objects from a new class. But then how are face recognition systems supposed to work at all? The vast majority of face recognition use cases break down if we require hundreds of photos in different contexts taken for every person we need to recognize. How can a face recognition system hope to recognize a new face when it usually has at most a couple of shots for every new person?

The answer is that face recognition systems are a bit different from regular image classifiers. Any machine learning system working with unstructured data (such as photographs) is basically divided into two parts:

- feature extraction, the part that converts an image into a (much smaller) numerical vector, usually called an *embedding* or a *latent code*, and
- a machine learning model (e.g., a classifier) that uses extracted features to actually solve the problem.

So a regular image classifier consists of a feature extraction network followed by a classification model, as shown in Fig. 1.3; this kind of architecture was used by the first neural networks that brought the deep learning revolution to computer vision such as *AlexNet*, *GoogLeNet*, and others (we will discuss them in Section 3.1). In deep learning, the classifier is usually very simple, in most cases basically equivalent to logistic regression, and feature extraction is the interesting part. Actually, most of the advantages of deep learning come from the fact that neural networks can learn to be much better at feature extraction than anything handcrafted that we humans had been able to come up with.

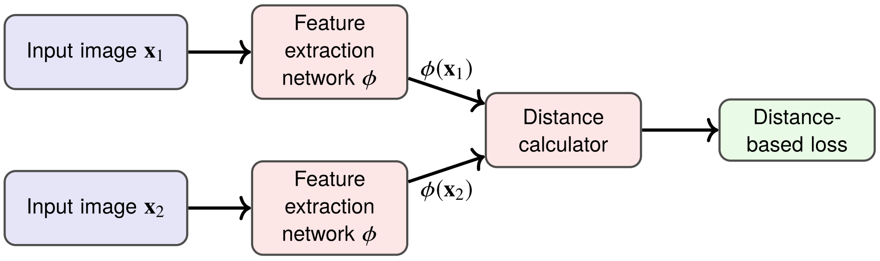
Input image $\mathbf{x}_1$
Feature extraction network ϕ
$\phi(\mathbf{x}_1)$
Distance calculator
Distance-based loss
Input image $\mathbf{x}_2$
Feature extraction network ϕ
$\phi(\mathbf{x}_2)$

Fig. 1.4 General architecture of a Siamese network.

To train this kind of network, you do indeed need a lot of labeled faces, and to add a new face, you need quite a few examples of the new class. However, this is not the only way. An important direction in the development of modern face recognition systems is related to learning face embeddings (learning feature extraction) in various ways. For example, *FaceNet* [771] learns with a modification of the Siamese network approach, where the target is not a class label but rather the distances or similarities between face embeddings. The goal is to learn embeddings in such a way that embeddings of the same face will be close together while embeddings of different faces will be clearly separated, far from each other.

The general architecture of this approach is shown in Fig. 1.4: feature extractors produce numerical features that are treated as vectors in a Euclidean space, and the loss function is designed to, roughly speaking, bring the vectors corresponding to the same person close to each other and push vectors extracted from images of different people apart.

After the *FaceNet* model shown in Fig. 1.4 has been trained with distance-based metrics in mind, we can use the embeddings to do what is called *one-shot learning*. Assuming that the embedding of a new face will have the same basic properties, we can simply compute the embedding for a new person (with just a single photo as input!) and then do classification by looking for nearest neighbors in the space of embeddings. While this approach has met with some problems specifically for face recognition due to complications in the mining of hard negative examples, and some state-of-the-art face recognition systems are currently trained as classifiers with additional tricks and modified loss functions [188, 915, 966], this approach still remains an important part of the research landscape.

This is a simplified but realistic picture of how one-shot learning systems work. But one can go even further: what if there is no data available at all for a new class? This setting is known as *zero-shot learning*. The problem sounds impossible, and it really is: if all you know are images from “Class 1” and “Class 2”, and then you are asked to distinguish between “Class 3” and “Class 4” with no additional information, no amount of machine learning can help you. But in real life, it does not work this way: we usually have some background knowledge about the new classes even if we do not have any images. For example, when we are asked to recognize a

"Yorkshire terrier"[2], we know that it is a kind of dog, and maybe we even have its verbal description that can be lifted, e.g., from *Wikipedia*. With this information, we can try to learn a joint embedding space for both class names and images, and then use the same nearest neighbors approach but look for the nearest label embedding rather than other images (which we do not have for a new class).

This kind of cross-modal zero-shot learning was initiated by Socher et al. [810], who trained a model to recognize objects on images based on knowledge about class labels learned from unsupervised text corpora. Their model learns a joint latent space of word embeddings and image feature vectors. Naturally, this approach will not give the same kind of accuracy as training on a large labeled set of images, but zero-shot learning systems are increasingly successful. In particular, a more recent paper by Zhu et al. [1029] uses a generative adversarial network (GAN) to "hallucinate" images of new classes by their textual descriptions and then extracts features from these hallucinated images; this comes close to the usage of GANs to produce and refine synthetic data that we explore in Chapter 10; see also recent surveys of zero-shot and few-shot learning [911, 916, 917, 949].

Note, however, that one- and zero-shot learning still require large labeled datasets. The difference is that we do not need a lot of images for each new class any more. But the feature extraction network has to be trained on similar labeled classes: a zero-shot approach will not work if you train it on birds and then try to look for a chair based on its textual description. Until we are able to use super-networks trained on every kind of images possible (as we will see shortly, we still have quite a way to go before this becomes possible, if it is even possible at all with our current approaches), this is still a data-intensive approach, although restrictions on what kind of data to use are relaxed.

A middle ground is taken by models that try to generalize from several examples, a field known as *few-shot learning* [916]. Similar to one-shot learning, generalizing from few examples in complex problems is a process that has to be guided by expressive and informative prior distributions—otherwise, the data will simply not be enough. In many ways, this is how human learning works: we usually need a few examples to grasp a novel concept, but never thousands of examples, because the new concept probably fits into the prior framework about the world that we have built over our lifetimes.

To get these prior distributions in a machine learning model, we can use a number of different approaches. We have seen a couple of examples above, and the general classification of one- and few-shot approaches includes at least the following:

- *data augmentation* that extends the small available dataset with transformations that do not change the properties that we need to learn;
- *multitask learning* that trains a model to solve several problems, each of which may have a small dataset, but together these datasets are large enough;
- *embedding learning* that learns latent representations which can be generalized to new examples, as we have discussed above;

[2]*ImageNet* [187], the main basic dataset for computer vision models, is very fond of canines: it distinguishes between 120 different dog breeds from around the world.

- *fine-tuning* that updates existing models that have been pretrained on different tasks (possibly unsupervised) with small amounts of new data.

We will encounter all of these techniques in this book.

1.3 Weakly Supervised Training: Trading Labels for Computation

For many problems, obtaining labeled data is expensive but unlabeled data, especially data that is not directly related to the specific problem at hand, is plentiful. Consider again the basic computer vision problems we have talked about: it is very expensive to obtain a large dataset of labeled images of cow herds, but it is much less expensive to get a large dataset of unlabeled such images, and it is almost trivial to simply get a lot of images with and without cows.

But can unlabeled data help? Basic intuition tells that it may not be easy but should be possible. After all, we humans learn from all kinds of random images, and during infancy, we develop an intuition about the world around us that generalizes exceptionally well. Armed with this intuition, we can later in life do one-shot and zero-shot learning with no problem. And the images were never actually labeled, the learning we do can hardly be called supervised. Although it is still a far cry from human abilities (see, e.g., a recent treatise by Francois Chollet [154] for a realistic breakdown of where we stand in terms of artificial general intelligence), there are several approaches being developed in machine learning to make use of all this extra unlabeled data lying around.

First, one can use unlabeled data to produce new labeled data; although the new "pseudolabels" are not guaranteed to be correct, they will still help, and one can revisit and correct them in later iterations. A striking illustration of this approach has appeared in a recent work by Xie et al. [956], where researchers from Google Brain and Carnegie Mellon University applied the following algorithm:

- start from a "teacher" model trained as usual, on a (smaller) labeled dataset;
- use the "teacher" model on the (larger) unlabeled dataset, producing pseudolabels;
- train a "student" model on the resulting large labeled dataset;
- use the trained student model as the teacher for the next iteration, and then repeat the process iteratively.

This is a rather standard approach, used many times before in semi-supervised learning and also known as *knowledge distillation* [129, 293, 345, 541, 603]. But by adding noise to the student model, Xie et al. managed to improve the state-of-the-art results on *ImageNet*, the most standard and beaten-down large-scale image classification dataset. For this, however, they needed a separate dataset with 300 million unlabeled images and a lot of computational power: 3.5 days on a 2048-core Google TPU, on the same scale as *AlphaZero* needed to outperform every other engine in Go and chess [800]; we will shortly see that this kind of computation does not come for free.

Another interesting example of replacing labeled data with (lots of) unlabeled data comes from the problem we already discussed: segmentation. It is indeed very hard to produce labeled data for training segmentation models... but do we have to? Segmentation models from classical computer vision, before the advent of deep learning, do not require any labeled data: they cluster pixels according to their features (color and perhaps features of neighboring pixels) or run an algorithm to cut the graph of pixels into segments with minimal possible cost [657, 839]. Modern deep learning models work better, of course, but it looks like recent advances make it possible to train deep learning models to do segmentation without labeled data as well.

Approaches such as W-Net [948] use unsupervised autoencoder-style training to extract features from pixels and then segment them with a classical algorithm. Approaches such as invariant information clustering [399] develop image clustering approaches and then apply them to each pixel in a convolution-based way, thus transforming image clustering into segmentation. One of the most intriguing lines of work that results in unsupervised clustering uses GANs for image manipulation. The "cut-and-paste" approach [716] works in an adversarial way:

- one network, the mask generator, constructs a segmentation mask from the features of pixels in a detected object;
- then the object is cut out according to this mask and transferred to a different, object-free location on the image;
- another network, the discriminator, now has to distinguish whether an image patch has been produced by this cut-and-paste pipeline or is just a patch of a real image.

The idea is that good segmentation masks will make realistic pasted images, and in order to convince the discriminator, the mask generator will have to learn to produce high-quality segmentation masks. We will discuss this approach and its ramifications for synthetic data in Section 9.3.

Semi-supervised teacher–student training and unsupervised segmentation via cut-and-paste are just two directions out of many that are currently being developed. In these works, researchers are exploring various ways to trade the need for labeled datasets for extra unlabeled data, extra unrelated data, or extra computation, all of which are becoming more and more readily available. Still, this does not completely solve the data problem, and the computational challenges might prove to be insurmountable.

1.4 Machine Learning Without Data: Leaving Moore's Law in the Dust

Interestingly, some kinds of machine learning do not require any external data at all, let alone labeled data. Usually, the idea is that they are able to generate data for themselves, and the catch is that they need *a lot* of generation power.

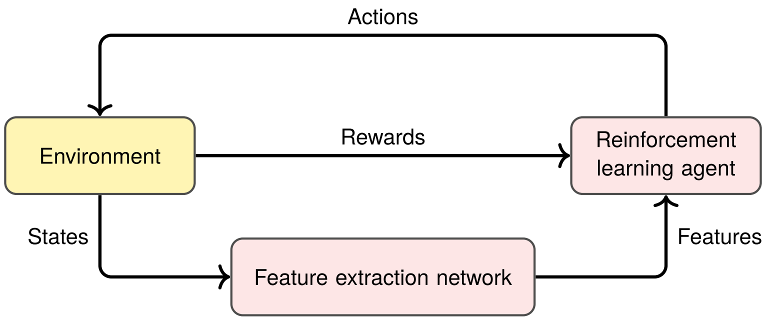

Fig. 1.5 General reinforcement learning flowchart.

The main field where this becomes possible is *reinforcement learning* (RL), where an agent learns to perform well in an interactive environment [788, 831]. An agent can perform actions and receive rewards for these actions from the environment. Usually, modern RL architectures consist of the feature extraction part that processes environment states into features and an RL agent that transforms features into actions and converts rewards from the environment into weight updates; see an illustration in Figure 1.5.

The poster child of these approaches is *AlphaZero* by *DeepMind* [800]. Their original breakthrough was *AlphaGo* [799], a model that beat Lee Sedol, one of the top human Go players, in an official match in March 2016. Long after *DeepBlue* beat Kasparov in chess, professional-level Go was remaining out of reach for computer programs, and *AlphaGo*'s success was unexpected even in 2016. The match between Lee Sedol and *AlphaGo* became one of the most publicized events in AI history and was widely considered as the "Sputnik moment" for Asia in AI, the moment when China, Japan, and South Korea realized that deep learning is to be taken seriously. But *AlphaGo* utilized a lot of labeled data: it had a pretraining step that used a large database of professional games.

AlphaZero takes its name from the fact that it needs zero training data: it begins by knowing only the rules of the game and achieves top results through self-play, actually with a very simple loss function combined with tree search. *AlphaZero* beat *AlphaGo* (and its later version, *AlphaGo Zero*) in Go and one of the top chess engines, *Stockfish*, in chess.

A recent result by the *DeepMind* reinforcement learning team, *MuZero* [770], is even more impressive. *MuZero* is an approach based on *model-based RL*, that is, it builds a model of the environment as it goes and does not know the rules of the game beforehand but has to learn them from scratch; e.g., in chess, it cannot make illegal moves as actions but can consider them in tree search and has to learn that they are illegal by itself. With this additional complication, *MuZero* was able to achieve *AlphaZero*'s skill in chess and shogi and even outperform it in Go. Most importantly, the same model could also be applied to situations with more complex environment

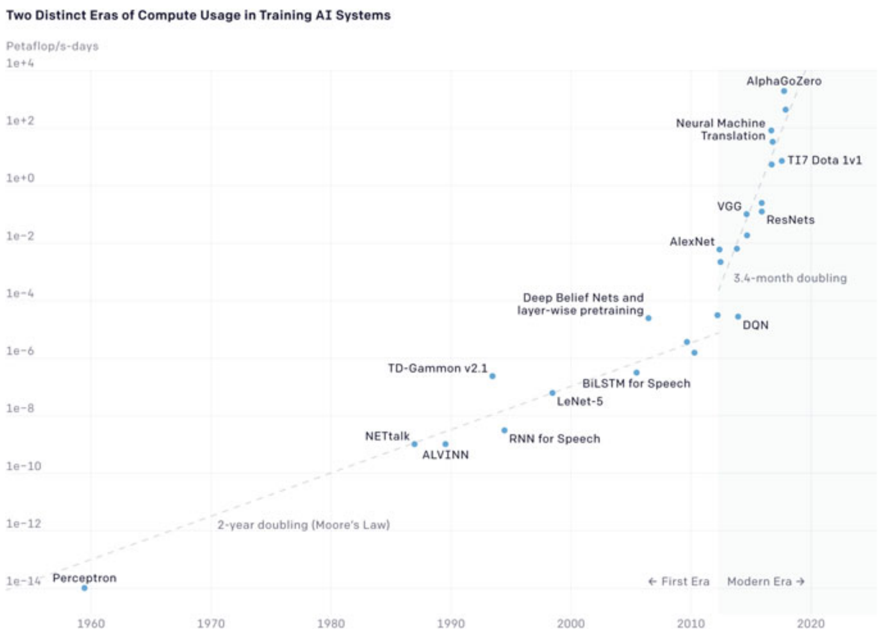

Fig. 1.6 The changing trend in deep learning: a comparison of “Moore’s law” in machine learning before and after the rise of deep learning. Chart by *OpenAI* [16].

states, e.g., to computer games in *Atari* environments (a standard benchmark in reinforcement learning).

So what’s the catch? Is this the end of the data problem? One drawback is that not every problem can be formulated in terms of RL with no data required. You can learn to play games, i.e., self-contained finite structures where all rules are known in advance. But how do we learn, say, autonomous driving or walking, with a much wider variety of possible situations and individual components of these situations? One possible solution here is to use synthetic virtual environments, and we will discuss them in detail in Chapter 7.

Another, perhaps even more serious, problem is the amount of computation needed for further advances in reinforcement learning. To learn to play chess and Go, *MuZero* used 1000 third-generation Google TPUs to simulate self-play games. This does not tell us much by itself, but here is an interesting observation made by the *OpenAI* team [16], illustrated in Fig. 1.6. They noticed that before 2012, computational resources needed to train state-of-the-art AI models grew basically according to Moore’s Law, doubling their computational requirements every two years. But with the advent of deep learning, in 2012–2019, computational resources for top AI model training doubled on average every 3.4 months! This is a huge rate of increase, and, obviously, it cannot continue forever, as the actual hardware computational power growth is only slowing down compared to Moore’s Law. Replication of *AlphaZero* experiments has been recently estimated to cost about $35 million at current *Google*

Cloud rates [365]; while the cost of computation is dropping, it does so at a much slower rate than the increase of computation needed for AI.

Thus, one possible scenario for further AI development is that yes, indeed, this "brute force" approach might theoretically take us very far, maybe even to general artificial intelligence, but it would require more computational power than we actually have in our puny Solar System. Note that a similar phenomenon, albeit on a smaller scale, happened with the second wave of hype for artificial neural networks: researchers in the late 1980s had a lot of great ideas about neural architectures (including CNNs, RNNs, RL, and much more), but neither the data nor the computational power was sufficient to make a breakthrough, and neural networks were relegated to "the second best way of doing just about anything"[3].

Still, at present, reinforcement learning represents another feasible way to trade labeled data for computation, as the example of *AlphaGo* blossoming into *AlphaZero* and *MuZero* clearly shows. With this, we finish a brief overview of alternatives and come to the main subject of this book: *synthetic data*.

1.5 Why Synthetic Data?

Let us go back to segmentation, a standard computer vision problem that we already discussed in Section 1.1. How does one produce a labeled dataset for image segmentation? At some point, all images have to be manually processed: humans have to either draw or at least verify and correct segmentation masks. Making the result pixel-perfect is so laborious that it is commonly considered to be not worth the effort. Figure 1.7a–c shows samples from the industry standard *Microsoft Common Objects in Context* (MS COCO) dataset [525]; you can immediately see that the segmentation mask is a rather rough polygon and misses many finer features. It did not take us long to find such rough segmentation maps, by the way; these are some of the first images found by the "dog" and "person" queries.

How can one get a higher quality segmentation dataset? To manually correct all of these masks in the MS COCO dataset would probably cost hundreds of thousand dollars. Fortunately, there is a different solution: *synthetic data*. In the context of segmentation, this means that the dataset developers create a 3D environment with modes of the objects they want to recognize and their surroundings and then render the result. Figure 1.7d–e shows a sample frame from a synthetic dataset called *ProcSy* [449] (we discuss it in more detail in Section 6.5): note how the segmentation map is now perfectly correct. While 3D modeling is still mostly manual labor, this is a one-time investment, and after this investment, one can get a potentially unlimited number of pixel-perfect labeled data: not only RGB images and segmentation maps but also depth images, stereo pairs produced from different viewpoints, point clouds, synthetic video clips, and other modalities.

[3] A quote usually attributed to John Denker; see, e.g., [339].

In general, many problems of modern AI come down to insufficient data: either the available datasets are too small or, also very often, even while capturing unlabeled data is relatively easy, the costs of manual labeling are prohibitively high. *Synthetic data* is an important approach to solving the data problem by either producing artificial data from scratch or using advanced data manipulation techniques to produce novel and diverse training examples. The synthetic data approach is most easily exemplified by standard computer vision problems, as we have done above, but it is also relevant in other domains (we will discuss some of them in Chapter 8).

Naturally, other problems arise; the most important of them being the problem of *domain transfer*: synthetic images, as you can see from Figure 1.7, do not look exactly like real images, and one has to make them as photorealistic as possible and/or devise techniques that help models transfer from synthetic training sets to real test sets; thus, domain adaptation becomes a major topic in synthetic data research and in this book as well (see Chapter 10). Note, however, that a common theme in synthetic data research is whether realism is actually necessary; we will encounter this question several times in this book.

We begin with a few general remarks regarding synthetic data. First, note that synthetic data can be produced and supplied to machine learning models on the fly, during training, with software synthetic data generators, thus alleviating the need to ever store huge datasets; see, e.g., Mason et al. [583] who discuss this "on the fly" generation in detail. Second, while synthetic data has been a rising field for some time, I do not know of a satisfactory general overview of synthetic data in machine learning or deep learning, and this was my primary motivation for writing this book. I would like to note surveys that attempt to cover applications of synthetic data [157, 467, 875] and a special issue of the *International Journal of Computer Vision* [253], but I hope that the present work paints a more comprehensive picture.

Third, we distinguish between synthetic data and *data augmentation*; the latter is a set of techniques intended to modify real data rather than create new synthetic data. These days, data augmentation is a crucial part of virtually every computer vision pipeline; we refer to the surveys [792, 914] and especially recommend the *Albumentations* library [104] that has proven invaluable in our practice, but in this survey, we concentrate on synthetic data rather than augmentations (the latter will only be the subject of Section 3.4). Admittedly, the line between them is blurry, and some techniques discussed here could instead be classified as "smart augmentation".

Fourth, we note a natural application of synthetic data in machine learning: testing hypotheses and comparing methods and algorithms in a controlled synthetic setting. Toy examples and illustrative examples are usually synthetic, with a known data distribution so that machine learning models can be evaluated on how well they learn this distribution. This approach is widely used throughout the field, sometimes for entire meta-analyses [80, 491, 797], and we do not dwell on it here; our subject is synthetic data used to transfer to real data rather than direct comparisons between models on synthetic datasets.

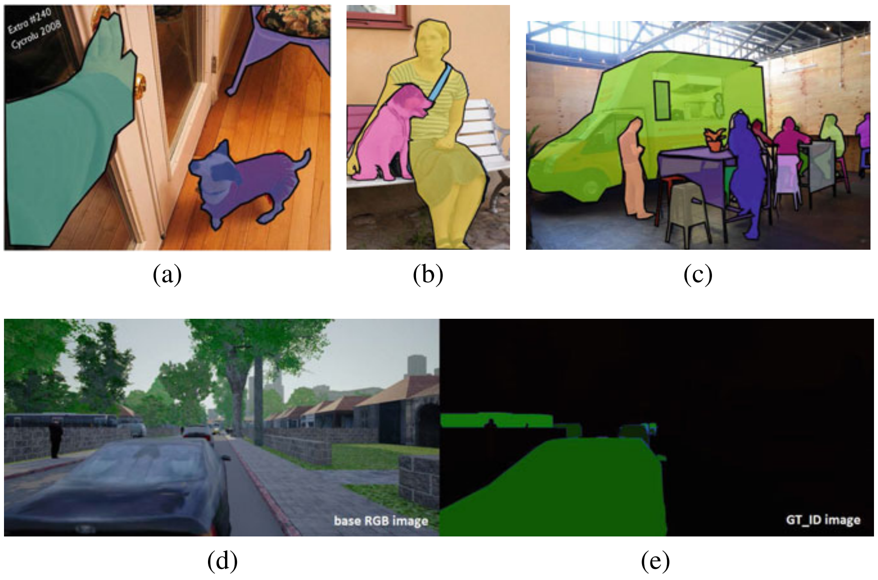

Fig. 1.7 Sample images: (a–c) MS COCO [525] real data samples with ground truth segmentation maps overlaid; (d–e) *ProcSy* [449]: (d) RGB image; (e) ground truth segmentation map.

1.6 The Plan

This book covers three main directions for the use of synthetic data in machine learning; in this section, we introduce all three, giving references to specific parts of the book related to these directions.

1. Using synthetically generated datasets to train machine learning models directly. This is an approach often taken in computer vision, and most of this book is devoted to variations of this approach. In particular, one can:
    - train models on synthetic data with the intention to use them on real data; we discuss this approach through most of Chapters 6, 7, and 8;
    - train (usually generative) models that change (refine) synthetic data in order to make it more suitable for training or adapt the model to allow it to be trained on synthetic data; Chapter 10 is devoted to this kind of models.
2. Using synthetic data to augment existing real datasets so that the resulting hybrid datasets are better suited for training the models. In this case, synthetic data is usually employed to cover parts of the data distribution that are not sufficiently represented in the real dataset, with the main purpose being to alleviate dataset bias. The synthetic data can either:
    - be generated separately with, e.g., CGI-based methods for computer vision (see examples in Chapters 3 and 7)

- or be generated from existing real data with the help of generative models (see Section 10.4).

3. Using synthetic data to resolve privacy or legal issues that make the use of real data impossible or prohibitively hard. This becomes especially relevant for certain specific fields of application, among which we discuss:

   - synthetic data in healthcare, which is not restricted to imaging but also extends to medical records and the like (Section 10.7);
   - synthetic data in finance and social sciences, where direct applications are hard but privacy-related ones do begin to appear (Section 11.5);
   - synthetic data with privacy guarantees: many applications are sensitive enough to require a guarantee of privacy, for example, from the standpoint of the differential privacy framework, and there has been an important line of work that makes synthetic data generation provide such guarantees, which we consider in Chapter 11.

The book is organized as follows. Chapter 2 gives a very brief introduction to deep learning; while one cannot hope to deliver an actual in-depth introductory text in the space of a single chapter, we will nevertheless start with the basics of how the deep learning revolution has transformed machine learning in the late 2000s (Section 2.1), how a neural network is organized (Section 2.3), and how it is trained via various modifications of gradient descent (Sections 2.4 and 2.5).

Next, since computer vision is by far the most common domain for applying synthetic data, we will devote Chapter 3 to deep architectures that are designed for computer vision problems and that will be used throughout other chapters of the book. Section 3.1 introduces the notion of convolutional neural networks, Section 3.2 shows several basic ideas that underlie modern convolutional architectures for image classification, object detection, and segmentation, and Section 3.3 provides a case study of neural architectures for object detection; we cannot hope to cover everything but at least try to take a deep dive into a single topic to illustrate the depth and variability of deep learning approaches in computer vision.

The final, most advanced introductory chapter deals with generative models in deep learning; they are the topic of Chapter 4. We begin by discussing generative models in machine learning in general (Section 4.1), introducing the difference between discriminative and generative models. Next, we discuss Ian Goodfellow's taxonomy of generative models in deep learning and give an overview of tractable density models, including an introduction to normalizing flows (Section 4.2). In Section 4.3, we talk about variational autoencoders as the primary example of approximate explicit density models. Section 4.4 introduces the class of generative models most directly relevant to synthetic data: generative adversarial networks (GAN). Section 4.5 discusses loss functions in modern GANs, Section 4.6 gives a brief overview of some important general GAN-based architectures, and Section 4.7 finishes the chapter with a case study of GAN-based style transfer, a problem which both serves as a good illustration for modern adversarial architectures and is directly relevant to synthetic-to-real domain adaptation.

Chapter 5 is devoted to the early history of synthetic data. It may seem that synthetic data has only been on the rise for the last few years, but we will see that the use of synthetic data dates back to the 1970s, when computer vision took its first steps (Section 5.1), was used as a testbed for experimental comparisons throughout the history of computer vision (Section 5.2), and was used to train one of the first self-driving cars in 1989 (Section 5.3). The final sections of this chapter are devoted to early robotics: we first discuss how synthetic data was used to train robot control systems from their very inception and how it was faced with some reservations and criticism (Section 5.4) and then make a more detailed example of an early robotic navigation system called MOBOT (Section 5.5).

Chapter 6 presents synthetic datasets and results for basic computer vision problems, including low-level problems such as optical flow or stereo disparity estimation (Section 6.2), datasets of basic objects (Section 6.3), basic high-level computer vision problems, including a case study on object detection (Section 6.4) and a discussion of other high-level problems (Section 6.5), human-related synthetic data (Section 6.6), and character and text recognition and visual reasoning problems (Section 6.7). Throughout Chapter 6, we refer to specific architectures whose results have been improved by training on synthetic data and show examples of images from synthetic datasets.

In Chapter 7, we proceed to synthetic datasets that are more akin to full-scale simulated environments, covering outdoor and urban environments (Section 7.2), indoor scenes (Section 7.3), synthetic simulators for robotics (Section 7.4), simulators for autonomous flying vehicles (Section 7.5), and computer games used as simulation environments (Section 7.6). Synthetic simulated environments are an absolute necessity for, e.g., end-to-end reinforcement learning architectures, and we will see examples of situations where they suffice to train a successful RL agent as well as situations where additional work on domain transfer is required.

While synthetic data has been most instrumental in computer vision, there are other domains of application for synthetic data as well, and Chapter 8 is devoted to exactly such domains. In particular, neural programming (Section 8.2) is a field completely reliant on automatically generated training samples: it does not make any sense to force humans to write millions of trivial computer programs. In bioinformatics (Section 8.3), most applications of synthetic data again lie in the domain of medical imaging, but there are fields where one learns to generate other kinds of information, for instance, fingerprints of molecules, and trains subsequent models on this synthetically generated data. Finally, natural language processing (Section 8.4) is not a field where synthetic data has really taken off despite obvious successes in text generation [94, 697, 698]. A computer program that can generate coherent text with predefined properties must be an AI model that captures a lot of useful features about the language, and it is usually more productive to use the model itself than try to learn a different model on its outputs; however, there are some examples of synthetic data in NLP as well.

Chapter 9 discusses research intended to improve synthetic data generation. The notion of *domain randomization* (Section 9.1) means trying to cover as much of the data distribution with synthetic samples as possible, making them maximally different and randomized, with the hope to capture the real data in the support of the synthetic data distribution as well. Section 9.2 discusses current developments in methods for CGI-based generation, including more realistic simulations of real-world sensors (cameras, LiDAR systems, etc.) and more complex ways to define and generate synthetic scenes. Synthetic data produced by "cutting and pasting" parts of real data samples is discussed in Section 9.3, and we end the chapter with a discussion of the direct generation of synthetic data by generative models (Section 9.4). It is rare to see such examples because, similar to natural language processing, a generative model trained to produce high-quality samples from a given domain probably already contains a model that can be used directly or fine-tuned for various applications; still, there are situations where GANs can help directly.

The next part of the book deals with the main research problem of synthetic data, namely *synthetic-to-real domain adaptation*: how can we make a model trained on synthetic data perform well on real samples? After all, the ultimate goal of the entire enterprise is always to apply the model to real data. With this, we get to Chapter 10 that discusses synthetic-to-real domain adaptation itself. There are many approaches to this problem that can be broadly classified into two categories:

- *synthetic-to-real refinement*, where domain adaptation models are used to make synthetic data more realistic (Section 10.1);
- *domain adaptation at the feature/model level*, where the model and/or the training process are adapted rather than the data itself (Section 10.5).

The difference is that with refinement, one usually can extract refined input data: either synthetic samples made "more realistic" or real samples made "more synthetic-like"; with domain adaptation at the model level, the architectures usually just learn to extract common features from both domains. In this chapter, we also discuss case studies of domain adaptation for control and robotics (Section 10.6) and medical imaging (Section 10.7).

Chapter 11 is devoted to the privacy side of synthetic data: can we generate synthetic data which is *guaranteed* not to contain personal information about individual entries from the original dataset? To get such guarantees, we need to venture into *differential privacy*, a field that belongs more to the domain of theoretical cryptography than machine learning. Sections 11.2 and 11.3 introduce differential privacy in general and specifically for deep learning models, Section 11.4 shows how to generate synthetic data with differential privacy guarantees, and Section 11.5 presents a case study about private synthetic data in finance and related fields, in particular, electronic medical records.

In an attempt to look forward, we devote Chapter 12 to directions for further work related to synthetic data that seem most promising. We consider four such directions:

- Section 12.1 considers *procedural generation* of synthetic data, where the data is made more realistic not by low-level refinement but by improving the high-level generation process: for instance, instead of refining the textures of wood and fabric on chairs, we are talking about a more consistent layout of the entire synthetic room's interior;

- in Section 12.2, we introduce the notion of *closing the feedback loop* for synthetic data generation: since the end goal is to improve the performance of models trained on synthetic datasets, maybe we can change the parameters of synthetic data generation in such a way as to directly increase this metric;
- Section 12.3 talks about introducing *domain knowledge* into domain adaptation; specifically, we consider an example where the model contains both a domain-specific generative model designed to produce synthetic images and a bottom-up model that estimates the necessary parameters in an image;
- Section 12.4 shows how domain adaptation models can be improved with additional modalities that are easy to obtain in synthetic datasets; for instance, in computer vision, it is trivial to augment synthetic data with 3D information such as depth maps of surface normals since synthetic data is produced from 3D scenes, so maybe this additional information can help a refiner to make this data even more realistic.

Finally, Section 12.5 concludes the book by drawing some general conclusions about the place of synthetic data in modern AI and possible future work in this direction.

By now, we have seen how the deep learning revolution makes demands on computational power and data that are increasingly hard to satisfy. There are some ways to get around the need for ever growing labeled datasets, but they usually seem to require even more computational resources, which are by now also not so easy to obtain. We have seen that synthetic data is one possible way out of this conundrum. But for the uninitiated, it is still unclear what this “deep learning” is all about, and this is exactly what awaits us in the next chapter.