OSTRAVSKÁ UNIVERZITA

DOKTORSKÁ PRÁCE

2021 MGA. JAN HŮLA

OSTRAVSKÁ UNIVERZITA
PŘÍRODOVĚDECKÁ FAKULTA
KATEDRA MATEMATIKY

Tři nové přístupy k data-driven modelování

Doktorská práce

Autor práce: MgA. Jan Hůla
Vedoucí práce: prof. Irina Perfilieva, CSc.

2021

UNIVERSITY OF OSTRAVA
FACULTY OF SCIENCE
DEPARTMENT OF MATHEMATICS

Three Novel Approaches to Data-driven Modeling

Doctoral Thesis

Author: MgA. Jan Hůla
Supervisor: prof. Irina Perfilieva, CSc.

2021

OSTRAVSKÁ UNIVERZITA

Přírodovědecká fakulta

Akademický rok: 2021/2022

ZADÁNÍ DISERTAČNÍ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	MgA. Jan HŮLA
Osobní číslo:	P15132
Studijní program:	P1103 Aplikovaná matematika
Studijní obor:	Aplikovaná matematika a Fuzzy modelování
Téma práce:	Tři nové přístupy k data-driven modelování
Téma práce anglicky:	Three Novel Approaches to Data-driven Modeling
Zadávající katedra:	Katedra matematiky

Zásady pro vypracování

Navrhnout a zdůvodnit způsoby integrování znalostí o dané doméně s neuronovými sítěmi. Konkrétně navrhnout způsoby integrace, které: 1) využívají simulátorů pozorovaných dat k obohacení datové sady, čímž má být dosaženo zefektivnění učící fáze, 2) umožňují zpracovávat strukturované vstupy ve formě grafu. 3) využívají znalost o transformacích dat, vůči kterým má být výsledný model invariantní. Prokázat, že navržené metody snižují náročnost na manuální anotaci dat a zvyšují robustnost a interpretovatelnost výsledného modelu. Empiricky ověřit navržené metody na úlohách počítačového vidění.

Rozsah pracovní zprávy:	
Rozsah grafických prací:	
Forma zpracování disertační práce:	tištěná/elektronická
Jazyk zpracování:	Angličtina

Seznam doporučené literatury:

1. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep learning. MIT press, 2016.
2. Szeliski, Richard. Computer vision: algorithms and applications. Springer Science & Business Media, 2010.
3. Hamilton, William L. „Graph representation learning." Synthesis Lectures on Artifical Intelligence and Machine Learning 14.3 (2020): 1-159.
4. Yuille, Alan, and Daniel Kersten. „Vision as Bayesian inference: analysis by synthesis?." Trends in cognitive sciences 10.7 (2006): 301-308.
5. Kulkarni, Tejas D., et al. „Deep convolutional inverse graphics network." arXiv preprint arXiv:1503.03167 (2015).
6. Cranmer, Kyle, Johann Brehmer, and Gilles Louppe. „The frontier of simulation-based inference." Proceedings of the National Academy of Sciences 117.48 (2020): 30055-30062.

Školitel disertační práce:	prof. Irina Perfiljeva, CSc. Centrum excelence IT4Innovations

Datum zadání disertační práce 20. prosince 2021
Termín odevzdání disertační práce: 31. ledna 2022

prof. Irina Perfiljeva, CSc.
vedoucí doktorské práce

RNDr. Zuzana Václavíková, Ph.D.
vedoucí katedry

V Ostravě dne 20. prosince 2021

Abstrakt

Tato práce popisuje přístupy, které integrují expertní znalosti s neuronovými sítěmi. Tímto má být dosaženo menší náročnosti na velikost trénovacích dat a zároveň vyšší robustnosti a snažší interpretovatelnosti výsledného modelu. Pozornost je věnována třem přístupům, které využívají expertní znalosti odlišnými způsoby. První přístup využívá generativních modelů, které popisují vznik pozorovaných dat. Druhý přístup je založen na grafových neuronových sítích, které zpracovávají strukturovaná data a třetí přístup se věnuje odstraňování symetrií z reprezentace dat. Účinnost popsaných metod je experimentálně ověřena na reálných problémech, z nichž velká část se týká počítačového vidění.

Klíčová slova: Znalost domény, neuronové sítě, generativní modely, grafové neuronové sítě, počítačové vidění, symetrie, modelování založené na datech

Abstract

This thesis describes approaches that integrate domain knowledge with neural networks. These approaches should lead to smaller requirements on the size of a training set, higher robustness, and easier interpretation of the resulting model. We focus on three approaches, which leverage domain knowledge in different ways. The first one uses generative models that model the creation of the observed data. The second one is based on graph neural networks, which process structured data and the third one is devoted to the removal of symmetries from the representation of data. The effectiveness of the methods described is experimentally verified on real problems, which are mostly related to computer vision.

Key Words: Domain knowledge, neural networks, generative models, graph neural networks, computer vision, symmetries, data-driven modeling

Já, níže podepsaný student, tímto čestně prohlašuji, že text mnou odevzdané závěrečné práce v písemné podobě i na CD nosiči je totožný s textem závěrečné práce vloženým v databázi DIPL2.

Prohlašuji, že předložená práce je mým původním autorským dílem, které jsem vypracoval samostatně. Veškerou literaturu a další zdroje, z nichž jsem při zpracování čerpal, v práci řádně cituji a jsou uvedeny v seznamu použité literatury.

V Ostravě dne 26. 12. 2021 .....

podpis

Rád bych zde poděkoval své školitelce, profesorce Irině Perfilievě, za její dlouhodobou podporu a přínosné podněty. Měl jsem štěstí, že jsem se dostal právě k Vám a velmi si vážím Vašich zkušeností, o které jste mne obohatila. Za podporu a kreativní pracovní prostředí chci poděkovat také všem kolegům z Ústavu pro výzkum a aplikace fuzzy modelování a konkrétně Petru Hurtíkovi. V neposlední řadě za trpělivost děkuji své ženě Jaruš.

Contents

Preface

As the title of this thesis suggests, the aim of the text is to describe three distinct approaches that combine data-driven methods with model-driven methods. These terms will be explained in detail in the first chapter, but in simple words, each of these three approaches provides a way to combine domain knowledge with neural networks.

I started my doctoral studies when *deep learning* was becoming popular and I was attracted by the idea that complicated algorithms would be discovered by an optimization algorithm instead of being designed by a researcher. With the increasing complexity of the problems we face, it seems that such declarative approaches, where we say what we want (for example, by means of a dataset and a cost function) and let the computer find a concrete solution, will become irreplaceable.

On the other hand, my intuition was that the enormous requirement on the size of datasets and amount of computation was not necessary and wasteful. This issue is becoming more relevant due to the popular belief that scaling datasets, the size of models, and computation time is the most straight-forward way to solve practical problems. As a result, state-of-the-art deep learning models contain billions of parameters and training such models is out of reach for individual researchers.

For this reason, we have agreed with my supervisor that I will explore possible ways which combine domain knowledge with deep learning to reduce the amount of data and computation needed. My exploration was unsystematic but intense. During my studies, I collaborated and published papers with a diverse group of researchers and as a result I created for myself a mental map of different topics and methods and connections between them. I considered this mental map as the main output of my studies.

This text describes three selected approaches which I worked on. Each chapter, except the introduction, is devoted to one approach and contains a general description of the important ideas together with high-level overview of my published contributions. These contributions are provided as part of the appendix.

The whole text is optimized for readability rather than mathematical rigor, which may not be a standard way of writing a thesis in applied mathematics. I use mathematical arguments at several places, but most of the time I focus on conveying the conceptual understanding, which is supplemented by many images and cartoon drawings. Regarding the style of writing, this is the last sentence where I use the pronoun “I” instead of the plural form “we”.

1 Introduction

1.1 Data-driven vs model-driven approaches

We can distinguish two main axes for the classification of algorithms that try to automate some real-world tasks. On one axis, we have purely data-driven approaches, and on the other, we have model-driven approaches. Purely data-driven approaches make no assumptions about the task they automate and instead fit a statistical model to produce the desired behavior, specified either by training examples and a loss function or by the rewards given during trial and error exploration. *Deep neural networks* [1] (DNNs) and *reinforcement learning* (RL) [2] are probably the best-known examples of this direction. Even better adjective for naming such methods would be *optimization-driven* because in some cases data may not be necessary. For example, the RL agent called *AlphaZero* [3] has learned to play the game of Go and chess just from the rules of these games, without any examples of play. The crucial feature of these methods is that the final algorithm is obtained by an optimization/search.

On the other hand, model-driven approaches encode the knowledge about the problem domain into an algorithm using variables that have a clearly defined meaning. *Expert systems* [4] constitute a well-known example of this direction. Of course, there is a whole continuum of methods that contain some data-driven aspects and some model-driven aspects.

The benefits of purely data-driven approaches are as follows - we do not need to understand the problem we are trying to solve, and we are also not limiting the hypothesis space with assumptions that may not hold in the real world. The downside is that we may require large amounts of data to fit our model, and the final model will probably not be interpretable, and therefore not easily modifiable.

The benefits of purely model-driven approaches are interpretability, easy modifiability by reasoning about the model and criticizing it, and no need for training data. The downsides are that the assumptions encoded in the model may be wrong, incomplete, or time-consuming to acquire. It is preferable to be aware of these tradeoffs and be able to navigate between them according to the need of the problem and the resources available.

This thesis will describe methods and algorithms which allow for the successful synthesis of model-driven and data-driven approaches.

1.2 Analogy from nature

Nature provides us with a nice analogy for the combination of model-driven and data-driven algorithms. In this context, it would be more appropriate to say a combination of hard-wired and experience-driven algorithms. To simplify the discussion, let us assume that the goal of every creature is to stay in existence as long as possible. To achieve this, it needs to adopt a certain behavior which allows it to endure various forces of the environment it lives in. This behavior could be either produced by evolution (which can also be viewed as a kind of learning) or learned during the lifetime of the creature.

In very simple organisms which live in more stationary environments, the behavior is rather fixed and encoded by genes. As we go up along the axis of complexity of the organism and intelligence, we see an increasing adaptability. More intelligent organisms need to adapt to the constantly changing environment and other adapting organisms, and therefore they need to be more "experience-driven". Assuming that certain properties of the environment are stable (such as physical laws, the fact that the world is three-dimensional, or even a crude template of a human face), it makes sense for evolution to encode the knowledge of these properties into our genes.

On the other hand, other aspects, such as a concrete map of the environment with locations of food sources or visual features of various objects, have to be learned from the experience during the lifetime of an organism because they could not be accounted for by evolution. This tension between *Nature vs Nurture* was the subject of many famous debates in cognitive science and philosophy, dating back at least to Hume and Kant [5]. These debates revolved around issues dealing with the innate abilities of humans.

Today, this debate is still alive in artificial intelligence, where many argue against using prior knowledge in the design of algorithms with the hope that everything could be learned from data [6]. This desire reflects the recent success of data-driven algorithms which are trained on large datasets or, in the case of RL, by a long process of trial and error. For example, as mentioned above, the RL agent called *AlphaZero* [3], learned to play the game of Go and chess just by trial and error learning without any hard-coded knowledge. The trained agent outperformed all other heursistics designed over several decades of human effort.

Nonetheless, the search space for the game of Go or chess is relatively small in comparison to the search space that could arise in real applications. As we do not have computational resources available to simulate the search process of such extent as evolution, we will need to supplement the search with prior knowledge and in this way restrict the search space or bias it towards certain solutions.

1.3 Model-driven vs Simulator-based

The word "model" may be used in many different contexts. We want to distinguish between two approaches which could be prefixed by this word. The first, which we call *model-driven* approaches, represent algorithms which in some sense reflect our understanding/model of the problem domain. We can use this understanding/model to imagine possible scenarios which could arise and devise heuristics that would work well in these scenarios. We can also use our understanding of the symmetries present in the problem and make the algorithm invariant with respect to these symmetries.

For example, let us imagine that we want to design an algorithm that detects whether a person in a photograph is smiling. Doing this in a purely data-driven way would, for instance, mean training a generic neural network classifier to predict whether the person in the image is smiling or not. Leveraging our understanding of the problem, we can design an algorithm to detect certain features corresponding to the lips. Based on these, the algorithm would classify whether the person is smiling or not. In our terminology, this algorithm would be partly model-driven (depending on whether the decision boundary is manually designed or learned) because we are

encoding assumptions into the algorithm and restricting the search space.

The second case denotes algorithms which have an explicit model/simulator of the problem domain embedded inside. We will call such algorithms *simulator-based* to distinguish them from model-driven algorithms. In the case of the smiling face recognition, this would mean that the algorithm contains a simulator of a human face which can simulate images of human faces from some high-level descriptions. The classification would work by inverting the simulator to recover the high-level description of the face and then classify the face based on this high-level description.

Sometimes, the expression *model-based* is used. For example, in reinforcement learning, where we distinguish between model-free and model-based methods [7]. In model-based methods, the agent disposes of a simulator and can "imagine" the effects of its actions. We use the expression "simulator-based" to make it more distinct from model-driven.

To summarize, by model-driven we understand algorithms that in some way encode assumptions about the problem domain. By simulator-based, we understand algorithms that contain a generative model/simulator for the problem domain. A simulator-based algorithm is of course an example of a model-driven algorithm. In this thesis, only Chapter 2 is devoted specifically to simulator-based methods. The rest of the thesis is devoted to model-driven methods in general.

1.4 Chapter summaries

The following text is divided into three chapters devoted to three different topics. Each of them provides a particular way to leverage domain knowledge in applications of neural networks. Chapter 2 focuses on the usage of generative models/simulators as a tool to encode domain knowledge. Such simulators can be leveraged in combination with neural networks in various ways, which are described in the text. Chapter 3 deals with graph neural networks, which are neural networks that can process arbitrary graphs. Here, the domain knowledge can be encoded by making the input structured in the form of a graph. Finally, Chapter 4 deals with the removal of symmetries in computer vision. Symmetries represent transformations of the input with respect to which should the model be invariant. By removing the known symmetries in the given domain, we may significantly simplify the task of the model. Each chapter contains an overview of a given topic together with a high-level description of our published work. The individual publications can be found in the appendix 4.9.

2 Encoding domain knowledge using generative models

2.1 Introduction

This chapter is devoted to generative models which represent a very useful tool for encoding domain knowledge in machine learning applications. We start by explaining the difference between generative and discriminative models. Next, we explain how game engines and other simulators could be used as generative models for the purposes of computer vision. We describe three different scenarios of how such generative models could be used: as generators of synthetic datasets, as a virtual playground for reinforcement and active learning algorithms, and lastly, how they can be used as a part of the inference procedure. We also look at manually designed generative models from the point of view of bias-variance trade-off.

Then, we explain how the inference procedure may be realized for complex and realistic generative models for which exact inference is intractable. Finally, we describe techniques that can be used to diminish the gap between synthetic and real data.

This chapter describes general ideas which are useful when we want to leverage simulators for the purpose of computer vision applications. Most of these ideas are applicable in other domains where simulators are being developed using an understanding of a given phenomenon. Throughout the text, we introduce our work in this direction, which is focused specifically on synthetic datasets in computer vision.

2.2 Discriminative vs generative models

A very useful ingredient that enables a successful combination of model-driven approaches with the data-driven one is a generative model. A generative model is a model that describes how observable data is generated, most of the time, by some causal processes. Such generative models are often probabilistic and capture the uncertainty of the model. In real applications, this is often crucial because we do not fully understand the given phenomena.

Formally, such a model specifies a probability distribution of a multivariate random variable (sometimes called observables), which takes values in the space of possible measurements and which will be denoted by $\mathbf{X} = (X_1, ..., X_n)$ in this chapter. We can use such a model to sample a datapoint $\mathbf{x}$ which is given by concrete values of the random variable $\mathbf{X}$ or to evaluate its probability $P(\mathbf{X} = \mathbf{x})$. Most of the time, our model contains also latent random variables $\mathbf{Y} = (Y_1, ..., Y_m)$ which are not directly measurable. Such models are called *latent variable models* and specify the joint probability distribution $P(\mathbf{X}, \mathbf{Y})$. *Factor analysis* is a well-known and representative example of latent variable models.

This goes in contrast with discriminative models, sometimes called recognition models, which only specify a conditional distribution $P(\mathbf{Y} \mid \mathbf{X})$ and do not say anything about the distribution of observed data $P(\mathbf{X})$. Here, a representative example would be a logistic regression or a feed-forward neural network. A schema depict-

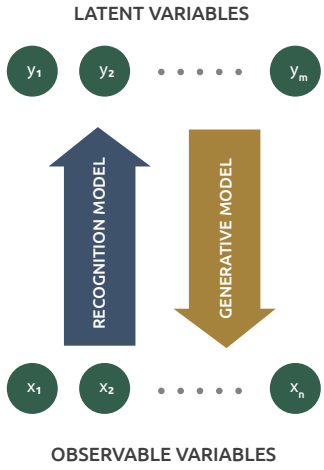


Figure 1: A schema depicting a difference between generative and discriminative/recognition models. Generative models specify the joint distribution of observable and latent variables while discriminative models specify a conditional distribution of latent variables given the observables.

ing the conceptual difference between the generative and discriminative/recognition model is shown in Figure 1.

Mostly, it is much easier to encode domain knowledge using generative models, also called forward models, than using discriminative models. That is the case because our domain knowledge usually consists of facts about abstract, directly unobservable entities in the world (which can be expressed in the model specifying $P(\mathbf{Y})$) and facts about how the world is captured by our senses or measurement devices (expressed by a model which specifies $P(\mathbf{X} \mid \mathbf{Y})$). It does not consist of facts about how we recover the hidden causes from our senses or measurement devices, which would be encoded by a model specifying $P(\mathbf{Y} \mid \mathbf{X})$. In other words, we understand the world through causal mechanisms. This idea is illustrated in Figure 2.

Using a generative model, we can recover the distribution $P(\mathbf{Y} \mid \mathbf{X})$ by inverting it with the help of the *Bayes rule*:

$$P(\mathbf{Y} \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid \mathbf{Y}) \cdot P(\mathbf{Y})}{P(\mathbf{X})}.$$

$P(\mathbf{Y} \mid \mathbf{X})$ is called *posterior distribution*, $P(\mathbf{X} \mid \mathbf{Y})$ is called a *likelihood*, $P(\mathbf{Y})$ is called a *prior distribution* and $P(\mathbf{X})$ is called *evidence*.

As described later, inverting a forward model is usually computationally intractable for realistic scenarios. Therefore, we need to do it approximately. Whereas the current trends of machine learning mostly favor discriminative models (with supervised deep learning methods leading the way), most of the sciences prefer the generative ones because they enable to directly express various theories or facts of the given field.

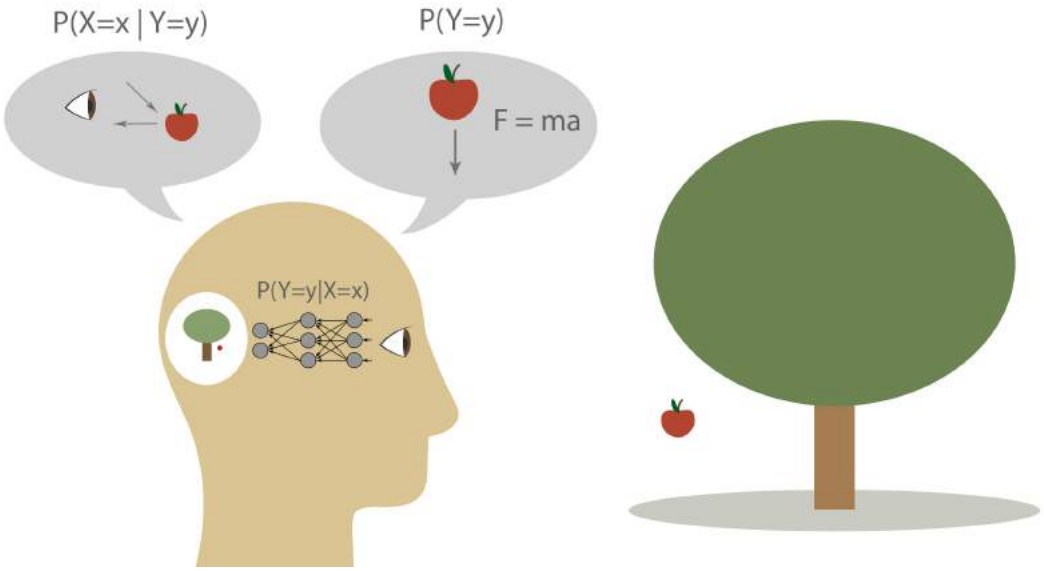


Figure 2: An illustration of the idea that our domain knowledge usually consists of facts describing the generative model and not the discriminative one. In this example, our knowledge may explain why certain patterns of light appear on our retina/sensors but we have no idea how we recover the state of the world from these projected patterns.

2.3 The role of simulators and synthetic datasets

A notable example of generative models are simulators which simulate some real-world phenomena. Computer simulation has been a part of scientific research for a long time [8], from the simulation of large-scale phenomena such as weather to the simulation of small-scale phenomena such as interacting particles. Today, simulators are an essential part of software packages such as Matlab. For our purposes, simulators will be used as forward/generative models that we will want to invert to estimate the latent variables that produced the simulator's output.

The purpose of the simulator dictates the simulation fidelity, which in turn dictates its performance (how long it takes to create the simulation). In this chapter, we will focus on simulators that produce visual outputs simulating photographs or videos, but we note that similar usage can be achieved in other domains.

We aim to use them for various tasks in computer vision (e.g., image classification, object detection, etc.) where it is not essential to have a physically accurate simulation of light, but only a crude approximation of it. This leads to much faster simulation times. It turns out that such simulators have been developed for computer games where it is desirable that the imagery is highly photorealistic and, at the same time, is rendered in real-time. A lot of effort and resources have been put into the creation of these simulators, usually called game engines. Many of them are available as open-source software and therefore can be repurposed for various other goals.

We can repurpose game engines for computer vision purposes in three obvious ways. One way is to use them to generate synthetic training data for classical

learning algorithms such as *convolutional neural networks*. The second way is to use them as virtual playgrounds for reinforcement learning agents and the third way is to use them, as mentioned above, as forward models that will be inverted during inference to recover the latent variables. The following three sections describe these three scenarios.

2.4 Using simulators for the generation of synthetic datasets

This way of using simulators is the most straightforward one because it separates the simulator from the machine learning algorithm. The simulator is used only to provide the training data. It is well-known that the performance of machine learning algorithms is mostly dependent on the amount of data on which it is trained and that collecting a large labeled dataset is expensive. Many machine learning practitioners either use various kinds of data augmentations to enlarge the training set or create synthetic datasets. For this purpose, Hula et al. [9] created a generic framework (described below) built on top of a game engine to streamline the creation of synthetic datasets.

There are two main advantages of using such a framework. Firstly, it can provide automatic labels for every pixel in the generated image because it has a complete control over the generated images. This capability may be indispensable for tasks such as computing an *optical flow* in a video, where the goal is to match the corresponding pixels in adjacent frames. To manually create labels for this task may be practically impossible, whereas using this framework, we can generate tens of thousands of image pairs with the correct matching automatically in a very short amount of time.

For classical computer vision tasks such as image classification, object detection, or semantic segmentation, such frameworks for generating synthetic datasets may be understood as a powerful way of data augmentation.

Computer vision algorithms should be invariant with respect to all kinds of symmetries, such as translations, rotations, or a change of lighting. Classical data augmentations for images can simulate only a few of these symmetries (e.g., translations and rotations in 2D) but cannot, for example, simulate a change of lighting or apply an arbitrary 3D transformation to individual objects. All these transformations are possible using the above-mentioned framework.

If we have 3D models of the objects we are interested in, these objects can be combined in many ways by varying their position, orientation, their material properties or textures, and finally, by applying different lighting conditions to the whole scene. Therefore, from only a few examples, a combinatorially large labeled dataset can be generated with no human input.

The second advantage of synthetic datasets is that they allow us to conduct computational experiments in a tightly controlled fashion. This is especially useful for deep learning models where it often is not clear whether poor performance should be blamed on the model itself or the data it was trained on. Generating training data in a controlled fashion allows us to systematically test the performance of a model on increasingly complex datasets and watch where the model is failing. Therefore, such synthetic datasets can function as an efficient debugging tool for machine learning

Figure 3: The comparison of a scene rendered using Unreal Engine (left) and the same scene rendered using a path tracing renderer (right). The images were rendered in the same resolution (1600 × 1024 px). The left image was rendered under 100 ms, the right one took around 20 minutes to render.

models.

2.4.1 Synthetic data generating framework

In this section, we describe a framework developed by Hula et al., which was presented in their publication [9]. We only describe the high-level idea, the particular details are described in the original publication.

The framework is built on top of Unreal game engine[1], which is able to produce highly photorealistic images in real-time. Ignoring the time of loading 3D models from a hard disk, the framework can render around 15 images per second and, therefore, 1 million images in less than a day. In Figure 3, we show a comparison of an image quality rendered using Unreal Engine and a path tracing renderer, which simulates the behavior of light more accurately but with much higher rendering times.

Along with rendered images, the framework can also produce labels for semantic segmentation, edge detection, normal maps which reflect the orientation of a 3D surface, and depth maps which reflect the distance between the surface and the camera. These outputs are depicted in Figure 4. As a practical example, this framework was used to construct a benchmark for a competition on geometrical edge detection[2]. A sample from this benchmark is depicted in Figure 5.

The framework takes as an input a configuration file from which it assembles a 3D scene. This scene is, in turn, rendered into a 2D image and saved together with the required labels. The configuration file contains an array of objects, an array of lights, and an array of cameras. Each element in each of these arrays is specified by various properties. For objects, these properties include the 3D geometry of the object, its material properties and textures, its size, location, and orientation.

An image showing different materials applied to the same object is depicted in

[1]We used a custom version of Unreal Engine, which uses Voxel Cone Tracing technology for rendering (https://developer.nvidia.com/unrealengine). Using this version, we could achieve a much higher frame rate than with the original version of the engine.

[2]`https://irafm.osu.cz/edge2017/main.php`

Figure 4: An example of ground truth images. Top left: surface normals, Top right: Object ID, Bottom left: surface edges, Bottom right: depth map.

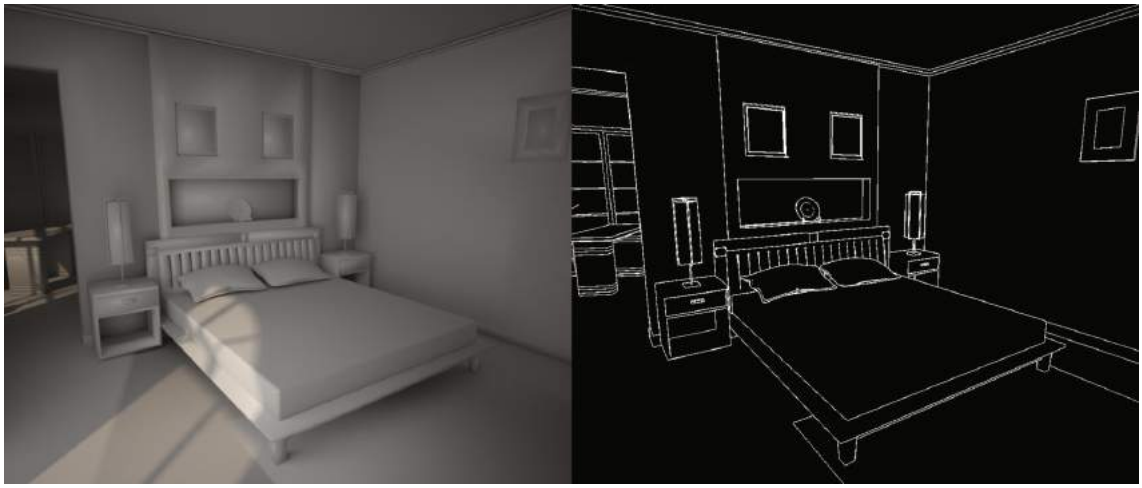


Figure 5: An example training pair from a benchmark for geometrical edge detection.

Figure 6: LEFT: An image of a 3D model composed with a background captured by a real camera. RIGHT: A 3D model of a coffee mill with different materials applied.

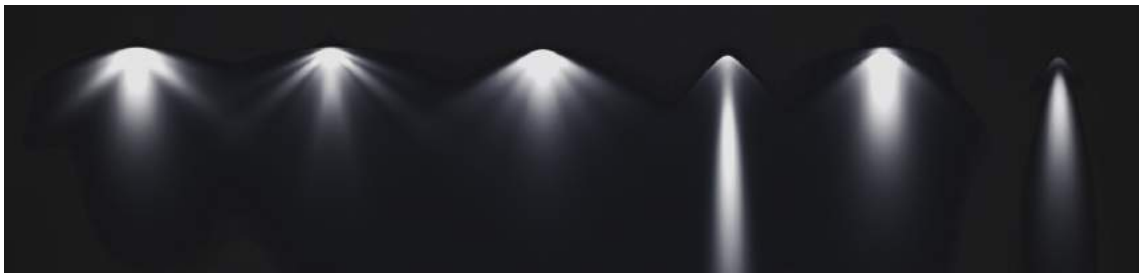


Figure 7: Examples of various IES profiles of lights.

Figure 6. Lights are specified by their position and by IES profiles, which is an industry standard for specifying the luminance properties of light bulbs. These allow us to create complex lighting effects, as depicted in Figure 7. Lastly, cameras are specified by their location, orientation, and various properties, which simulate the behavior of real cameras.

We also show that it is possible to create even more photorealistic images by placing 3D objects into a video by matching the perspectives of the real and the synthetic camera. A video with such augmentation can be found at the following URL: `https://www.youtube.com/watch?v=lUtrQ3kV3AE`. A snapshot from this video can be seen in Figure 6. This can be a beneficial augmentation for unbalanced datasets, where certain classes are underrepresented. For example, in a dataset for self-driving cars, we may have a class "animal", but the dataset may contain very few images capturing an animal standing on the road. Using this augmentation, we may insert 3D models of animals into videos capturing a road and, in this way, balance the presence of this class within the dataset. We also mention that such synthetic data generating frameworks were recently developed by organizations such as NVIDIA or Microsoft, which can devote much larger resources to such projects. A notable example is a project called *Fake It Till You Make It* [10] where the authors trained a neural network to detect keypoints on a human face with a very high accuracy solely on synthetic images.

2.4.2 Synthetic datasets for compositional learning

By using the framework described above, Hula and Molek [11] demonstrated how it could be used for what they call *compositional learning*. Compositional learning leverages the fact that the signal we usually deal with has a compositional structure. This means that it is composed of recurring parts that are, in turn, composed of other parts and so on. Taking photographs as an example, they usually contain independent objects composed of parts that can again have subparts. It is also well-known that convolutional neural networks trained on real images become sensitive to these parts [12], as depicted in Figure 10. Because these local parts are indicative of the objects that contain them (for example, a wheel is indicative of a car), some convolutional filters learn to be sensitive to them. Using this insight, we may explicitly force certain filters to be sensitive to parts, which we believe are essential for recognition of the desired class.

For example, when we try to train a classifier for different species of birds, we may use the fact that the shape of the beak or some colored pattern on the wings is a robustly discriminative feature for the classification of a certain class of birds. We may use this type of domain knowledge and explicitly force certain convolutional filters to be sensitive to these features. This can be done by adding a binary classifier on top of the feature maps at the convolutional layer with a resolution that matches the size of the discriminative feature in the image[3]. This binary classifier is added at every position of the output of the convolutional layer and is trained to classify whether the feature is present or not. This forces the particular convolutional filter to be sensitive to this feature. For clarity, we illustrate this idea in Figure 8.

For this type of training, the training set has to contain labels for these discriminative features alongside the labels for objects. Marking these features manually on each image would be time-consuming, but using the framework described above, we only need to mark the features on the 3D model of the object, and the labels for each rendered image will be generated automatically as depicted in Figure 9.

In an experiment, testing the benefits of compositional learning, the model trained with compositional learning converged approximately five times faster to the same accuracy as the model, which was not trained using compositional learning. The dataset contained synthetic images of different kinds of plants and the labels for subparts corresponding to individual leaves, petals, and other parts of the plant. The creation of this dataset was described in our contribution [11].

2.5 Simulators as a virtual playground for active learning and reinforcement learning agents

Previous sections described synthetic datasets that are generated before training and remain fixed during the training procedure. Some applications may nevertheless require the learning algorithm to interact with the simulator. In this scenario, there is no fixed dataset, and the training examples are generated dynamically on the fly. Two settings that correspond to these requirements are *active learning* [14] and

[3]The size of the feature in the image should fit within a receptive field of one neuron in this convolutional layer

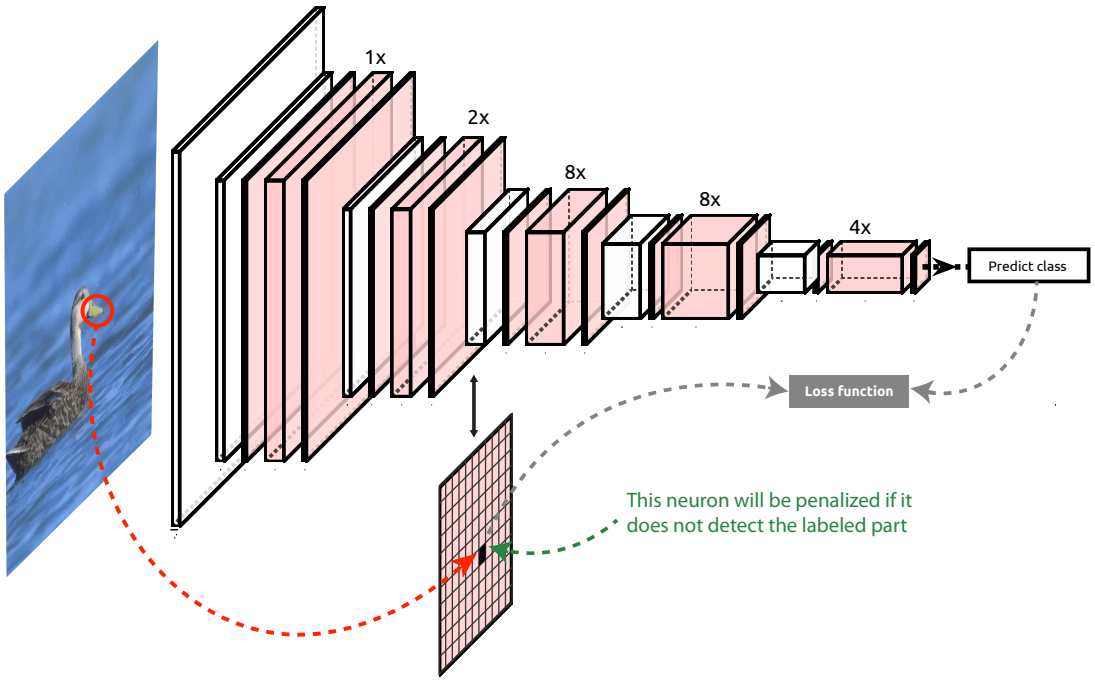


Figure 8: An illustration of a compositional learning. Using labels for certain subparts of a given object, we can penalize certain convolutional layers for not detecting these subparts.

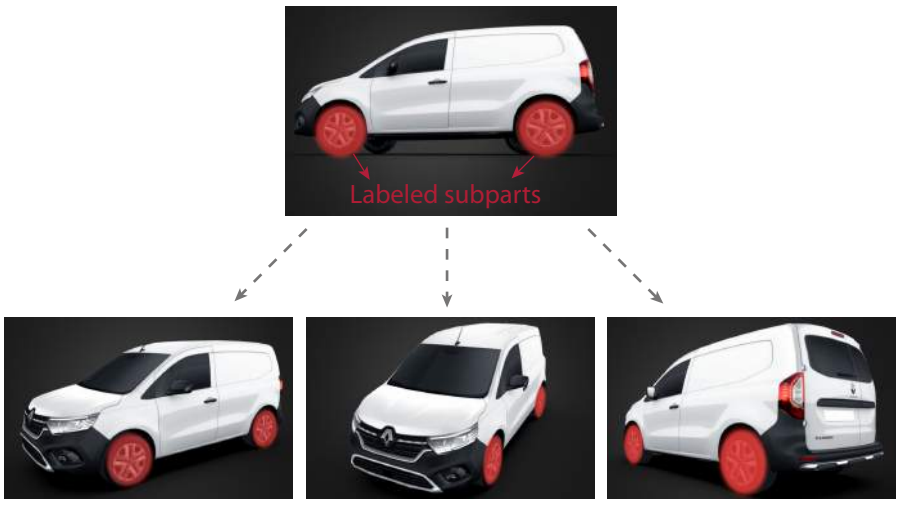


Figure 9: Labeling the subparts (wheels in this case) on the 3D model allows us to create many images with the subparts labeled automatically.

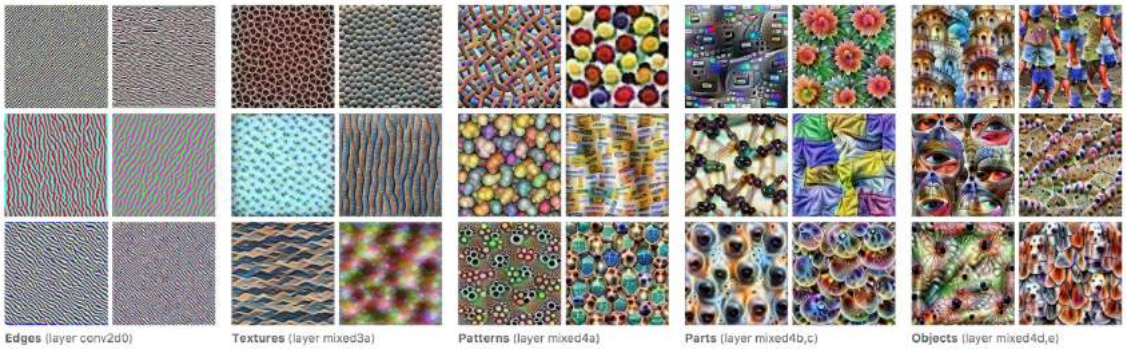


Figure 10: A feature visualization showing patterns that maximally excite certain neurons in different layers of a convolutional neural network. We can see that filters in convolutional layers detect meaningful and recognizable parts. Taken from distill.pub [13].

reinforcement learning.

In active learning, the learning algorithm may interactively query the teacher to label new data points. In classical supervised learning, there may be many redundant training examples within the dataset, which do not provide any new information to the learning algorithm. The goal of active learning is to let the learning algorithm query the labels for images that would be most informative for finding the right decision boundary.

In reinforcement learning, the learning agent continuously interacts with the environment and learns a policy by a trial and error process using a sparse reward signal (e.g., the rewards are given only in some interactions).

Both of these settings can benefit from simulators, which in the case of active learning substitutes a person providing labels and in the case of reinforcement learning substitutes the real environment. For example, in robotics applications, the robot may learn its motoric behavior in a virtual environment that simulates real physics and then fine-tune it in the real environment with real physics. Problems that arise because of the mismatch of real and synthetic environments will be described in the Section 2.7.

This use of simulators may be viewed as a middle ground between the usage of simulators for the generation of static datasets and their usage during inference to recover the latent variables. If we anthropomorphize this situation to make an analogy, using the simulator to create a learning environment for the agent can be compared to the use of simulators for the training of pilots. Using the simulator during inference to recover the latent variables can be compared to a situation in which a person is using his mental model of some phenomena to find a cause of some observation. This latter case will be described in the following section.

2.6 Simulators as forward models which are inverted during inference

This use of a simulator is more intricate than the other two described in the previous two sections because here, the simulator tightly interacts with the learning

algorithm. Inference[4] is understood here as an inversion of the simulator/generative model. In machine learning, this view has a very long history [15, 16], especially in Bayesian approaches where the inversion is realized by the application of the Bayes rule. It is sometimes known under the slogan "Analysis by synthesis" [17] or, in computer vision, by "Computer vision as inverse computer graphics" [18]. It also has a long tradition in cognitive science, where it is used to explain perception as an inversion of our mental model of the environment [19], which is a view initiated by Helmholtz already in the 19th century [20].

As mentioned in the first chapter, incorporating the simulator into the machine learning algorithm lets us directly encode the domain knowledge into the algorithm. In a way, the simulator represents constraints on the vector of latent variables. Given the image $\mathbf{x}$, the vector of latent variables $\mathbf{y}$ should be such that the image $\mathbf{x'}$ generated from this vector is the same or similar to the original image. If we define a function s which measures the similarity between two images, we can use it in place of the likelihood of the original image given the latent variables y, denoted by $P(\mathbf{X} = \mathbf{x} \mid \mathbf{Y} = \mathbf{y})$ (In the following text, we will use the notation $P(\mathbf{x})$ to stand for $P(\mathbf{X} = \mathbf{x})$.).

To simplify the discussion, let us neglect the fact that the simulator could be stochastic and let also assume that the vector $\mathbf{y}$ contains only continuous variables. Then we can treat the generator as a function g which takes the vector of continuous latent variables $\mathbf{y}$ as an input and outputs the image $\mathbf{x}' = g(\mathbf{y})$. We can define the unnormalized likelihood by $P(\mathbf{x} \mid \mathbf{y}) \sim s(\mathbf{x}, g(\mathbf{y}))$.

Our goal is to solve the inverse problem, which in this context means to estimate the posterior distribution $P(\mathbf{y} \mid \mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{y}) \cdot P(\mathbf{y})}{P(\mathbf{x})}$ or often to solve only the *maximum a posteriori* (MAP) problem which means to find the most probable vector $\mathbf{y} = \text{argmax}_y P(\mathbf{y} \mid \mathbf{x})$. If we further assume that the prior distribution $P(\mathbf{y})$ is uniform, which means that we have no preference for any $\mathbf{y}$, we can solve the MAP problem as an optimization over $\mathbf{y}$:

$$\text{argmax}_y P(\mathbf{y} \mid \mathbf{x}) = \text{argmax}_y P(\mathbf{x} \mid \mathbf{y}) \cdot P(\mathbf{y}) = \text{argmax}_y P(\mathbf{x} \mid \mathbf{y}) = \text{argmax}_y s(\mathbf{x}, g(\mathbf{y})).$$

If the inverse problem is ill-posed, which could, for example, mean that there are more solutions to this optimization problem, we can utilize the prior distribution $P(\mathbf{y})$ that can make certain solutions more probable than others. The prior can be also understood as a regularization term that penalizes certain solutions more than others. For complex simulators, this optimization problem will be highly non-convex. It may also be non-differentiable if the simulator g contains discrete decisions[5].

We have multiple choices on how to approach it. The simplest one is to use a gradient-free black-box optimizer such as CMA-ES [21]. To make the search less computationally demanding, we could also use a surrogate model of the simulator as was done, for example, by Munk et al. [22]. We can also train another function f which takes the input image $\mathbf{x}$ as input and produces a vector $\mathbf{y}$ which can serve as

[4] By the word inference we mean the process of obtaining the distribution or the most probable value of latent variables $\mathbf{y}$ conditioned on the observed value $\mathbf{x}$.

[5] Recently, differentiable simulators have been proposed, which enable to compute the gradients of the input parameters $\mathbf{y}$ with respect to the output of the simulator. In this case, gradient-based optimizers can be used.

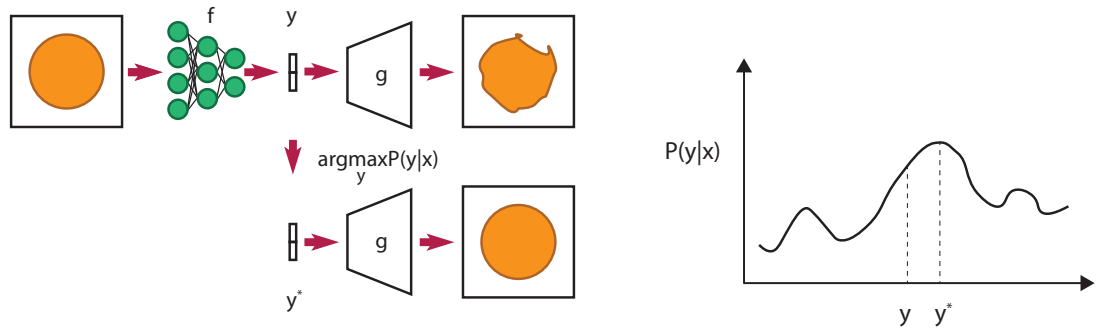


Figure 11: A schematic illustration of the idea of a learned initializer f. Given the image, the function f predicts the latent vector $\mathbf{y}$ which is then optimized to find the best reconstruction $\mathbf{y}^*$ using the simulator g.

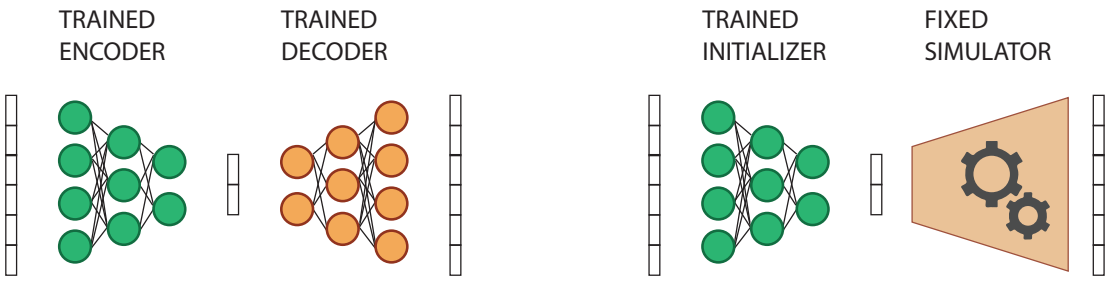


Figure 12: A comparison of a variational autoencoder and a learned initializer with a fixed manually designed simulator.

initialization for the optimizer that in turn fine-tunes the vector $\mathbf{y}$ to find a better MAP estimate $\mathbf{y}^*$. This idea is depicted in Figure 11.

When we use the function f for the initialization, the whole framework resembles the *variational autoencoder* [23] where f plays the role of the encoder and g of the decoder. This is depicted in Figure 12.

In the variational autoencoder, the generator g and the function f are trained jointly, and in the case described above, the generator is fixed. Nevertheless, we emphasize the fact that the problem of MAP inference, which means to find $\text{argmax}_y P(\mathbf{y} \mid \mathbf{x})$, could be realized simply as a search for the best $\mathbf{y}$ for which no training may be necessary. The inference is simply an inversion of the generative model/simulator which is designed by the domain expert.

In most cases, the simulator may still contain parameters we are uncertain about and these may be estimated from data. For example, in the case of the generator of human faces, we may want to estimate the distribution of the size of the nose, given the ethnicity of the person, and we may believe that the distribution is normal with an unknown mean and variance. The estimation can be done, for example, by using the *maximum likelihood principle* where we adjust parameters of the probabilistic model (in this case, the mean and variance) to maximize the likelihood of the data given the model. In comparison to standardly used neural networks, there would be only a few of such trainable parameters, and therefore we would need only a

few training samples. The number of training parameters will allow us to trade the flexibility of the model with the required number of training samples. This can be well understood through the bias-variance tradeoff.

2.6.1 Bias-variance tradeoff

The bias-variance tradeoff analyzes the source of the generalization error of a model on data outside of the training set. Here we describe it for the case of Least Squares Regression, which is the setting where it was originally developed.

We assume that we have a training dataset $D \coloneqq \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, drawn *i.i.d.*[6] from data generating distribution $P(\mathbf{X}, Y)$ where $\mathbf{x}_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$. For Least Squares Regression we assume that the relationship between $\mathbf{x}_i$ and y_i can be explained by $y = f(\mathbf{x}) + \varepsilon$ where $f : \mathbb{R}^n \to \mathbb{R}$ is some unknown function and ε is Gaussian noise with zero mean and variance σ^2.

We would like to find a function $\hat{f}$ which is the best approximation of f from a family of functions belonging to some hypothesis space parametrized by $\phi \in \Phi$. By best, we mean the function with a lowest mean square error:

$$Err(f_\phi) \coloneqq E_{\mathbf{x},y\sim P(\mathbf{X},Y)}\left[(f_\phi(\mathbf{x}) - y)^2\right], \tag{1}$$

where the expectation is taken over the data generating distribution $P(\mathbf{X}, Y)$. In practice, we cannot find this function because we do not know $P(\mathbf{X}, Y)$. Instead, we try to find a function f_{ϕ^D} parameterized by ϕ^D which minimizes the empirical loss function on the training set D:

$$\phi^D \coloneqq \underset{\phi\in\Phi}{\operatorname{argmin}} \frac{1}{m} \sum_{(x_i,y_i)\in D} (f_\phi(\mathbf{x_i}) - y_i)^2. \tag{2}$$

We can view f_{ϕ^D} as a random variable because it is a function of another random variable, which is D. If we sample different $D \sim P(\mathbf{X}, Y)^m$ we may obtain different f_{ϕ^D}. We denote this dependence by $f_{\phi^D} \coloneqq \mathcal{A}(D)$ where $\mathcal{A}$ is a learning algorithm implementing a minimization mentioned in Definition 2. We can write down its expectation over all possible datasets sampled from $P(\mathbf{X}, Y)^m$:

$$\bar{f_{\phi^D}}(\mathbf{x}) \coloneqq E_{D\sim P(\mathbf{X},Y)^m}\left[f_{\phi^D}(\mathbf{x})\right]. \tag{3}$$

Using this, we can analyze the expected error of the function obtained by the learning algorithm, which searches over the hypothesis space parametrized by $\phi \in \Phi$ and uses a randomly sampled dataset D for training. This can be written as:

$$\begin{aligned} Err(\Phi) &\coloneqq E_{D\sim P(\mathbf{X},Y)^m}\left[E_{\mathbf{x},y\sim P(\mathbf{X},Y)}\left[\left(f_{\phi^D}(\mathbf{x}) - y\right)^2\right]\right] \\ &= E_{D\sim P(\mathbf{X},Y)^m}\left[Err(f_{\phi^D})\right]. \end{aligned} \tag{4}$$

We emphasize that this does not quantify an error for any concrete realization f_{ϕ^D} but an **expected** error for the whole hypothesis space parametrized by Φ. We

[6] Independently and identically distributed.

can analyze this error by decomposing it into meaningful terms. In the following, we will simplify the notation for the sake of readability by using $E_{\mathbf{x},y,D}[...]$ for $E_{D\sim P(\mathbf{X},Y)^m}\left[E_{\mathbf{x},y\sim P(\mathbf{X},Y)}[...]\right]$:

$$\begin{aligned}
E_{\mathbf{x},y,D}\left[\left(f_{\phi^D}(\mathbf{x})-y\right)^2\right] &= E_{\mathbf{x},y,D}\left[\left(\underbrace{\left(f_{\phi^D}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x})\right)}_{\text{a}}+\underbrace{\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)}_{\text{b}}\right)^2\right] \\
&= \underbrace{E_{\mathbf{x},D}\left[(f_{\phi^D}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x}))^2\right]}_{\text{a}^2} \\
&+ \underbrace{2\,E_{\mathbf{x},y,D}\left[\left(f_{\phi^D}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x})\right)\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)\right]}_{\text{2ab}} \\
&+ \underbrace{E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)^2\right]}_{\text{b}^2}
\end{aligned}$$

The term $2ab$ is equal to zero as shown below:

$$\begin{aligned}
E_{\mathbf{x},y,D}\left[\left(f_{\phi^D}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x})\right)\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)\right] &= E_{\mathbf{x},y}\left[E_D\left[f_{\phi^D}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x})\right]\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)\right] \\
&= E_{\mathbf{x},y}\left[\left(E_D\left[f_{\phi^D}(\mathbf{x})\right]-\bar{f_{\phi^D}}(\mathbf{x})\right)\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)\right] \\
&= E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x})\right)\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)\right] \\
&= E_{\mathbf{x},y}\left[0\right]. \\
&= 0
\end{aligned}$$

We therefore have:

$$E_{\mathbf{x},y,D}\left[\left(f_{\phi^D}(\mathbf{x})-y\right)^2\right] = E_{\mathbf{x},D}\left[(f_{\phi^D}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x}))^2\right]+E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)^2\right]. \quad (5)$$

The second term can be expanded further by introducing the unknown function f:

$$\begin{aligned}
E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-y\right)^2\right] &= E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-f(\mathbf{x})\right)+\left(f(\mathbf{x})-y\right)^2\right] \quad (6) \\
&= \underbrace{E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-f(\mathbf{x})\right)\right)^2\right]}_{\text{a}^2} \\
&+ \underbrace{2\,E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-f(\mathbf{x})\right)\left(f(\mathbf{x})-y\right)\right]}_{\text{2ab}} \\
&+ \underbrace{E_{\mathbf{x}}\left[\left(f(\mathbf{x})-y\right)^2\right]}_{\text{b}^2}
\end{aligned}$$

The term $2ab$ is again zero:

$$\begin{aligned}
E_{\mathbf{x},y}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})-f(\mathbf{x})\right)\left(f(\mathbf{x})-y\right)\right] &= E_{\mathbf{x}}\left[E_{y|\mathbf{x}}\left[f(\mathbf{x})-y\right]\left(\bar{f_{\phi^D}}(\mathbf{x})-f(\mathbf{x})\right)\right] \\
&= E_{\mathbf{x}}\left[\left(f(\mathbf{x})-E_{y|\mathbf{x}}\left[y\right]\right)\left(\bar{f_{\phi^D}}(\mathbf{x})-f(\mathbf{x})\right)\right] \\
&= E_{\mathbf{x}}\left[\left(f(\mathbf{x})-f(\mathbf{x})\right)\left(\bar{f_{\phi^D}}(\mathbf{x})-f(\mathbf{x})\right)\right] \\
&= E_{\mathbf{x}}\left[0\right] \\
&= 0.
\end{aligned}$$

Therefore, if we plug Equation 6 into 5, we obtain the final form of the bias-variance trade-off equation:

$$\underbrace{E_{\mathbf{x},y,D}\left[\left(f_{\phi^D}(\mathbf{x})-y\right)^2\right]}_{\text{Expected Test Error}} = \underbrace{E_{\mathbf{x},D}\left[\left(f_{\phi^D}(\mathbf{x})-\bar{f_{\phi^D}}(\mathbf{x})\right)^2\right]}_{\text{Variance}} + \underbrace{E_{\mathbf{x}}\left[\left(\bar{f_{\phi^D}}(\mathbf{x})\right)-f(\mathbf{x})\right)^2\right]}_{\text{Bias}^2} + \underbrace{E_{\mathbf{x},y}\left[(f(\mathbf{x})-y)^2\right]}_{\text{Noise}} \tag{7}$$

In the third term, notice that f was defined by $y = f(\mathbf{x}) + \varepsilon$ and therefore the third term reduces to σ^2 which is the irreducible error. The first term is called variance and tells us how much the error will vary when we vary the training dataset D. The second term is called bias (squared) and tells us how well our model is able to approximate the unknown function f in expectation. A model with a very high variance may produce very different functions when trained on different datasets $D \sim P(\mathbf{X}, Y)^m$, and a model with a high bias will not be able to approximate the unknown function well enough, no matter how many training examples will be used for training. If we train a model with a high variance on a larger dataset, the expected error should decrease.

With this analysis of the bias-variance tradeoff, we can reason about the hypothesis space of our model. Ideally, it should be as restricted as possible and therefore have low variance, and at the same time, it should contain functions that approximate the unknown function f well enough. Machine learning researchers use the term inductive bias to indicate that the model is biased towards functions with some specific properties. For example, convolutional neural networks have an inductive bias for translation invariant functions.

From this point of view, purely data-driven approaches, such as the ones using over-parametrized deep neural networks have a very high variance and low bias because they can approximate many different functions. On the other hand, model-driven approaches usually have a high bias because of the wrong assumptions, which may not be reflected in the unknown function behind the data. Ideally, we would like to encode the assumptions which we are sure about into the model and learn the rest from data. Such combinations will be described in Section 2.7.

2.6.2 Simulators as probabilistic programs

Typically, the posterior inference is studied with probabilistic generative models which are encoded by *probabilistic graphical models*. These consist of a graph with nodes corresponding to variables and edges corresponding to relationships between these variables. Two main families of probabilistic graphical models consist of *Bayesian networks* that encode the relationship between variables $\mathbf{x}$ and $\mathbf{y}$ by a conditional probability distribution $P(\mathbf{x} \mid \mathbf{y})$ and *Markov random fields*, which encode the relationship by joint probability distribution $P(\mathbf{x}, \mathbf{y})$.

Recently, more expressive languages for probabilistic generative models have been developed. They allow us to succinctly encode very complex distributions by

probabilistic programs containing loops and other control structures. They also contain random variables that can be sampled during the execution of the program. The output of such a program can be understood as the state of the observable variables $\mathbf{X}$. The set of random variables sampled during the execution of the program can be understood as the set of latent variables $\mathbf{Y}$. The difference between probabilistic programs and classical stochastic simulators, which are present in software packages such as Matlab, is that we can condition the probabilistic program on a particular output $\mathbf{x}$ and produce posterior inference over the latent variables $\mathbf{Y}$. That is, the the probabilistic program automatically comes with an inference procedure.

There is an apt analogy between different encodings of truth functions and probability distributions. Truth functions can be encoded by truth tables, propositional formulae, or formulae in predicate logic. These are ordered from less expressive to more expressive ways to encode truth functions. Likewise, probability distributions can be encoded by probability tables, probabilistic graphical models, or probabilistic programming languages, which are the most expressive of these three.

Programs written in a probabilistic programming language can, for example, realize a rendering engine that produces realistic images and we can condition the program on a real image $\mathbf{x}$ to estimate the probability distribution of the latent variables $\mathbf{Y}$ given the image $\mathbf{x}$. This was done, for example, by Kulkarni et al. [24]. Recently, Baydin et al. [25] introduced a generic framework that enables probabilistic inference for simulators that have not been developed with probabilistic inference in mind (such as the game engine used in our work). For a general overview of probabilistic programming languages see [26].

2.6.3 Approximate inference

For now, let us assume that we have at our disposal the desired generative model/simulator of the data, and we want to use it for the posterior inference of the unobserved latent variables. If we have the whole posterior distribution, as opposed to only the most probable value, we can sample from it to obtain a concrete realization of the latent variables $\mathbf{y}$. In this case, we do not need to commit ourselves to a particular value $\mathbf{y}$, but we can account for multiple possible hypotheses.

Unfortunately, for complex and realistic generative models such as the ones given by probabilistic programs, it is practically infeasible to compute the posterior distribution exactly. The problem arises in the computation of the normalization constant $P(\mathbf{x})$ in the denominator of $P(\mathbf{y} \mid \mathbf{x}) = \frac{P(\mathbf{x}|\mathbf{y})\cdot P(\mathbf{y})}{P(\mathbf{x})} = \frac{P(\mathbf{x}|\mathbf{y})\cdot P(\mathbf{y})}{\sum_{\mathbf{y}} P(\mathbf{x}|\mathbf{y})\cdot P(\mathbf{y})}$ where we need to sum over the combinatorially large space of all possible values $\mathbf{y}$. We may, therefore, only hope to find an approximate solution. This line of research is called *approximate inference* [26].

Two main directions studied in this area are *variational methods* which convert the inference problem to an optimization problem, and *Monte Carlo methods* which approximate the solution by sampling.

Variational inference Variational inference is a set of techniques that provide an analytical approximation to the posterior distribution $P(\mathbf{Y} \mid \mathbf{x})$. The aim is to approximate the posterior $P(\mathbf{Y} \mid \mathbf{x})$ with some other distribution $Q(\mathbf{Y})$ which is

restricted to a family of simpler distributions (for example, to the family of normal distributions). We want to find a distribution $Q(\mathbf{Y})$ which is most similar to the distribution $P(\mathbf{Y} \mid \mathbf{x})$. The adjective *variational* points to the connection to *variational calculus* which deals with the minimization of functionals that in this case measure the similarity to the posterior distribution $P(\mathbf{Y} \mid \mathbf{x})$. The similarity between distribution P and Q is measured by a similarity (or rather a dissimilarity) function $d(Q, P)$ such as the *Kullback-Leibler (KL) divergence*:

$$KL(Q(\mathbf{X}) \| P(\mathbf{X})) = \sum_{\mathbf{x}} Q(\mathbf{x}) \log \frac{Q(\mathbf{x})}{P(\mathbf{x})}.$$

The posterior inference is therefore realized as a search for the distribution Q that minimizes $d(Q, P)$. If we use the KL divergence to find the best approximation $Q(\mathbf{Y})$ of $P(\mathbf{Y} \mid \mathbf{x})$ we can obtain the following simplification:

$$\begin{aligned}
KL(Q(\mathbf{Y}) \| P(\mathbf{Y} \mid \mathbf{x})) &= \sum_{\mathbf{y}} Q(\mathbf{y}) \log \frac{Q(\mathbf{y})}{P(\mathbf{y} \mid \mathbf{x})} = \sum_{\mathbf{y}} Q(\mathbf{y}) \log \frac{Q(\mathbf{y}) \cdot P(\mathbf{x})}{P(\mathbf{y}, \mathbf{x})} \\
&= \sum_{\mathbf{y}} Q(\mathbf{y}) \left[\log \frac{Q(\mathbf{y})}{P(\mathbf{y}, \mathbf{x})} + \log P(\mathbf{x}) \right] \\
&= \sum_{\mathbf{y}} Q(\mathbf{y}) \left[\log Q(\mathbf{y}) - \log P(\mathbf{y}, \mathbf{x}) \right] + \sum_{\mathbf{y}} Q(\mathbf{y}) \left[\log P(\mathbf{x}) \right] \\
&= \sum_{\mathbf{y}} Q(\mathbf{y}) \left[\log Q(\mathbf{y}) - \log P(\mathbf{y}, \mathbf{x}) \right] + \log P(\mathbf{x}) \\
&= \mathbb{E}_{\mathbf{Q}} \left[\log Q(\mathbf{y}) - \log P(\mathbf{y}, \mathbf{x}) \right] + \log P(\mathbf{x}).
\end{aligned}$$

Therefore, we obtain:

$$\log P(\mathbf{x}) = KL(Q(\mathbf{Y}) \| P(\mathbf{Y} \mid \mathbf{x})) - \underbrace{\mathbb{E}_{\mathbf{Q}} \left[\log Q(\mathbf{y}) - \log P(\mathbf{y}, \mathbf{x}) \right]}_{L(Q)}$$

The term $\log P(\mathbf{x})$ on the left side is constant, and therefore we know that if we maximize the term $L(Q)$, we will minimize $KL(Q(\mathbf{Y}) \| P(\mathbf{Y} \mid \mathbf{x}))$. $L(Q)$ is sometimes called *evidence lower bound* (or ELBO) because it is a lower bound of the term $\log P(\mathbf{x})$, which is called evidence. By an appropriate choice of the family of distributions Q, the term $L(Q)$ becomes tractable to optimize. Often the family is restricted to distributions Q which fully factorize over individual random variables: $Q(\mathbf{y}) = \prod_i q(y_i)$. This approximation is called *mean field approximation*. If we make the parameters of the distribution $Q(\mathbf{Y})$ explicit, our optimization objective will have the following form:

$$\theta^* = \underset{\theta \in \Theta}{\operatorname{argmax}} L(Q(\mathbf{Y}; \theta)).$$

We can optimize the parameters $\theta \in \Theta$ using the gradient descent or a gradient-free optimizer. If we want to use gradient descent, we need to compute a gradient of $L(Q(\mathbf{Y}; \theta)$, which is an expectation that may not have a simple analytical form. In this case, we can use gradient estimation techniques [27] to obtain an approximate

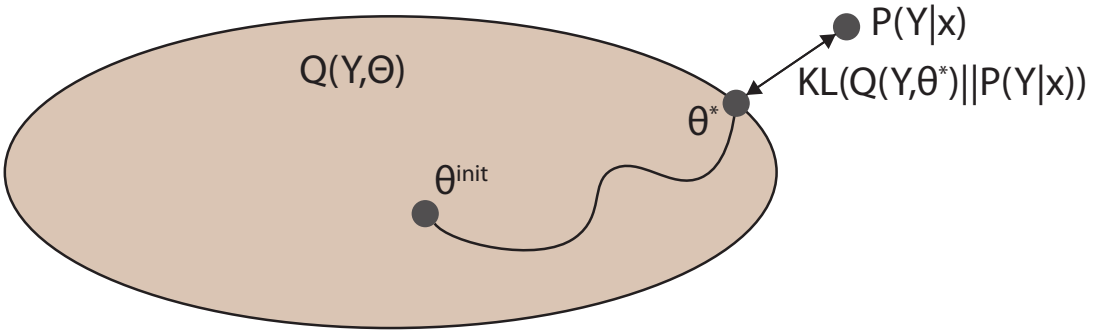


Figure 13: A schematic illustration of the variational inference procedure where the goal is to find the most suitable approximation of the distribution $P(\mathbf{Y} \mid \mathbf{x})$ from a restricted family of distributions $Q(\mathbf{Y}, \Theta)$. The quality of the approximation is here measured by KL-divergence.

gradient using a finite number of samples from $Q(\mathbf{Y}; \theta)$. Such methods are called *black-box variational inference*.

To summarize, we turned the inference problem into an optimization problem whose goal is to find the "best" approximation to the distribution $P(\mathbf{Y} \mid \mathbf{x})$ from a restricted family of simple distributions. This idea is depicted in Figure 13.

the variational inference is a broad subject because different assumptions about the generative model, the distribution Q and different measures of similarity between distributions lead to different optimization problems.

Monte Carlo methods Monte Carlo methods are a broad class of methods that use sampling to estimate quantities that may be intractable to compute exactly. We can use them, for example, to estimate expectations of complex functions $\mathbb{E}_{\mathbf{P}}[f(x)]$. Using a finite set of samples $(x_1, \ldots, x_t) \sim P$ drawn from P, we can estimate the expectation by:

$$\mathbb{E}_{\mathbf{P}}[f(x)] \approx \frac{1}{t} \sum_{i=1}^{t} f(x_i).$$

For some distributions, such as the normal distribution, it may be straightforward to obtain the samples, but for others it is not.

In posterior inference, we are concerned with the sampling from the posterior distribution $P(\mathbf{Y} \mid \mathbf{x})$, but we already mentioned the problems due to the intractability of computing the normalization constant $P(\mathbf{x})$.

This problem is circumvented by *Markov Chain Monte Carlo* (MCMC) algorithms which allow us to obtain samples using unnormalized distributions (without the need to compute the normalization constant $P(\mathbf{x})$). This is achieved by constructing a *Markov chain* whose stationary distribution is the desired distribution from which we do not know how to sample. Markov chain is a sequence of random variables/states $s_1, s_2, \ldots$ which satisfies the Markov property. This property says that the distribution of the state s_{t+1} is fully determined by the state s_t. The whole sequence is specified by one transition operator specifying $P(s_{t+1} \mid s_t)$. It

can be shown that every Markov chain which satisfies a few technical assumptions converges to a stationary distribution.

Markov chain Monte Carlo methods work by designing the transition operator in such a way that the chain converges to the distribution from which we want to sample. Therefore, we can start from some random assignment s_1 and repeatedly apply the transition operator and after several steps t (taking the issue about the rate of convergence aside), the assignment s_t will be a sample that respects the target distribution.

In our case, we want to sample from the posterior distribution $P(\mathbf{Y} \mid \mathbf{x})$. There exist multiple ways how to design the transition operator, and among those, the *Metropolis-Hastings* algorithm is probably the most popular one.

To summarise the section about approximate inference, given an expressive generative model $P(\mathbf{X}, \mathbf{Y})$, it is often intractable to obtain the posterior distribution $P(\mathbf{Y} \mid \mathbf{x})$, which quantifies the uncertainty over the latent variables $\mathbf{Y}$ given the observed variables $\mathbf{x}$. There are two main approaches to solve this problem approximately. Variational methods solve the problem by optimization and Monte Carlo methods by sampling.

2.7 Dealing with the approximation error of the simulator

When we create a simulator of some real-world phenomena, the simulator may not produce an output that would be indistinguishable from reality. In the case of rendering/game engines, the images may not look exactly like photographs. This mismatch between the simulator and reality can cause problems in the inference or learning from the synthetic data. Typically, this depends on the complexity of the domain.

For example, Hula et al. [28] leveraged synthetic data to train a detection model of individual letters in historical printed texts. In this case, the synthetic data looked realistic enough, and a detector trained on this data was applicable to the real data. In other domains, it may not be that straightforward. We may have a simulator that generates synthetic images, but we may want to to infer the latent variables on real images and there could be a noticeable residuum between the synthetic, and real images. The situation is depicted in Figure 14.

2.7.1 Domain adaptation

. This problem could be approached by methods of *domain adaptation* that deal with a situation in which we have a machine learning model trained on one domain and we want to adapt it to a new domain (for example, one domain may correspond to images taken in daylight and the other to images taken in the night). We can either learn a model which maps the real images to the synthetic ones and run the inference for the synthetic image or extend the simulator by a learnt model which takes the output of the simulator as an input and produces a realistic version of that input. This second case can be understood as a kind of superresolution. Both scenarios are depicted in Figure 15 .

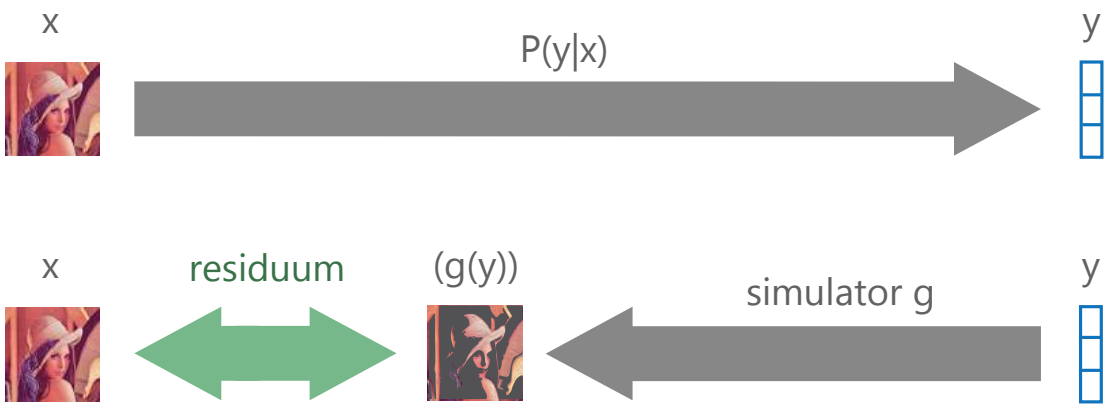


Figure 14: A schema showing the residuum between real and synthetic images. We want to run inference on real images, but the images generated by the simulator may not look photorealistic.

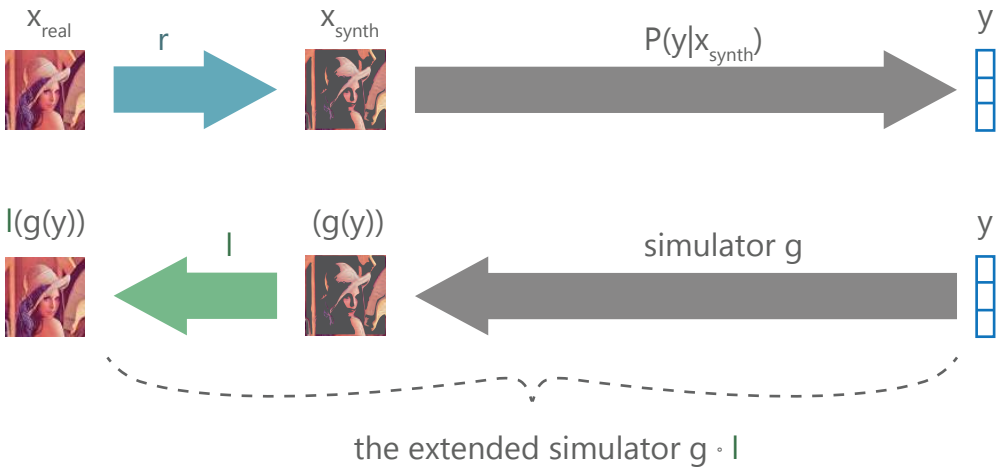


Figure 15: An illustration of the two functions r and l converting real to synthetic images, and synthetic to real images, respectively. If we compose the manually designed simulator g with the learned function l, we obtain an extended simulator that will generate images with a higher degree of photorealism.

The problem with learning these two models l and r is that it would be difficult to obtain training pairs of images where one would be a rendered image, and the other would be a realistic version of that rendered image. Therefore, we would not be able to train these models in a supervised fashion. Nevertheless, these two models can be trained using a *cycle consistency* constraint, an idea popularized by the model called *CycleGAN* [29]. In this setup, we only require having unpaired samples from these two domains (rendered and realistic). The constraint forces the models l and r to satisfy the following two relations:

$$id(x_{synth}) = r(l(x_{synth})), \quad id(x_{real}) = l(r(x_{real})),$$

where id is an identity mapping and x_{synth} and x_{real} is synthetic and real image respectively.

2.7.2 Comparison to generative adversarial networks

We also want to point out the difference between the combination of a manually designed simulator with a model that learns only the residuum and a fully trained generative model, such as Generative adversarial network (GAN) [30], which is a neural network that is trained to transform a Gaussian noise to realistically looking images. When we combine a manually designed simulator with a model that learns the residuum, the learning of this model may be much easier than training a GAN, which needs to learn the full generative process. At the same time, this combination with a manually designed simulator allows us to extrapolate to unseen configurations. For example, a GAN trained on photographs of faces captured from a front view will never generate an photograph of a face captured from the side view, whereas with the combination mentioned above, the extrapolation to the side view should be possible with only a slight degradation of the photorealism. Lastly, the whole model is modifiable and explainable.

2.7.3 Domain randomization

Another problem that needs to be considered is the fact that the data from the simulator may be biased towards certain attributes, and this bias may not hold in the real world. For example, let us imagine that we are generating synthetic images for the application of self-driving cars and it happens that every car in these images is red in color and at the same time no other object is red in color. If we train an object detector on such a dataset, it will probably learn to associate the class "car" with red color and will detect every red object as a car. To remove such spurious correlations, we may randomize the attributes to which we would like to be invariant. In the example with the red car, we would randomize the color and in this way remove the correlation between red and the class "car". This method called *domain randomization* was popularized by Tobin et al. [31]. In Chapter 4, we will see a better solution how to achieve such invariance.

2.8 Chapter summary

To summarize this chapter, we showed how simulators could be effectively used in machine learning applications, concretely for the purposes of computer vision. We described three different scenarios of how such simulators could be used. We could use them to generate synthetic datasets with automatic labels, to train active learning or reinforcement learning algorithms and finally, as forward models which are inverted during inference. For the last scenario, we showed connections with the bias-variance analysis and showed different approximations, which make the inference tractable for complex and realistic simulators. Lastly, we have also described methods that reduce the residuum between real and synthetic examples and remove spurious correlations in synthetic datasets. Throughout the text, we introduced our work focused specifically on the generation of synthetic images.

2.9 Summary of my contribution

Our contribution devoted to the described topic was published in three papers included in the appendix A. The paper titled *Towards Visual Training Set Generation Framework* [9] introduced a framework for the generation of synthetic data described in Section 2.4.1. From a conceptual point of view, our contribution demonstrated the viability of this idea (in terms of photorealism and runtime performance). From a technical point of view, we created and easy-to-use interface to the Unreal game engine, which enabled to flexibly create 3D scenes and render them directly in the Python programming language.

The paper titled *Synthetic Dataset for Compositional Learning* [11] introduced and demonstrated the idea of compositional learning described in Section 2.4.2. We also showed the benefits of using synthetic data in this particular setting.

The last paper titled *Acquiring custom OCR system with minimal manual annotation* [28] described the use of synthetic data for the purpose of optical character recognition in historical texts. This work was done as part of an ongoing project focused on the analysis of historical texts called *Kramářské písně*. The application of synthetic data was particularly beneficial in this scenario because the input data was well understood and modeled.

List of references:

- J. Hůla, I. Perfilieva, and A. A. M. Muzaheed, "Towards visual training set generation framework," in *International Work-Conference on Artificial Neural Networks*. Springer, 2017, pp. 747–758
- V. Molek and J. Hula, "Synthetic dataset for compositional learning," in *Data Science and Knowledge Engineering for Sensing Decision Support: Proceedings of the 13th International FLINS Conference (FLINS 2018)*. World Scientific, 2018, pp. 1440–1445
- J. Hula, D. Mojžíšek, D. Adamczyk, and R. Čech, "Acquiring custom ocr system with minimal manual annotation," in *2020 IEEE Third International*

Conference on Data Stream Mining & Processing (DSMP). IEEE, 2020, pp. 231–236

3 Graph neural networks

3.1 Introduction

One disadvantage of classical neural networks such as multilevel perceptrons (MLPs), convolutional neural networks (CNNs), and recurrent neural networks (RNNs) is that they process only unstructured data in the form of vectors, tensors, or sequences of these. In many scenarios, we may want to encode our knowledge about a given problem and make the input structured. This structure may, for example, encode various relationships between individual elements of the input.

Good motivating examples arise in applications of neural networks for processing expressions written in a formal language. We may imagine a task in which our goal is to translate a formal expression into a natural language. We may either process the formal expression as a sequence of tokens and apply a *sequence-to-sequence* model which would constitute of RNN encoder and decoder [32]. Or we may produce a parse tree of the expression, but then we need an encoder neural network that is able to process trees. In general, we would like to process arbitrary graphs, which motivates the development of graph neural networks (GNNs).

This chapter is devoted to GNNs because they constitute an important tool for incorporating expert knowledge into neural network models. We describe them from the point of view of *graph signal processing*, which allows us to reuse intuition from classical signal processing methods.

We start by introducing the main ideas in graph signal processing, such as *graph Fourier transform* and *graph convolution*. Then, we show computationally efficient approximations of the graph convolution and finally, we generalize this approximation to give a generic formula for GNNs. We also show connections to models called *Transformers*, which have recently started replacing RNNs in sequence processing and CNNs in image processing. Finally, we describe our application of GNNs for algorithm scheduling, which is our contribution to this area.

3.2 Graph signal processing

Graph signal processing (GSP) is an emerging field [33] that can be understood as an extension of classical signal processing to the domain of graphs. Sometimes the adjective non-Euclidean or geometric is also used, which subsumes both discrete and continuous cases. In the continuous case, we deal with signals on manifolds and graphs, which could be actually understood as a discrete version of manifolds in the same way as we can understand regular grids as a discrete version of Euclidean spaces. Figure 16 depicts the differences between Euclidean and non-Euclidean cases.

We will focus on a discrete setting as it is the most amenable for computation, but we mention that most of the methods can be transferred to the continuous case. In the case of graphs, we may identify two types of tasks: one dealing only with the structure of the graph and the other dealing with signals defined on graphs. In the former category, we may identify tasks such as community detection [34], subgraph matching [35], etc., and in the latter, we may identify tasks such as node classification [36] or graph-based time series prediction [37].

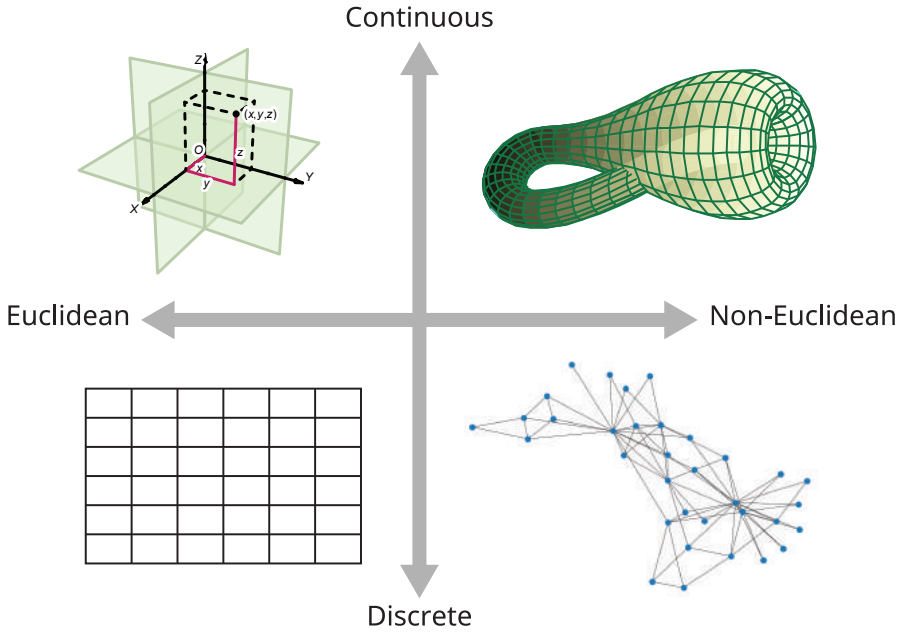


Figure 16: An analogy between continuous and discrete settings. If regular grids (bottom left) can be understood as a discrete version of Euclidean spaces (top left), then graphs (bottom right) could be understood as a discrete version of manifolds (top right).

In classical signal processing, we work with concepts such as the *Laplace operator*, convolution, or the Fourier transform, and these have direct analogs in the domain of graphs. We will define these concepts in the rest of this section, and once we arrive at the definition of graph convolution, we may extend convolutional neural networks to work with arbitrary graphs instead of regular grids.

3.2.1 Graph Laplacian

Classical Laplace operator (sometimes called Laplacian) is a differential operator defined as the divergence of the gradient of a function defined on a Euclidean space. Intuitively, the application of the Laplace operator to a function f, denoted by Δf, gives a new function that tells us (for every x) how much the average value of f deviates from $f(x)$ over a "small" ball centered at x.

In a cartesian coordinate system, Laplacian is computed as the sum of the second derivatives:

$$\Delta f = \sum_{i=1}^{n} \frac{\partial^2 f}{\partial x_i^2},$$

where x_i are axes of the coordinate system.

In the continuous case, the Laplace operator plays an important role in the modeling of diffusion. In image processing, which is often considered discrete, the Laplace operator is used to detect edges that consist of pixels where abrupt changes happen. Figure 17 shows the application of the Laplace operator to a greyscale image.

Figure 17: Left: Grey-scale image; Right: The same image convolved with the Laplace kernel.

The application of the Laplacian to an image works by convolving the image with a *kernel* that is a discrete approximation of the derivative. There exist multiple approximations which one could use. Figure 18 depicts the most basic version.

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

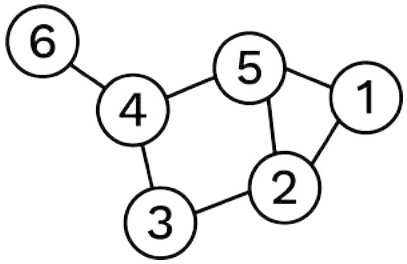
Figure 18: A discrete approximation of the Laplace kernel.

Notice that the value of the center is negative of the sum of the remaining values. This reflects the fact that the application of the Laplace operator to a constant signal should be equal to zero signal. We can now generalize the Laplacian to arbitrary graphs.

First, let us introduce a bit of terminology. Let $G = (V, E)$ denote a directed graph with a set of vertices $v_i \in V$ indexed by a set $I = \{1, \ldots, n\}$ and a set of edges E. The edge $e_{ij} \in E$ with a source vertex v_i and target vertex v_j is given by an ordered tuple (v_i, v_j). The connectivity of the graph is often represented by an adjacency matrix $A \in \mathbb{R}^{n \times n}$, where $A_{ij} = 1$ if $e_{ij} \in E$ and $A_{ij} = 0$ otherwise. We can also define a degree matrix $D \in \mathbb{R}^{n \times n}$ by $D_{ik} = \sum_{j \in V} A_{ij}$ if $i = k$ and $D_{ik} = 0$ otherwise.

Given an adjacency matrix A and a degree matrix D, we can finally define a Laplacian matrix[7]. It is given by a matrix $L = D - A$. Figure 19 depicts the

[7]Similarly, as with different discrete approximations to the Laplacian in an Euclidean case, there exist multiple definitions of the graph Laplacian. The one defined here is the most frequently used in graph signal processing.



$$\underset{\text{D}}{\begin{pmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}} - \underset{\text{A}}{\begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}} = \underset{\text{L}}{\begin{pmatrix} 2 & -1 & 0 & 0 & -1 & 0 \\ -1 & 3 & -1 & 0 & -1 & 0 \\ 0 & -1 & 2 & -1 & 0 & 0 \\ 0 & 0 & -1 & 3 & -1 & -1 \\ -1 & -1 & 0 & -1 & 3 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \end{pmatrix}}$$

Figure 19: The Laplacian matrix of a simple graph.

Laplacian matrix for a simple graph. It is easy to check that the sum across each row is equal to 0.

From now on, we will assume that we work with undirected graphs for which $e_{ij} \in E$ whenever $e_{ji} \in E$. This means that the Laplacian matrix is a symmetric real-valued matrix. Once we have a Laplace operator defined for arbitrary graphs, we can apply it to real-valued signals defined on such graphs.

Let $x : V \to \mathbb{R}$ be a real-valued signal on graph $G = (V, E)$[8]. Given the indexing set I, we can represent the signal x as a real-valued vector $\bar{x} = (x(v_1), \ldots, x(v_n))$. We can now apply the Laplace operator on a signal $\bar{x}$ by simple matrix multiplication $L \cdot \bar{x}$. The resulting vector will contain one value for each node, which indicates how much variation is there in the neighborhood of this node.

3.2.2 Graph Fourier transform

Given an operator, it is common to study its spectrum, which in the discrete case means the set of eigenvalues. These eigenvalues will be real because the Laplacian matrix is symmetric and real-valued. We can decompose the Laplacian matrix into an orthogonal matrix U with eigenvectors in each column and diagonal matrix Λ with eigenvalues on a diagonal:

$$L = U \Lambda U^T.$$

To get the intuition of what these eigenvectors reflect, we visualize them for two example graphs. In the bottom row of Figure 20, we can see the eigenvectors for a

[8]In practical applications, the signal is often vector-valued. The vector of each node represents some features of that node. If, for example, the graph represents a social network where each node corresponds to a user, the feature vector can reflect attributes of each node, such as gender, age, etc.

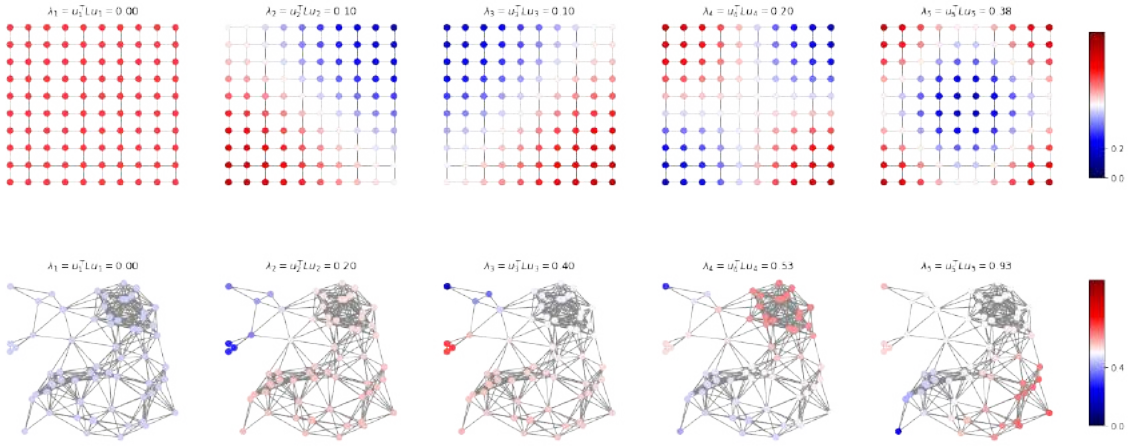


Figure 20: Visualization of eigenvectors of a Laplacian matrix. Each eigenvector is visualized by coloring the vertices of a graph. The eigenvectors are ordered by corresponding eigenvalues from left to right. The top row visualizes eigenvectors for a regular grid and the bottom row for a random graph.

graph with no visible structure. By looking at the eigenvectors, we can see that the ones corresponding to lower eigenvalues vary more slowly in terms of the neighborhood structure. We can therefore understand them as signals of different frequencies. This analogy to the frequency of the signal is more visible if the graph is a regular grid of nodes such as the one depicted in the top row of the same figure. For this reason, the eigenvectors of the Laplacian matrix are often called *Fourier modes* in the reference to the classical Fourier modes (sines and cosines of different frequency) and a projection of a signal defined on a graph onto these eigenvectors a *graph Fourier transform*. It is the basic tool of graph signal processing. If we convert an arbitrary signal into a Fourier domain, we can detect certain global properties of the signal or modulate the signal by suppressing certain "frequencies".

3.2.3 Graph convolution

The second important tool in standard signal processing is convolution. With discrete signals such as images, convolution may be realized by filtering the signal with a localized filter which is being shifted over the signal. At every point, the filter aggregates values from a localized neighborhood of the point.

Convolution can be also defined as any linear shift-equivariant operation. This means that if we have a shift operator S which translates the signal in some direction and convolution C, it holds that $CS = SC$. In matrix form, both convolution and the shift operator are realized as circulant matrices, and these are commutative. This definition allows us to generalize the convolution to arbitrary groups by replacing the translation group with another group (that acts on signals defined on a given domain). We require that the convolution is linear (because the signals on a given domain form a vector space) and equivariant with respect to a given transformation of the space, as depicted in the commutative diagram in Figure 21. Nevertheless, in this chapter, we do not need this level of generality, and we only describe the generalization of the convolution to arbitrary graphs.

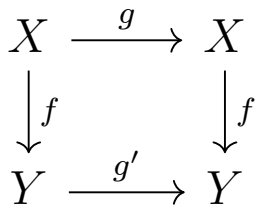


Figure 21: Commutative diagram illustrating equivariance of a function f with respect to group transformation of the space g (which can have a different representation g' on the domain of the function f).

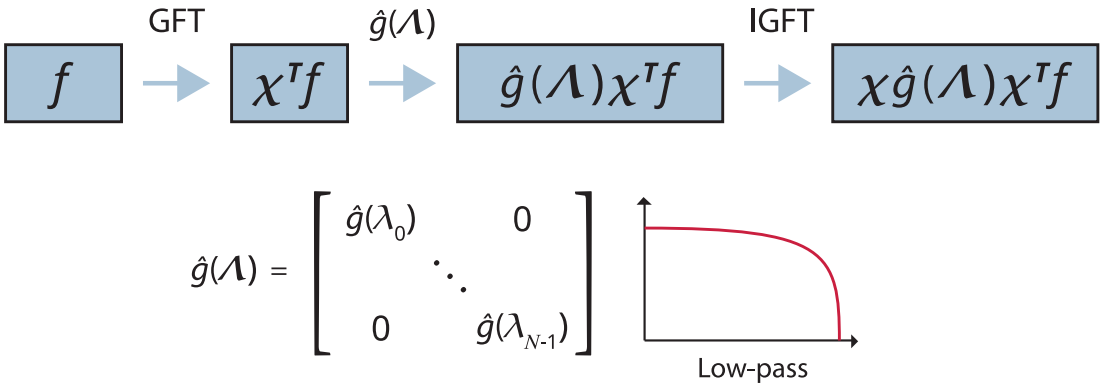


Figure 22: Application of the graph convolution. The signal f is converted to the Fourier domain by projecting it to the eigenvectors (χs) of the Laplacian matrix. The signal is then modulated in the Fourier domain by $\hat{g}(\Lambda)$ and finally converted back to the spatial domain. GFT is the abbreviation for graph Fourier transform, IGFT is the abbreviation for inverse graph Fourier transform.

At first sight, the convolution does not have a direct analogue for signals on a graph because the structure of the neighborhood could be different at every node. Nevertheless, we can easily define convolution in the Fourier domain due to the well-known fact (known as the *Convolution theorem*): convolving the signal g with a kernel h in the spatial domain is the same as if we first transform g and h to the Fourier domain, where we multiply the resulting signals in a point-wise manner, and finally transform the result back to the spacial domain with inverse Fourier transform. Formally, we can write it as:

$$g * h = \mathcal{F}^{-1}(\mathcal{F}(g) \cdot \mathcal{F}(h))$$

where $\mathcal{F}$ is Fourier transform (projection to Fourier modes), $*$ is convolution and $\cdot$ is pointwise multiplication. Therefore, to realize convolution on a graph, we can project the signal onto the eigenvectors of the graph Laplacian, then multiply it elementwise by a vector, which we can treat as a Fourier representation of the convolutional filter and finally reproject it back into the spatial domain. The whole process is depicted in Figure 22. For a more complete overview of the methods of graph signal processing, see the survey by Ortega et al. [33].

3.2.4 Graph convolutional networks

Spectral graph neural networks Using the definition of convolution from the previous paragraph, Bruna et al. [38] developed a graph version of convolutional neural networks, where the learnable parameters of one layer are contained in the vector of numbers $[\hat{g}(\lambda_i)]_i$ (shown in Figure 22). Their network was trained using gradient descent to maximize a given objective.

GNNs which compute the convolution in a spectral/Fourier domain are called *spectral graph neural networks*. They have two major issues. First, the learnt filter is not robust to small perturbations of the topology of the graph, which means that it can be used only on a fixed graph. The second issue is that to project the signal into the Fourier domain may be computationally expensive because of the cubic complexity of eigendecomposition.

Chebyshev approximation of the spectral filter The problem with computational complexity was diminished by Defferrard et al. [39] by imposing smoothness on the Fourier representation of the convolutional filter. Concretely, they use *Chebyshev polynomials* to approximate the convolutional filter and learn the individual coefficients of every monomial. These polynomials can be computed recursively using the following recurrence relation:

$$T_k(x) = 2xT_{k-1}(x) - T_{k-2}(x),$$

$$T_0 = 1,$$

$$T_1 = x.$$

In this case, this polynomial is a function of the eigenvalues, which play the role of various frequencies. If we abuse the notation and make each polynomial a function of a matrix, the approximation has the following form:

$$g_\theta(\Lambda) = \sum_{k=0}^{K-1} \theta_k T_k(\bar{\Lambda}),$$

where $\bar{\Lambda}$ is a diagonal matrix with scaled eigenvalues. Our goal is then to learn the coefficients θ_k from data. By imposing smoothness on the convolutional filter in the Fourier domain, we make the filter act locally in the spatial domain. We skip the details of the explanation, but in the case of the Chebyshev polynomial, the degree of the polynomial dictates how local the filter will be. For example, if the polynomial is of degree two, the nodes which will affect a given node in the spacial domain will be at most 2 edges away. This is depicted in Figure 23. The Chebyshev polynomials are orthogonal, which means that the coefficients can be learnt independently. Most importantly, the filter can be applied and trained directly in the spatial domain without the need of the eigendecomposition. For a more detailed explanation, see the original paper [39].

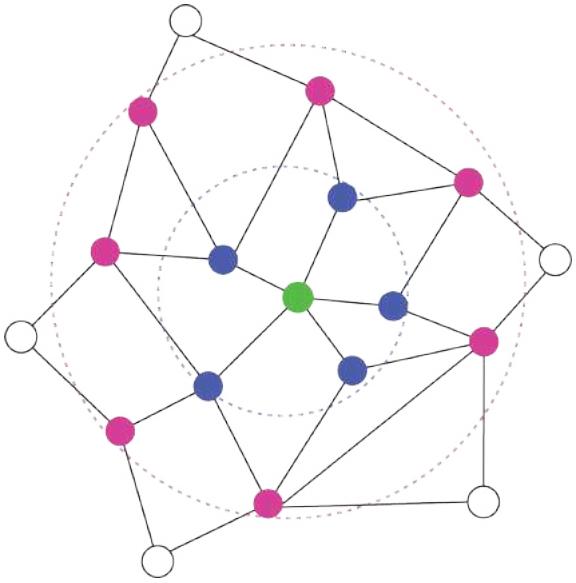


Figure 23: Neighborhood of a given node (green). Blue nodes are 1 edge away from the green node. Pink nodes are 2 edges away from the green node.

Graph-agnostic graph neural networks The approach with Chebyshev polynomials can still be applied only to a fixed graph if the approximating polynomials contain higher-order terms. If we consider only a first-order approximation of the Chebyshev polynomials, we obtain a version of a graph convolutional network in which each node u is affected only by its direct neighbors, which we denote by $N(u)$. The filter can be expressed as a linear function of the Laplacian matrix. In terms of individual nodes within the graph, the node update rule has the following form:

$$h_u^{(k)} = \sigma\left(W^{(k)} \sum_{v \in N(u) \cup \{u\}} \frac{h_v^{(k-1)}}{\sqrt{|N(u)||N(v)|}}\right),$$

where $h_u^{(i)}$ is a vector representation of the node u in layer i. The matrix $W^{(k)} \in \mathbb{R}^{dim(h_u^k) \times dim(h_u^{k-1})}$ contains trainable parameters, the square root in the denominator scales the vectors according to their degree, and σ is the application of a non-linear activation function (such as ReLU). We stress that $W^{(k)}$ is shared by all nodes in layer k but may differ across layers, allowing successive changes of feature vector size. Furthermore, notice that the update rule is invariant with respect to permutations of the neighbors, because the aggregation operator is a summation in this case.

Later, Gilmer et al. [40] generalized this update rule to arbitrary aggregation and update functions using the following equations:

$$a_u^{(k)} = \mathsf{AGGREGATE}^{(k)}\left(\left\{h_v^{(k-1)} \mid v \in N(u)\right\}\right)$$
$$h_u^{(k)} = \mathsf{COMBINE}^{(k)}\left(h_u^{(k-1)}, a_u^{(k)}\right).$$

Here, AGGREGATE could be an arbitrary aggregation operator that is permutation invariant (such as sum, mean, max, etc.). The role of COMBINE is to produce the updated feature vector $h_u^{(k)}$ using the feature vector from the previous layer

$h_u^{(k-1)}$ and the aggregated feature vector $a_u^{(k)}$. It can, for example, concatenate these two vectors, multiply the result by a matrix with trainable parameters, and finally apply a nonlinear activation function. The final layer of a GNN can aggregate feature vectors from all nodes to a single feature vector. This is depicted in Figure 24.

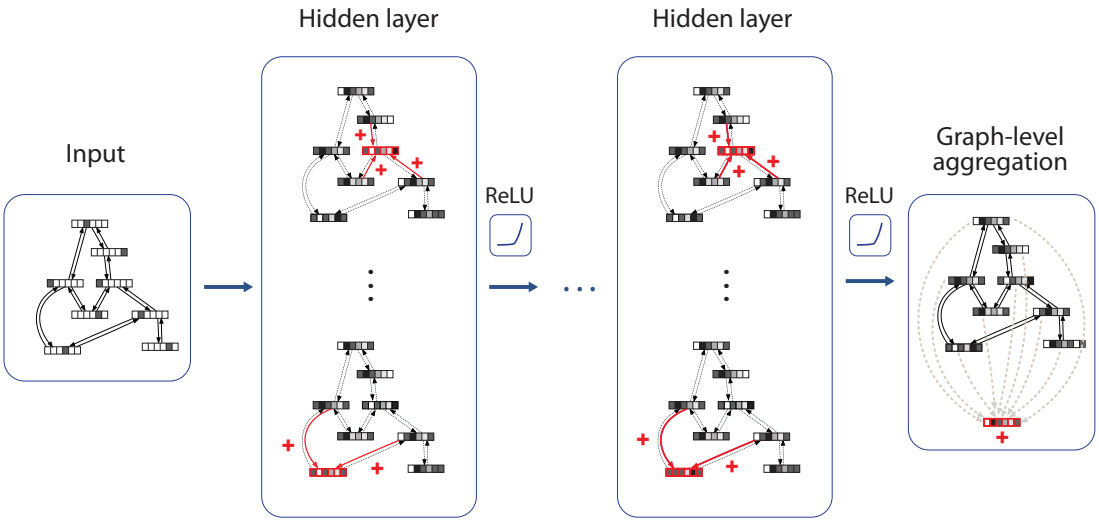


Figure 24: Schema of a GNN. Each node in a layer aggregates feature vectors from its neighbors. The last layer aggregates feature vectors from all nodes to a single feature vector.

3.2.5 Transformers

Transformers, introduced by Vaswani et al. [41], are neural network models which use the *self-attention mechanism* at every layer to produce a new set of feature vectors from the input set of feature vectors. During the last few years, these models started to dominate other types of neural networks in natural language processing, computer vision and other applications [41–45]. The main reason behind this popularity is that these models often achieve a higher accuracy when trained on large training datasets. In this section, we will describe the connections between Transformers and GNNs.

As mentioned previously, every layer of a Transformer takes a set of feature vectors as input and produces a set of feature vectors as an output. A layer in a GNN, on the other hand, takes as an input a graph of feature vectors and produces a graph of feature vectors as an output. We will see that a Transformer can actually be understood as a GNN processing a fully-connected graph induced by the set, using the self-attention mechanism as the aggregation operator.

Self-attention mechanism We now describe how one layer of a Transformer produces a new feature vector for *one* given element from the set.

Let $X = \{x_1^l, \ldots, x_n^l\}$ be a set of real-valued feature vectors $x_i^l \in \mathbb{R}^k$ for some $k \in \mathbb{N}$. For a given $j \in I = \{1, \ldots, n\}$, we want to produce x_j^{l+1} from the set X.

This process will be an implementation of the aggregation function:

$$x_j^{l+1} = \mathsf{AGGREGATE}\left(x_j^l, X\right).$$

In words, this aggregation works by first computing scalar weights a_{ji} for all $x_i \in X$ and then aggregating a new representation of each x_i using a weighted sum.

Concretely, we first obtain a new feature vector for x_j and two new feature vectors for every $x_i \in X$ using a linear transformation:

$$q_j = W_q \cdot x_j \qquad \left.\begin{aligned} k_i &= W_k \cdot x_i \\ v_i &= W_v \cdot x_i \end{aligned}\right\} \text{ for all } i \in I.$$

Reminiscent to the nomenclature in database systems, the vectors q_j, k_i, v_i are called *query*, *keys*, and *values* respectively. The matrices $W_q \in \mathbb{R}^{o_1 \times k}$,$W_k \in \mathbb{R}^{o_1 \times k}$, and $W_v \in \mathbb{R}^{o_2 \times k}$ contain learnable parameters. o_1 and o_2 are hyperparameters.

Using the vectors q_j and each k_i, we can compute the weights a_{ji} (also called the *attention scores*) by:

$$s_{ji} = \frac{k_i^T \cdot q_j}{\sqrt{o_1}}$$

$$a_{ji} = \frac{e^{s_{ji}}}{\sum_{k \in I} e^{s_{jk}}}.$$

Finally, we aggregate the vectors v_i into a weighted sum:

$$x_j^{l+1} = \sum_{i \in I} a_{ji} \cdot v_i.$$

Therefore, as mentioned in the beginning, we can understand Transformers as GNNs that process fully connected graphs using a self-attention mechanism for aggregation. As could be expected, several authors have already tried to extend Transformers to arbitrary graphs by restricting the aggregation to subsets of the set X [46, 47].

3.3 Using GNNs to process expressions written in a formal language

As mentioned at the beginning of this chapter, the main advantage of GNNs is their ability to process structured data. Plenty of examples of structured inputs can be found in tasks that deal with formal expressions. Such expressions can be naturally represented as syntactic trees or directed acyclic graphs as depicted in Figure 25. There are many possible applications of using machine learning methods for such formal expressions. For example, in *Automated theorem proving*, a typical task called *premise selection* [48] deals with the selection of formulas that may be useful in the search for a proof for a given statement. This may be approached using a machine learning model which is trained to score the suitability of a given formula for a given statement.

Another related application may be a *heuristic selection* for combinatorial problems written in a formal language, such as *SAT, Satisfiability modulo theories*, or

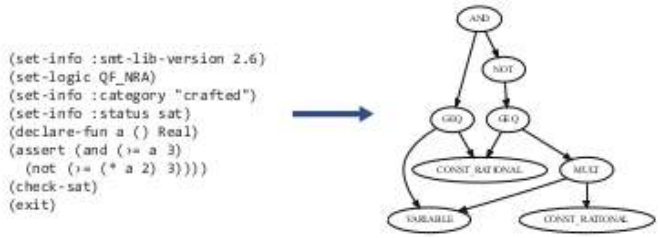


Figure 25: A formal expression parsed to a directed acyclic graph.

Integer linear programs. These problems are known to be *NP-hard* but for some subsets of such problems, effective heuristics exist. It is therefore desirable to predict which heuristics from a given portfolio may be effective for a given problem. In the past, such predictors were trained using manually designed feature vectors extracted from the formula [49,50]. In our contribution [51], we replace these manually designed feature vectors with a GNN which processes the raw graph representation of each formula. Concretely, we apply our method to solvers in the domain of Satisfiability modulo theories, but the method applies to any other domain where we are able to encode the problem as a graph.

Satisfiability modulo theories is a decision problem that deals with a satisfiability of formulas written in the language of first order logic with a dedicated theory (such as nonlinear integer arithmetic - NIA). Different solvers exist for different theories and they are often compared using a Par-2 score. Par-2 stands for penalized average runtime and is computed by penalizing the unsolved problems and averaging solving times of a given solver on the benchmark-set. The penalization is set to $2 \times$ timeout, where timeout is the maximum runtime allowed. The Par-2 score can be also computed for a portfolio from which solvers are selected or scheduled using a machine learning algorithm.

In our contribution we showed that we can significantly improve the Par-2 score if we schedule solvers by using predictions from a GNN which takes a given formula as an input. Quantitative results for selected benchmark sets can be seen in Table 1. For the benchmark set called UFNIA (nonlinear integer arithmetic with uninterpreted functions) we also show a *cactus plot* in Figure 26. Our method labeled by *dynamic schedule (GNN)* achieves significant improvement over individual solvers and a machine learning selector (labeled by *BOW single*), which was published recently [50]. For the detailed description of other baselines, see our publication [51], attached in the appendix B[9].

3.4 Chapter summary

To summarize, in this chapter we have described graph neural networks, which are neural networks that can process arbitrary graphs. Our description was based on ideas from graph signal processing. We have explaied concepts such as graph Laplacian, graph Fourier transform and graph convolution. Then we showed how

[9]The attached paper is an updated version of our original publication.

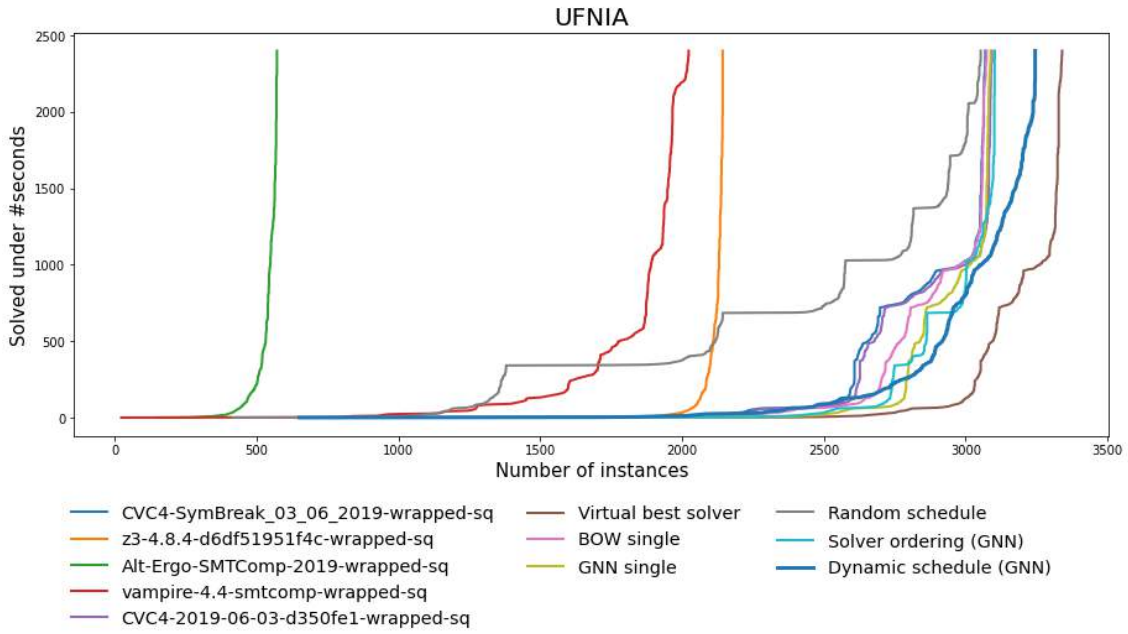


Figure 26: Cactus plot of the results on the UFNIA benchmark set: x-axis represents the number of instances and y-axis the time up to which this number was solved by respective schedule/solver. Our method is labeled as *dynamic schedule (GNN)*. *Virtual best solver* is an upper bound of possible improvement. It represents an algorithm which would pick the best solver for every formula.

to make graph neural networks computationally efficient by using approximations and how they relate to a popular model called Transformer. Finally, we described a high-level overview of our contribution in the domain of SMT solvers.

3.5 Summary of my contribution

Our contribution devoted to the described topic was published under the title *Graph Neural Networks for Scheduling of SMT Solvers* [51]. The core idea was to apply a machine learning model to the selection and scheduling of solvers in the domain of Satisfiability modulo theories (SMT). Such solvers take as input a formula written in the language of first-order logic with dedicate theory (such as non-linear integer arithmetic). Their goal is to find an assignment of variables that makes the formula true or find a proof showing that no such assignment exists.

Different solvers use different search heuristics and therefore their solving time varies from formula to formula. Our aim was to select/schedule solvers using a machine learning algorithm based on the features of the formula. Instead of manually designed features, we trained a graph neural network to produce a low-dimensional vector embedding of the formula represented as a directed acyclic graph. This vector embedding then goes as an input to the final classification/regression model. Our method achieved a significant improvement on a few selected benchmark sets. According to our knowledge, we were the first to a use graph neural network in the domain of SMT solvers. We note that the paper attached in the appendix B is an

Benchmark set		**QF-NRA**	**UFNIA**	**UFNIA-CONF**	**TPTP**
Best Solver	solver	Z3	CVC4	-	Vampire
	solved	2120	3093	2494	3204
BOW single	PAR-2 impr.	117.10%	-0.64%	0.32%	13.56%
	solved	2343	3074	2586	3315
GNN single	PAR-2 impr.	231.19%	1.8%	56.73%	18.23%
	solved	2403	3085	2644	3320
Dynamic schedule	PAR-2 impr.	**1162.87**%	**76.91**%	**113.54**%	**51.92**%
	solved	**2498**	**3245**	**2891**	**3388**

Table 1: Comparison of evaluated approaches. *Best solver* is the best individual solver from the portfolio. The PAR-2 improvement is relative to the *Best solver*. *solved* is the number of solved instances. *BOW single* is a machine learning selector published recently [50], which selects a solver based on bag-of-words features of a given formula. *GNN single* replaces the bag-of-word features with a GNN. *Dynamic schedule* uses same GNN as *GNN single* but additionally creates a schedule instead of selecting only one solver.

updated version of the orignal contribution.

List of references:

- J. Hůla, D. Mojžíšek, and M. Janota, "Graph neural networks for scheduling of smt solvers," in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2021, pp. 447–451

4 Symmetries in computer vision

4.1 Introduction

This chapter is devoted to the removal of symmetries present in computer vision. We will show that if we remove these symmetries from the representation of an image, we will obtain a much more meaningful notion of similarity that ignores the inessential information such as the viewing angle of a camera.

We start in Section 4.2 by describing symmetries that are standardly leveraged in computer vision. Next, in Section 4.3, we compare symmetry removal with a very popular technique used in machine learning applications called *data augmentation*. In Section 4.4, we describe more general symmetries which are often ignored by deep learning approaches to computer vision. Section 4.5 contains a short overview of our work focused on this topic in which we focus on the removal of background pixels. In Section 4.6, we show the positive effects of background removal on other downstream tasks[10] such as clustering of video sequences. We show that if we remove background pixels from videos capturing unknown objects, we are able to discover classes of objects by an unsupervised clustering algorithm. We finish this chapter in Section 4.7 by describing other symmetries whose removal would lead to a further positive impact on downstream tasks. At the same time, we describe a possible way how to remove these symmetries, which will be a focus of our future work.

4.2 Standardly used symmetries in computer vision

Aside from the informal meaning of the word "symmetry", this word also denotes a precisely defined concept studied in *group theory*. Concretely, the *symmetry group* of a geometric object is a group of all transformations under which is the given object invariant. By invariant, we mean that some properties of the object will not change if we apply any transformation from the given symmetry group. To form a group, these transformations need to be invertible, contain an identity transformation, and need to be endowed with some notion of composition.

In computer vision, we often require invariance with respect to the transformations of a given image. For example, we may require that an image classifier would be invariant with respect to translation of the image, which means that the classifier would produce the same class if we shift every pixel by some fixed amount Δ. This is illustrated in Figure 27.

Apart from translations, we may also require invariance with respect to rotation and scaling. A well-known example of an algorithm that models the mentioned invariances is a *Scale-invariant feature transform* (SIFT) [52]. If we define an image as a function that assigns a color value to every point in the image plane, then all these transformations are *similarity transformation* (transformations that preserve shape) of the domain of this function. More general transformations of the domain would be arbitrary linear transformations (such as skew and homography) or even arbitrary nonlinear deformations. We could also require invariance with respect to a

[10]The term downstream task typically denotes a task which is using representations of inputs optimized for some other task.

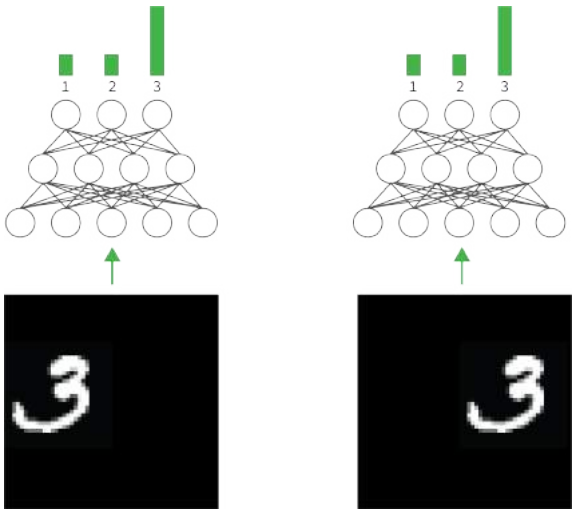


Figure 27: An illustration of translation invariance of a neural network. The network should produce the same output if every pixel in the image is shifted to the right.

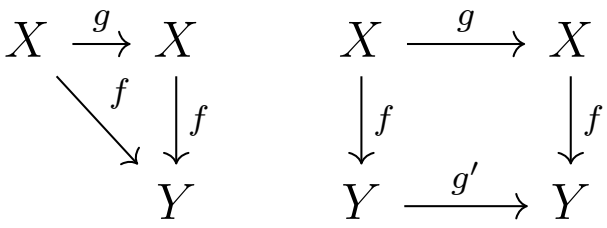


Figure 28: Commutative diagrams for invariance (left) and equivariance (right).

codomain of the function, for example, to the shift of brightness values or a change of contrast.

Sometimes, we are also interested in equivariant functions/algorithms, as described in Section 3.2.3. These are functions f, which respect the transformations g in a sense that: $g \circ f = f \circ g$. Here, we assume that both the function f and the transformation g, are functions from a space X to space X. In general, the function f may have different codomain Y and the transformations may be represented differently on space X and Y. In Figure 28, we show commutative diagrams for invariance and equivariance.

4.3 Data augmentation vs symmetry breaking

With deep learning approaches to computer vision, the technique of data augmentation became a very popular way, how to avoid explicit modelling of invariance. Instead of hard-coding the invariance with respect to some transformations in a model, we enlarge the training set by applying these transformations and combinations of them to the original images.

From an empirical point of view, such data augmentations provide us with an increase in performance in most of the scenarios. From the theoretical point of view, such solutions are more problematic because it is possible that the model (neural

Figure 29: Different augmentations applied to the same image. Individual augmentations were created by composing scaling, cropping, and change of color values.

network) only interpolates between memorized patterns corresponding to different versions of the augmented images. Be it the case, then the number of augmented images required would grow exponentially with the number of possible transformations because we would need to create all possible combinations of them for training. In Figure 29, we show typical augmentations which are used in computer vision applied to a single image.

With more complex augmentations, we actually do not care whether the individual transformations form a group. For example, individual transformations may not have a unique inverse and it may not be clear how to compose them. Nevertheless, we may still informally use the word "symmetry". In the next section, we will see that such transformations become relevant when we take into consideration the fact that images arise as 2D projections of a 3D space.

As mentioned above, data augmentation works by creating many possible transformations of a given image. We then train the neural network to produce the same output on the augmented images. It would be preferable if the neural network ignored these transformations by design. One way to achieve this is by transforming the image into a canonical representation so that all images, which are augmentations of the same image, would be mapped to the same canonical representation.

We can call this process *symmetry breaking*. Symmetry breaking is a concept that originated in physics [53] but later was adapted in the study of combinatorial problems, such as SAT [54], where the problems may exhibit a lot of symmetries. Various preprocessors are being used that transform the initial problem in such a way that it forces the solver to ignore these symmetries. A similar idea can be applied in computer vision where we may transform the input image to a canonical representation so that the network does not need to learn the symmetries. This idea is depicted in Figure 30.We present our work in this direction in Section 4.5.

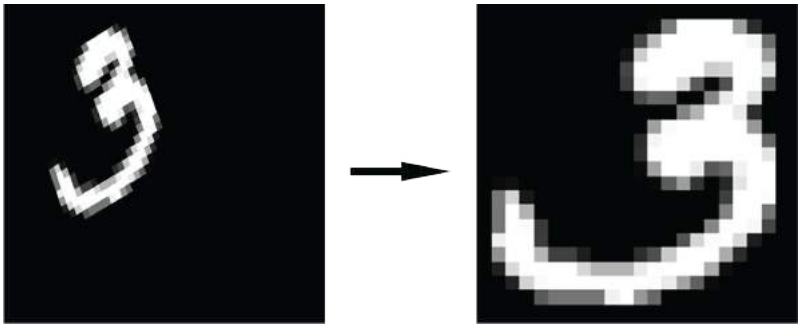


Figure 30: Transforming an image to a canonical representation.

4.4 Other types of symmetries present in computer vision

There exist a lot of other known symmetries which are not taken into account in deep learning approaches to computer vision. Some of them are very general and applicable in most of the scenarios, and some are task-specific.

The general ones are applicable for photographs or videos of natural scenes. We know that such photographs are created by a perspective projection of a 3D scene, and this perspective projection is fully understood. We also know that the 3D world consists of independent objects whose movement can be often described by a rigid 3D transformation (these are transformations that do not change the distances between points). The exploitation of this idea will be described in Section 4.6. Furthermore, we know how light behaves and how it affects the resulting image. All these symmetries are very hard to model and use for data augmentation if we do not have 3D models at our disposal to create synthetic datasets. As described in the previous section, it may be more desirable to ignore them instead of using them for data augmentation.

If our goal is to provide semantic labels for images, we know that the class of the object in the image should not change if we change the perspective of the camera, the lighting of the scene, or the environment the object is placed in. If we consider time, we also know that an object should not change the class from one moment to another. All these facts are neglected by standard deep learning approaches for computer vision. For example, convolutional neural networks for object segmentation in videos - such as the one developed by [55] - do not take the fact that the world is three-dimensional into account and process every frame in a video independently from other frames. Therefore, nothing prevents such networks from classifying the same object as a car in one frame and as a pedestrian in the next frame.

We believe that if we remove all symmetries and represent the object in some canonical form, then the semantic tasks, such as object classification will become much easier and will only require a few training examples for a class instead of thousands. The experiment described in Section 4.6 supports this hypothesis.

There is also evidence that our brains remove such symmetries when processing visual stimuli. It is, for example, known that our brains impose local coordinate frames to individual objects or automatically remove lighting effects to perceive the true color of an object. This phenomenon is well-illustrated by the visual illusion

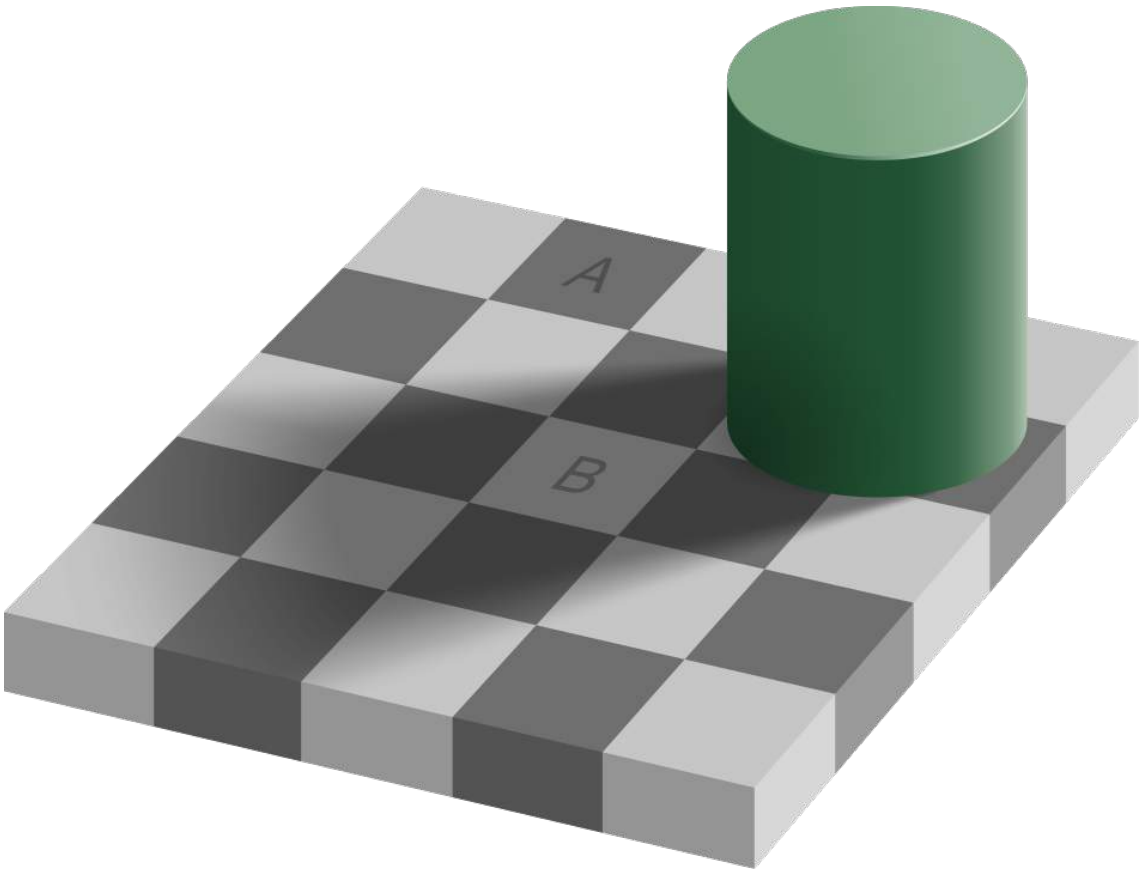


Figure 31: A visual illusion demonstrating that our brain is automatically removing lightning effects to recover the true color of a surface. The squares marked by letters A and B contain pixels with the same RGB values. Image taken from: https://en.wikipedia.org/wiki/Checker_shadow_illusion.

depicted in Figure 31.

4.5 Segmenting out generic objects in videos

In this section, we present our work focused on the segmentation of generic objects in monocular videos. This work is a part of a larger project where the goal is to discover classes of objects in videos without any labels. Concretely, we assume a dataset that contains sequences of frames capturing unknown objects. The task is to cluster these sequences in such a way that all sequences that capture objects of the same class, constitute one cluster. The concrete details about the clustering task will be provided in Section 4.6.

Our goal was to show that if we segment out the object in the video and remove the background pixels, we will achieve higher accuracy in the clustering. The reason is that the clustering reflects the similarity between frames, and the background acts as a kind of noise that pollutes the similarity between objects. This idea is depicted in Figure 32. In the language of the previous section, our goal was to remove the symmetry of the environment the object is placed in.

We also point out that one should not confuse this task with classical object

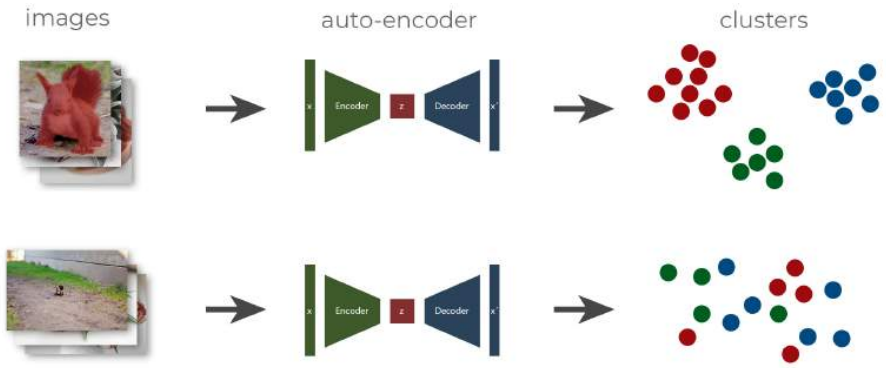


Figure 32: A schema illustrating the improvement of clustering when the background is removed from the image. The images are clustered using a low-dimensional feature vectors obtained from a trained autoencoder.

segmentation, which is done on a predefined set of classes, and the model is trained in a supervised way. In our case, we want to segment out objects without knowing the possible classes in advance. For this reason, we call this task *generic object segmentation*. In comparison to classical object segmentation or detection, very little attention was given to this task, and we believe it has a great potential for computer vision applications.

4.5.1 Ensemble of models

Our approach to generic object segmentation is based on three insights:

1. Objects are generally well separable from the background in the *depth map*, which measures distances of points on a captured surface to a camera.
2. Objects sometimes move independently of the rest of the scene.
3. Objects generally do not change appearance in small intervals of time.

These three insights could be leveraged using three different models:

1. A model for depth map estimation
2. A model for motion estimation
3. A model for object tracking

Independently, these models could fail to detect the objects in some scenarios. For example, the model for motion estimation will not detect the object if it is not moving. Therefore, we use an ensemble of these three models to obtain more robust results.

The model for object tracking (tracker) requires initialization in the form of a bounding box around the object. We initialize the tracker from the predictions of the other two models in which they agree the most. For concrete instantiation of these models, we use state-of-the-art architectures described in the recent literature. Their description is provided below:

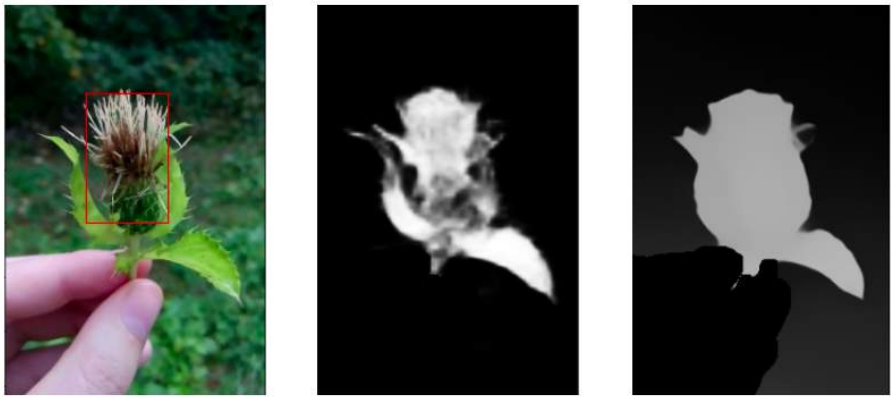


Figure 33: Left: the bounding box for the initialization of the tracker computed from the predictions of the other two models. The bounding box is initialized in frames where the other two models agree the most in their predictions. Middle and right: output from the model for motion estimation and depth map prediction, respectively.

Depth Prediction For the depth prediction, we use the model introduced by Ranftl et al. [56], available from the author's repository[11]. This transformer-based model predicts a scalar value for each pixel, which represents the distance of the surface captured by that pixel from the camera center.

Optical Flow Estimation For optical estimation, we use the model introduced by Teed et al. [57]. It is also a transformer-based model which requires two consecutive frames of video to produce the optical flow field. The optical flow field assigns two scalar values to each pixel. These values represent the pixel displacement on the x and y axes, relative to the previous frame. To obtain one scalar value for each pixel, we take the magnitude of the displacement. We used the implementation of the model with trained weights provided in the authors repository[12].

Object tracking For tracking objects, we use a model called SiamMask [58], a neural network trained as a Siamese architecture that simultaneously performs both visual object tracking and object segmentation in a video. We used the implementation available online[13].

The predictions of individual models are depicted in Figure 33. We convert these predictions for every pixel to values between 0 and 1 and aggregate them using the following formula (for each pixel i, we aggregate the values from the three models):

$$\text{output}(x)_i = \frac{\min\left(e^{\sum_{j=1}^{3} f_j(x)_i} - 1, e^2 - 1\right)}{e^2 - 1}, \tag{8}$$

[11]https://github.com/intel-isl/DPT
[12]https://github.com/princeton-vl/RAFT
[13]https://github.com/foolwood/SiamMask

Figure 34: Example images from the organic objects dataset. The images were already processed with the background removal algorithm. Concretely, each image was cropped with the bounding box of the detected object and the background was blurred.

We do not explicitly measure the accuracy of the ensemble for the generic object segmentation. Instead we test the effects of background removal on a downstream task as described in the next section. For more details about the background removal, see our original publication [59]

4.6 Effects of background removal on clustering accuracy

As described in the previous section, our goal was to show the effects of background removal on clustering accuracy. In this section, we describe the dataset on which we test our method, the clustering algorithm we devise specifically for the task of sequence clustering, and the experimental results.

4.6.1 Organic objects dataset

We test our algorithm for class discovery on a dataset of video sequences which we collected specifically for this purpose. Each video sequence captures one organic object, such as a flower or a mushroom. We used organic objects because they provide a lot of variability within a given class. The dataset contains 18 classes (different types of flowers, mushrooms, etc.) and for each class, it contains 10 video sequences capturing different instances of the class, on a different background and in different lighting conditions. Each video contains approximately 300 frames capturing the object from different viewing angles. Samples from the dataset are depicted in Figure 34.

4.6.2 Clustering algorithm for video sequences

To cluster the video sequences, we leverage the constraint that all frames that track a particular object within one video sequence should belong to the same cluster. In other words, we use the fact that the object will not change its identity over

time. This is useful if two videos capture an object of the same class, but only very few frames capture it from the same viewing angle where the frames will be highly similar. This similarity will be propagated to all other frames within these two videos.

The clustering algorithm has a form of a community detection algorithm that runs on a similarity graph of individual videos (nodes correspond to videos, and edges reflect the similarity between them). Concretely, we use the *Louvain* community detection algorithm which does not need to know the number of clusters in advance. The clusters are detected as "communities" of highly interconnected nodes within a graph. The similarity graph is constructed by connecting every video sequence to its n most similar video sequences (in our case n is set to 5).

To measure the similarity between video sequences, we exploit the fact that only a few frames of the video sequence may capture the object from the same viewing angle. Concretely, to measure this similarity, we use an average similarity of k most similar pairs of frames (where one frame in the pair is from the first video and the other from the second video). To measure the similarity between frames, we first obtain a low-dimensional representation of every frame using a variational autoencoder [23] which is trained to reconstruct the image from a low-dimensional feature vector. Then we measure the cosine distance between these low-dimensional representations. For technical details, see our original publication [59].

Therefore, the quality of the clustering algorithm is mostly dependent on whether the similarity between the representations of individual frames reflects the similarity between objects. As describe further, we obtain noticeable improvements if we remove the background pixels from each frame. Then the similarity between frames will reflect the similarity between objects and not between backgrounds, which can take a large part of the image.

4.6.3 Experiments

To test the effects of background removal on the clustering accuracy, we run two experiments with the same clustering algorithm with the difference that in one experiment we remove the background and in the other we leave the image intact. To compute the accuracy, we first assign a class to each cluster by finding the most frequent class in the given cluster, and then we compute the proportion of incorrectly assigned videos. We also compare the number of clusters discovered.

With the background removed, out of 173 videos, only five were assigned to the wrong component and the algorithm discovered the correct number of classes. The result of community detection on the constructed similarity graph is shown in Figure 35. The accuracy is much worse in the case without background removal, as can be seen in the numerical results presented in Table 2.

4.6.4 Drawbacks of our approach

Our work on class discovery and background removal was meant only as a proof of concept which demonstrates the effectivity of background removal for downstream tasks. We restricted our experiments to a few constraints which may not always hold in practice. Concretely, we assume that each video contains only one object,

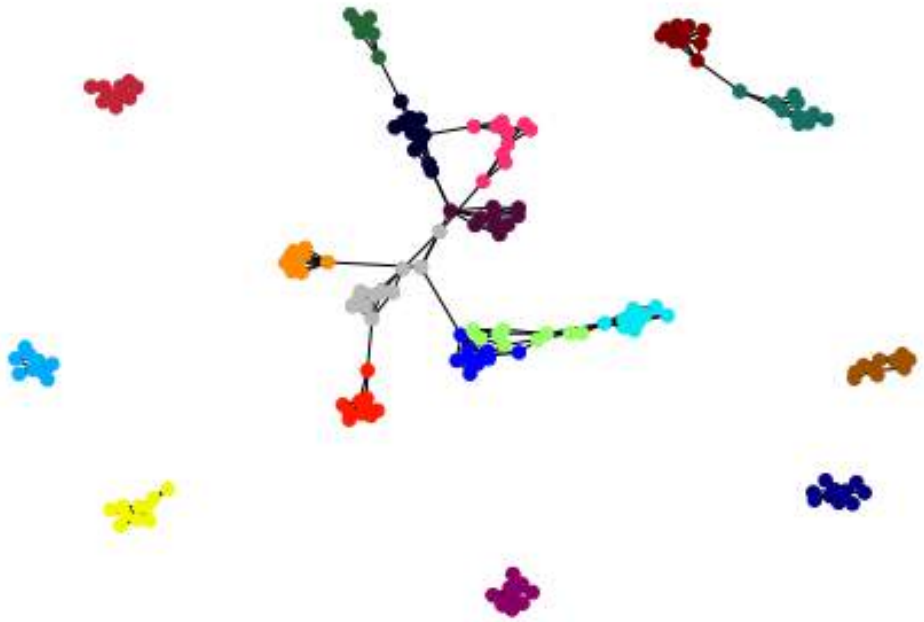


Figure 35: Visualization of detected communities in the Organic Objects dataset with the Louvain method. Nodes are colored according to the component (community) they are assigned to. The method discovered 18 components that correspond to 18 different classes in the dataset. The visualization of the graph pulls together nodes that are connected so that highly interconnected subgraphs form clusters.

Pre-segmenting the objects	Number of discovered classes	Accuracy
Yes	18	97.1%
No	15	41.0%

Table 2: Comparison of the clustering of videos with and without the background removed. The accuracy is computed by checking how many times a video was assigned to an incorrect cluster.

and, as described in our paper, we explicitly segment out hands from each video. The object in the video is often held by a hand and therefore the hand is detected as a part of the object. To remove the hand from the object, we run a segmentation model trained for hand segmentation and subtract the mask of the hand from the mask of the object. Finally, we did not optimize the whole algorithm with respect to speed. We plan to focus on these drawbacks in our future work.

4.7 Removing symmetries one by one

In this section, we describe the overall vision of symmetry removal and our future plans. In the last section, we have described how background removal affects the accuracy of downstream tasks. There are still other symmetries, which if removed, should improve the accuracy of downstream tasks even further. In the following text, we describe two symmetries we plan to focus on in our future work.

4.7.1 Invariance with respect to a viewing angle

Ideally, we would like to obtain such a representation of an object that is invariant with respect to a viewing angle. The only reasonable way to achieve this is to obtain a 3D representation of the object and then represent the object in its local coordinate system of the object, which ignores the position of the camera. Such methods, which provide low-dimension feature vectors of 3D meshes, exist and usually use graph neural networks and a training method similar to an autoencoder [60].

To obtain the 3D mesh of the object, we can use methods of 3D reconstruction which take as input several images of the object captured from different viewing angles and produce the reconstructed 3D mesh [61]. Such methods are well-developed but in general, they assume that the transformation of the scene from one viewing angle to another can be described by a rigid transformation. This assumption is broken, if different angles are obtained from different frames of the video, and one object in the video is moving independently of the rest of the scene. That is the case in our dataset. Nevertheless, if we mask out the background pixels as described previously, this assumption will often hold again (if the object is not deformed during the movement).

4.7.2 Invariance with respect to deformations

Many objects have some degrees of freedom and can be deformed. For example, the movement of certain animals can be well-approximated by a simple skeleton that has only a few degrees of freedom. During such a movement, the geodesic distances between points lying on the surface of the object will not change. In the local coordinate system defined on the surface of the object, the deformation of the object can be described by an isometry (a transformation that preserves distances). Therefore, we would like to obtain a representation of the object which is invariant with respect to isometries. Such representations could be obtained by graph neural networks which are explicitly designed to be invariant with respect to isometry using the Laplace-Beltrami operator [62].

If we successively remove the symmetries (backgrounds, viewing angle, deformation, etc.), we obtain a canonical representation of the object which will not be "polluted" by irrelevant information. In the original pixel space representation, this irrelevant information is entangled with the relevant information and therefore, tasks such as object classification require large labeled datasets for training.

4.8 Chapter summary

To summarise, in this chapter we showed the benefits of symmetry removal in computer vision. We compared symmetry removal to a standardly used technique called data augmentation. Apart from the removal of background pixels, we described other symmetries that occur in computer vision and whose removal would have a positive impact on computer vision tasks such as object classification. Throughout this chapter, we have also described our work focused on this topic together with experimental results.

4.9 Summary of my contribution

Our contribution devoted to the described topic was published in two papers. The paper titled *Unsupervised Object-aware Learning from Videos* [63] demonstrated the effects of background removal on the task of image clustering. We also demonstrated the benefits of working with videos instead of separate images and introduced a clustering algorithm for videos that does not need to know the number of clusters in advance. Together with the paper, we released a dataset containing videos of organic objects. A presentation of our contribution can be found at the following video: https://youtu.be/UZ8wh6nB_bE.

One drawback of this contribution was the fact that the removal of the background was not fully automatic. It required an initialization in the form of one bounding box per video. This issue was resolved in our following contribution titled *Segmenting out Generic Objects in Monocular Videos* [59] where we demonstrated a way to automatically segment out unknown objects from the background. The solution consisted of an ensemble of three complementary models trained by different objectives. The publication describing the segmentation of generic objects can be found in the appendix C.

List of references:

- J. Hula, "Unsupervised object-aware learning from videos," in *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*. IEEE, 2020, pp. 237–242
- J. Hula, D. Adamczyk, D. Mojzisek, and V. Molek, "Segmenting out generic objects in monocular videos," 2021

Resumé

Metody, které se učí z dat - jako jsou například neuronové sítě - vyžadují velké trénovací sady, nejsou lehce interpretovatelné a často nedokáží generalizovat, pokud se testovací distribuce značně liší od té trénovací. Zmíněné nevýhody lze částečně eliminovat využitím našich znalostí o dané doméně. Tato práce popisuje tři odlišné přístupy, které kombinují znalost domény s neuronovými sítěmi. První přístup je založen na generativních modelech/simulátorech, které simulují vznik pozorovaných dat. Kapitola věnovaná tomuto přístupu zároveň popisuje naši práci zaměřenou na syntetická data. Druhý přístup využívá grafové neuronové sítě, ktere dokáží zpracovávat data strukturovaná ve formě grafu. Zde jsme představili vužití těchto sítí pro vyběr/rozvrhování solverů v doméně Satisfiability modulo theories. Poslední přístup je založen na odstraňování symetrií, které vznikají v oblasti počítačového vidění. Náš příspěvek na toto téma demonstroval výhody automatického odstranění pozadí.

Summary

Data-driven methods, such as neural networks, require large training sets, are not easily interpretable, and often do not generalize if the distribution of test examples is shifted in some sense. These drawbacks can be diminished if we leverage our knowledge about a given domain. This thesis describes three different approaches that combine domain knowledge with neural networks. The first approach is based on generative models/simulators that simulate the creation of the observed data. We described our contributions focused on synthetic datasets. The second approach uses graph neural networks, which can process inputs structured in the form of a graph. Here we described our contribution focused on solver selection/scheduling in the domain of Satisfiability modulo theories. The last approach is based on the removal of symmetries that arise in computer vision. Our contribution devoted to this topic demonstrated the benefits of background removal for the task of image clustering.

References

[1] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning.* MIT press Cambridge, 2016, vol. 1.

[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction.* MIT press, 2018.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *ArXiv*, vol. abs/1712.01815, 2017.

[4] E. H. Shortliffe, "Computer-based medical consultations, mycin," 1976.

[5] E. S. Spelke, "Core knowledge, language, and number," *Language Learning and Development*, vol. 13, pp. 147 – 170, 2017.

[6] B. Kogut, B. Cowgill, S. Helper, H. Kim, G. F. Marcus, and F. Teodoridis, "Debates on artificial intelligence," 2020.

[7] J. Gläscher, N. D. Daw, P. Dayan, and J. P. O'Doherty, "States versus rewards: Dissociable neural prediction error signals underlying model-based and model-free reinforcement learning," *Neuron*, vol. 66, pp. 585–595, 2010.

[8] G. Harvey, J. Tobochnik, and W. Christian, "An introduction to computer simulation methods: applications to physical systems," 1988.

[9] J. Hůla, I. Perfilieva, and A. A. M. Muzaheed, "Towards visual training set generation framework," in *International Work-Conference on Artificial Neural Networks.* Springer, 2017, pp. 747–758.

[10] E. Wood, T. Baltruvsaitis, C. Hewitt, S. Dziadzio, M. Johnson, V. Estellers, T. J. Cashman, and J. Shotton, "Fake it till you make it: Face analysis in the wild using synthetic data alone," *ArXiv*, vol. abs/2109.15102, 2021.

[11] V. Molek and J. Hula, "Synthetic dataset for compositional learning," in *Data Science and Knowledge Engineering for Sensing Decision Support: Proceedings of the 13th International FLINS Conference (FLINS 2018).* World Scientific, 2018, pp. 1440–1445.

[12] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[13] C. Olah, A. Mordvintsev, and L. Schubert, "Feature visualization," *Distill*, 2017, https://distill.pub/2017/feature-visualization.

[14] B. Settles, "Active learning literature survey," University of Wisconsin-Madison Department of Computer Sciences, Tech. Rep., 2009.

[15] V. Nair, J. Susskind, and G. E. Hinton, "Analysis-by-synthesis by learning to invert generative black boxes," in *International Conference on Artificial Neural Networks*. Springer, 2008, pp. 971–981.

[16] P. Dayan, G. E. Hinton, R. M. Neal, and R. S. Zemel, "The helmholtz machine," *Neural computation*, vol. 7, no. 5, pp. 889–904, 1995.

[17] A. Yuille and D. Kersten, "Vision as bayesian inference: analysis by synthesis?" *Trends in cognitive sciences*, vol. 10, no. 7, pp. 301–308, 2006.

[18] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum, "Deep convolutional inverse graphics network," in *Advances in neural information processing systems*, 2015, pp. 2539–2547.

[19] N. Chater, M. Oaksford *et al.*, *The probabilistic mind: Prospects for Bayesian cognitive science*. Oxford University Press, USA, 2008.

[20] G. Westheimer, "Was helmholtz a bayesian?" *Perception*, vol. 37, no. 5, pp. 642–650, 2008.

[21] K. Varelas, A. Auger, D. Brockhoff, N. Hansen, O. A. ElHara, Y. Semet, R. Kassab, and F. Barbaresco, "A comparative study of large-scale variants of cma-es," in *PPSN*, 2018.

[22] A. Munk, A. Scibior, A. G. Baydin, A. Stewart, G. Fernlund, A. Poursartip, and F. D. Wood, "Deep probabilistic surrogate networks for universal simulator approximation," *ArXiv*, vol. abs/1910.11950, 2019.

[23] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *CoRR*, vol. abs/1312.6114, 2014.

[24] T. D. Kulkarni, P. Kohli, J. B. Tenenbaum, and V. K. Mansinghka, "Picture: A probabilistic programming language for scene perception," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4390–4399, 2015.

[25] A. G. Baydin, L. Shao, W. Bhimji, L. Heinrich, L. Meadows, J. Liu, A. Munk, S. Naderiparizi, B. Gram-Hansen, G. Louppe, M. Ma, X. Zhao, P. H. S. Torr, V. W. Lee, K. Cranmer, Prabhat, and F. D. Wood, "Etalumis: bringing probabilistic programming to scientific simulators at scale," *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019.

[26] J.-W. van de Meent, B. Paige, H. Yang, and F. Wood, "An introduction to probabilistic programming," *ArXiv*, vol. abs/1809.10756, 2018.

[27] S. Mohamed, M. Rosca, M. Figurnov, and A. Mnih, "Monte carlo gradient estimation in machine learning." *J. Mach. Learn. Res.*, vol. 21, no. 132, pp. 1–62, 2020.

[28] J. Hula, D. Mojžíšek, D. Adamczyk, and R. Čech, "Acquiring custom ocr system with minimal manual annotation," in *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*. IEEE, 2020, pp. 231–236.

[29] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 2242–2251, 2017.

[30] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014.

[31] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 23–30, 2017.

[32] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014.

[33] A. Ortega, P. Frossard, J. Kovacevic, J. Moura, and P. Vandergheynst, "Graph signal processing: Overview, challenges, and applications," *Proceedings of the IEEE*, vol. 106, pp. 808–828, 2018.

[34] S. Fortunato, "Community detection in graphs," *ArXiv*, vol. abs/0906.0612, 2009.

[35] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *ArXiv*, vol. abs/1205.6691, 2012.

[36] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2016.

[37] A. Derrow-Pinion, J. She, D. Wong, O. Lange, T. Hester, L. Perez, M. Nunkesser, S. Lee, X. Guo, B. Wiltshire, P. Battaglia, V. Gupta, A. Li, Z. Xu, A. Sanchez-Gonzalez, Y. Li, and P. Velivckovi'c, "Eta prediction with graph neural networks in google maps," *ArXiv*, vol. abs/2108.11482, 2021.

[38] J. Bruna, W. Zaremba, A. D. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *CoRR*, vol. abs/1312.6203, 2014.

[39] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *NIPS*, 2016.

[40] J. Gilmer, S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *ArXiv*, vol. abs/1704.01212, 2017.

[41] A. Vaswani, N. M. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *ArXiv*, vol. abs/1706.03762, 2017.

[42] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, "End-to-end object detection with transformers," *ArXiv*, vol. abs/2005.12872, 2020.

[43] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. C.-F. Lin, and B. Guo, "Swin transformer: Hierarchical vision transformer using shifted windows," *ArXiv*, vol. abs/2103.14030, 2021.

[44] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," *ArXiv*, vol. abs/2103.13413, 2021.

[45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," in *NAACL*, 2019.

[46] Z. Hu, Y. Dong, K. Wang, and Y. Sun, "Heterogeneous graph transformer," *Proceedings of The Web Conference 2020*, 2020.

[47] P. Zhong, D. Wang, and C. Miao, "Knowledge-enriched transformer for emotion detection in textual conversations," in *EMNLP/IJCNLP*, 2019.

[48] B. Piotrowski and J. Urban, "Stateful premise selection by recurrent neural networks," in *LPAR*, 2020.

[49] E. Nudelman, A. Devkar, Y. Shoham, K. Leyton-Brown, and H. Hoos, "Satzilla: An algorithm portfolio for sat," 05 2004.

[50] J. Scott, A. Niemetz, M. Preiner, S. Nejati, and V. Ganesh, "Machsmt: A machine learning-based algorithm selector for smt solvers," *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 12652, p. 303, 2021.

[51] J. Hůla, D. Mojžíšek, and M. Janota, "Graph neural networks for scheduling of smt solvers," in *2021 IEEE 33rd International Conference on Tools with Artificial Intelligence (ICTAI)*, 2021, pp. 447–451.

[52] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[53] H. Nishimori, "Statistical physics of spin glasses and information processing," 2001.

[54] F. A. Aloul, A. Ramani, I. L. Markov, and K. A. Sakallah, "Solving difficult sat instances in the presence of symmetry," in *DAC '02*, 2002.

[55] P. Hurtík, V. Molek, J. Hula, M. Vajgl, P. Vlasánek, and T. Nejezchleba, "Poly-yolo: higher speed, more precise detection and instance segmentation for yolov3," *ArXiv*, vol. abs/2005.13243, 2020.

[56] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," *arXiv preprint arXiv:2103.13413*, 2021.

[57] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*. Springer, 2020, pp. 402–419.

[58] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1328–1338.

[59] J. Hula, D. Adamczyk, D. Mojzisek, and V. Molek, "Segmenting out generic objects in monocular videos," 2021.

[60] Q. Tan, L. Gao, Y.-K. Lai, and S. Xia, "Variational autoencoders for deforming 3d mesh models," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 5841–5850.

[61] R. Szeliski, *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[62] J. Masci, D. Boscaini, M. Bronstein, and P. Vandergheynst, "Geodesic convolutional neural networks on riemannian manifolds," in *Proceedings of the IEEE international conference on computer vision workshops*, 2015, pp. 37–45.

[63] J. Hula, "Unsupervised object-aware learning from videos," in *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*. IEEE, 2020, pp. 237–242.

List of Figures

List of Tables

A Encoding domain knowledge using generative models

Towards Visual Training Set Generation Framework

Jan Hůla( ), Irina Perfilieva, and Ali Ahsan Muhummad Muzaheed

University of Ostrava, 701 03 Ostrava, Czech Republic
{jan.hula,irina.pefilieva}@osu.cz
http://www.osu.cz, http://irafm.osu.cz

Abstract. Performance of trained computer vision algorithms is largely dependent on amounts of data, on which it is trained. Creating large labeled datasets is very expensive, and therefore many researchers use synthetically generated images with automatic annotations. To this purpose we have created a general framework, which allows researchers to generate practically infinite amount of images from a set of 3D models, textures and material settings. We leverage Voxel Cone Tracing technology implemented by NVIDIA to render photorealistic images in realtime without any kind of precomputation. We have build this framework with two use cases in mind: (i) for real world applications, where a database with synthetically generated images could compensate for small or non existent datasets, and (ii) for empirical testing of theoretical ideas by creating training sets with known inner structure.

1 Introduction

We introduce a framework for generating training data for supervised learning algorithms in computer vision. Currently, the most successful algorithms for computer vision tasks are very data hungry. They require large amount of training images, to be able to achieve low test errors. Creation of large training sets like ImageNet [RDS+15], which contain millions of labeled images is often considered as one of the main factors influencing the progress in machine learning. The creation of such large training sets is prohibitively expensive and for tasks like optical flow, where one would need to label every pixel, it is practically impossible. Some researches try to avoid this bottleneck by creating synthetic datasets, where the training examples are generated along with the labels [dSGCP16, SQLG15, GAGM15, XVL+13, IMS+16]. Because computer graphics can today create imagery, which has high degree of photorealism in reasonable time, it offers cheap way how to generate practically infinite amount of annotated images from limited amount of 3D models. Different configurations of 3D models, with different textures and materials applied to them, with different lighting and viewing conditions create combinatorially very large number of possible scenes. Due to the complete control over the rendering algorithm, all kinds of ground truth annotations could be generated along with the photorealistic images. These, beside other things, include: semantic labels,

I. Rojas et al. (Eds.): IWANN 2017, Part II, LNCS 10306, pp. 747–758, 2017.
DOI: 10.1007/978-3-319-59147-6_63

edges and depth maps (See Fig. 1). From these, other kinds of ground truth data could be calculated [MHLD16]. Unlike many of the previous attempts [MHLD16,HPB+15,RSM+16], that were concentrated on creation of specific datasets, we instead create a general framework, which allows to generate data according to the needs of the researcher. Our main contribution does not lie at bringing some new knowledge to the community, but at building a framework, which we hope will be useful for researchers in computer vision. In this paper we describe technical details, that were made to meet this purpose.

Fig. 1. Example of ground truth images. Top left: surface normals, Top right: Object ID, Bottom left: surface edges, Bottom right: depth map.

2 Problem Statement

Our aim is to create a framework which, given a database of 3D models, textures and scene descriptions, generates large amount of images together with ground truth annotations in reasonable time (e.g. days). In order to be sure about the results, we constrain ourselves to images of indoor scenes, which are much easier to make photorealistic than for example images of organic nature. To achieve this goal, two main subtasks should be identified. Assembling 3D scenes from individual models and rendering images from these scenes. Two main criteria, which influence our solution, are photorealism of the rendered images and the speed of rendering. We also develop this framework with two use case scenarios in mind:

- As a tool for real world applications, which compensates for small or non existent training sets.
- As an experimental framework, which generates data according to the need of the experiment. Because with computer graphics we could control almost every detail of the image, we could conduct truly controlled experiments.

More generally we try to accommodate our framework to the changing needs for different datasets and to prevent researchers from wasting their efforts by creating one shot datasets.

3 Related Work

Many researchers in the field of machine learning successfully leveraged synthetic datasets in the past. [GVZ16, WWCN12] generated large datasets of synthetic images with text to train OCR systems with great accuracy. [dSGCP16] trained a classifier for action recognition from rendered videos containing procedurally guided models of humans. [SQLG15] outperform state of the art view estimation methods on PASCAL 3D+ benchmark with convolutional neural network (CNN) trained on rendered objects. [GAGM15] use synthetic images of objects to train CNN, which replaces objects in RGB-D scenes with 3D models from a database. [PSAS14, AR15] study the invariances and features of CNN in experiments done on synthetically generated images. [HPB+15, MHLD16] assemble indoor scenes using models from ShapeNET database [CFG+15] in a same fashion as we do, and render large amount of RGB-D images with ground truth annotations from these scenes. Synthetic images are also used a lot for the task of pedestrian detection [XVL+13, HNBKK15, VLM+14]. [IMS+16] use CNN to estimate optical flow. Due to unavailability of training data for this task, they generate their own using computer graphics. [LVVG16] uses synthetic images to learn to decompose shading from albedo in real images. [RSM+16] produce virtual environment with the aim to train algorithms for self driving cars. [SDLK17] does the same but for drones. [RVRK16] figured out a method, how to leverage computer games for generation of labeled images. [QY16] released a plugin for Unreal Engine, which allows to render images from premade scenes. [LGF16, KWR+16, BCP+16] use game environments to train reinforcement learning agents. [SPT+16, BSD+16] make the synthetic images more realistic by using adversarial network to change the distribution of pixels. They then show, that thus modified images increase the accuracy of the classifier trained on them (compared to classifier trained on the original synthetic images). The main difference in our work is, that we are creating a general framework for rendering datasets instead of creating one concrete dataset or studying effects of learning from synthetic data.

4 Effectiveness of Synthetic Data Sets

As mentioned earlier, there are two use cases which could benefit from synthetic datasets. For real world applications, synthetic datasets could serve as data

augmentation technique. It is well known, that data augmentation is very useful technique, which could besides other thing reduce overfitting. In computer vision data augmentation is often done by introducing noise and different kinds of occlusion, mirroring the image horizontally or cropping the image at different positions, rotations and scales. Thus, from one image one creates many others. All these transformations are very limited and obviously do not capture too much of variations which arise in real world images.

On the other hand, when generating images synthetically, one could model any variation one wants. Every dimension (e.g. texture, lighting parameters, camera parameters) of variation creates many possible images, which could be generated, combinatorially giving rise to enormous pool of possible images, which could be very tedious to label manually. Because in computer graphics we have complete information about every pixel of the generated image, we get the labels for free. Lot of previous work which was described in last section has benefited from this fact.

The second use case for synthetic datasets is empirical testing of theoretical ideas. Here we believe our framework brings much larger gains and it was also our main motivation. As an analogy, one could imagine that the framework for generating synthetic images would in computer vision play similar role as synthetic biology plays in molecular biology. Synthetic biology constitutes great experimental framework, which allows researchers to test their theories quickly and interpret the results of the experiment with more certainty. For example when conducting experiments with synthetic cell, there is much less unknown variables involved than in experiments with biological cell. Similarly there is a lot of unknowns in natural images, which makes the interpretation of machine learning algorithms in computer vision very hard.

With training set generation framework one could generate dataset tailored for the experiment in mind. As an illustration example, one could test how the order of training examples affects the convergence of learning, the idea behind curriculum learning [BLCW09]. With such framework it is very easy to generate the training images in the order of some predefined criterion, and thus test the idea of curriculum learning. If, as in our case, the generation is realtime, the learning process could be made dynamic by establishing two way communication between the learner and the teacher. By teacher we here mean the score function together with the generation framework. The teacher could generate images on the fly, taking into account the behaviour of the learner.

5 Componentization

Instead of creating one monolithic piece of software, we have divided our framework into two separate components, which could be used independently. These two components can be used for following tasks:

– Stochastic scene generation: generating empty rooms with walls, windows and doors, generating furniture layouts, creating lighting of a scene

– Rendering: assigning materials to objects, creating camera settings and rendering images

Tasks in stochastic scene generation are specific to indoor scenes, the rest could be used for all other projects. This componentization offers easier reuse. Researchers could easily exchange any component with their own, if they adopt our interface.

6 Stochastic Scene Generation

For generating empty rooms and placing lights we use very simple heuristics, because we do not have any hints, why modeling them in more complex ways would benefit the learning. For empty room we first sample the type of room (e.g. bathroom, kitchen) which influences the size of the room and also the distribution of types of furniture. We set the dimensions of the room according to the distribution for that type of room. Next we sample count of windows and doors to use for that type of room and also their placement. The models of doors and windows are sampled from the database. We use open sourced modeling program Blender to model the rooms and save them to .obj files.

When creating lighting for the scene we sample number of lights according to the type of room and place the lights randomly below the ceiling and to objects (e.g. lamps) which have special sockets indicating the placement of light. We utilize IES profiles of lights, which are freely available and create realistically looking light effects (Fig. 2).

Fig. 2. Examples of various ies profiles of lights.

For generating furniture layouts we utilize factor graphs, kind of graphical model which allows us to specify probable configurations of objects by defining local constraints (factors) between them. In our case, these constraints encode information like distance of objects from each other (objects should not overlap, chairs should be near a table), distance of objects from walls (objects should not intersect with the wall, objects like wardrobe should be placed near the wall) and other relationships or ergonomic factors. For more involved set of factors see [YYT+11]. The value in which these constraints are unsatisfied in some

configuration is used as a penalty for the probability of that configuration. The probability of a configuration is equal to the product of local factors:

$$P(X) \sim \prod_{i=1}^{n} f_i$$

where X is a global configuration of variables and f_i is a local factor, whose arguments are variables, on which this factor depends.

One way to get a sample from this probability distribution is to use Markov Chain Monte Carlo algorithm (MCMC), where one creates Markov chain whose stationary distribution is equal to the desired probability distribution. After certain number of steps in this Markov chain, one can get correlated samples, whose probability is given by the desired distribution. Pure MCMC algorithm is very slow, therefore other variations of it are used in practice (e.g. simulated annealing, parallel tempering). For more complete description of how factor graphs and MCMC are used for stochastic scene generation see [YYT+11, YYW+12]. For our proof of concept we have used Locally annealed Reversible Jump MCMC developed by [YYW+12], mainly because it can generate layouts, where the count of objects is not fixed.

When synthesizing furniture layouts from databases of models like ShapeNET [CFG+15], there arises a problem with inconsistent metric units. We have used same heuristics as [MHLD16], who leveraged distribution of height of object categories, which is encoded in a dataset called SUN RGD-B [SLX15]. For drawbacks of this heuristic see [MHLD16].

7 Rendering, Materials and Camera Settings

Most of our attention during the process of designing this framework went to the rendering part, because it is most reusable. This part consist of three main tasks: assigning materials and textures to models, setting the camera parameters and rendering the images.

For assignment of materials it would be great to use semantic labels (information about the type of material) of 3D model, in order to choose texture and material to apply. We have implemented this functionality, but it relies on specific details, which may not be present. Concretely on the type of material of the model (or its subpart) and real world scale of its UV maps and textures from which we choose. Given the information about material type, we could assign a material and texture, which have the properties fulfilling the properties of the material type. To this purpose we have collected set of material settings, which imitate some real world materials. As a concrete example, if the model would have material type *wood*, then we could choose from different material settings, which imitate different types of wood (e.g. polished, brushed) and for this again different types of textures could be used (e.g. oak, beech, maple). This would create very large number of possible object appearances.

Unfortunately, the database of models we currently use (ShapeNET [CFG+15]) does not contain semantic labels for materials types or information

about the scale of UV maps. Therefore, in these cases, we load the materials, which were created by the author of the model. We hope for better standardization, so that curated collections of models and textures can be used for creating very large number of possible appearances of objects.

The most critical part of our framework was the rendering engine. As mentioned previously, the main criterion affecting our framework was photorealism of images and speed of rendering. One could rightly ask, how is the accuracy of a trained algorithm affected by the photorealism of the training set, when it is tested on real images. It could possibly be the case, that the algorithm trained on large sample of artificially looking images would give similar accuracy as if it was trained on photorealistic images. Results of [SPT+16] show, that it is not so, as one would intuitively expect. They have trained a kind of adversarial network [GPAM+14], which takes as input the synthetic image and as an output it produces semantically same image, but whose pixel distribution is much closer to that of natural images (they look more realistic). They then show that discriminative algorithm trained on thus modified images achieves significant improvement over the same algorithm trained on the original synthetic images. This is our justification for the effort to reach high photorealism.

There is obviously a trade off between the photorealism of rendered images and speed of rendering. On one side, in game industry there exist rendering engines, which produce dozens of images per second, but one could definitely recognize that they are not real. On the other side, rendering engines developed for special effects industry produce images, which are often indistinguishable from real images, but take hours to render. In the following paragraphs we describe rationale behind decisions we made when choosing a rendering technology.

We base our decisions on the number of images we would like to generate. This number is specific to the use case. For small experiments we would like to generate thousands of images, but for real world applications the number of generated images would be ideally around one million and more. If the image would take 10 s in average to render, then million images would render for almost 4 months. This could be possibly justifiable for some large projects, or if the resulting dataset would be reused many times by some community, but for our purposes this is too prohibitive. Also for small experiments we would like to be able to iterate the generation many times as new ideas appear or some deficiencies are discovered. Generation of 10 000 images with rendering time 10 s per image would take almost whole day.

The rendering technology for creating highly realistic images is often based on some form of path tracing, which simulates how the light rays bounce around the scene. It is often implemented in progressive form, which means that the image is being continuously updated as the simulation progresses. One can stop the simulation, when the quality of the image is satisfying. This allows the user to make a trade off between quality and speed. Unfortunately, for indoor scenes, the quality of the images rendered under 10 s is very poor. One could utilize few tricks like caching the information needed to calculate the radiance values (photon mapping) and reusing this cache across many frames from the same

scene and thus amortizing the cost of the calculation, but the results are still unsatisfying and it works only for static scenes. For few examples see [MHLD16]. They managed to achieve render times below 3 s on average, but the resulting images look noisy with visible splotches from the under sampled photon map. The problem with noise could be partially remedied by denoising algorithms tailored specially for Monte Carlo type renderers [BRM+16]. These algorithms could improve the quality of the image significantly, but their run times is in tens of seconds. They are used to denoise images, which normally take many minutes to render, so there the run time of denoising is negligible. For our purposes their run time is unacceptable. Possibly in few years the path traced and similar renderers will be producing images of satisfying quality in seconds or even less, but for our project we have finally started to search for something else.

Eventually we have decided to render the images with game engine, which could render many frames per second. When neglecting the loading time of the scenes and saving the images, we could render around 15 high resolution frames per second on NVIDIA GTX980Ti graphics card. Concrete frame rates depends on lighting setup and few other settings. Thus, neglecting the cost of loading the models, we could render one million images in less than a day.

Concretely, we have chosen Unreal Engine 4 (UE), which is currently among the leaders in terms of photorealism and at the same time it is open sourced. The photorealism of UE is constantly getting better and its maintenance is almost guaranteed, thus we get improvements for free. The main hindrance for us was the fact, that until recently the photorealism in game engines was achieved by light maps, which are precomputed during the creation of the game environment. These light maps often take few minutes to precompute, which would make whole engine unusable for our purposes.

Fortunately for us, new kind of technology was recently developed for real-time graphics, which achieves very realistically looking images without the need of precomputing lightmaps. It is called Voxel Cone Tracing and it computes Indirect illumination by dynamically dividing 3D scene to voxels, which approximate the original geometry [CNS+11]. Researchers from NVIDIA have implemented this algorithm under codename VXGI and they also released a version of UE, which uses their VXGI technology. We are using this version of UE for rendering part of our framework. UE also provides setting, which enables to mimic real world camera. It contains settings for exposure, depth of field, lens distortions, motion blur and dozen other settings. The code for loading a geometry, applying materials, creating lights and rendering the images is packaged in a plugin, which could be installed into UE. We have reused lot of work from [QY16], who released a plugin called UnrealCV, which allows to call commands in UE from programming language Python. This enables researchers to use our commands without learning about the architecture of UE. At the same time it allows to combine our generation framework with machine learning libraries like TensorFlow [AAB+16]. The framework could generate images at the time of learning, although at this point this would make the learning prohibitively slow. We will be releasing the code for whole project after the presentation of this paper (Fig. 3).

Fig. 3. Comparison of scene rendered using UE (left) and the same scene rendered using off the shelf path tracing renderer (right). The images were rendered in same resolution (1600×1024 px). The left image was rendered under 100 ms, the right took around 20 min to render.

8 Use Case Example

We provide simple use case example of how our framework simplified the creation of testing data for geometrical edge detection algorithm. By the term geometrical edge we mean every edge which was created by discontinuities in the geometry (as opposed to discontinuities in light or texture).

We were faced with the problem of creating a small benchmark for a competition in geometrical edge detection. Creating ground truth annotations for natural images manually would be very time consuming. At the same time, we wanted also to create easier version of the benchmark, which would not contain textures. This was very easy to create with our framework. We have reused the scenes from [HPB+15] and rendered test images with ground truth annotations in matter of seconds. Unfortunately some images required manual correction, but

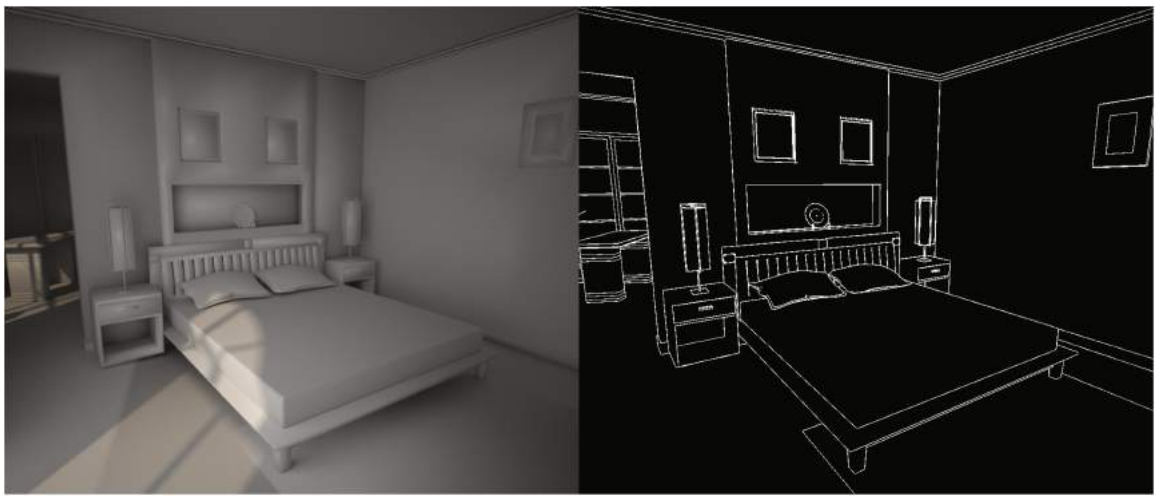


Fig. 4. Example of images from benchmark in geometrical edge detection

it still spared us a lot of manual work. See Fig. 4 for example pair of images. All images are placed on website of the competition: http://irafm.osu.cz/edge2017.

9 Conclusion

We have described an implementation of a framework, which allows to generate large collection of realistically looking images with ground truth annotations in small amount of time. It is achieved by exploiting combinatorially large set of possible configurations of different models, materials, lighting and viewing conditions. We have developed our framework with two use cases in mind. For augmenting a training set in order to achieve higher accuracy in real world applications and also for generating datasets in a very controlled manner, which is very useful for testing theoretical ideas empirically. As a main bottleneck we regard the lack of databases with high quality models and material annotations. Our future work will focus in this direction. We will also focus on photorealism of the generated images, mainly by incorporating the work of [SPT+16] in adversarial image modification. Lastly we will incorporate more recent algorithms for stochastic scene generation.

References

[AAB+16] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., et al.: Tensorflow: Large-scale machine learning on heterogeneous distributed systems. arXiv preprint arXiv:1603.04467 (2016)

[AR15] Aubry, M., Russell, B.C.: Understanding deep features with computer-generated imagery. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2875–2883 (2015)

[BCP+16] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym. arXiv preprint arXiv:1606.01540 (2016)

[BLCW09] Bengio, Y., Louradour, J., Collobert, R., Weston, J.: Curriculum learning. In: Proceedings of the 26th Annual International Conference on Machine Learning, pp. 41–48. ACM (2009)

[BRM+16] Bitterli, B., Rousselle, F., Moon, B., Iglesias-Guitián, J.A., Adler, D., Mitchell, K., Jarosz, W., Novák, J.: Nonlinearly weighted first-order regression for denoising monte carlo renderings. Comput. Graph. Forum **35**, 107–117 (2016). Wiley Online Library

[BSD+16] Bousmalis, K., Silberman, N., Dohan, D., Erhan, D., Krishnan, D.: Unsupervised pixel-level domain adaptation with generative adversarial networks. arXiv preprint arXiv:1612.05424 (2016)

[CFG+15] Chang, A.X., Funkhouser, T., Guibas, L., Hanrahan, P., Huang, Q., Li, Z., Savarese, S., Savva, M., Song, S., Hao, S., et al.: Shapenet: An information-rich 3d model repository. arXiv preprint arXiv:1512.03012 (2015)

[CNS+11] Crassin, C., Neyret, F., Sainz, M., Green, S., Eisemann, E.: Interactive indirect illumination using voxel cone tracing. Comput. Graph. Forum **30**, 1921–1930 (2011). Wiley Online Library

[dSGCP16] de Souza, C.R., Gaidon, A., Cabon, Y., López Peóa, A.M.: Procedural generation of videos to train deep action recognition networks. arXiv preprint arXiv:1612.00881 (2016)

[GAGM15] Gupta, S., Arbeláez, P., Girshick, R., Malik, J.: Aligning 3D models to RGB-D images of cluttered scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 4731–4740 (2015)

[GPAM+14] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative adversarial nets. In: Advances in Neural Information Processing Systems, pp. 2672–2680 (2014)

[GVZ16] Gupta, A., Vedaldi, A., Zisserman, A.: Synthetic data for text localisation in natural images. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2315–2324 (2016)

[HNBKK15] Hattori, H., Boddeti, V.N., Kitani, K.M., Kanade, T.: Learning scene-specific pedestrian detectors without real data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3819–3827 (2015)

[HPB+15] Handa, A., Patraucean, V., Badrinarayanan, V., Stent, S., Cipolla, R.: Scenenet: Understanding real world indoor scenes with synthetic data. arXiv preprint arXiv:1511.07041 (2015)

[IMS+16] Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., Brox, T.: Flownet 2.0: Evolution of optical flow estimation with deep networks. arXiv preprint arXiv:1612.01925 (2016)

[KWR+16] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., Jaśkowski, W.: Vizdoom: A doom-based AI research platform for visual reinforcement learning. arXiv preprint arXiv:1605.02097 (2016)

[LGF16] Lerer, A., Gross, S., Fergus, R.: Learning physical intuition of block towers by example. arXiv preprint arXiv:1603.01312 (2016)

[LVVG16] Lettry, L., Vanhoey, K., Van Gool, L.: Darn: a deep adversial residual network for intrinsic image decomposition. arXiv preprint arXiv:1612.07899 (2016)

[MHLD16] McCormac, J., Handa, A., Leutenegger, S., Davison, A.J.: Scenenet RGB-D: 5m photorealistic images of synthetic indoor trajectories with ground truth. arXiv preprint arXiv:1612.05079 (2016)

[PSAS14] Peng, X., Sun, B., Ali, K., Saenko, K.: Exploring invariances in deep convolutional neural networks using synthetic images. CoRR, abs/1412.7122 **2**(4) (2014)

[QY16] Qiu, W., Yuille, A.: UnrealCV: connecting computer vision to unreal engine. In: Hua, G., Jégou, H. (eds.) ECCV 2016. LNCS, vol. 9915, pp. 909–916. Springer, Cham (2016). doi:10.1007/978-3-319-49409-8_75

[RDS+15] Russakovsky, O., Deng, J., Hao, S., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)

[RSM+16] Ros, G., Sellart, L., Materzynska, J., Vazquez, D., Lopez, A.M.: The synthia dataset: a large collection of synthetic images for semantic segmentation of urban scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3234–3243 (2016)

[RVRK16] Richter, S.R., Vineet, V., Roth, S., Koltun, V.: Playing for data: ground truth from computer games. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9906, pp. 102–118. Springer, Cham (2016). doi:10.1007/978-3-319-46475-6_7

[SDLK17] Shah, S., Dey, D., Lovett, C., Kapoor, A.: Aerial informatics and robotics platform. Technical report MSR-TR-9, Microsoft Research (2017)

[SLX15] Song, S., Lichtenberg, S.P., Xiao, J.: Sun RGB-D: a RGB-D scene understanding benchmark suite. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 567–576 (2015)

[SPT+16] Shrivastava, A., Pfister, T., Tuzel, O., Susskind, J., Wang, W., Webb, R.: Learning from simulated and unsupervised images through adversarial training. arXiv preprint arXiv:1612.07828 (2016)

[SQLG15] Su, H., Qi, C.R., Li, Y., Guibas, L.J.: Render for CNN: viewpoint estimation in images using CNNs trained with rendered 3D model views. In: Proceedings of the IEEE International Conference on Computer Vision, pp. 2686–2694 (2015)

[VLM+14] Vazquez, D., Lopez, A.M., Marin, J., Ponsa, D., Geronimo, D.: Virtual and real world adaptation for pedestrian detection. IEEE Trans. Pattern Anal. Mach. Intell. **36**(4), 797–809 (2014)

[WWCN12] Wang, T., Wu, D.J., Coates, A., Ng, A.Y.: End-to-end text recognition with convolutional neural networks. In: 21st International Conference on Pattern Recognition (ICPR), pp. 3304–3308. IEEE (2012)

[XVL+13] Xu, J., Vázquez, D., López, A.M., Marin, J., Ponsa, D.: Learning a multiview part-based model in virtual world for pedestrian detection. In: IEEE Intelligent Vehicles Symposium (IV), pp. 467–472. IEEE (2013)

[YYT+11] Yu, L.F., Yeung, S.K., Tang, C.K., Terzopoulos, D., Chan, T.F., Osher, S.J.: Make it home: automatic optimization of furniture arrangement (2011)

[YYW+12] Yeh, Y.-T., Yang, L., Watson, M., Goodman, N.D., Hanrahan, P.: Synthesizing open worlds with constraints using locally annealed reversible jump MCMC. ACM Trans. Graph. (TOG) **31**(4), 56 (2012)

Synthetic Dataset for Compositional Learning

Vojtech Molek and Jan Hula

Institute for Research and Applications of Fuzzy Modelling, University of Ostrava, Ostrava, 701 03, Czech Republic
E-mail: vojtech.molek@osu.cz
E-mail: jan.hula@osu.cz
irafm.osu.cz

This contribution presents a framework for a generation of synthetic images. The framework is built on top of the Unreal Engine 4, a software kit capable of rendering realistic images. Besides image data, additional label information, such as depth, normal maps and object components masks, are generated. Hierarchical nature of generated labels corresponds to hierarchical representations which we want to be captured by the neural network. Such labels enable training of deep models in a compositional manner. This leads to the better understanding of the internal representations of the models and acceleration of the learning procedure. The framework allows users to render arbitrary scenes and objects according to their specific domain.

Keywords: dataset, synthetic data, unreal engine, Compositional Learning

1. Introduction

In the last decade, deep neural networks have overcome traditional approaches in tasks such as regression or classification.[1,2] Such progress was enabled by new processing devices, new algorithms and last but not least availability of large labeled datasets.[3–6] Large datasets are key to proper supervised learning, however, the creation of such datasets is very tedious and time-consuming. Moreover, a lot of information about data is not explicitly contained in the labels (such as edges, contours, depth maps, segmentation masks, bounding boxes, occlusion, orientation etc. in case of image data). In some instances, it is even practically impossible to create all such labels manually. Having more control over data gathering process and more sensor types would lead to additional label information, although it could be impractical or even impossible in real world.

One way to simplify the process of data gathering is to use data from

simulation instead of real world. Instead of gathering, we *generate* synthetic data. During generation of synthetic images, we generate different labels, including labels for object components - i.e., paws, body, tail, ears, eyes, and whiskers which together form a concept of a cat. Obviously, this hierarchy has variable complexity. We discuss the relation of hierarchical labels to training neural networks in section 2. In section 3 we describe our synthetic dataset and issues it potentially solves. In the last section 4, we introduce our framework for generating synthetic datasets.

2. Idea of Compositional Learning

During the learning process, Deep Learning[7] models learn a hierarchy of representations (in form of *features*) where some of the learned features have recognizable meaning and correspond to components of objects. For example, to recognize the aforementioned cat, a deep model will learn to detect components of a cat, even without explicitly assigned labels to them. The hierarchy of features was previously demonstrated by various features visualization methods.[8–13]

The main idea of *Compositional Learning* is to help a model in searching the parameter space by explicitly forcing hierarchical parts of the model to learn the features we believe to be important for recognition. In the context of image recognition, this is achieved by providing additional labels for the components of the object (paws, body, tail, etc.) and extending the objective function to take them into account. Concretely, certain convolutional kernels in the layers of a convolutional network would be forced to detect given components of the object. Thus, the objective function measures the accuracy of the whole network as well as how the said kernels detect the assigned components.

3. Synthetic Dataset

For the purposes of testing the idea of Compositional Learning, we have created synthetic dataset of images of plants. Because of the complete control over the 3D environment and complete knowledge about the objects it contains, we have generated the images with labels for every pixel in a trivial manner.

The 3D models used for the rendering may be obtained in multiple ways. The 3D models can be created by artists or by scanning real objects with a 3D scanner. However, such 3D models, which are not procedural, have fixed geometry. Therefore the rendered images will not contain any

variation in the geometry. Another issue with scanned 3D models is the need to manually separate components of a 3D object if one wants to render different variations of textures for each component separately.

A much better approach is to create the geometry using a generative modeling. A generative modeling consists of programs for generating the geometry according to some parameters. By changing the parameters, the program generates different variations of the same object. For our purposes, we have decided to use generative models of plants as plant generation is a well-developed area of computer graphics and therefore many generative models of various plants are available. By using these 3D models, we generate variations of the same plant species with separate components of the plant that can also vary.

4. Synthetic data engine

For the purposes of rendering images from 3D models, we have chosen Unreal Engine 4 (UE4). UE4 is a game engine with open source code, capable of realistic rendering[a]. Realistic rendering is crucial to bridge the gap between synthetic data and real-world data. Moreover, it contains design tools for creating 3D sceneries. This enables to create realistic background context for rendered images.

Because our goal was to generate data as fast as possible, ideally in real time, we have used UE4 with NVIDIA *Voxel Global Illumination*[14] technology (VXGI) publicly available at Github[b]. The VXGI is global illumination approximation technique able to illuminate scenes in roughly real time and without pre-computations. This enables us to dynamically modify scenes during a game play.

To control the environment in UE4, we have extended a project called *UnrealCV*.[15]UnrealCV is UE4 plugin that let users manipulate certain aspects of UE4 through pre-defined commands such as camera manipulation, objects masking and rendering the images. These commands are invokable from Python which makes integration with popular machine learning frameworks easy and convenient.

[a] `https://docs.unrealengine.com/latest/INT/Resources/Showcases/RealisticRendering/`
[b] `https://github.com/NvPhysX/UnrealEngine/`

4.1. *Extending UnrealCV*

The UnrealCV uses server-client architecture. The server is embedded into the running game and communicates with the client. The user uses the client to send commands and receive responses. The set of featured commands, however, does not allow to load objects into a scene during a game play. Therefore a user is not able to add and/or remove objects from a scene.

In our work, we have added a possibility to load objects during the game play with the command: `vget /load [example]` where `example` is a name of a configuration file. The configuration file is in the JSON format and includes the following information about the object: labels, meshes, materials, lights, and properties such as scale, location, rotation and material characteristics. All data has to be in a proprietary format (*uasset*), and we, therefore, convert the models from *fbx* to uasset file format with our file converter.

4.2. *Generating data*

Let us consider an example, where we generate image data of a sunflower. We start with picking scene. The unreal engine has a wide range of assets (paid and free) available at its online shop. We pick adequate scene[c] (Fig.1) where the flower fits in naturally. In the next step, we load our model(s) through JSON file (Fig.2). The loaded sunflower consists of multiple separated parts (Fig.3). These parts can be visualized with view mode `object_mask` and such representation serves as ground truth labels for these parts. The generated data are shown in Fig.3.

The source code used to generate example at Fig.3 is hosted at Github[d]. The repository will include an example of Compositional learning as well as the generated dataset.

5. Conclusion

In our contribution, we have described how 3D graphics and its sophisticated tools can help to create and expand training datasets. We have extended UnrealCV by ability of dynamically load object into scene during game play. The object loading is done by adding new command to UnrealCV. Properties of the loaded object is specified in JSON file and has to

[c]Publicly available at `https://www.unrealengine.com/marketplace/open-world-demo-collection`.
[d]`https://github.com/Jan21/unrealcv`

Fig. 1. 3D scene of the forest.

Fig. 2. 3D scene of the forest with loaded flower.

be in the proprietary format called uasset. For this purpose we have created a format converter. We have also provided helper method to render images of an object with labels from different angles. This way, we have made the whole process of generating synthetic data easily manageable. The combination of user-friendly Python programming language, rich assets of UE4, and easy access to control rendering is a powerful tool for continuous synthetic data generation during/for training. Further, we have discussed how Compositional Learning can take advantage of synthetic data generation and its rich labels. The future work will include experimental usage of Compositional Learning and expansion of the dataset generating framework.

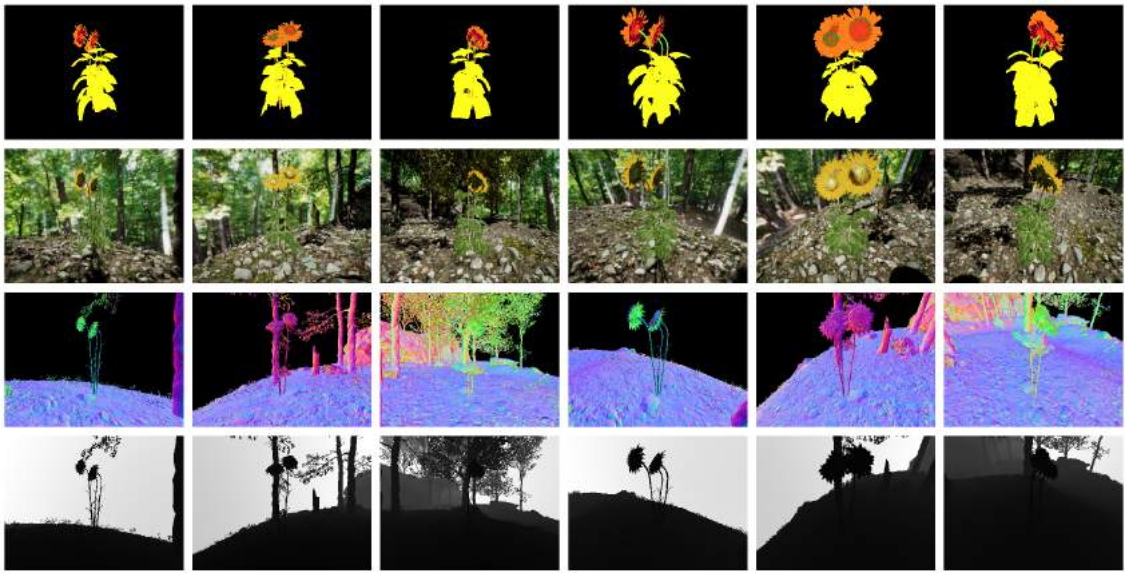


Fig. 3. Columns: Sunflower from 0°, 120°and 240°with 0°and 30°rotation in upward direction (along y axis); Rows: First row are semantic labels (masks), second row are images of fully lit sunflower, third row are images of normal maps of a model and last row is depth of scene.

References

1. K. He, X. Zhang, S. Ren and J. Sun, *CoRR* **abs/1512.03385** (2015).
2. K. He, G. Gkioxari, P. Dollár and R. B. Girshick, *CoRR* **abs/1703.06870** (2017).
3. O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg and F. Li, *CoRR* **abs/1409.0575** (2014).
4. J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick and R. B. Girshick, *CoRR* **abs/1612.06890** (2016).
5. R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L. Li, D. A. Shamma, M. S. Bernstein and F. Li, *CoRR* **abs/1602.07332** (2016).
6. T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár and C. L. Zitnick, *CoRR* **abs/1405.0312** (2014).
7. I. Goodfellow, Y. Bengio, A. Courville and Y. Bengio, *Deep learning* (MIT press Cambridge, 2016).
8. D. Erhan, Y. Bengio, A. Courville and P. Vincent, *University of Montreal* **1341**, p. 1 (2009).
9. K. Simonyan, A. Vedaldi and A. Zisserman, *arXiv preprint arXiv:1312.6034* (2013).
10. A. Mahendran and A. Vedaldi (2015).
11. A. Nguyen, J. Yosinski and J. Clune, *arXiv preprint arXiv:1602.03616* (2016).
12. D. Bau, B. Zhou, A. Khosla, A. Oliva and A. Torralba, Network dissection:

Quantifying interpretability of deep visual representations, in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, 2017.
13. A. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox and J. Clune, Synthesizing the preferred inputs for neurons in neural networks via deep generator networks, in *Advances in Neural Information Processing Systems*, 2016.
14. A. Panteleev, *ACM SIGGRAPH 2014 presentations* (2014).
15. W. Qiu, F. Zhong, Y. Zhang, S. Qiao, Z. Xiao, T. S. Kim and Y. Wang, Unrealcv: Virtual worlds for computer vision, in *Proceedings of the 2017 ACM on Multimedia Conference*, 2017.

Acquiring custom OCR system with minimal manual annotation

Jan Hula*, David Mojžíšek, David Adamczyk, Radek Čech
Institute for Research and Applications of Fuzzy Modeling
Ostrava, Czechia
*jan.hula@osu.cz

***Abstract*—We describe a development of a custom OCR system, which is designed specifically for a linguistic analysis of texts printed during the early modern period. This analysis requires precise detection of individual graphemes, and we, therefore, could not apply standard approaches that transcribe whole lines in an end-to-end fashion. We also describe our use of synthetically generated images, which allow us to avoid manual annotation of a large training set.**

***Index Terms*—OCR, Synthetic Data, Historical Texts, Neural Networks**

I. Overview

Optical Character Recognition (OCR) is nowadays considered a solved problem with very little space for innovation [1]. Still, some applications have very specific requirements for which custom solutions are needed. Here, we focus on a problem arising in a linguistic analysis of printed texts from the early modern period (1500-1750). The solution to this problem must include a precise detection of graphemes because the analysis will deal with relative sizes of printed graphemes and spaces between them as described in the next section. Because we have not found an open-source system which would fulfill our requirements, we decided to design the system by ourselves. It follows a two-stage pipeline. The first stage is based on a state-of-the-art model for object detection to detect individual graphemes and in the second stage, individual graphemes are recognized using bidirectional LSTM [2] which recognizes every grapheme in a context in which it appears. We solve these two stages separately. After training the grapheme detection model, we use it to detect thousands of instances of generic graphemes which we cut-out and cluster using a simple K-means algorithm. We then label these clusters and use them to pre-label training examples for the recognition model.

As annotating bounding boxes for individual graphemes can be a tedious task, we wanted to avoid as much of manual annotation as possible. For this purpose, we synthesize artificial examples of printed pages which help us to bootstrap a labeling process with a model pre-trained on these examples. Our methodology follows a simple principle where we train a weaker method requiring few training examples to pre-label training data for a stronger method. In this contribution, we describe the whole pipeline of our solution. We believe that it contains ideas which can generalize to similar problems.

II. Motivation and Problem Statement

The approach presented in this paper is motivated by needs that have emerged in linguistics focused on the development of the Czech orthographic system. Specifically, Voit [3] introduced new explanation of the usage of orthographic variations (such as ”uo” $\sim$ ”ů” / ”ú”, ”ie” $\sim$ ”ij” $\sim$ ”j”) in 16th century. He claims that the usage of either singlegraphic or digraphic grapheme was caused by pragmatic factors related to typesetting praxis. According to him, a typesetter was forced to fulfill the following requirements: 1) to align the right edge of the text, 2) to avoid splitting of words at the end of the line. In other words, an option to use either longer or shorter realizations of the grapheme was a tool for dilation or compression of text in the line.

This explanation offers setting up several empirically testable hypotheses. For instance, ”the higher the number of types in the line, the higher probability of occurrence of digraphic grapheme” or ”the higher the number of types in the line, the lower the number of spaces in the line”. Testing of hypotheses of this kind must be performed on a large sample of original texts. Further, proper testing needs careful operalization, which is not a trivial task in this case. For instance, a width of both the grapheme and space must be determined unambiguously. To our knowledge, up to now, only in [4], the problem had been analyzed empirically. However, they used the sample consisting of only four texts which were transcribed and annotated manually. The lack of adequately processed documents is the main obstacle for a thorough analysis of this phenomenon, and this could be solved by automatization of the annotation process.

In other words, we want to prepare an annotated dataset (together with the annotation tool) for further linguistic analysis. This analysis should confirm or disprove the hypothesis that changes in written Czech language were driven by technological limitations and needs in typesetting practice.

III. Description of the data

In this section, we briefly describe our data and their specifics. We are working with scanned printed documents mainly from the second half of the 17th century. They were printed in different Czech cities (Praha, Litomyšl, Olomouc, etc.). Not all of the documents are clearly readable - few pages are torn, or the text is faded out. Also, the typeset is very specific and different from contemporary documents. Even Czech

native speakers are not able to fluently read such texts if they are not trained. In figures 1,2,3 we present a few examples that point out some problematic areas in our dataset. Our dataset contains hundreds of scanned documents without annotations. We also dispose of dozens of transcriptions containing similar language, which we leverage when training the classification model, as described in section VIII.

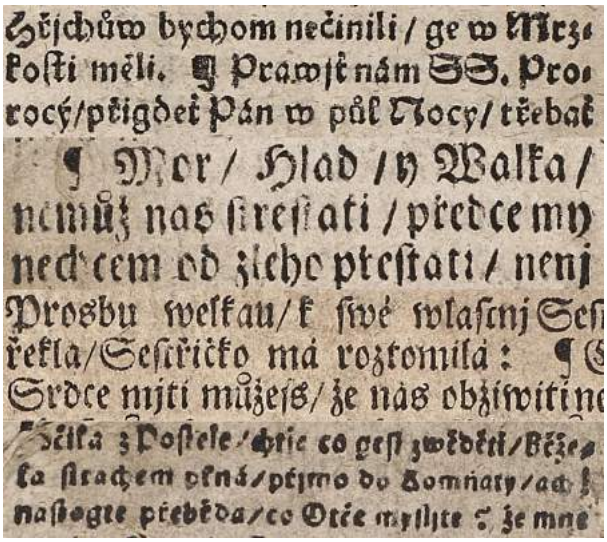


Fig. 1: Altough these text are from approximately the same period, they contain different typesets.

Fig. 2: Character difficulties. a) Ligatures, i.e. two or more characters printed as one. b) Some glyphs are hard to read and distinguish when the context is unknown.

IV. Outline of Our Pipeline

As most of OCR systems in use [5]–[8], we developed a pipelined system in which we solve individual steps independently of others. On a high level, we rotate each document so that lines are horizontally oriented, then we detect all graphemes in a given document, and finally, we classify each grapheme in the context of the text it appears in. The schematic representation of this process is depicted in figure 4. Most of modern OCR systems do not work by detecting individual graphemes but transcribe whole lines in an end-to-end fashion. This approach has the benefit that it does not require annotated

Fig. 3: Difficult pages. a) Some documents are damaged (for example folded or torn) or the qaulity of scan is low. b) Page containing an image and faded graphemes of different size.

bounding boxes. In our case, obtaining these bounding boxes is necessary, so we decided to split the pipeline to grapheme detection stage and grapheme classification stage. We could have also tried to classify and detect the graphemes in parallel, as it is usually done in object detection. By doing so, we would miss the opportunity to incorporate the structure of the text into the classification. The detection network would classify the grapheme as a patch in the image, instead of a grapheme in the sequence of graphemes. The second minor benefit of separating these two stages is the fact that we do not need to annotate the class of every bounding box, as will be described in section VIII. To obtain a final version of our system, we rely heavily on synthetic data and other tricks which allow us to avoid as much manual annotation as possible. Here follows a description of all the steps we execute during the construction of our system:

Alignment part

1) Train a neural network *NN-rot* (VGG-11) to regress angles of rotated images.
2) Use *NN-rot* to straighten all images in the dataset.

Detection part

3) Annotate bounding boxes around graphemes in 3 random pages.
4) Use the annotated graphemes to generate a large training set of synthetic images.
5) Pre-train a neural network *NN-det* to detect graphemes in synthetic images.
6) Fine-tune *NN-det* on the 3 annotated real images.
7) Pre-label 6 new pages with *NN-det* and correct wrong detections to enlarge the annotated set.
8) Repeat steps 4-7 two more times to increase the variability of the dataset and to obtain an accurate detection model.

Classification part

9) Detect few thousands of graphemes in still unlabeled images using *NN-det* trained in previous steps.
10) Cut out the detected graphemes and train an autoencoder to obtain low-dimensional representations for every grapheme.

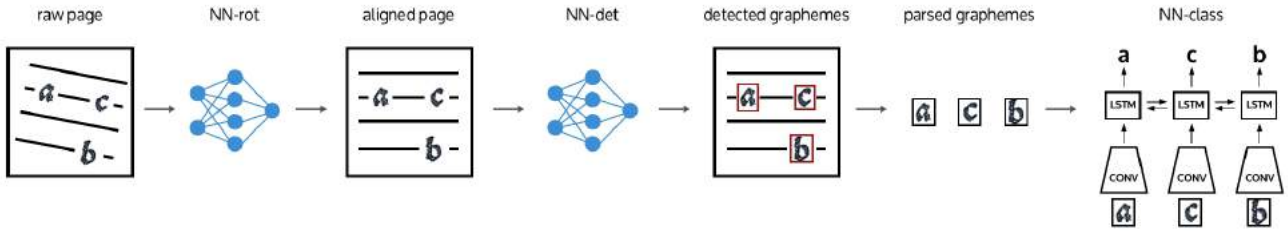


Fig. 4: A schematic representation of the pipeline used for every new image. The system first alignes the page according to a predicted angle from *NN-rot*; next it detects all graphems on the page with *NN-det*; these graphemes are parsed into a sequence which is finally classified by *NN-class*

11) Cluster the graphemes using K-means based on the representation from the autoencoder.
12) Create a labeled dataset of graphemes by manually labeling the clusters and deleting incorrectly assigned graphemes.
13) Use available transcriptions of texts from the same period containing the same language to generate training sequences of cut-out graphemes.
14) Train a convolutional bi-LSTM *NN-class* to classify each grapheme in the context of other graphemes using training examples from step 13.

By following these steps, we avoid manual labeling of hundreds of pages. Their details will be described in the subsequent section. At the end, we use *NN-rot*, *NN-det*, and *NN-class* for every new image.

V. Page Alignment

The scanned documents in our dataset were not always aligned, and therefore, individual lines were not aligned horizontally. Page alignment is a part of all OCR systems because when the letters are aligned, the subsequent recognition model does not need to learn rotation invariance. We decided to train a neural network to predict the angles from cropped patches of the image. Fortunately, obtaining a labeled dataset for this task does not require a lot of manual effort. We manually aligned 30 pages, and these aligned images are then rotated by a random angle from an interval -15 to +15 degrees, quantized to increments of 0.5. Subsequently, crops are taken from these rotated images, and the rotation angle is saved as their target label. We train a convolutional network with ResNet-18 backbone [9] to predict these angles, and for every new image, we average predictions from crops taken from it. We achieve nearly perfect accuracy ($\sim$98%) with this approach.

VI. Synthetic Data and Domain Knowledge

One of the biggest drawbacks of data-driven approaches for problem-solving is that they often need a lot of labeled examples to be trained on. To avoid this problem and save a lot of manual annotation, we decided to use synthetic data, which we generate using our understanding of the problem domain. Synthetic data has been recently used in all kinds of domains of Computer Vision and Machine Learning in general [10]–[13], and their use in OCR for modern print marks one of their first successful use-case [14]. The main pitfall of using synthetic datasets to train systems for real data is that the training distribution may be very different from the testing distribution because, in some domains, it may not be trivial to synthesize realistic examples (e.g., synthesizing realistic images of human faces). OCR for modern print was a successful use-case mainly because it is possible to create highly realistic synthetic examples using known fonts and simple image distortions and noise. Creating realistic examples of old prints is more involved because the variability of the appearance of graphemes is larger due to all kinds of problems arising during the printing process (e.g., leakage of ink) and degradation of documents after long periods. Examples of such problems can be seen in figures 2 and 3.

After a visual inspection of many real documents, we model the realism and variability of the page as closely as possible. For this, we use cut-out examples of various graphemes [1] with background removed and documents containing blank pages. We generate each page by sampling random words from which we create whole lines and place them to an empty page. On each grapheme, we apply a random set of augmentations simulating fading of the ink, elastic distortion, and various kinds of noises and scratches. To gain the variability in the background, we also apply similar augmentations on the few blank pages we had at our disposal. Also, many documents contained random drops of ink, which could be possibly mistaken for a grapheme, and therefore we add such drops to the background at random positions. An example of a such generated page can be seen in figure 5. For our experiments, we created 300 synthetic pages together with ground truth bounding boxes for every grapheme.

VII. Glyph Detection

As described in section IV, we first detect all graphemes as one generic class, then we parse the segmented graphemes to a sequence of these graphemes, and finally, we classify each grapheme using a sequence-based model. For the detection of generic graphemes, we use a state-of-the-art model for object

[1]Synthetic data described in this section are used only to detect generic graphemes, i.e., to detect a grapheme without classifying it into a class.

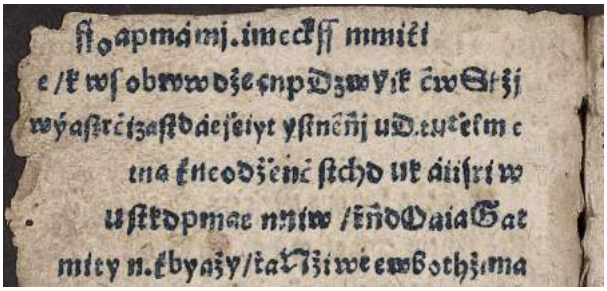


Fig. 5: An example of a crop from artificially generated page. We used different grapheme and background augmentations to add variability.

detection called RetinaNet [15], which we slightly modify to suit our task. Object detection models are usually categorized into one-stage and two-stage methods. As the names suggest, one-stage methods classify and detect objects in one stage, whereas two-stage methods first propose a candidate bounding boxes, which are then refined and classified in the second stage. One-stage methods are usually faster but less precise due to the imbalance of positive and negative bounding box proposals. RetinaNet solves this short-coming of one-stage methods by using special loss function called Focal Loss [15], which takes this imbalance into account. Therefore RetinaNet is a fast and accurate architecture for object detection.

Most of the time, models for object detection use pre-trained backbones (feature extractors) trained on big classification datasets such as ImageNet [16]. These backbones extract already useful features that can be leveraged by subsequent layers in the network. The pre-trained backbone is most useful when the pre-training domain is similar to the target domain. In our case, the images of scanned printed documents are very different from photographs capturing random objects, and therefore we do not use a pre-trained backbone but train it from scratch on our synthetic dataset.

As most of the one-stage detectors, RetinaNet uses anchors when detecting individual objects. During detection, the image is divided into a grid of rectangular cells, and inside each cell, multiple anchors of predefined sizes and proportions are used to detect possible objects. The final bounding box is being regressed from each such anchor and classified as a particular class or as a background. Setting up the sizes and ratios correctly is an important step. It, for example, does not make sense to have anchors large in size if we know that there won't be large objects within any image. We, therefore, take special care to set up these sizes and ratios so that they cover the sizes of graphemes in our dataset. For a more complete overview of current methods in object detection see [17].

Also, we wanted to retain the resolution of images as high as possible, and so we cut the whole page into overlapping patches of size 256x256 px and process each patch separately. After the system processes all patches from an image, we merge all boundary boxes in a post-processing stage. We train the detection model on 300 synthetically generated pages, and

TABLE I: Comparison of average precision (AP) and Focal loss between the same model (RetinaNet) trained with and without syntetic images.

Training Data	AP	Focal Loss
With synthetic data	**0.3110**	**1.143**
Without synthetic data	0.03767	2.181

then we fine-tune it on three real and manually labeled pages. This model already produces quite precise bounding boxes, so we use it to pre-label six more pages in which we manually correct wrong predictions. We then enlarge the training set of synthetic and real images using these newly labeled pages. We repeat this process two times and thus acquire a model of satisfying accuracy. In table I, we compare average precision and Focal loss between the same model trained with and without synthetic images created from three annotated pages. As these metrics are not easily interpretable, we show a visual example in figure 6.

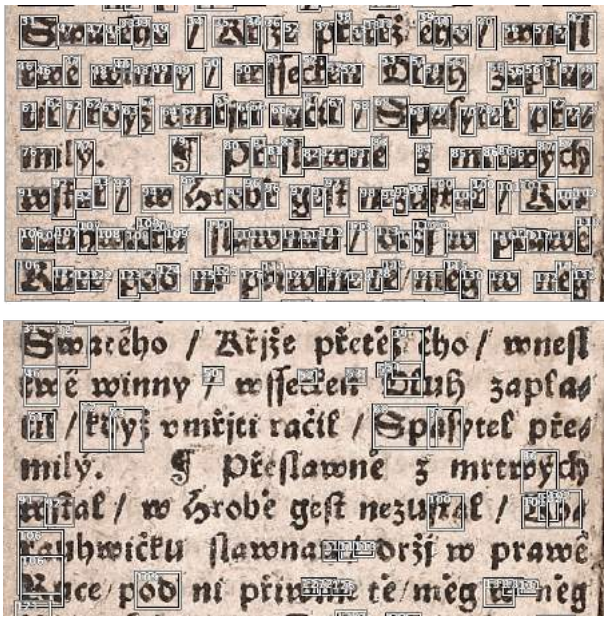


Fig. 6: A visual example of a quality of detection after the first round using only 3 annotated images. The top image shows results from a model trained on synthetic images and fine-tuned on the 3 annotated images. The bottom image shows results from a model trained only on the 3 annotated images (with standard augmentations, i.e. resize, lightness, etc.).

VIII. Glyph Classification

In the last section, we described the model for grapheme detection. We use this model to detect generic graphemes, which will be parsed into a sequence where the order of graphemes is the same as the order in which we would read

the text. These sequences are then classified with a sequence-based model, which we describe in this section. We use a sequence-based model because, in some instances, the identity of a grapheme may not be recognizable without a context. The sequence-based model can leverage statistical regularities within sequences of letters found within the use of a language.

First, we need to create a dataset for classification. Again, we want to avoid manual annotation as much as possible. We came up with two ways how to achieve it. Our classification model is convolutional bidirectional LSTM which takes a sequence of cut-out graphemes and produces a sequence of labels, one label for each grapheme. Therefore we need to obtain labeled training examples of such sequences. Our idea was to use transcriptions of texts which contain a language being used in our documents. Given these transcripts, we can create many different training sequences by sampling graphemes according to the letters in the text. This has an advantage that using one sentence, we can generate many different training examples by sampling different examples for the same letter each time. In order to do this, we need to have many examples of each letter. To avoid manual annotation of separate graphemes, we use the detection model to cut-out thousands of generic graphemes, which we then cluster and label only the clusters. This lowers the amount of work we need to do by order of magnitude.

In order to cluster the graphemes, we first train an autoencoder with ResNet-18 in the encoder to obtain a low-dimensional representation of every grapheme. Umap [18] visualization of this low-dimensional space can be seen in 7. We then cluster all graphemes using a simple K-means algorithm. We set K to 2*C where C is the number of classes because when K is close to C, the algorithm mixes too many examples of different classes together in the same cluster. We manually check all clusters and remove incorrectly assigned examples. Thus, we acquire thousands of labeled examples with very little manual effort. From these, we construct a dataset of labeled sequences.

Our classification model first extracts low-dimensional (512) representation of each grapheme by processing it with a backbone from ResNet-18, and the sequence of these low-dimensional representations then goes as an input into bidirectional LSTM (with 2 layers, both of which have the output dimension equal to 512). Finally, the sequence of representations in the hidden layer of the LSTM is then processed by a linear layer with a softmax to predict the class of the grapheme at every position of the sequence.

In table II, we show a comparison between our model, which takes the context of each grapheme into account and a model with the same backbone that classifies each grapheme separately.

IX. Related Work

The OCR problem has been studied for many years [1]. Especially for modern prints, there exist software solutions with nearly perfect accuracy. Most of these systems use a pipelined approach such that in one step, individual lines are segmented out, and subsequently, they are transcribed into sequences of letters by a neural network that processes the whole line as a sequence of vertical strips of pixels. The number of such strips in a line does not correspond to the number of letters (labels) in that line, and therefore special loss function called CTC loss [19] is used to account for the alignment of these strips and labels. The same loss function is frequently being used in speech recognition, where the same problem arises. Examples of systems based on this approach include Tesseract [7], Calamari [6] build on top of TensorFlow and OCRopus [5] build on top of PyTorch. We started our development with OCRopus but soon realized that we need a custom solution. We have also tried a system called OCR4ALL [8], which was developed specifically for historical OCR. It is targeted mostly on people with no coding skills, so the emphasis is being given mainly on user-friendliness and not much on customizability. In most of the use-cases, it is not needed to segment out individual graphemes, and so there is no need to innovate over these solutions. As mentioned in the motivation, our use case was quite specific, and therefore we needed to develop our own solution.

The usefulness of synthetic datasets for training machine learning models was realized by many [10]–[12]. Synthetic

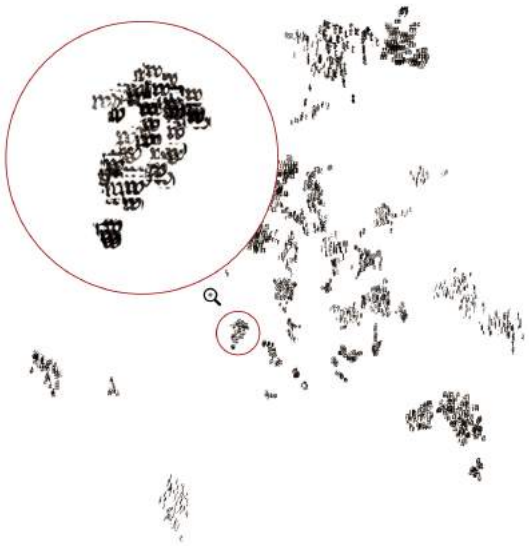


Fig. 7: Umap visualization of low-dimensional representation of cut-out graphemes obtained from a trained autoencoder.

TABLE II: Comparison between a model which classifies each grapheme separately and a model which takes the ohter graphemes in the same context into account. Both models share the same backbone (ResNet-18).

Method	Validation Accuracy
With bi-LSTM	**0.9337**
Without bi-LSTM	0.498

datasets are especially useful in robotics [13] and other domains where training on real data would be too expensive or infeasible. The main problem arising with the use of such datasets is that a machine learning model may overfit to specifics of synthetic examples and may not transfer well to real examples. As a possible solution to this problem, a technique called Domain Randomization became popular in recent years [20]–[22]. The idea behind Domain Randomization is that if we do not want to overfit to particular properties of data, such as for example color of objects when training object classifier, we should randomize that property as much as possible so that the model can not learn a statistical correlation between this property and some other variables of interest. We took inspiration in this idea when we were creating our synthetic dataset and randomized some properties of the generated images such as the size of graphemes, their sharpness, and other distortions.

X. Conclusion

In this contribution, we described creation of a custom OCR system explicitly designed to help linguistic analysis of printed texts from the early modern period. In contrast to mainstream OCR systems, we do not transcribe whole lines in an end-to-end fashion, but we first segment out individual graphemes, which are then classified using a sequence-based model. We also showed the usefulness of synthetically generated images and a bootstrapping process for annotation, which reduced the amount of manual work we needed to do by order of magnitude. In the future, we aim to design an intuitive user interface for our system which will be released together with the final version of the source code. We believe that our work, driven by the practical needs of linguists, is a valuable contribution to the interdisciplinary research between computer science and humanities.

Acknowledgment

The work was supported from ERDF/ESF "Centre for the development of Artificial Intelligence Methods for the Automotive Industry of the region" (No. CZ.02.1.01/0.0/0.0/17_049/0008414).

References

[1] D. Doermann, K. Tombre *et al.*, *Handbook of document image processing and recognition*. Springer, 2014.

[2] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[3] P. Voitem and J. Linka, "Udržet pravỳ okraj stránkové sazby (od literární historie k samostudiu)," *Česká literatura*, vol. 59, no. 2, pp. 242–260, 2011.

[4] R. Čech and J. Mačutek, "Orthography system of broadside ballads from 17. and 18. century. quantitative approach." Transformations of Czech 'kramářské písně' (Broadside Ballads) – media, traditions, contexts. Masaryk University in Brno, 2019.

[5] T. M. Breuel, "The ocropus open source ocr system," in *Document Recognition and Retrieval XV*, vol. 6815. International Society for Optics and Photonics, 2008, p. 68150F.

[6] C. Wick, C. Reul, and F. Puppe, "Calamari-a high-performance tensorflow-based deep learning package for optical character recognition," *arXiv preprint arXiv:1807.02004*, 2018.

[7] R. Smith, "An overview of the tesseract ocr engine," in *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2. IEEE, 2007, pp. 629–633.

[8] C. Reul, D. Christ, A. Hartelt, N. Balbach, M. Wehner, U. Springmann, C. Wick, C. Grundig, A. Büttner, and F. Puppe, "Ocr4all—an open-source tool providing a (semi-) automatic ocr workflow for historical printings," *Applied Sciences*, vol. 9, no. 22, p. 4853, 2019.

[9] S. Wu, S. Zhong, and Y. Liu, "Deep residual learning for image steganalysis," *Multimedia tools and applications*, vol. 77, no. 9, pp. 10 437–10 453, 2018.

[10] J. Hula, I. Perfilieva, and A. A. M. Muzaheed, "Towards visual training set generation framework," in *International Work-Conference on Artificial Neural Networks*. Springer, 2017, pp. 747–758.

[11] S. Sankaranarayanan, Y. Balaji, A. Jain, S. Nam Lim, and R. Chellappa, "Learning from synthetic data: Addressing domain shift for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 3752–3761.

[12] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*. Springer, 2016, pp. 102–118.

[13] P. Martinez-Gonzalez, S. Oprea, A. Garcia-Garcia, A. Jover-Alvarez, S. Orts-Escolano, and J. Garcia-Rodriguez, "Unrealrox: an extremely photorealistic virtual reality environment for robotics simulations and synthetic data generation," *Virtual Reality*, pp. 1–18, 2019.

[14] T. K. Ho and H. S. Baird, "Evaluation of ocr accuracy using synthetic data," in *Proceedings of the 4th Annual Symposium on Document Analysis and Information Retrieval*. Citeseer, 1995.

[15] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

[17] L. Jiao, F. Zhang, F. Liu, S. Yang, L. Li, Z. Feng, and R. Qu, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, pp. 128 837–128 868, 2019.

[18] L. McInnes, J. Healy, and J. Melville, "Umap: Uniform manifold approximation and projection for dimension reduction," *arXiv preprint arXiv:1802.03426*, 2018.

[19] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 369–376.

[20] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2017, pp. 23–30.

[21] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[22] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4243–4250.

B Graph neural networks

Graph Neural Networks for Dynamic Scheduling of SMT Solvers

1st Jan Hůla
University of Ostrava
Ostrava, Czechia

2nd David Mojžíšek
University of Ostrava
Ostrava, Czechia

3rd Mikoláš Janota
Czech Technical University in Prague
Prague, Czechia

***Abstract*—This paper develops an approach to the dynamic scheduling of solvers in the domain of Satisfiability Modulo Theories (SMT) using a Graph Neural Network (GNN). In contrast to related methods, GNNs do not require manual feature design as they enable discovering relevant features in the raw data. We train them to predict the effectivity of individual solvers on a given problem. Rather than choosing only one solver with the best prediction, we dynamically schedule the solvers by greedily choosing the solver with the smallest expected solving time according to the predictions until a timeout is reached. We compare our approach to several baselines. In the selected benchmarks, we show a substantial improvement over these baselines in terms of the number of solved problems and overall solving time.**

I. Introduction

In recent years, there has been a lot of interest in the use of machine learning (ML) for problems dealing with combinatorial search. Such problems are known to be NP-complete or harder, but in practice, we are often not interested in the worst-case scenario but in the average behaviour of a concrete algorithm on some specific distribution of problem instances. At the same time, different heuristics may show very different per-instance behaviour in terms of runtime or even in the ability to find a successful solution. This per-instance behaviour is hard to understand for a human, but we may try to predict it using ML methods.

In this contribution, we focus on solvers in the domain of Satisfiability Modulo Theories (SMT) which try to decide the satisfiability of logical formulas in various theories. As mentioned above, individual solvers or configurations of the same solver may produce different per-instance behaviour. As a result, there has been an interest to design portfolios of solvers together with predictors which select a solver or a subset of solvers with a schedule on a per-instance basis. These predictors are conditioned on various features of the SMT formula.

In the past, most of the approaches have been based on manually designed features which reflected the intuition of a domain expert. These features can vary from very complex, such as the ones given by linear programming relaxation of the formula, or various runtime statistics of a particular solver [1], to very simple, such as counts of occurrence of various symbols within the formula [2].

Instead of designing better features, we follow the recent trend of learning features from raw data together with the predictions in an end-to-end fashion. In our case, we use a Graph Neural Network (GNN) [3] that takes as input an SMT formula represented as a graph with labeled nodes and predicts the efficiency of available solvers for the given formula. Using the predictions from the GNN, we construct a dynamic schedule by greedily choosing the solvers with the smallest expected solving time until a timeout is reached. We compare our approach to several baselines and show an substantial improvement in terms of the PAR-2 score, which is commonly used as a metric to compare SMT solvers in competitions. This score is the sum of runtimes overall problems within the dataset. For problems that are notsolved, the runtime is set to twice the timeout. We also demonstrate a significant drawback of choosing only the solver with the best prediction, which was done in a previous work [2], by showing that it is outperformed by a random schedule which splits the time across the individual solvers and runs them in an random order.

To summarize, the paper has the following main contributions.

- It applies GNN to predict the performance of SMT solvers on a given instance. To the best of our knowledge, this is the first application of GNN in the context of SMT.
- The proposed approach dynamically schedules the solvers rather than just picking the best one, which further improves the robustness of the approach.

II. Satisfiability Modulo Theories

Solvers for *Satisfiability Modulo Theories (SMT)* are the driving force behind software verification, software testing, or software synthesis, among others [4]–[7]. These applications often require repeated queries to an SMT solver. This means that quick response times of the solver are paramount.

An SMT solver receives as input a formula and responds if it is satisfiable or not. Since the problem is generally undecidable, solvers often timeout or give up.

The language and the semantics of the given formula depends on the theories being used. For instance, the formula $(3 < x) \wedge (x < 4)$ is satisfiable in the theory of real arithmetic but unsatisfiable in the theory of integer arithmetic.

The language and the possible theories are standardized in the *SMT-LIB standard* [8]. A repository of benchmarks is maintained in the *SMT-LIB* [9]. A combination of theories is called a *logic*. For instance, the logic UFNIA supports uninterpreted functions (theory UF) and nonlinear integer

```
(set-logic UFNIA)
(declare-fun f (Int) Int)
(declare-const c Int)
(assert (= (f c) 0))
(assert (forall ((x Int)) (>= (f x) (* c (f x)))))
(check-sat)
```

Fig. 1. Example SMT input using the UFNIA logic

arithmetic (theory NIA). Hence, it is mandatory for each problem file to specify the intended logic in the header.

The SMT input format uses LISP-like notation. Figure 1 shows an example with one uninterpreted (unknown) function f from integers to integers and one uninterpreted integer constant c. The formula consists of two *assertions*, i.e. subformulas, that must hold. The first assertion requires that f evaluates to 0 on c and the second assertion that $f(x) \geq cf(x)$ for any integer x. The problem is classified as non-linear because there is a multiplication between two unknowns.

The command check-sat tells the solver to check satisfiability of the assertions made so far. This example problem is trivially satisfiable, for instance by f being constantly 0. Problems can contain multiple check-sat statements but in this article we focus only on single-query problems.

Various SMT solvers support various theories and their combinations. Furthermore, not all solvers support all the features of the SMT language. This alone makes the choice of the right solver for a given instance a nontrivial task.

A dedicated format for formulas in purely first-order logic (FOL) also exists, called TPTP (thousand problems for theorem provers) [10]. In the parlance of SMT, first-order logic is the logic consisting only of the theory of uninterpreted functions (UF). Even though FOL and SMT solvers can in principle be used in both types of problems, the two paradigms represent two different scientific communities. Nevertheless, the FOL solver Vampire [11] routinely participates in the SMT competitions and we include it in our evaluation together with the TPTP benchmark which we convert to SMT-LIB format.

III. Problem Statement

Let $S = \{s_1, \ldots, s_l\}$ be a set of l SMT solvers which we have at our disposal. Our goal is to produce an effective algorithm which on per-instance basis creates a dynamic schedule from the set S. By dynamic, we mean that we can run the individual solvers while creating the schedule.

More formally, we want to obtain a function f_θ (parametrized by learnable parameters θ) which takes a representation of an SMT formula q and the set S as an input and outputs an ordered tuple $f_\theta(q, S) = ((i_1, t_1), \ldots, (i_n, t_n))$ where i_j's are indices of selected solvers and t_j's are times assigned to these solvers, such that $\sum_j t_j = t_{max}$, where t_{max} is the maximum time we are willing to spend on a problem. In other words, for a given formula q, $f_\theta(q, S)$ represents a schedule of chosen solvers for this formula. To avoid the trivial case, in which the function f_θ runs all available solvers and then outputs the one with the shortest runtime, we add the constraint that f can run a solver only if it was added to the schedule and only for the scheduled duration. Effectively, the function f may create the schedule on the fly.

Given a formula q and its schedule $f_\theta(q, S)$, we measure how long it takes to solve the formula using this schedule. We denote this measurement by $M(q, f_\theta(q, S))$ and set it to a constant number t_{pen} (with $t_{pen} > t_{max}$) if the formula was not solved under the time limit t_{max}.

We assume that the problems/formulas we want to solve come from an unknown distribution P and that we have a finite set of independent and identically distributed samples $Q = \{q_1, \ldots, q_m\}$ where $q_i \sim P$. Our task may be posed as the following optimization problem:

$$\theta^* = \arg\min_\theta \int M(q, f_\theta(q, S))\, dP(q)$$

Because the distribution P is unknown, we can only try to minimize an approximation to the objective function given by the m samples in Q:

$$\hat{\theta}^* = \arg\min_\theta \frac{1}{m} \sum_{q_i \in Q} M(q_i, f_\theta(q_i, S)). \tag{1}$$

The empirical objective function $\frac{1}{m}\sum_{q_i \in Q} M(q_i, f_\theta(q_i, S))$ can be also used to compare different scheduling functions f_θ. The learnable parameters of this function cannot be directly optimized with respect to the objective function by gradient-based methods, because it involves discrete choices. We will therefore construct the function f_θ in two stages, by first learning to approximate the runtimes of individual solvers conditioned on the problem and then constructing the final schedule by postprocessing the predicted runtimes.

IV. Graph Neural Networks

In this section, we describe Graph Neural Networks (GNNs) used in our experiments. GNNs are neural networks which process inputs structured as a graph. This makes them different from other types of neural networks such as Multilayer Perceptrons, Recurrent Neural Networks, Convolutional Neural Networks, or Transformers, which do not assume any special structure of the input. For this reason, GNNs became popular for processing all kinds of formal structures such as logical expressions, which are naturally represented as trees or directed acyclic graphs.

Most often, additional meta-information for nodes within an input graph is available. For a specific node, this information is encoded as a feature vector of a fixed size. In the case of logical expressions, these feature vectors encode the symbols used within the expression. The encoded symbols are usually abstractions over the original symbols so that the final alphabet is not too large. In our case of SMT formulas, we abstract away specific numeric values of constants and function names. The feature vector often encodes the symbol in one-hot fashion, which means that the length of the vector is equal to the length of the alphabet and it contains all zeros, except having value 1 at the position that corresponds to the symbol being encoded.

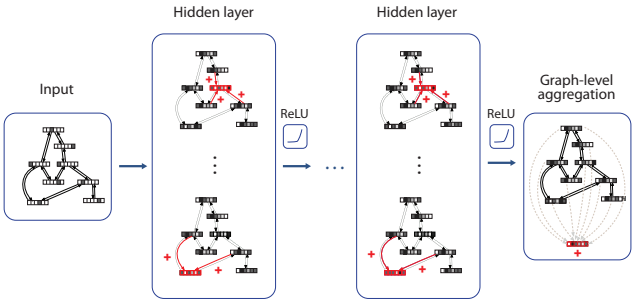


Fig. 2. Evaluation of the GNN on an input graph to obtain a single feature vector. The vectors highlighted in red are the once being aggregated.

Each layer of a GNN updates the feature vectors of all nodes by transforming and aggregating the feature vectors of its neighbour nodes. GNN architectures differ in how they achieve this transformation and aggregation.

Let $G = (V, E)$ be a graph with a feature vector x_v assigned to each node ($x_v \in \mathbb{R}^m$ where m is the length of the alphabet of possible symbols). Furthermore, let $N(v)$ denote the neighbourhood of a node v, i.e. the set of all nodes adjacent to v. In each propagation step $k = 1, 2, 3 \ldots$, a new feature vector $h_v^{(k)}$ (s.t. $h_v^{(0)} = x_v$) is computed in the following way [12]:

$$a_v^{(k)} = \mathsf{AGGREGATE}^{(k)} \left(\left\{ h_u^{(k-1)} \mid u \in N(v) \right\} \right)$$

$$h_v^{(k)} = \mathsf{COMBINE}^{(k)} \left(h_v^{(k)-1}, a_v^{(k)} \right)$$

Often these two steps are integrated and aggregation is done over $N(v) \cup \{v\}$. For example, a simple node update rule used in the basic Graph Convolutional Network model (GCN) [13] has the following form:

$$h_u^{(k)} = \sigma \left(W^{(k)} \sum_{v \in N(u) \cup \{u\}} \frac{h_v^{(k-1)}}{\sqrt{|N(u)||N(v)|}} \right)$$

The matrix $W^{(k)} \in \mathbb{R}^{dim(h_u^k) \times dim(h_u^{k-1})}$ contains trainable parameters, the square root in the denominator scales the vectors according to their degree, and σ is the application of a non-linear activation function (such as ReLU). We stress that $W^{(k)}$ is shared by all nodes in layer k but may differ across layers, allowing successive change of feature vector sizes.

GCN was one of the first instances of a GNN and over the years, lots of modifications have been proposed. They differ mostly by using different aggregation functions. In our initial experiments, we tried different architectures, but in the final implementation we use the basic GCN described above because the other architectures produced no or only negligible improvements.

After updating the feature vector of each node k times (with k layers of the GNN), the resulting feature vectors are influenced by all nodes which can be reached by k hops from the given node. Therefore, the GNN can in principle detect structural features present in small subgraphs.

GNNs can either be used to produce *node-level predictions* or *graph-level predictions*. In the case of graph-level predictions, we have to aggregate all feature vectors from every node within a graph to one output feature vector, which is then used to make the final prediction. This is done in the same way as with the aggregation of feature vectors from neighbour nodes.

The process for obtaining one feature vector from the input graph is depicted in Figure 2. For the final prediction, we use a simple Multi-layer perceptron and train the whole network using backpropagation. In our case, we train the network to predict the intervals in which the given problem will be solved by individual solvers and use these predictions as described in the next section.

The advantage of using a GNN is that the trained transformations are applied locally to each node and the final aggregation operator does not require a specific number of inputs. It allows the graphs to have different number of nodes and structure. Therefore, the trained GNN is applicable to arbitrary graphs.

V. Solver Scheduling

As mentioned in Section III, we aim to obtain a problem-dependent scheduler that minimizes Equation 1. The objective function in Equation 1 cannot be optimized directly by gradient-based methods, and therefore we solve the problem in two stages. First, we train our GNN to predict the runtimes of individual solvers and then create a dynamic schedule based on these predictions. These predictions are probabilistic in the sense that we predict a distribution over intervals in which the given problem could be solved.

Concretely, we split the whole available runtime into multiple intervals $I = (i_1, \ldots, i_n)$ where we assign an endpoint to each interval and train the GNN to score these intervals independently for each solver. The last interval corresponds to a timeout and its $t(i)$ is equal to $2 \times$ *timeout*.

For each solver, the GNN predicts in which interval the problem will be solved. More precisely, it predicts a score for each (solver, interval) pair and these scores can then be normalized to probabilities with the softmax function across all intervals in each solver separately.

Once we have the probabilities across possible intervals for a given solver, we can compute an upper bound of the expected runtime by taking the expected value of the endpoints in each interval $\mathbf{E}_{i \in I}[t(i) \times p_s(i)]$ where $p_s(i)$ is the probability that the solver s will solve the given formula in interval i.

For example, the following table shows two solvers with interval endpoints 300 and 2400 with probabilities predicted by the network for a single problem. Note that the last interval always represents an unsolved scenario, giving the expected runtime as $300p_1 + 2400p_2 + 2 \times 2400(1 - p_1 - p_2)$.

solver / time (s)	300	2400	TO	Expected runtime
s_1	0.3	0.5	0.2	2250
s_2	0.4	0.1	0.5	2760

Algorithm 1: Dynamic scheduling with predictions

Data: D: array of lists. List $D[i] = [(t_1^i, d_1^i), \ldots, (t_n^i, d_n^i)]$ corresponds to solver i with t_j^i being the length of the j-th interval and d_j^i its score.
Result: Total runtime spent on the formula

```
runtime ← 0
while runtime < timeout do
    expectedTimes ← getExpectedTimes(D)
    currBest ← arg min(expectedTimes)
    nextIntLen, score ← getFirst(D[currBest])
    solved, time ← runSolver(currBest, nextIntLen)
    runtime ← runtime + time
    if solved then
        return runtime
    else
        D[currBest] ← removeFirst(D[currBest])
return 2 × timeout // unsolved instance penalty
```

The schedule is constructed greedily by relying on the GNN's predictions; see Algorithm 1 and Figure 3 for a schematic overview. The process starts by scheduling the solver with the smallest expected value and running it for the duration of the first interval. If the problem is unsolved during this interval, we recompute the expected value for the chosen solver by ignoring the first interval.

Concretely, we apply the softmax function only over the remaining intervals and then compute the expected value over these intervals. If we denote the tuple of intervals for which the given solver has already run by I_s then the new expected value can be written as $\mathbf{E}_{i \in I \setminus I_s}[t(i) \times p_s(i)]$ where $p_s(i)$ is recomputed by ignoring the intervals in I_s. We also need to subtract the time for which the solver has already run to obtain the upper bound of the expected *remaining* runtime $\mathbf{E}_{i \in I \setminus I_s}[t(i) \times p_s(i)] - \max_{i \in I_s} t(i)$.

If this new expected value is smaller than the second smallest expected value computed previously, then we run the same solver for the next interval, otherwise we switch to the solver with the second smallest expected value.

We continue in this fashion, always deciding which solver to run after every interval, until the timeout is reached. For this reason, the schedule is dynamic as it is computed "on the fly". It is also a preemptive schedule, which means that individual solvers can be stopped and resumed again later.

A. Splitting the available time into intervals for classification

We want to split the available time into intervals so that the network can be trained to classify in which interval the problem is solved. In our first set of experiments, we trained the network to regress the precise runtimes of each solver, but with such predictions it was not possible to construct the dynamic schedules. With probabilistic predictions over the intervals, we can compute the expected values of runtimes and recompute them after every interval for which the problem was not solved.

The number of intervals and their lengths is an important hyperparameter which could lead to poor results if set naively. In empirical datasets, it can be observed that the probability of solving a problem decreases exponentially with time [14]. Therefore "most" problems are solved in a "short" time. Hence, if the time is split into uniform intervals, the first few intervals would contain "almost all" ground-truth labels.

On the other hand, if we split the available time in such a way that every interval would contain the same number of ground-truth labels, the first few intervals would be "too short" and the last interval would be "too long" (for example, in the UFNIA benchmark, with timeout 2400 and the number of intervals set to 15, the last interval would cover more than half of the available time).

We would like to have something in the middle of these two extremes. That is, to have intervals whose length increases but not too rapidly. In our experiments, we found out that the power functions work well for this purpose. Hence, we set the length of an interval n to n^p for some $p \in \mathbb{R}$. If we aim to have N intervals in total, p is chosen so that $N^p = t_{max}$.

For example, for $N = 14$ and $t_{max} = 2400$, yields $p = \log_{14}(2400) \simeq 2.9$ and interval endpoints: $(1, 8, 26, 60, 115, 197, 311, 461, 652, 890, 1178, 1523, 1929, 2400)$. If the problem is unsolved and thus does not fall into any interval, we classify it as an additional class of unsolved problems.

The classes/intervals obtained this way are unbalanced, which we tackle by using a weighted cross-entropy loss, with the weight of each class being inversely proportional to the number of ground-truth labels contained in that class.

VI. Experiments

A. Datasets

We test our approach and the other baselines on 4 representative benchmarks. As our goal is to show that we can improve upon the best individual solver and the other baselines on a given benchmark, we choose benchmarks with a noticeable gap between the best individual solver and the virtual best solver. The virtual best solver represents an upper bound of what could be achieved. It simply selects the best solver for each problem independently. When selecting the benchmarks, we have also taken into consideration the number of problems on a benchmark because our GNN contains a large number of learnable parameters and therefore could overfit on small datasets. A summary of the chosen benchmarks follows.

QF-NRA Is the logic of non-linear real arithmetic without quantifiers (QF stands for quantifier free). It involves arithmetic operations on real numbers. Since it is non-linear multiplications between two unknowns are allowed.

UFNIA The logic combines two theories, uninterpreted functions (UN) and non-linear integer arithmetic (NIA). The formulas may contain quantifiers. The logic is highly useful in software verification but at the same time is undecidable.

UFNIA-CONF It is customary that in competitions solvers run multiple strategies in sequence to tackle any given problem. This is especially true for formulas containing quantifiers where a number of different approaches exist, which exhibit strong degree of orthogonality [15]. To further diversify our

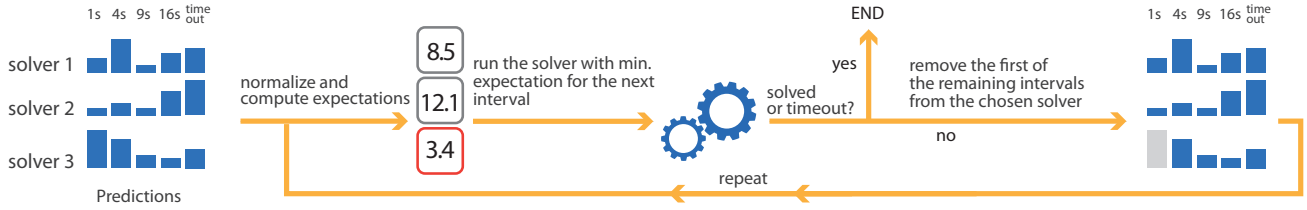


Fig. 3. A simplified scheme of the algorithm used to dynamically schedule the solvers. The expected time for each solver is calculated from the interval probabilities. After the solver with the smallest expected time runs for the length of first interval we remove this interval and recalculate the probabilities together with new expected times. The process is repeated until solved or timeout is reached.

experimental results, we have collected the different strategies that the solver CVC5 uses to solve UFNIA formulas in the competition.[1] This gives 23 different option settings for the solver, which we then use as our portfolio.

TPTP A set of problems in purely first-order logic coming from the TPTP library [10]. We used the problems available at TPTP website[3] and converted them to SMT-LIB format using Vampire. We filtered a subset of problems for which results of 25 selected solvers were available.

Benchmark name	# of problems	# of solvers	Timeout (s)
QF_NRA	2654	9	2400
UFNIA	5659	7	2400
UFNIA-CONF	5659	23	60
TPTP	4741	25	300

TABLE I
NUMBER OF PROBLEMS AND SOLVERS PER BENCHMARK.

To obtain the ground truth runtimes for QF-NRA, UFNIA, and TPTP, we reused the publicly available data[2] [3]. For UFNIA-CONF, we obtain the results by running the solver with a timeout of 60 seconds.

B. Data processing

For the processing of individual SMT formulas, we used the PySMT library [16]. Concretely, we used the built-in SMT-LIB parser and our custom printer which outputs the formula as a directed acyclic graph (DAG) with labeled nodes. The labels come from an alphabet which could be benchmark specific (depending on the kind of logical symbols which could appear within the benchmark). Therefore, every benchmark could have input feature vectors of different lengths. The alphabet of possible symbols is handled by PySMT, which abstracts away details such as function names or specific numeric values.

Because some SMT formulas may produce graphs with millions of nodes, we need to subsample these large graphs to fit them into GPU memory. As one SMT formula may consist of multiple conjunctive assertions, we decided to subsample the graphs according to these assertions. We first sort them by their size[4] in an ascending order and then build the final graph by adding assertions, one by one, while the size of the resulting graph does not exceed a certain threshold. In our experiments, we set the maximum graph size to 12 thousand nodes. For every formula, we also add one root node which connects all assertions together.

To create the final form of the graph which goes as an input to the GNN, we augment the DAG we obtain from PySMT with edges going backwards. That is, for every edge (a, b) we add edge (b, a) to the graph.

C. Training the GNN

As outlined in Section V, we train our GNN to predict the intervals in which the problem will be solved by individual solvers. This prediction is realized as a classification of an interval in which the problem will be solved, one for each solver. Therefore, we pair every input graph with a tuple of l different class labels corresponding to ground truth intervals of the l solvers. To be more explicit, if there are two solvers where the first solver solves the problem within the 3^{rd} interval and the second solver within the 4^{th} interval, then the labels will be (3,4). If a given solver does not manage to solve the problem under the timeout, we classify it to an additional class for unsolved problems. The input graph together with the tuple of labels comprises one training example.

We collect all examples on a given benchmark and split them into training and testing samples as described in the next section. When searching for hyperparameters, we set aside 15% of the training set and use it as a validation set.

As a loss function for training, we use a sum of weighted cross-entropy loss functions, one for each solver. The weights are inversely proportional to the number of ground truth labels in a given class. That is, we count how many times a given solver solved a problem within a given interval and set the weight for this interval to 1 over this value.

As mentioned previously, we used GCN for our experiments. We set the number of graph convolutional layers to $k = 6$ and the hidden representation vector size to 150 for all of them with exception of the final GCN layer size set to 700. Not counting the first layer, which has a different number of parameters, depending on size of the input feature vectors, the whole model has ∼725K parameters. To aggregate feature

[1] https://github.com/cvc5/cvc5/blob/master/contrib/competitions/smt-comp/run-script-smtcomp-current
[2] https://github.com/SMT-COMP/smt-comp
[3] http://www.tptp.org/cgi-bin/SeeTPTP
[4] Number of nodes in the resulting graph

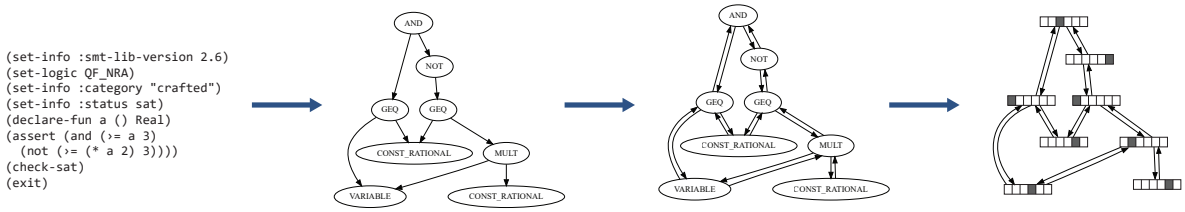


Fig. 4. The steps conducted during the creation of the input graph from a given SMT formula. Parsing yields a directed graph, which is then augmented by edges going in reverse direction. Finally, symbols on nodes are encoded to one-hot vectors.

vectors from all nodes into one feature vector, we use the max aggregation operator. It takes element-wise maximum across all feature vectors. This feature vector goes as an input to a 2 layer MLP with the size of the hidden layer equal to 350. The output size is equal to the number of solvers on a given benchmark times the number of classes/intervals (we used 15 classes per solver, including the class for unsolved problems).

To optimize the weights of the network, we use the ADAM optimizer [17] with the learning rate set to 0.001. Each update of the weights is made with respect to a batch of multiple training examples. It is believed that larger batch sizes (number of examples within a batch) lead to a higher accuracy, so it is desirable to set the batch size as high as possible [18].

To create batches for training, we use the batching functionality in PyTorch Geometric [19] which internally concatenates multiple graphs to one big graph which is processed at once. During training, the examples in each batch are sampled randomly, which can cause problems when multiple large graphs meet in the same batch. As each layer within a network needs to store the computed values for each node, this can quickly lead to memory issues. We trained the GNNs on a workstation with 24GB of GPU RAM which allowed us to set the batch size equal to 16. Each GNN was trained for 100 epochs before evaluated on a test set. Depending on the size of the training set and the architecture used, this took approximately 30 to 90 minutes. Inference on individual examples takes approximately 0.02 sec for graphs with more than 15000 edges and less than 0.01 sec for smaller graphs.

D. Evaluation protocol

For the evaluation of individual models, we run the same procedure as Scott et al. [2]. That is, we shuffle the dataset and split it into training and testing subsets five times ($K = 5$) so that $80\,\%$ of the data is used for training and $20\,\%$ is used for testing. This is the same procedure as used in K-fold cross-validation. Testing sets from all 5 folds are disjoint and cover the whole dataset. For each fold, the model is always initialized with random weights, trained for 100 epochs and then evaluated on the respective testing set. This ensures that we obtain predictions for every problem within the dataset.

To compare individual solvers and schedules, we compute their PAR-2 score. This score is the sum of runtimes over all problems within the dataset. For problems that are not solved, the runtime is set to twice the timeout. To compute the runtime for various schedules, we iterate over individual solvers within the schedule and check if the problem would be solved by the given solver within the interval dedicated to it. If the problem is solved by the n-th solver within the schedule by using portion t of its current interval and the length of each previous interval i would be s_i, we would set the runtime to $t + \sum_{i=1}^{n-1} s_i$. If the problem is not solved by any solver within the interval dedicated to it, we set its runtime to twice the timeout. This procedure is denoted by the function M in Equation 1. The value t_{max} is then equal to the timeout and t_{pen} is equal to $2 \times t_{max}$.

E. Description of the tested approaches

The results for the following baselines and our approach are presented in the Table II.

Best solver. Is a single solver with the best PAR2 score. We present our results as an relative improvement over the best solver.

Virtual best solver (VBS). This is a hypothetical algorithm that selects the best solver for each problem separately. It represents an upper bound for a possible improvement.

BOW-single and GNN-single. These two baselines choose the best solver according to the predictions of the model and run it until the timeout. They differ only in the ML model. Both models are trained to regress the ground truth runtimes and we select the solver with the shortest predicted runtime.

BOW-single uses a model similar to the one used in MachSMT. It uses bag-of-words features, which contain the counts of each symbol within the formula. Depending on the benchmark, the size of the feature vector is in the range 12–20. For the regression, we use LightGBM [20] and a grid search to find suitable hyperparameters.

GNN-single uses the same model described in the Section VI-C with the only difference that it predicts runtimes instead of the probabilities for the intervals.

Random schedule. Random schedule divides the whole available time to all available solvers and runs them in a random order. We found out that for 2 of the tested benchmark sets, this simple schedule outperforms the approach which selects only the solver with the best prediction. This shows serious drawbacks of the results presented in MachSMT and also the importance of scheduling more solvers. Random

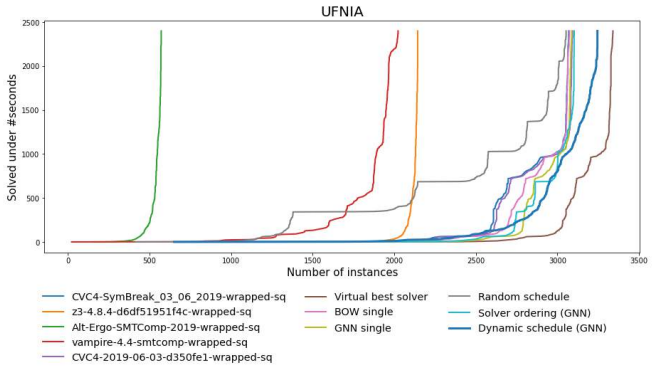


Fig. 5. Cactus plot of the results on the UFNIA benchmark set: x-axis represents the number of instances and y-axis the time up to which this number was solved by respective schedule/solver.

Benchmark		QF-NRA	UFNIA	UFNIA-CONF	TPTP
Best Solver	solver	Z3	CVC4	-	Vampire
	solved	2120	3093	2494	3204
VBS	solved	2516	3339	3118	3564
BOW single	PAR-2 impr.	117.10%	-0.64%	0.32%	13.56%
	solved	2343	3074	2586	3315
GNN single	PAR-2 impr.	231.19%	1.8%	56.73%	18.23%
	solved	2403	3085	2644	3320
Random schedule	PAR-2 impr.	269.33%	-21.62%	69.59%	-15.05%
	solved	2494	3053	2812	3266
Solver ordering	PAR-2 impr.	913.05%	8.30%	88.59%	6.84%
	solved	2494	3053	2812	3266
Dynamic schedule	PAR-2 impr.	**1162.87%**	**76.91%**	**113.54%**	**51.92%**
	solved	**2498**	**3245**	**2891**	**3388**

TABLE II
COMPARISON OF EVALUATED APPROACHES. THE PAR-2 IMPROVEMENT IS RELATIVE TO THE BEST SOLVER.

schedule is effective because many problems are solved in a short time by at least one solver.

Solver ordering (GNN). An improvement over choosing only the predicted best solver is to choose k best solvers and split the available time across them. In the extreme case, we can choose all available solvers, split the available time across them uniformly, and order them by the predicted time starting with the solver with the shortest predicted time. For the predictions, we use the same model as in GNN-single.

Dynamic schedule (GNN). This is the approach described in this contribution.

F. Results

The results of our experiments are visible in the Table II. The PAR-2 improvement is relative to the best solver and it takes into consideration only the problems which were solved by at least one solver. Our dynamic schedule clearly outperforms the other approaches. The most noticeable improvement (over the other baselines) can be seen in the column for UFNIA, for which we also include a cactus plot in the Figure 5. We can also notice that for QF-NRA and UFNIA-CONF, the random schedule outperforms GNN-single and BOW-single, which demonstrates the importance of a schedule.

VII. RELATED WORK

Algorithm selection and scheduling [21] is recognized as an important topic as a consequence of the need for reliable and fast problem handling in practical applications. Using machine learning methods for selection of a solver from a portfolio was popularized by Leyton-Brown et al. [22]. In the literature, models which predict the runtime of individual algorithms are usually called Empirical Hardness Models [23].

In terms of the goal and used benchmarks, our work is most similar to MachSMT [2]. The main difference is that they try to select only one solver from the whole portfolio and use bag-of-words as the representation of a formula. We found out that in many cases, the best solver according to the prediction of the model does not solve the formula at all. In our initial experiments, selecting multiple solvers and delegating part of the total time budget to each of them produced noticeably better results than selecting only one solver. We also train the feature extraction together with the final regressor end-to-end, and therefore our approach may discover more complex and useful features.

There are many other approaches that demonstrate the possibility of using Graph Neural Networks to extract the features of various formulas and learn the final prediction in an end-to-end fashion. In the domain of SAT solvers, Selsam et al. [24] train a GNN to predict the satisfiability of a formula. The trained network can later be used to find a solution for new formulas. The same problem is studied by Cameron et al. [25] who used different architectural choices for the GNN. In the following work, Selsam et al. uses a GNN to guide a SAT solver [26]. The GNN was trained to predict the unsatisfiable core of a given formula and these predictions were then used for variable selection inside the SAT solver. Wang et al. [27] use a GNN to embed and predict the relevance of logical formulas in the task of premise selection.

For tasks not amenable to supervised learning formulation, GNNs are often trained by reinforcement learning or evolutionary algorithms. Vaezipoor et al. [28] use GNNs for propositional model counting. The GNN is used as a branching heuristic and trained by an evolutionary algorithm. GNNs have also been used as branching heuristics inside solvers for Integer Linear Programming [29].

Our work is also related to various approaches using ML for solver scheduling, especially in the domain of SMT. Balunovic et al. [30] use imitation learning techniques to schedule strategies within the Z3 solver. Simalarly, Ramírez et al. [31] use evolutionary algorithm to generates strategies for the Z3 solver.

Pimpalkhare et al. [14] avoid the need for offline pretraining and instead learn to select individual solvers with their runtimes in an online fashion. In the domain of Constraint Programming, O'Mahony et al. [32] allocate time between solvers by solving an NP-hard problem.

For an overview of various use cases of ML methods for combinatorial problems and algorithm selection, see the following survey papers: [33]–[35]. For a more specific overview focused on GNNs see [36].

VIII. Discussion

We mention several drawbacks of our approach. In comparison to MachSMT, our models contain a much larger number of parameters. To train them successfully, we need many more training examples. Therefore, we limited all our experiments to benchmarks that contain thousands of problems.

Moreover, in our experiments we have not taken into consideration the time used to make the prediction. Although the processing of the input by a GNN is fast (see Section VI-C), to construct the input graph may take a non-negligible time if the formula is large. In this work, we have not focused on this issue and prepared the whole dataset before the training and testing phase. For practical applicability, we would need to replace the PySMT parser, which is written in Python, and ideally apply a more aggressive subsampling strategy. We leave that as a subject of future work.

Lastly, we mention that GNNs may be an overkill for many scenarios, as the improvement over a simple classifier using a bag-of-words representation may not be worth of longer training times. Interestingly, we also found that in the case of QF-NRA, using only one feature (the number of real constants within a formula) produces comparable results as the bag-of-words representation. Nonetheless, GNNs provide more flexibility and their usefulness should scale with the size of the training set.

IX. Conclusion

This paper presents an application of GNNs for SMT solver scheduling. We showed that GNNs can be successfully used as Empirical Hardness Models. Their main advantage, in comparison to other ML methods, is that they can be used without manual feature engineering. We also showed the benefits of a dynamically constructed schedule. In our experiments, we compared our approach to several baselines and demonstrated significant improvements in terms of the number of solved problems and overall solving time. In future work, we plan to focus on meta-learning of GNNs to quickly adapt to new problem distributions and on the augmentation and sub-sampling strategies for individual formulas.

References

[1] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Satzilla: Portfolio-based algorithm selection for SAT," *J. Artif. Intell. Res.*, vol. 32, pp. 565–606, 2008. [Online]. Available: https://doi.org/10.1613/jair.2490

[2] J. Scott, A. Niemetz, M. Preiner, S. Nejati, and V. Ganesh, "MachSMT: a machine learning-based algorithm selector for SMT solvers," in *TACAS*, 2020.

[3] W. L. Hamilton, "Graph representation learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 14, pp. 1–159, 2020.

[4] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli, "Satisfiability modulo theories," in *Handbook of Satisfiability*, A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds. IOS Press, 2009, pp. 825–885.

[5] A. Reynolds, V. Kuncak, C. Tinelli, C. W. Barrett, and M. Deters, "Refutation-based synthesis in SMT," *Formal Methods Syst. Des.*, vol. 55, no. 2, 2019.

[6] L. de Moura and N. Bjørner, "Applications and challenges in satisfiability modulo theories," in *Workshop on Invariant Generation (WING)*, vol. 1. EasyChair, 2012, pp. 1–11.

[7] P. Godefroid, M. Y. Levin, and D. A. Molnar, "SAGE: whitebox fuzzing for security testing," *Commun. ACM*, vol. 55, no. 3, pp. 40–44, 2012.

[8] C. Barrett, A. Stump, and C. Tinelli, "The SMT-LIB Standard: Version 2.0," in *SMT Workshop*, 2010.

[9] C. Barrett, P. Fontaine, and C. Tinelli, "The Satisfiability Modulo Theories Library (SMT-LIB)," `www.SMT-LIB.org`, 2016.

[10] G. Sutcliffe, "The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0," *Journal of Automated Reasoning*, vol. 59, no. 4, pp. 483–502, 2017.

[11] L. Kovács and A. Voronkov, "First-order theorem proving and Vampire," in *CAV*, 2013, pp. 1–35.

[12] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[13] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[14] N. Pimpalkhare, "Dynamic algorithm selection for SMT," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 1376–1378.

[15] A. Reynolds, H. Barbosa, and P. Fontaine, "Revisiting enumerative instantiation," in *Tools and Algorithms for the Construction and Analysis of Systems*, vol. 10806, 2018, pp. 112–131.

[16] M. Gario and A. Micheli, "PySMT: a solver-agnostic library for fast prototyping of SMT-based algorithms," in *SMT Workshop*, 2015.

[17] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *ICLR*, Y. Bengio and Y. LeCun, Eds., 2015.

[18] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay," *CoRR*, vol. abs/1803.09820, 2018.

[19] M. Fey and J. E. Lenssen, "Fast graph representation learning with pytorch geometric," *CoRR*, vol. abs/1903.02428, 2019.

[20] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *NeurIPS*, 2017.

[21] S. Kadioglu, Y. Malitsky, A. Sabharwal, H. Samulowitz, and M. Sellmann, "Algorithm selection and scheduling," in *International Conference on Principles and Practice of Constraint Programming*, 2011.

[22] K. Leyton-Brown, E. Nudelman, G. Andrew, J. McFadden, and Y. Shoham, "A portfolio approach to algorithm selection," in *IJCAI*, vol. 3, 2003, pp. 1542–1543.

[23] K. Leyton-Brown, E. Nudelman, and Y. Shoham, "Empirical hardness models: Methodology and a case study on combinatorial auctions," *J. ACM*, vol. 56, no. 4, pp. 22:1–22:52, 2009.

[24] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill, "Learning a SAT solver from single-bit supervision," in *International Conference on Learning Representations*, 2019.

[25] C. Cameron, R. Chen, J. Hartford, and K. Leyton-Brown, "Predicting propositional satisfiability via end-to-end learning," in *AAAI*, 2020.

[26] D. Selsam and N. Bjørner, "Guiding high-performance sat solvers with unsat-core predictions," in *International Conference on Theory and Applications of Satisfiability Testing*. Springer, 2019, pp. 336–353.

[27] M. Wang, Y. Tang, J. Wang, and J. Deng, "Premise selection for theorem proving by deep graph embedding," in *NeurIPS*, 2017.

[28] P. Vaezipoor, G. Lederman, Y. Wu, C. J. Maddison, R. B. Grosse, S. A. Seshia, and F. Bacchus, "Learning branching heuristics for propositional model counting," in *AAAI*, 2021.

[29] E. Khalil, P. Le Bodic, L. Song, G. Nemhauser, and B. Dilkina, "Learning to branch in mixed integer programming," in *AAAI*, 2016.

[30] M. Balunovic, P. Bielik, and M. T. Vechev, "Learning to solve SMT formulas." in *NeurIPS*, 2018, pp. 10 338–10 349.

[31] N. G. Ramírez, Y. Hamadi, É. Monfroy, and F. Saubion, "Evolving SMT strategies," in *ICTAI*, 2016.

[32] E. O'Mahony, E. Hebrard, A. Holland, C. Nugent, and B. O'Sullivan, "Using case-based reasoning in an algorithm portfolio for constraint solving," in *Irish conference on artificial intelligence and cognitive science*, 2008, pp. 210–216.

[33] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: a methodological tour d'horizon," *European Journal of Operational Research*, 2020.

[34] P. Kerschke, H. H. Hoos, F. Neumann, and H. Trautmann, "Automated algorithm selection: Survey and perspectives," *Evolutionary computation*, vol. 27, no. 1, pp. 3–45, 2019.

[35] E.-G. Talbi, "Machine learning into metaheuristics: A survey and taxonomy of data-driven metaheuristics," *ACM Computing Surveys*, 2020.

[36] Q. Cappart, D. Chételat, E. Khalil, A. Lodi, C. Morris, and P. Veličković, "Combinatorial optimization and reasoning with graph neural networks," *arXiv preprint arXiv:2102.09544*, 2021.

C Symmetries in computer vision

Segmenting out Generic Objects in Monocular Videos

Jan Hula, David Adamczyk, David Mojzisek, and Vojtech Molek

CE IT4I - IRAFM, University of Ostrava
30. dubna 22, 701 03 Ostrava, Czech Republic
{jan.hula, vojtech.molek, david.adamczyk, david.mojzisek}@osu.cz

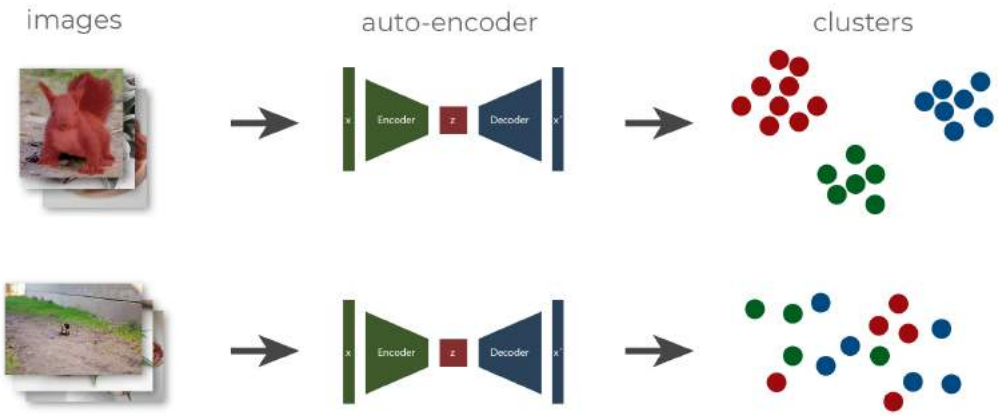


Figure 1: A schema depicting the improvement of clustering based on features from autoencoder trained on segmented objects using our approach vs. the one trained on the original images.

Abstract: We present an approach for generic object detection and segmentation in monocular videos. In this task, we want to segment objects from a background with no prior knowledge about the possible classes of objects which we may encounter. This makes this task much harder than the classical object detection and segmentation, which can be posed as a supervised learning problem. Our approach uses an ensemble of 3 different models which are trained by different objectives and have different failure modes and therefore complement each other. We demonstrate the usefulness of our approach on a custom dataset containing 18 classes of organic objects. Using our method, we were able to recover the classes of objects in a fully unsupervised way.

1 Introduction

Separating generic objects from a background in monocular videos is a challenging task. We believe that this problem is essential to Computer Vision and, as such, gained an unproportionally small amount of attention from the research community. The ability to separate objects from the background would vastly simplify other tasks as it can be viewed as a kind of dimensionality reduction on relevant features.

In image classification, object separation prevents a classifier from learning spurious correlations, which could arise when a certain class is often captured on a particular background. Object separation from a background automatically restricts a classifier to consider only features directly tied to the class label, as opposed to features only correlated with it.

As the generic object separation is a rather nonstandard task and still vaguely defined – it is not clear what should be considered an independent object – we focus on a simplified setting in which a camera captures a single salient object. Our solution is an ensemble of three different models trained for different objectives.

Furthermore, we study the impact of the background removal on the clustering properties of the resulting representations. The representations are obtained by training a neural network in an unsupervised way. We are able to recover the categories of objects in a fully unsupervised way, using our custom dataset containing videos of organic things.

Our main contributions are:

- We present an ensemble model that can separate objects from the background in monocular videos containing one salient object. The ensembled models compensate for each other failure modes.
- We demonstrate the benefits of object separation by comparing the classification accuracy of objects with and without background using clustering.

Section 2 contains related work. Section 3 describes our approach for detecting and segmenting generic objects within monocular videos. Section 4 describes how the detected objects enable unsupervised discovery of object

classes. In section 5 we describe our experiments and the dataset we test our approach on, and finally we provide a conclusion in section 6.

2 Related Work

Generic object separation is a largely an unexplored area, and therefore similar works are scarce. Our approach is most related to DINO method [1] which uses self-supervised learning with Vision Transformers. The authors introduced DINO as a form of self-distillation with no labels. They emphasize that DINO automatically learns an interpretable representation and separates the main object from the background clutter.

Lu et al. introduced an approach called CO-attention Siamese Network (COSNet) [2] for unsupervised video object segmentation. It is based on two ideas. The first is the importance of inherent correlation among video frames, and the second is the global co-attention mechanism responsible for learning motion in short-term temporal segments. The COSNet is trained on pairs of video frames, which increases the learning capacity.

The task of class discovery is marginally related to current self-supervised approaches using large amounts of unlabeled data such as [3, 4] and approaches that try to exploit coherency in the data [5].

Lastly, our approach for class discovery can be seen as a version of clustering with constraints. It has been heavily studied in the past, for example, by [6, 7].

3 Generic Object Detection and Segmentation

This section describes our approach to generic object detection and segmentation. By a generic object we mean an object of an unknown class. We use this term to distinguish it from classical object detection and segmentation, which can deal only with a concrete set of specified classes. Classical object detection and segmentation is much easier because it can be approached as a supervised learning problem on a dataset with annotated bounding boxes and segmentation masks. With generic objects, it is not that straightforward because it is not known in advance what kind of objects we will encounter at the test time.

Moreover, at first it may not be obvious how to define what should be considered as a separate object. One useful definition would be that an object is anything that can move independently from the rest of the environment. In this view, we can understand generic object segmentation as a way to factorise the visual stream into independent components. We need to mention that this definition does not cover all cases in which we would like to detect something as a separate object. Examples include buildings, letters on a sheet of paper, and other "entities" which can not move independently. Nonetheless, we consider this as our working definition because it allows us to make progress in generic object detection and segmentation.

3.1 Ensemble of Models Trained for Different Objectives

Our approach to the problem of generic object detection and segmentation is based on an ensemble of 3 models which are trained by different objectives. Even though each of these models has its own failure modes, together they constitute a robust ensemble. Concretely, we use one model trained for depth map prediction, one model trained for optical flow estimation, and one model trained for tracking of objects. Using the model for depth prediction, we can separate foreground objects based on depth, using the model trained for optical flow estimation, we can separate moving objects, and finally, using the tracker, we can verify the temporal coherency of our predictions. The tracker is initialised with a bounding box obtained from the predictions of the two other models in the frames where these predictions are most consistent. The following paragraphs provide a high-level description of these three models. For a more complete description of these models, see the respective publications.

Depth Prediction For the depth prediction, we use the model introduced by Ranftl et al. [8], available from the author's repository[1]. This transformer-based model predicts a scalar value for each pixel, which represents the distance of the surface captured by that pixel from the camera center.

Optical Flow Estimation For optical estimation, we use the model introduced by Teed et al. [9]. It is also a transformer-based model which requires 2 consecutive frames of video to produce the optical flow field. The optical flow field assigns two scalar values to each pixel. These values represent the pixel displacement on the x and y axes, relative to the previous frame. To obtain one scalar value for each pixel, we take the magnitude of the displacement. We used the implementation of the model with trained weights provided in the authors repository[2].

Object tracking For tracking objects, we use a model called SiamMask [10], a neural network trained as a Siamese architecture which simultaneously performs both visual object tracking and object segmentation in a video. We used the implementation available online[3].

[1] https://github.com/intel-isl/DPT
[2] https://github.com/princeton-vl/RAFT
[3] https://github.com/foolwood/SiamMask

3.2 Segmenting out Known Classes

In our dataset, each video captures a hand holding one object. Using our approach for generic object segmentation which is based on predicted depth maps and optical flow, our model segments out the hand together with the object. We fix this issue by segmenting out hands separately by a model trained specifically for hand segmentation.

To obtain training data for hand segmentation, we downloaded the following 4 datasets: GTEA, HandOverFace, GTEA_GAZE_PLUS, and EgoHands[4][5][6]. The architecture of the model is UNet with timm_regnetty_160 [11] as encoder and softmax2D as activation. The encoder weights were pretrained on ImageNet.

Using freely available datasets for hand segmentation mentioned above, the trained model was not working well on our dataset, probably because of a large distribution shift (most of the images in these public datasets contained hands in front of the face or were captured in the interior).

To mitigate this problem, we used a simple trick to enlarge the training data with images of hands which are similar to the images in our target dataset. Concretely, we captured our hands from a similar viewpoint as in our dataset, and then used the same model for depth prediction to produce depth maps for every 10th frame within the video. Finally, we thresholded the predicted death maps to obtain reliable segmentation masks of hands. In this way, we obtained hundreds of labeled images of hands with minimal effort. After adding this dataset to the other datasets, we obtained an accurate model for hand segmentation.

We use this model to remove hands from the mask predicted by our ensemble. More precisely, we remove the hands from the outputs of the model predicting the depth map and the model predicting the optical flow before we initialise the bounding box for the tracker. In this way we obtain masks only for the object, ignoring the hands.

3.3 Bounding Box Initialization

As mentioned above, we initialize the tracker with a bounding box obtained from the predictions of depth and optical flow maps within each frame. These predictions are two rectangular matrices of the same shape. We rescale the range of values to the interval between 0 and 1 and denote the final matrices by $f_1(x)$ and $f_2(x)$, respectively.

We first choose k frames where the predictions from these two models are most consistent. To measure this consistency, we devise the following heuristic. We first detect edges using a Canny edge detector [12] in both predictions and then measure the overlap of the resulting edges. To account for small deviations of edges in the two predictions, we blur them with a gaussian kernel of the width set to 7px to achieve their overlap if they are close to each other. Then we compute the consistency score c of frame x using the following formula:

$$c(x) = \sum_{i \in Pixels(x)} e^{\varepsilon 1_i + \varepsilon 2_i}, \tag{1}$$

where $\varepsilon 1$ and $\varepsilon 2$ are the two blurred edge maps from the two predictions and i indexes individual pixels. Next, we obtain the aggregated predictions for each pixel i by:

$$y(x)_i = e^{f_1(x)_i + f_2(x)_i} + \frac{1}{|Pixels(x)|} \sum_{j \in Pixels(x)} e^{f_1(x)_j + f_2(x)_j}. \tag{2}$$

We exponentiate the sum of the predictions from the two models because we want these predictions to interact superlinearly. We also subtract the mean of this value taken over all pixels within the image to make the aggregated predictions centered at zero. Therefore, the pixels where no object was predicted will contain negative values.

Once we have the aggregated predictions for the selected frames, we initialize the bounding boxes for the tracker. For this, we again devise a score which captures how well a given bounding box ($bbox$) covers pixels with high values (signifying that an object is present) and at the same time excludes pixels with low values. It has the following form:

$$\text{bboxScore}(bbox, y(x)) = \sum_{i \in Pixels(x)} \text{isInBbox}(i, bbox) \cdot y(x)_i, \tag{3}$$

where the function isInBbox returns -1 if the pixel is not contained in the bounding box and 1 otherwise.

Finally, we optimize the coordinates of the bounding box using CMA-ES [13] which is a derivative-free optimization algorithm used for the optimization of continuous parameters. The optimization tries to find coordinates which maximize this score. At the end of this procedure, we obtain k frames with bounding boxes in each video. The quality of predictions and the resulting bounding box is shown in Figure 2.

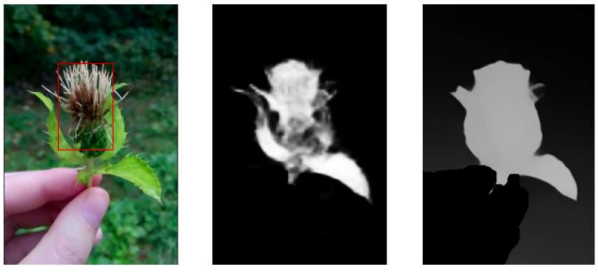


Figure 2: Predictions from the model for optical flow estimation (middle) and depth estimation (right). The initialized bounding box is depicted in the RGB image.

[4] http://cbs.ic.gatech.edu/fpv/

[5] https://www.cl.cam.ac.uk/research/rainbow/emotions/hand.html

[6] http://vision.soic.indiana.edu/projects/egohands/

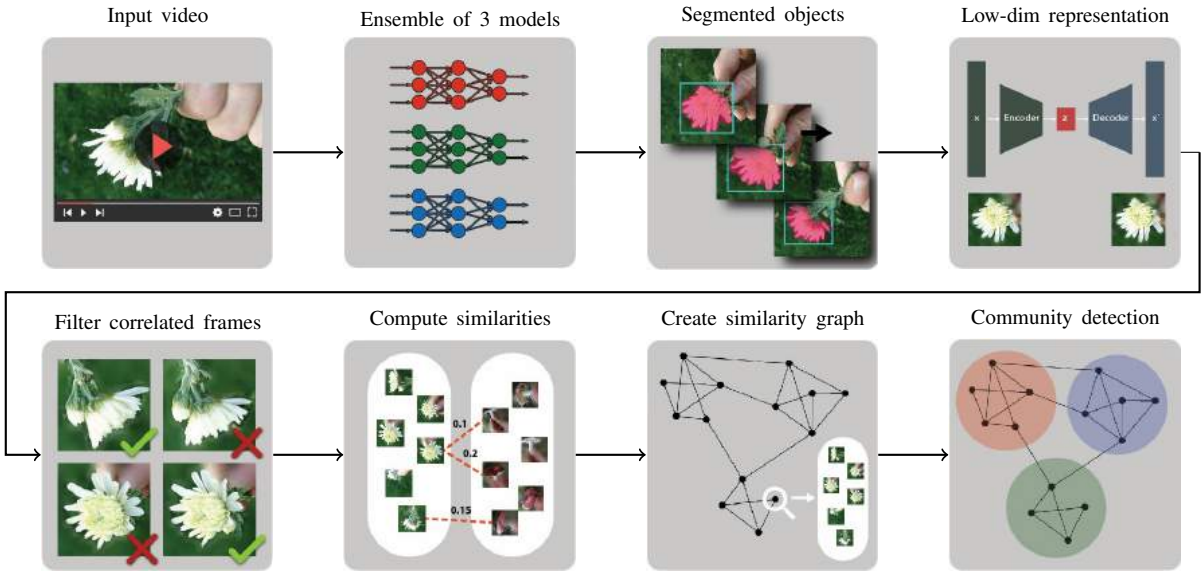


Figure 3: This figure depicts the whole pipeline of our approach. The diagram describes the process of our approach from the first step, where we preprocess the input video for the last step, where we obtain sets of similar objects.

3.4 Verification with the Tracker

Using the chosen frames and their bounding boxes, we initialize the tracker and let it track the object in between the selected frames. The tracker provides another layer of consistency check. Once we have the predictions from the three models (denoted by $f_1(x)$, $f_2(x)$ and $f_3(x)$), we can treat the consistency of these predictions as a certainty of the whole ensemble. To measure this certainty, we again compute the consistency score as we did in the selection of reliable frames in the Equation 1. Using an empirically estimated threshold, we filter out frames with low consistency and for each pixel i in the filtered frames, we aggregate the predictions of the ensemble with the following formula:

$$\text{output}(x)_i = \frac{\min\left(e^{\sum_{j=1}^{3} f_j(x)_i} - 1, e^2 - 1\right)}{e^2 - 1}, \qquad (4)$$

The subtraction of 1 ensures that we get 0 when all three models predict 0. Thresholding and dividing by $e^2 - 1$ insures that we obtain a value close to 1 when at least two models predict values close to 1. Finally, we obtain a bounding box for each frame using the same method as in the bounding box initialization (optimization using CMA-ES), i.e., minimizing the objective in Equation 1.

The whole process can be viewed as certainty propagation. We first select a few frames where the first two models agree on their predictions and from these the tracker propagates the certainty to other frames.

Evaluating The Quality of the Aggregated Predictions Our final task is a discovery of classes of objects within monocular videos. We view the generic object detection and segmentation as an intermediate step towards this goal. Therefore, we evaluate the usefulness of our approach on this target task. That is, we test how well are we able to recover classes of objects from images where the objects were segmented out by our approach. We compare it to the setup where we use the same algorithm for class discovery but where we use the original images with a background. We also mention that we do not require pixel-perfect segmentation masks as our goal is only to focus on the relevant parts of the image, so that the measured similarity between images will mostly reflect the similarity of objects and not of backgrounds. The next section describes our pipeline for the task of class discovery

4 Class Discovery

The algorithm for class discovery was proposed in [14]. Its input is a set of videos, each containing one object and each represented as a sequence of images. The goal is to find clusters of videos based on the similarity between them. Generally, our algorithm works in 3 steps:

1. Measure the similarity between every pair of videos with the method described in Section 4.1.
2. Construct a similarity graph by connecting each video to its five most similar videos.
3. Apply the Louvain community detection algorithm [15] to detect the highly interconnected parts of the graph and consider these as the discovered classes.

The advantage of the Louvain algorithm is that it needs no apriori knowledge of the number of clusters/communities.

The accuracy of our approach is measured in two ways. First, by counting how many times a video was assigned

to an incorrect cluster. Second, whether the algorithm discovered all clusters. It is clear that the final accuracy mostly reflects the measured similarity between individual videos.

4.1 Computing Similarity between a Pair of Videos

Each video is represented by a sequence of images, but to compute the similarity, we ignore the ordering and treat the sequence as a set. The similarity between the two videos is computed in the following four steps:

1. Train an autoencoder using images from all videos to obtain a low-dimensional representation z_i of each image x_i.
2. In each video, select n representative frames which are not correlated, described in 4.2.
3. For each pair of videos, compute all pairwise similarities $d(z_i, z_j)$ with cosine distance.
4. Finally, select the l most similar pairs of images and average their similarities to obtain the final similarity between two videos.

The intuition behind step 4 is that videos of similar objects may contain only a few frames where these objects are captured from the same angle or in the same situation.

4.2 Filtering out Correlated Frames

Step 2 of similarity computation takes n representative frames. If we would simply use all frames from each video, the distribution of the dataset may end up skewed because some parts of a video may be more static than others. These static parts would produce many correlated frames. Therefore, the correlated frames need to be filtered out from a given video. We first test whether the subjective visual similarity of images can be captured by cosine similarity between their low-dimensional representations obtained in step 1 of the similarity computation. As can be seen in Figure 4, it captures the visual similarity well enough.

To extract n uncorrelated frames from each video, we run k-means clustering, where $k = n$, on the low-dimensional representations and take the most similar frame to every centroid of the resulting clusters. This simple heuristic produces uncorrelated images.

To conclude this section, if the low-dimensional representation of individual images obtained by the autoencoder reflects the similarity between the captured objects and not some other irrelevant factors, we may expect to obtain meaningful clusters. Moreover, note the benefit of creating the similarity graph of videos instead of individual images. All images in one video are automatically linked together. If a few images in 2 videos are similar, this similarity is propagated to other frames within the video,

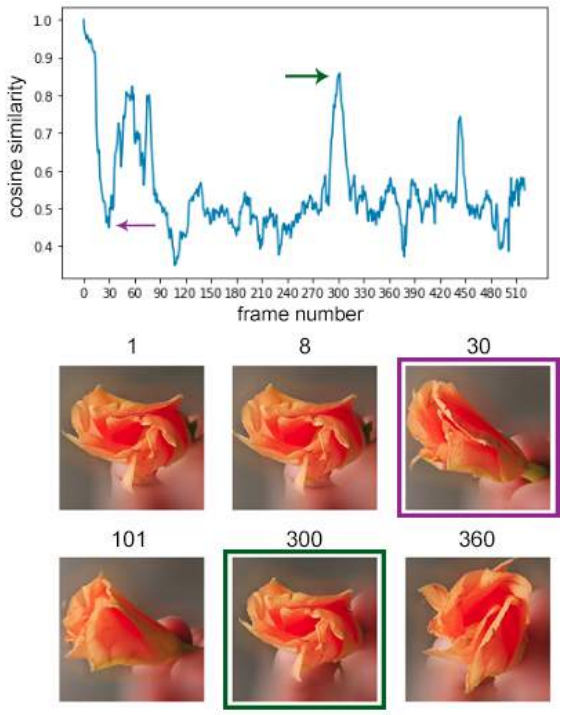


Figure 4: ***Top***: A graph of cosine similarity between the first and other frames in the video. ***Bottom***: Selected frames from the video with their corresponding frame numbers. The two highlighted frames correspond to two arrows in the graph.

Figure 5: Samples from the Organic Objects dataset. Images are cropped and have blurred background.

which would otherwise not be linked based only on the similarity. The video contains a constraint that says that the object cannot change its class in time.

5 Experiments

To test the algorithm for object discovery, we assembled a custom dataset of organic objects. The dataset contains 18 classes of organic objects, some of which are depicted in Figure 5. We have chosen organic objects because they naturally produce large variability between instances. For every class, we capture ten different samples from different viewpoints. The final dataset can be downloaded at the following address – `github.com/Jan21/`

`Organic-objects-dataset`.

Using our ensemble described in Section 3, we segment the object in every frame of each video. Using the resulting bounding boxes and segmentation masks, we crop each image and blur the background to suppress the distinctive features present in the background.

To obtain the low-dimensional representations used to filter out correlated frames and construct the similarity graph, we resize all images to a fixed resolution of 64×64 pixels and train a convolutional autoencoder. The autoencoder has five convolutional layers (16, 32, 64, 128, 256 filters with stride 2) and one fully connected layer[7] with dimensions $1024 \rightarrow 96$. [8]

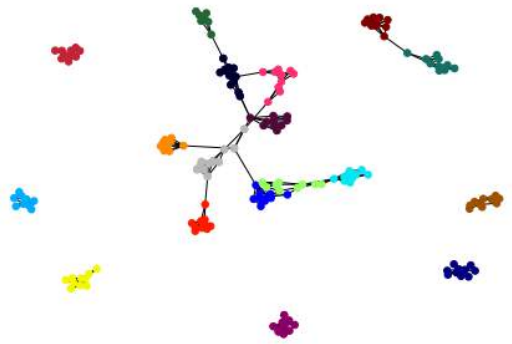


Figure 6: Visualization of detected communities in the Organic Objects dataset with the Louvain method. Nodes are colored according to the component (community) they are assigned to. The method discovered 18 components which belong to 18 different classes.

5.1 Results

After running community detection on the similarity graph, we inspected how many image bundles were assigned to the wrong component. Out of 173 videos, only 5 in the training set were assigned to the wrong component. The result of community detection on the constructed similarity graph is shown in Figure 6. Numerical results and comparison with clustering of non-segmented images are presented in Table 1. From the accuracy and number of discovered classes, it is clear that background removal created a significant accuracy difference of 56.1% and a difference in the number of correctly discovered classes.

6 Conclusion

In this contribution, we present a method for generic object detection and segmentation, which uses an ensemble of three different models trained by three different objectives. To demonstrate the effectiveness of the approach, we have created a custom dataset of organic objects. The dataset was used in our pipeline to remove background, create low-dimensional representations, and perform class discovery and classification. We have shown that background removal significantly increases the accuracy of the classification and the number of correctly discovered classes.

In future work, we plan to optimize our approach for speed, generalize it to work with videos containing multiple objects, and make the class discovery an online process that can discover new classes on-the-fly.

[7] Image is downscaled to 2×2 times 256 filters $\rightarrow$ 1024 input vector to the fully-connected layer.

[8] We also tried to extract representations by using VGG16 which was pre-trained on ImageNet. These representations better discriminated very similar objects (e.g., two types of red flowers).

References

[1] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin, "Emerging properties in self-supervised vision transformers," *arXiv preprint arXiv:2104.14294*, 2021.

[2] X. Lu, W. Wang, C. Ma, J. Shen, L. Shao, and F. Porikli, "See more, know more: Unsupervised video object segmentation with co-attention siamese networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3623–3632, 2019.

[3] Q. Xie, E. Hovy, M.-T. Luong, and Q. V. Le, "Self-training with noisy student improves imagenet classification," *arXiv preprint arXiv:1911.04252*, 2019.

[4] H. Bagherinezhad, M. Horton, M. Rastegari, and A. Farhadi, "Label refinery: Improving imagenet classification through label progression," *ArXiv*, vol. abs/1805.02641, 2018.

[5] P. Bhattacharjee and S. Das, "Temporal coherency based criteria for predicting video frames using deep multi-stage generative adversarial networks," in *Advances in Neural Information Processing Systems*, pp. 4268–4277, 2017.

[6] I. Davidson and S. Ravi, "Clustering with constraints: Feasibility issues and the k-means algorithm," in *Proceedings of the 2005 SIAM international conference on data mining*, pp. 138–149, SIAM, 2005.

[7] S. Basu, M. Bilenko, A. Banerjee, and R. J. Mooney, "Probabilistic semi-supervised clustering with constraints," *Semi-supervised learning*, pp. 71–98, 2006.

[8] R. Ranftl, A. Bochkovskiy, and V. Koltun, "Vision transformers for dense prediction," *arXiv preprint arXiv:2103.13413*, 2021.

[9] Z. Teed and J. Deng, "Raft: Recurrent all-pairs field transforms for optical flow," in *European Conference on Computer Vision*, pp. 402–419, Springer, 2020.

[10] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1328–1338, 2019.

[11] P. Yakubovskiy, "Segmentation models pytorch." `https://github.com/qubvel/segmentation_models.pytorch`, 2020.

Pre-segmenting the objects	Number of discovered classes	Accuracy
Yes	18	97.1%
No	15	41.0%

Table 1: Comparison of the clustering of videos with and without the background removed. The accuracy is computed by checking how many times a video was assigned to incorrect cluster.

[12] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[13] N. Hansen, "The cma evolution strategy: a comparing review," *Towards a new evolutionary computation*, pp. 75–102, 2006.

[14] J. Hula, "Unsupervised object-aware learning from videos," in *2020 IEEE Third International Conference on Data Stream Mining & Processing (DSMP)*, pp. 237–242, IEEE, 2020.

[15] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.