Leveraging Structure in Computer Vision Tasks for Flexible Deep Learning Models

by

Amélie Royer

September, 2020

A thesis presented to the
Graduate School
of the
Institute of Science and Technology Austria, Klosterneuburg, Austria
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

IST AUSTRIA
Institute of Science and Technology

The thesis of Amélie Royer, titled *Leveraging Structure in Computer Vision Tasks for Flexible Deep Learning Models*, is approved by:

Supervisor: Prof. Christoph Lampert, IST Austria, Klosterneuburg, Austria

Signature: ______________________

Committee Member: Prof. Vladimir Kolmogorov, IST Austria, Klosterneuburg, Austria

Signature: ______________________

Committee Member: Prof. Devi Parikh, Georgia Institute of Technology, Atlanta, USA

Signature: ______________________

Committee Member: Prof. Bernt Schiele, Max Planck Institut für Informatik, Saarbrücken, Germany

Signature: ______________________

Defense Chair: Defense Chair Name, IST Austria, Klosterneuburg, Austria

Signature: ______________________

signed page is on file

IST Austria Thesis, ISSN: 2663-337X

ISBN: 978-3-99078-007-7

I hereby declare that this thesis is my own work and that it does not contain other people's work without this being so stated; this thesis does not contain my previous work without this being stated, and the bibliography contains all the literature that I used in writing the dissertation.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee, and that this thesis has not been submitted for a higher degree to any other university or institution.

I certify that any republication of materials presented in this thesis has been approved by the relevant publishers and co-authors.

Signature: ______________________

Amélie Royer

September, 2020

signed page is on file

Abstract

Deep neural networks have established a new standard for data-dependent feature extraction pipelines in the Computer Vision literature. Despite their remarkable performance in the standard supervised learning scenario, *i.e.* when models are trained with labeled data and tested on samples that follow a similar distribution, neural networks have been shown to struggle with more advanced generalization abilities, such as transferring knowledge across visually different domains, or generalizing to new unseen combinations of known concepts.

In this thesis we argue that, in contrast to the usual black-box behavior of neural networks, leveraging more structured internal representations is a promising direction for tackling such problems. In particular, we focus on two forms of structure. First, we tackle *modularity*: We show that *(i)* compositional architectures are a natural tool for modeling reasoning tasks, in that they efficiently capture their combinatorial nature, which is key for generalizing beyond the compositions seen during training. We investigate how to to learn such models, both formally and experimentally, for the task of abstract visual reasoning. Then, we show that *(ii)* in some settings, modularity allows us to efficiently break down complex tasks into smaller, easier, modules, thereby improving computational efficiency; We study this behavior in the context of generative models for colorization, as well as for small objects detection. Secondly, we investigate the inherently layered structure of representations learned by neural networks, and analyze its role in the context of transfer learning and domain adaptation across visually dissimilar domains.

Acknowledgments

First of all, I would like to thank my advisor Christoph H. Lampert, who made it all possible by accepting me into his group and mentoring me. My first experience with Christoph's research group occurred was when I did an internship during the first year of my Masters studies. I liked it so much that I decided to come back for a PhD at IST Austria and affiliated with the, then, Computer Vision and Machine Learning group. Thanks for fostering varied research interests in the group, for instigating our weekly reading group tradition, and for providing a good balance between research independence and advice, be it during my internship, rotation or PhD. Thank you also Christoph for always being available for discussion, and for giving me plenty of opportunities to grow as a researcher and as a person, by going to conferences, summer schools, assisting in teaching courses, and even going on internships. I would also like to thank my thesis committee, Devi, Bernt and Vladimir for providing me useful feedback throughout this journey, from the qualifying exam to the defense. Thanks also to Krish and Chris for hosting my first year's rotations.

Keeping motivated through PhD years can be challenging at times, so I would like to thank my friends (and colleagues) for their support over the years. In particular, I thank the PhD students of the 2015 cohort for making the first year of grad school a lot of fun, despite the amount of work (and living on a remote campus). I was really happy that our paths did not diverge after surmounting defeating the modeling course and the qualifying exam together: True to IST's interdisciplinary tradition, I have made friends across varied fields and I think this really helped broaden my horizon, as well as my vocabulary (at least by an aliquot of words). Thank you Feyza* and Lena* (*: equal authorship) for always being there for the past few years, with fun times and trips, and helping me to finally be arrived. I wish you to graduate with a lot of success and happiness ♡ ! Thanks Priscila for organizing so many fun activities in the first year, and making the office(s) life much more lively. Thanks also to the "Food people" for the social dinner meetings and great Viennese restaurants discoveries. Thank you Sergey, Rok and Peter for the DnD games and fun discussions. Thanks Nish for sharing so many pearls of wisdom. Finally, I would also like to thank my family for being so supportive through my studies, even without fully understanding what I have been working on :).

I would also like to thank my group, former and current members, as well as officemates, for making everyday work more fun and exciting through, among other things, philosophical lunch discussions and intense table soccer games. Thank you Alex, Alex, Alex (hope I did not forget any), Mary and Niko, my fellow PhD students through this journey :). Thank you also Asya, Bernd, Chris, Csaba, Ehsan, Elias, Emilie, Georg, Hung, Jonny, Kimia, Marian, Michelle, Paul, Remy, Soroush, Sylvestre, Viktoriia. Thanks guys for creating a great work atmosphere ! Thank you Alex K. for teaching me all your Tensorflow (and Jax) tricks. It was great collaborating with you both at IST and Google, I've learned a lot ! Thank you Asya for being my travel buddy at my first conference back in 2015. And thank you Niko for taking over the reading group leadership, and helping "third floor's science" to prosper.

Finally, I've spent a few months of my PhD as a research intern in the Google Brain team. I would like to thank everyone who advised me there, Konstantinos, Stephan, Fred, Lukas, Alex, for creating such a welcoming and productive environment. Thanks in particular to Lukas and Alex for being great mentors during my "work from home 2020 internship". I would also like to thank all my fellow "funterns": Thanks for making my first internship such a fun experience, filled with fun activities, free food and swag hunting, and endless Codenames games :).

Last but not least, I would like to acknowledge the support of the IST IT and scientific computing team for helping provide a great work environment. I also gratefully acknowledge the support of NVIDIA Corporation and Facebook with the donation of the GPUs used for this research.

About the Author

Amélie graduated with a Masters degree from École Normale Supérieure de Rennes (ENS Rennes), France, in 2015. During her studies, she visited IST Austria for a summer internship in 2013. This motivated her to later join IST Austria as a PhD student in 2015, before affiliating with the Computer Vision and Machine Learning group. During her PhD, Amélie worked on structured representations for Computer Vision, with a focus on object detection and visual reasoning, and presented her work at several international conferences. She also completed two research internships at Google Brain, in London and Zurich.

List of Publications

1. **A. Royer**, A. Kolesnikov and C. H. Lampert. *Probabilistic Image Colorization*. British Machine Vision Conference, 2017.

2. **A. Royer**, K. Bousmalis, S. Gouws, F. Bertsch, I. Mosseri, F. Cole and K. Murphy. *XGAN: Unsupervised Image-to-Image Translation for Many-to-Many Mappings*. Domain Adaptation for Visual Understanding Workshop at the International Conference of Machine Learning, 2018.

3. K. Chatterjee, M. Chmelík, D. Karkhanis, P. Novotný and **A. Royer**. *Multiple-Environment Markov Decision Processes: Efficient Analysis and Applications*. International Conference on Automated Planning and Scheduling, 2020.

4. **A. Royer**, C. H. Lampert. *Localizing Grouped Instances for Efficient Detection in Low-Resource Scenarios*. Winter Conference on Applications of Computer Vision, 2020.

5. **A. Royer**, C. H. Lampert. *A Flexible Selection Scheme for Minimum-Effort Transfer Learning*. Winter Conference on Applications of Computer Vision, 2020.

Contents

List of Tables

List of Figures

List of Abbreviations

$\mathcal{X}$ denotes the input space of a neural network, in our case images. Additionally, X denotes a random variable on that space and x a realization (observation) of this variable

$\mathcal{Y}$ denotes the output space of a neural network

$\mathcal{D}$ usually denotes a distribution over $\mathcal{X} \times Y$

f_θ denotes a neural network with parameters θ

$\mathcal{R}$ is the risk function, and $\hat{R}$ is its empirical counterpart

$\mathcal{L}$ denotes a loss function

AVR Abstract Visual Reasoning

CNN Convolutional Neural Network

DANN Domain Adversarial Neural Network

GAN Generative Adversarial Network

iid independently identically distributed

ODGI Object Detection with Grouped Instances

MLP Multilayer Perceptron

PIC Probabilistic Image Colorization

SGD Mini-batch Stochastic Gradient Descent

VAE Variational Autoencoder

1 Introduction

Computer vision is a vast research field, that is driven by the ambition of achieving automatic image understanding. This ranges from recognizing and processing low-level image statistics (*e.g*. edge detection, texture analysis), to extracting higher-level semantics of objects (*e.g*. image classification, object detection, semantic segmentation), and eventually building models able to provide a rich and meaningful description of an image (*e.g*. image captioning, scene graph generation). Computer vision models have been successfully applied to numerous real-life tasks. They are now part of our daily lives, for instance in item recommendation systems [Zalando Research, 2018] or image filters on social networks [Facebook Research, 2016], and are leading the way to exciting future applications, such as medical diagnostic assistance [Razzak *et al*., 2018], rescue drones [Giusti *et al*., 2016] and self-driving cars [Sun *et al*., 2019a].

A key problematic of computer vision revolves around mapping images to adequate *feature representations*. In fact, digital images are usually represented as arrays of discrete pixel intensities expressed in the RGB color space; Besides being extremely high dimensional (each pixel has 255^3 possible values), these representations are hard to reason with and lack natural properties of images: For instance, a minor translation of an image, although unnoticeable to the human eye, significantly alters its representation in pixel space. Instead, early computer vision systems capitalized on hand-crafted *feature extraction* pipelines for mapping an image to a lower dimensional vector representation, while preserving its most meaningful features and ideally displaying interesting properties such as translation or rotation invariance: This includes specific image analysis techniques, such as the Hough transform [Duda and Hart, 1972], used to detect lines or circular patterns, to local feature descriptors of keypoints of interest such as SIFT [Lowe, 1999; Bay *et al*., 2006], which have then been extended to more compact structured alternatives, for instance using bag-of-words models [Csurka *et al*., 2004] or Fisher vectors [Jégou *et al*., 2012]. On one hand, these representations have been used with great success in computer vision pipelines for image retrieval or keypoint matching [Sivic and Zisserman, 2003;

Zheng *et al.*, 2018]. On the other hand, for tasks that require a higher-level semantic understanding of a scene, for instance, to identify and reason about objects rather than local keypoints, *neural networks* have established a new standard for automatic image feature extraction over recent years. Following their historical achievement on the ImageNet classification challenge in 2012 [Deng *et al.*, 2012; Krizhevsky *et al.*, 2012], they are now widely used in contemporary computer vision systems.

The main strength of neural networks lies in that they are *end-to-end* models that learn rich feature representations of images directly from their pixel specification, provided that enough data samples are available, rather than relying on hand-crafted techniques. Furthermore, even though such pipelines are always learned from a specific set of training images, they exhibit remarkable transferability properties across different datasets in practice [Yosinski *et al.*, 2014]. Unfortunately, their strength is a double-edged sword, as neural networks are essentially *black boxes*: Because their "train of thought" is concealed, it is nearly impossible to characterize what the learned representations capture [Olah *et al.*, 2017; Zeiler and Fergus, 2014]. This is obviously an issue when it comes to interpreting such systems, and, more importantly, this lack of transparency may lead to inexplicable and serious failure cases for real-life applications. From a broader perspective, understanding *if* and *how* neural networks adapt to new unseen inputs is a very active research question, both from a theoretical [Kawaguchi *et al.*, 2020; Ben-David *et al.*, 2010] and practical [Frankle and Carbin, 2019; Morcos *et al.*, 2018; Szegedy *et al.*, 2014] perspective.

In contrast, humans are equipped with numerous explainable priors for solving vision tasks. For instance, when looking for a specific item in a scene, rather than attending to every single object, we will first take a short glance to locate its most likely position, and then refine our location estimate if need be. This prior enables *fast* detection, and is for instance used in cascade models for object detection [Viola and Jones, 2004]. Or, we might use some mental layout to guide our estimate, *e.g.* we know that books are more likely to be located on a bookshelf; This illustrates our ability to extract *abstract patterns* and to efficiently reuse them in new scenes. Similarly, incorporating structure priors in machine learning systems, and in particular neural networks, has been shown to be beneficial for a variety of computer vision tasks; A key challenge being to do so without limiting their ability to freely learn from data nor falling back to hand-crafted heuristics. Successful examples include both the use of *implicit* priors, *e.g.* using pre-trained word embeddings for one-shot learning systems [Frome *et al.*, 2013], and *explicit* priors, the most common example being the use of graphical models to capture the relationship between *e.g.* neighboring pixels [Zheng *et al.*, 2015] or parts of an object [Felzenszwalb *et al.*, 2008].

In this thesis, we investigate scenarios in which the structure of some computer vision tasks can be leveraged for building efficient neural networks, with a focus on *(i)* improving the trade-off between inference speed and prediction accuracy, and *(ii)* improving a model's generalization and transfer abilities. In this section, we introduce the background and relevant literature necessary to appreciate the material in subsequent chapters. We then conclude this section with a general outline of the thesis.

1.1 Machine Learning Fundamentals

Machine learning refers to the problem of automatically *learning* general trends from finite data. Formally, the goal is to estimate a target *ground-truth* function $f^*: \mathcal{X} \rightarrow \mathcal{Y}$, from input space $\mathcal{X}$ to output space $\mathcal{Y}$, given observed realizations of this function, which we denote as samples (x, y) drawn from a *training* distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$.

In general, a machine learning model is mainly characterized by three components: Firstly, an *hypothesis space*, $\mathcal{F}$, which defines the set of candidates that the model will search through to approximate the target function. Secondly, a *loss function* (also known as objective function), $\mathcal{L}: \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$, is used to estimate the quality of a candidate by comparing its outputs to the observed realizations of f^*. Formally, we define the risk of a candidate hypothesis $f \in \mathcal{F}$ as:

$$\mathcal{R}(f) \triangleq \mathbb{E}_{(x,y)\sim\mathcal{D}}\ \mathcal{L}(f(x), y) \tag{1.1}$$

Based on this quantity, the learning task amounts to finding the hypothesis with the lowest risk, by solving the minimization problem in Equation 1.2. Therefore, the *optimization procedure* chosen to that end is the third decisive component of the model.

$$\arg\min_{f \in \mathcal{F}}\ \mathbb{E}_{(x,y)\sim\mathcal{D}}\ \mathcal{L}(f(x), y) \tag{1.2}$$

However, in the real world, the distribution $\mathcal{D}$ is unknown, thus we cannot directly compute the expected risk in (1.2). Instead, we are only given a finite set of observations, the *training set*, and must rely on empirical estimates of the risk. In the rest of the section, we introduce the standard framework for training machine learning models in practice, with a detailed example for the image classification task.

1.1.1 Training machine learning models with supervision

A machine learning model is usually trained under the *supervision* of a training set D, consisting in a finite set of n observations, $D = \{(x_1, y_1), \dots (x_n, y_n)\}$, drawn from the training distribution $\mathcal{D}$. We say the model is *supervised* because each input sample $x \in \mathcal{X}$ is annotated with a label $y \in \mathcal{Y}$, which provides a ground-truth target to the model. Based on the observed samples in D, we can derive an estimate for the expectation in (1.1), which leads to the definition of the *empirical risk*, $\hat{\mathcal{R}}$:

$$\forall f \in \mathcal{F},\ \mathcal{R}(f) \simeq \hat{\mathcal{R}}(f) \triangleq \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f(x_i), y_i) \tag{1.3}$$

Moreover, the hypothesis space is often defined as a family of functions that follow a given parametric form, $\mathcal{F} = \{f_\theta,\ \forall \theta \in \mathbb{R}^d\}$, where d is the dimension of the parameter space; For instance, the space of linear functions for $\mathcal{X} = \mathbb{R}^d$, $\mathcal{F} = \{x \mapsto \theta^T x,\ \forall \theta \in \mathbb{R}^d\}$.

To summarize, the search for a function hypothesis in $\mathcal{F}$ that best matches the training dataset D is captured by the following *Empirical Risk Minimization* (ERM) problem:

Empirical Risk Minimization (ERM)

$$\boxed{\hat{\theta} = \underset{\theta \in \mathbb{R}^d}{\arg\min} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(f_\theta(x_i), y_i)} \tag{1.4}$$

Gradient descent (GD). The mathematical optimization literature provides us with a framework to tackle the minimization problem in (1.4). The most commonly used optimization algorithm in machine learning is *gradient descent* (GD), which relies on the fact that, given a function $\mathcal{L}$ observed at a point x, the steepest decrease direction is the one opposite of the gradient at x, *i.e.*- $\nabla \mathcal{L}(x)$. When applied to the ERM problem, the gradient descent consists in updating the parameters θ of the model by taking iterative small steps in the direction opposite of the gradient of the expected risk, as a function of the parameters θ, until a stopping criterion is reached (*e.g*. convergence criterion, time limit ...). Formally, the gradient update rule at each step is:

$$\forall t,\, \theta_{t+1} = \theta_t - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(f_\theta(x_i), y_i) \tag{1.5}$$

where α, the learning rate, determines the magnitude of the update. The resulting gradient descent algorithm is summarized in Algorithm 1. In particular, it requires an initial estimate for the model parameters, θ_0, and a stopping criterion to determine the final step T at which the optimization algorithm outputs its prediction $\hat{\theta} = \theta_T$. We will discuss how these components are selected in practice in Section 1.1.2.

Algorithm 1: Gradient Descent (GD)

Input : $\boldsymbol{D}$ – Training dataset
$\boldsymbol{\mathcal{L}}$ – Loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$
$\boldsymbol{\alpha}$ – Learning rate
$\boldsymbol{\theta_0}$ – Initial model parameters

$\theta \leftarrow \theta_0$;
while *stopping criterion is not met* **do**
  $\theta \leftarrow \theta - \alpha \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta \mathcal{L}(f_\theta(x_i), y_i)$
return θ

Unfortunately, there are two main limitations inherent to gradient descent. First, in order to define a gradient at all, we need f_θ to be *differentiable* with respect to the parameters θ, which limits the choice of $\mathcal{F}$. Similarly, the loss function $\mathcal{L}$ needs to be differentiable with respect to its first input: For instance, the Euclidean distance is a simple differentiable function that is commonly used to measure a distance between ground-truth values and predicted outputs. However, standard metrics, in particular for discrete output spaces, are not always differentiable. A classical example is the so-called "0-1 loss" for binary classification, which is defined as $\mathcal{L}(\hat{y}, y)$ taking value 1 when $y = \hat{y}$ and otherwise 0. It is non-differentiable (and even non-discontinuous). While there exists methods to handle non-differentiable optimization problems, these are often more complex and constraining than gradient descent. Thus, when encountering a non-differentiable objective function, it is common to approximate it with a differentiable function that can be optimized with standard gradient descent. We will consider an example of such an approximation in the course of this section, for the task of image classification, in Section 1.1.3.

A second downside of gradient descent is that it is only guaranteed to converge to a global minimum when the function to optimize is *convex*, for a proper choice of the learning rate α [Boyd and Vandenberghe, 2004]. If the function is non-convex, which is the case for instance for neural networks, GD may only converge to a local minimum.

Mini-batch stochastic gradient descent (SGD). One practical drawback of the gradient descent update rule stated in (1.5) is that it requires computing the average of gradients over the whole dataset D for a *single update* of the parameters, which is both computationally and memory demanding when dealing with large datasets. Instead, most machine learning models are trained via *mini-batch stochastic gradient descent* (SGD), which consists in approximating the gradient sum from (1.5) in an online manner, by repeatedly estimating gradients on a small random sample of the data rather than the whole dataset. The stochasticity arises from the fact that this sample of data, also called *batch* or *mini-batch*, is chosen randomly at each iteration. SGD thus introduces a new training parameter, bs, which determines the size of the batches. In particular when $bs = |D|$, we recover the gradient descent algorithm. The pseudo-code for SGD algorithm is described in Algorithm 2.

Algorithm 2: Mini-batch Stochastic Gradient Descent (SGD)

```
Input  : D – Training dataset
         ℒ – Loss function ℒ : 𝒴 × 𝒴 → ℝ
         α – Learning rate
         bs – Batch size
         θ_0 – Initial model parameters
θ ← θ_0;
while stopping criterion is not met do
    shuffle dataset D;
    for batch = 1 ... ⌊|D| // bs⌋ do
        k ← batch * bs;
        θ ← θ − α (1/n) Σ_{i=k}^{k+bs} ∇_θ ℒ(f_θ(x_i), y_i)
return θ
```

In theory, SGD enjoys the same convergence guarantees as GD, albeit with slower convergence rates, as well as the same drawbacks (*e.g.* potential convergence to a local minimum). Nevertheless, in practice, mini-batch stochastic gradient descent is the *de facto* optimization algorithm for neural networks, due to its simplicity, scalability to large datasets, and quality of the found solutions [Bottou *et al.*, 2018].

1.1.2 Model selection and generalization

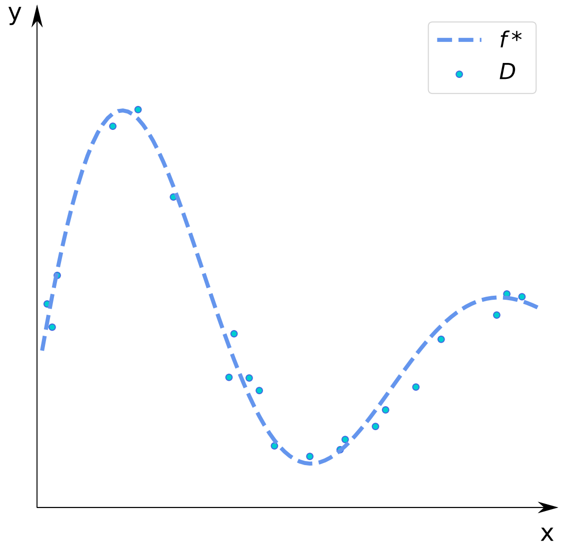
So far, we have only described how to *learn* a model from a training dataset D. However, ultimately, we also need to assess how well the learned function $f_{\hat{\theta}}$, actually fits the ground-truth function f^* that underlies the dataset D.

Influence of the hypothesis space. Because the model's search is limited to the hypothesis space $\mathcal{F}$ by definition, the *capacity* of $\mathcal{F}$ directly impacts how closely one can approximate the true underlying data distribution.

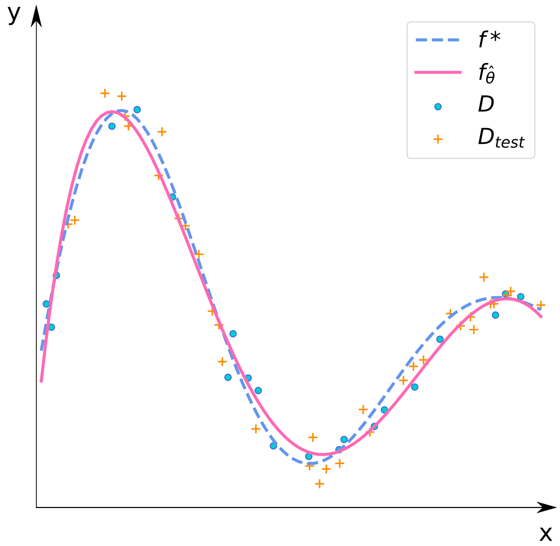
On the one hand, it may happen that functions in $\mathcal{F}$ are simply not complex enough to capture f^*; This is for instance the case when $\mathcal{F}$ is the space of polynomials of degree 2 and $f^* : x \mapsto x^3$. This phenomenon is called *underfitting*. On the other hand, if the capacity of $\mathcal{F}$ is too high, the learned function may end up capturing noise specific to the training samples, hence becoming a very good approximation of f^* on D but a poor one for new inputs drawn from the true distribution $\mathcal{D}$. This is a common problem when working with real-life data, due to *e.g*. sensor noise, annotation mistake, data deterioration etc. We say that the model *overfits* to the statistical noise present in the data and fails to generalize. Both phenomena are illustrated in Figure 1.1.

In summary, there is an inherent trade-off when choosing how complex the hypothesis space should be. Intuitively, the capacity of $\mathcal{F}$ represents how many of degrees of freedom it has, *e.g*. the degree of polynomial functions. More generally, there exists other formally defined measures of the capacity of an hypothesis class, for instance the Vapnik-Chervonenkis (VC) dimension [Shalev-Shwartz and Ben-David, 2014].

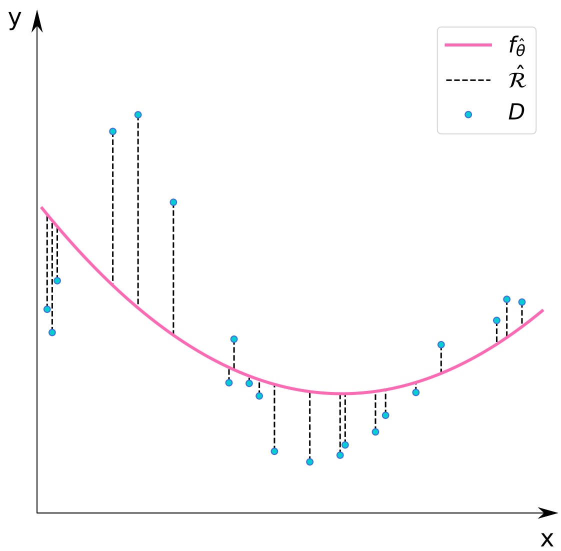
Model evaluation. Once trained, a natural way to evaluate the model would be via its empirical risk as, by definition, it measures how far $f_{\hat{\theta}}$ is from f^* on the dataset D, under the loss function $\mathcal{L}$. However this is an insufficient benchmark as it does not speak for how well the model *generalizes* outside of the training set: For instance, a model that overfits to the training data would have a very low empirical risk. Instead, it is common to assess the *generalization* of a machine learning model by evaluating its performance on a separate test dataset, D_{test}. In general, it is assumed that the test dataset is sampled from the same distribution $\mathcal{D}$ as the training data, but independently from the latter, such that the training process is independent from the data it is evaluated on.



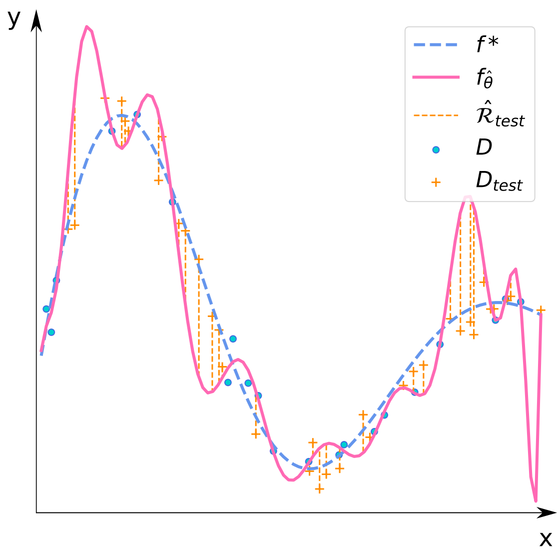
(a) **[Ground-truth]** $f^* : x \mapsto e^{-x}sin(x)$. 20 training points are uniformly sampled, observed with Gaussian noise $\mathcal{N}(0, 0.05)$



(b) $\mathbf{[d = 5]}$ The learned predictor achieves low empirical risk on both the training and unseen samples.



(c) $\mathbf{[d = 2]}$ *underfits* the ground-truth function, resulting in a high empirical risk (illustrated with L1 norm as the loss function)



(d) $\mathbf{[d = 20]}$ *overfits* to the statistical noise in the training data and fails at generalizing to new unseen samples drawn from the same distribution.

Figure 1.1: Illustration of the underfitting and overfitting phenomena. Given noisy observations of the ground-truth damped sinus function (a), a predictor is learned from the family of polynomial functions of degree d, for different values of d. Under– (c) or overparametrizing (d) the hypothesis space results in under– and overfitting scenarios.

Model selection As we have seen with GD and SGD, optimization algorithms are often dependent on a set of *hyperparameters*, h, which are parameters of the optimization procedure itself. For instance, choosing a higher learning rate means browsing the search space faster, but also increases the risk of overshooting a minimal solution.

Similarly, a large batch size yields a more accurate estimate of the empirical risk, but significantly slows down each parameter update. These hyperparameters often incur a trade-off between training speed and quality of the output function $f_{\hat{\theta}}$ as a solution of the empirical risk minimization problem.

A natural way to choose optimal values for the hyperparameters is by a simple exhaustive search: *Model selection* consists in training the model with different configurations of hyperparameters, evaluating each of the resulting learned hypothesis, and finally selecting the best model (hence, set of hyperparameters) as the one that yields the best candidate, under the evaluation metric. Furthermore, as we noted earlier, evaluating a trained model requires an external testing set to assess its ability to generalize, rather than just how well it fits the training data. However, if we directly used the test set D_{test} for selecting optimal hyperparameters, this would introduce an undesirable bias between the training procedure and the test samples, hence lead to unfair evaluation. Instead, we introduce a *validation dataset*, D_{val}, once again sampled from the same distribution $\mathcal{D}$ as the training data, but independently from both D and D_{test}. Intuitively, D_{val} can be seen as a small subset of the training dataset which is left out of the optimization procedure, and is only used for selecting the best set of hyperparameters for training the model.

Stopping criterion. As we have mentioned before, gradient descent-based optimization on non-convex functions, such as neural networks, do not provide strong convergence guarantees. Furthermore, achieving a low empirical risk on the training set can be a symptom of overfitting. Thus, waiting until the optimization procedure converges to a low empirical risk is often not an optimal stopping criterion. Instead, D_{val} is often used to define the stopping criterion of the optimization algorithm: This *early stopping* procedure consists in regularly evaluating the model on D_{val} during training, for instance by evaluating the empirical risk on the validation dataset. At first, the risk should decrease, as the model captures information about the true distribution $\mathcal{D}$. Once it starts to increase, however, it is usually a sign that the model is overfitting to D instead of learning a general trend, hence training should be stopped.

The usual workflow of training a machine learning model, from optimization to evaluation and including model selection, is summarized in Figure 1.2.

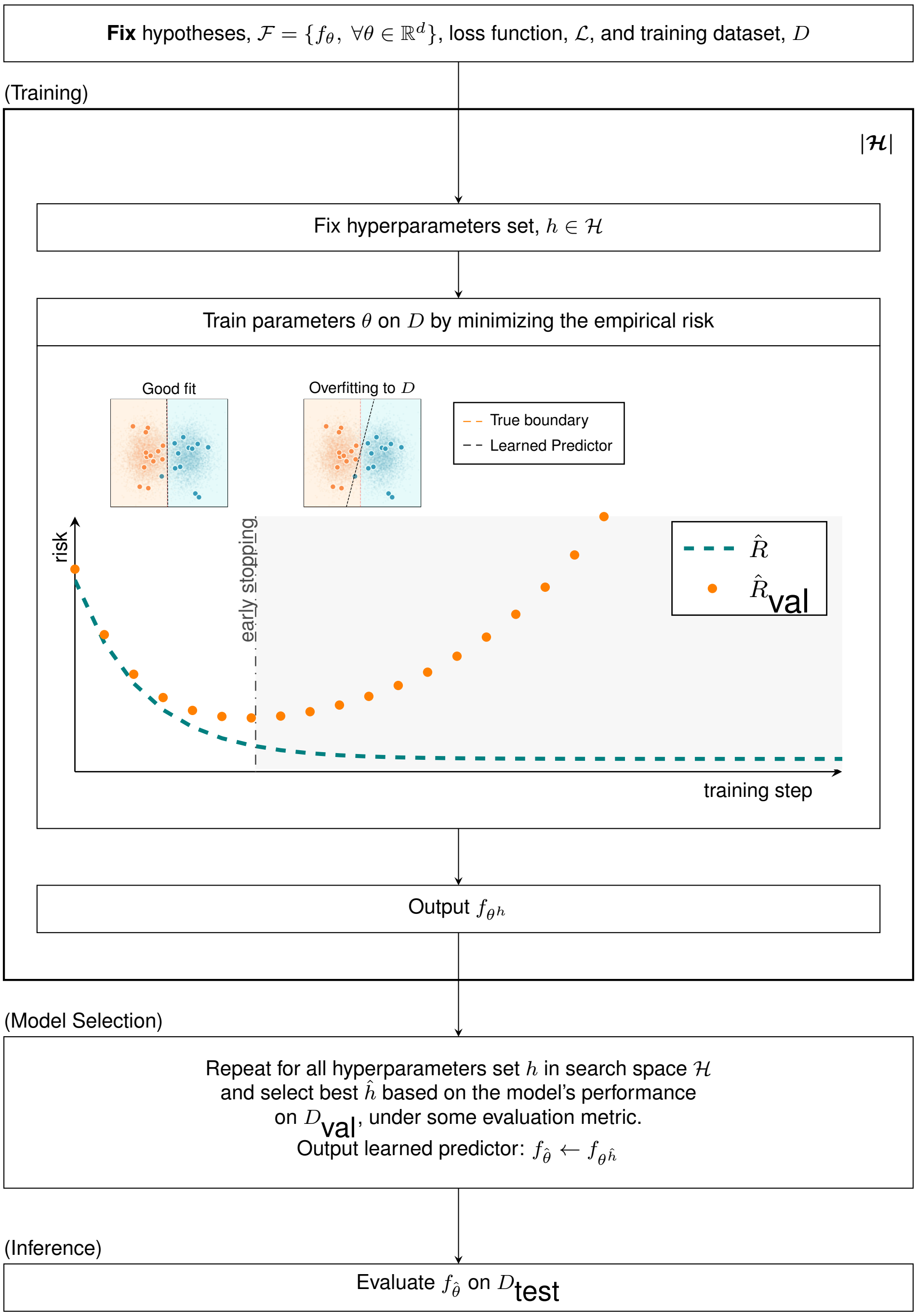


Figure 1.2: A machine learning model is defined by a set of parametric hypotheses, $\mathcal{F}$, a loss function $\mathcal{L}$ to evaluate how far a prediction is from the ground-truth, and an optimization algorithm (*e.g.*, Mini-batch Stochastic Gradient Descent). Given a training dataset D, one trains the hypothesis's parameters by minimizing the empirical risk $\hat{R}$, *i.e.* the average loss value on the training dataset (or on the current training mini-batch). To avoid overfitting, it is common to use an early stopping criterion based on an external, independent, validation set, D_{val}. The model's *hyperparameters*, *i.e.* any variable outside of θ that influences the training outcome, are also *selected* based on performance on the validation set. The final trained model is then evaluated on an independent test set D_{test} to assess how well it generalizes to new inputs drawn from the same distribution as the training samples.

1.1.3 Example: Binary classification

To conclude this section, we briefly review the previous notions in the context of binary classification. The training dataset D consists of a collection of images, each annotated as one of two classes, either positive ($+1$) or negative (-1). For instance, the task could be to predict whether the picture of a dish looks appetizing or not, based on previous user ratings. For simplicity, we consider pre-extracted feature embeddings of images as inputs. Thus, let $\mathcal{X} = \mathbb{R}^d$ denote the input space, where d is the dimension of the embeddings, and let $\mathcal{Y} = \{-1; +1\}$ denote the two classes we are interested in.

We restrict ourselves to gradient descent-based optimization algorithms, as these are by far the most commonly used in practice. Given a vector embedding of images, linear functions, $\mathcal{F} = \{f_\theta : x \mapsto \text{sign}\left(\theta^T x\right), \theta \in \mathbb{R}^d\}$, are a popular choice of hypotheses as they have a simple parametric form, and provide an intuitive geometric interpretation: The vector θ induces an hyperplane that splits $\mathbb{R}^d$ into two pieces, such that points are classified as either negative or positive depending on which side of the hyperplane they lie on. Finally, a natural loss function for binary classification is the 0-1 *loss*, which takes value 0 if and only if $f_\theta(x)$ is equal to the ground-truth label y of x. In summary, the model is specified as:

Binary Classification

$$f^* : x \in \mathcal{X} \mapsto y \in \mathcal{Y} = \{0, 1\}$$

$$f_\theta : x \mapsto [\![\theta^T x > 0]\!]$$

$$\mathcal{L}(f_\theta(x), y) = [\![f_\theta(x) \neq y]\!]$$

Where $[\![\cdot]\!]$ is the indicator function, which takes value 1 if its Boolean argument evaluates to `true`, and 0 otherwise.

Unfortunately, the resulting empirical risk is non-differentiable with respect to the model parameters θ, because of the indicator functions both in the loss function and in f_θ's definition, thus we cannot directly apply SGD for minimizing the empirical risk.

A differentiable relaxation. A common relaxation of the problem is for the model to not directly output a discrete label $f_\theta(x) \in \mathcal{Y}$, but rather a *continuous probabilistic estimate* of $p(+1|x)$, by mapping its output through the (differentiable) sigmoid function $\sigma : z \mapsto \frac{1}{1+e^{-z}} \in [0, 1]$ instead of the indicator function. In other words, we have that

$f_\theta(x) = p_\theta(+1|x) = \sigma(\theta^T x)$, and consequently $p_\theta(-1|x) = 1 - f_\theta(x)$.

With probabilistic outputs, a natural training objective is to maximize the likelihood of the training data in D under the distribution output by the model, *i.e.* $\prod_{(x,y)\in D} p_\theta(y|x)$. Equivalently, we can minimize the negative log-likelihood, which yields the following:

$$\begin{aligned} -\log\left(p_\theta(y|x)\right) &= -\begin{cases} \log\left(\sigma(\theta^T x)\right), & \text{if } y = +1 \\ \log\left(1 - \sigma(\theta^T x)\right), & \text{otherwise} \end{cases} \\ &\propto (1+y)\log\left(1 + e^{-\theta^T x}\right) + (1-y)\log\left(e^{\theta^T x} + 1\right) \end{aligned} \tag{1.6}$$

where (1.6), commonly referred to as the *log-loss* or *binary cross-entropy loss*, is indeed differentiable with respect to the parameters θ. The following model can thus now be optimized with a gradient descent-based algorithm, for instance SGD.

Binary Classification (relaxed)

$$f^* : x \in \mathcal{X} \mapsto y \in \{0, 1\}$$
$$f_\theta : x \mapsto \sigma(\theta^T x)$$
$$\mathcal{L}(f_\theta(x), y) = -(1+y)\log\left(f_\theta(x)\right) - (1-y)\log\left(1 - f_\theta(x)\right)$$

Model evaluation. Usually, the loss function used to train the model is also a good fit as an evaluation metric, since they both aim to compare the model's outputs to expected ground-truth values. For instance, the 0-1 loss naturally extends to the *classification accuracy metric* that amounts to counting the number of correct predictions:

$$\text{acc}(f_{\hat{\theta}}, D_{\text{test}}) = \frac{1}{|D_{\text{test}}|} \sum_{(x,y)\in D_{\text{test}}} [\![f_{\hat{\theta}}(x) = y]\!]$$

Similarly, in the relaxed model, evaluating the log-likelihood of the model would be a possible metric. However, this quantity is harder to interpret numerically as it expresses how much confidence the model puts in the correct label. Instead, we simply extend the definition of accuracy for the relaxed scenario as:

$$\text{acc}_{\text{relaxed}}(f_{\hat{\theta}}, D_{\text{test}}) = \frac{1}{|D_{\text{test}}|} \sum_{(x,y)\in D_{\text{test}}} \left[\!\!\left[y \log\left(\frac{f_{\hat{\theta}}(x)}{1 - f_{\hat{\theta}}(x)}\right) > 0 \right]\!\!\right]$$

Note that the model behaves differently at *training* time, where outputs lie in $[0, 1]$, and at *inference* time where the outputs are passed through a *decision function*, here $z \mapsto \log \frac{z}{1-z}$, that maps to the original output space $\mathcal{Y} = \{-1, 1\}$. This often happens in machine learning models relying on differentiable relaxations, taking advantage from the fact that evaluation metrics need not be differentiable.

1.2 Neural Networks

Deep Learning refers to machine learning models in which the hypothesis space is defined as a family of *neural networks*. In this section, we will first briefly review the definition of neural networks, in particular in the context of computer vision.

1.2.1 Formal definition

Definition. A fully-connected neural network is essentially a concatenation of simple linear functions alternated with non-linearities. Formally, let $L \in \mathbb{N}^*$ and d be a vector of $L + 1$ positive integers; We define $\mathcal{F}_{\boldsymbol{d}}^L$ the family of L-layered fully-connected neural networks in the following recursive manner, as illustrated in Figure 1.3:

Fully-connected neural networks (def)

$$\begin{aligned} &\mathcal{F}_{\boldsymbol{d}}^0 \triangleq \{id : x \in \mathbb{R}^{d_0} \mapsto x\} \\ \forall \ell,\ &\mathcal{F}_{\boldsymbol{d}}^\ell \triangleq \{f_\theta : x \in \mathbb{R}^{d_0} \mapsto \sigma\left(\theta^\ell\ f_{\theta^{\ell-1}}(x)\right) \in \mathbb{R}^{d_\ell}, \\ &\qquad\qquad \forall \theta^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, \forall f_{\theta^{\ell-1}} \in \mathcal{F}_{\boldsymbol{d}_{0\dots\ell-1}}^{\ell-1}, \forall \sigma : \mathbb{R}^{d_\ell} \rightarrow \mathbb{R}^{d_\ell}\} \end{aligned}$$

We say that L is the number of layers, or the depth, of the network, where the ℓ-th layer of the network designates the function $z \mapsto \sigma\left(\theta^\ell z\right)$, or, by abuse of notation, its output. d_ℓ is the *dimension* or *number of channels* of the ℓ-th layer; with the two special cases of d_0 and d_L being the input and output dimension respectively.

The individual components in each layer are called *neurons*: Each *neuron* $f_{\theta^\ell}(x)_i$ in layer ℓ is connected to all the neurons in the previous layer, with weights defined by θ^ℓ, hence the name of *fully-connected neural network*. Finally, σ is an *activation function*, whose role is to introduce non-linearities in the network. Typical examples include the sigmoid function, or the Rectified Linear Unit (ReLU) function, ReLU $: x \mapsto \max(0, x)$.

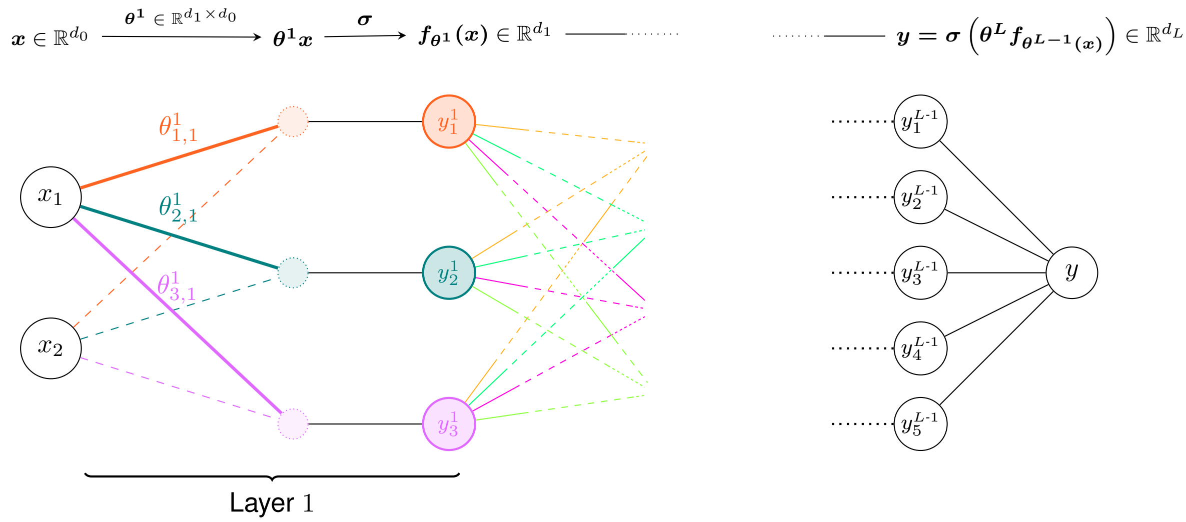


Figure 1.3: A fully-connected neural network with inputs $x \in \mathbb{R}^{d_0=2}$ and outputs $y \in \mathbb{R}^{d_L=1}$. Each layer of the network is a function f_{θ^ℓ} that consists of a matrix multiplication by θ^ℓ, mapping inputs in $\mathbb{R}^{d_{\ell-1}}$ to $\mathbb{R}^{d_\ell}$, followed by a non-linear activation function, σ. In total, L Layers are concatenated together, forming a *deep* neural network.

Expressivity of neural networks. The main strength of neural networks lies in that they combine a simple, easy to implement, parametric form with a high expressive power: A first key factor to this is the activation function, σ. Without it, a neural network would reduce to a simple linear transformation $\theta^\ell\theta^{\ell-1}\dots\theta^1 x$. A second factor is the size of the neural network, characterized by its *depth*, L, and its *width*, *i.e.* the vector $\boldsymbol{d}$ containing the number of channels in each layer. For instance, the *Perceptron* [Rosenblatt, 1958], which is equivalent to a one-layer neural network, cannot capture the simple XOR function, irregardless of its width. In contrast, a network with at least two layers (also called a *Multilayer Perceptron*) easily can. More generally, the *universal approximation theorem* [Cybenko, 1989; Hornik *et al.*, 1989] states that any continuous function can be closely approximated by a 2-layers neural network with appropriately chosen width. Similar results also exist for the orthogonal scenario, *i.e.* for neural networks with bounded width and unbounded depth [Lu *et al.*, 2017].

Nevertheless, the interplay of the width and depth of a neural network, and how it impacts both the training process and the final learned model's performance, is not a well understood phenomenon [He *et al.*, 2016; Zagoruyko and Komodakis, 2016]. In the current state-of-the-art, neural networks usually consist of *several* concatenated layers, each with finite width, hence the name of *deep* learning. Furthermore, in practice, choosing an adequate depth and width is not only a matter of expressivity, but also of computational speed, as both of these parameters increase the number of operations in the model.

1.2.2 Convolutional neural networks

So far, we have only discussed neural networks taking vector inputs. However, these are ill-suited for images which are inherently two-dimensional, and exhibit locality constraints: For instance, neighboring pixels are more likely to belong to the same object than distant ones. To capture this spatial information, *convolutional neural networks* [Fukushima *et al.*, 1988; LeCun *et al.*, 1999] extend the fully-connected networks formalism by replacing the usual linear operation with a two-dimensional convolution operation to form *convolutional layers*.

Convolutional layer. Formally, let $x \in \mathbb{R}^{w \times h}$ be a two-dimensional input, let $\theta \in \mathbb{R}^{k_w \times k_h}$ be the parameters, or *weights* of the convolution, and s_w and s_h be integers in the intervals $\{1, \ldots, w\}$ and $\{1, \ldots, h\}$ respectively. We define the 2D convolution operator $\odot$ with kernel θ and stride $s = (s_w, s_h)$ as:

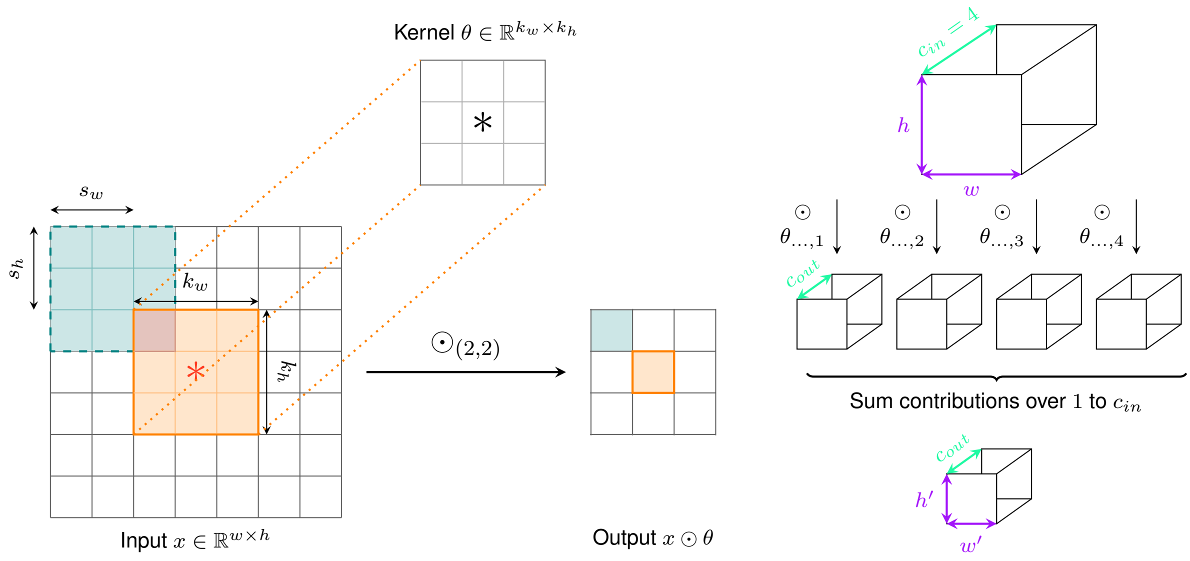
$$x \odot_s \theta \in \mathbb{R}^{(\lfloor (w-k_w)/s_w \rfloor + 1) \times (\lfloor (h-k_h)/s_h \rfloor + 1)} \tag{1.7}$$

$$\forall i \in \{1, \ldots, \lfloor (w - k_w)/s_w \rfloor + 1\}, \forall j \in \{1, \ldots, \lfloor (h - k_h)/s_h \rfloor + 1\},$$

$$(x \odot_s \theta)_{i,j} \triangleq \sum_{p=1}^{k_w} \sum_{q=1}^{k_h} x_{i \times s_w + p, j \times s_h + q} * \theta_{k_w - p + 1, k_h - q + 1} \tag{1.8}$$

In other words, the operator performs element-wise multiplications between the kernel θ and submatrices of x, in a sliding window manner, where the stride s defines the distance between each window. An illustration of this definition is given in Figure 1.4(a).

In practice, however, images are usually 3D arrays that contain two spatial dimensions and one *channel* dimension. For instance, an input RGB image has three channels, representing the red, green and blue color information. Extending (1.8) to multi-channel inputs and outputs is done by considering a different convolution kernel matrix for each input and output channel, and applying the 2D convolution operator with each one of them independently. The contributions of the different input channels are then summed, such that the output once again has only two spatial dimensions and one channel dimension. See also Figure 1.4(b) for an example.



(a) 2D convolution operation example on a 7×7 input with a kernel of size 3×3 and a stride of 2 in both the horizontal and vertical direction. The resulting output is a 2D matrix of size 3×3.

(b) Multi-channel convolution for c_{in} input channels and c_{out} output channels.

Figure 1.4: Illustration of the 2D convolution operation (left) and its extension to multi-channel inputs and outputs (right).

Based on these definitions, we can now introduce the family of 2D convolutional neural networks (*CNN*) $\mathcal{F}^L_{\boldsymbol{d},\boldsymbol{k}}$. Once again, L denotes the number of layers and $\boldsymbol{d}$ is the number of channels in every layer. Finally, $\boldsymbol{k}$ denotes the kernel size of the convolution operator in each layer respectively. Formally,

Convolutional neural networks (def)

$$\begin{aligned}
&\mathcal{F}^0_{\boldsymbol{d},\boldsymbol{k}} \triangleq \{id : x \in \mathbb{R}^{w \times h \times d_0} \mapsto x\} \\
\forall \ell,\ &\mathcal{F}^\ell_{\boldsymbol{d},\boldsymbol{k}} \triangleq \Bigg\{f_\theta : x \mapsto \sigma\left(\sum_{c_{in}=1}^{d_{\ell-1}} f_{\theta^{\ell-1}}(x) \odot \theta^\ell_{c_{out},c_{in}}\right)_{c_{out}\in\{1,\dots,d_\ell\}} \\
&\qquad \forall \theta^\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1} \times k^\ell_w \times k^\ell_h}, \forall f_{\theta^{\ell-1}} \in \mathcal{F}^{\ell-1}_{(\boldsymbol{d},\boldsymbol{k})_{0\dots\ell-1}}, \forall \sigma \text{ activation function}\Bigg\}
\end{aligned}$$

As previously, the function $z \mapsto \sigma(z \odot \theta^\ell)$ is called the ℓ-th (convolutional) layer of the network. It consists in a 2D convolution operation with weight matrix θ, followed by an element-wise activation function. In particular, if using only scalar kernels ($k_w = k_h = 1$), then a convolutional layer is equivalent to applying the same fully-connected linear layer at every spatial position of the input.

Spatial dimensions and field of view. By definition, the output of every convolutional layer is a 3D *feature map* which preserves spatial information, in the sense that a neuron at spatial position (i, j) can be directly linked to the input spatial positions involved in its computation. The *receptive field* of a neural network is the number of input pixels which are involved in the computations of any neuron in the last layer. This is an important parameter as it essentially represents “the field of view” of the network and the range of interactions across pixels it can capture. The receptive field can be increased by using larger kernel matrices or a deeper architecture. The latter option is usually preferred as, *(i)* in practice, depth is beneficial to the model's performance and *(ii)* working with smaller kernel matrices is more computationally efficient.

Nevertheless, while capturing spatial coherence is important for image understanding, it is not always necessary to explicitly represent every single pixel in the input image. For instance, the task of classification only requires to output information at the global image-level. Taking this fact into account, CNNs usually incorporate additional *pooling layers*, which aim to downscale the spatial resolution of the feature maps, essentially getting rid of redundant spatial information, and allowing us to obtain a more compact representation. Furthermore, this reduces the number of convolution operations to apply in subsequent layer, hence improves computational efficiency. In practice, pooling is implemented either as a convolution operation with a stride greater than 1 (as shown in Figure 1.4(a)), or with specific lightweight pooling layers such as max-pooling or average-pooling, which consist in taking small groups of neighboring pixels and replacing them with a unique value, *e.g*. their maximum or average.

CNNs as feature extractors. As summarized in Figure 1.5, a convolutional network is usually built as a long sequence of convolutional layers, with decreasing spatial dimensions via pooling operations. These layers form the *feature extraction* part of the network. Typically, if the output does not require spatial information (for instance, an image-level classification task), then the extracted features are flattened to form a compact one-dimensional vector embedding of the image. Otherwise, if the network preserves spatial dimensions, the architecture is said to be *fully-convolutional*. The resulting embedding is then fed to a shallow *decoding* stage which maps the learned features to the output space, and consists in either convolutional or fully-connected layers, depending on whether the embedding is two- or one-dimensional.

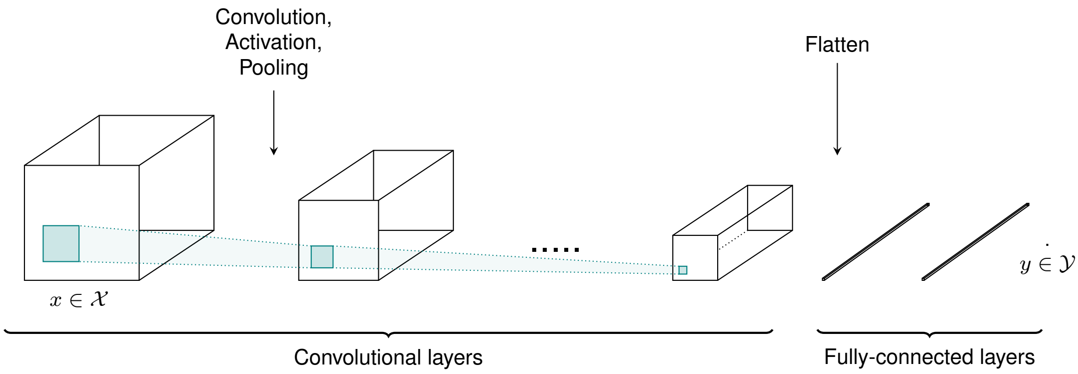


Figure 1.5: Typical example of a convolutional neural network (CNN) mapping an RGB image to a one-dimensional output (*e.g.*, in the context of a classification task). The network consists in a sequence of convolutional layers, forming a *feature extraction pipeline* in which the field of view (here, represented in teal) increases with the depth. If the task output does not require spatial information, as it is the case here, the resulting feature map is then flattened and fed to a short sequence of fully-connected layers, which forms a shallow decoding stage mapping to the output space $\mathcal{Y}$.

1.2.3 Optimization of neural networks

Granted that we choose a differentiable activation function, which is generally the case in practice, a neural network $f_\theta \in \mathcal{F}^L_{\boldsymbol{d},\boldsymbol{k}}$ is also differentiable with respect to its parameters $\theta = \left(\theta^i \in \mathbb{R}^{d_i \times d_{i-1} \times k_w^i \times k_h^i}\right)_{i=1 \dots L}$, which makes this hypothesis class well suited for gradient descent-based optimization, such as SGD.

Gradient computation. Given a differentiable loss function $\mathcal{L}$, the gradient of the loss for sample (x, y), $\nabla_\theta \mathcal{L}(f_\theta(x), y)$, can be computed efficiently using the *backpropagation* algorithm [Rumelhart *et al.*, 1986], which is directly derived from the chain rule of derivation. In summary, each training iteration requires a *forward pass*, to compute the empirical risk on the given inputs, and a *backward pass*, which consists in re-traversing the network from the last to first layer, to derive the gradient at the current point. Furthermore, most deep learning frameworks [Abadi *et al.*, 2015; Paszke *et al.*, 2019] only require the forward pass to be implemented, from which they directly deduce an implementation for the backward pass via *automatic differentiation*.

Nevertheless, this means that every parameter update in the optimization algorithm requires two traversals of the whole network for each training sample in the current mini-batch, which can be very costly for deep and/or wide networks. To improve computational efficiency, modern software usually make use of *(i)* designated hardware, such as GPUs, that can efficiently perform matrix multiplications, one of the core operations in neural networks, and *(ii)* vectorization and parallelization, capitalizing on the fact that training iterations for different input samples can be executed in parallel, and then gathered together to compute the empirical risk on the current mini-batch.

Optimization landscape. Despite the lack of theoretical convergence guarantees of SGD for non-convex functions, optimizing the empirical risk of neural networks usually yields excellent solutions in practice. Even though neural networks are often overparametrized, they also rarely overfit, and trained networks often reach low risk on the training set, while also generalizing well to new unseen test inputs. While this phenomenon is not well understood, it is actively studied: For instance [Zou *et al.*, 2019; Ma *et al.*, 2018] analyse the fast convergence rate of SGD for overparametrized neural networks, and [Kawaguchi, 2016; Choromanska *et al.*, 2015] inspect the quality of local minima found by gradient decent-based optimization of neural networks.

Hyperparameters. One drawback of neural networks compared to other function families is that they depend on numerous hyperparameters, with very little theoretical insights on how to tune them. For instance, choosing the *architecture* of the network requires selecting an appropriate value for its depth, width, activation functions, additional normalization operations, etc. Clearly, performing an exhaustive search for all possible combinations would be extremely cumbersome.

Instead, it is common to use a combination of "folk knowledge", by relying on existing architectures whose performance have been demonstrated, and of model section on an external validation dataset, focusing on a few hyperparameters, typically the one whose changes the model is least robust to. In particular, appropriately setting the learning rate has a significant influence on whether a deep learning model converges at all, which solution it converges to, and how fast. While the original SGD algorithm uses a fixed learning rate throughout training, various learning rate *decay* routines have shown great success when training deep learning models. A well-known example is the *ADAM* optimizer [Kingma and Ba, 2015], which only requires setting an initial learning rate, and automatically adapts the decay parameters based on the magnitude of the latest gradient steps.

1.3 A Lack of Structured Representations

One of the key strengths of deep neural networks is their surprising ability to generalize well to new unseen data, as long as it follows the same distribution $\mathcal{D}$ as the one seen during training. However, outside of this scenario, they exhibit several pitfalls. Perhaps one of the most well-known examples is the case of adversarial samples [Szegedy *et al.*, 2014], which are very small, imperceptible to the human eye, local changes of an image, that significantly alter the response of a neural network. This example suggests that, despite their success at solving some tasks as well (or better) than humans [Geirhos *et al.*, 2018], neural networks are inherently *black boxes*. This is coupled with a lack of theoretical understanding which makes it hard to provide any strong convergence or generalization guarantee for deep neural networks.

Thus, understanding and designing neural networks that have better generalization properties is a key challenge of deep learning. In this thesis, we argue in favor of constraining neural networks to learn more structured, organized, representations, with the aim to better adapt or generalize to unseen scenarios. In particular, we will focus on *(i)* how compositional task structure can be reflected in feature representations for more efficient models, and *(ii)* how the layered structure of neural networks impact learned representations across different domains. In this section, we review related literature and introduce important notions for understanding the remainder of this manuscript.

1.3.1 Compositionality

Compositionality of representations. Compositionality is a key property of human reasoning. For instance if one understands the concept of “green” and that of “mouse”, one can easily imagine a “green mouse”, even without having seen it before. In particular, representing the world in a compositional manner has several benefits including *interpretablity*, as complex concepts can be expressed as compositions of simpler ones, and *generalization*, since it allows us to fantasize unseen concepts by combination. In the machine learning literature, *compositional learning* encompasses scenarios where the output space is described as the composition of two smaller spaces, $\mathcal{Y} = \mathcal{Y}_1 \times \mathcal{Y}_2$, often, but not necessarily, independent from one another. Typical examples include learning interactions between semantic attributes (*e.g.* color, shape, parts) and object categories [Misra *et al.*, 2017; Nagarajan and Grauman, 2018; Lampert *et al.*, 2014; Reed *et al.*, 2015]: Semantic attributes can be seen as operators that change the visual appearance of object categories. Similarly, affordance discovery [Kato *et al.*, 2018; Chao *et al.*, 2018; Peyre *et al.*, 2019] can be seen as the task of modeling how human actions impact the visual appearance of the world.

A key difficulty of learning adequate compositional representations lies in the reasoning aspect of the task: Without adequate constraints on their representations, neural networks tend to easily be overconfident on the training distribution and simply ignore unseen compositions. A second difficulty is the model's ability to automatically learn independent concepts, due to existing biases in the input visual domain [Sadeghi and Farhadi, 2011]; For instance, *melted cheese* occurs in different contexts – and is visually different – than *melted chocolate*, which makes learning a unique concept for the operator "melted" particularly difficult. Even more so, because no supervision is available for the different independent concepts, but only for their compositions, which amounts to learning *disentangled* representations in a semi-supervised setting [Higgins *et al.*, 2017; Locatello *et al.*, 2019; Bouchacourt *et al.*, 2018].

Trade-off with model efficiency. While compositionality has direct links with generalization, it also has important implications for building more efficient models, in terms of computational, memory and data efficiency. In fact, compositionality can also be seen as the problem of decomposing a task into smaller, often easier, sub-goals. In practice, building modular architectures has potential benefits for the computational and memory efficiency of the model, as it involves combining and reusing smaller lightweight networks rather than building a one-block complex model. Furthermore, once the modules are trained, they can be freely combined, and thus they can potentially be applied to even more complex problems as the ones seen at training time.

A particular example of modularity are recurrent architectures, which consist in iteratively applying a small module to solve a more complex tasks. While these are often used in domains that have an inherent recurrent structure, *e.g.* for text data or reinforcement learning, they are also used in some computer vision tasks. For instance, *cascade ensembles* for object detection [Viola and Jones, 2001; Viola and Jones, 2004] iteratively refine the predicted object detections through a sequence of classifiers, each iteration being trained to accept the current input if it contains an object of interest, otherwise rejecting it. Therefore if an image region is empty, it is likely to be rejected very early in the pipeline, thereby saving up on further computations.

A second benefit of modularity is data-efficiency. A compositional model should be able to learn a composition operator and independent representations of the different concepts from *limited data*, as it sees only a few possible compositions. And in fact, compositional learning models are usually evaluated on how well they generalize to new unseen compositions of known concepts, from which they have seen no samples. This setting is a particular instance of *few-* or *zero-shot learning* which consists in generalizing to unseen concepts given only a few, or zero, samples. In fact, modular architectures have been recently applied to the zero-shot learning framework [Nagarajan and Grauman, 2018; Purushwalkam *et al.*, 2019; Alfassy *et al.*, 2019].

1.3.2 Transferring deep learned representations

As we mentioned in Section 1.2.2, the representations learned by a CNN are organized in a layered architecture, such that the receptive field increases with the depth, while spatial dimensions decrease. Since the first successes of deep learning, several works have aimed to understand the nature of the captured representations, and in particular how they *transfer* across different input and output spaces. Below, we describe the fine-tuning and domain adaptation tasks, two instances of *transfer learning* that are well-studied in the machine learning literature, and in particular for computer vision.

Fine-tuning. A well-known practical result [Yosinski *et al.*, 2014] is that early layers of a trained neural network often transfer well for feature extraction on different datasets. This is the core assumption of *fine-tuning*, which is a common technique consisting in training a neural network on a training set S, sampled from some *source* distribution $\mathcal{D}_S$ over the input-output domain $\mathcal{X} \times \mathcal{Y}$, then freezing the early layers of the architecture, and re-training (tuning) only the shallow decoding stage for another *target* distribution, $\mathcal{D}_T$. We illustrate this procedure in Figure 1.6. This strategy yields surprisingly good results, and allows the user to *(i)* leverage pre-extracted knowledge from the potentially much larger dataset S and transfer it to the target task, and *(ii)* to only train a few layers, which is much faster than training the whole architecture from scratch.

Motivated by these results, several public frameworks [TensorNets; TensorFlow Hub; Model Zoo] now provide implementations and weights of CNNs which have been pre-trained on very large image datasets, such as ImageNet [Russakovsky *et al.*, 2015]. This also contributed to making fine-tuning an extremely popular and inexpensive transfer learning strategy in practice.

In parallel, another line of work focuses on understanding *why* these features transfer so well. Several studies [Zeiler and Fergus, 2014; Mahendran and Vedaldi, 2015; Olah *et al.*, 2017; Cadena *et al.*, 2018] propose to analyze the information captured by the different layers via various reverse engineering and neural feature visualization techniques. A common observation, which holds for different architectures and computer vision tasks, is that early convolutional layers tend to capture local pixel-level information, such as edges, specific texture or basic patterns, which are likely to be important feature for any image datasets. As depth grows, later layers instead respond to more complex patterns, such as parts, or specific objects, which are often more task-specific. While this is of course a very informal characterization, it does suggest that the layered architecture of standard neural networks induces a natural organization of their representations, which can be similar across different architectures and tasks.

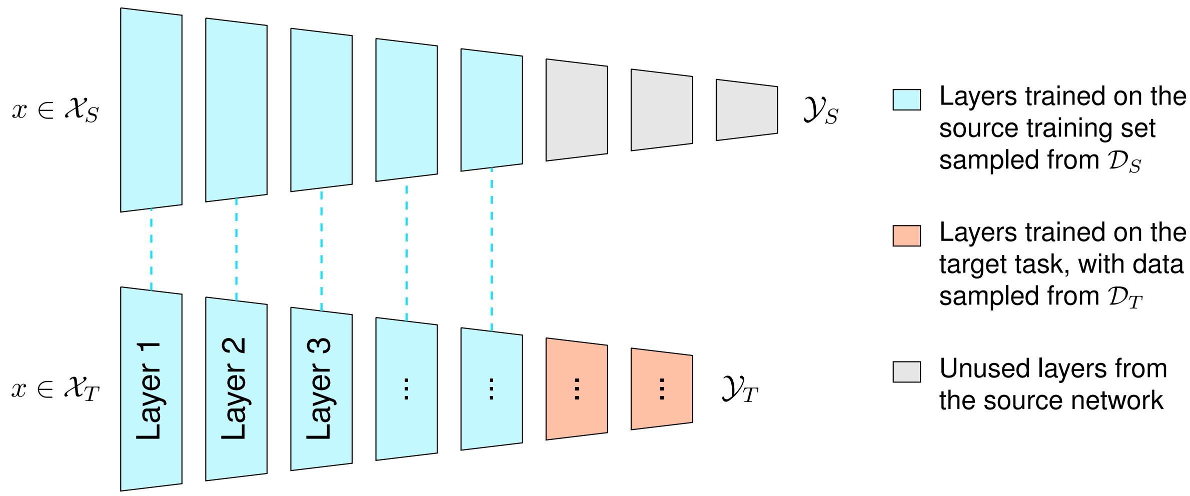


Figure 1.6: Fine-tuning consists in using a feature extraction pipeline, pre-trained on a source domain $\mathcal{X}_s \times \mathcal{Y}_s$, and transfer them to a new task on a target domain $\mathcal{X}_t \times \mathcal{Y}_T$. When training for the target task, the pre-trained layers are either kept frozen or fine-tuned jointly with the new layers, depending on available resources. Similarly, the new layers are either initialized from scratch, at random, or from the pre-trained source network's weights, if their architectures match.

Domain Adaptation. Fine-tuning is extremely successful when transferring across semantically related tasks on similar visual inputs, *i.e.* $p_{\mathcal{D}_S}(x) \simeq p_{\mathcal{D}_T}(x)$. This is the case for instance when transferring knowledge from a pre-trained image classifier to the task of object detection [Redmon *et al.*, 2016] or semantic segmentation [Long *et al.*, 2015] on natural images. However, the outcome of transferring representations is much less clear when there is a more important *domain shift*, *i.e.* when the source and target distribution differ more significantly. For instance, [Crowley and Zisserman, 2014] show that features learned from natural images transfer well to art painting datasets, while in the case of sketches, it is more beneficial to train from scratch [Yu *et al.*, 2015]. Similarly, [Raghu *et al.*, 2019] observe that transferring natural image features to medical images offers little benefits compared to random initialization.

These scenarios are typical examples of *domain adaptation* problems, and are symptomatic of real-life applications where there is a lack of data, either in terms of number of samples or annotations. In contrast, annotated natural images are abundant in the real world. Thus being able to transfer knowledge learned from these large, fully annotated, datasets to data-scarce applications is a highly desirable goal.

More specifically, *unsupervised domain adaptation* is the task of learning a model that performs well on both the source and target distribution, given labeled samples from $\mathcal{D}_S$ and unlabeled ones from $\mathcal{D}_T$. Most domain adaptation techniques aim to learn a joint aligned embedding of the source and target input domains, from which the subsequent task can be solved, thus heavily relying on the notion of shared features. In fact, most domain adaptation methods assume a unique, shared, neural network to embed both the source and target domain, thus making the implicit assumption that the model should focus on extracting features which are common to both domains. Nevertheless, several recent work [Tzeng *et al.*, 2017; Rozantsev *et al.*, 2018; Rozantsev *et al.*, 2019; Bousmalis *et al.*, 2016] investigate more flexible parameter sharing frameworks, and suggest that it is instead more beneficial to explicitly model both information shared across the two domains as well as "private" information specific to each domain. In fact, how layered representations transfer across domains is not well understood, and is particularly hard to analyze due to *(i)* the black-box nature of deep neural networks and *(ii)* the difficulty to meaningfully quantify the domain shift between the two underlying unknown distributions $\mathcal{D}_S$ and $\mathcal{D}_T$.

1.4 Thesis Outline

In this thesis, we focus on improving deep convolutional neural networks' abilities to adapt to unseen inputs, in particular in scenarios where the task's structure itself can be exploited:

- In **Chapter 2**, we tackle the problem of abstract visual reasoning, as a specific instance of compositional learning [Royer and Lampert, 2020c]. The ability to infer abstract rules from visual information and to apply them to new situations is one of the cornerstones of human intelligence. Despite recent advances in visual reasoning, computer vision systems still fail at this goal. In this chapter, we study the foundations of this problem, which we show does not lie in specialized architectures, but rather in under-constrained latent representations. We propose three formal criteria that, in combination, constrain the model to construct and operate on intermediate representations which reflect the abstract nature of the task in order to improve generalization. We then perform experiments on two novel benchmarks and propose quantitative metrics to assess to what extent these criteria are met in practice with current representation learning techniques.

- In **Chapter 3**, we investigate two instances where modular architectures leads to improved model efficiency. First, we tackle the problem of image colorization by leveraging recent progresses on autoregressive image generative models [Royer *et al.*, 2017]. We show that this modular approach enables tractable computation of the likelihood. The resulting model, trained on the true data likelihood, better captures the distribution of chrominance in color images, compared to standard feed-forward architectures. Secondly, we study the task of object detection for low computational and memory resource scenarios [Royer and Lampert, 2020b], such as for instance when working with drones. We show that a modular architectures based on the notion of groups of objects, allows us to obtain a better trade-off between model accuracy and computational efficiency, when dealing with sparsely distributed objects, such as for instance in aerial imagery.

- Finally, in **Chapter 4**, we study how the layered organization of neural networks' representations impacts feature sharing. First, we consider the problem of transfer learning. We propose a flexible weights tuning scheme, which aims to efficiently estimate the individual impact of the different layers with respect to the source-target domain shift, in order to select the best layer to tune [Royer and Lampert, 2020a]. Furthermore, we study the performance of the proposed selection criteria over several different visual shifts, of increasing difficulty, as well as different data scarcity scenarios. Secondly, we tackle the task of unsupervised image-to-image translation [Royer *et al.*, 2018]. We show that explicitly modeling both shared information and private information specific to each domain is key to the model's performance, in particular when the visual shift between the source and target input domains is significant.

2 Exploiting Compositional Structure for Generalization

In this chapter, we focus on the compositional learning task. We will show that, without proper constraints on their representations, neural networks struggle to generalize well in these scenarios and instead tend to overfit to held-out compositions. In particular, we will focus on the task of *abstract visual reasoning* (AVR), that, in contrast to other compositional learning tasks, suffers little from visual biases, which allows for a clearer interpretation of experimental results. Furthermore, the ability to map visual signals to abstract concepts without explicit supervision is a major component of natural intelligence for visual reasoning. Such representations allow humans to perform abstract reasoning, *e.g*. to apply logical rules or test numerical properties, across a wide and diverse range of objects and visual properties, even previously unseen ones. Consequently, it is of key interest for artificial intelligence research to build systems with the ability to represent and reason on visual inputs at an abstract level.

Unfortunately, current artificial intelligence systems are unable to autonomously discover and reason about abstract concepts from general visual inputs without strong supervision. This includes in particular deep neural networks: Despite their well-established proficiency in various computer vision tasks, several works have questioned whether this reflects an ability to learn high-level "intelligent" concepts [Chollet, 2019; Hudson and Manning, 2019]. For instance, while neural networks set a new standard for visual question answering (VQA), a more careful analysis showed that little high-level reasoning took place, and that instead the network often exploited data biases, such as strong correlations between questions and answers [Jabri *et al.*, 2016; Chattopadhyay *et al.*, 2017; Johnson *et al.*, 2017; Wang *et al.*, 2018a]. Besides VQA, standard convolutional networks have also been shown to struggle at seemingly simple high-level reasoning tasks, such as distinguishing between "same-different" shapes [J.K Kim and Serre, 2018] or predicting 2D coordinate transforms of simple shapes [Liu *et al.*, 2018].

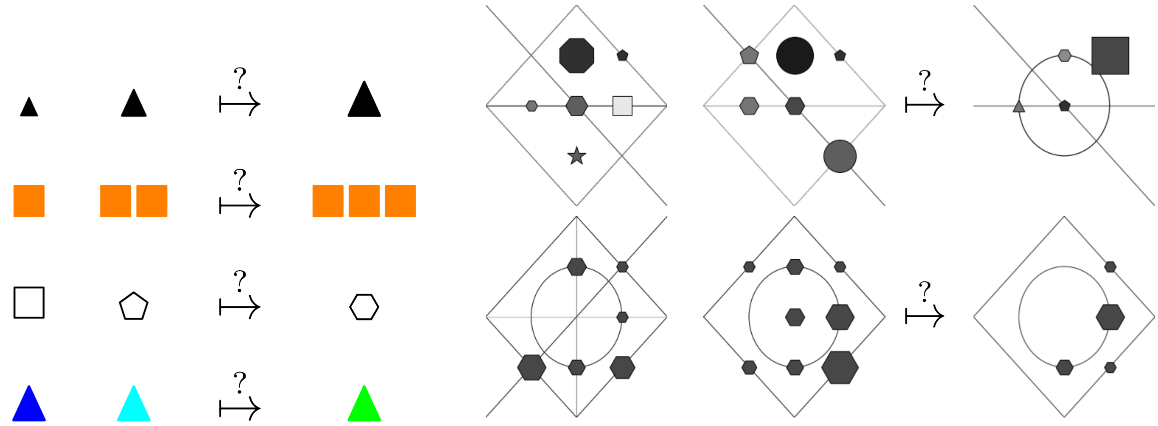


Figure 2.1: We tackle the task of *Abstract Visual Reasoning* (AVR), which requires inferring an abstract rule (on the example: the answer for the left figure is *increase*, and for the right right, *set intersection*), and the visual property it applies on (respectively, on the left: *size*, *number*, *shape* and *hue*, and right: *object positions* and *line type*); while the remaining properties are either kept constant (left example) or are distractors that vary randomly (right eaxample). As shown on the figure, there is usually little to no local visual similarity between triplets following the same rule. Instead, the task requires high-level reasoning to *(i)* identify the different visual properties of interest and *(ii)* model how rules operate in the abstract space.

Recent work [Hoshen and Werman, 2017; Santoro *et al.*, 2018; Zhang *et al.*, 2019a; van Steenkiste *et al.*, 2019; Zhang *et al.*, 2019b] has tackled the task of figural matrices completion as a benchmark to quantify the abstract visual reasoning (AVR) abilities of deep neural networks. Inspired from classical IQ tests for natural intelligence [Carpenter *et al.*, 1990; Raven, 2003] such as Raven matrices, as illustrated in Figure 2.1 (right), these problems pose a challenge even to humans as they require extracting visual properties of interest in the presence of distractors, and recognizing common relations across diverse visual modalities. Despite recent improvements, current baselines often ignore a key component of AVR, which is the ability to learn abstract rules that *generalize across diverse visual properties*. In fact, when these methods are evaluated on their ability to generalize to unseen rule-property combinations, they display a significant decrease in performance [Santoro *et al.*, 2018; Steenbrugge *et al.*, 2018].

In this chapter, we propose to take an in-depth look at *why* existing AVR methods fail, and *what* is needed to overcome it. As a benchmark, we consider the problem of rationale learning: Given a sequence of images in which a specific visual property is controlled by a rule, the task consists in predicting this rule and the visual property it acts on, as illustrated in Figure 2.1. We put an emphasis on generalization by evaluating performance on unseen *combinations* of known rules and properties, which requires abstract rather than low-level reasoning. Our contributions are three-fold:

- First, we introduce the task of *rationale learning* with two benchmarks of varying difficulty. We show that existing architectures fail to solve the general task, but succeed when provided with adequate supervision. This shows that, in principle, deep neural networks are powerful enough for abstract reasoning tasks.

- Second, we introduce three criteria, two of which are architectural, while the third one describes an invariance property of the learned data representation. We show that, in combination, they guarantee the ability to solve abstract visual reasoning tasks.

- Third, we report experimental results suggesting that no current representation learning methods is able to reliably enforce the invariance criterion during training, thus it remains the missing piece for AVR.

2.1 Related Work

Abstract reasoning has a long tradition in artificial intelligence research where it is typically studied based on symbolic or very restrictive visual inputs [McGreggor and Goel, 2014; Shegheva and Goel, 2018]. Recently, it has attracted interest also in the computer vision and deep learning community, in particular through the task of Raven figural matrices completion [Raven, 2003]. Starting from a simple convolutional neural network baseline [Hoshen and Werman, 2017], current state-of-the art methods have focused on structured architectures including relation networks [Hill *et al.*, 2019; Santoro *et al.*, 2018; van Steenkiste *et al.*, 2019; Steenbrugge *et al.*, 2018], Dynamic Residual Trees [Zhang *et al.*, 2019a] or explicit contrastive modules [Zhang *et al.*, 2019b]. However, and most of the works indeed acknowledge this, the performance of all these methods decreases significantly when evaluated for generalization on missing relations. This suggests that they do not actually perform abstract, but rather low-level reasoning, as truly abstract rules would be applicable to new visual properties, even if their combination has never been seen before.

Similar problems have been observed in other research areas that also require strong forms of generalization, in particular attribute-based zero-shot learning [Lampert *et al.*, 2014; Palatucci *et al.*, 2009]. Methods in this area aim at identifying abstract semantic properties (attributes) that carry over from seen to unseen categories. In practice, however, this transfer is only partially successful, and attribute classifiers tend to be biased towards correlated low-level features [Gan *et al.*, 2016; Jayaraman and Grauman, 2014]. We are therefore hopeful that progress in the field of abstract reasoning would also trigger new progress in zero-shot learning.

For a long time it has also been questioned how well image-based reasoning tests actually correlate with true reasoning abilities or "intelligence". For instance, [J.K Kim and Serre, 2018] proposed a new benchmark which highlighted the fact that standard convolutional networks failed at understanding simple visual relations and require explicit structure, such as relational networks or explicit coordinates modeling [Liu *et al.*, 2018], to handle the combinatorial number of objects templates. The same holds for the perceptual grouping problem [Kim *et al.*, 2020]. Similarly, [Saxton *et al.*, 2019] analyze the performance of deep network on mathematic problems. More recently, [Chollet, 2019] draws a parallel with natural intelligence to establish desirable properties of artificial intelligence systems, in particular with respect to their generalization abilities, and proposes interesting leads to evaluate them.

2.2 Abstract Visual Reasoning (AVR)

In this section, we formalize the notion of AVR, encompassing recent benchmarks [Santoro *et al.*, 2018; Zhang *et al.*, 2019a; Teney *et al.*, 2020], and introduce the exemplary task of rationale learning. We then show there is a clear failure mode of existing AVR techniques on generalized rationale learning, which is however not symptomatic of a problem in architecture as performance can be recovered with adequate supervision.

2.2.1 Setting definition

An *abstract visual reasoning universe*, $(\mathcal{P}, \mathcal{V}, \mathcal{R}, \psi)$ consists of four parts: Two finite sets, $\mathcal{P}$ (the visual *properties*) and $\mathcal{V}$, (the *abstract values*), a set $\mathcal{R}$ (*rules*) of functions mapping a sequence of elements of V to V, and finally, the *rendering operator*, ψ, grounding abstract values for a specific visual property into a visual signal.

More intuitively, the properties, $\mathcal{P}$, are grounded visual concepts, such as *"the color of an object"* or its *"number of corners"*. The value space, $\mathcal{V}$, is a finite set of values that acts as an abstraction layer for realizations of the properties. For example, the number of corners of an object could be a number between 1 and 4. The rules, $\mathcal{R}$, are functions of arbitrary arity that act on a sequence of elements of V and output an element of V value again. Examples are logical rules that act on boolean values, such as $r(v_1, v_2) = v_1 \wedge v_2$, or arithmetic rules acting on numeric values, such as $r(v) = v + 1$.

The rendering operator, ψ, is a stochastic function that maps an *assignment* of values to properties, represented as tuple $\boldsymbol{v} = (v_p)_{p \in \mathcal{P}}$, to a probability distribution over images that we can sample from. Its probabilistic nature allows for a broad range of appearance variations including nuisance factors, such as noise. Finally, properties, values and rules interact by the notions of *compliance* and *rationales*:

Definition 2.2.1. *Let $r \in \mathcal{R}$ be a rule with arity k and $p \in \mathcal{P}$ be a property. A sequence of assignments, $V = (\boldsymbol{v}^1, \ldots, \boldsymbol{v}^n)$,* ***complies with the rule-property pair*** *(r, p), if*

$$r(v_p^{i-k}, \ldots, v_p^{i-1}) = v_p^i \qquad \text{for } i = k+1, \ldots, n. \tag{2.1}$$

In that case, we call (r, p) a ***rationale*** *that underlies the sequence V.*

2.2.2 Rationale Learning as a benchmark for AVR

Recent work [Hoshen and Werman, 2017; Santoro *et al.*, 2018; Zhang *et al.*, 2019a; Zhang *et al.*, 2019b] has studied abstract visual reasoning using the *candidate prediction* task as a benchmark, inspired by human IQ tests: Given an input sequence $\boldsymbol{x} = (x^1, \ldots, x^{n-1})$, and candidate images $c_1, \ldots, c_k$ the goal is to pick c^* such that the complete sequence $(\boldsymbol{x}, c^*)$ complies with the largest number of rationales. Several architectures to model complex relations between images have been proposed and shown to achieve good results in a standard supervised prediction setting.

However, as our experiments in Section 2.6 will show, these architectures do not exploit the abstract nature of the task, but instead make use of low-level statistical properties of the input visual signal. As a first contribution of this work, we aim to diagnose and understand this failure mode. While candidate prediction is a plausible task for benchmarking abstract visual reasoning, it also has some shortcomings: In particular, as observed in [Hill *et al.*, 2019], performance on the candidate prediction task heavily depends on how the wrong candidate images are generated. As an alternative more objective benchmark for analysis, we introduce the task of *rationale learning*:

Definition 2.2.2. *Let $(\mathcal{P}, \mathcal{V}, \mathcal{R}, \psi)$ be an abstract visual reasoning universe. The goal of **rationale learning** is to build a model that, for any given image sequence underlied by a unique rationale, predicts which rationale it complies with. It is essentially the counterpart to Definition 2.2.1.*

To do so, only a training set of tuples $(\boldsymbol{x}_i; (r_i, p_i))_{i=1,\dots,N}$ is available, where each x_i is an image sequence and (r_i, p_i) is the rationale it was generated from, following Definition 2.2.1, or, equivalently, it is the only rationale in $\mathcal{R} \times \mathcal{P}$ that x_i complies with.

In particular, rationale learning is at least as powerful as candidate prediction, because it yields a natural method for scoring a candidate c by simply counting how many rationales the model detects on the sequence $(\boldsymbol{x}, c)$.

Solving *rationale learning* in a standard supervised setting is however not an indicator of abstract visual reasoning. In fact, by treating each rationale as an independent class it could, *e.g.* be solved by a standard multi-class CNN, provided the training set is rich enough. Following [Santoro *et al.*, 2018], we therefore adopt a challenging extrapolation scenario, in which the model is evaluated on rationales that were not part of the training set. Let $\mathcal{Y}_{\text{heldout}} \subsetneq \mathcal{R} \times \mathcal{P}$ be a *held out* subset of rationales, and $\mathcal{Y}_{\text{train}} = \mathcal{R} \times \mathcal{P} \setminus \mathcal{Y}_{\text{heldout}}$ the set of remaining rationales. We generate training samples, X_{train} from $\mathcal{X} \times \mathcal{Y}_{\text{train}}$, and evaluate the model performance on test samples, X_{test}, generated from $\mathcal{X} \times \mathcal{Y}_{\text{heldout}}$. To make sure that the task remains feasible, we construct the split such that for any $(r, p) \in \mathcal{Y}_{\text{heldout}}$, its rule r appears at least once in $\mathcal{Y}_{\text{train}}$ (though in combination with a different property), and analogously its property p appears in $\mathcal{Y}_{\text{train}}$ (though combined with a different rule).

The above construction guarantees that every visual property and rule is seen at least once during training time. Consequently, the data contains enough information to infer how visual properties align with the abstract value space and how rules operate on it. If a method fails to infer a rationale $(r, p) \in \mathcal{Y}_{\text{heldout}}$, this can likely be attributed

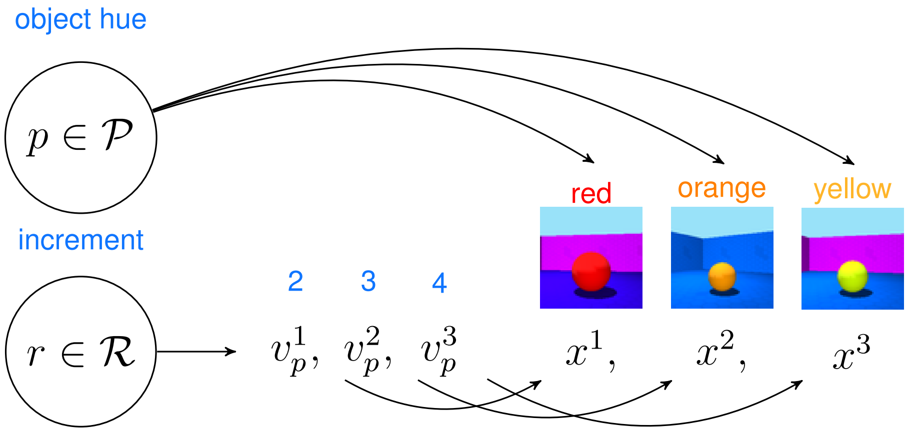


Figure 2.2: Generative model for an abstract visual reasoning universe, here with a sequence of $n = 3$ images controlled by incrementing (rule r) the object's hue (property p). For clarity, the value of other properties are not shown.

to an incapacity to generalize to the novel rationale combination rather than lack of information about individual rules or properties.

2.2.3 A generative process for rationale learning

The notion of rationales, together with the rendering operator, induces a **generative probabilistic model** of image sequences, in which rules control specific visual properties at the abstract level and the resulting sequences of assignments are rendered in the image domain. Figure 2.2 summarizes the high-level process.

Formally, we can build a sequence of n images from a given rationale with property p and $k-$ary rule r as follows: Let q be a distribution over the abstract value space (typically, uniform). We first sample context values $v_p^1, \ldots, v_p^k \overset{iid}{\sim} q$, and then generate the remaining ones as $v_p^i = r(v_p^{i-k} \ldots v_p^{i-1})$ for $i = k+1, \ldots, n$. Thus, the sequence complies with (r, p) by construction. For the remaining properties $p' \neq p$, we either sample independently from q, in which cases they act as *distractors* not controlled by any rule, or we consider a simpler "no distractors" setting where they are kept constant across the sequence *i.e.* $v_{p'}^1 \sim q$ and $v_{p'}^1 = \cdots = v_{p'}^n$. We then map these assignments to a sequence of images by passing each assignment through the rendering operator.

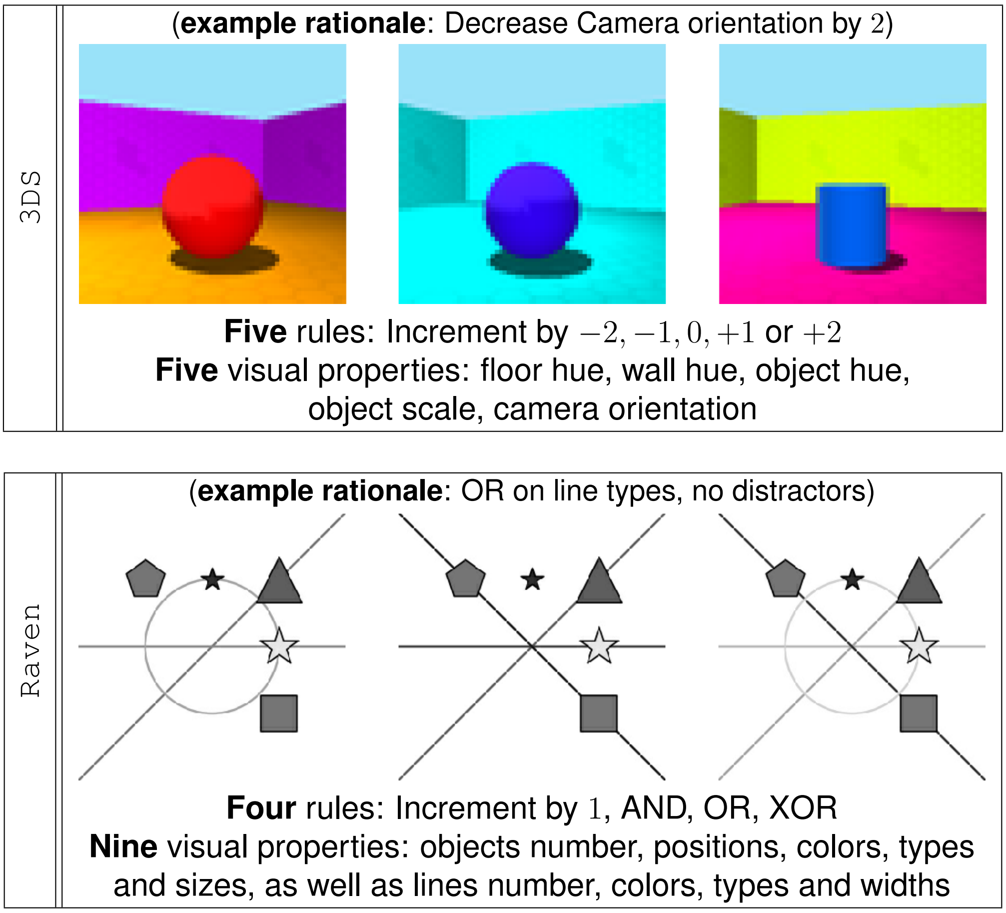


Figure 2.3: Characteristics of the two proposed *rationale learning* benchmarks, with varying difficulty in terms of input visual complexity and number of abstract rules.

2.3 Dataset Generation

Following Definition 2.2.2, we generate two benchmarks for the task of rationale learning, which are illustrated in Figure 2.3. The first one is based on the 3D-Shapes dataset [Burgess and Kim, 2018] and contains five simple numerical rules acting on five different visual properties. The rendering operator produces simple 3D scenes with a central object where the object's shape is left to the randomness in the renderer. The second dataset is inspired by the PGM dataset [Santoro *et al.*, 2018] and the classical Raven matrices [Raven, 2003] used in human IQ tests. While visually simpler, this benchmark is much more challenging at the reasoning level (even for humans) as it considers nine very different visual properties and four rules with different arities. To allow for easier analysis, we only use sequences that comply each with a single rationale, such that every inference task will have a unique correct answer.

2.3.1 Implementation

In order to generate a benchmark for rationale learning, we first need to simulate the abstract visual reasoning generative process described in Figure 2.2. The main components of the implementation are as follows:

- **Property, Rule and Images:** A rule is defined as a function of any arity that acts on the space of abstract values (typically, $\mathbb{N}$). Each image in the sequence is defined by a collection of visual properties, where each visual property is characterized by its current assigned value (in the abstract value space). Finally, a rendering operator maps this set of assignments to a RGB image.
- **Rationale:** A rationale (rule r, property p) is implemented as a function that takes as input a sequence of context images and returns an image in which the value assigned to property p complies to rule r with respect to the context sequence. Additionally, the function raises an exception in any of these cases:
  - Applying the rationale would lead to an ambiguous situation. For instance, if values are Boolean and $v_p^1 = v_p^2 = 1$ then applying rule "AND" or "OR" would both results to $v_p^3 = 1$.
  - Applying the rationale would lead to an invalid assignment. For instance if the value of property p is upper bounded and applying the rule would require setting v_p outside of its bounds.
- **Verifier**: Given a sequence of images, the verifier outputs the list of rationales the sequence complies to.

Based on these components, the generation process for one sequence of length n is based on rejection sampling and goes as follows:

- **[Step 1]** First, sample rationale (r, p). Let k denote its arity.
- **[Step 2]** For all properties p' that are different from p *and* not determined by p: decide whether they will be *distractors* (i.e., sampled at random for each image) or *non-distracting* (kept fixed across the sequence): For instance, if figural elements in the image cannot overlap, knowing the positions of elements fully determines

their number. Hence having a rule on positions prevents us from choosing the number of objects as a distractor sampled randomly.

- **[Step 3]** Sample the first k images by sampling values for each property independently at random (taking into account whether each property is distracting or not according to step 2).
- **[Step 4]** Create the $(k+1)$-th image by copying all non-distracting entities from the k-th image, and then generating value from property p such that it complies to the rule r, *i.e.* $v_p^{k+1} = r(v_p^1, \ldots, v_p^k)$. For all remaining properties, sample their assigned values at random.
- **[Step 5]** Repeat **step 4** until the sequence has n images.
- **[Step 6]** Break any *specious rationales* by running the verifier on the sequence:
  - If it detects a rationale (r', p) for any $r' \neq r$ then raise an exception.
  - If it detects a rationale (r', p') for any $r' \in \mathcal{R}$ and any $p' \neq p$, then resample the value assigned to p' in the last image until it does not comply to any rule. This can be easily done by sampling while excluding values $\{r(v_{p'}^{n-k}, \ldots, v_{p'}^{n-1}), \forall r \in \mathcal{R}\}$. If no such sampling exist, raise an exception.
- **[Step 7]** Render every image independently using the rendering operator and the obtained "visual property to abstract value" assignments.
- **[Final]** If at any point an exception is raised, simply *reject* the sample and restart from **step 1**.

In order to define new benchmarks of varying difficulty we simply vary the rules, visual properties, probability of a property to be a distractor, and the rendering operator. Finally, the same process could be applied for sequences complying to multiple rationales, however we only describe the setting with one rationale for simplicity, as this is the one we tackle in the main manuscript.

First benchmark: 3DS. We use the 3D shapes dataset [Burgess and Kim, 2018] as our first visual domain. Visual properties are defined as: Wall's hue, floor's hue, central object's hue, central object's scale, camera orientation. The abstract value space is the range of natural integers $\{1, \ldots, 15\}$ (15 being the number of possible camera orientations, which is the property with the highest number of discrete values). An image is hence defined by these five visual properties and their assigned values. Applying the rendering operator simply consists in looking in the dataset for an image that verifies the corresponding assignment (in that case, the shape of the object being sampled at random). Finally, there are five rules, each of arity 1 and of the form $r : v \in \mathcal{V} \mapsto v + \text{inc} \in \mathcal{V}$ where inc $\in \{-2, -1, 0, 1, 2\}$. In particular, because the identity (inc = 0) is one of the rule, visual properties are always taken as distractors in **step 2**, as to avoid ambiguity with this rule.

In total, we use 100k samples for training, 10k for validation and 50k for testing. Visual examples of the dataset are given in Figure 2.5. Finally, we also consider three distinct held-out sets of rationales for experimental validation.

Second benchmark: `Raven`. The `Raven` benchmark is inspired from standard Raven matrices and, more specifically, from the PGM dataset [Santoro *et al.*, 2018] for figural matrices completion. The nine visual properties are: objects' number, objects' positions, objects' colors, objects' scales, objects' types, lines' number, lines' colors, lines' widths and lines types. The abstract value space is the range of natural integers $\{1, \ldots, 10\}$ (for instance, colors are grayscale intensities ranging from black to white with ten thresholds). Here the rendering operator is based on simple `matplotlib` scatter plots. Furthermore, the rendering is deterministic, in the sense that knowing the assignment of all visual properties perfectly determines the result image.

There are four rules $\{$ Progression, XOR, OR, AND $\}$. Note that these rules are applied at the global image level, over the set of elements. Hence we can interpret them as set operations (respectively: increment cardinality, symmetric difference, union and intersection): For instance, if the first panel contains two white, one black and three gray objects and if the second panel contains one white and five gray objects, then the corresponding sets of values are $\{$white, black, gray$\}$ and $\{$white, gray$\}$. Hence applying the rule AND to the color property would yield a panel that only contains white and gray objects. Finally, due to the definition of the setting, some rationales are undefined. For instance, applying AND on the objects' number does not have any reasonable interpretation. In total, only 28 out of 36 rationales are permitted.

We consider two sub-settings: One where no properties is distracting (no distractors), and one where any property has a fifty percent chance to be sampled as distracting in **step 2**. As for the 3DS example, we use 100k for training, 10k for validation and 50k for testing. Visual examples of the dataset are given in Figure 2.6. As for the previous benchmark, we also consider three different held-out settings in our experiments.

2.4 Can Deep Networks Solve AVR?

In this section, we define three representation learning criteria, and show that, taken together, they allow the model to perform abstract reasoning, yet prevent it from taking shortcuts through trivial low-level reasoning. We then discuss how these criteria relate to existing representation learning techniques, and introduce quantitative metrics that allow us to assess how far any real-world model is from satisfying them.

2.4.1 Formal criteria

In line with the image generation process of Section 2.2.1, we argue that the problem of abstract visual reasoning is two-faceted. The first facet is one of perception, as we need to extract relevant information from a complex, possibly noisy, visual signal. The second facet is one of reasoning, where one combines the information extracted from multiple images into a final decision.

Therefore, the models we consider reflect the same dichotomy in their architecture: They consist of an *encoder* component, $f : \mathcal{X} \to \mathcal{Z}$, that maps images to a low-dimensional representation space, $\mathcal{Z}$, followed by a *reasoning* module, $\phi : \mathcal{Z}^n \to \mathbb{R}^{\mathcal{R} \times \mathcal{P}}$, that takes as input the representations of all available images and outputs scores for each rationales, as a standard multi-class classification model. Equipped with these notations, we introduce three representation learning criteria. The first one ensures that the model is able to identify and isolate the various visual properties of interest:

Criterion 1. *A model is called* ***systematic*** *if its representation space, $\mathcal{Z}$, and its encoder, $f : \mathcal{X} \to \mathcal{Z}$ decompose according to the visual properties of the task,* i.e.

$$\mathcal{Z} = \prod_{p \in \mathcal{P}} \mathcal{Z}_p \quad \text{and } f(x) = \big(f_p(x)\big)_{p \in \mathcal{P}} \tag{2.2}$$

where $f_p : \mathcal{X} \to \mathcal{Z}_p \subset \mathbb{R}^{d_p}$ for some dimensionality d_p.

The second criterion states that the reasoning module should treat each visual property in the same way, thereby reflecting the fact that rules are functions only of the abstract values, not of the properties from which they were derived:

Criterion 2. *For any rationale (r, p), let $s_{r,p}(\boldsymbol{x})$ denote the scores that the model outputs for input $\boldsymbol{x} = (x^1 \dots x^n)$. Then the model is called **consistent** if*

$$\forall p \in \mathcal{P} : \quad s_{r,p}(\boldsymbol{x}) = \phi\big(f_p(x^1), \dots, f_p(x^n)\big), \tag{2.3}$$

In words, the reasoning module is identical for all properties $p \in \mathcal{P}$.

Finally, the third criterion is an invariance property that reflects that all properties share a common abstract value space on which rules operate:

Criterion 3. *For any rationale (r, p), let $\boldsymbol{X}^{r,p}$ be the random variable associated to image sequences that complies with the rationale (r, p).*

Let $Z^{r,p}$ be the variable associated to their latent codes for the property of interest, i.e. *$[f_p(x^1) \dots f_p(x^n)]$ where $\boldsymbol{x} = (x^1, \dots, x^n)$ is an observation from $\boldsymbol{X}^{r,p}$.*

*The model is called **oblivious** if for any fixed $r \in \mathcal{R}$, the distributions of $Z^{r,p}$ are identical for all $p \in \mathcal{P}$.*

An illustration of the the three criteria and how they help constraining the model's learned representations can be found in Figure 2.4.

2.4.2 A proof of sufficiency

First, we note that making the model systematic is a simple architecture design choice which allows us to reason across different visual properties, and without which the other criteria cannot be defined. In particular, it does not restrict the encoder's representational power as any model that uses a feature representation can trivially be made systematic by simply duplicating the latent representation as often as there are properties.

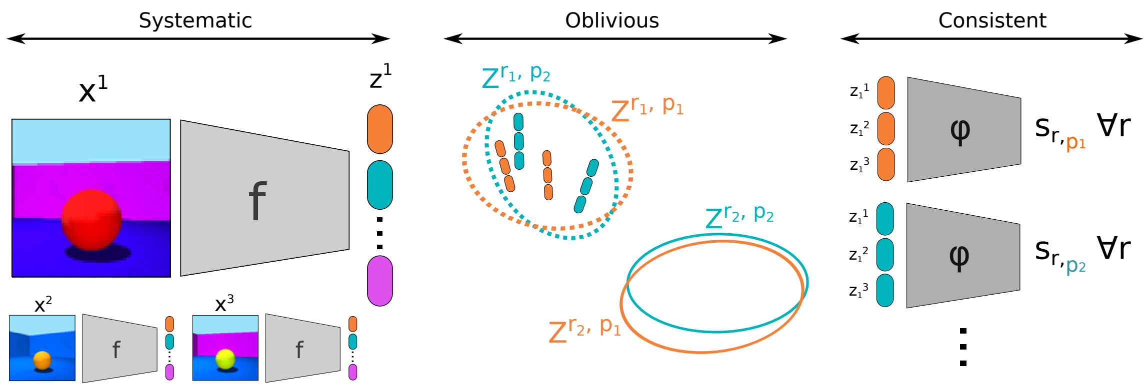


Figure 2.4: Our three proposed criteria require the model to be *systematic*, by enforcing the encoder to distinguish across extracted visual properties, *oblivious* to visual property identities, by aligning distributions in the latent space, and finally, *consistent*, in the way it infers abstract rules using the same module across all properties.

More interestingly, enforcing the model to be both consistent and oblivious introduces a trade-off between the performance on the training and held out distribution, which essentially prevents it from trivially isolating the missing rationales. The high-level proof goes as follows: Let (r, p) be one of the held-out rationale. By construction, there exists a property $p' \neq p$ such that examples for the rationale (r, p') are in the training set. Let $\boldsymbol{x}^{r,p}$ and $\boldsymbol{x}^{r,p'}$ be two random image sequences with underlying rationale (r, p) and (r, p') respectively. Because the model is oblivious, the corresponding latent representations, $z^{r,p}$ and $z^{r,p'}$ are identically distributed. Moreover, because the model is consistent, the reasoning module acts identically across visual properties, so the scores output by the model, $s_{r,p}(\boldsymbol{x}^{r,p})$ and $s_{r,p'}(\boldsymbol{x}^{r,p'})$, are also identically distributed. Consequently, if the model achieves a small expected loss for rationale (r, p') during training, it will also achieve a small expected loss on the held-out rationale (r, p) at prediction time. Therefore, this means that the model is unable to trivially memorize which specific property-rule combinations are present in the training set nor overfit to specific visual artifacts without hurting its performance on the training data as well.

Additionally, we observe that each of these two criteria loses meaning if taken individually: In fact, if the model is oblivious but not consistent, the reasoning module is unconstrained, thus it can easily learn to ignore specific inputs, for instance by setting all weights of the final classification layer for a missing rationale to 0, without hurting performance for the remaining rationales. Conversely, if a model is consistent but not oblivious, the encoder can leak information about the specific visual property, which the reasoning module can then exploit: For instance, if $f_p : x \mapsto [v_p; p]$, then even a consistent cognitive module can learn to always outputs 0 for a certain rationale (r, p), because it knows the identity of the property it acts on.

In summary, it is only in combination that the two criteria guarantee that the cognitive module cannot trivially overfit to low-level cues, neither from the values of the latent representations nor from their position in the embedding.

2.5 Practical Implementation and Evaluation

In this section, we propose quantitative evaluation metrics to assess how well each criterion is met, and we then discuss how to enforce these criteria in practice, based on state-of-the-art representation learning techniques.

Evaluation. Whether a model is systematic (Criterion 1) and consistent (Criterion 2) are direct statements about its internal structure. Therefore, we can easily check whether these are either fulfilled or not, once we know the model's architecture.

Obliviousness (Criterion 3), on the other hand, is a statement about the model's learned function, and requires that different parts of its internal representation, treated as random variables, are identically distributed. Unfortunately, it is not possible to verify or refute this fact for a real-world model from finite samples. Therefore, in this section, we introduce a quantitative version of this criterion, which can be estimated from finite amounts of data. It thereby provides us with a computable way of determining how far a practical model is from being oblivious.

First, we observe that the condition of identical distributions in Criterion 3 is equivalent to the statement "$Z^{r,P} \perp\!\!\!\perp P$", where P is the random variable associated to the latent position in the embedding, $p \in \mathcal{P}$. This can in turn be phrased in terms of *mutual information*. In fact, a model is oblivious if and only if

$$I[\, Z^{r,P}\, ;\, P\,] = 0 \quad \text{for all } r \in \mathcal{R}. \tag{2.4}$$

In general, estimating mutual information for continuous variables is non-trivial [Belghazi *et al.*, 2018; Moddemeijer, 1989]. However, P is a discrete variable, which makes (2.4) easier to estimate than if both arguments had been continuous. In practice, we use the mutual information estimate formulated in [Ross, 2014] which also relies on a nearest-neighbor-based estimator of the Shannon entropy [L. F. Kozachenko, 1987; Lombardi and Pant, 2015].

While Criteria 1 to 3 constrain the model's internal structure, they do not guarantee that the resulting network actually has high accuracy. For instance, a network with an encoder that maps all images to a single constant representation would be systematic, consistent and oblivious, but probably fail to predict meaningful rationales.

Therefore, we introduce two additional measures that reflect how relevant the learned embeddings are. First, the representation $f_p(x)$ should not depend on other visual distractors but should be fully determined by the underlying abstract value v_p, *i.e.*

$$H[f_p(x)|v_p] = 0 \tag{2.5}$$

where $H[\cdot|\cdot]$ denotes the conditional entropy. Furthermore, the representation should be informative about the values of the property they are meant to encode, *i.e.* the mutual information

$$I[f_p(x); v_{p'}] \tag{2.6}$$

should be high for $p = p'$ and as low as possible otherwise. We note that using mutual information as a metric has the disadvantage of not being comparable across datasets. Therefore, we also adopt the *mutual information gap* criterion as introduced in [Chen *et al.*, 2018], which computes a normalized relative difference between the highest and the second highest mutual information scores (2.6). As such, it takes a value of 1 if each property p is perfectly associated with the component $f_p(x)$ of the representation, and 0 in the case where every $f_p(x)$ is highly correlated with another value $v_{p' \neq p}$. All introduced quantities are again mixed continuous-discrete entropic metrics and thus can also be estimated with the method discussed above.

2.5.1 Relation to representation learning literature

In this section, we put the criteria defined above in context with existing literature on latent representation learning.

Disentanglement. By definition, a systematic model has an encoding mechanism that factorizes across visual properties. For simplicity, we propose to directly integrate this property in the architecture, as visualized in the left segment of Figure 2.4.

In that setting, properties (2.5) and (2.6) can be seen as a specific, constrained, form of disentanglement [Locatello *et al.*, 2019; Higgins *et al.*, 2017]. Unsupervised disentangled representation learning usually refers to the problem of learning a latent embedding that consists of multiple components that are independent of another, *i.e.* have zero mutual information. In fact, disentangled representations have recently been observed to help with visual reasoning tasks [van Steenkiste *et al.*, 2019] but do not solve the performance gap in the generalization setting [Steenbrugge *et al.*, 2018]. In contrast, systematicity only requires that each component accounts for a specific visual property. In other words, the latent components need not be independent with one another, but only independent conditioned on the visual property they encode.

Information bottleneck. The conditions (2.5) and (2.6) also resemble the *information bottleneck (IB) principle* for general representation learning [Tishby and Zaslavsky, 2015; Vera *et al.*, 2018]. Like IB, the goal is to learn a representation that captures the relevant information, here the values v_p, while being as ignorant as possible about the rest of the input signal x itself. However, in our setting, the quantity of interest, v_p, is not observed at training time, so one cannot simply directly optimize for the IB objective.

Consistent network architectures. Given a systematic model, we enforce the property of being consistent in the network architecture by applying the same reasoning module network to all visual properties, as shown in the right segment of Figure 2.4. Similar designs have been explored in *compositional learning* [Misra *et al.*, 2017; Nagarajan and Grauman, 2018], where visual attributes are sometimes modeled as linear modules that should act independently from the object categories they apply on.

Domain adaptation. A model is oblivious (Criterion 3) if for any rule $r \in \mathcal{R}$ the latent variables $(Z^{r,p})_{p \in \mathcal{P}}$ have the same distribution for all properties p. This resembles the problem of *domain adaptation*. There, in order to learn a predictor that works for different data domains, one learns intermediate representations that are identically distributed for all domains [Ben-David *et al.*, 2010; Ganin *et al.*, 2016; Schoenauer-Sebag

$\mathcal{P}$ \ $\mathcal{R}$	r_1	r_2	r_3
p_1	O	O	O
p_2	?	?	?

$\mathcal{P}$ \ $\mathcal{R}$	r_1	r_2	r_3
p_1	O	X	O
p_2	X	O	O

O: labeled
?: present, usually unlabeled
X: absent

Table 2.1: Domain adaptation (left) cannot be directly applied to enforce obliviousness as it requires very different training supervision than the held out rationales setting (right) which essentially has incomplete domains.

et al., 2019]. Doing so, however, usually requires at least unlabeled samples from all domains, *i.e.* in our case, for the different rationale. In contrast, the AVR setting has *incomplete domains* as we have fully labeled samples for the training set rationales and *no samples* for the remaining ones. The difference in settings is summarized in Table 2.1. Therefore, domain adaptation techniques are not directly applicable to enforce the desired obliviousness criterion.

To circumvent this issue, we use a *weaker* form of obliviousness, which we discuss further in the next section. It relies on enforcing identical distributions of the per-property embeddings $f_p(X)$ across *all images* rather than of the embeddings derived from *specific rule-properties pairs*, $Z^{r,p}$. Because every visual property is seen during training (although, each in combination with different rules), this situation now matches the standard domain adaptation setting, where we aim to align distributions from samples from different domains and we can now apply existing algorithms in practice.

2.5.2 Weak obliviousness

In this section, we discuss the link to domain adaptation in more details and formally establish the link between *obliviousness* and *weak obliviousness*.

Definition. Remember that obliviousness states that for any given relation r, the distribution of $Z^{r,p}$ is the same whatever p is. We can phrase this as a multi-source domain adaptation problem, where labeled sources corresponds to (r, p) tuples present in the training set and the target domains correspond the missing rationales. Because we cannot directly apply domain adaptation techniques in our setting, we consider a weaker but more practical criterion, which essentially forces the model to be oblivious of visual property identities at the single image level rather than at the sequence level:

Criterion 4. *Let Z^p be the variable associated to observations $f_p(x)$ for any image x. Then the model is **weakly oblivious** if the distributions of Z^p are identical for all $p \in \mathcal{P}$.*

This definition does not depend on rationales but only on visual properties. Since the set of missing rationales is by definition constructed such that every visual property is seen at least one during training, we recover a setting closer to the usual domain adaptation regime.

Intuitively, Criterion 4 is *weaker* than Criterion 3 because it does not guarantee that the learned embeddings will follow the same distribution, once the structure of the rule space comes into play. This can be shown on a simple example: Suppose we have sequence of images of length $n = 2$, where each image is defined by two visual properties p_1 and p_2, with underlying abstract value space $\mathcal{V} = \{0, 1, 2\}$, and let the rule $r : x \mapsto x + 1$. We now define the encoder $f(x) \mapsto [v_1, 2 - v_2]$. This model is *weakly oblivious* because each latent is uniformly distributed in $\mathcal{V}$ but not *oblivious* because sequences that complies under the relation (r, p_1) will be mapped through $f(\cdot)_1$ into the space $\{[0, 1], [1, 2]\}$, while those complying to (r, p_2) will be mapped through $f(\cdot)_2$ to $\{[2, 1], [1, 0]\}$, hence the distributions are different as they have different support.

As a result, the sufficiency guarantee presented in Section 2.4.2, obtained when combining obliviousness and consistency, would break in the scenario where only weak obliviousness is verified. In the next paragraph, we discuss the link between weak obliviousness and obliviousness in more details, to provide further insights in how they differ formally.

How “weak” is weak obliviousness ? For simplicity, we only consider sequences of length 2. We also denote the embeddings as $f_p(x) = z_p$ for concision. Given sequences of images $\boldsymbol{x}$ and $\boldsymbol{x}'$ following rationale (r, p) and (r, p') respectively, *obliviousness* requires that the variables associated to the corresponding embeddings (z_p^1, z_p^2) and $(z_p'^1, z_p'^2)$ have the same distribution.

Intuitively, the link between z_p^2 and z_p^1 depends on *(i)* the rule r applied to the underlying values and *(ii)* the information that z_p encodes about value v_p, which is measured by the informativeness metrics. Thus we can write:

$$
\begin{aligned}
\mathbb{P}(z_p^2,\ z_p^1) &= \sum_v \mathbb{P}(z_p^2,\ z_p^1 \mid v_p^1 = v)\mathbb{P}(v_p^1 = v) \\
&= \sum_v \mathbb{P}(z_p^2 \vert\ z_p^1, v_p^1 = v)\mathbb{P}(z_p^1 \mid v_p = v)\mathbb{P}(v_p^1 = v) \\
&= \sum_v \mathbb{P}(z_p^2 \mid z_p^1, v_p^2 = r(v))\ \mathbb{P}(z_p^1 \mid v_p^1 = v)\mathbb{P}(v_p^1 = v)
\end{aligned} \tag{2.7}
$$

Assuming the informativeness criterion (2.5) is true, *i.e.* $H(z_p \mid v_p) = 0$, then we also have that for any variable Y, $H(z_p|v_p, Y) \leq H(z_p|v_p) = 0$. In other words, $x \to v_p \to z_p$ is a Markov chain and z_p is perfectly defined by knowing the ground-truth value v_p. Denoting $\delta^p_{v,z} = \mathbb{P}(z_p = z \mid v_p = v)$, and applying the Markov property to (2.7), this means that the model is oblivious if and only if:

$$
\begin{aligned}
&\mathbb{P}(z_p^2 = z^2,\ z_p^1 = z^1) = \mathbb{P}(z_{p'}^2 = z^2,\ z_{p'}^1 = z^1) \\
\iff &\sum_v \delta^p_{r(v),z^2}\ \delta^p_{v,z^1}\mathbb{P}(v_p^1 = v) = \sum_v \delta^{p'}_{r(v),z^2}\ \delta^{p'}_{v,z^1}\mathbb{P}(v_{p'}^1 = v)
\end{aligned}
$$

Because values are sampled i.i.d. in the abstract visual reasoning generative process, and because we consider sequences with the same underlying rule r, we can assume that here v_p have the same distribution for all $p \in \mathcal{P}$. Therefore, what really matters for this equality to hold, is how the terms $\delta^p_{v,z}$ varies for different p, *i.e.* how does the encoders for different visual properties differ in the information they encode about their respective property v_p.

In contrast, *weak obliviousness* requires that the representation z_p has the same distribution for all positions in the latent code $p \in \mathcal{P}$. In other words, that

$$
\begin{aligned}
&\mathbb{P}(z_p = z) = \mathbb{P}(z_{p'} = z) \\
\iff &\sum_v \mathbb{P}(z_p = z \mid v_p = v)\ \mathbb{P}(v_p = v) = \\
&\sum_{v'} \mathbb{P}(z_{p'} = z \mid v_{p'} = v')\ \mathbb{P}(v_{p'} = v') \\
\iff &\sum_v \delta^p_{v,z}\ \mathbb{P}(v_p = v) = \sum_{v'} \delta^{p'}_{v',z}\ \mathbb{P}(v_{p'} = v')
\end{aligned}
$$

This property is *weaker* than the obliviousness criterion because it only imposes a constraint on the average δ terms, irregardless of the rule r or the image's position in the sequence. This is also highlighted in the counter example introduced in the previous paragraph.

In summary, weak obliviousness does not yield as strong a guarantee as obliviousness in theory, however it is a more practical and easier constraint to implement. In fact, as we show in experiments, this criterion can be achieved using domain adaptation techniques.

2.6 Experimental Analysis

In this section, we aim to quantitatively answer two questions: First, we analyze performance of different models on the rationale learning task and show a sharp transition when the model does not reflect the abstract structure of the task. Second, we investigate how well the metrics we propose in Section 2.5 correlate with abstract reasoning performance and to what extent existing representation learning methods are able to achieve the desired criteria.

We conduct all experiments on the `3DS` and `Raven` benchmarks introduced in Section 2.3. For each, we use 100k training samples, 10k validation samples (used for instance to tune the learning rate) and 40k test samples. In order to avoid biased results due to the choice of held-out rationales, we consider three different held out settings for each dataset with non-intersecting set of held out rationales. The number of held-out rationales is always 5 out of 25 for `3DS` and 9 out of 28 for `Raven`.

Baselines. We study different variants of two base architectures that have been proposed for abstract visual reasoning in previous work: namely, the convolutional network (CNN) of [Hoshen and Werman, 2017] and the relation network (RN) of [Santoro *et al.*, 2017; Santoro *et al.*, 2018]. CNN is a straightforward convolutional architecture that, despite its simplicity, has been reported to perform as well [Santoro *et al.*, 2018], or even better [Zhang *et al.*, 2019a] on the candidate prediction task than sequential architectures such as LSTMs. RN proposes a simple yet effective way to model relations across panels in the sequence: First, each image is embedded in a latent space by a shared encoder. Then each, pairwise combination of these embeddings is processed by a small Multi-Layer Perceptron (MLP). The results are collected then summed together before being passed through a final MLP that outputs the model answer.

Proposed models. First, we create variants for each of these models that explicitly fulfill the criteria of being systematic and consistent, which we denote CNN-c and RN-c. The implementation is straightforward following the scheme of Figure 2.4: The model first maps each image of the sequence to a real-valued embedding vector, split across visual properties. These embeddings are concatenated and passed through a consistent cognition module: For CNN-c, this is a standard MLP, and for RN-c, a MLP that uses the same "sum pairwise contributions" structure as the original RN.

As discussed in Section 2.5.1, we are not aware of any technique that guarantees the model to be oblivious without additional supervision. As a closest approximation, we use a domain adversarial neural network [Ganin *et al.*, 2016] (DANN), which is a common domain adaptation method: It consists in jointly training the classification task (here, rationale classification) with a *domain classifier*, in an adversarial manner, which pushes the embeddings of both domains to lie close to one another in the latent representations' space. Following [Schoenauer-Sebag *et al.*, 2019], we tackle our multi-domain setting by introducing a domain classifier for each pair of visual properties. The domain classifiers are trained adversarially using the Gradient Reversal Layer as in [Ganin *et al.*, 2016], as well as the same training scheme (Momentum SGD optimizer, and, starting from 0, slowly increasing the weight of the domain classification loss during training). The resulting models are named CNN-co and RN-co.

Finally, as a sanity check, we also introduce a strongly-supervised "oracle" setting, where obliviousness of the model can be successfully achieved: For this, we assume additional supervision in form of the abstract values that underlie the generated images and train the encoder to map images to these intermediate ground-truth representations. We again consider two variants of the model: In one, the learned representations are passed through an unstructured MLP, while in the other a consistent MLP is used to yield the final decision. We denote these as CNN* and CNN*-c respectively.

2.6.1 Main results

We report our main results on the rationale learning task in Table 2.2, for both the situation that requires abstract reasoning (heldout compositions, $\mathcal{Y}_{\text{heldout}}$), as well as in the standard supervised learning setting ($\mathcal{Y}_{\text{train}}$).

First, we observe a significant drop in accuracy between the $\mathcal{Y}_{\text{train}}$ and the $\mathcal{Y}_{\text{heldout}}$ scenarios for all models except CNN*-c, which excels in all situations but requires additional supervision. A partial exception is CNN-co and RN-co for the `3DS` dataset, which we discuss in more details below. This shows that without adequate constraints, current systems learn to always exclude the held-out rationales, hence entirely ignoring the abstract reasoning aspect of the task. Furthermore, it confirms experimentally that it is not enough to enforce only two of the three criteria from Section 2.4.1. In fact, the CNN-c and RN-c models are systematic and consistent, but not oblivious, and the CNN* is systematic and oblivious, but not consistent. Neither of them achieve better than chance accuracy for held-out rationales. The consistent success of a simple architecture such as CNN*-c shows that the bottleneck for abstract reasoning lies not in the network architecture itself, but rather in setting adequate constraints on the intermediate representations.

Finally, the CNN-co and RN-co models, which use a systematic and consistent architecture and aim to enforce obliviousness by integrating a domain adversarial network, can be considered a partial success. On the simple `3DS` setting, the models indeed achieve high accuracies even for held-out rationales: in fact, here, 3 out of 5 properties are visually similar (wall's, floor's and object's hue) and the underlying rules themselves are close in functional form. For the more difficult challenging `Raven` data, however, the model struggles at aligning the visual properties of interest and do not manage to close the performance gap. Nevertheless, we take these results as evidence that trying to enforce obliviousness during training is indeed a promising way towards achieving deep networks with the ability for abstract reasoning.

Additionally, we note that the generalization abilities of the "-co" models, directly correlate to the performance domain adaptation modules: For the `3DS` benchmark, the domain classifiers all usually reach around 50% accuracy, indicating that the encoder does a good job at fooling the adversarial classifier. On the other hand, despite ex-

	3DS random chance: 0.04		Raven no distractors random chance: 0.036		Raven w/ distractors random chance 0.036	
	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$
CNN	0.997	0.000	0.753	0.000	0.330	0.000
RN	0.767	0.000	0.736	0.000	0.312	0.000
CNN-c	0.997	0.000	0.723	0.003	0.394	0.000
RN-c	0.978	0.020	0.785	0.000	0.425	0.000
CNN-co	0.999	0.988	0.675	0.070	0.327	0.030
RN-co	0.970	0.797	0.726	0.080	0.333	0.000
CNN*	0.996	0.000	0.941	0.000	0.639	0.000
CNN*-c	1.000	0.910	0.998	0.974	0.994	0.975

(a) Held out setting 1

	3DS random chance: 0.04		Raven no distractors random chance: 0.036		Raven w/ distractors random chance 0.036	
	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$
CNN	0.996	0.000	0.767	0.000	0.339	0.000
RN	0.763	0.000	0.753	0.000	0.301	0.000
CNN-c	0.998	0.042	0.762	0.000	0.415	0.000
RN-c	0.971	0.131	0.759	0.000	0.406	0.000
CNN-co	0.998	0.913	0.710	0.042	0.329	0.000
RN-co	0.965	0.946	0.690	0.000	0.356	0.002
CNN*	0.993	0.000	0.958	0.000	0.658	0.000
CNN*-c	1.000	0.918	0.994	0.950	0.987	0.966

(a) Held out setting 2

	3DS random chance: 0.04		Raven no distractors random chance: 0.036		Raven w/ distractors random chance 0.036	
	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$	$\mathcal{Y}_{\text{train}}$	$\mathcal{Y}_{\text{heldout}}$
CNN	0.994	0.000	0.765	0.000	0.324	0.000
RN	0.754	0.000	0.763	0.000	0.291	0.000
CNN-c	0.998	0.000	0.750	0.000	0.360	0.000
RN-c	0.971	0.002	0.814	0.000	0.430	0.000
CNN-co	0.999	0.978	0.692	0.048	0.327	0.000
RN-co	0.971	0.923	0.699	0.000	0.335	0.000
CNN*	0.989	0.000	0.951	0.000	0.647	0.000
CNN*-c	1.000	0.899	0.994	0.982	0.994	0.984

(a) Held out setting 3

Table 2.2: Complete accuracy results on the Rationale prediction task for the different models we consider. The $\mathcal{Y}_{\text{heldout}}$ column corresponds to evaluation on the test set, containing held out rationales. The $\mathcal{Y}_{\text{train}}$ column corresponds to a set sampled from the same distribution as the training set (yet distinct from the actual training instances).

	3DS		Raven no distractors		Raven w/ distractors	
	Obl.	MIG	Obl.	MIG	Obl.	MIG
CNN-c	1.61	0.50	2.20	0.10	2.08	0.06
RN-c	1.57	0.76	2.20	0.14	2.20	0.05
CNN-co	1.53	0.53	2.19	0.11	1.95	0.03
RN-co	1.25	0.91	2.16	0.10	2.10	0.03
CNN*-c	0.15	1.00	0.71	0.51	0.70	0.55

Table 2.3: Estimates of *obliviousness* by measuring mutual information (2.4) (lower is better) and *mutual information gap* (higher is better) for consistent models.

periment with different settings and hyperparameters, this regime is never reached in the Raven experiments. One possible explanation is that, in that setting, the different visual properties are harder to capture and to align, as they have very varied appearance, and only knowing the rationale underlying the sequence is not a strong enough supervisory signal for this task.

2.6.2 Relation to the information-theoretical metrics

We now investigate the information-theoretic metrics we defined in Section 2.5.

Measuring obliviousness. First, we estimate obliviousness using the mutual information of Equation 2.4 and report the result in Table 2.3 (left columns). The results correlate well with Table 3.2: CNN*-c has significantly smaller value of the measured mutual information hence is closer to obliviousness, which correlates with it performing the best on the abstract reasoning task. Furthermore the fact that it does not perfectly reach mutual information of 0 shows that obliviousness is a sufficient but not necessary condition. This gives us hope that for real-world abstract reasoning tasks, excellent performance might be achievable even if obliviousness is fulfilled only approximately. For the models that do not rely on this additional supervision, *i.e*. CNN-co and RN-co, we observe that they are closer to obliviousness according to (2.4) than their CNN-c and RN-c counterparts on 3DS which is also consistent with their improved performance in Table 3.2.

Furthermore, the fact that CNN-c and RN-c achieve similar obliviousness values as CNN-co and RN-co respectively, yet fail at generalizing to held out rationales, shows that obliviousness alone does not suffice: Systematic-consistency is also required.

Relevance of the learned embeddings. Second, we consider the two metrics of informativeness: high $I[f_p(x); v_p]$ (2.6) and low $H[f_p(x)|v_p]$ (2.5), as introduced in Section 2.5. First we observe that Equation 2.6 is always fulfilled, *i.e.* the network indeed learns to push relevant information about the visual property into the designated part of the representation. On the other hand, (2.5) is not necessarily low, except when additional supervision is used at training time. This means that the latent captures more information than strictly necessary, which in some situations could lead to overfitting.

To better analyze how the different visual properties embeddings interact with one another, we measure all pairwise mutual informations $I(f_p(x); v_{p'})$. We report the full pairwise mutual information matrices for the CNN*-c models in Table 2.4: *i.e.* the pairwise mutual information terms between the ground-truth values and the embedding learned with these values as additional supervision. In particular, this gives us a good insight into how challenging each benchmark is: In `3DS`, every visual property are independent from one another hence we obtain a diagonal matrix. While in `Raven`, the definition of the task introduces some correlations between properties, which is highlighted by the block pattern in the mutual information matrix: For instance since objects do not overlap, the positions of the object is correlated with their number. Because of these biases and correlations it might be more difficult for the model to learn to identify the different visual properties.

Finally, because it is hard to directly compare these matrices, we also report their mutual information gap (MIG) [Chen *et al.*, 2018] as a summary statistics:

$$\text{MIG}(f) = \sum_{p=1}^{|\mathcal{P}|} \frac{1}{H(v_i)} \left(I(f_{m(p)}(x); v_p) - \max_{q \neq m(p)} I(f(x); v_p) \right)$$
$$\text{where } \forall p,\; m(p) = \arg\max_q I(f_q(x); v_p)$$

It corresponds to the average of the differences between the two highest mutual information terms for each row in the matrix. It takes a value of 1 when every component is independent from one another, and 0 if every component is perfectly correlated with at least one other. This gives us a single quantitative metric to assess how correlated the learned visual property embeddings are in practice. We report corresponding MIGs in Table 2.3 and observe that MIG correlates well with the difficulty of the dataset.

2.7 Conclusion

In this chapter, we took a foundational view on the problem of abstract visual reasoning. Inspired by the fact that none of the methods proposed so far actually succeed on this task, we derived three criteria that in combination enable a model to perform abstract visual reasoning. Two of them can be achieved simply by a suitable choice of architecture, while the third one is an invariance condition on the learned representations that currently no existing representation learning method seems to reliably achieve. We show theoretically and experimentally that this third criterion, obliviousness, is the crucial missing puzzle piece for giving computer vision models the ability for abstract reasoning from visual data. However, enforcing obliviousness without additional supervision is a challenging task. We show in our experiments that this can be achieved in some simple settings using domain adaptation techniques, leading to improved generalization. However, this approach fails on the more complex Raven-inspired dataset.

We hope that our analysis will inspire researchers in the community to take a fresh look at representation learning, in particular for the task of abstract visual reasoning. Overall, there is no consensus in the community if and under what conditions feed-forward deep networks are capable of performing abstract visual reasoning. With this work, we hope to shed some more light on the question in an objective and quantifiable way. A key insight of our analysis is that deep networks are in fact able to perform abstract visual reasoning as long as they reflect the abstract nature of the task, in particular with rules that act *consistently* across the properties through a proxy of *abstract values* rather than low-level image features.

$I[v_{p'}; f_p(x)]$	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$
Floor Hue (v_0)	2.30	0.00	0.00	0.00	0.00
Wall Hue (v_1)	0.00	2.30	0.00	0.00	0.00
Object Hue (v_2)	0.00	0.00	2.30	0.00	0.00
Scale (v_3)	0.00	0.00	0.00	2.08	0.00
Orientation (v_4)	0.00	0.00	0.00	0.00	2.71

(a) Results on `3DS`

$I[v_{p'}; f_p(x)]$	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_7(x)$	$f_8(x)$
Object Number (v_0)	2.24	2.24	0.73	0.73	0.59	0.00	0.07	0.07	0.02
Object Positions (v_1)	2.24	5.54	1.75	1.76	0.97	0.03	1.02	1.02	0.27
Object Colors (v_2)	0.73	1.75	6.09	2.01	0.96	0.05	1.29	1.28	0.43
Object Sizes (v_3)	0.73	1.76	2.01	6.09	0.97	0.05	1.28	1.28	0.44
Object Types (v_4)	0.59	0.97	0.96	0.97	4.56	0.01	0.50	0.49	0.07
Line Number (v_5)	0.01	0.03	0.05	0.05	0.01	1.92	0.83	0.83	1.92
Line Colors (v_6)	0.07	1.02	1.29	1.28	0.50	0.83	5.36	1.60	0.96
Line Widths (v_7)	0.07	1.02	1.28	1.27	0.49	0.83	1.60	5.35	0.95
Line Types (v_8)	0.01	0.27	0.43	0.44	0.07	1.92	0.96	0.95	3.78

(b) Results on `Raven` (no distractors)

$I[v_{p'}; f_p(x)]$	$f_0(x)$	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$	$f_6(x)$	$f_7(x)$	$f_8(x)$
Object Number (v_0)	2.23	2.21	0.70	0.70	0.57	0.00	0.04	0.04	0.00
Object Positions (v_1)	2.21	5.57	1.44	1.44	0.79	0.02	0.81	0.82	0.16
Object Colors (v_2)	0.70	1.44	6.10	1.69	0.75	0.03	1.08	1.09	0.29
Object Sizes (v_3)	0.70	1.44	1.70	6.10	0.75	0.03	1.08	1.08	0.30
Object Types (v_4)	0.57	0.79	0.75	0.75	4.56	0.01	0.37	0.37	0.04
Line Number (v_5)	0.00	0.02	0.03	0.03	0.01	1.91	0.79	0.79	1.87
Line Colors (v_6)	0.04	0.81	1.08	1.08	0.37	0.79	5.40	1.30	0.85
Line Widths (v_7)	0.04	0.82	1.09	1.09	0.37	0.79	1.30	5.41	0.85
Line Types (v_8)	0.01	0.16	0.29	0.30	0.04	1.87	0.85	0.85	3.82

(c) Results on `Raven` (with distractors)

Table 2.4: Pairwise mutual information terms between the representations learned by CNN*-c and the true abstract values, $I(v_{p'}(x); f_p(x))$, for `S3D` and `Raven` (no distractors). The background color of each cell is determined by the magnitude of the mutual information relatively to maximum *in each row* (the darker the higer).

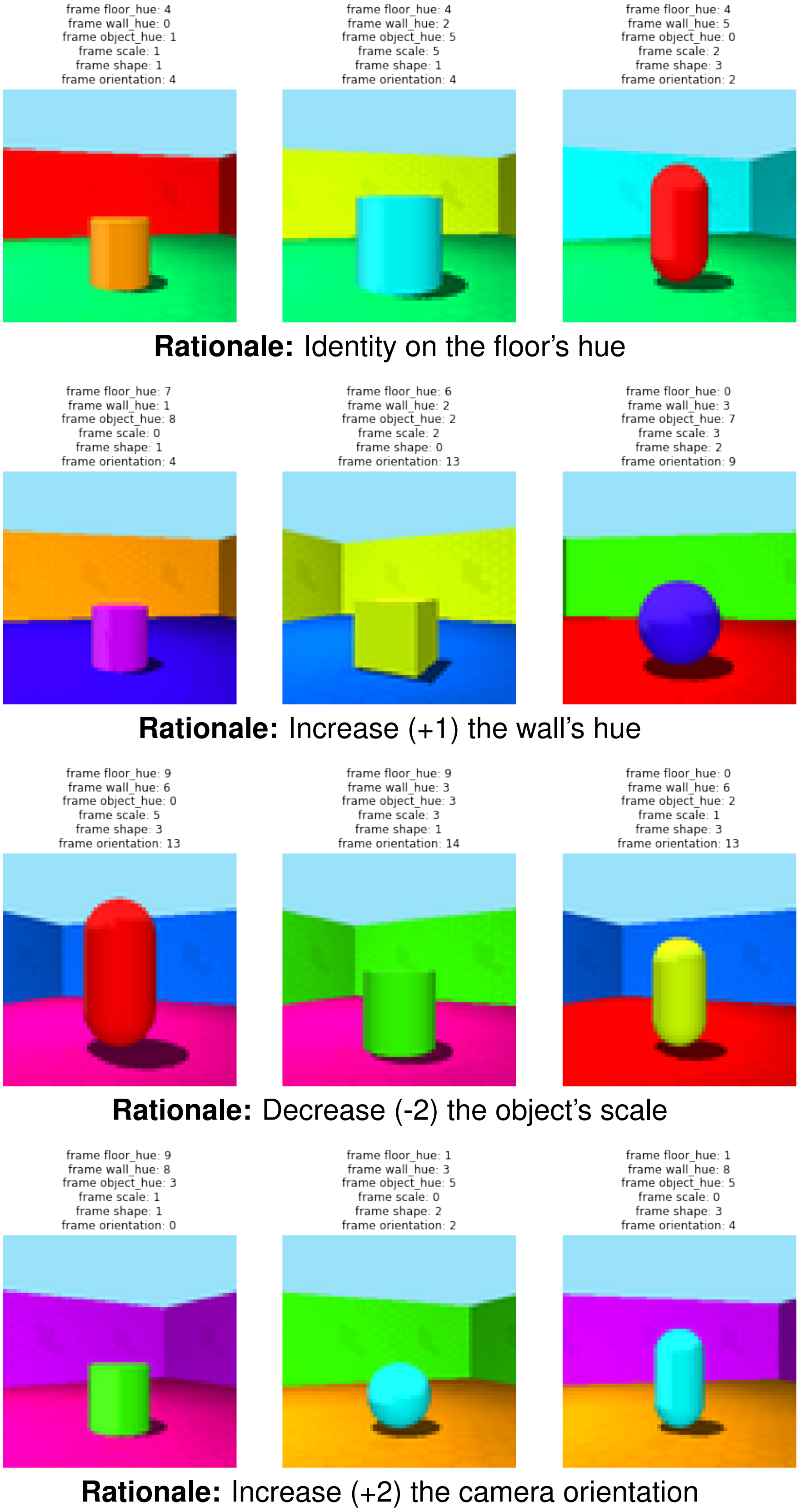


Figure 2.5: Examples from the `3DS` dataset, including the per-panel values to visual property assignment.

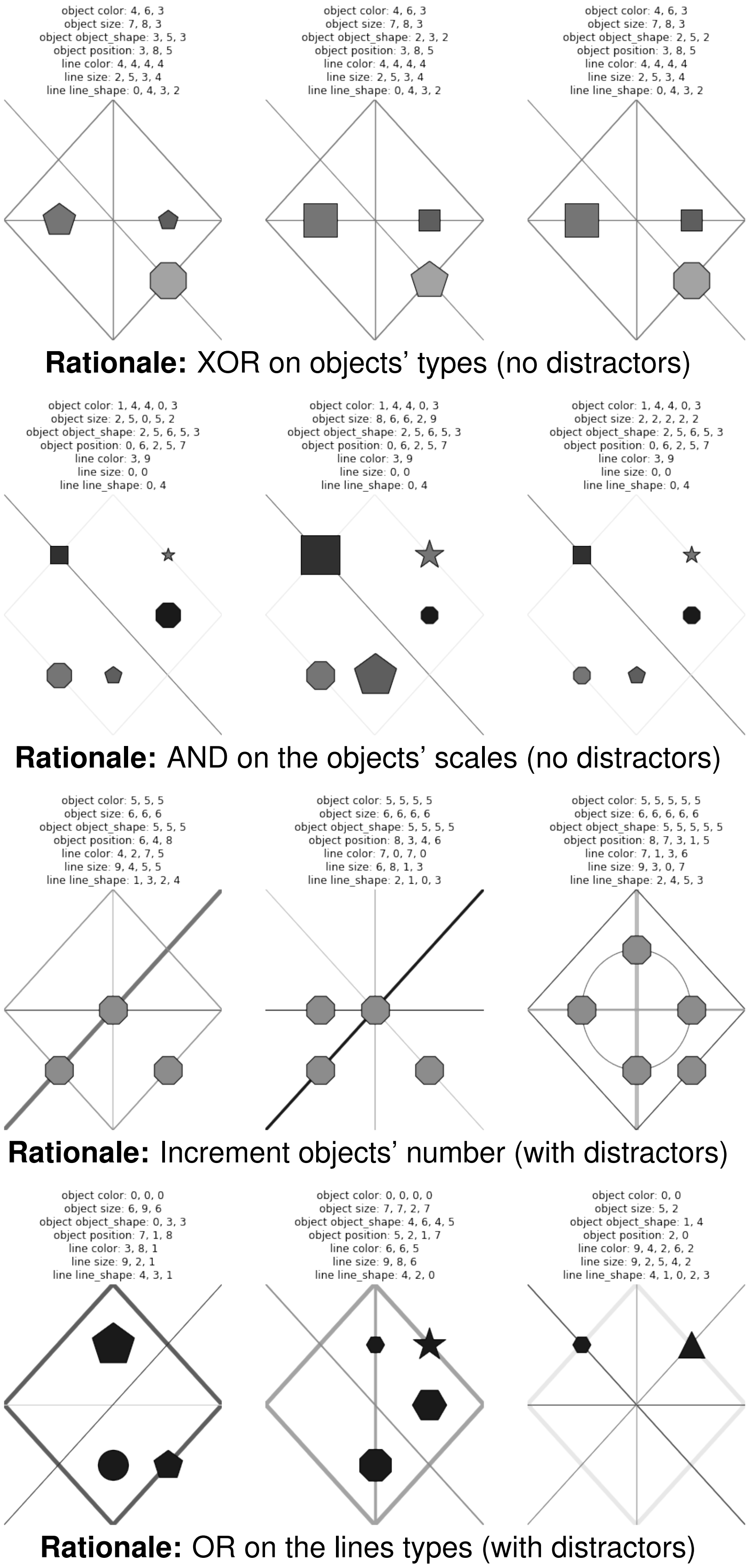


Figure 2.6: Examples from the `Raven` dataset, including the per-panel values to visual property assigment.

3 Modularity for Computationally Efficient Models

In this chapter, we investigate computer vision tasks that can be rephrased as modular recurrent problems. We highlight two specific examples where this structure can be exploited to improve the accuracy versus computational efficiency trade-off, which is a common bottleneck of neural networks.

In fact, the growing amount of available public data over recent years have enabled the use of deep neural networks as powerful discriminative models, able, for instance, of predicting the category of an object out of a thousand possibilities [Russakovsky *et al.*, 2015]. Furhermore, it is often commonly accepted that increasing the depth of neural networks leads to improved accuracy in practical applications. Even though this behavior is not well understood yet [Frankle and Carbin, 2019; Sun *et al.*, 2016], it has given rise to new architecture standards of gradually increasing depth over recent years: Starting from the 8-layered AlexNet in 2012 [Krizhevsky *et al.*, 2012], to VGG-19 and Inception-V1 with around 20 layers in 2014 [Simonyan and Zisserman, 2015; Szegedy *et al.*, 2015], until the introduction of residual blocks [He *et al.*, 2016], which allowed for a significant increase in depth while bypassing associated training caveats, the first ResNet architecture boasting 152 layers. Unfortunately, this increase in depth often comes at the cost of a slower runtime and heavier memory footprint, with a trade-off that heavily depends on architecture choices [Canziani *et al.*, 2016]. To palliate this issue, there has been effort towards reducing the runtime and memory usage of neural networks, for instance by relying on specific optimized hardware [Wang *et al.*, 2019], or by post-processing trained networks to reduce the number of operations, *e.g.* via pruning [Han *et al.*, 2015; Molchanov *et al.*, 2017], or their memory usage, *e.g.* via quantization [Rastegari *et al.*, 2016; Gong *et al.*, 2015]. However, these solutions are usually designed for generic architectures and come with their own caveats (*e.g.* monetary cost, decrease in prediction accuracy).

Orthogonal to these directions, we focus on scenarios where the structure of the prediction task can be directly leveraged to improve the accuracy versus efficiency trade-off. More specifically, we consider prediction tasks of the form $f^* : \mathcal{X} \rightarrow \mathcal{Y}$ that can be *decomposed into recursive applications of an easier sub-task*, f^*_{sub}. Our goal is to exploit this recurrence for building more computationally efficient models, by relying on two key insights: First, due to the recursive structure, learned parameters can be shared across sub-tasks, thereby saving up on memory. Second, f^*_{sub} should be easier to solve than the initial task f^* itself, which, in the context of deep learning, often means that f^*_{sub} can be efficiently modeled with a smaller and faster neural network.

In Section 3.1, we focus on the task of object detection in aerial images. Currently, one-stage detectors such as YOLO [Redmon *et al.*, 2016; Redmon and Farhadi, 2018] or SSD [Liu *et al.*, 2016] provide the best accuracy versus computational efficiency trade-off for this task: They are designed as fully-convolutional architectures that output bounding box candidates for every region of an image on a low resolution regular grid of pre-determined size. These methods are usually parametrized as to favor exhaustivity, and are thus better tailored to images containing a dense distribution of large objects. As a result, their resource usage is suboptimal when applied to real-life aerial images, as these usually contain only a few sparsely distributed objects of small size, thus a lot of computations end up wasted on processing empty image sub-regions. Instead, we propose to reformulate the detection problem as a sequence of easier *group detection* sub-tasks. The resulting multi-stage architecture spares computational effort by discarding large irrelevant regions of the image early during the detection process. Further computational and memory savings arise from the fact that we can accurately solve the group detection sub-tasks with lower image resolutions in early stages, where groups are often more easily detected than individual objects, as they are larger.

In Section 3.2, we investigate the task of *image colorization*: Given an input grayscale image X, the goal is to generate diverse yet realistic colormaps Y by sampling from the conditional likelihood $P(Y|\ X)$. In other words, here, the task f^* is to model the whole image's color distribution, $P(Y|\ X)$, from the training dataset. However, the complexity of directly computing this density grows exponentially with the size of the image. Thus for practical purposes, standard feed-forward colorization models [Zhang *et al.*, 2016; Larsson *et al.*, 2016] only model the independent pixelwise color distributions, $P(Y_i|\ X)$ for all pixel positions i. Unfortunately, in practice, this leads to a lack of variety and vibrancy in the generated color samples. Instead, we propose to make use of recent advances in autoregressive generative models, that decompose the likelihood computation into smaller estimation problems using the chain rule of probability theory. Intuitively, this corresponds to iteratively solving the smaller sub-task f^*_{sub} of predicting the color distribution of one pixel, given observed values of all the pixels that precede it. We show that this allows us to build a fully probabilistic image colorization model that is trained to maximize the exact data likelihood and offer a proper sampling framework.

3.1 Grouped Instances for Object Detection

As a core component of natural scene understanding, object detection in natural images has made remarkable progress in recent years through the adoption of deep convolutional networks. A driving force in this growth was the rise of large public benchmarks, such as PASCAL VOC [Everingham *et al.*, 2015] and MS-COCO [Lin *et al.*, 2014], which provide extensive bounding box annotations for objects in natural images, across a large diversity of semantic categories and appearances. However, many real-life detection problems exhibit drastically different data distributions and computational requirements, for which state-of-the-art detection systems are not well suited, as summarized in Table 3.1. For example, object detection in aerial or satellite imagery often requires localizing objects of a *single class*, *e.g.*, cars [Zhao and Nevatia, 2001], houses [Müller and Zaum, 2005] or swimming pools [Steinvorth, 2010]. Similarly, in biomedical applications, only some specific objects are relevant, *e.g.* certain types of cells [Xie *et al.*, 2018]. Moreover, input images in practical detection tasks are often of much higher resolution, yet contain small and sparsely distributed objects of interest, such that only a very limited fraction of pixels is actually relevant, while most academic benchmarks often contain more salient objects and cluttered scenes. Last but not least, detection speed is often at least as important as detection accuracy for practical applications. This is particularly apparent when models are meant to run on embedded devices, such as autonomous drones, which have limited computational resources and battery capacity.

In this work, we propose **ODGI** (**O**bject **D**etection with **G**rouped **I**nstances), a *top-down* detection scheme specifically designed for efficiently handling inhomogeneous object distributions, while preserving detection performance. Its key benefits and components are summarized as follows:

(i) A *multi-stage pipeline*, in which each stage selects only *a few promising regions* to be analyzed by the next stage, while discarding irrelevant image regions.

(ii) Fast single-shot detectors augmented with the ability to identify *groups of objects* rather than just individual objects, thereby substantially reducing the number of regions that have to be considered.

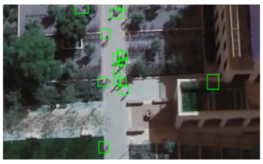
Dataset	**VEDAI** 	**SDD** 	**MS-COCO** 
Average object size (in fraction of image pixels)	small (0.113%)	small (0.159%)	large (14.96%)
Average empty cell ratio (on a 16x16 uniform grid)	95.1%	97.1%	49.4%
Object distribution	few, specific, classes. Sparse and heterogeneous object distribution		high object density across many and varied categories
Typical applications	remote sensing, traffic monitoring, medical imaging...		robotics, artificial intelligence, HCO
Resource constraints	imposed by hardware, *e.g.* limited battery, no GPU		generally none

Table 3.1: Several benchmarks and challenges highlight the task of detecting small objects in aerial views, in particular for real-life low-resource scenarios. However, the data distribution and computational constraints for such tasks often vastly differ from the most commonly used benchmarks, for instance MS-COCO, for which state-of-the-art detection models are usually optimized.

(iii) ODGI reaches similar accuracies than ordinary single-shot detectors while operating at *lower resolution* because groups of objects are generally larger and easier to detect than individual objects. This allows for a further reduction of computational requirements.

We first review related work and compare ODGI to existing detection paradigms in Section 3.1.1. We then present the proposed method, ODGI, and its training procedure in Section 3.1.2 and Section 3.1.3. Finally, we then report main quantitative results as well as several ablation experiments results in Section 3.1.4.

3.1.1 Related work

Cascaded object detection. A popular approach to object detection consists in extracting numerous region *proposals* and then classifying them as one of the object categories of interest. This includes models such as RFCN [Dai *et al.*, 2016], RCNN and variants [Girshick, 2015; Girshick *et al.*, 2014; Cai and Vasconcelos, 2018], or

SPPNet [He *et al.*, 2014]. Proposal-based methods are very effective and can handle inhomogeneously distributed objects, but are usually too slow for real-time usage, due to the large amount of proposals generated. Furthermore, with the exception of [Ren *et al.*, 2015], the proposals are generally class-independent, which makes these methods more suitable for general scene understanding tasks, where one is interested in a wide variety of classes. When targeting a specific object category, class-independent proposals are wasteful, as most proposal regions are irrelevant to the task.

Single-shot object detection and multi-scale pyramids. In contrast, single-shot detectors, such as SSD [Liu *et al.*, 2016], or YOLO [Redmon *et al.*, 2016; Redmon and Farhadi, 2017; Redmon and Farhadi, 2018], split the image into a regular grid of regions and predict object bounding boxes in each grid cell. These single-shot detectors are efficient and can be made fast enough for real-time operation, but provide an optimal speed-versus-accuracy trade-off when the objects of interest are distributed homogeneously on the output grid. In fact, the grid size has to be chosen with worst case scenarios in mind: in order to identify all objects, the grid resolution has to be fine enough to capture all objects even in image regions with high object density, which might rarely occur, leading to numerous empty cells. Furthermore, the number of operations scales quadratically with the grid size, hence precise detection of individual small objects in dense clusters is often mutually exclusive with fast operation. Recent work [Lin *et al.*, 2017; Singh *et al.*, 2018; Najibi *et al.*, 2019; Liu *et al.*, 2016; Murari *et al.*, 2019; Yang *et al.*, 2019] proposes to additionally exploit multi-scale feature pyramids to better detect objects across varying scales. This helps mitigate the aforementioned problem but does not suppress it, and, in fact, these models are still better tailored for dense object detection.

Orthogonal to this, ODGI aims to make the best of the given input resolution and resources: It instead resorts to grouping objects when individual small instances are too hard to detect, following the paradigm that “coarse predictions are better than none”. These groups are then refined in subsequent stages, if necessary for the task at hand. Nevertheless, since ODGI is not limited by the choice of backbone network, it could be further augmented by multi-scale training and other architectural improvements.

Speed-versus-accuracy trade-off. Both the cascaded and single-shot designs introduce an intrinsic trade-off between runtime and accuracy, as discussed for instance in [Huang *et al.*, 2016], that make neither of them entirely satisfactory for real-world challenges, such as controlling an autonomous drone [Zhu *et al.*, 2018], localizing all objects of a certain type in aerial imagery [Airbus, 2018] or efficiently detecting spatial arrangements of many small objects [Tuggener *et al.*, 2018].

Our proposed method, ODGI, falls into neither of these two designs, but rather combines the strength of both in a flexible multi-stage pipeline: It identifies a small number of specific regions of interest, on which it concentrates most of its computations. Despite the sequential nature of the pipeline, each individual prediction stage is based on a coarse, low resolution, grid, and thus very efficient. ODGI's design resembles classical detection cascades [Li *et al.*, 2015; Rowley *et al.*, 1998; Viola and Jones, 2004], but differs from them in that it does not sequentially refine classification decisions for individual boxes but rather refines the actual region coordinates. As such, it is conceptually similar to techniques based on branch-and-bound [Lampert, 2010; Lampert *et al.*, 2009], or on region selection by reinforcement learning [Gao *et al.*, 2018]. Nonetheless, it strongly differs from these on a technical level as it only requires minor modifications of existing object detectors and can be trained with standard backpropagation instead of discrete optimization or reinforcement learning.

3.1.2 ODGI: Object Detection with Grouped Instances

We design ODGI as a multi-stage detection architecture $\phi_S \circ \cdots \circ \phi_1$, $S > 1$, in which each stage ϕ_s is a detection network, whose outputs can either be *individual objects* or *groups of objects*. In the latter case, the predicted bounding box defines a relevant image sub-region, for which detections can be refined by feeding it as input to the next stage. An overview of the ODGI multi-stage prediction pipeline is given in Figure 3.1.

In this section, we will first detail the design of the intermediate stages in the network, and in particular how we define groups of individual objects. Secondly, we will discuss the crop extraction procedure that we use to propagate relevant regions across stages. Next, in Section 3.1.3 we will discuss the training and evaluation procedure of the proposed model.

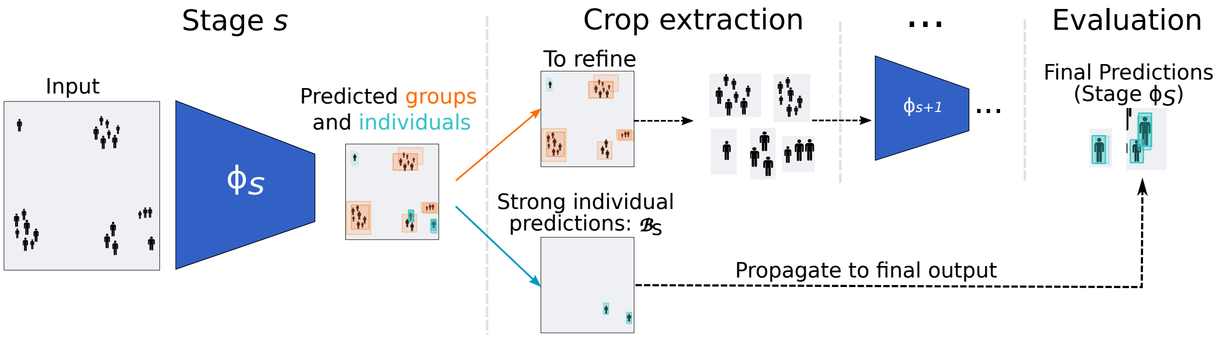


Figure 3.1: Overview of ODGI: Each stage consists of a single-shot detector that detects groups and individual objects, which are further processed to produce a few relevant image regions to be fed to subsequent stages and refine detections as needed.

Stage architecture. We design each stage as a lightweight neural network that performs fast object detection. In practice, we build on standard single-shot detectors such as YOLO [Redmon *et al.*, 2016] or SSD [Liu *et al.*, 2016]. More precisely, ϕ_s consists of a *fully-convolutional* network with output map of size $[I, J]$ directly proportional to the input image resolution. For each of the $I \times J$ cells in this uniform grid, the model predicts bounding boxes characterized by four coordinates – the box center (x, y), its width w and height h, and a predicted confidence score $c \in [0, 1]$. Following common practice [Redmon *et al.*, 2016; Redmon and Farhadi, 2017; Liu *et al.*, 2016], we express the width and height as a fraction of the total image's width and height, while the coordinates of the center are parametrized relatively to the cell it is linked to; All coordinates take values in $[0, 1]$. The confidence score $c \in [0, 1]$ is used for ranking the predicted bounding boxes at inference time. as well as for optional filtering steps, such as non-maximum suppression.

In order to compare the model with standard detection systems, we constrain the last stage's network ϕ_S to only output individual objects. In contrast, we augment intermediate stages $s \leq S - 1$ with the ability to predict groups of objects, which we parametrize as follows: *First*, we augment each predicted box with a binary *group flag,* g, as well as two real-valued *offset values* (o_w, o_h): The flag indicates whether the detector considers the prediction to be a single object, $g = 0$, or a group of objects, $g = 1$. The offset values are used to appropriately rescale the stage outputs which are then passed on to subsequent stages. *Second*, we design the intermediate stages to predict *one bounding box* per cell. This choice provides us with an *intuitive definition of groups, which automatically adapts itself to the input image resolution* without introducing additional hyperparamaters: If the model resolution $[I, J]$ is fine enough, there is at most one individual object per cell, in which case the problem reduces to standard object detection. Otherwise, if a cell is densely occupied, then the model resorts to predicting one group enclosing the relevant objects. We provide further details on the group training process in Section 3.1.3.

Stage transition. The outputs of each intermediate stage are further processed to extract image regions, which are then passed on to the next stage: Let B be a bounding box predicted at stage ϕ_s, with confidence c and binary group flag g. We distinguish three possibilities: *(i)* the box can be discarded, *(ii)* it can be accepted as an individual object prediction, or *(iii)* it can be passed on to the next stage for further refinement. This decision is made based on two confidence thresholds, τ_{low} and τ_{high}, leading to one of the three following actions:

(i) if $c \leq \tau_{\text{low}}$: The box B is discarded.

(ii) if $c > \tau_{\text{high}}$ and $g = 0$: The box B is considered a strong individual object candidate: We make it "exit" the pipeline and directly propagate it to the last stage's output as it is. We denote the set of such boxes as $\mathcal{B}_s$.

(iii) if ($c > \tau_{\text{low}}$ and $g = 1$) or ($\tau_{\text{high}} \geq c > \tau_{\text{low}}$ and $g = 0$): The box B is either a group or an individual with medium confidence and is a candidate for refinement.

We then apply standard non-maximum suppression (NMS) with threshold τ_{nms} to the set of refinement candidates $\mathcal{B}_s$, in order to obtain (at most) γ_s boxes with high confidence and little overlap. The resulting γ_s bounding boxes are then processed to build the image regions that will be passed on to the next stage by multiplying each box's width and height by $1/o_w$ and $1/o_h$, respectively, where o_w and o_h are offset values learned by the detector: This rescaling step ensures that the extracted patches cover the relevant region well enough, and compensates for the fact that the detectors are trained to predict the *exact* ground-truth coordinates, rather than to encompass them, hence sometimes underestimate the extent of the relevant region. The resulting rescaled regions are extracted from the input image and passed on as inputs to the next stage. The final output of ODGI is the combination of object boxes predicted in the last stage, ϕ_S, as well as the kept-back outputs from previous stages: $\mathcal{B}_1 \dots \mathcal{B}_{S-1}$.

3.1.3 Training the model

We train each ODGI stage independently, using a combination of three loss terms that we optimize with standard backpropagation:

$$\mathcal{L}_{\text{ODGI}} = \mathcal{L}_{\text{groups}} + \mathcal{L}_{\text{coords}} + \mathcal{L}_{\text{offsets}} \tag{3.1}$$

$\mathcal{L}_{\text{coords}}$ is a standard mean squares regression loss on the predicted coordinates and confidence scores, as described for instance in [Redmon *et al.*, 2016; Liu *et al.*, 2016]. The additional two terms are part of our contribution: The *group loss*, $\mathcal{L}_{\text{groups}}$, drives the model to classify outputs as individuals or groups, and the *offsets loss*, $\mathcal{L}_{\text{offsets}}$, encourages better coverage of the extracted regions. Note that in the last stage of the pipeline, only the second term is active, as only individual objects are predicted.

The rest of this section is dedicated to formally defining each loss term as well as explaining how we obtain ground-truth coordinates for group bounding boxes.

Group loss. Let $\mathbf{b} = b_{n=1\dots N}$ be the original ground-truth individual bounding boxes. We define $A_{i,j}(n)$ as an indicator which takes value 1 *iff* ground-truth box b_n is assigned to output cell (i,j) (and 0 otherwise) by measuring their intersection:

$$A_{i,j}(n) = [\![|b_n \cap \text{cell } (i,j) \mid > 0]\!], \tag{3.2}$$

$$\text{where } [\![\cdot]\!] : x \mapsto 1 \text{ if } x \text{ is true, else } 0$$

In principle, we need to consider all possible unions of subsets of the ground-truth bounding boxes $\mathbf{b}$ as potential targets for group prediction. However, in the previous paragraph, we have defined our intermediate detectors such that they predict only one bounding box per cell by design, which allows us to avoid this combinatorial problem. Formally, let $B_{i,j}$ be the predictor associated to cell (i,j). We define its target ground-truth coordinates $B^*_{i,j}$ and group flag $g^*_{i,j}$ as:

$$B^*_{i,j} = \bigcup_{n \mid A^{ij}(n)=1} b_n \tag{3.3}$$

$$g^*_{ij} = [\![\#\{n \mid A^{ij}(n) = 1\} > 1]\!], \tag{3.4}$$

where $\#$ denotes the cardinality of a set, and $\cup$ denotes the smallest enclosing bounding box of an ensemble of bounding boxes. We then define $\mathcal{L}_{\text{groups}}$ as a binary classification objective between predicted group flags $\mathbf{g}$ and corresponding targets $\mathbf{g}^*$:

$$\mathcal{L}_{\text{groups}}(\mathbf{g}, \mathbf{g}^*) = -\sum_{i,j} E_{i,j} \Big(g^*_{i,j} \log(g_{i,j}) + (1 - g^*_{i,j}) \log(1 - g_{i,j}) \Big), \tag{3.5}$$

where $E_{i,j} = [\![\sum_n A_{i,j}(n) > 0]\!]$ is a binary flag indicating whether cell (i,j) contains at least one ground-truth object or not.

In summary, we build ground-truth $B^*_{i,j}$ and $g^*_{i,j}$ as follows: For each cell (i,j), we build the set $G_{i,j}$ containing all ground-truth boxes that intersects with the cell. If the set is non empty and only a single object box, b, falls into this cell, we set $B^*_{i,j} = b$ and $g^*_{i,j} = 0$. Otherwise, $|G_{i,j}| > 1$; then we define $B^*_{i,j}$ as the union of bounding boxes in $G_{i,j}$, and set $g^*_{i,j} = 1$. In particular, this procedure automatically adapts to the resolution $[I, J]$ in a data-driven way, and can be implemented as a pre-processing step, thus does not produce any overhead at training time.

Coordinates loss. Following the definition of target group bounding boxes $B^*_{i,j}$ in (3.3) , we define the coordinates loss as a standard regression objective between the box coordinates $\mathbf{B}$ and confidences $\mathbf{c}$ and their respective targets, similarly to existing detectors [Girshick *et al.*, 2014; Girshick, 2015; Liu *et al.*, 2016; Erhan *et al.*, 2014; Redmon *et al.*, 2016]:

$$\mathcal{L}_{\text{coords}}(\mathbf{B}, \mathbf{B}^*) = \sum_{i,j} E_{i,j} \big(|B_{i,j} - B^*_{i,j}|^2 + \omega_{\text{c}} \, |c_{i,j} - c^*_{i,j}|^2 \big) + \omega_{\text{no-obj}} \sum_{i,j} (1 - E_{i,j}) c^2_{i,j} \tag{3.6}$$

$$\text{where } c^*_{i,j} = \texttt{IoU}(B_{i,j}, B^*_{i,j}) = \frac{|B_{i,j} \cap B^*_{i,j}|}{|B_{i,j} \cup B^*_{i,j}|} \tag{3.7}$$

The first two terms in (3.6) are ordinary least squares regression objectives between the predicted coordinates and confidence scores and their respective assigned ground-truth. The ground-truth for the confidence score is defined as the intersection over union (`IoU`) between the corresponding prediction and its assigned target. Finally, the last term of the loss is a weighted *penalty term* that pushes confidence score predictions in empty cells towards zero. In practice, we use the same weights as in the standard YOLO architecture [Redmon *et al.*, 2016], i.e. $\omega_{\text{c}} = 5$ and $\omega_{\text{no-obj}} = 1$.

Offsets loss. In intermediate stages, ODGI additionally predicts two offset values for each box, ow and oh, that are used to rescale the region of interest when it is passed as input to the next stage, as described in Section 3.1.2. The corresponding predictors are trained using the following *offsets loss*:

$$\mathcal{L}_{\text{offsets}}((ow, oh), (ow^*, oh^*)) = \sum_{i,j} E_{i,j} \Big[\left|ow_{i,j} - ow^*(B_{i,j}, B^*_{i,j})\right|^2 + \left|oh_{i,j} - oh^*(B_{i,j}, B^*_{i,j})\right|^2 \Big]. \tag{3.8}$$

The target values, $oh^*(B_{i,j}, B^*_{i,j})$ and $ow^*(B_{i,j}, B^*_{i,j})$, for vertical and horizontal offsets, are defined such that the rescaled version of $B_{i,j}$ encompasses both the original B^{ij} and its assigned ground-truth box $\bar{B}^{ij}$ with a certain margin δ. In practice, since we deal with very small objects, we set δ to be half the average object size ($\delta = 0.0025$). Formally, we denote α and h the functions mapping a bounding box to its center y-coordinate and its half-height respectively. Then we define:

$$\begin{aligned} h^{\text{scaled}}(B, B^*) &= \max\big(\,|(\alpha(B^*) + h(B^*) + \delta) - \alpha(B)|, \\ &\qquad\qquad |(\alpha(B^*) - h(B^*) - \delta) - \alpha(B)|\,\big) \\ oh^*(B_{i,j}, B^*_{i,j}) &= \max(1, h(B_{i,j})/h^{\text{scaled}}(B_{i,j}, B^*_{i,j})) \end{aligned} \tag{3.9}$$

For the horizontal offset ow^*, we use the analogous construction but with $B_{i,j}$'s center x-coordinate and its half-width instead. See Figure 3.2 for an illustration.

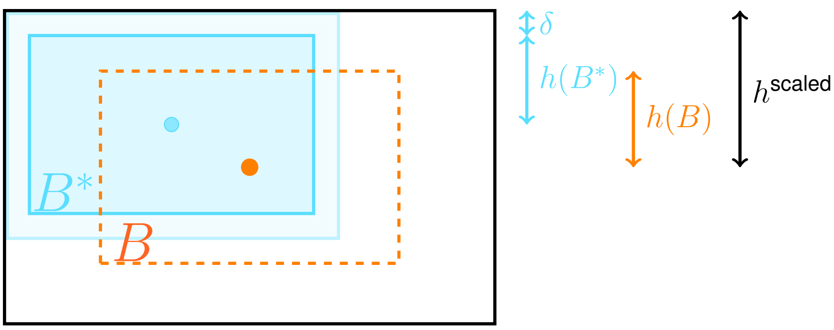


Figure 3.2: Illustration of (3.9) for defining the offsets ground-truth in the loss $\mathcal{L}_{\text{offsets}}$. The black box, with height h^{scaled}, is the smallest bounding box that is centered around the predicted bounding box center (•) and that encompasses both the ground-truth bounding box (B^*) with δ-margin and the predicted box (B). The offset oh^* is defined as the multiplicative factor needed to reach h^{scaled} from the predicted height $h(B)$.

Multi-stage training. By design, the inputs of stage s are obtained from the outputs of stage $s-1$. However, it would be cumbersome to wait for each stage to be fully trained before starting to train the next one. In practice we notice that even after only a few epochs, the top-scoring predictions of intermediate detectors often detect image regions that can be useful for the subsequent stages, thus we propose the following *delayed training procedure*: After $n_e = 3$ epochs of training the first stage, we start training the second, querying new inputs from a queue fed by the outputs of the first stage. This allows us to jointly and efficiently train the two stages, and this delayed training scheme works well in practice. Other than this, the model parameters are trained with a standard Adam optimizer [Kingma and Ba, 2015] (each detection stage being an independent optimization problem) and an initial learning rate of $1\text{e-}3$.

Additionally, the patch extraction procedure that occurs when transiting from one stage to the next can be tuned via four hyperparameters: $\tau_{\text{low}}, \tau_{\text{high}}, \tau_{\text{nms}}, \gamma_s$. At training time, we allow as many boxes to pass as the memory budget allows, because both negative and positive patches can be useful for training subsequent stages. For our experiments, this meant setting $\gamma_s^{\text{train}} = 10$. In the same vein, we also do not use any filtering during training, *i.e.* no thresholding ($\tau_{\text{low}}^{\text{train}} = 0$, $\tau_{\text{high}}^{\text{train}} = 1$) nor non-maximum suppression ($\tau_{\text{nms}}^{\text{train}} = 1$) , For test-time prediction we use a held-out validation set to determine their optimal values, as described in Section 3.1.4. Moreover, these hyperparameters can be easily changed on the fly, without retraining. This allows the model to easily adapt to changes of the input data characteristics, and to make better use of an increased or reduced computational budget for instance.

3.1.4 Experiments

In this section, we report our main experimental results. We analyze the proposed pipeline in terms of both detection accuracy and efficiency (computational as well as memory-wise). We perform experiments on two different datasets and three different backbones. In the next two sections, we discuss results of ablation experiments that provide further insights about the model.

Datasets. We consider two aerial views datasets: VEDAI [Razakarivony and Jurie, 2015] contains 1268 aerial views of countryside and city roads for vehicle detection.

Images are 1024x1024 pixels and contain on average 2.96 objects of interest. We perform 10–fold cross validation, as in [Razakarivony and Jurie, 2015], with 8 folds for training, one for validation and one for testing for each run. All reported metrics are averaged over the 10 runs. Our second benchmark, SDD [Robicquet *et al.*, 2016], contains multiple drone videos with bounding box annotations of road users. To reduce redundancy, we extract still images every 40 frames, which we then pad and resize to 1024x1024 pixels to compensate for different aspect ratios. For each location, we perform a random train/val/test split with ratios 70%/5%/25%, resulting in a total of 9163, 651 and 3281 images respectively. On average, the training set contains 12.07 annotated objects per image. SDD is overall much more challenging than VEDAI: At full resolution, objects are small and hard to detect, even to the human eye.

Baselines. We consider three backbone networks commonly used in detection models: `tiny`, a simple 7-layer fully convolutional network based on the tiny-YOLOv2 architecture, `yolo`, a VGG-like network similar to the one used in YOLOv2 [Redmon *et al.*, 2016] and finally `MobileNet` [Sandler *et al.*, 2018], which is for instance used in SSD Lite [Liu *et al.*, 2016]. We implement all models in Tensorflow and to facilitate reproducibility, we will make our code publicly available [Royer, 2020].

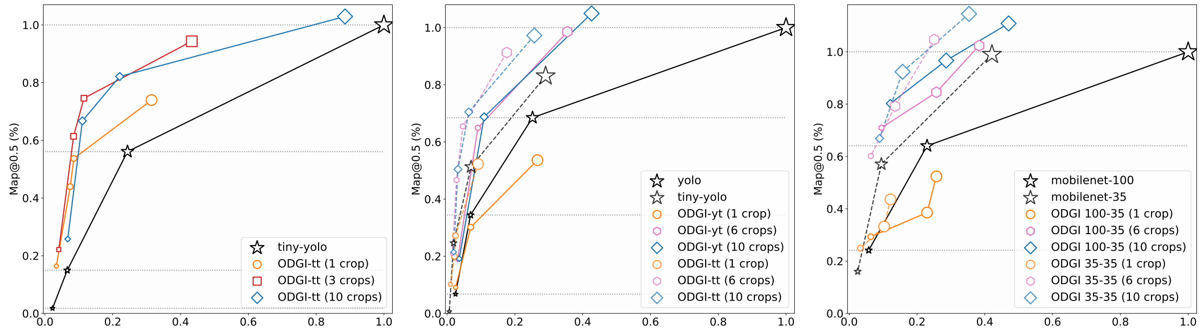
More specifically, on the VEDAI dataset, we train a standard tiny-YOLOv2 detector as baseline and compare it to ODGI-*teeny-tiny* (ODGI-tt), which refers to two-stage ODGI with `tiny` backbones. For SDD, objects are much harder to detect, thus we use a stronger YOLOv2 model as baseline. We compare this to ODGI-*teeny-tiny* as above, as well as to a stronger variant, ODGI-*yolo-tiny* (ODGI-yt), in which ϕ_1 is based on the `yolo` backbone and ϕ_2 on `tiny`. Finally we also experiment with the lightweight `MobileNet` architecture as baseline and backbones, with depth multipliers 1 and 0.35. The corresponding ODGI models are denoted as ODGI-*100-35* and ODGI-*35-35*. All models are trained and evaluated at various resolutions to investigate different grouping scenarios. In all cases, the detector grid size scales linearly with the image resolution, because of the fully convolutional network structure, ranging from a 32×32 grid for 1024px inputs to 2×2 for 64px.

Evaluation metrics. We quantitatively evaluate the ODGI pipeline as a standard object detector: Following the common protocol from PASCAL VOC 2010 and later challenges [Everingham *et al.*, 2015], we sort the list of predicted boxes in decreasing order of confidence score and compute the *average precision (MAP)* respectively to the ground-truth, at the IoU cut-offs of 0.5 (*standard*) and 0.75 (*more precise*). As is often done, we also apply non-maximum suppression to the final predictions, with IoU threshold of 0.5 and no limit on the number of outputs, to remove near duplicates for all methods. In line with our target scenario of single-class object detection, we ignore class information in experiments and focus on raw detection, although class labels could easily be added, for instance as a post-processing classification operation.

Besides retrieval performance, we also assess the computational and memory resource requirements of the different models: We record the number of boxes predicted by each model, and measure the average runtime of our implementation for one forward pass on a single image. As reference hardware, we use a server with a *2.2 GHz Intel Xeon processor (short: CPU)* in single-threaded mode. Additional timing experiments on weaker and stronger hardware, as well as a description of how we pick ODGI's test-time hyperparameters can be found later in this section.

Main results. We report experiment results in Figure 3.3 (see Table 3.2 (left) for the corresponding numbers).

We find that the proposed method improves over standard single-shot detectors in two ways: *First*, when comparing models with similar accuracies, ODGI generally requires fewer evaluated boxes and shorter runtimes, and often lower input image resolution. In fact, only a few relevant regions are passed to the second stage, at a smaller input resolution, such that they incur a small computational cost, and yet, the ability to selectively refine the boxes can substantially improve detection. *Second*, for any given input resolution, ODGI's refinement cascade generally improves detection retrieval, in particular at lower resolutions, *e.g.* 256px: In fact, ODGI's first stage can be kept efficient and operate at low resolution, because the regions it extracts do not have to be very precise. Nonetheless, the regions selected in the first stage form an easy-to-solve detection task for the second stage (see for instance Figure 3.7 (d)), which leads to more precise detections after refinement. This also motivates our choice of mixing backbones, *e.g.* using ODGI-*yolo-tiny*, instead of the heavier variant ODGI-*yolo-yolo*, since as detection in stage 2 is usually much easier.



(a) VEDAI (`tiny`) (b) SDD (`yolo`) (c) SDD (`MobileNet`)

Figure 3.3: Plots of `MAP@0.5` versus runtime (CPU) for our different experimental settings. The metrics are reported as percentages relative to the corresponding baseline run at full resolution (1024x1024 pixels). Each marker corresponds to a different input resolution, which the marker size is proportional to. The black line with ☆ markers represents the baseline model, while each colored line corresponds to an DOGI model with different number of extracted crops, γ_1, For readability, we only report results for a subset of γ_1 values, and provide full plots in the supplemental material.

As can be seen in Figure 3.3, a higher number of crops, γ_1^{test}, improves detection, but comes at a higher computational cost. Nonetheless, ODGI appears to have a better accuracy-speed ratio for most values of γ_1^{test}. For practical purposes, we suggest to choose γ_1^{test} based on how many patches are effectively used for detection. We define the *occupancy rate* of a crop as the sum of the intersection ratios of ground-truth boxes that appear in this crop. We then say a crop is *relevant* if it has a non-zero occupancy rate, *i.e.* it contains objects of interest: For instance, at input resolution 512px on VEDAI's validation set, we obtain an average of 2.33 relevant crops , hence we set $\gamma_1^{\text{test}} = 3$. The same analysis on SDD yields $\gamma_1^{\text{test}} = 6$ at the same resolution. In particular, these are the values we used for the models reported in Table 3.2.

For the sake of completeness, in Figure 3.6, we experiment results comparing baselines to ODGI for all number of crops $\gamma_1 \in [1, 10]$. As expected, increasing the number of crops always improves the final detection accuracy. However, the increase is not worth the additional computational effort in some cases. Nonetheless, the curves corresponding to ODGI methods are always above the ones for the standard detectors baselines, even for large number of crops, showing that the proposed method is computationally interesting for different budget allocations.

VEDAI (tiny)	MAP@0.5	MAP@0.75	CPU [s]	Raspi [s]	GPU [ms]	$\#_{\text{parameters}}$	$\#_{\text{pixels}}$
ODGI-tt 512-256	0.646	0.422	0.83	4.89	13.95	22M	458k
ODGI-tt 512-64	0.562	0.264	0.58	3.32	13.27	22M	274k
ODGI-tt 256-128	0.470	0.197	0.22	1.18	11.69	22M	98k
ODGI-tt 256-64	0.386	0.131	0.16	0.87	11.70	22M	73k
ODGI-tt 128-64	0.143	0.025	0.08	0.44	11.70	22M	25k
tiny-yolo 1024	0.684	0.252	1.93	10.47	14.28	11M	1M
tiny-yolo 512	0.383	0.057	0.47	2.62	8.16	11M	262k
tiny-yolo 256	0.102	0.009	0.13	0.70	6.97	11M	65k

SDD (yolo)	MAP@0.5	MAP@0.75	CPU [s]	Raspi [s]	GPU [ms]	$\#_{\text{parameters}}$	$\#_{\text{pixels}}$
ODGI-yt 512-256	0.463	0.069	2.35	16.40	24.49	62M	655k
ODGI-tt 512-256	0.429	0.061	1.17	7.03	14.68	22M	655k
ODGI-yt 256-128	0.305	0.035	0.60	4.55	18.75	62M	164k
ODGI-tt 256-128	0.307	0.044	0.31	1.78	11.97	22M	164k
yolo 1024	0.470	0.087	6.63	46.94	14.28	51M	1M
yolo 512	0.309	0.041	1.67	12.06	8.16	51M	262k
yolo 256	0.160	0.020	0.46	3.42	6.97	51M	65k

SDD (mobile-100)	MAP@0.5	MAP@0.75	CPU [s]	Raspi [s]	GPU [ms]	$\#_{\text{parameters}}$	$\#_{\text{pixels}}$
ODGI-100-35 512-256	0.434	0.061	0.76	6.63	19.89	2.6M	655k
ODGI-100-35 256-128	0.294	0.036	0.19	1.55	17.62	2.6M	164k
mobile-100 1024	0.415	0.061	1.99	17.35	23.10	2.2M	1M
mobile-100 512	0.266	0.028	0.46	4.01	10.98	2.2M	262k
mobile-100 256	0.100	0.009	0.11	0.92	9.49	2.2M	65k

SDD (mobile-35)	MAP@0.5	MAP@0.75	CPU [s]	Raspi [s]	GPU [ms]	$\#_{\text{parameters}}$	$\#_{\text{pixels}}$
ODGI-35-35 512-256	0.425	0.055	0.50	4.09	17.83	800k	655k
ODGI-35-35 256-128	0.250	0.029	0.13	1.01	17.40	800k	164k
mobile-35 1024	0.411	0.054	0.84	6.79	13.91	400k	1M
mobile-35 512	0.237	0.026	0.19	1.51	9.81	400k	262k
mobile-35 256	0.067	0.007	0.050	0.42	9.32	400k	65k

Table 3.2: MAP and timing results for our differet experimental settings. The results for ODGI models are reported with γ_1^{test} chosen as described in the **Main results** paragraph. Time is indicated in seconds for a *Raspberry Pi (Raspi)*, and in milliseconds for an *Nvidia GTX 1080Ti graphics card (GPU)*. $\#_{\text{pixels}}$ is the total number of processed pixels, and $\#_{\text{parameters}}$, the number of model parameters.

Finally, in Figure 3.7-3.10 , we report qualitative results. We observe that the detected groups exhibit three nice properties: *First,* they are of relatively small size. Consequently, the regions fed as inputs to stage 2 are well localized and effectively provide a beneficial “zoom-in” effect on relevant regions. *Second,* they contain in general only a few objects, which makes the detection task for stage 2 easier. *Finally*, the sparse distribution of objects in the input images leads to only a few, non-overlapping, generated relevant regions. Combined, these conditions contribute to improving the speed-versus-accuracy trade-off by providing the second stage with few relevant regions containing a detection problem easier than the one provided to stage 1.

Measuring runtime. Absolute runtime values always depend on several factors, in particular the software implementation and hardware. In our case, software-related differences are not an issue, as all models rely on the same core backbone implementation. To analyze the effect of hardware, we performed additional experiments on weaker hardware, a *Raspberry Pi 3 Model B with 1.2 GHz ARMv7 CPU (Raspi)*, as well as stronger hardware, an *Nvidia GTX 1080Ti graphics card (GPU)*. Table 3.2 (right) shows the resulting runtimes of one feed-forward pass for all models. We also report the total number of pixels processed by each method, *i.e.* that have to be stored in memory during one feed-forward pass, as well as the number of model parameters.

Our main observations again hold: On the Raspberry Pi, timing ratios are roughly the same as on the Intel CPU, only the absolute scale changes. The differences are smaller on GPU, but ODGI is still faster than the baselines in most cases at similar accuracy levels. Note that for the application scenario we target, the GPU timings are the least representative, as systems operating under resource constraints typically cannot afford the usage of a 250W graphics card (for comparison, the Raspberry Pi has a power consumption of approximately 1.2W).

Hyperparameters selection. ODGI's performance are strongly influenced by the crop extraction process that occurs when transiting from one stage to the next: Too many redundant patches leads to slower runtimes, while extracting too few patches reduces the coverage of relevant regions and may hurt detection accuracy. Besides the number of extracted crops, three additional parameters influence this extraction process: $\tau^{\text{test}}_{\text{low}}$, $\tau^{\text{test}}_{\text{high}}$, and $\tau^{\text{test}}_{\text{nms}}$. In our experiments, we select these thresholds by measuring model performance on a held-out validation set.

More precisely, for each model, for the range $\gamma_1 \in [1, 10]$, we perform a parameter sweep over the ranges $\tau_{\text{low}} \in \{0., 0.1, 0.2, 0.3, 0.4\}$, $\tau_{\text{high}} \in \{0.6, 0.7, 0.8, 0.9, 1.0\}$, and $\tau_{\text{nms}} \in \{0.25, 0.5, 0.75\}$. Note that network training is independent from these parameters as discussed in Section 3.1.2. Therefore the sweep can be done efficiently using pre-trained stages ϕ_1 and ϕ_2, changing only the patch extraction process. Our main observations on varying these parameters are as follows:

- (i) $\tau^{\text{test}}_{\text{low}}$ usually lies in $\{0, 0.1\}$. This indicates that the low confidence patches are generally true negatives that need not be filtered out.

- (ii) $\tau_{\text{high}} \in \{0.8, 0.9\}$ for VEDAI and $\tau_{\text{high}} \in \{0.6, 0.7\}$ for SDD. This reflects intrinsic properties of each dataset: VEDAI images contain only few objects which are easily covered by the extracted crops. Therefore it is always beneficial for the detection accuracy to refine these predictions, even when they are predicted as individuals with high confidence, hence a high value of τ_{high}. In contrast, on the more challenging SDD, ODGI instead focuses refinement efforts on groups and lower-confidence individuals, hence tend to accept more individual objects' predictions after the first stage.

- (iii) $\tau_{\text{nms}}^{\text{test}}$ is usually equal to 0.25, which encourages non-overlapping patches and reduces the number of redundant predictions. Furthermore, it is often higher (0.5) at low input resolution (256 pixels), reflecting the fact that candidate relevant regions are more likely to overlap at lower resolutions.

3.1.5 Ablation study

We report results of ablation experiments to highlight the influence of the proposed contributions. In particular, we investigate the impact of predicting groups rather than individual objects, as well as learning offsets to rescale relevant regions passed to intermediate stages. Finally, we discuss the memory requirements of the proposed model in more details, as well as the influence of the number of refinement stages.

Grouped instances. We first compare ODGI against a variant without group information, ODGI$^{\text{singles}}$: In other words, we drop the loss term $\mathcal{L}_{\text{groups}}$ in (3.1) and ignore group flags in the transition between stages. From Table 3.3 (row *no groups*), we observe that ODGI consistantly improves over ODGI$^{\text{singles}}$ in terms of detection accuracies for equivalent number of crops, γ_1. This also holds consistently for different values of γ_1.

Furthermore, we observe that both methods behave qualitatively differently. Intuitively, detecting groups in early stages is more efficient as it allows for extracting larger image regions, accounting for multiple individuals. In contrast, ODGI$^{\text{singles}}$ usually needs to extract more crops, and thus requires more computations, to achieve similar coverage as ODGI. On the other hand, ODGI$^{\text{singles}}$ focuses on detecting individuals, hence the individual boxes early-exiting the pipeline after the first stage are usually of better precision. In other words, ODGI makes better use of the refinement stage which is reflected by the results in Table 3.3.

Offsets. The rescaling step introduced in Section 3.1.3 strongly impacts the model's detection accuracy. On one hand, if the learned scale offsets are too low, the extracted patches might cut objects rather than englobe them, hence leading to missed detections. On the other hand, if they are too high, the extracted regions become very large, making detection harder for subsequent stages as their input resolution is smaller. We perform two sets of ablation experiments to analyze the influence of this rescaling:

First, instead of using learned offsets we test the model with offset values fixed to $\frac{2}{3}$, *i.e.* 50% expansion of the bounding boxes, which corresponds to the value of the target offsets margin δ we chose for training ODGI. Our experiments in Table 3.3 show that this variant is inferior to ODGI, confirming that the model benefits from learning offsets tailored to its predictions. Second, we entirely ignore the rescaling step during the patch extraction step (row *no offsets*). This affects the MAP even more negatively: Extracted crops are generally localized close to the relevant objects, but do not fully enclose them. Consequently, the second stage retrieves partial objects, but with very high confidence, resulting in strong false positives predictions. In this case, most correct detections emerge from stage 1's early-exit predictions, hence increasing γ_1, *i.e.* passing more crops to later stages, does not improve the MAP in this scenario.

SDD	$\gamma_1 = 1$	$\gamma_1 = 3$	$\gamma_1 = 5$	$\gamma_1 = 10$
ODGI-tt 512-256	**0.245**	**0.361**	**0.415**	**0.457**
no groups	0.225	0.321	0.380	0.438
fixed offsets	0.199	0.136	0.246	0.244
no offsets	0.127	0.127	0.125	0.122
ODGI-tt 256-128	**0.128**	**0.243**	**0.293**	**0.331**
no groups	0.122	0.229	0.282	0.326
fixed offsets	0.088	0.136	0.150	0.154
no offsets	0.030	0.040	0.040	0.040

Table 3.3: MAP@0.5 results comparing two ODGI settings with three ablation variants, *no groups*, *fixed offsets* and *no offsets* (see text) on the SDD dataset.

Memory requirements. ODGI stages are applied consequently, hence only one network needs to live in memory at a time. However, having independent networks for each stage can still be prohibitory when working with very large backbone architec-

MAP0.5 ($\gamma = 6$)	no sharing	sharing
ODGI-tt 512-256	0.429	0.385
ODGI-tt 256-128	0.307	0.197
ODGI-tt 128-64	0.098	0.051

Table 3.4: Weights sharing experiments on SDD for the ODGI teeny-tiny models. Sharing weights halves the number of model parameters but hurts the model performance

tures. To palliate this problem, we also experimented with sharing weights across different stages: In this setting, we have only one backbone network, common to the first and second stage, while only the last fully connected layer is specific to each stage. We make two main observations: *(i)* In that setting, the delayed training schedule is particularly beneficial to the final model's performance, and, *(ii)* while this decreases the number of model parameters, weights sharing significantly decreases the detection performance. See Table 3.4 for quantitative results.

A likely explanation is that the data distribution in stage 1 and stage 2 are drastically different in terms of object resolution and distribution, effectively causing a *domain shift*. In fact, the visual appearance of input images to stage 1 (small relevant regions in large sparse images) and stage 2 (often densely covered patch pre-selected by stage 1) are drastically different. This can be seen from the qualitative examples in Figure 3.7 and following. This introduces a visual domain shift between the two stages which explains why sharing representations for these two domains might not be adequate.

3.1.6 Analyzing the flexibility of ODGI

Adding more refinement stages potentially improves the model accuracy but comes at a high computational cost. Intuitively, refinement stages benefit the speed-vs-accuracy trade-off when the following two criteria are met: *First*, a low number of non empty cells; This directly ties to the number of extracted crops, thus to the number of feed-forward passes through intermediate stages. *Second*, a small average group size: Smaller extracted regions lead to increased resolution once rescaled to the input size of the next stage, making the detection task for subsequent stages effectively easier.

Datasets of aerial views such as VEDAI [Razakarivony and Jurie, 2015] or SDD [Robicquet *et al.*, 2016] meet these criteria: They contain small-sized group structures in large sparse areas. This is a typical scenario where the proposed refinement stages

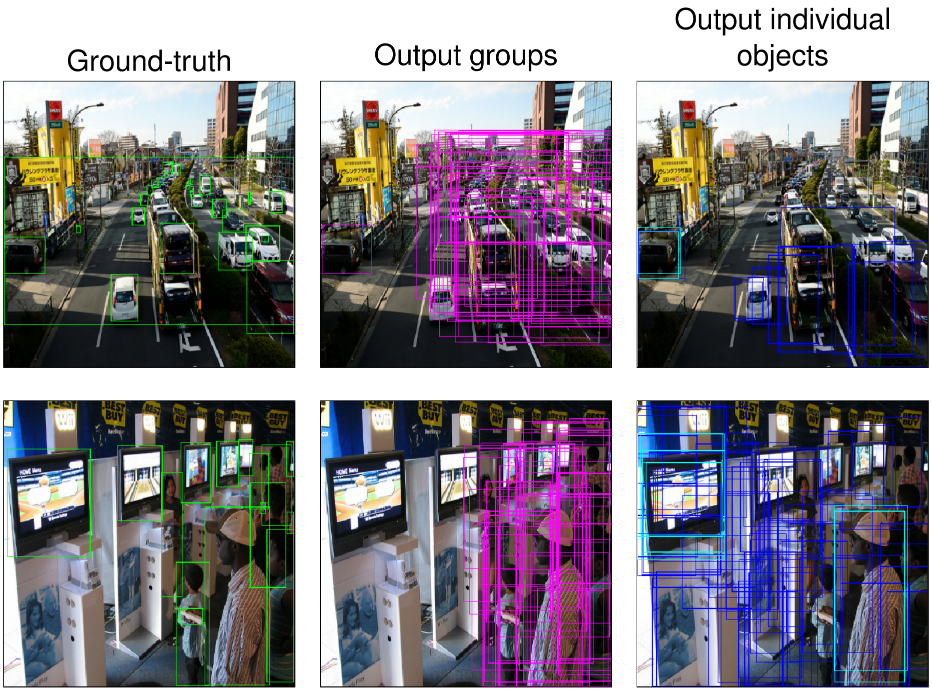
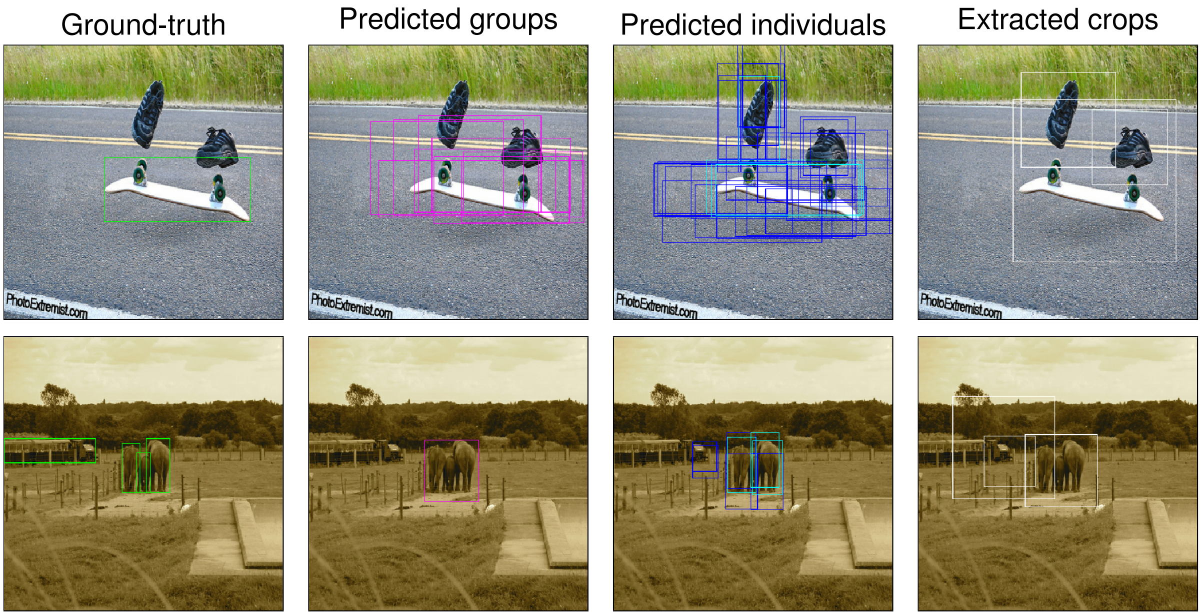


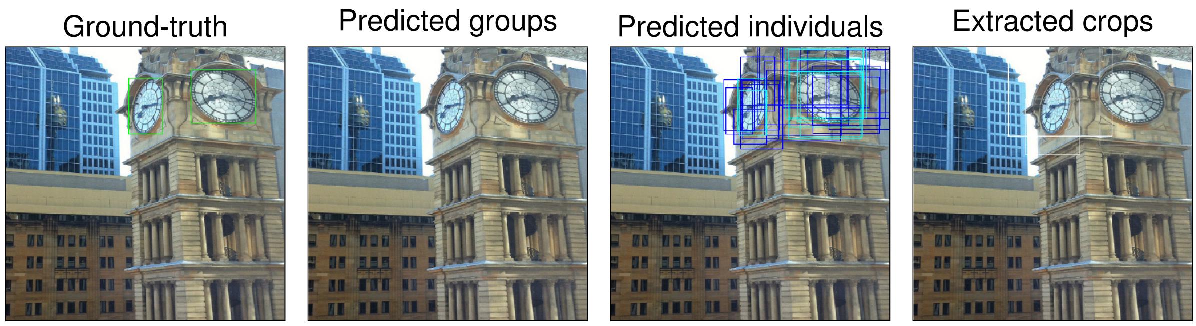
Figure 3.4: Example of stage 1's outputs when training a simple ODGI model on the MS-COCO dataset. Here, only groups and individuals with confidence $c > 0.25$ are shown. Cyan boxes indicate individual objects' predictions with high confidence $c > 0.75$. We observe that the dense overlapping of ground-truth objects results into numerous and large groups that provide limited "time vs detection boost" improvement.

on groups improve the speed-accuracy trade-off. We find that for the datasets used in our experiments $S = 2$ is sufficient, as regions extracted by the first stage typically exhibit a dense distribution of large objects. Nonetheless, we expect the case $S > 2$ to be beneficial for very large, *e.g.* gigapixel images.

At the other end of the spectrum we find classical benchmarks such as MS-COCO. They differ drastically in terms of objects' size and distribution across the image, as can be seen quantitatively from the data statistics reported in the introduction in Table 3.1. In such scenarios, using only one stage suffices, and thus ODGI collapses to using a single-shot detector. In fact, when objects overlap too densely, the notion of groups becomes fuzzy and the extracted relevant regions are often quite large and numerous. We show an example of this in Figure 3.4. Consequently, the crops received by the second stage often contain a detection problem almost as hard as the first stage. Furthermore they come in large numbers, which triggers numerous feed-forward passes of the second stage. When the object distribution is sparser (see Figure 3.5), the detected groups define relevant localized image regions, as was the case in the aerial view dataset settings. However the objects being quite large to begin with, they



(a) In sparser scenes, fewer groups are detected but they yield large image regions as the objects themselve are quite large to begin with.



(b) Example with large objects that do not overlap: no groups detected.

Figure 3.5: Qualitative results on the MS-COCO dataset: For sparse distributions, the learned groups lead to informative and well localized relevant image regions. However, because the objects are large themselves, they are often detected well after only one stage, which reduces the impact of a potential refinement stage.

are often well detected as individuals. As a result, feeding the extracted crops to the second stage might help refining the bounding box coordinates, however the potential improvement is limited, relatively to the cost of an additional feed-forward pass.

3.1.7 Conclusions

We introduce ODGI, a novel cascaded scheme for object detection that identifies *groups of objects* in early detection stages, and refines them in later stages *as needed*: Consequently, *(i)* empty image regions are discarded, thus saving computations especially in situations with heterogeneous object density, such as aerial imagery, and *(ii)* groups are easier to detect at lower resolutions, as they are typically larger structures than individual objects . Furthermore, ODGI can be easily added to off-the-shelf backbone networks commonly used for single-shot object detection: In extensive experiments, we show that the proposed method offers substantial computational savings without sacrificing accuracy.

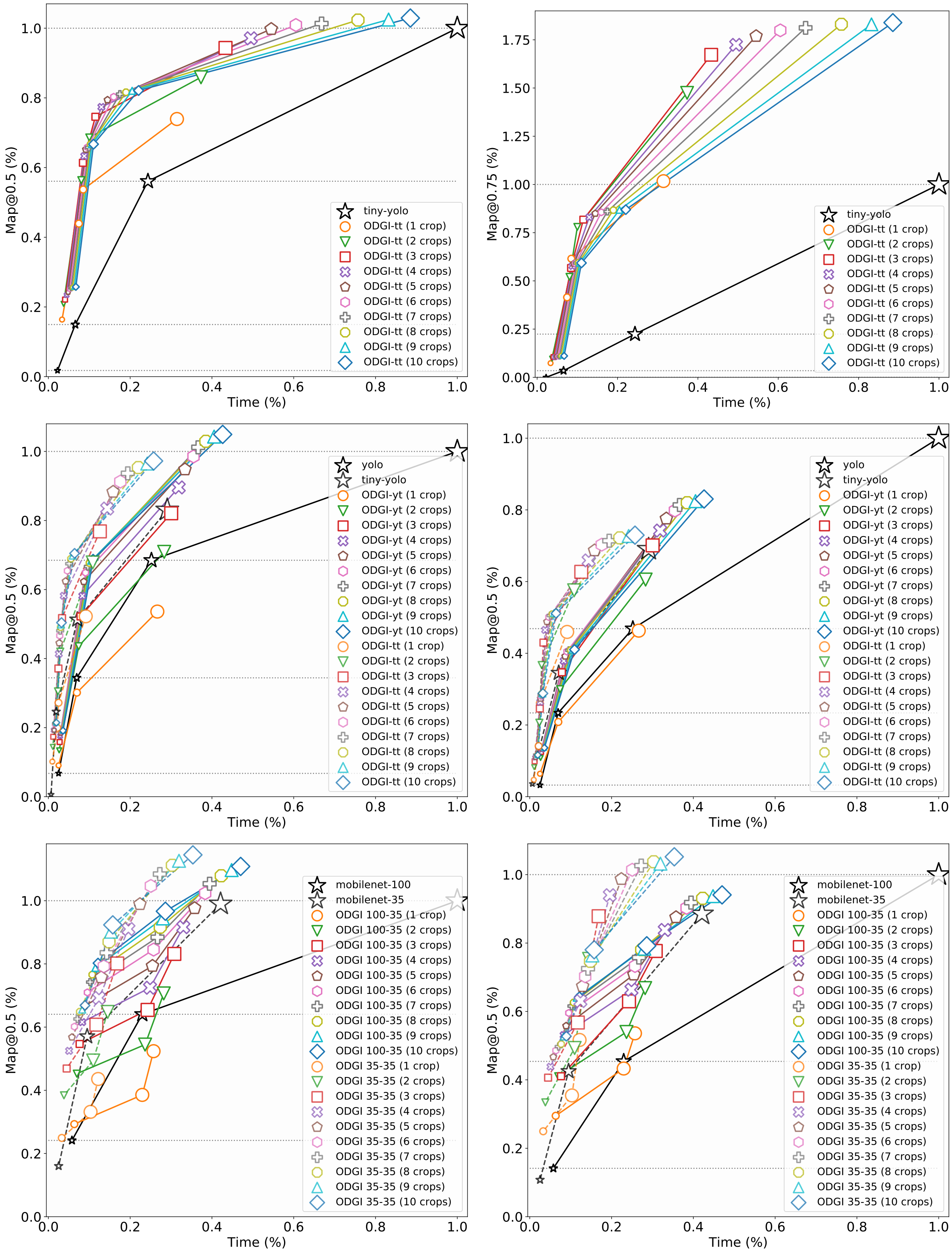
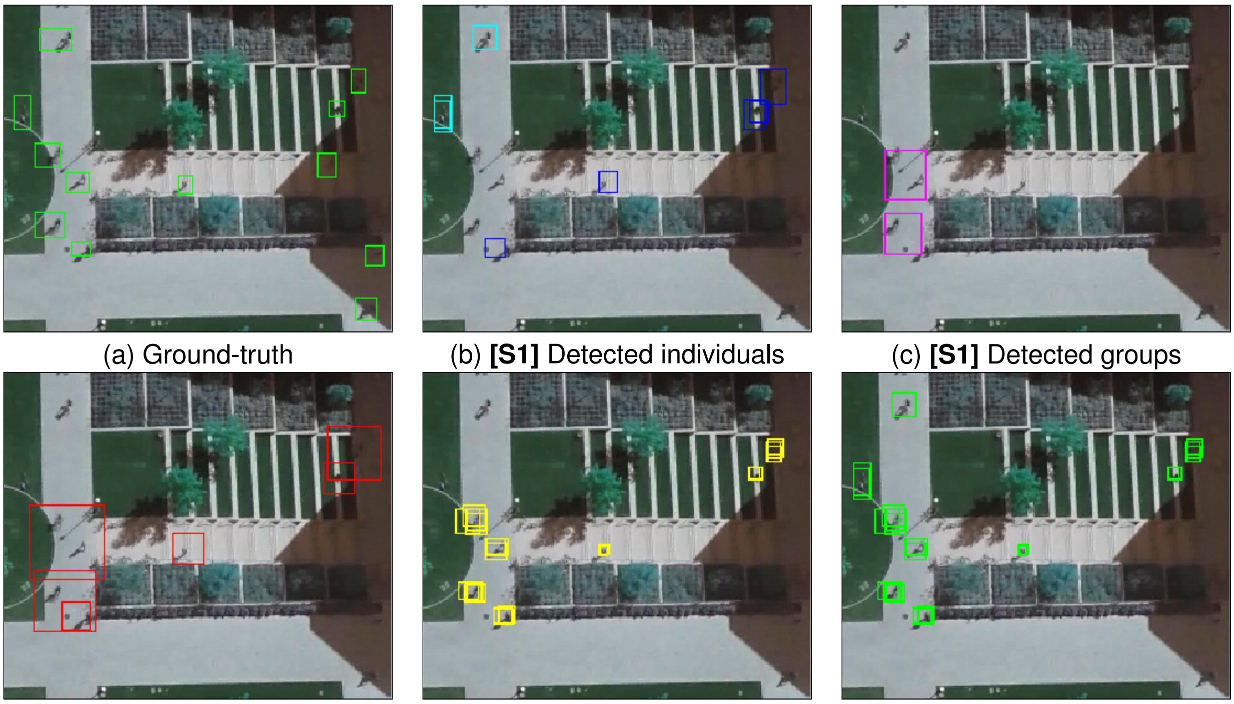


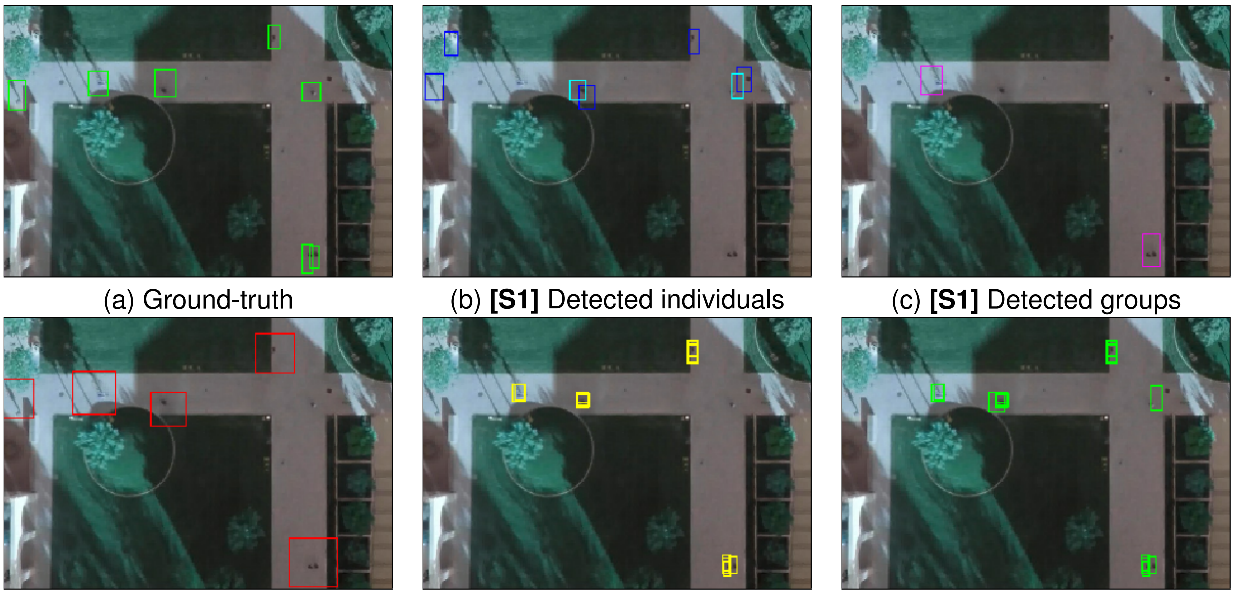
Figure 3.6: Plots of MAP0.5 (left) and MAP0.75 (right) versus runtime (CPU) for all number of extracted crops in ODGI. This is an extension of the plots in Figure 3.3



(a) Ground-truth (b) **[S1]** Detected individuals (c) **[S1]** Detected groups

(d) Crops passed to stage 2 (e) **[S2]** Detected objects (f) ODGI: all detections

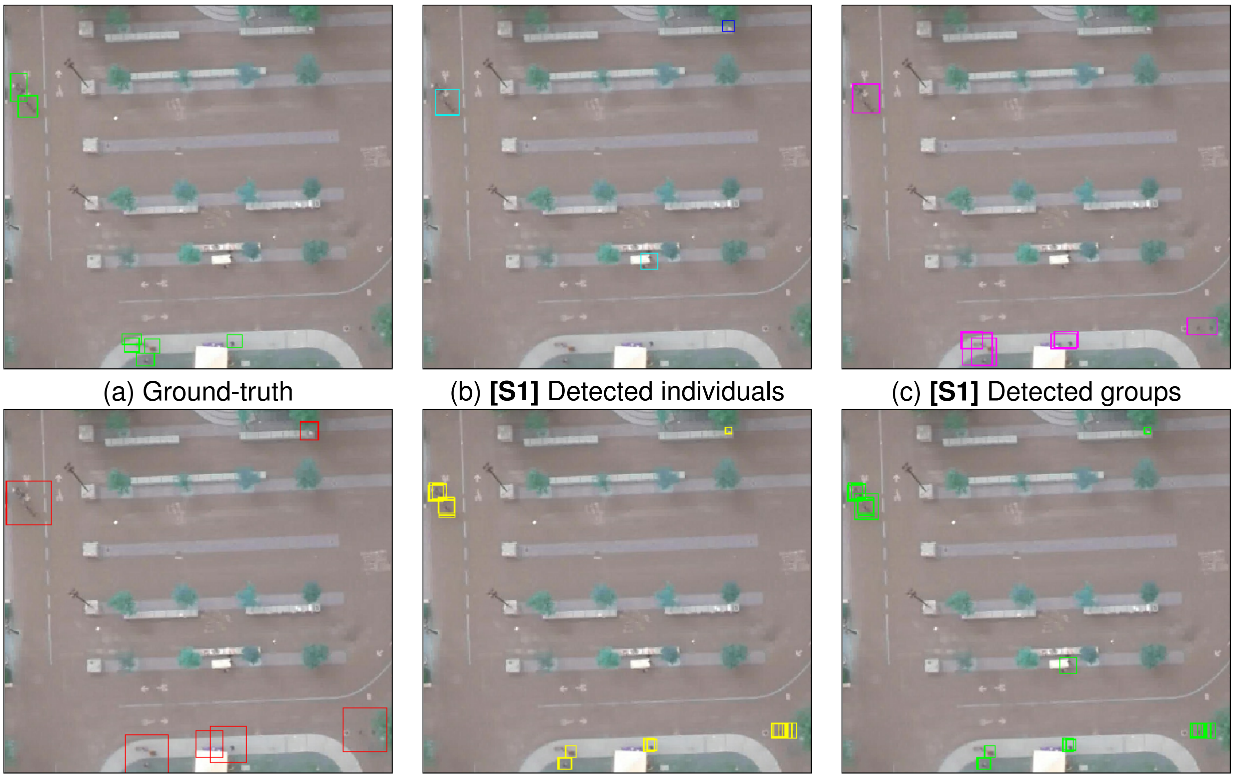
Figure 3.7: Qualitative results for ODGI. Best seen on PDF with zoom. No filtering step was applied, but for readability we only display boxes predicted with confidence at least 0.5. **S1** and **S2** indicate outputs relative to the first and second stage respectively. In (b), cyan boxes are individual object predictions with a high confidence ($c \geq \tau_{\text{high}}$).



(a) Ground-truth (b) **[S1]** Detected individuals (c) **[S1]** Detected groups

(d) Crops passed to stage 2 (e) **[S2]** Detected objects (f) ODGI: all detections

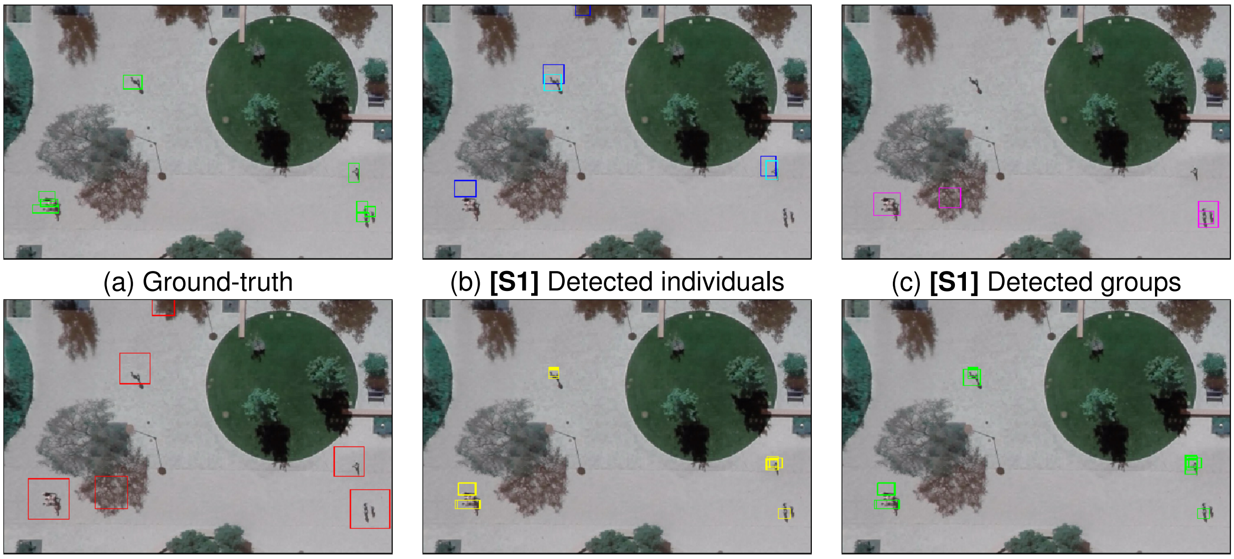
Figure 3.8: Qualitative results for ODGI (continued).



(a) Ground-truth (b) **[S1]** Detected individuals (c) **[S1]** Detected groups

(d) Crops passed to stage 2 (e) **[S2]** Detected objects (f) ODGI: all detections

Figure 3.9: Qualitative results for ODGI (continued).



(a) Ground-truth (b) **[S1]** Detected individuals (c) **[S1]** Detected groups

(d) Crops passed to stage 2 (e) **[S2]** Detected objects (f) ODGI: all detections

Figure 3.10: Qualitative results for ODGI (continued).

3.2 Probabilistic Image Colorization

Image colorization is the task of recovering missing color information from a given input grayscale image. Besides its direct applications, *e.g*. photo editing or restoring historical images, colorization is a powerful self-supervised learning technique: Grayscale/color image pairs can be created automatically from readily available color images, which allows for abundant and easy-to-collect training data; Moreover it has been shown that intermediate representations learned by colorization models carry meaningful semantics that can be exploited for scene understanding tasks such as image classification or semantic segmentation [Zhang *et al*., 2016; Larsson *et al*., 2017].

Colorizing natural images requires a model that not only learns about different object categories, but also captures their variety in appearance: For instance, while the sky is almost certainly always blue, the color of other objects can have a much higher entropy, *e.g*. buses have different color depending on the geographic location. Previously proposed colorization models are able to capture the evident mappings abounding in the training data, *e.g*., blue sky, but often fail to capture the multiple modes of more complex color distributions, leading to desaturated images. In fact, most state-of-the-art colorization techniques do not offer a proper image sampling framework as they only model pixelwise color distributions. Consequently, they often lack two main appealing properties: *(i) diversity*, *i.e*. being able to produce several plausible colorizations, as there is generally no unique solution, and *(ii) color vibrancy* of the colorized samples, as they should display proper level of saturation and contrast like natural images, not look desaturated. Recent work [Zhang *et al*., 2016] addresses these caveats by *(i)* treating colorization as a classification task to avoid the problem of using a regression objective which leads to unimodal, and thus, desaturated predictions, and *(ii)* introducing rebalancing weights to favor rare colors present in natural images and more difficult to predict. In contrast, we argue that the key issue is that current state-of-the-art models fail to properly capture the joint pixel colors distribution of natural images. We instead propose a method that explicitly models pixel interactions and does not require any *ad-hoc* modifications of the training procedure.

Our proposed model, Probabilistic Image Colorization (PIC) models the joint distribution of color intensities over all pixels by leveraging recent advances in *autoregres-*

sive models [van den Oord *et al.*, 2016b; Kingma *et al.*, 2016; Salimans *et al.*, 2017; van den Oord *et al.*, 2016c] for image generative modeling. Specifically, our architecture is composed of two networks: A deep feed-forward network first maps the input grayscale image to a lower dimensional embedding which encodes plausible color information, much like current state-of-the-art colorization schemes. This embedding is then fed to an autoregressive network, which in turn predicts a proper distribution of the image chrominance conditioned on the grayscale input. This step relies on the observation that estimating the underlying true likelihood of the joint pixel color intensities can be made tractable by *decomposing the computation into smaller chained subproblems*. In particular, this allows us to train the model to directly maximize the data likelihood, and also yields a quantitative metric that can be used to compare models.

In Section 3.2.2, we introduce the theoretical framework behind the autoregressive component of our model, as well as our training and inference procedures. Modeling the full multimodal joint distribution over color values offers a solution to the *diversity* problem, as it provides us with a simple, computationally efficient, and yet powerful probabilistic framework for generating different plausible colorizations. Furthermore, the model likelihood can be used as a principled quantitative evaluation measure to assess the model performance. We then report experimental results in Section 3.2.4, including qualitative comparison to several baselines: We show that our proposed model also tackles the *vibrancy* problem as it generally produces vivid samples, without any *ad hoc* modifications of the training procedure.

3.2.1 Related work

Automatic image colorization has been a goal of image processing and computer vision research since at least the 1980s, after movie studies started releasing re-colorized movies from the black-and-white era [Novak, 1972]. Because manually colorizing every frame of a movie is tedious and expensive work, semi-automatic systems soon emerged, *e.g*. based on the manual colorization of key frames followed by motion-based color propagation [Markle and Hunt, 1988]. Subsequently, techniques that required less and less human interaction were developed, *e.g*., requiring only user scribbles [Levin *et al.*, 2004; Kawulok and Smolka, 2010], reference color images [Charpiat *et al.*, 2008; Morimoto *et al.*, 2009], or scene labels [Deshpande *et al.*, 2015].

Recent successful fully automatic approaches make use of deep architectures [Iizuka *et al.*, 2016; Larsson *et al.*, 2016; Zhang *et al.*, 2016; Isola *et al.*, 2017; Deshpande *et al.*, 2017; Cao *et al.*, 2017b]: A straight-forward approach is to train a convolutional feed-forward model to independently predict a color value for each pixel [Iizuka *et al.*, 2016; Larsson *et al.*, 2016; Zhang *et al.*, 2016] conditioned on the grayscale input.

By design, these techniques do not capture crucial interactions between pixel colors of natural images. As a result, probabilistic sampling often yields color artifacts, *e.g.* pixels of a same regions taking different color values as shown in Figure 3.11. Furthermore, predicting the mode or expectation of the learned distribution often results in grayish desaturated colorizations which has so far been addressed with ad-hoc training heuristics, such as class rebalancing [Zhang *et al.*, 2016], to encourage rare colors.

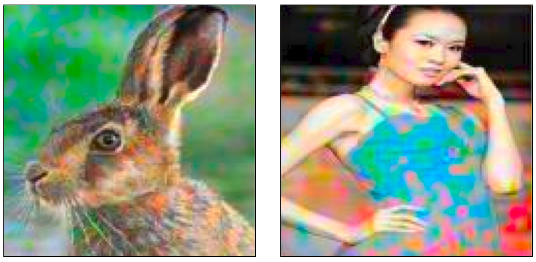


Figure 3.11: Feed-forward colorization models consider pixels independently which may lead to high-level artifacts such as incoherent colorizations of continuous regions of the image.

An additional limitation of the models discussed above is their lack of *diversity*. They only produce the most likely colored version from each grayscale image, despite the fact that there are typically multiple plausible colorizations. Instead, recent work has investigated the use of *image generative models*, which are designed to produce several plausible samples, for colorization: For instance, [Isola *et al.*, 2017; Cao *et al.*, 2017b] propose to train a colorization model using generative adversarial networks (GANs) [Goodfellow *et al.*, 2014]. GANs, however, are known to suffer from unstable training and lack of a consistent quantitative metric to compare models. To our knowledge, the only work besides ours aiming at representing a fully probabilistic multi-modal joint distribution of pixel colors is [Deshpande *et al.*, 2017]. It relies on the variational autoencoder (VAE) framework [Kingma and Welling, 2014], which, however, only estimates a *lower bound* on the likelihood and tends to produce more blurry outputs than other image generating techniques. In contrast, the autoregressive network [van den Oord *et al.*, 2016c; Salimans *et al.*, 2017] we employ directly models the true data likelihood and is able to produce crisp high-quality and diverse colorizations.

Concurrent to this work is [Guadarrama *et al.*, 2017], a closely related work in which the authors also tackle the image colorization task using recent advances in probabilistic autoregressive models.

3.2.2 Autoregressive models for image colorization

In this section, we introduce the technical background of the autoregressive model framework, and how we make use of it to build the proposed architecture, **P**robabilistic **I**mage **C**olorization model (PIC), for image colorization.

Estimating the data likelihood. Our goal is to predict a probabilistic distribution of an image's color channels, given an input grayscale view. We assume that images are encoded in the LAB color space, which has three channels: the luminance channel (L) and the two chrominance channels (a and b). Denoting by X the random variable associated to the space of color images, we denote by X^L and X^{ab} the projection of image X to its luminance channel and chrominance channels respectively. By convention, the chrominance in a LAB triplet belongs to the discrete range $\mathcal{C} = [-127; 128] \times [-128; 127]$. Consequently, each pixel in X^{ab} can take $|\mathcal{C}| = 256 \times 256 = 65536$ possible values. With these notations, our aim is to model the conditional distribution $p(X^{ab}|\ X^L)$ from a given training set of color images. This is a challenging and computationally demanding task, as X^{ab} is a high dimensional object with a rich internal structure.

To tackle this intractable estimation problem, we rely on recent advances in autoregressive probabilistic models [van den Oord *et al.*, 2016c; Salimans *et al.*, 2017]. *The main insight is that the distribution of interest can be decomposed into elementary per-pixel conditional distributions using the chain rule of probability theory.* Without loss of generality, we assume that images are indexed in raster scan order, *i.e.* from top to bottom and from left to right, such that the value of the i-th pixel in image X is denoted as X_i. Denoting by n the number of pixels in image X, the chain rule yields:

$$p(X^{ab}|\ X^L) = \prod_{i=1}^{n} p(X_i^{ab}|\ X_1^{ab}, \ldots, X_{i-1}^{ab};\ X^L). \tag{3.10}$$

Note that Equation 3.10 makes no assumptions on the modeled distribution. It is merely a direct application of the chain rule of probability theory that provides us with a tractable estimate of the likelihood: Instead of directly estimating the full joint distribution ($|\mathcal{C}|^n$ possibilities), we only need to iteratively model smaller factors (n factors, each with C outputs) conditioned on the previously sampled pixels: At training time, all color information $X^{ab}_{1\dots n}$ is known, thus the model can be efficiently trained by learning all factors in parallel, conditioned on the observed ground-truth colors. At test time, we draw samples from the joint distribution pixel by pixel using the same chain sequential structure: First sampling x^{ab}_1 from $p(X^{ab}_1|\ X^L)$, then iteratively sampling x^{ab}_i from $p(X^{ab}_i|\ X^{ab}_1 = x^{ab}_1, \dots X^{ab}_{i-1} = x^{ab}_{i-1}; X^L)$ for all i in $\{2 \dots n\}$.

We model the factors in (3.10) using a fully-convolutional autoregressive network, denoted as f_θ, which outputs a vector of normalized probabilities over the set $\mathcal{C}$ of all possible chrominance (a, b) values for each pixel. Additionally, to model the conditional dependency on the input grayscale image X^L we introduce a deep convolutional neural network e_ϕ, which acts as an encoder and produces a real-valued vector embedding of X^L. To summarize, each factor in (3.10) has the following functional form:

$$p(X^{ab}_i|\ X^{ab}_i, \dots, X^{ab}_{i-1};\ X^L) = f_\theta(X^{ab}_1, \dots, X^{ab}_{i-1};\ e_\phi(X^L)) \tag{3.11}$$

We discuss architecture choices for both f_θ and e_ϕ in Section 3.2.3.

Modeling the color distribution. By design, the autoregressive network f_θ outputs a probability distribution over the set of discrete color values $\mathcal{C}$. The standard way to encode such a distribution is to parametrize f_θ as a multi-class classifier which outputs a score for each of the possible color values in $\mathcal{C}$ and then apply the softmax operation to obtain a normalized probability distribution. In our case, however, the output space is very large (65536 values per pixel), hence this standard approach will result in a very slow convergence of the training procedure and will require a vast amount of data to generalize. It is possible to alleviate this shortcoming by quantizing the color space at the expense of a slight drop in colorization accuracy and possible visible quantization artifacts. However, this still results in a large number of classes, typically a few hundreds. Further heuristics, such as soft label encoding [Zhang *et al.*, 2016], are then required to speed up the training and avoids slow convergence.

Instead, we approximate the output distribution from Equation 3.11 with a mixture of logistic distributions, as described in [Salimans *et al.*, 2017]. In other words, f_θ

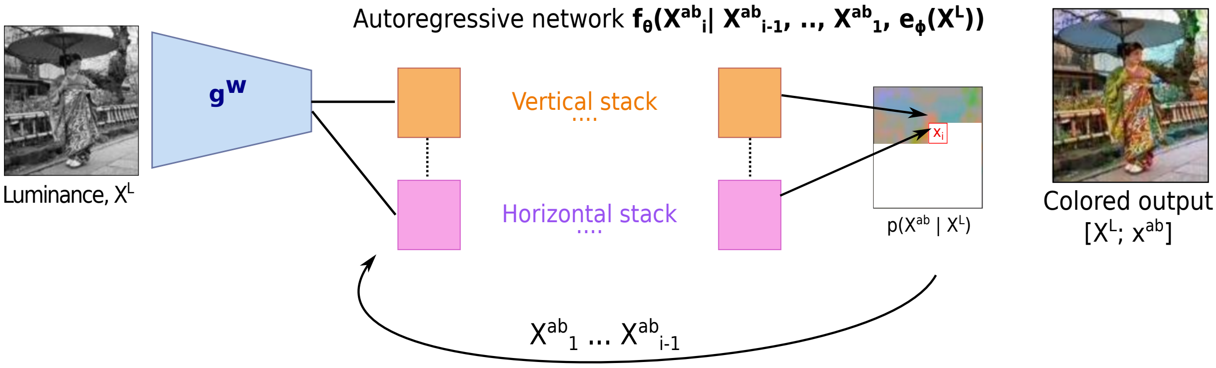


Figure 3.12: Overview of the proposed PIC model: The input grayscale image is first fed to an embedding network e_ϕ, whose output is then passed through the autoregressive network f_θ that predicts a probability distribution for chrominance X^{ab}. At training time, the model is trained by maximizing the likelihood of the initial ground-truth color images under its output distribution. At test time, we sample color information from this distribution pixel by pixel, conditioning the decision on the previously sampled pixel colors. The resulting colorized image is obtained by concatenating the input grayscale luminance with the generated chrominance in the LAB color space.

outputs the mixture distribution parameters, *i.e.* the mixture weights and the first and second-order statistics of each logistic component; We use 10 mixture components in practice. This model is powerful enough to represent a multimodal discrete distribution over all values in $\mathcal{C}$, and requires less than 100 output values per pixel to encode, which is significantly fewer than for the standard discrete distribution representation (roughly 65000 possible values for each pixel). Furthermore, since the representation is partially continuous, the model can also exploit the distance between chrominance values in $\mathbb{R}$, contrarily to a purely categorical encoding, which results in faster convergence.

3.2.3 Model architecture and training procedure

We present a high-level overview of our model in Figure 3.12. It contains two major components: the embedding network e_ϕ and the autoregressive network f_θ. Intuitively, we expect that e_ϕ, which only has access to the grayscale input, produces an embedding that encodes information about plausible image colors based on the semantics available in the grayscale image. The autoregressive network f_θ then makes use of this lower dimensional embedding to produce the final chrominance distribution for each pixel, while being able to model complex interactions between image pixels.

Architecture. Our design choices for parametrizing networks e_ϕ and f_θ are motivated by [Salimans *et al.*, 2017], who report state-of-the-art results for the challenging and related problem of natural image modeling. In particular, both networks rely on *gated residual blocks* as their main building components: Each residual block has 2 convolutions with 3×3 kernels, a skip connection [He *et al.*, 2016] and gating mechanism [van den Oord *et al.*, 2016b; Salimans *et al.*, 2017]. The convolutions are preceded by concatenated [Shang *et al.*, 2016] exponential linear units (ELU) non-linearities [Clevert *et al.*, 2016] and are parametrized with the weight normalization technique proposed in [Salimans and Kingma, 2016]. If specified, the first convolution of the residual block may have a dilated receptive field [Yu and Koltun, 2016]: We use dilation to increase the network's field of view without reducing its spatial resolution.

Building on these residual blocks, we define the embedding network e_ϕ as a standard feed-forward deep convolutional neural network consisting in a sequence of gated residual blocks and standard convolutions with stride 2 to decrease spatial resolution. For parametrizing f_θ, we use the Pixel-CNN**++** architecture from [Salimans *et al.*, 2017]. f_θ takes as input a fixed-size image, representing the previously sampled pixel color values (padded with 0 for unkown pixel values). To model the conditioning on the grayscale input, X^L, we bias the output of the first convolution of every residual block by the learned embedding $e_\phi(X^L)$. On a high level, the network consists of two flows of residual blocks with constant spatial resolution, which encode the horizontal and vertical pixel dependency respectively. In each flow, the output of every convolution is appropriately shifted and masked to achieve the sequential dependency defined in Equation 3.11, *i.e.* to guarantee that the prediction for pixel i only depends on the input values for pixels $1, \ldots, i-1$. The horizontal and vertical dependencies are modeled separately to avoid "blind spot" problems in the field of view that occur when relying on masked convolutions only, as described in [Salimans *et al.*, 2017]. For further details, refer to our implementation [Royer and Kolesnikov, 2017].

Spatial chromatic subsampling. The human visual system resolves color less precisely than luminance information [Van der Horst and Bouman, 1969]. We exploit this fact by performing *chromatic subsampling*, *i.e.* we model the chrominance channels at a lower resolution than the input luminance, and then rescale it using bilinear upsampling to generate the final colorized image. Image compression schemes such as

JPEG or previously proposed techniques for automatic colorization also make use of similar chromatic subsampling. This allows us to reduce computational and memory requirements without losing perceptual quality.

Optimization. We train the model parameters (θ, ϕ) by minimizing the negative log-likelihood of the chrominance channels under the predicted distribution. In other words, following (3.10), we minimize the following loss function over every training image x:

$$\mathcal{L}_{\text{PIC}}(x) = -\log p(x^{ab}|x^L) = -\sum_{i=1}^{n} \log f_\theta(x^{ab}_{1:i-1};\ e_\phi(x^l)) \tag{3.12}$$

Following previous implementations of autoregressive models [Salimans *et al.*, 2017], we use the Adam optimizer [Kingma and Ba, 2015] with an initial learning rate of $1\text{e-}3$, momentum of 0.95 and second momentum of 0.9995. We also apply Polyak averaging over the model parameters [Polyak and Juditsky, 1992].

3.2.4 Experiments

In this section we present quantitative and qualitative evaluation of of our proposed colorization model, PIC. We evaluate our model on two challenging image datasets, namely CIFAR-10 and ImageNet ILSVRC-2012. We also qualitatively compare our method to previously proposed colorization approaches and perform additional studies to better understand various components of our model. Our Tensorflow implementation and pre-trained models are publicly available [Royer and Kolesnikov, 2017].

CIFAR-10 experiments. We first study the colorization abilities of our method on CIFAR-10 [Krizhevsky and Hinton, 2009], which contains 50000 training images and 10000 test images of 32x32 pixels, categorized in 10 semantic classes. We fix the architecture of the embedding network e_ϕ as specified in Table 3.5 (left). For the autoregressive network f_θ we use 4 residual blocks and 160 output channels for every convolution. We subsample the spatial chromatic resolution by a factor of 2, *i.e.* we model the color channels at a resolution of 16x16 pixels, and use bilinear upsampling to recover the full 32x32 pixel resolution. We train the resulting model as explained in Section 3.2.3 with batch size of 64 images for 150 epochs. We also decay the learning rate after every training iteration with constant multiplicative rate 0.99995.

CIFAR-10 embedding $e_\phi(X^L)$			
Operation	Res.	Width	D
Conv. 3x3/1	32	32	–
Resid. block $\times$ 2	32	32	–
Conv. 3x3/2	16	64	–
Resid. block $\times$ 2	16	64	–
Conv. 3x3/1	16	128	–
Resid. block $\times$ 2	16	128	–
Conv. 3x3/1	16	256	–
Resid. block $\times$ 3	16	256	2
Conv. 3x3/1	16	256	–

ILSVRC-2012 embedding $e_\phi(X^L)$			
Operation	Res.	Width	D
Conv. 3x3/1	128	64	–
Resid. block $\times$ 2	128	64	–
Conv. 3x3/2	64	128	–
Resid. block $\times$ 2	64	128	–
Conv. 3x3/2	32	256	–
Resid. block $\times$ 2	32	256	–
Conv. 3x3/1	32	512	–
Resid. block $\times$ 3	32	512	2
Conv. 3x3/1	32	512	–
Resid. block $\times$ 3	32	512	4
Conv. 3x3/1	32	512	–

Table 3.5: Architecture of the embedding network e_ϕ for the CIFAR-10 (left) and ILSVRC-2012 (right) setting. In the Operation column the notation "$\times\ k$" means that the corresponding residual block is repeated k times, and the notation "$/k$" indicatess the stride of convolution operations. The second column, Res., contains each layer's spatial resolution, Width is the number of output channels and finally, D is the dilation rate if using dilated convolutions in the residual block.

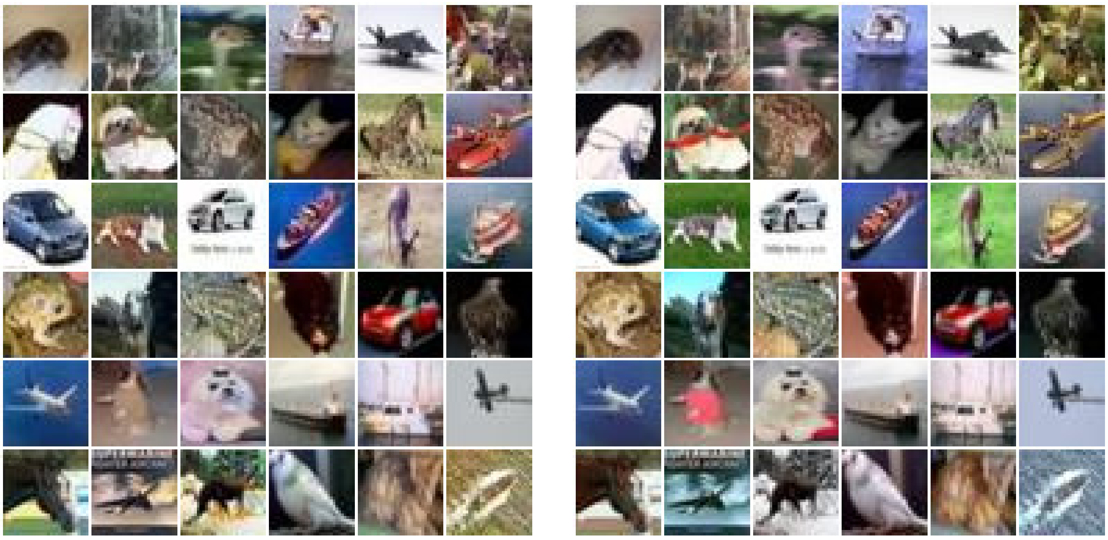


Figure 3.13: Colorized image samples from our model (left) and the corresponding original CIFAR-10 images (right). Images are selected randomly from the test set.

In Figure 3.13 we visualize random test images colorized by PIC (left) and the corresponding real CIFAR-10 color images (right). We note that the samples produced by PIC appear to have natural colors and are hardly distinguishable from the real ones. This speaks in favor of our model being adequate for modeling the color distribution of natural images. Furthermore, because autoregressive probabilistic models compute the exact data likelihood, we can also report that PIC achieves a negative log-likelihood of **2.72**, measured in bits-per-dimension on the CIFAR test set. Intuitively, this metric in-

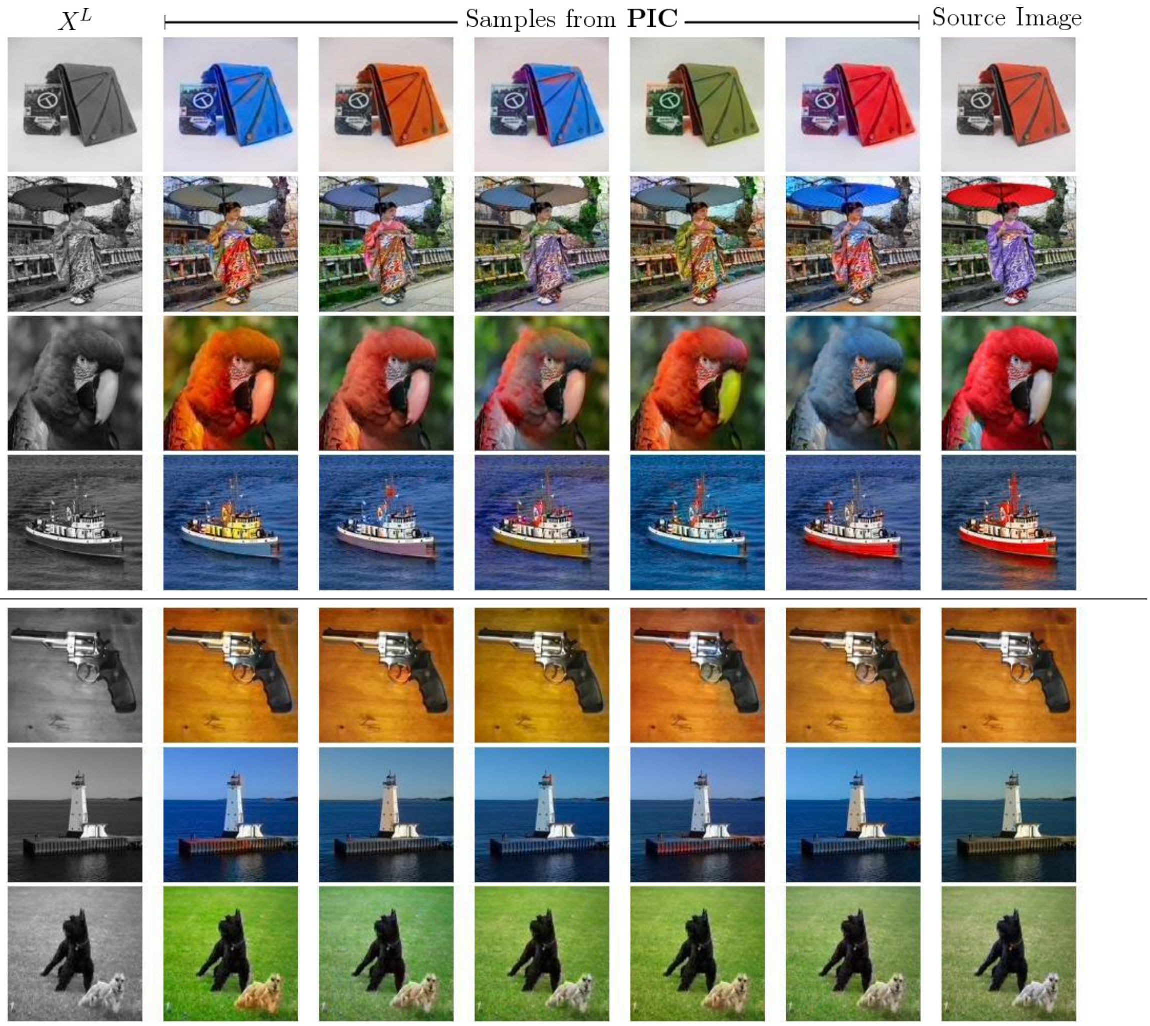


Figure 3.14: Colorized samples from our model illustrate its ability to produce diverse (top) or consistent (bottom) samples depending whether the image semantics are ambiguous or not.

dicates the average amount of uncertainty in the image colors under the trained model: It is a principled measure that can be used to perform model selection and compare various probabilistic colorization techniques.

ILSVRC-2012 experiments. We now report experimental evaluation results of PIC on the more challenging ILSVRC-2012 dataset [Russakovsky *et al.*, 2015]. The dataset contains 1.2 million high-resolution training images spread over 1000 different semantic categories, and a held-out set of 50000 validation images. Images are also larger than in CIFAR-10: In our experiments we scale input images to 128x128 pixels, which is enough to capture essential image details and remains a challenging scenario. However in principle our method is applicable – and scales to – higher resolutions.

As ILSVRC images are of higher resolution and contain more details than CIFAR-

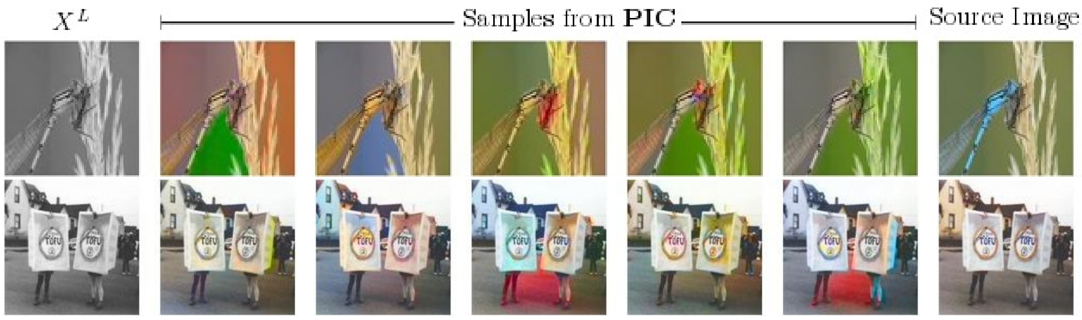


Figure 3.15: Failure cases: PIC may fail to reflect very long-range pixel interactions (top) and, *e.g.*, assign different colors to disconnected parts of an occluded object, or it may fail to understand semantics of complex scenes with unusual objects (bottom).

10 images, we use a slightly deeper and wider architecture for the embedding function e_ϕ, as specified in Table 3.5 (right), and a chroma-subsampling factor of 4, as in [Zhang *et al.*, 2016]. For the autoregressive component f_θ, we use the same characteristics as in the CIFAR-10 case (4 residual blocks and 160 channels for every convolution). We train the model for 20 epochs using batches of 64 images, with learning rate decaying after every iteration with a rate of 0.99999.

In Figure 3.14 we present random colorized samples from the ILSVRC validation set. These demonstrate that our model is capable of producing spatially coherent and semantically plausible colors. Moreover, in the case where the color is ambiguous, the produced samples often demonstrate wide color diversity. Nevertheless, if the color is mostly determined by the semantics of the object (grass or sky), then PIC produces consistent colors. Additional samples are reported in Figure 3.19. To provide further insights into the model's behavior, we also highlight two failure cases in Figure 3.15: First, PIC may not always fully capture complex long-range pixel interactions, *e.g.*, if an object is split due to occlusion, the two parts may have different colors. Second, for some complex images with unusual objects, PIC may fail to understand semantics of the image and produce not visually plausible colors.

Our model achieves a negative log-likelihood of **2.51** bits-per-dimension (the lower the better). As a reference, the ImageNet generative model from [van den Oord *et al.*, 2016c], which is based on a similar but deeper architecture, reports a negative log-likelihood of **3.86** for the ILSVRC validation images modeled at the same resolution. As our model has access to additional information (grayscale input), it is not surprising that we achieve a lower likelihood; Nevertheless, this result confirms that PIC learns

Figure 3.16: Qualitative results from several automatic colorization methods and our method on the same grayscale input (left) compared to the original ground-truth (right).

non-trivial colorization model and strengthens our qualitative evaluation.

Qualitative comparison to baselines. In Figure 3.16 we present a few colorization results on the ImageNet validation set for our model (random sample) as well as three recent colorization baselines: [Zhang *et al.*, 2016] propose a deep feed-forward VGG architecture trained on ImageNet for automatic colorization. Additionally, they treat colorization as a classification rather than regression task, combined with class-rebalancing in the training loss to favor rare colors and more vibrant samples. [Larsson *et al.*, 2016] introduce a very similar model except for a few architectural differences (*e.g.*, use of hypercolumns) and heuristics. Finally, [Iizuka *et al.*, 2016] propose a non-probabilistic model with a regression objective. Their architecture is also more complex as they use two distinct flows for local and global features. We also note that their model was trained on the MIT Places dataset, while ours and the two previous baselines use ImageNet. We use the publicly available implementation for each baseline.

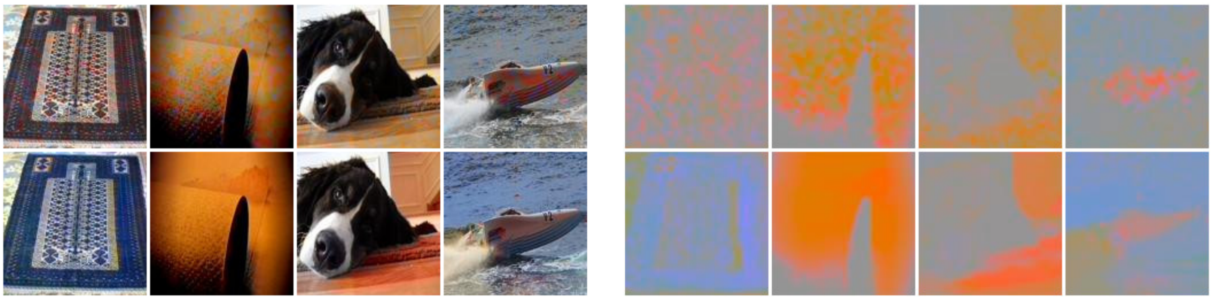
In general, we observe that our model is highly competitive with other approaches and tends to produce more saturated colors on average. We also report additional random samples from our model in Figure 3.19.

3.2.5 Ablation experiments

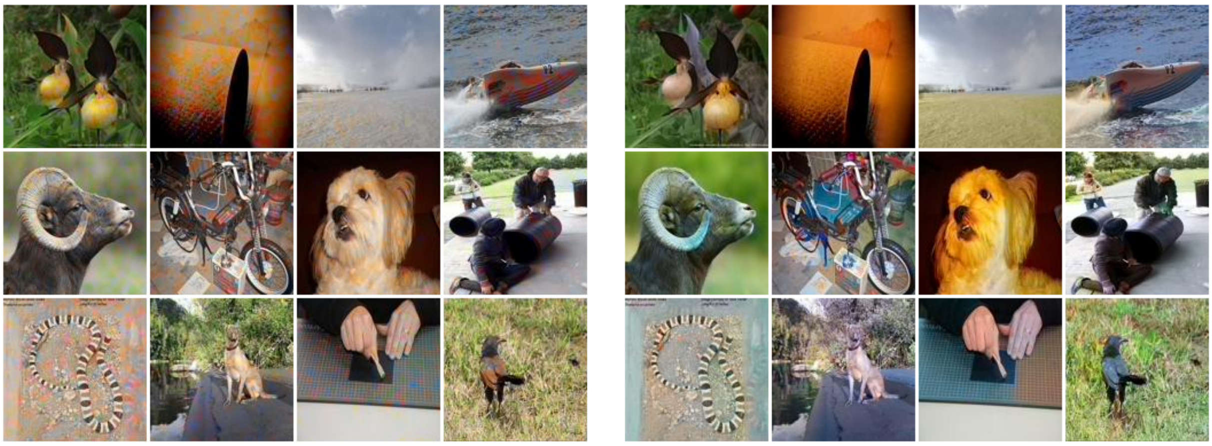
Importance of the autoregressive component. One of the main novelties of our model is the autoregressive component, f_θ, which drastically increases the colorization performance by modeling the joint distribution over all pixels. In this section we perform an ablation study in order to investigate the importance of the autoregressive component alone: In fact, without f_θ, our model essentially reduces to the feed-forward embedding network e_ϕ, similar to recent colorization techniques [Zhang *et al.*, 2016; Larsson *et al.*, 2016]. More specifically, we use PIC pre-trained on the ILSVRC dataset, discard the autoregressive component f_θ, and fine-tune the remaining embedding network, e_ϕ, for the task of image colorization. At test time, we draw maximum a posteriori (MAP) samples from this model: Stochastic sampling from the output of e_ϕ would produce very noisy colorizations as the pixelwise predicted distributions are independent. Alternatively, one could predict the mean color of the predicted distribution for each pixel, but the averaging effect would result in desaturated colors.

Comparing the output samples of PIC and e_ϕ in Figure 3.17, it appears that the benefit brought by the autoregressive component is two-fold: First, it explicitly models relationships between neighboring pixels, which leads to visually smoother samples. Second, the samples generated from PIC tend to display more saturated colors. We also verify this quantitatively by computing the average perceptual saturation [Lübbe, 2010] for both methods: Based on 1000 random image samples, the PIC model and e_ϕ have an average saturation of **36.4%** and **32.7%**, respectively.

Gating and model selection. Recently, it has been demonstrated that gated convolutional layers are useful for the task of natural image modeling [van den Oord *et al.*, 2016b]. Thus, we explore whether using gated activation units is also beneficial for our architecture. We first perform a qualitative analysis, comparing samples drawn from PIC, with and without gated non-linearities, as illustrated in Figure 3.18. Although the samples obtained with the gating mechanism appear to have higher visual quality, *i.e.* have slightly more saturated colors and better global consistency, it is hard to



(a) Closeup comparison between the embedding network e_ϕ (top row) and the autoregressive PIC model (bottom row). Final colored images are reported on the (left) and predicted chrominance (displayed for fixed luminance $\mathbf{L} = 50$) on the right



(b) Additional samples from e_ϕ (left) and from PIC (right)

Figure 3.17: Comparison on the ImageNet validation set between MAP samples from the embedding network e_ϕ and random samples from the proposed PIC model.

make definitive conclusions. Thus, we also perform quantitative analysis by comparing the data likelihood reported by each model: PIC with gating achieves the negative log-likelihood of **2.72**, while its counterpart without gating achieves **2.78**, which is consistent with our preliminary qualitative evaluation.

Therefore, we argue that negative log-likelihood on the hold-out image set can serve as a principled measure for model selection: This is an additional advantage of autoregressive models for image colorization, as, contrary to other generative models such as GANs or VAEs, they provide a way to compute the exact data likelihood. Importantly, this metric measures how well *the joint distribution* of image colors is explained by the model. Hence, unlike previous metrics that were used to evaluate image colorization performance, it also accounts for the intrinsic probabilistic uncertainty of the task.

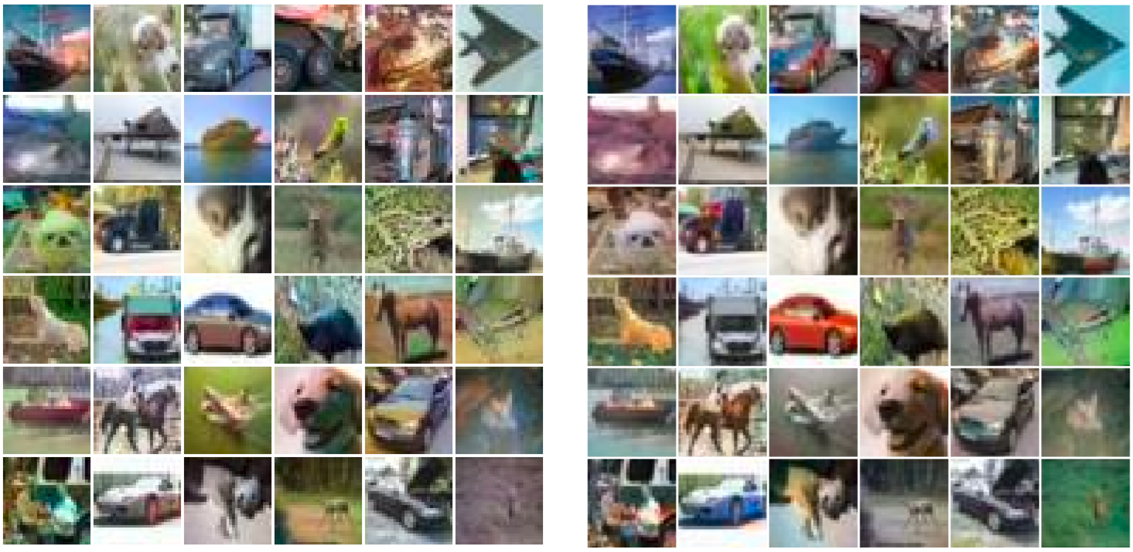


Figure 3.18: Comparison of CIFAR-10 colorization samples obtained without gating *(left)* and with gating *(right)*.

3.2.6 Conclusions and discussions

While deep feed-forward networks achieve promising results on the task of colorizing natural grayscale images, the generated samples often suffer of a lack of *diversity* and *color vibrancy*. We tackle both aspects by modeling the full joint distribution of pixel color values using an autoregressive network conditioned on a learned embedding of the grayscale image. The fully probabilistic nature of this framework provides us with a proper and straightforward sampling mechanism, hence the ability to generate diverse plausible samples from a given grayscale input. Furthermore, the data likelihood can be efficiently computed from the model and used as a quantitative evaluation metric for model comparison. We report quantitative and qualitative evaluations of our model, and show that colorizations sampled from our architecture often display vivid colors, indicating that the model effectively captures the underlying color distribution of natural images, without requiring any *ad hoc* heuristics during training.

Since the time this work was published, extensive progresses has been made in the field of *flow-based generative models* [Dinh *et al.*, 2014; Dinh *et al.*, 2017; Kingma and Dhariwal, 2018]: Based on the VAE framework, these models rely on simple flow transformations of the latent space and provide a tractable computation of the exact likelihood. However they typically require a lot of memory and they are still outperformed by autoregressive models, in terms of data likelihood [Ho *et al.*, 2019].

Therefore, autoregressive models still seem to be the most fitting approach to properly model the joint pixels colors distribution in a fully probabilistic manner for the task of image colorization. Finally, the main drawback of our proposed model is the sampling time at inference, as the iterative nature of autoregressive models requires a forward pass through the network for each pixel. A recently proposed autoregressive architecture [Reed *et al.*, 2017] alleviates this problem by modeling certain pixels as conditionally independent, allowing for parallel computations. Since this improvement essentially relies on a minor changes in the autoregressive network, it could easily be adapted into our architecture for image colorization

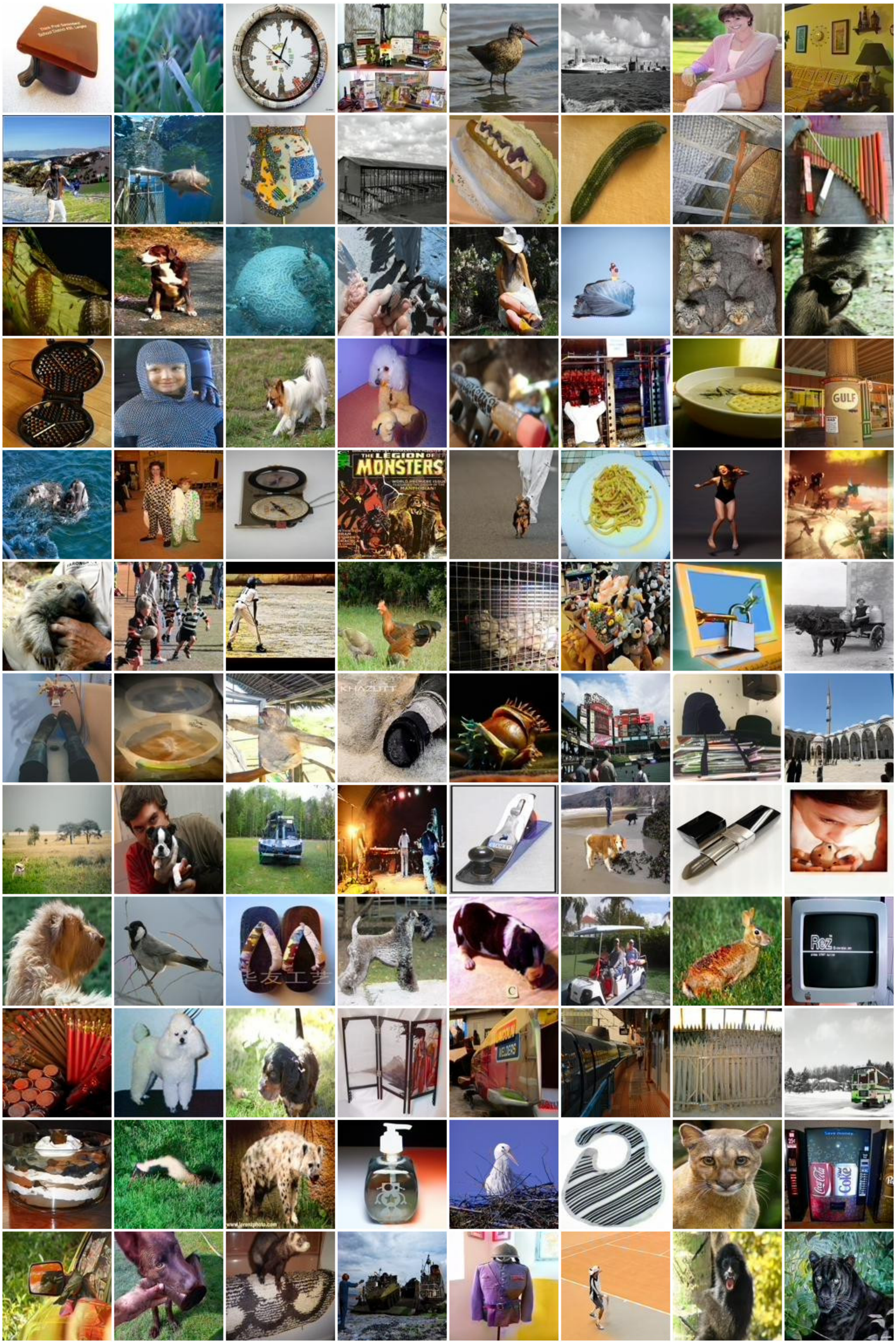


Figure 3.19: Random colorized images from the ILSVRC-2012 validation set.

4 Isolating Domain-specific Information for Transfer Learning

In the first chapter, we investigated neural networks' ability to *generalize* outside of their training conditions in the context of compositional learning, and more precisely, abstract visual reasoning (AVR). In particular, we observed that the underlying key problem stems from the *domain shift* between the training and testing distribution, which we quantified and partially addressed by leveraging the compositional structure of AVR. However, this analysis does not hold in the generic transfer learning scenario, where the shift between the training and test domain does not have a specific compositional structure. In this chapter, we focus on this more general setting in the context of deep learning: In particular, we investigate how the layered structure of neural networks comes into play when encoding shared knowledge across visually different domains.

Generally speaking, *transfer learning* encompasses all methods that exploit knowledge learned from a *source* task $S : \mathcal{X}_S \rightarrow \mathcal{Y}_S$ in order to solve a different *target* task $T : \mathcal{X}_T \rightarrow \mathcal{Y}_T$. For practical purposes, it is assumed that the tasks relate to – and can benefit from – one another, otherwise negative transfer may occur [Pan and Yang, 2010; Wang *et al.*, 2018b]. In the current computer vision literature, transfer learning is most often encountered under one of the two following forms. In *unsupervised domain adaptation*, given labeled samples from a source domain and unlabeled samples from a target domain, the goal is to learn a model that can jointly solve a common task for both domains with high accuracy. Secondly, *fine-tuning* consists in adapting the weights of a given network pre-trained on the source task such that it performs well for the target task, for which labeled samples are available. In summary, these two settings differ *(i)* in terms of available supervision – there are no source samples for fine-tuning, and only unlabeled target samples are given for domain adaptation – and *(ii)* in terms of evaluation metric – fine-tuning usually aims to maximize performance on the target task, while domain adaptation models are evaluated on both source and target samples. Nevertheless, both problems share the same domain supervision asymmetry: In domain adaptation, the target domain is usually unlabeled, while in practical fine-tuning applications, often only few target samples are usually available (in comparison to the number of source samples the network was pre-trained on).

Both domain adaptation and fine-tuning methods ultimately aim to *align* the two domains by learning a *joint intermediate representation* over the source and target domains, from which the task of interest can be subsequently solved. This often implies using partially shared architectures between the two domains. In this regard, a key problem is that neural architectures behave as *black boxes*, and there's only little understanding about the information that each layer captures in practice [Olah *et al.*, 2017] and even less in theory. Hence, it is challenging to determine what features this shared representation should encode and to what degree layers in the neural architecture should be shared. In particular, these issues are also highly dependent on the domain shift between the source and target domain we aim to align. Unfortunately, quantifying this shift is a hard problem: Pixel-level metrics are often uninformative when comparing images, and feature-level metrics requires a trained representation of both domains, leading to a vicious cycle. Some discrepancy measures have been introduced in the domain adaptation literature [Ben-David *et al.*, 2010], however these are hard to approximate for real-life data.

In this chapter, we will investigate experimentally how the layered structure of neural network relates to the source-to-target domain shift in transfer learning problems. First, in Section 4.1, we will analyze the correlation between the visual input domain shift and the optimal layer fine-tuning strategy for generic architectures. In fact, it is commonly accepted that early layers capture *generic* information such as edges, common to most image domains, while higher layers capture *specific* information, which is often discarded when fine-tuning on a new domain. However, there's little understanding of what intermediate layers capture, and in particular to what extent this information depend on the input image domain or on the output target task. To answer these questions, we conduct an experimental study with a newly proposed flexible transfer learning scheme, called *flex-tuning*.

Second, in Section 4.2 we will tackle the problem of unsupervised pixel-to-pixel level transformation across visual domains. While a more challenging task, learning such a mapping yields much more interpretable outputs than feature-level adaptation, as we can directly observe the transformed source to target samples. Furthermore, this transformation does not depend on a particular task, but only on the two input domains, hence is more generic and can be applied to further tasks. In particular, we use a shared autoencoder architecture that explicitly constrains the model to distinguish between *private* information, specific to each domain, and *shared* information, which is encoder in a joint representation of the two domains. In our experiments, we compare the proposed model to a baseline that use only private modules (no shared representation space), and one that use a fully shared encoder. We show that our mixed approach yields better results than both variants.

4.1 Flex-tuning : A Flexible Layer Selection Scheme for Minimum-Effort Transfer Learning

Deep convolutional networks have substantially advanced the state of the art in many areas of computer vision, such as object recognition, detection or image segmentation. These networks are often interpreted as a feature extraction stage (typically convolutional layers), followed by a small classifier (fully connected layers), and have the ability to learn features from data directly instead of having them hard-coded, as was the case for previous vector embedding techniques [Lowe, 1999; Bay *et al.*, 2006; Csurka *et al.*, 2004]. However, training these networks comes with a cost, as it requires a large training dataset in contrast to methods relying on fixed ad-hoc feature extraction. Consequently, it is not surprising that the first successes of deep networks in image classification occurred as large annotated datasets were made available, *e.g.* MNIST [LeCun *et al.*, 1998] for digit recognition (60,000 training samples) or ImageNet [Russakovsky *et al.*, 2015] for object classification (1.2 million).

When only little available training data is available, however, training a deep feature extraction pipeline from scratch is not possible, as it often leads to severe overfitting. Instead, two main *fine-tuning* strategies have emerged, exploiting the fact that deep convolutional networks pre-trained on large datasets are freely available these days [Model Zoo; TensorFlow Hub; TensorNets]: *Either*, one isolates and “freezes” the feature extraction stage of the pre-trained model and then uses the available new data to train only the smaller, less prone to overfitting, classifier stage, *or alternatively*, one fully fine-tunes the model, *i.e.* initializes the parameters from the pre-trained network, and then trains all layers, typically only for a few steps, to avoid overfitting. Which of the two approaches is more promising depends on the situation at hand. The folk wisdom in the community is that fully fine-tuning leads to better results, but requires more training data than for only learning a new classification layer. Furthermore, choosing the best solution depends on the data characteristics. For example, it has been observed that features learned on large and varied natural images datasets, *e.g.* ImageNet, transfer well to related domains such as aerial or even biomedical images [Crowley and Zisserman, 2014; Kornblith *et al.*, 2019], while fine-tuning all layers is preferable for domains with different low-level statistics, *e.g.* sketches [Ballester and Araujo, 2016].

In this work, we argue for a more systematic approach to exploiting pre-trained networks, in situations where the new input domain can *vary greatly in terms of visual appearance*, but its output space shares *similar semantics* with the one the model was pre-trained on. We introduce the idea of *flex-tuning*, a general-purpose transfer learning scheme that leverages the information of an available pre-trained model by fine-tuning a *targeted part of the model*, not necessarily the last layer or all layers, *but any individual layer or block of consecutive layers, selected in a data-dependent way*. In fact, the idea of focusing training resources on specific intermediate layers draws inspiration from an important transfer learning paradigm: It has been consistently observed across various architectures in the literature that early convolutional layers capture elementary local properties of images such as edges or local textures, while middle layers rather represent configurations of several such elements, and the last feature layers extract information about high-level concepts, such as object parts and their configurations [Cadena *et al.*, 2018; Olah *et al.*, 2017; Zeiler and Fergus, 2014]. Thus, in order to adapt, for instance, a network trained on clean natural images to work with noisy ones, we hypothesize it is easier to fine-tune early layers, while for adapting the same network to artistic paintings, focusing on a later layer would be more promising. Based on this intuition, our contribution is three-fold:

- *First*, we formally define *flex-tuning*, which is a model selection strategy for, given a pre-trained network and a new training dataset, deciding in a data-dependent and automatic way which of the available layers to fine-tune.
- *Second*, in order to make flex-tuning more appealing for practical use, we further introduce two variants based on a more efficient selection criterion, called *fast flex-tuning* and *even faster flex-tuning*, that avoid the need to train multiple fine-tuned models for the selection process.
- *Finally*, we design an extensive experimental setup that covers varied visual domain shifts, data scarcity scenarios and architectures: We show that flex-tuning almost always improves classification accuracy over standard fine-tuning techniques, particularly in settings where fine-tuning all layers is prone to overfitting, such as settings with small sample size and large networks. Furthermore, the (even) faster flex-tuning variants are generally on par with flex-tuning while providing a much lighter selection procedure.

4.1.1 Related work

Transferrability of pre-trained convolutional networks across visual tasks has been often observed and extensively studied in the computer vision literature [Azizpour *et al.*, 2016; Chu *et al.*, 2016; Donahue *et al.*, 2014; Yosinski *et al.*, 2014; Zhang *et al.*, 2018a]. In fact, many state-of-the-art computer vision models are not trained from random initialization, but rely crucially on the re-use of weights from networks pre-trained on large classification tasks, such as ImageNet [Russakovsky *et al.*, 2015]. Popular examples include the YOLO object detector [Redmon *et al.*, 2016] or fully-convolutional networks for segmentation [Long *et al.*, 2015]. In the weakly supervised learning literature, pre-trained features are also used as compact and meaningful image representations, *e.g.* for image retrieval [Babenko and Lempitsky, 2015], style transfer [Gatys *et al.*, 2016; Johnson *et al.*, 2016], colorization [Larsson *et al.*, 2016], or unsupervised part detection [Simon and Rodner, 2015]. All these approaches aim at transferring knowledge between two tasks that have different output structures but similar input domain appearances and distributions. Closest to our work is [Yosinski *et al.*, 2014], which studies the outcome of fine-tuning convolutional architectures, starting from different levels of a pre-trained network, in the standard transfer learning setting. In contrast, we analyze the effect of tuning a single unit of a pre-trained network, in particular for situations where source and target domains are visually dissimilar but semantically close.

In fact, our interest lies exactly in these orthogonal scenarios, where the source and target domain share a similar output task, typically multi-class classification, but with – potentially significantly – different input distributions. This setting resembles, yet differs from, the problem of *domain adaptation* [Saenko *et al.*, 2010; Gopalan *et al.*, 2011; Ganin *et al.*, 2016], whose goal is to construct a classifier for a, usually unlabeled, target task, by exploiting one or more source tasks,for which annotated samples are available. In contrast, in the fine-tuning scenario, one only has access to a target training dataset and the pre-trained network, but not to the samples it was trained on: This lack of supervision rules out domain adaptation techniques such adversarial training [Kim *et al.*, 2017; Zhu *et al.*, 2017], paired samples [Isola *et al.*, 2017], or more generally, exploiting any concrete knowledge from the source distribution to improve predictions on the target domain.

In fact, with the growth of datasets and necessary compute resources, the ability to tune networks without access to the original training data is becoming more and more important: First, when dealing with very large source datasets, training jointly on the source and target domains, as many domain adaptation methods require, is computationally impractical. Second, source training data is sometimes non-public, especially in commercial settings. Third, specific applications require data privacy, preventing public data release, for instance for protecting individuals identities in face recognition models. As such, learning under privacy constraints has become a popular topic in recent years [Papernot *et al.*, 2017].

Finally, recent work has also tackled the problem of domain adaptation by transferring from the source to target domain directly at the pixel level, either via generative models [Bousmalis *et al.*, 2017] or by identifying simpler causal transformations [Parascandolo *et al.*, 2018]. Weight tuning methods are nonetheless simpler to use, as they directly act on feature representations, rather than learning a transformation that holds independently of the pre-trained network.

4.1.2 A flexible transfer learning scheme: Flex-tuning

Our first contribution in this work is to highlight that simple and lightweight, but surprisingly effective, model adaptation is possible by fine-tuning the weights of only a single unit in a pre-trained network, provided that the right unit is chosen. Which is the right unit depends crucially, and in a non-trivial way, on the relation between source and target domains as well as on the amount of available data. We propose to identify the best unit automatically in a data-dependent manner using a procedure we call *flex-tuning*.

Background. First, we formally introduce the transfer learning scenario we are interested in: We are given a pre-trained convolutional network, f_θ, mapping input space $\mathcal{X}$ to an output space $\mathcal{Y}$, whose weights θ were pre-trained on a *source* task, using a dataset $D_{\text{src}} \overset{iid}{\sim} \mathcal{P}(\mathcal{X}, \mathcal{Y})$ that is however not available anymore. Our goal is to learn a new set of parameters θ' for the network in order to solve a different *target task*, for which a new annotated training dataset, $D \overset{iid}{\sim} \mathcal{P}(\mathcal{X}, \mathcal{Y})$, is available. In our experiments, we work with RGB images as inputs, and discrete labels as outputs. However, the underlying principles apply equally to other input domains and tasks.

We focus on thoroughly analyzing and characterizing the influence of *individual units* on transferring knowledge across visually different domains under various data scarcity settings. To this end, we consider practical settings where the target domain is *semantically* close but *visually* different from the source domain. By semantically close, we mean that the output space of the target task is a subset of the source task. This setting encompasses a variety of real-world scenarios, where the source and target input domains do not overlap well: For instance, we can consider a source network trained on natural images, with the target task of classifying monochrome sketches; or a network trained on scenes under daylight, that should also operate at night, *etc*.

Moreover, extending the framework to different output structures, *e.g*. from a classification task to a detection task, would be possible by fine-tuning both the unit selected by flex-tuning and the last fully-connected layer. However, for the sake of analysis this would be a much less precise scenario as *(i)* we could not single out the effect of individual units and *(ii)* we could not decouple the influence from the visual domain shift and the one of the output domain shift.

Proposed method. Given the pre-trained neural network f_θ, we first decompose the architecture into smaller *units*, which we denote as $f_\theta = f_\theta^L \circ \cdots \circ f_\theta^1$. For simplicity, we will ignore the θ index when it is clear from context. Each unit can be defined as a single convolutional or fully-connected layer, or, for more complex architectures, a block of consecutive layers. The method applies to arbitrary decompositions.

Algorithm 3: Flex-Tuning (`flex`)

Input : $\boldsymbol{D}$ – Training dataset
$\boldsymbol{D}_{\text{val}}$ – Validation dataset
$\boldsymbol{f}$ – Pre-trained network with L units, $f = f^L \circ \cdots \circ f^1$

for $\ell = 1, \dots, L$ **do**
  $f_{\texttt{ft-}\ell} \leftarrow$ fine-tune unit ℓ of f on D until accuracy on D_{val} stops improving;
  $a_{\texttt{ft-}\ell} \leftarrow$ accuracy of $f_{\texttt{ft-}\ell}$ on D_{val}

$f_{\texttt{ft-all}} \leftarrow$ fine-tune all units of f on D until accuracy on D_{val} stops improving;
$a_{\texttt{ft-all}} \leftarrow$ accuracy of $f_{\texttt{ft-all}}$ on D_{val};
$f_{\texttt{flex}} \leftarrow f_{\texttt{best}}$ for $\texttt{best} \leftarrow \operatorname*{arg\,max}_{X \in \{\texttt{1,...,L,all}\}} a_{\texttt{ft-X}}$;
return $f_{\texttt{flex}}$

Given such a decomposition, our goal is to analyze the influence of tuning specific units, for transferring knowledge across domains with different visual appearances. Algorithm 3 describes the steps of flex-tuning in pseudo-code: For each unit of the network, we construct a fine-tuned network $f_{\texttt{ft-}\ell}$ by training the network on the available target data, allowing only the weights of the ℓ-th unit to change, while keeping all the others frozen. We also create a network $f_{\texttt{ft-all}}$, for which all layers are fine-tuned. We train each network with an early stopping criterion, monitoring its performance on a validation set, D_{val}. This helps prevent overfitting in a way that is tailored to each tuning setting. In fact, different units might have very different numbers of weight parameters, and therefore will often need different numbers of epochs to converge. Finally, we choose the best model out of these $L+1$ networks by comparing their accuracy on the validation set and output it as the *flex-tuned* model, $f_{\texttt{flex}}$.

We illustrate flex-tuning's process in Figure 4.1: We apply the proposed method on a small network (5 convolutional and 2 fully connected layers) pre-trained on CIFAR [Krizhevsky and Hinton, 2009] and to be adapted to a subset of the "Quick, Draw!" dataset [Cheema *et al.*, 2012] and a blurred variant of CIFAR, for different sizes of the target training dataset. These preliminary results show that *(i)* it is often beneficial to fine-tune an intermediate layer rather than the last one and that *(ii)* well-performing units strongly depend on the dataset and in a non-trivial way, but can be efficiently pin-pointed with a simple selection criterion such as flex-tuning.

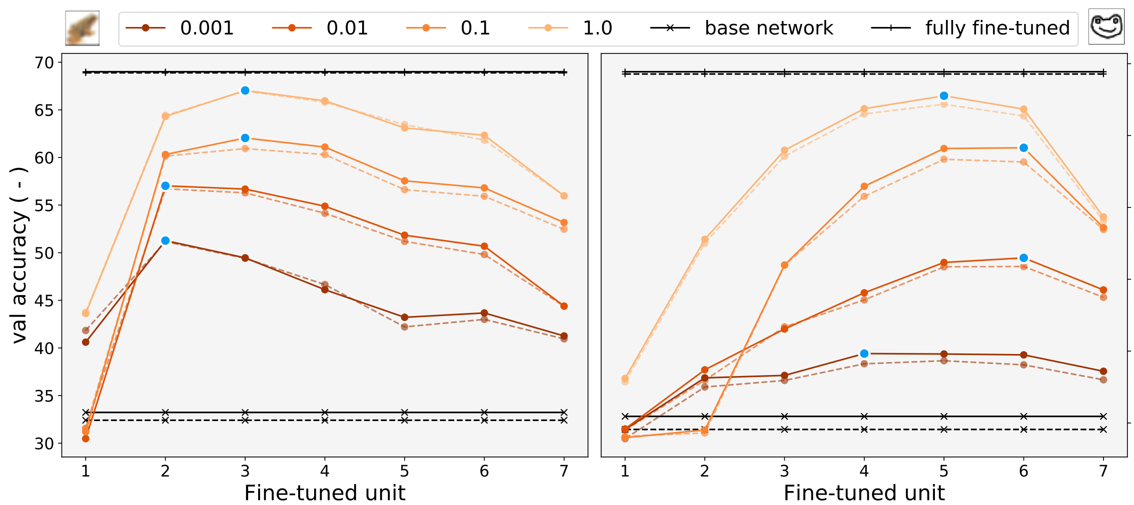


Figure 4.1: Validation (-) and test (- -) accuracies for fine-tuning a single unit of a pre-trained CIFAR network to the Blurry CIFAR (left) and Quick, Draw! (right) datasets. Each line color represents a different subsampling ratio of the target training dataset, while blue markers indicate the unit picked based on validation accuracy.

Practicality of the method. Technically, Algorithm 3 performs an exhaustive search over the potential fine-tuned models. Therefore, the existing theoretical results for model selection [Shalev-Shwartz and Ben-David, 2014] apply, and we obtain that, in the limit, flex-tuning will indeed choose the best of the possible models. Moreover, the difference between flex-tuning's accuracy estimated from the validation data and the expected accuracy on future data decreases with a rate of $O(\sqrt{\frac{\log L}{|D_{\text{val}}|}})$.

For deep networks however, flex-tuning can be computationally costly in practice: It requires training as many networks as there are units, plus another one in which all units are fine-tuned. Let us denote the average number of training epochs by E_{one} when fine-tuning a single unit, and by E_{all} when fine-tuning all. Also, let us denote the corresponding average computational cost of one such epoch as c_{one} and c_{all}, respectively. Then the total runtime complexity of flex-tuning is $O(LE_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}})$. Even when taking into account that typically $E_{\text{all}} > E_{\text{one}}$ and $c_{\text{all}} > c_{\text{one}}$, for reasonably large networks the complexity is often dominated by the computational cost of fine-tuning the network once for each unit. Since ultimately only one of the models is chosen, these computations end up wasted. To address this issue, we introduce two improved selection criteria in the following section to efficiently approximate flex-tuning.

4.1.3 Efficient selection criteria

To overcome the aforementioned computational inefficiency of flex-tuning, we propose two new criteria, *fast flex-tuning* and *even faster flex-tuning*, for selecting the unit to be fine-tuned while minimizing training effort.

Fast flex-tuning relies on the idea that any unit's influence can be approximated by *a few feed-forward passes* rather than a full training process. While it does not come with formal guarantees, we found it to work nearly as well as the exhaustive search in practice, while at the same time requiring only 2 networks to be trained instead of $L + 1$. Algorithm 4 describes fast flex-tuning in pseudo-code: We first start one new model, $f_{\texttt{ft-all}}$, by fine-tuning all units of the pre-trained network on the training data available for the target domain. From this, we construct L new networks by *network surgery*. For any $\ell = 1, \dots, L$, we create a proxy network, $f_{\texttt{prox-}\ell}$, by copying all units from f, except the ℓ-th one, which is copied from the fine-tuned network, $f_{\texttt{ft-all}}$.

Algorithm 4: Fast Flex-Tuning (`fast-flex`)

Input : $\boldsymbol{D}$ – Training dataset
$\boldsymbol{D}_{\text{val}}$ – Validation dataset
$\boldsymbol{f}$ – Pre-trained network with L units, $f = f^L \circ \cdots \circ f^1$

$f_{\texttt{ft-all}} \leftarrow$ fine-tune all units of f on $\mathcal{D}$ until accuracy on D_{val} stops improving;
$a_{\texttt{ft-all}} \leftarrow$ accuracy of $f_{\texttt{ft-all}}$ on D_{val};
for $\ell = 1, \ldots, L$ **do**
    $f_{\texttt{prox-}\ell} \leftarrow f^L \circ \cdots \circ f^{l+1} \circ f^{\ell}_{\texttt{ft-all}} \circ f^{\ell-1} \circ \cdots \circ f^1$;
    $a_\ell \leftarrow$ accuracy of $f_{\texttt{prox-}\ell}$ on D_{val};
$\texttt{best} \leftarrow \arg\max_\ell \; a_\ell, \; \ell \in \{\texttt{1, ..., L}\}$;
$f_{\texttt{best}} \leftarrow$ fine-tune unit `best` of f on $\mathcal{D}_{\text{train}}$ until accuracy on $\mathcal{D}_{\text{val}}$ stops improving;
$a_{\texttt{ft-best}} \leftarrow$ accuracy of $f_{\texttt{best}}$ on $\mathcal{D}_{\text{val}}$;
$f_{\texttt{flex}} \leftarrow$ **if** $a_{\texttt{ft-best}} \geq a_{\texttt{ft-all}}$ **then** $f_{\texttt{best}}$ **else** $f_{\texttt{ft-all}}$;
return $f_{\texttt{flex}}$

Clearly, the resulting hybrid networks are not functional models, as the ℓ-th unit and the other units were not trained together. Nevertheless, this construction allows us to derive a measure of which of the network units is the most promising candidate for fine-tuning, namely the one that leads to the *biggest improvement in accuracy (if any) when applied to the target domain*. Numerically, we compute the accuracy of each model $f_{\texttt{prox-}\ell}$ on the validation dataset and identify the value of ℓ that yields the model with highest accuracy. We then create a viable model by fine-tuning the selected unit on the target dataset $\mathcal{D}$. Finally, we output either this model, or the one in which all layers were fine-tuned (which is available as we created it at the beginning of the procedure), depending on which achieved the higher validation accuracy.

In comparison to flex-tuning, fast flex-tuning only has to fine-tune two networks instead of $L+1$. Its runtime complexity is hence $O(E_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}})$, thereby providing substantial computational savings for large networks.

Even faster flex-tuning. In some situations, training from scratch or fine-tuning the complete network is simply computationally too costly: Neither flex-tuning nor fast flex-tuning are applicable, as both require training a network by fine-tuning all units as the first step of their selection process. To overcome this, we propose an even faster variant, as described in Algorithm 5.

Even faster flex-tuning resembles fast flex-tuning in that it selects a unit to be fine-

Algorithm 5: Even Faster Flex-Tuning (`faster-flex`)

Input : D – Training dataset
D_{val} – Validation dataset
f – Pre-trained network with L units, $f = f^L \circ \cdots \circ f^1$

$f_{\texttt{ft-all}} \leftarrow$ fine-tune all units of f on $\mathcal{D}$ for a single epoch;
for $\ell = 1, \ldots, L$ **do**
  $f_{\texttt{prox-}\ell} \leftarrow f^L \circ \cdots \circ f^{l+1} \circ f^{\ell}_{\texttt{ft-all}} \circ f^{\ell-1} \circ \cdots \circ f^1$;
  $a_\ell \leftarrow$ accuracy of $f_{\texttt{prox-}\ell}$ on D_{val}
`best` $\leftarrow \arg\max_\ell \ a_\ell,\ \ell \in \{\texttt{1, ..., L}\}$;
$f_{\texttt{flex}} \leftarrow$ fine-tune unit `best` of f on $\mathcal{D}_{\text{train}}$ until accuracy on $\mathcal{D}_{\text{val}}$ stops improving;
return $f_{\texttt{flex}}$

tuned based on the accuracies of different proxy models that are obtained by network surgery, each time preserving $L-1$ units from the pre-trained source network and replacing the remaining one with its fine-tuned counterpart. The difference lies in that the fine-tuned units are obtained from a network in which all units have been fine-tuned for just a *single epoch*. This results in a total computational runtime of $O(E_{\text{one}}c_{\text{one}} + c_{\text{all}})$. We consider this close to optimal for an adaptive technique, as at least the cost $E_{\text{one}}c_{\text{one}}$ clearly cannot be avoided, if the goal is to produce a network in which at least one unit has been fine-tuned. The drawback of the acceleration is that the *even faster flex-tuning* algorithm does not have access to a reliable estimate of what performance a network with all units fully fine-tuned would have achieved. This is however not relevant here as, by assumption, the computational budget does not suffice for training such a model.

Table 4.1 summarizes the runtime complexity of all proposed models, as well as the two main transfer learning baselines, which we also compare to in our experiments: `ft-fc`, which fine-tunes always the last unit (*i.e.* the fully-connected layer(s)), and `ft-all`, which fine-tunes always all layers.

4.1.4 A visually diverse benchmark for transfer learning

In this section, we introduce our experimental setup, covering a large number of domain shifts and data scarcity settings. We build several domain shift scenarios, ranging from simple parametric transformations to severe visual appearance shifts. In order to explore the impact of data scarcity, we additionally consider several subsampled ver-

method	computational cost
`flex`	$LE_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}}$
`fast-flex`	$E_{\text{one}}c_{\text{one}} + E_{\text{all}}c_{\text{all}}$
`faster-flex`	$E_{\text{one}}c_{\text{one}} + c_{\text{all}}$
`ft-fc`	$E_{\text{one}}c_{\text{one}}$
`ft-all`	$E_{\text{all}}c_{\text{all}}$

Table 4.1: Runtime complexities. L is the number of units in the network, E_{one} and c_{one} are the average number of epochs until early stopping for fine-tuning one unit, and the estimated cost of one such epoch. E_{all} and c_{all} are the analogous quantities when fine-tuning all network units. In general, $E_{\text{all}} > E_{\text{one}}$ and $c_{\text{all}} > c_{\text{one}}$.

sions of each target dataset, ranging from a few images per class to hundreds of them. The different settings are thus mainly characterized by: *(i)* the depth of the base source network, *(ii)* the size of the target dataset we tune on, and *(iii)* the type of input domain shift: simple parametric transformations, *e.g*. manipulating color channels, complex (non-trivially invertible) parametric transformations, and general free-range transformations. We summarize our complete setup in Table 4.2.

Small and medium-sized experiments. We first consider a small 4 layers network (which we decompose in 4 one-layer units: 2 convolutional layers followed by 2 fully-connected ones) pre-trained on a subset of the MNIST training dataset [LeCun *et al.*, 1998]. We use the remaining samples (except 5000 of them that we keep for validation) to build synthetic domain shifts such as *affine transformations* (randomized or fixed for all images), or *random occlusions*. Second, we build a 7 layers network (7 one-layer units: 5 convolutional and 2 fully connected ones) that we pre-train on half of the CIFAR training set [Krizhevsky and Hinton, 2009]. As target domains, we consider several synthetic transformations of the remaining samples, as well as a subset of the QuickDraw dataset [Cheema *et al.*, 2012]: We restrict ourselves to the object classes they have in common, *i.e*. all CIFAR classes except for “deer”. We also consider the converse setting, *i.e*. pre-training on QuickDraw and using as target domains CIFAR and synthetically generated blurry and noisy QuickDraw samples.

Large-scale setting. Finally, we consider two large-scale settings using the Inception2 architecture [Ioffe and Szegedy, 2015; TensorNets; Szegedy *et al.*, 2015]. We decompose the model so as to not separate layers belonging to the same Inception module, which results in 13 units, the last one being the single fully-connected classification layer of the architecture. We first experiment on synthetic transformations of natural images. For this setting, we use a network pre-trained on ILSVRC2012-train. We then split ILSVRC2012-val in three parts. 25k images are used to create target datasets, 5k are kept for validation and the remaining 20k are used for testing. Second, we consider the more challenging setting of stylistic transformations using the PACS dataset [Li *et al.*, 2017], initially introduced for the task of domain generalization: We use art paintings, cartoons and sketches, as target domains, which we further split into train/val/test sets. In this setting, the target task is a subset of the source ILSVRC classification task (ignoring the "person" class in PACS as it does not have a straight-forward equivalent in ILSVRC classes).

Evaluation. We measure performance as `top-1` classification accuracy, and `top-5` for ILSVRC-based domains. We use the same hyperparameters as were used during training of the base source network. As is common, for fine-tuning, we use a lower base learning rate: 10^{-3} for the small convolutional networks, and 10^{-4} for the Inception2 networks. We train all models using the Adam [Kingma and Ba, 2015] optimizer. As mentioned previously, we also employ an early stopping criterion based on validation accuracy, regularly computed during training (every 5-10 epochs). This also dampens the negative effect of overfitting in scenarios that are overly prone to it (*e.g.* `ft-all` with small sample size and a large network). Finally, in the very scarce data setting ($\sim$ 1 image per class) we report metrics averaged over 20 runs, to avoid a potential bias towards the sampled training images.

Baselines. We consider the two most common fine-tuning strategies as baselines: Starting from the source pre-trained network, `ft-all` consists in fine-tuning *all* layers on the training set from the target domain, while `ft-fc`, corresponds to fine-tuning only the last *fully-connected* units of the network, while keeping earlier units frozen. For architectures that have two fully connected layers, we consider baselines `ft-fc` (1) and `ft-fc` (2), corresponding to fine-tuning only the last, or the last two fully-connected layers respectively. Additionally, we consider scaling and shifting operations as in [Sun *et al.*, 2019b] and refer to this baseline as `ft-ss`: It consists in fine-tuning the last classification layer as well as lightweight kernel-scaling and bias-shifting parameters at every layer. Thus, similar to `ft-all`, `ft-ss` acts on all levels of the architecture, but on a *small* set of additional trainable parameters, hoping to prevent overfitting problems while retaining the knowledge present in the pre-trained network.

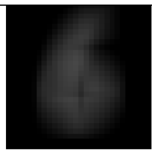
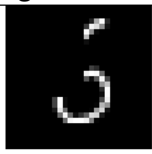
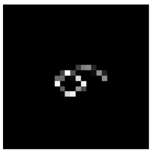
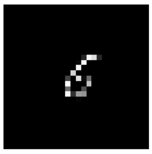
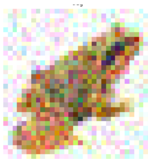
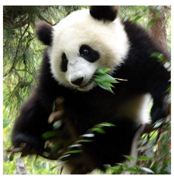
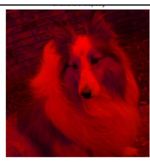
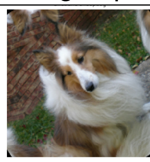
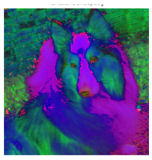
Source	Target domains
 **MNIST** (subset) [LeCun *et al.*, 1998] 25k images, 10 classes 4-layers `top-1`: 0.989	 **Blurry** `top-1`: 0.748  **Occluded** `top-1`: 0.581  **MNIST-M** [2016] `top-1`: 0.439  **Transform (random)** `top-1`: 0.322  **SVHN** [2011] `top-1`: 0.211  **Transform (fixed)** `top-1`: 0.160 **ratios** $\sim$ 3, 30, 300 and 3k images per class
 **CIFAR** (subset) [Krizhevsky and Hinton, 2009] 18k images, 9 classes 7-layers `top-1`: 0.738	 **Noisy** `top-1`: 0.540  **Blurry** `top-1`: 0.324  **QuickDraw** [2012] `top-1`: 0.291 **ratios** $\sim$ 2, 20, 200 and 2k images per class
 **ILSVRC** [Russakovsky *et al.*, 2015] **('12 train split)** 1M images, 1000 classes Inception2 `top-5`: 0.918	 **YUV** `top-5`: 0.841  **Fixed rotation** `top-5`: 0.743  **Fixed scaling (symmetric pad)** `top-5`: 0.519  **Fixed scaling (stretch pad)** `top-5`: 0.440  **HSV** `top-5`: 0.384 **ratios** $\sim$ 2, 12 and 25 images per class  **Art** [2017] `top-1`: 0.532  **Cartoon** `top-1`: 0.346  **Sketch** `top-1`: 0.142 **ratios** $\sim$ 2, 20 and 200 images per class

Table 4.2: Source domains and architectures (*left*) we consider, with the corresponding target domains (*right*) and the training dataset **subsampling ratios** we consider, as the average number of images per class: the last entry corresponds to the full dataset size.

4.1.5 Main results

In Tables 4.3, 4.4 and 4.5, we report results of the proposed methods and baselines on the MNIST, CIFAR and ILSVRC-based settings respectively. We observe that `flex` outperforms fine-tuning baselines in almost all settings. It very rarely loses to the `ft-fc` baseline, but is sometimes tied with `ft-all`, which is a subcase of `flex` and `fast-flex` through the selection criterion. More interestingly, in terms of absolute values, we observe that when `flex` strictly wins, *i.e*. when it reaches the best accuracy and not in a tie with `ft-all`, it typically does so by a higher margin than in the reverse scenario, *i.e*. when one of the baselines strictly wins.

MNIST		flex			ft-			
		flex	fast	faster	fc (1)	fc (2)	ss	all
ratio: 3 images per class								
Blurry	(0.75)	**0.890**	0.860	0.857	0.867	0.846	0.862	0.851
Occluded	(0.58)	**0.695**	**0.664**	0.661	0.644	0.654	0.662	0.660
MNIST-M	(0.44)	**0.564**	**0.564**	0.528	0.524	0.523	0.540	0.528
SVHN	(0.21)	**0.426**	**0.426**	0.414	0.365	0.412	0.403	**0.426**
Rand Tr.	(0.32)	0.400	0.400	0.399	0.391	**0.401**	0.395	0.396
Fixed Tr.	(0.16)	**0.776**	**0.776**	0.770	0.743	0.768	0.752	**0.776**
ratio: 30 images per class								
Blurry	(0.75)	0.926	0.926	0.926	0.921	**0.928**	**0.928**	0.921
Occluded	(0.58)	**0.806**	**0.806**	0.801	0.785	0.801	0.792	**0.806**
MNIST-M	(0.44)	**0.683**	**0.683**	0.671	0.615	0.670	0.675	**0.683**
SVHN	(0.21)	**0.669**	**0.669**	0.572	0.451	0.595	0.657	**0.669**
Rand Tr.	(0.32)	**0.644**	**0.644**	**0.644**	0.573	0.638	0.634	0.625
Fixed Tr.	(0.16)	**0.908**	0.887	0.879	0.839	0.875	0.866	0.887

MNIST		flex			ft-			
		flex	fast	faster	fc (1)	fc (2)	ss	all
ratio: 300 images per class								
Blurry	(0.75)	**0.967**	**0.967**	**0.967**	0.941	0.964	0.957	0.965
Occluded	(0.58)	0.873	0.873	0.873	0.849	**0.875**	0.865	0.870
MNIST-M	(0.44)	**0.828**	**0.828**	0.795	0.704	0.797	0.783	**0.828**
SVHN	(0.21)	**0.849**	**0.849**	0.812	0.558	0.800	0.798	**0.849**
Rand Tr.	(0.32)	**0.843**	**0.843**	0.830	0.683	0.836	0.760	**0.843**
Fixed Tr.	(0.16)	**0.955**	**0.955**	0.947	0.876	0.945	0.916	**0.955**
ratio: 3000 images per class								
Blurry	(0.75)	**0.987**	**0.987**	0.981	0.955	0.982	0.970	**0.987**
Occluded	(0.58)	**0.917**	**0.917**	0.915	0.867	0.912	0.893	**0.917**
MNIST-M	(0.44)	**0.895**	**0.895**	0.857	0.739	0.868	0.846	**0.895**
SVHN	(0.21)	**0.909**	**0.909**	0.851	0.670	0.877	0.848	**0.909**
Rand Tr.	(0.32)	**0.941**	**0.941**	0.900	0.733	0.909	0.835	**0.941**
Fixed Tr.	(0.16)	**0.979**	**0.979**	0.963	0.900	0.969	0.943	**0.979**

Table 4.3: Small-scale (MNIST) settings results comparing our proposed `flex`, `fast-flex` and `faster-flex`, to fine-tuning baselines `ft-all` and `ft-fc`, and the `ft-ss` baseline. In each table, the first column lists each *source*→ *target* domain shifts, with the base accuracy reached by the pretrained source network on the target test set. Bold entries indicate the score is better than that of *all* baselines (`ft-`).

CIFAR		flex			ft-			
		flex	fast	faster	fc (1)	fc (2)	ss	all
ratio: 2 images per class								
Blurry	(0.32)	**0.514**	**0.514**	0.344	0.408	0.427	0.490	0.476
Noisy	(0.54)	**0.618**	**0.618**	**0.618**	0.555	0.566	0.603	0.582
QDraw	(0.29)	**0.392**	0.391	0.386	0.359	0.380	0.378	0.391
ratio: 20 images per class								
Blurry	(0.32)	**0.577**	**0.577**	0.512	0.444	0.501	0.569	**0.577**
Noisy	(0.54)	**0.624**	**0.624**	**0.624**	0.583	0.597	0.618	0.621
QDraw	(0.29)	0.518	0.517	0.517	0.475	**0.525**	0.495	0.501
ratio: 200 images per class								
Blurry	(0.32)	0.609	0.608	0.566	0.525	0.552	**0.623**	0.608
Noisy	(0.54)	0.644	0.629	0.629	0.602	0.620	**0.653**	0.613
QDraw	(0.29)	0.663	0.667	0.667	0.569	0.662	0.643	**0.671**
ratio: 2000 images per class								
Blurry	(0.32)	**0.689**	**0.689**	0.644	0.560	0.609	0.685	**0.689**
Noisy	(0.54)	0.685	0.685	0.651	0.633	0.640	**0.705**	0.685
QDraw	(0.29)	**0.786**	**0.786**	0.744	0.581	0.721	0.702	**0.786**

QuickDraw		flex			ft-			
		flex	fast	faster	fc (1)	fc (2)	ss	all
ratio: 2 images per class								
Blurry	(0.19)	**0.560**	**0.560**	**0.525**	0.290	0.325	0.327	0.386
Noisy	(0.63)	0.763	0.760	0.758	0.766	0.768	**0.782**	0.757
CIFAR	(0.20)	0.241	0.241	0.230	0.203	0.228	**0.264**	0.241
ratio: 20 images per class								
Blurry	(0.19)	0.642	0.631	0.560	0.426	0.468	**0.707**	0.631
Noisy	(0.63)	0.801	0.801	0.795	0.788	0.792	**0.805**	0.801
CIFAR	(0.20)	**0.424**	**0.424**	0.401	0.333	0.347	0.388	**0.424**
ratio: 200 images per class								
Blurry	(0.19)	**0.726**	0.722	0.595	0.471	0.583	0.724	0.722
Noisy	(0.63)	**0.815**	0.797	0.797	**0.815**	0.797	0.812	0.772
CIFAR	(0.20)	**0.601**	**0.601**	0.53	0.394	0.413	0.543	**0.601**
ratio: 2000 images per class								
Blurry	(0.19)	**0.776**	**0.776**	0.613	0.453	0.629	0.658	**0.776**
Noisy	(0.63)	0.817	**0.822**	**0.822**	0.789	0.818	0.814	0.794
CIFAR	(0.20)	**0.718**	**0.718**	0.632	0.432	0.529	0.697	**0.718**

Table 4.4: Medium-scale settings results. On the left are the CIFAR settings, and on the right are the QuickDraw settings.

ILSVRC	flex			ft-		
	flex	fast	faster	fc	ss	all
ratio: 2 images per class						
Art (0.53)	**0.669**	**0.703**	**0.655**	0.626	0.630	0.628
Cartoon (0.32)	0.639	**0.683**	0.593	0.618	0.647	0.507
Sketch (0.14)	**0.625**	**0.606**	0.414	0.554	0.581	0.337
ratio: 20 images per class						
Art (0.53)	**0.870**	**0.851**	**0.861**	0.729	0.849	0.724
Cartoon (0.32)	**0.912**	**0.893**	0.841	0.820	0.887	0.709
Sketch (0.14)	**0.852**	0.638	0.638	0.766	0.801	0.542
ratio: 200 images per class						
Art (0.53)	**0.906**	**0.906**	0.823	0.791	0.887	0.746
Cartoon (0.32)	**0.958**	0.956	0.952	0.868	0.956	0.925
Sketch (0.14)	**0.924**	**0.924**	0.890	0.767	0.916	0.875

ILSVRC	flex			ft-		
	flex	fast	faster	fc	ss	all
ratio: 1 image per class						
YUV (0.84)	**0.856**	**0.857**	**0.830**	0.775	0.574	0.817
HSV (0.38)	**0.750**	**0.750**	**0.750**	0.422	0.374	0.582
Scaling (stretch) (0.44)	**0.626**	**0.612**	**0.596**	0.361	0.425	0.595
Scaling (sym.) (0.52)	**0.703**	**0.700**	**0.700**	0.531	0.525	0.659
Rotation (0.74)	**0.771**	**0.769**	**0.750**	0.621	0.499	0.718
ratio: 12 images per class						
YUV (0.84)	**0.893**	**0.893**	**0.893**	0.835	0.699	0.808
HSV (0.38)	**0.856**	**0.856**	**0.856**	0.533	0.646	0.687
Scaling (stretch) (0.44)	**0.724**	**0.696**	**0.696**	0.502	0.584	0.653
Scaling (sym.) (0.52)	**0.770**	**0.757**	**0.757**	0.663	0.650	0.716
Rotation (0.74)	**0.826**	**0.832**	**0.812**	0.667	0.652	0.771
ratio: 25 images per class						
YUV (0.84)	**0.897**	**0.897**	**0.897**	0.839	0.716	0.818
HSV (0.38)	**0.863**	**0.863**	**0.863**	0.545	0.676	0.670
Scaling (stretch) (0.44)	**0.750**	0.706	0.706	0.515	0.597	0.675
Scaling (sym.) (0.52)	**0.787**	0.770	0.770	0.668	0.655	0.728
Rotation (0.74)	**0.837**	0.829	0.812	0.674	0.663	0.764

Table 4.5: Large-scale settings results. On the left are the ILSVRC to PACS settings, and on the right are the ILSVRC to synthetic transformations settings.

More precisely, we first observe that in the small and medium network settings as well as in the large sample size setting, as expected, flex-tuning generally chooses to fine-tune all layers, *i.e.* `flex` recovers `ft-all`. However, as the dataset size to network depth ratio decreases, fine-tuning all layers becomes more prone to overfitting. In that case, `flex` prefers to fine-tune a specific unit, which generally performs better than the `ft-fc` baseline. More generally, the behavior of `ft-fc` strongly correlates with the difficulty of the input domain shift: it performs best in settings where the source domain early layers generalize well to the target domain, *e.g.* in the noisy CIFAR setting where the small additive random noise does not impact activations significantly.

When the domain shift is more pronounced however, `ft-fc` is often outperformed by `flex` and its variants, which pick a more adequate unit to tune. This shows there is a benefit to select the unit to fine-tune, rather than restricting transfer learning to the last fully-connected layers. These conclusions also hold for `ft-ss`, although it provides a much stronger baseline than `ft-fc` and is sometimes on-par or outperforms the faster flex-tuning variants. However, its performance seems to depend on the type of domain shift: For instance, `ft-ss` performs moderately well on the colorized-ILSVRC setting. We attribute it to the fact that this setting involves a recombination of the channels which is not well captured by affine transformations of the parameters. Finally, `flex` and its variants are easier to implement in practice as they do not introduce additional parameters nor require to know how layers actually operate.

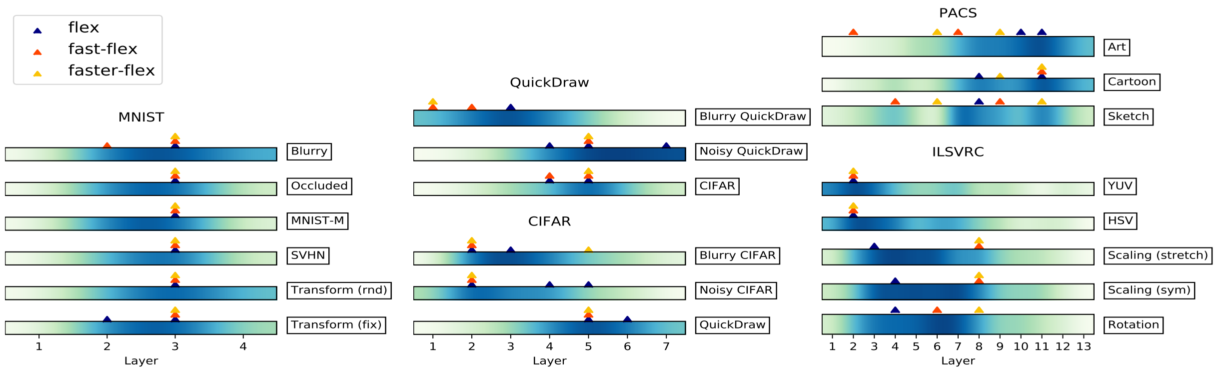


Figure 4.2: Individual units selected by `flex`, `fast-flex` and `faster-flex`, based on *validation* set accuracy. Triangles denote actual picks for `flex`, `fast-flex` (if ignoring the option to fine-tune all units) and `faster-flex`. The background values is obtained by summing the selection ranks of each unit across ratios, based on their *test performance*: in other words, the darker the color, the best performance fine-tuning this unit yields on the test set. We observe that flex-tuning's selection criterion generally chooses the best performing unit. The two variants' choices are more scattered, but overall positively correlate with flex-tuning's decisions.

Effect of the domain shift on the unit selection process. Taking a closer look at the proposed selection criteria, we observe that the most promising unit selected by flex-tuning is often an intermediate one and does not follow an obvious pattern, showing that different domain shifts affect layer representations at different depths of the architecture: This is illustrated in Figure 4.2. In particular, we observe that fast flex-tuning and even faster flex-tuning are good approximations of flex-tuning as they often pick similar units. This shows that only a few gradient updates, as is done in even faster flex-tuning, are enough to pin-point relevant units.

Based on Figure 4.2, we also identify three input domain shifts categories: For local *pixel-level transformations*, such as noisy CIFAR, or YUV/HSV ILSVRC, flex-tuning tends to choose early units. This behavior is consistent across different data scarcity settings. This coincides with the fact that *(i)* early layers are most affected by local pixel-level changes, and *(ii)* such transformations are easy to correct in early layers: *e.g*. YUV is a linear transformation of RGB. For *geometric affine transformations*, flex-tuning picks more central units of the architecture. In fact, such transformations do not change the global appearance of images and, moreover, most modern deep learning architecture are trained for invariance to small geometric manipulations (*e.g*. flip, rotations) via synthetic data augmentation, hence earlier layers are more easily transferable across these domain shifts.

Finally, the free-transform scenarios are harder to generalize: First, we observe that natural images features transfer particularly well across various domains. As such, flex-tuning often picks a late layer in the architecture for general transforms scenarios with natural images as their source domain, *e.g. photo*$\rightarrow$ $\{$*art, cartoon, sketch*$\}$. However, this does not seem to be the case in the reverse scenario, *e.g. QuickDraw*$\rightarrow$ *CIFAR* and *MNIST*$\rightarrow$ *SVHN*, which indicates that features learned from the simple structure and particular distribution of binary sketches do not generalize as well to natural images. Second, in some complex settings such as PACS, it can be the case that *two non-consecutive units* appear to be good fine-tuning candidates. This suggests that units sometimes interact in complex patterns and that considering combination of units rather than single ones could also be a viable option for complex scenarios.

Retrieval experiments. A practical benefit of fine-tuning the last layer only (`ft-fc`) is that it induces a common feature representation across the source and target domain. However this property breaks in our experimental setting, as target images are visually different from the source training set, thus fall out of the usual operation regime of the feature extractor. One can still learn a good classifier from these features [Rahimi and Recht, 2008], but the representations themselves are meaningless with respect to the source domain. In contrast, we hypothesize that tuning an *intermediate* unit as in flex-tuning can help to “mend” the feature extractor and recover aligned representations.

To analyze this phenomenon, we designe the following retrieval experiment: We extract features for the initial source domain through the source network, and for the target domain through the flex-tuned or fine-tuned network. For each target sample, we then retrieve its top-k nearest neighbors in the source domain and consider them correctly retrieved if they share the same semantic class. We then evaluate the average precision (AP@k) on this set. We report quantitative results for $k \in \{1, 10, 100\}$, average over all classes, in Table 4.6, which confirm the working hypothesis.

For reference, we also report results for representations of a network where all weights have been tuned (`ft-all`). The results follow our previous observations: For small networks, (MNIST, CIFAR) the result of fine-tuning all layers is often better aligned with the initial representations. However for the larger architectures, tuning an intermediate unit better recovers the initial source embedding space.

Source	MNIST				CIFAR			ILSVRC		PACS		
Target	Blurry	Occl.	-M	SVHN	Blurry	Noisy	QDraw	YUV	HSV	Art	Cartoon	Sketch
`ft-fc`	0.32	0.67	0.44	0.15	0.27	0.55	0.23	0.86	0.45	0.62	0.45	0.35
`ft-all`	**0.91**	**0.84**	**0.74**	**0.58**	**0.58**	0.58	**0.65**	0.79	0.48	0.82	0.81	0.87
`flex`	0.73	0.80	0.57	0.34	0.44	**0.63**	0.45	**0.94**	**0.95**	**0.90**	**0.84**	**0.90**

a) mAP@1 results

Source	MNIST				CIFAR			ILSVRC		PACS		
Target	Blurry	Occl.	-M	SVHN	Blurry	Noisy	QDraw	YUV	HSV	Art	Cartoon	Sketch
`ft-fc`	0.38	0.70	0.47	0.19	0.36	0.58	0.27	0.80	0.47	0.66	0.53	0.39
`ft-all`	**0.91**	**0.85**	**0.76**	**0.63**	**0.64**	0.64	**0.69**	0.75	0.52	0.80	0.83	0.85
`flex`	0.76	0.83	0.61	0.40	0.52	**0.67**	0.54	**0.85**	**0.85**	**0.88**	**0.86**	**0.86**

b) mAP@10 results

Source	MNIST				CIFAR			ILSVRC		PACS		
Target	Blurry	Occl.	-M	SVHN	Blurry	Noisy	QDraw	YUV	HSV	Art	Cartoon	Sketch
`ft-fc`	0.28	0.61	0.41	0.16	0.28	0.47	0.24	0.66	0.39	0.50	0.42	0.27
`ft-all`	**0.82**	**0.80**	**0.70**	**0.54**	**0.55**	0.53	**0.63**	0.61	0.42	0.65	0.70	0.66
`flex`	0.61	0.76	0.53	0.32	0.37	**0.56**	0.43	**0.71**	**0.72**	**0.75**	**0.74**	**0.71**

c) mAP@100 results

Table 4.6: Mean average precision (mAP) retrieval results for the fine-tuned and flex-tuned network embeddings of the target domain (second row), queried against the source domain embeddings (first row). Bold entries mark the highest average precision for each column.

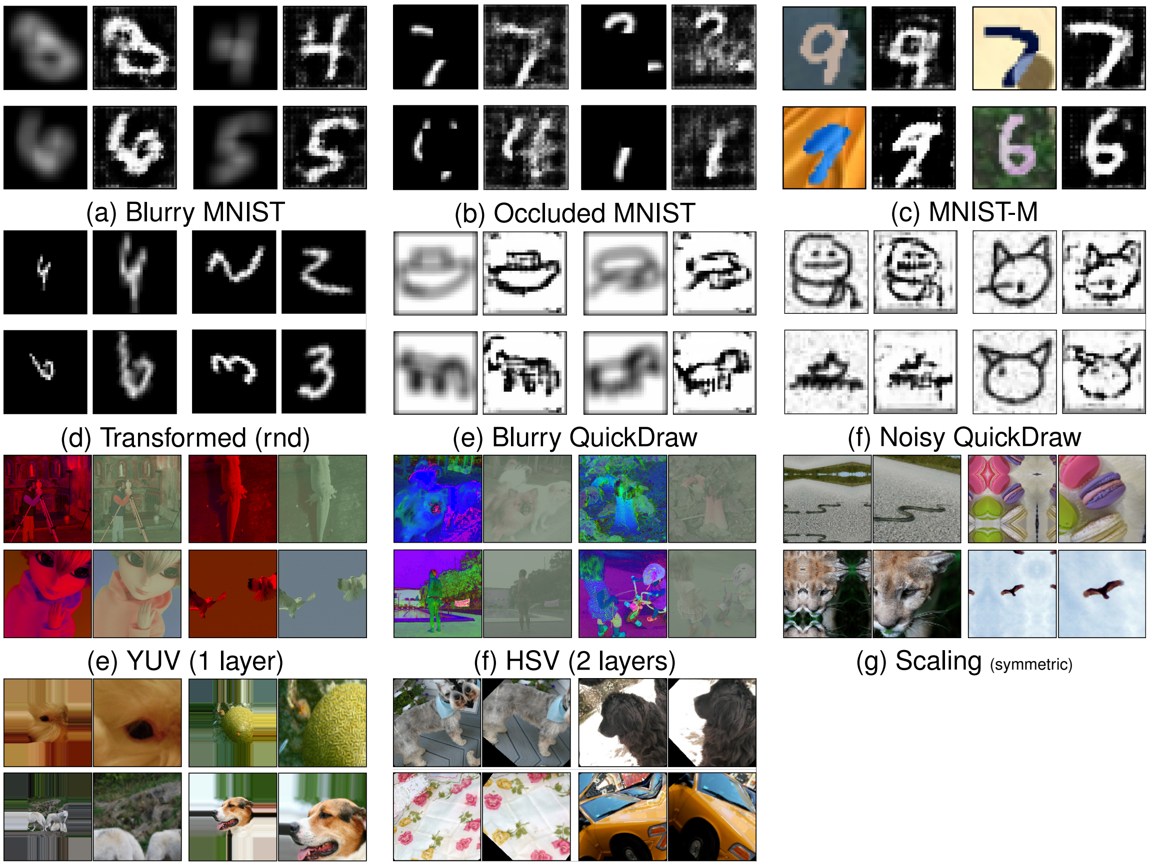
4.1.6 Towards pixel-level adaptation

While most transfer learning methods act at the feature-level, in that they learn a joint representation of two misaligned domains, recent work has also investigated the problem of directly mapping target samples back to the source domain, while keeping the network's weights untouched. In fact, pixel-level adaptation has the advantage that it only depends on the visual shift between the source and target domain and not on the architecture nor on the task at hand. Such *image-to-image transformations* have for instance been studied for domain adapation, but typically require data from both the source and target domain [Bousmalis *et al.*, 2017; Zhu *et al.*, 2017]. Building on this idea, we introduce an *image-to-image transformation unit*. It consists in a trainable pre-processing module that operates before the feature extraction phase of the pre-trained source network. We now consider this unit as an additional option in the model selection process of flex-tuning. In other words, the resulting model can either *(i)* ignore pre-processing and fine-tune an intermediate unit, or *(ii)* train the pre-processing module and feed the resulting output image to the untouched source network. In practice, we implement this image-to-image transformation unit as a small Pix2Pix network [Isola *et al.*, 2017] except in a few scenarios where we leverage our prior knowledge of the domain shift: For example, since color channel transformations occur pixel-wise, we build the preprocessing module for YUV and HSV ILSVRC as a shallow network with only 1x1 convolutions. Similarly, for geometric transforms, we use a Spatial Transformer Network [Jaderberg *et al.*, 2015] to model 2D affine transformations.

The experimental results highlight two distinct trends. First, when the gap between the source and target domains is the results of local pixel-level transformations (blur, noise, color channel changes. . .), the pre-processing unit performs very well in recovering the original source domain, as shown in Figure 4.3. This is also observed quantitatively, as the resulting model performs on par or even better (YUV and HSV cases) than other transfer learning baselines. Furthermore, in all these succesful cases, flex-tuning's selection criterion accordingly selects the image-to-image unit as the most promising unit to tune. For complex transformations, *e.g*. *photo*→ *sketch*, however, the pre-processing module performs poorly (see Figure 4.4). Furthermore, because the pre-processing module is trained from scratch, it is heavily affected by the lack of training samples in the scarce data settings. Nevertheless, flex-tuning is able to notice these failure cases and falls back to one of the other units to adapt.

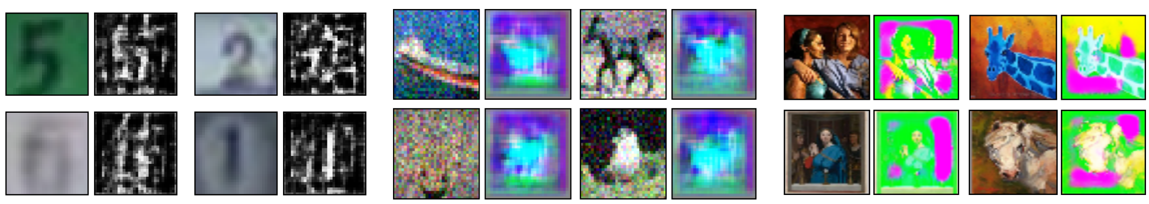
4.1.7 Conclusions

We introduce a new transfer learning method for neural networks, *flex-tuning*, that adapts a pre-trained network to a new domain by tuning just a single network unit (*e.g*. a layer or block of layers). Our experiments on a variety of scenarios show that this is a surprisingly strong adaptation technique, as long as the right unit is chosen. Specifically, we study the case where output classes stay consistent but the input data characteristics change, potentially dramatically, *e.g*. from images to sketch drawings. We find that, contrary to common practice, it is then rarely the last fully-connected unit, but rather an intermediate or early unit, that leads to the best adaptation results, and *flex-tuning* reliably identifies it. We also introduce two accelerated variants that perform almost equally good but are significantly more computationally efficient in selecting the unit to be fine-tuned.



(a) Blurry MNIST (b) Occluded MNIST (c) MNIST-M
(d) Transformed (rnd) (e) Blurry QuickDraw (f) Noisy QuickDraw
(e) YUV (1 layer) (f) HSV (2 layers) (g) Scaling (symmetric)
(h) Scaling (stretch) (i) Rotation

Figure 4.3: Example of images generated by the pre-processing module in the successful settings. For each pair, the left image is the input from the target domain and the right one is the output of the pre-processing module, mapping to the source domain.



(a) SVHN (b) Noisy CIFAR (c) Art

Figure 4.4: Example of failure cases of the pre-processing module,occuring in the most complex source-to-target domain shift settings.

4.2 Unsupervised image-to-image translation

Image-to-image translation is the problem of mapping an image from a source domain to a distinct, visually different, target domain while retaining its semantic content. Learning such mappings therefore requires an underlying understanding of the shared information between the two domains: In many cases, supervision encapsulates this knowledge in the form of labels or paired samples. This holds for instance for colorization [Zhang *et al.*, 2016]), where ground-truth pairs are easily obtained by generating grayscale images from colored inputs, or for semantic segmentation [Long *et al.*, 2015] for which pixelwise labels are usually available. Similarly, style transfer methods [Gatys *et al.*, 2016; Dumoulin *et al.*, 2017] use implicit supervision in the form of intermediate representations of an external, pre-trained, neural network.

In contrast, we consider the task of *unsupervised image-to-image translation*, which consists in learning to map an image from one domain into the style of another domain without altering its semantic content, as illustrated in Figure 4.5, from unlabeled data. We only rely on two unlabeled training image collections, one for each domain, with no known image pairings across domains. Hence, we are faced with a double *domain shift*, first in terms of global domain appearance, and second in terms of the content distribution of the two collections. This more challenging unsupervised scenario is much less explored, despite being of practical use: In fact, contrary to usual feature-based domain adaptation methods, this setting requires no annotated source data and is, by design, task-independent. This however comes at a cost, as we cannot rely on a proxy task to guide and evaluate learned representations, nor have access to explicit supervision on shared semantic content: For instance in the face-to-cartoon example, these could be captured by facial attributes such as hair color, eye color, etc. Recent work [Kim *et al.*, 2017; Zhu *et al.*, 2017; Yi *et al.*, 2017; Bousmalis *et al.*, 2017; Hoffman *et al.*, 2018] report good performance for unsupervised image-to-image translation problems where the two input domains share similar pixel-level structure (*e.g*. horses and zebras) but struggle in the presence of for more significant domain shifts (*e.g*. dogs and cats). We argue that the pixel-level constraint used in these approaches is not sufficient in general, and that we instead need a constraint in *feature space* to allow for more permissive transformations of the pixel input.

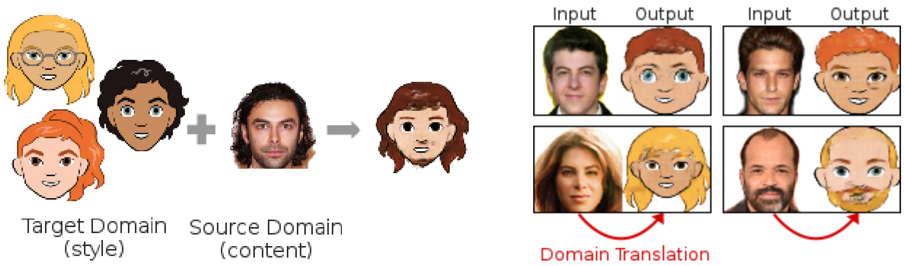


Figure 4.5: Unsupervised image-to-image translation is the task of adapting an image to the visual appearance of another domain, while preserving shared semantic content, given only two unpaired image collections (*left*). We define semantic content as characteristic attributes which are shared across domains, but do not necessarily appear the same at the pixel-level: For instance, human portraits have a pre-defined range of hair color but these take very different appearances depending on the visual domain, *e.g*. blonde hair is bright yellow in cartoons. The proposed XGAN applied on the face-to-cartoon task preserves important face characteristics such as hair style or face shape (*right*) without the need for explicit paired samples.

To this end, we propose XGAN ("Cross-GAN"), a dual adversarial autoencoder which constructs a shared representation of the two input domains, while jointly learning generative modules from one domain to the other. The main novelty lies in how we constrain the shared embedding using techniques from the domain adaptation literature, as well as a novel feature-level *semantic consistency loss*, which acts as a form of self-supervision and alleviates the need for paired examples. In particular, we evaluate the proposed model on the face-to-cartoon task, for which obtaining labeled examples is ambiguous and costly, as it is a *many-to-many mapping* problem: a photo of a face can be mapped to many valid cartoons, and vice-versa. In the following section, we review relevant recent work before discussing the proposed XGAN model in more details in Section 4.2.2 and 4.2.3. In Section 4.2.4, we then introduce *CartoonSet*, a new benchmark for unsupervised image-to-image translation. Finally, in Section 4.2.5 we report experimental results of XGAN on the face-to-cartoon task.

4.2.1 Related work

Style Transfer. Neural style transfer refers to the task of transferring the texture of a *specific* style image while preserving the pixel-level structure of an input content image [Gatys *et al.*, 2016; Johnson *et al.*, 2016]. Recent work [Li and Wand, 2016; Liao *et al.*, 2017] propose to use a dense local patch-based matching approach in the feature space, as opposed to global feature matching, allowing for convincing transformations across domains. Still, these models only perform image-specific transfer rather than learning a global *corpus-level* style. Furthermore, the generated images are usually very close to the original input in terms of pixel structure (e.g., edges) as these models focus on local texture matching, which is not suitable for more complex transformations such as face-to-cartoon.

Domain adaptation. Domain adaptation models aim to learn an aligned feature representation of two visually dissimilar domains, in order to solve a common task with high accuracy on both domains. In particular, recent domain adaptation work have tackled the problem of mapping synthetic images, easy to generate, to natural images, which are more difficult to obtain and annotate [Bousmalis *et al.*, 2016; Shrivastava *et al.*, 2017; Bousmalis *et al.*, 2017]. The generated samples are then used to train a model later applied to natural images. Contrary to our work however, they only consider pixel-level transformations. Furthermore, these methods often assume partial supervision, in that labels are available for one of the domain: These act as a way to guide the representations to capture information relevant for the task at hand. In contrast, we do not make use of such external knowledge and instead rely on self-supervision techniques. Despite these differences, we will draw inspiration from recent domain adaptation techniques [Ganin *et al.*, 2016] for minimizing the discrepancy between learned representations from two different domains.

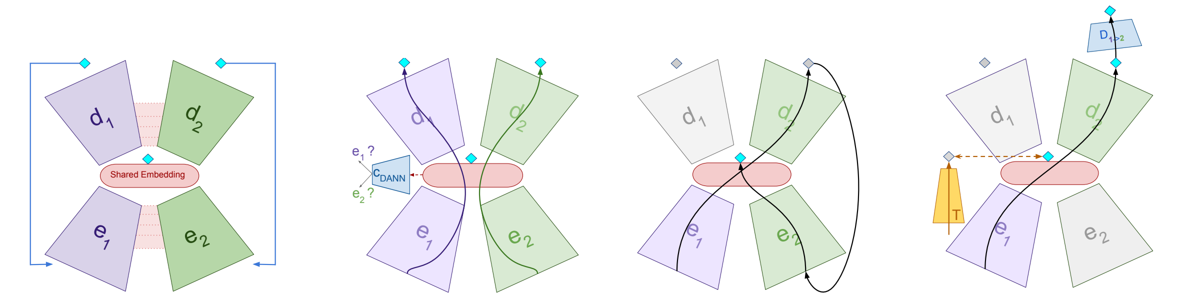
Unsupervised Image-to-Image translation. Recent work [Kim *et al.*, 2017; Zhu *et al.*, 2017; Yi *et al.*, 2017; Hoffman *et al.*, 2018] tackle the unsupervised pixel-level image-to-image translation task by jointly learning two generative adversarial networks (GANs), each capturing the transformation from one domain to the other. The model is augmented with a pixel-level cycle-consistency loss which ensures that applying each mapping followed by its reverse yields the identity function. This intuitive form of self-supervision leads to good results for pixel-level transformations, but often fails to capture significant structural changes [Zhu *et al.*, 2017]. In comparison, our proposed semantic consistency loss acts at the feature-level, allowing for more flexible transformations. Orthogonal to this line of work is UNIT [Liu *et al.*, 2017; Huang *et al.*, 2018; Ma *et al.*, 2019]. This model consists of a coupled VAEGAN architecture [Larsen *et al.*, 2016; Liu and Tuzel, 2016] with a shared embedding bottleneck, trained with pixel-level cycle-consistency. Similar to XGAN, UNIT aims to learn a joint *feature-level* representation of the two domains, however it implicitly assumes that sharing high-level layers in the architecture is a sufficient constraint, while XGAN's objective explicitly introduces a semantic consistency loss to guide the learned embeddings.

Finally, the *Domain Transfer Network* (DTN) [Taigman *et al.*, 2017; Wolf *et al.*, 2017] is closest to our work in terms of objective and applications. The DTN architecture is a single autoencoder trained to map images from a source to a target domain with self-supervised semantic consistency feedback. It was also successfully applied to the problem of feature-level image-to-image translation, in particular to the face-to-cartoon problem. Contrary to XGAN however, DTN makes use of a unique encoder for both domains, which is pre-trained on the source domain, and frozen. This assumption is very restrictive and unrealistic in many scenarios, as off-the-shelf models pre-trained on natural images do not usually generalize well to other domains. In fact, we show in Section 4.2.5 that a fixed encoder does not generalize well in the presence of a large domain shift between the two domains.

4.2.2 Proposed model: XGAN

Given two image domains, $\mathcal{X}_1$ and $\mathcal{X}_2$, for which unlabeled image collections are available, we aim to learn a mapping from one domain to the other. In practice, we assume that the two domains differ in terms of *visual appearance* but share a common *semantic content*. We however do not have any information on this shared content at training time, and only consider it as a potential proxy task for evaluating the model: In that respect, *shared semantics* can be defined as a common classification task (same object categories, but different visual modalities), or semantic attributes common to both domains (*e.g*. facial attributes in the face to cartoon task). Our underlying goal is thus to learn in an unsupervised fashion, *i.e*. without paired examples, a joint domain-invariant embedding, such that semantically similar inputs are embedded nearby in the learned feature space, regardless of the visual domain they stem from.

We build XGAN as a dual autoencoder taking inputs from domains $\mathcal{X}_1$ and $\mathcal{X}_2$, as shown in Figure 4.6 (A). We denote by e_1 the encoder and by d_1 the decoder relative to domain $\mathcal{X}_1$; likewise e_2 and d_2 for $\mathcal{X}_2$. For simplicity, we also denote by $g_{1\to 2} = d_2 \circ e_1$ the mapping from $\mathcal{X}_1$ to $\mathcal{X}_2$; likewise $g_{2\to 1} = d_1 \circ e_2$ for the reverse mapping.



(A) High-level view of the XGAN dual autoencoder architecture

(B1) Domain-adversarial autoencoder

(B2) Semantic consistency loop

(B3) GAN and Teacher loss modules

Figure 4.6: The XGAN *(A)* objective encourages the model to learn a meaningful joint embedding *(B1)* ($\mathcal{L}_{\text{rec}}$ and $\mathcal{L}_{\text{dann}}$), which should be preserved through domain translation *(B2)* ($\mathcal{L}_{\text{sem}}$), while producing output samples of good quality *(B3)* ($\mathcal{L}_{\text{gan}}$ and $\mathcal{L}_{\text{teach}}$).

The training objective is decomposed into five main components: The *reconstruction* loss, $\mathcal{L}_{\text{rec}}$, encourages the learned embedding to encode meaningful knowledge for each domain; the *domain-adversarial* loss, $\mathcal{L}_{\text{dann}}$, pushes embeddings from $\mathcal{X}_1$ and $\mathcal{X}_2$ to lie in the same subspace, bridging the domain gap at the semantic level; the *semantic consistency* loss, $\mathcal{L}_{\text{sem}}$, ensures that the learned features are invariant through the domain translation; $\mathcal{L}_{\text{gan}}$ is a standard generative adversarial network (GAN) objective, encouraging the model to generate more realistic samples. Finally, $\mathcal{L}_{\text{teach}}$ is an optional teacher loss that distills prior knowledge from a fixed pre-trained teacher embedding, when available. The total loss function is defined as the following weighted sum:

$$\mathcal{L}_{\text{XGAN}} = \mathcal{L}_{\text{rec}} + \omega_d \mathcal{L}_{\text{dann}} + \omega_s \mathcal{L}_{\text{sem}} + \omega_g \mathcal{L}_{\text{gan}} + \omega_t \mathcal{L}_{\text{teach}},$$

where the ω hyper-parameters control the contributions from each of the individual loss terms. An overview of the model is given in Figure 4.6.

A major component of XGAN is its source and target specific encoders, which incorporate *shared intermediate representations*. In contrast, most unsupervised image-to-image translation methods consider either fully separate encoders [Zhu *et al.*, 2017; Kim *et al.*, 2017; Yi *et al.*, 2017], or one unique common encoder for both domains [Taigman *et al.*, 2017]. Besides weight sharing, each loss term aims to constrain the learned representations towards alignment. We illustrate each loss term's effect on the joint

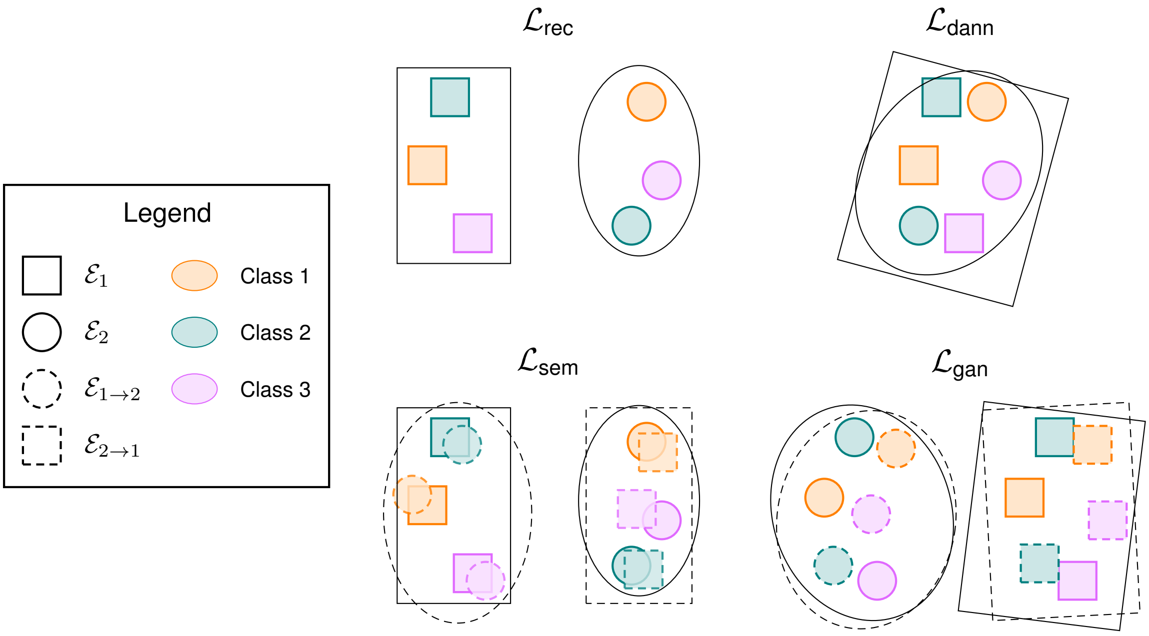


Figure 4.7: High-level illustration of the different loss terms in the XGAN objective, and their effect on the shared representation space. In this example, the two input domains, $\mathcal{X}_1$ and $\mathcal{X}_2$, contain three underlying categories, here represented in three different colors, which constitute their (unknown) shared semantics. We denote by $\mathcal{X}_{1\to2}$, the distribution of images mapped from domain $\mathcal{X}_1$ to $\mathcal{X}_2$ through the translation module $g_{1\to2}$, and likewise for $\mathcal{X}_{2\to1}$. We denote by $\mathcal{E}_1$ the image of the distribution $\mathcal{X}_1$ mapped through the corresponding encoder e_1. Similarly, $\mathcal{E}_{1\to2}$ denotes the distribution of the embeddings of $\mathcal{X}_{1\to2}$ encoded though e_2. We define $\mathcal{E}_2$ and $\mathcal{E}_{2\to1}$ in a similar manner, for the reverse transformations.

embedding space in Figure 4.7. In particular, we note the two following important insights: First, $\mathcal{L}_{\text{gan}}$ and $\mathcal{L}_{\text{sem}}$ complement each other, and taken together, they subsume the effect of $\mathcal{L}_{\text{dann}}$, in theory, as they also push the embeddings of $\mathcal{X}_1$ and $\mathcal{X}_2$ to lie close to one another. However, in practice, $\mathcal{L}_{\text{dann}}$ is much easier to train, and is particularly useful for aligning the embeddings of both domains at the beginning of training.

Second, due to the unsupervised nature of the task, the semantic loss $\mathcal{L}_{\text{sem}}$ only guarantees a *relative* alignment effect: For instance, on the example shown in Figure 4.7, it could happen that the embeddings $\mathcal{E}_{1\to2}$ for class 2 ( ) are mapped close to the ones in $\mathcal{E}_2$ for class 1 ( ). As a result, samples of class 2 in domain $\mathcal{X}_1$ would be translated to samples of class 1 in domain $\mathcal{X}_2$. Moreover, xbecause $\mathcal{L}_{\text{dann}}$ and $\mathcal{L}_{\text{gan}}$ are defined at the population level, and not conditioned on the class, they would not help in fighting this behavior. Nonetheless, we do not observe such behavior in practice.

In the rest of this section, we define and discuss each objective in more details.

Reconstruction loss, $\mathcal{L}_{\text{rec}}$. $\mathcal{L}_{\text{rec}}$ encourages the model to encode enough information on each domain to perfectly reconstruct the inputs. More specifically $\mathcal{L}_{\text{rec}} = \mathcal{L}_{\text{rec},1} + \mathcal{L}_{\text{rec},2}$ is the sum of reconstruction losses for each domain. In practice, we use the mean squared distance for image comparison:

$$\mathcal{L}_{rec,1} = \mathbb{E}_{x \sim \mathcal{P}(\mathcal{X}_1)} \left(\|x - d_1(e_1(x))\|_2 \right), \quad \text{and likewise for } \mathcal{L}_{\text{rec, 2}} \tag{4.1}$$

Domain-adversarial loss, $\mathcal{L}_{\text{dann}}$. $\mathcal{L}_{\text{dann}}$ is the domain-adversarial loss between $\mathcal{X}_1$ and $\mathcal{X}_2$, as introduced in [Ganin *et al.*, 2016]. It encourages the embeddings learned by e_1 and e_2 to lie in the same subspace, which, in turn, guarantees the soundness of the cross-domain transformations $g_{1\to2}$ and $g_{2\to1}$. More formally, this is achieved by introducing a binary classifier, c_{dann}, on top of the embedding layer that aims to categorize encoded images from *both* domains as coming from either $\mathcal{X}_1$ or $\mathcal{X}_2$ (see Figure 4.6 (B1)). The embeddings are thus trained in an adversarial manner: c_{dann} is trained to distinguish between the two domains while the encoders e_1 and e_2 simultaneously strive to minimize its classification accuracy, *i.e.* to confuse the domain classifier. Denoting model parameters by θ and by $\texttt{bce}$ the standard binary cross-entropy, we thus have the following optimization problem:

$$\min_{\theta_{e_1}, \theta_{e_2}} \max_{\theta_{\text{dann}}} \mathcal{L}_{\text{dann}}, \text{ where} \tag{4.2}$$

$$\mathcal{L}_{\text{dann}} = \mathbb{E}_{x \sim \mathcal{P}(\mathcal{X}_1)} \left[\texttt{bce}(1, c_{\text{dann}}(e_1(x))) \right] + \mathbb{E}_{x \sim \mathcal{P}(\mathcal{X}_2)} \left[\texttt{bce}(2, c_{\text{dann}}(e_2(x))) \right]$$

Semantic consistency loss, $\mathcal{L}_{\text{sem}}$. Our key contribution is a semantic consistency feedback loop that acts as self-supervision for the cross-domain translation modules $g_{1\to2}$ and $g_{2\to1}$. Intuitively, we want the learned representation of input $x \in \mathcal{X}_1$ to be preserved when translated to the other domain, $g_{1\to2}(x) \in \mathcal{X}_2$, and similarly for the reverse mapping. However this consistency property is hard to assess directly at the pixel-level as we do not have paired data or explicit supervision on the content shared across domains. Instead, we introduce a feature-level semantic consistency loss, which encourages the network to preserve the learned embedding during domain translation, as highlighted in Figure 4.6 (B2). Formally, $\mathcal{L}_{\text{sem}} = \mathcal{L}_{\text{sem},1\to2} + \mathcal{L}_{\text{sem},2\to1}$, where:

$$\mathcal{L}_{\text{sem},1\to2} = \mathbb{E}_{x \sim \mathcal{P}(\mathcal{X}_1)} \|e_1(x) - e_2(g_{1\to2}(x))\|, \quad \text{and likewise for } \mathcal{L}_{\text{sem},2\to1}, \tag{4.3}$$

where $\|\cdot\|$ denotes a distance between vectors.

GAN objective, $\mathcal{L}_{\text{gan}}$. We find that generating realistic image transformations has an incidental positive effect on the learned embeddings, as the produced samples are fed back through the encoders when computing the semantic consistency loss according to Equation 4.3. Therefore, making the distribution of translated images $\mathcal{P}(g_{2\to1}(\mathcal{X}_2))$ as close as possible to the original domain $\mathcal{P}(\mathcal{X}_1)$ ensures that the encoder e_1 does not have to cope with an additional domain shift.

In order to improve the generated samples' quality we thus add a pair of standard generative adversarial objectives [Goodfellow *et al.*, 2014], $\mathcal{L}_{\text{gan}} = \mathcal{L}_{\text{gan},1\to2} + \mathcal{L}_{\text{gan},2\to1}$, in which the generator $g_{1\to2}$ is paired against a newly introduced discriminator D_2, and likewise for $g_{2\to1}$ and D_1. In this scheme, the discriminator D_2 strives to distinguish generated samples, translated from domain $\mathcal{X}_1$ to $\mathcal{X}_2$, from real samples from $\mathcal{X}_2$. In contrast, the generator $g_{1\to2}$ aims to produce samples that confuse the discriminator.

Formally, this is captured by the following min-max objective:

$$\min_{\theta_{g_{1\to2}}} \max_{\theta_{D_2}} \mathcal{L}_{\text{gan},1\to2} \tag{4.4}$$
$$\mathcal{L}_{\text{gan},1\to2} = \mathbb{E}_{x\sim\mathcal{P}(\mathcal{X}_2)} \left(\log(D_2(x))\right) + \mathbb{E}_{x\sim\mathcal{P}(\mathcal{X}_1)} \left(\log(1 - D_2(g_{1\to2}(x)))\right),$$

and likewise for the reverse term $\mathcal{L}_{\text{gan},2\to1}$.

Optional teacher loss, $\mathcal{L}_{\text{teach}}$. We introduce an optional component to easily incorporate prior knowledge in the model when available, *e.g.* in a semi-supervised setting. $\mathcal{L}_{\text{teach}}$ encourages the learned embeddings to lie in a region of the subspace defined by the output representation of a pre-trained teacher network, T. This can be seen as a form of regularization of the learned embedding, relative to the task T was trained on. Moreover, $\mathcal{L}_{\text{teach}}$ is asymmetric by definition. It should not be used for both domains simultaneously as each term would potentially push the learned embedding in two different directions. Formally, we define $\mathcal{L}_{\text{teach}}$ as:

$$\mathcal{L}_{\text{teach}} = \mathbb{E}_{x\sim\mathcal{P}(\mathcal{X}_1)} \|T(x) - e_1(x)\| \tag{4.5}$$

where $\|\cdot\|$ is a distance between vectors.

(a) Encoder

Layer	Size
Inputs	64x64x3
conv1	32x32x32
conv2	16x16x64
(//) conv3	8x8x128
(//) conv4	4x4x256
(//) FC1	1x1x1024
(//) FC2	1x1x1024

(b) Decoder

Layer	Size
Inputs	1x1x1024
(//) deconv1	4x4x512
(//) deconv2	8x8x256
deconv3	16x16x128
deconv4	32x32x64
deconv5	64x64x3

(c) Discriminator

Layer	Size
Inputs	64x64x3
conv1	32x32x16
conv2	16x16x32
conv3	8x8x32
conv4	4x4x32
FC1	1x1x1

Table 4.7: Overview of the XGAN architecture used in practice. The encoder and decoder have the same architecture for both domains, and (//) indicates that the layer is shared across domains.

4.2.3 Architecture and Training procedure

Architecture. We use a simple mirrored convolutional architecture as the base of our architecture. Each domain autoencoder consists of an encoder e built with 5 convolutional blocks, the two last ones being shared across domains, and a decoder d with 5 transposed convolutions blocks, with the two first ones shared. Explicitly sharing parameters encourages the model to share representations at different levels of the architecture rather than only in the intermediate layer. For the discriminator, we use a similar architecture as the encoder, only without weight sharing. All convolutional blocks consist of a standard convolution, with stride 2, and followed by batch normalization and a leaky ReLU activation function. A more detailed description is given in Table 4.7. Finally, for the optional teacher network, we use the highest convolutional layer of FaceNet [Schroff *et al.*, 2015], a state-of-the-art face recognition model trained on natural images, on the domain of face images.

Training. We train our model with the ADAM optimizer [Kingma and Ba, 2015] and an initial learning rate of $1\text{e-}4$. However, the XGAN objective defined in Equation 4.1 contains two adversarial training schemes, $\mathcal{L}_{\text{gan}}$ and $\mathcal{L}_{\text{dann}}$, that lead to min-max optimization problems requiring careful optimization.

For the GAN objective, $\mathcal{L}_{\text{gan}}$, we use a standard adversarial training scheme [Goodfellow *et al.*, 2014]. However, for simplicity we only use one discriminator in practice, namely D_2. First, because GANs are notoriously unstable and hard to train, and second, because in practice one translation direction is often more relevant than the other for practical applications: In our case, this corresponds to the face-to-cartoon transformation, which is our target application. We first update the parameters of the generators $g_{1\to 2}$ and $g_{2\to 1}$ in one step. We then keep these fixed and update the parameters for the discriminator D_2. We iterate this alternating process throughout training.

The adversarial training scheme for $\mathcal{L}_{\text{dann}}$ can be implemented in practice by connecting the classifier c_{dann} and the embedding layer *via* a gradient reversal layer [Ganin *et al.*, 2016]: The feed-forward pass is unaffected, however the gradient is backpropagated to the encoders with a sign-inversion representing the min-max alternation. We perform this update simultaneously with the update of the generator's parameters.

4.2.4 The CartoonSet Dataset

Recent work on unsupervised image-to-image translation [Zhu *et al.*, 2017; Liu *et al.*, 2017] have focused on textural transformations, such as photos to realistic paintings, edges to textured image, horses to zebras etc. In contrast, in the vein of [Taigman *et al.*, 2017], we tackle the more complex face-to-carton transformation. Although previous work on this task exist, none of them provide a publicly available dataset that fits our purpose. Therefore, we introduce a new dataset of cartoon faces, CartoonSet[1], which we release publicly to further aid research on this topic.

Each cartoon face is defined by 16 components including 12 facial attributes (*e.g.* facial hair, eye shape, etc) and 4 color attributes (such as skin or hair color) which are chosen from a discrete set of RGB values. The number of options per attribute category ranges from 3 to 111, for the largest category, hairstyle. Each of these components and their variations were drawn by the same artist, resulting in approximately 250 cartoon components artworks and 10^8 possible combinations. The artwork components are divided into a fixed set of layers that define a Z-ordering for rendering. For instance, face shape is defined on a layer below eyes and glasses, so that the artworks are rendered in the correct order. Similarly, hair style needs to be defined on two layers, one behind the face and one in front. There are 8 total layers: hair back, face, hair front, eyes, eyebrows, mouth, facial hair, and glasses. The mapping from attribute to artwork

[1] CartoonSet, `https://github.com/google/cartoonset`

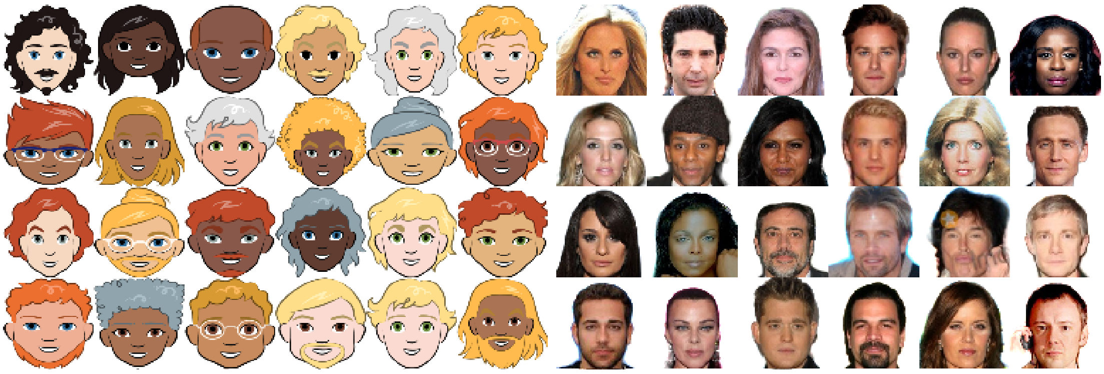


Figure 4.8: Random training images from our target cartoon domain, based on the CartoonSet dataset (left), and our source domain based on VGG-Face (right). VGG-Face samples were also centered, aligned and pre-processed with automatic portrait matting to avoid dealing with background noise.

is also defined by the artist such that any random selection of attributes produces a visually appealing cartoon without any misaligned artwork. This sometimes involves handling interaction between attributes. For example, the proper way to display a "short beard" changes for different face shapes, which required the artist to create a "short beard" artwork for each face shape.

Based on this rendering process, we create the CartoonSet dataset from arbitrary cartoon faces by randomly sampling a value for each attribute. We then filter out unrealistic attribute combinations (*e.g.* pink hair), which results in approximately 9,000 cartoons. We use this filtered set as our target image collection. Our source domain is composed of real-world frontal-face images from the VGG-Face dataset [Parkhi *et al.*, 2015]. In particular, we use an image collection consisting of 18,054 uncropped celebrity frontal face pictures. As a pre-processing step, we align the faces based on eyes and mouth location and remove the background. Finally, we randomly select and take out 20% of the images from each dataset for testing purposes, and use the remaining 80% for training. We also resize all images to 64x64 pixels.

As shown in Figure 4.8, the two domains vary significantly in appearance. In particular, cartoon faces are rather simplistic compared to real faces, and do not display as much variety (*e.g.* noses or eyebrows only have a few shape options). Furthermore, we observe a major *content distribution shift* between the two domains due to the way we collected the data: For instance, certain hair color shades (*e.g.* bright red, gray) are over-represented in the cartoon domain compared to real faces. Similarly, the cartoon dataset contains many more samples with eyeglasses compared to the source dataset.

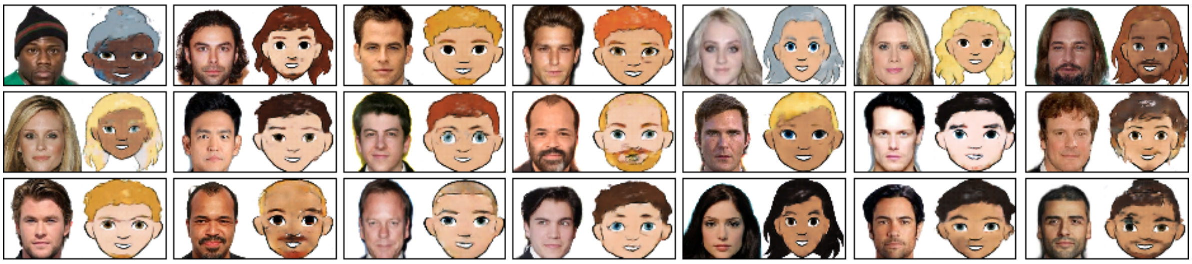


Figure 4.9: Selected samples generated by XGAN on the VGG-Face to CartoonSet task. The figure reads row-wise: For each face-to-cartoon pair, the target image (cartoon) on the right was generated from the source image (face) on the left.

4.2.5 Experiments

We experimentally evaluate XGAN on the task of converting images of frontal faces (source domain) to images of cartoon avatars (target domain), given unpaired collections of samples from each domain. Overall, XGAN is able to produce sensible cartoons both visually – the resulting cartoons look crisp and respect the specific CartoonSet style – and in terms of semantic similarity to the input samples from VGG-Face. There are some failure cases such as hair or skin color mismatch, which emerge from the weakly supervised nature of the task and the significant content shift between the two domains (*e.g*. red hair is over-represented in the target cartoon dataset). We also report selected XGAN samples that we think best illustrate its semantic consistency abilities in Figure 4.9. These show that XGAN learns a meaningful shared feature representation that preserves common face semantics. Finally, additional random samples for both cross-domain mappings are reported in Figure 4.14.

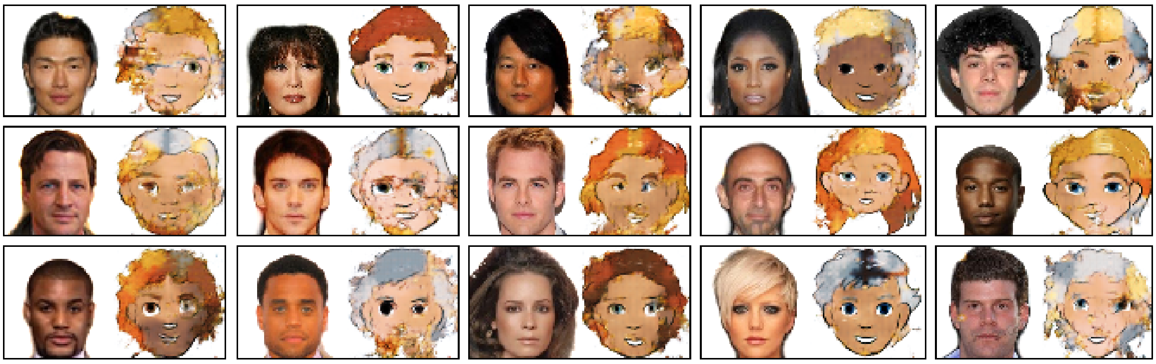
In the rest of the section, we qualitatively compare XGAN to two recent image-to-image transformation baselines, and then report results of ablation experiments.

Comparison to the DTN baseline. Our first evaluation is a qualitative comparison between the Domain Transfer Network (DTN) [Taigman *et al.*, 2017] and XGAN on the face-to-cartoon task. In particular, the DTN was also applied to the task of transferring face pictures to cartoons (bitmojis) in the original paper[2]. Following our notations, the DTN consists in a frozen, pre-trained, encoder, which is fully shared across the two domains ($e_1 = e_2$), and two decoders, specific to each domain. The training objective is

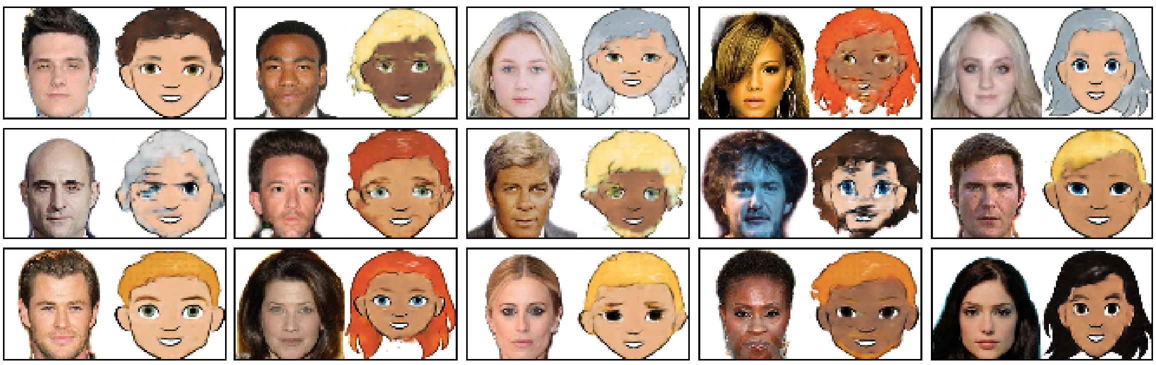
[2]The original DTN code and dataset are not publicly available, hence we instead report results from our implementation applied to the VGG-Face to CartoonSet setting.

similar to ours, except that *(i)* there is no domain adversarial loss $\mathcal{L}_{\text{dann}}$, in other words it is assumed that the fully shared encoder is sufficient to guarantee that the embeddings of $\mathcal{X}_1$ and $\mathcal{X}_2$ are well aligned, and *(ii)* the semantic consistency loss is only applied to one of the transformations, $\mathcal{L}_{\text{sem},1\to 2}$ (here, face to cartoon), and not the reverse one.

We follow the original implementation described in [Taigman *et al.*, 2017]. For the frozen encoder, we use a pre-trained FaceNet, same as our pre-trained teacher network, for fair comparison. Figure 4.10 shows the results of applying DTN and XGAN to random VGG-Face test samples. Fair quantitative evaluation metrics for style transfer are still an active research topic. Hence we choose optimal hyperparameters by manually evaluating the quality of resulting samples, focusing on accurate transfer of semantic attributes, similarity of the resulting samples to the target domain's style, and crispness of samples.



(a) *Baseline:* DTN



(b) *Proposed:* XGAN

Figure 4.10: Qualitative comparison between DTN and XGAN on random test samples. The tables are organized row-wise where each face input is mapped to the cartoon face immediately on its right.

It is clear from Figure 4.10 that DTN often fails to capture the face-to-cartoon transformation. We believe the failure of DTN is primarily due to its assumption of a fixed joint encoder for both domains, which does not model private knowledge specific to each domain. Although the decoder learns to reconstruct inputs from the target domain almost perfectly, the decoder yields samples of poor quality for the domain transfer. In fact, the encoder was originally trained on real faces inputs, hence there is no guarantee it can produce a meaningful representation for CartoonSet samples. In contrast to our dataset, the target bitmoji domain that is used in the original work [Taigman *et al.*, 2017] is visually closer to real face. This might explain the original work performance even with a fixed encoder. Nevertheless, our experiments suggest that using a fixed encoder is too restrictive and does not adapt well to our scenario. We also trained a variant of the DTN, in which we jointly fine-tune the encoder while training the decoders. This yields samples of better quality than the original DTN, however, this setup is very sensitive to training hyperparameters and prone to mode collapse.

Comparison to CycleGAN. As we have mentioned in the related work section, CycleGAN [Zhu *et al.*, 2017], DiscoGAN [Kim *et al.*, 2017] and DualGAN [Yi *et al.*, 2017] form another family of closely related work for image-to-image translation problems.

However, differently from DTN and the proposed XGAN, these models only consider a *pixel-level* cycle consistency loss and do not use any shared representations. More specifically, the model consists of two GANs, each taking care of a different domain translation, and trained with a pixel-level consistency loss, which states that mapping an image through each translation module should yield the same image.

We argue that the lack of a shared bottleneck representation hinders the CycleGAN's ability to capture high-level shared semantics between significantly different domains. To verify this hypothesis, we train CycleGAN for the face-to-cartoon task, using a pix2pix [Isola *et al.*, 2017] generator as in the original paper, which is close to the generator we use in XGAN in terms of architecture choices and size (depth and width of the network). As shown in Figure 4.11, this approach yields poor results, as the model tends to preserve local statistics, such as edges. This is in part due to the pixel-level consistency loss [Chu *et al.*, 2017], and also aggravated by the backwards connections between the encoder and the decoder in the pix2pix architecture, that further enhance pixel-level similarity.

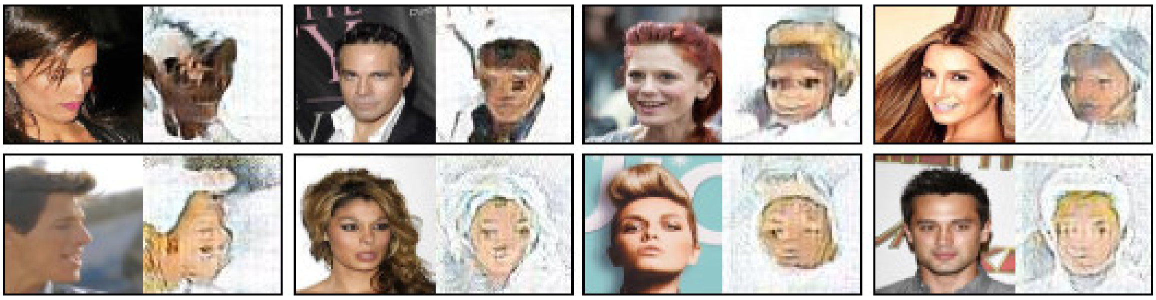
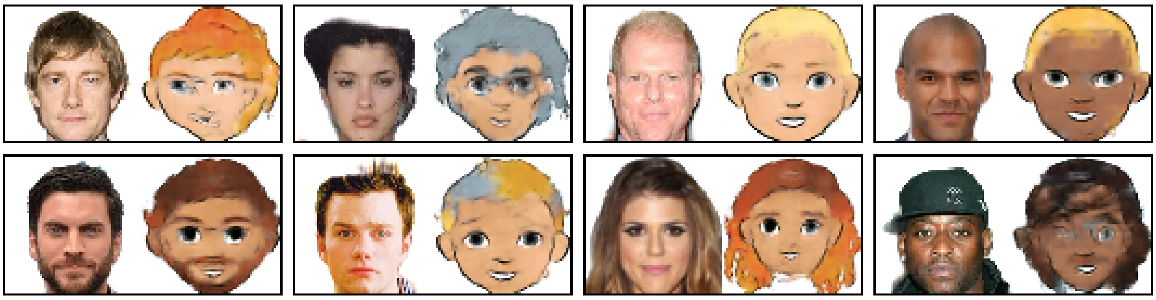


Figure 4.11: CycleGAN is not suitable for transformation between domains with very dissimilar appearances as it enforces strong pixel-level structural similarities

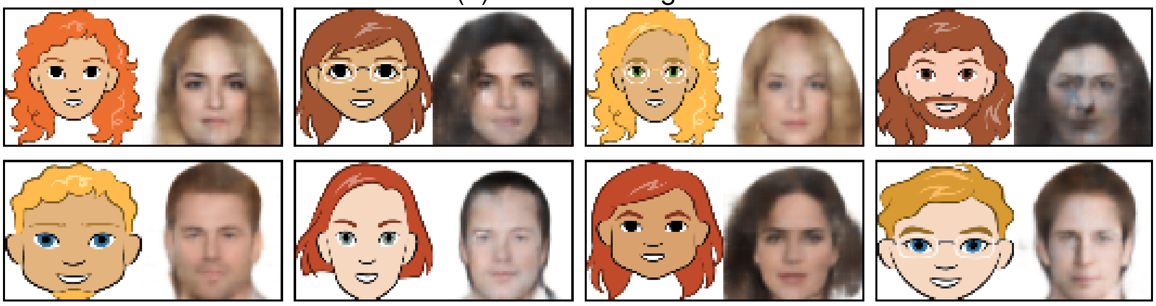
4.2.6 Ablation Experiments

We conduct a number of ablation experiments on XGAN to provide further insights in the model. We first consider training only with the reconstruction loss, $\mathcal{L}_{\text{rec}}$, and domain-adversarial loss, $\mathcal{L}_{\text{dann}}$. We show that these are a vital component of the model, and have stable training dynamics in practice. Secondly we experiment with the semantic consistency loss, $\mathcal{L}_{\text{sem}}$, and teacher loss, $\mathcal{L}_{\text{teach}}$. We show that both have complementary and beneficial regularization effects on the embedding space.

Domain-adversarial training. We first experiment with only the reconstruction and domain-adversarial losses active. These components prompt the model to *(i)* encode enough information for each decoder to correctly reconstruct images from the corresponding domain, and *(ii)* to ensure that the embedding lies in a common subspace for both domains. In practice in this setting, the model is robust to hyperparameter choices and does not require much tuning to converge to a good regime, *i.e.* low reconstruction error and around 50% accuracy for the domain-adversarial classifier. As a result of *(ii)*, applying each decoder to the output of the other domain's encoder yields reasonable cross-domain translations, albeit of low quality (see Figure 4.12). Furthermore, we observe that some simple semantics such as skin tone or gender are overall well preserved by the learned embedding due to the shared autoencoder structure. For comparison, failure modes occur in extreme cases, *e.g.* when the model capacity is too small, in which case transferred samples are of poor quality, or when the weight of the loss term $\mathcal{L}_{\text{dann}}$ is too low. In the latter case, the source and target embeddings are easily distinguishable and the cross-domain translations do not look realistic.



(a) source to target

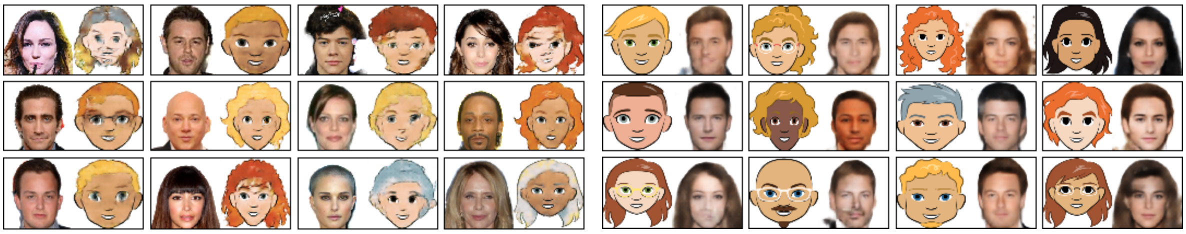


(b) target to source

Figure 4.12: Test results for XGAN with the reconstruction and domain-adversarial losses active only in the training objective $\mathcal{L}_{XGAN}$.

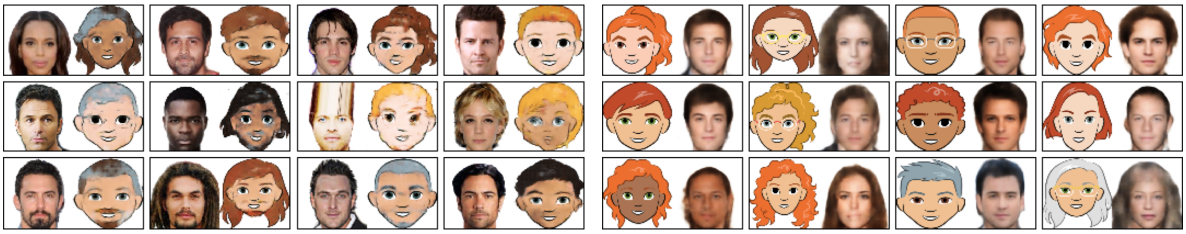
Semantic Consistency. Secondly, we investigate the benefits of adding semantic consistency in XGAN via the following three components: *Sharing high-level layers* in the autoencoder leads the model to capture common semantics earlier in the architecture. We also performed a few experiments when sharing only the middle layer in the dual autoencoder. As expected, the resulting embedding does not capture relevant shared domain semantics. Second, we use the *semantic consistency loss* as self-supervision for the learned embedding, ensuring that it is preserved through the cross-domain transformations. It also reinforces the action of the domain-adversarial loss as it constrains embeddings from the two input domains to lie close to each other. Finally, the optional *teacher loss* leads the learned source embedding to lie near the teacher output (in our case, FaceNet's representation layer). It acts in conjunction with the domain-adversarial loss and semantic consistency loss, which bring the source and target embedding distributions closer to each other.

In Figure 4.13 we report random test samples for both domain translations when ablating the teacher loss and semantic consistency loss respectively. While it is hard to draw conclusions from visual inspections, it seems that the teacher network has a positive regularization effect on the learned embedding by guiding it to a more realistic



(i) source to target (ii) target to source

(a) Teacher loss inactive



(i) source to target (ii) target to source

(b) Semantic consistency loss inactive

Figure 4.13: Results of ablating the teacher loss (top) and semantic consistency loss (bottom) in the XGAN objective $\mathcal{L}_{XGAN}$.

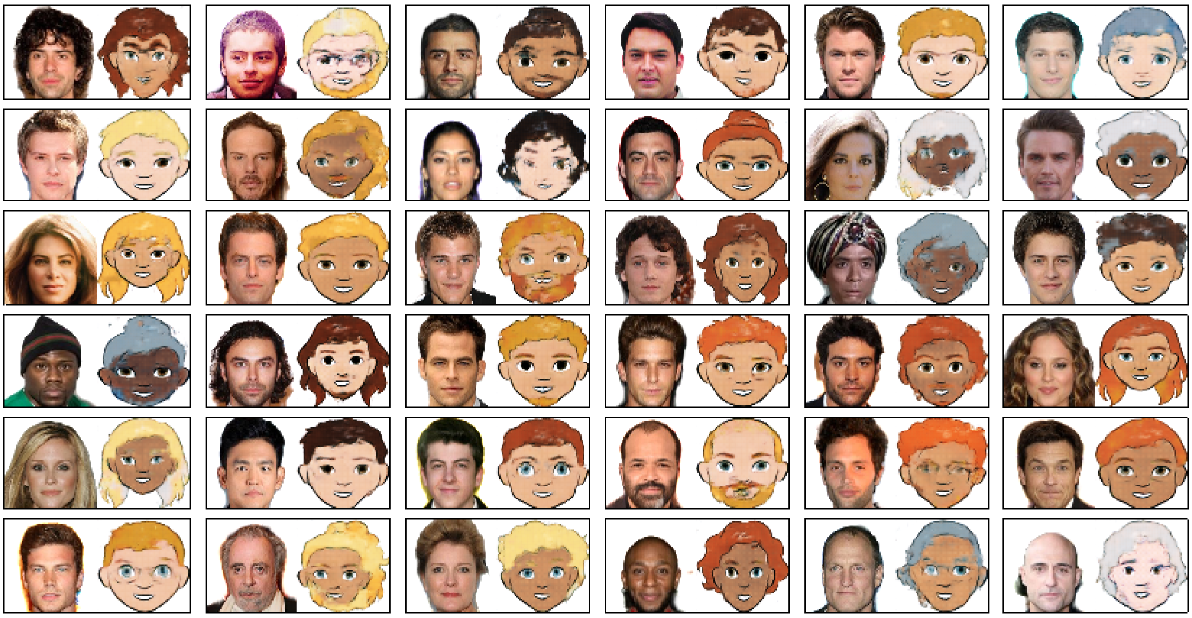
region: Training the model without the teacher loss (Figure 4.13 (a)) yields distorted samples, especially when the input is an outlier, *e.g.* person wearing a hat, or cartoons with unusual hairstyle. However, when the semantic consistency is inactive (Figure 4.13 (b)), the generated samples overall display less variety. In particular, rare attributes (*e.g.* unusual hairstyle) are not as well preserved as when this loss is present.

4.2.7 Conclusions and discussions

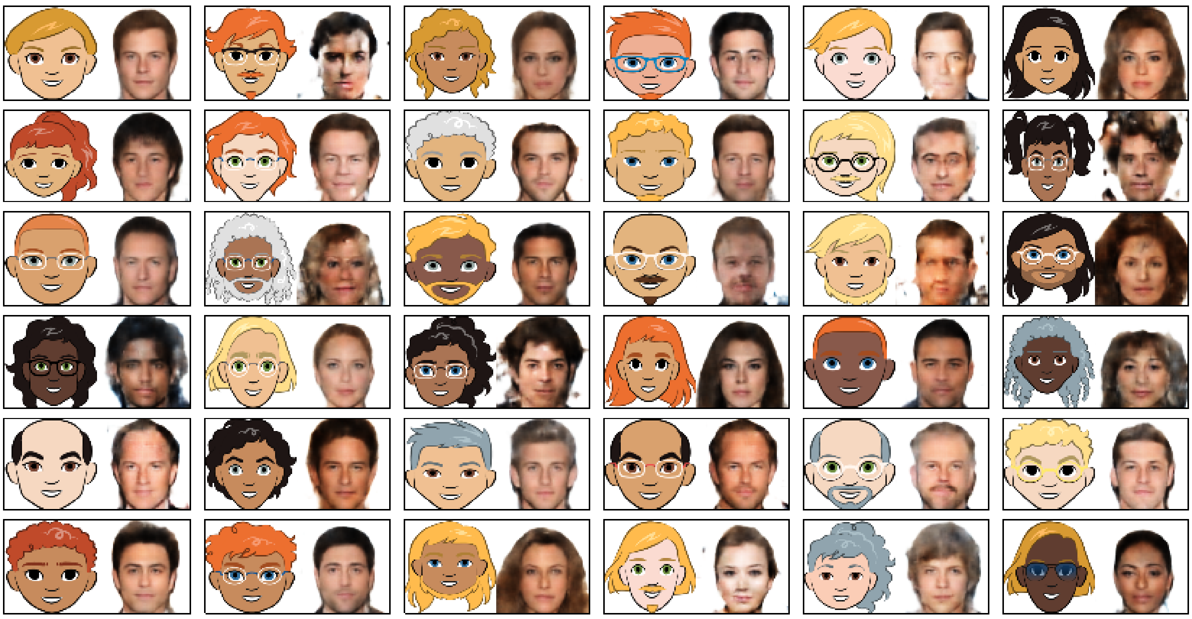
We introduced XGAN, a model for unsupervised domain translation applied to the task of semantically-consistent style transfer. In particular, we argue that, similar to the domain adaptation task, learning image-to-image translation between two structurally different domains requires learning a high-level joint semantic representation while discarding local pixel-level dependencies. Additionally, we proposed a semantic consistency loss acting on both domain translations as a form of self-supervision.

We reported promising experimental results on the task of face-to-cartoon, outperforming existing baselines. We first observe that using a simple setup where a partially shared dual autoencoder is trained with reconstruction and domain-adversarial losses already suffices to produce an embedding that captures basic semantics rather well (for instance, skin tone). However, the generated samples are of poor quality and fine-grained attributes such as facial hair are not well captured. These two problems are greatly diminished after adding the GAN loss and the proposed semantic consistency loss, respectively.

We also showed that additional weak supervision, such as a pre-trained feature representation, can easily be added to the model in the form of teacher knowledge. It acts as a good regularizer for the learned embeddings and generated samples. This is particularly useful for natural image datasets, for which off-the-shelf pre-trained models are abundant. Failure cases still exist, especially on non-representative input samples (*e.g*. a person wearing a hat) which are mapped to unrealistic cartoons. Adding the teacher loss reduces this problem by regularizing the learned embedding, however it requires additional supervision and makes the model dependent on the specific representation provided by the teacher network.



(a) Source to target mapping



(b) Target to source mapping

Figure 4.14: Additional random samples obtained by applying XGAN on faces and cartoons from the testing set

5 Conclusions and Future Work

While neural networks have demonstrated impressive performance on several computer vision tasks in recent years, one of their main limitations is their inability to generalize well outside of their training regime. Fast and data-efficient generalization to unknown scenarios is, however, a key component of “intelligence”, and thus a desirable property. We believe that constraining neural network to follow light structure priors, that do not impede their ability to freely learn feature representations from the training data itself, is a promising direction towards that goal. In that thesis, we considered two specific instances of this idea, specifically, *(i)* we studied compositionality for the task of abstract visual reasoning, and in the context of modular networks, and *(ii)* we investigated how knowledge transfer in neural networks interact with their natural layered structure.

Compositionality. Compositionality is a key component that enables for interpretable and modular models, and for generalizing to unseen concepts, without risking combinatorial explosion. However, neural networks crucially lack such a structure, and do not easily learn it without adequate constraints.

In Chapter 3, we investigated scenarios in which we can manually enforce compositionality in the case of modular, recurrent, architectures: For the task of image colorization, we leveraged recent progress on autoregressive models, which rephrase the computation of the data likelihood as a chain of easier conditional probability distributions. This chained structure allows for a tractable computation of the data likelihood. As a result, we show that our model trained directly on the exact likelihood provides a proper probabilistic framework, which is able to capture the diversity and vibrancy of chrominance in color images in practice, and induces a natural quantitative evaluation metric. For the task of object detection, we rephrase the problem of detecting small, sparsely distributed, objects as the problem of iteratively detecting *groups of objects*, until the resolution is high enough that individual objects can be detected accurately. We show that the resulting model obtains a better trade-off between the final detection accuracy and its computational efficiency. This is due to the fact that *(i)* the proposed method makes better use of available resources, as it is able to quickly assess if a region of the image is empty or not, and ignores it in the latter case, and *(ii)* that the group detection task is easier than directly detecting individual objects, which allows us to use lightweight intermediate modules without affecting accuracy.

However, both of these examples suffer from the caveat that they require *intermediate supervision* for the different modules: In the colorization case, we need color information for the whole image, and in the object detection scenario, we need to define the notion of group of objects. Unfortunately, even when a task can be decomposed in smaller modules, this supervision is not always available. We tackled such a problem in Chapter 2, for the problem of abstract visual reasoning. Given observations of specific (rule, property) compositions, the goal was to learn a model which is able to generalize to unseen compositions of known concepts. In particular, we showed that modern neural architectures, even ones which are aiming to solve similar reasoning tasks, fail at generalizing to these scenarios. We showed and formalized that, rather than a limitation of the architecture, the main bottleneck lies in the lack of structural constraints in the latent representations' space. We show that enforcing these criteria without external supervision is a challenging task, which can be linked to the domain adaptation literature. While we only provide a partial solution to this problem, we believe that this chapter constitutes an important first step towards understanding and solving the failure mode of neural networks for abstract visual reasoning and, more generally, compositional learning.

Transfer learning and layered architectures. Transfer learning is the task of transferring learned feature representations across different visual input domains. It is of key interest for real-life computer vision problems, as it allows the user to capitalize on knowledge contained in large annotated datasets, *e.g.* of natural images, and transfer it to more specific, and perhaps more data-scarce, applications. Transfer learning techniques rely on the idea that features extracted by neural networks on a specific dataset are often also meaningful for other visual tasks. However the quality of the transfer depends on *(i)* at which layer of the architecture the features lie, and *(ii)* how severe the shift between the source and target visual domains is. Both of these criteria are unfortunately hard to quantify in practice, due to the black-box nature of neural networks, and the lack of proper tools to model the distribution of images from the source and target collection.

In Chapter 4, we thus conducted an experimental analysis of how these criteria impact transfer learning. First, we investigated a flexible weight tuning scheme, which, contrary to the standard fine-tuning strategy, allows for tuning the weights of intermediate layers rather than only the last one. We conducted experiments over several domain shifts and data scarcity settings, which showed that in several scenarios, it is more beneficial to tune an intermediate layer. Motivated by this insight, we then proposed two fast and lightweight selection criteria for picking the most adequate unit to tune in generic architectures. Secondly, we investigated the problem of unsupervised image-to-image translation, which consists in learning to "translate" source images to a given target domain, without paired annotations. We proposed an hybrid model that captures both information shared across domains, by using a weight sharing strategy and drawing inspiration from domain adaptation techniques, as well as private information, specific to each domain, which is modeled by dedicated domain layers in the encoder and decoder architectures.

Based on these results, we believe that a better understanding of what information each layer of neural networks captures, and how this relates across different visual domains, is a promising direction for improving the state-of-the-art in transfer learning.

Bibliography

[Abadi *et al.*, 2015] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015, Software available from tensorflow.org. 18

[Agrawal *et al.*, 2018] A. Agrawal, D. Batra, D. Parikh, and A. Kembhavi, "Don't Just Assume; Look and Answer: Overcoming Priors for Visual Question Answering," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[Airbus, 2018] Airbus, "Airbus Ship Detection Challenge, A Kaggle competition," 2018. 63

[Alfassy *et al.*, 2019] A. Alfassy, L. Karlinsky, A. Aides, J. Shtok, S. Harary, R. S. Feris, R. Giryes, and A. M. Bronstein, "LaSO: Label-Set Operations networks for multi-label few-shot learning," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 22

[Azizpour *et al.*, 2016] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson, "Factors of Transferability for a Generic ConvNet Representation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 38(9):1790–1802, 2016. 106

[Babenko and Lempitsky, 2015] A. Babenko and V. S. Lempitsky, "Aggregating Local Deep Features for Image Retrieval," In *International Conference on Computer Vision (ICCV)*, 2015. 106

[Bachman, 2016] P. Bachman, "An Architecture for Deep, Hierarchical Generative Models," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[Ballester and Araujo, 2016] P. Ballester and R. M. Araujo, "On the Performance of GoogLeNet and AlexNet Applied to Sketches," In *Conference on Artificial Intelligence (AAAI)*, 2016. 104

[Bay *et al.*, 2006] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-Up Robust Features (SURF)," In *European Conference on Computer Vision (ECCV)*, 2006. 1, 104

[Belghazi *et al.*, 2018] M. I. Belghazi, A. Baratin, S. Rajeshwar, S. Ozair, Y. Bengio, D. Hjelm, and A. Courville, "Mutual Information Neural Estimation," In *International Conference on Machine Learing (ICML)*, 2018. 41

[Ben-David *et al.*, 2010] S. Ben-David, J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. W. Vaughan, "A Theory of Learning from Different Domains," *Machine Learning*, 79:151–175, 2010. 2, 44, 102

[Ben-David *et al.*, 2007] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira, "Analysis of Representations for Domain Adaptation," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2007.

[Blanchard *et al.*, 2011] G. Blanchard, G. Lee, and C. Scott, "Generalizing from Several Related Classification Tasks to a New Unlabeled Sample," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2011.

[Blitzer *et al.*, 2007] J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Wortman, "Learning Bounds for Domain Adaptation," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2007.

[Bottou *et al.*, 2018] L. Bottou, F. E. Curtis, and J. Nocedal, "Optimization Methods for Large-scale Machine Learning," *SIAM Review*, 60(2):223–311, 2018. 6

[Bouchacourt *et al.*, 2018] D. Bouchacourt, R. Tomioka, and S. Nowozin, "Multi-Level Variational Autoencoder: Learning Disentangled Representations from Grouped Observations," In *Conference on Artificial Intelligence (AAAI)*, 2018. 21

[Bousmalis *et al.*, 2017] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised Pixel-level Domain Adaptation with Generative Adversarial Networks," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 107, 121, 124, 126

[Bousmalis *et al.*, 2016] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain Separation Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 24, 126

[Boyd and Vandenberghe, 2004] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004. 6

[Burgess and Kim, 2018] C. Burgess and H. Kim, "3D Shapes Dataset," https://github.com/deepmind/3dshapes-dataset, 2018. 34, 36

[Cadena *et al.*, 2018] S. A. Cadena, M. A. Weis, L. A. Gatys, M. Bethge, and A. S. Ecker, "Diverse Feature Visualizations Reveal Invariances in Early Layers of Deep Neural Networks," In *European Conference on Computer Vision (ECCV)*, 2018. 23, 105

[Cai and Vasconcelos, 2018] Z. Cai and N. Vasconcelos, "Cascade R-CNN: Delving into High Quality Object Detection," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 61

[Canziani *et al.*, 2016] A. Canziani, A. Paszke, and E. Culurciello, "An Analysis of Deep Neural Network Models for Practical Applications," *arXiv preprint arXiv:1605.07678*, 2016. 57

[Cao *et al.*, 2017a] G. Cao, X. Xie, W. Yang, Q. Liao, G. Shi, and J. Wu, "Feature-Fused SSD: Fast Detection for Small Objects," In *International Conference on Graphic and Image Processing (ICGIP)*, 2017.

[Cao *et al.*, 2017b] Y. Cao, Z. Zhou, W. Zhang, and Y. Yu, "Unsupervised Diverse Colorization via Generative Adversarial Networks," In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, 2017. 86

[Carlsson *et al.*, 2008] G. Carlsson, T. Ishkhanov, V. De Silva, and A. Zomorodian, "On the Local Behavior of Spaces of Natural Images," *International Journal of Computer Vision (IJCV)*, 76(1):1–12, 2008.

[Carpenter *et al.*, 1990] P. Carpenter, M. A. Just, and P. Shell, "What One Intelligence Test Measures: A Theoretical Account of the Processing in the Raven Progressive Matrices Test," *Psychological Review*, 97(3):404–431, 1990. 28

[Chao *et al.*, 2018] Y.-W. Chao, Y. Liu, X. Liu, H. Zeng, and J. Deng, "Learning to Detect Human-Object Interactions," In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018. 21

[Charpiat *et al.*, 2008] G. Charpiat, M. Hofmann, and B. Schölkopf, "Automatic Image Colorization via Multimodal Predictions," In *European Conference on Computer Vision (ECCV)*, 2008. 85

[Chattopadhyay *et al.*, 2017] P. Chattopadhyay, R. Vedantam, R. R. Selvaraju, D. Batra, and D. Parikh, "Counting Everyday Objects in Everyday Scenes," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 27

[Cheema *et al.*, 2012] S. Cheema, S. Gulwani, and J. LaViola, "QuickDraw: Improving Drawing Experience for Geometric Diagrams," In *Conference on Human Factors in Computing Systems (SIGCHI)*, 2012. 109, 113, 116

[Chen *et al.*, 2018] R. T. Q. Chen, X. Li, R. B. Grosse, and D. Duvenaud, "Isolating Sources of Disentanglement in VAEs," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 42, 52

[Chen *et al.*, 2019] W.-Y. Chen, Y.-C. Liu, Z. Kira, Y.-C. F. Wang, and J.-B. Huang, "A Closer Look at Few-shot Classification," In *International Conference on Learning Representations (ICLR)*, 2019.

[Chen *et al.*, 2017] X. Chen, D. P. Kingma, T. Salimans, Y. Duan, P. Dhariwal, J. Schulman, I. Sutskever, and P. Abbeel, "Variational Lossy Autoencoder," In *International Conference on Learning Representations (ICLR)*, 2017.

[Chen *et al.*, 2014] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. L. Yuille, "Detect What You Can: Detecting and Representing Objects Using Holistic Models and Body Parts," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[Cheng *et al.*, 2015] Z. Cheng, Q. Yang, and B. Sheng, "Deep Colorization," In *International Conference on Computer Vision (ICCV)*, 2015.

[Chollet, 2019] F. Chollet, "The Measure of Intelligence," *arXiv preprint arXiv:1911.01547*, 2019. 27, 30

[Choromanska *et al.*, 2015] A. Choromanska, M. Henaff, M. Mathieu, G. B. Arous, and Y. LeCun, "The Loss Surfaces of Multilayer Networks," In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2015. 19

[Chu *et al.*, 2016] B. Chu, V. Madhavan, O. Beijbom, J. Hoffman, and T. Darrell, "Best Practices for Fine-Tuning Visual Classifiers to New Domains," In *ECCV Workshop TASK-CV: Transferring and Adapting Source Knowledge in Computer Vision*, 2016. 106

[Chu *et al.*, 2017] C. Chu, A. Zhmoginov, and M. Sandler, "CycleGAN, a Master of Steganography," *Workshop on Machine Deception at NeurIPS*, 2017. 137

[Clevert *et al.*, 2016] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," In *International Conference on Learning Representations (ICLR)*, 2016. 90

[Crowley and Zisserman, 2014] E. J. Crowley and A. Zisserman, "The State of the Art: Object Retrieval in Paintings using Discriminative Regions," In *British Machine Vision Conference (BMVC)*, 2014. 24, 104

[Csurka *et al.*, 2004] G. Csurka, C. R. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual Categorization with Bags of Keypoints," In *Workshop on Statistical Learning in Computer Vision, European Conference on Computer Vision (ECCV)*, 2004. 1, 104

[Cybenko, 1989] G. Cybenko, "Approximation by Superpositions of a Sigmoidal Function," *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989. 14

[Dahl *et al.*, 2017] R. Dahl, M. Norouzi, and J. Shlens, "Pixel Recursive Super Resolution," In *International Conference on Computer Vision (ICCV)*, 2017.

[Dai *et al.*, 2016] J. Dai, Y. Li, K. He, and J. Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 61

[Deng *et al.*, 2012] J. Deng, A. Berg, S. atheesh, H. Su, A. Khosla, and F.-F. Li, "Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) results," 2012. 2

[Denton *et al.*, 2015] E. L. Denton, S. Chintala, R. Fergus, *et al.*, "Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015.

[Deshpande *et al.*, 2017] A. Deshpande, J. Lu, M. Yeh, and D. A. Forsyth, "Learning Diverse Image Colorization," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 86

[Deshpande *et al.*, 2015] A. Deshpande, J. Rock, and D. A. Forsyth, "Learning Large-scale Automatic Image Colorization," In *International Conference on Computer Vision (ICCV)*, 2015. 85

[Dinh *et al.*, 2014] L. Dinh, D. Krueger, and Y. Bengio, "NICE: Non-linear Independent Components Estimation," In *International Conference on Learning Representations (ICLR)*, 2014. 98

[Dinh *et al.*, 2017] L. Dinh, J. Sohl-Dickstein, and S. Bengio, "Density Estimation using Real NVP," In *International Conference on Learning Representations (ICLR)*, 2017. 98

[Donahue *et al.*, 2014] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition," In *International Conference on Machine Learing (ICML)*, 2014. 106

[Duda and Hart, 1972] R. O. Duda and P. E. Hart, "Use of the Hough Transformation to Detect Lines and Curves in Pictures," *Communications of the ACM*, 15(1):11–15, 1972. 1

[Dumoulin *et al.*, 2017] V. Dumoulin, J. Shlens, and M. Kudlur, "A Learned Representation For Artistic Style," In *International Conference on Learning Representations (ICLR)*, 2017. 124

[Engstrom *et al.*, 2019] L. Engstrom, B. Tran, D. Tsipras, L. Schmidt, and A. Madry, "Exploring the Landscape of Spatial Robustness," In *International Conference on Machine Learing (ICML)*, 2019.

[Erhan *et al.*, 2014] D. Erhan, C. Szegedy, A. Toshev, and D. Anguelov, "Scalable Object Detection using Deep Neural Networks," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 67

[Everingham *et al.*, 2015] M. Everingham, S. M. A. Eslami, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The Pascal Visual Object Classes Challenge: A Retrospective," *International Journal of Computer Vision (IJCV)*, 11(1):98–136, 2015. 60, 71

[Facebook Research, 2016] "Learning to Segment," 2016. 1

[Felzenszwalb *et al.*, 2008] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A Discriminatively Trained, Multiscale, Deformable Part Model," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2008. 3

[Fergus *et al.*, 2003] R. Fergus, P. Perona, and A. Zisserman, "Object Class Recognition by Unsupervised Scale-Invariant Learning," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2003.

[Frankle and Carbin, 2019] J. Frankle and M. Carbin, "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," In *International Conference on Learning Representations (ICLR)*, 2019. 2, 57

[Frome *et al.*, 2013] A. Frome, G. S. Corrado, J. Shlens, S. Bengio, J. Dean, M. A. Ranzato, and T. Mikolov, "DeViSE: A Deep Visual-Semantic Embedding Model," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2013. 3

[Fukushima *et al.*, 1988] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A Neural Network Model for a Mechanism of Visual Pattern Recognition," In *Artificial Neural Networks: Theoretical Concepts*, pages 136–144. IEEE Computer Society Press, Washington, DC, USA, 1988. 15

[Gan *et al.*, 2016] C. Gan, T. Yang, and B. Gong, "Learning Attributes Equals Multi-Source Domain Generalization," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 30

[Ganin *et al.*, 2016] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain Adversarial Training of Neural Networks," *Journal of Machine Learning Research (JMLR)*, 17(1):2096–2030, 2016. 44, 48, 106, 116, 126, 130, 133

[Gao *et al.*, 2018] M. Gao, R. Yu, A. Li, V. I. Morariu, and L. S. Davis, "Dynamic Zoom-in Network for Fast Object Detection in Large Images," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 63

[Gatys *et al.*, 2016] L. A. Gatys, A. S. Ecker, and M. Bethge, "Image Style Transfer Using Convolutional Neural Networks," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 106, 124, 125

[Geirhos *et al.*, 2018] R. Geirhos, D. H. J. Janssen, H. H. Schütt, J. Rauber, M. Bethge, and F. A. Wichmann, "Comparing Deep Neural Networks Against Humans: Object Recognition when the Signal Gets Weaker," *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 20

[Geman and Geman, 1984] S. Geman and D. Geman, "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 6(6):721–741, 1984.

[Gerhard *et al.*, 2015] H. E. Gerhard, L. Theis, and M. Bethge, "Modeling Natural Image Statistics," In *Biologically-inspired Computer Vision—Fundamentals and Applications*, chapter 4, pages 53–80. Wiley, 2015.

[Ghiasi *et al.*, 2017] G. Ghiasi, H. Lee, M. Kudlur, V. Dumoulin, and J. Shlens, "Exploring the Structure of a Real-time, Arbitrary Neural Artistic Stylization Network," In *British Machine Vision Conference (BMVC)*, 2017.

[Girshick, 2015] R. Girshick, "Fast R-CNN," In *International Conference on Computer Vision (ICCV)*, 2015. 61, 67

[Girshick *et al.*, 2014] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 61, 67

[Giusti *et al.*, 2016] A. Giusti, J. Guzzi, D. C. Cireşan, F. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. D. Caro, D. Scaramuzza, and L. M. Gambardella, "A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots," *IEEE Robotics and Automation Letters*, 1(2):661–667, 2016. 1

[Gong *et al.*, 2015] Y. Gong, L. Liu, M. Yang, and L. Bourdev, "Compressing Deep Convolutional Networks using Vector Quantization," *International Conference on Learning Representations (ICLR)*, 2015. 57

[Gonzalez-Garcia *et al.*, 2018] A. Gonzalez-Garcia, D. Modolo, and V. Ferrari, "Objects as Context for Part Detection," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[Goodfellow *et al.*, 2014] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative Adversarial Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014. 86, 131, 132

[Gopalan *et al.*, 2011] R. Gopalan, R. Li, and R. Chellappa, "Domain Adaptation for Object Recognition: An Unsupervised Approach," In *International Conference on Computer Vision (ICCV)*, 2011. 106

[Gregor *et al.*, 2016] K. Gregor, F. Besse, D. Jimenez Rezende, I. Danihelka, and D. Wierstra, "Towards Conceptual Compression," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[Gregor *et al.*, 2015] K. Gregor, I. Danihelka, A. Graves, D. Rezende, and D. Wierstra, "DRAW: A Recurrent Neural Network For Image Generation," In *International Conference on Machine Learing (ICML)*, 2015.

[Guadarrama *et al.*, 2017] S. Guadarrama, R. Dahl, D. Bieber, M. Norouzi, J. Shlens, and K. Murphy, "PixColor: Pixel Recursive Colorization," In *British Machine Vision Conference (BMVC)*, 2017. 87

[Gulrajani *et al.*, 2017] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville, "PixelVAE: A Latent Variable Model for Natural Images," In *International Conference on Learning Representations (ICLR)*, 2017.

[Han *et al.*, 2015] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning Both Weights and Connections for Efficient Neural Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 57

[He *et al.*, 2014] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," In *European Conference on Computer Vision (ECCV)*, 2014. 62

[He *et al.*, 2016] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 14, 57, 90

[Higgins *et al.*, 2017] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner, "Beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework," In *International Conference on Learning Representations (ICLR)*, 2017. 21, 43

[Hill *et al.*, 2019] F. Hill, A. Santoro, D. Barrett, A. Morcos, and T. Lillicrap, "Learning to Make Analogies by Contrasting Abstract Relational Structure," In *International Conference on Learning Representations (ICLR)*, 2019. 29, 32

[Ho *et al.*, 2019] J. Ho, X. Chen, A. Srinivas, Y. Duan, and P. Abbeel, "Flow++: Improving Flow-Based Generative Models with Variational Dequantization and Architecture Design," In *International Conference on Machine Learing (ICML)*, 2019. 98

[Hoffman *et al.*, 2018] J. Hoffman, E. Tzeng, T. Park, J. Zhu, P. Isola, K. Saenko, A. A. Efros, and T. Darrell, "CyCADA: Cycle-Consistent Adversarial Domain Adaptation," In *International Conference on Machine Learing (ICML)*, 2018. 124, 126

[Hornik *et al.*, 1989] K. Hornik, M. Stinchcombe, and H. White, "Multilayer Feedforward Networks Are Universal Approximators," *Neural Networks*, 2(5):359–366, 1989. 14

[Hoshen and Werman, 2017] D. Hoshen and M. Werman, "IQ of Neural Networks," *arXiv preprint arXiv:1710.01692*, 2017. 28, 29, 31, 47

[Huang *et al.*, 2016] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, and K. Murphy, "Speed/Accuracy Trade-offs for Modern Convolutional Object Detectors," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 63

[Huang *et al.*, 2018] X. Huang, M.-Y. Liu, S. Belongie, and J. Kautz, "Multimodal Unsupervised Image-to-image Translation," In *European Conference on Computer Vision (ECCV)*, 2018. 126

[Hudson and Manning, 2019] D. A. Hudson and C. D. Manning, "GQA: A New Dataset for Compositional Question Answering over Real-world Images," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 27

[Iizuka *et al.*, 2016] S. Iizuka, E. Simo-Serra, and H. Ishikawa, "Let there be Color!: Joint End-to-end Learning of Global and Local Image Priors for Automatic Image Colorization with Simultaneous Classification," In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2016. 86, 95

[Ioffe and Szegedy, 2015] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," In *Journal of Machine Learning Research (JMLR)*, volume 37, pages 448–456, 2015. 113

[Isola *et al.*, 2017] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 86, 106, 122, 137

[Jabri *et al.*, 2016] A. Jabri, A. Joulin, and L. van der Maaten, "Revisiting Visual Question Answering Baselines," In *European Conference on Computer Vision (ECCV)*, 2016. 27

[Jaderberg *et al.*, 2015] M. Jaderberg, K. Simonyan, A. Zisserman, and K. Kavukcuoglu, "Spatial Transformer Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 122

[Jayaraman and Grauman, 2014] D. Jayaraman and K. Grauman, "Zero-shot Recognition with Unreliable Attributes," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014. 30

[J.K Kim and Serre, 2018] M. R. J.K Kim and T. Serre, "Not-So-CLEVR: Learning Same-different Relations Strains Feedforward Neural Networks," In *Royal Society Interface Focus*, 2018. 27, 30

[Johnson *et al.*, 2016] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual Losses for Real-time Style Transfer and Super-resolution," In *European Conference on Computer Vision (ECCV)*, 2016. 106, 125

[Johnson *et al.*, 2017] J. Johnson, B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick, and R. Girshick, "CLEVR: A Diagnostic Dataset for Compositional Language and Elementary Visual Reasoning," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 27

[Jégou *et al.*, 2012] H. Jégou, F. Perronnin, M. Douze, J. Sánchez, P. Pérez, and C. Schmid, "Aggregating Local Image Descriptors into Compact Codes," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 34(9):1704–1716, 2012. 1

[Kalchbrenner *et al.*, 2017] N. Kalchbrenner, A. van den Oord, K. Simonyan, I. Danihelka, O. Vinyals, A. Graves, and K. Kavukcuoglu, "Video Pixel Networks," In *International Conference on Machine Learing (ICML)*, 2017.

[Kato *et al.*, 2018] K. Kato, Y. Li, and A. Gupta, "Compositional Learning for Human Object Interaction," In *European Conference on Computer Vision (ECCV)*, 2018. 21

[Kawaguchi, 2016] K. Kawaguchi, "Deep Learning without Poor Local Minima," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 19

[Kawaguchi *et al.*, 2020] K. Kawaguchi, L. P. Kaelbling, and Y. Bengio, "Generalization in Deep Learning," *Mathematics of Deep Learning*, 2020. 2

[Kawulok and Smolka, 2010] M. Kawulok and B. Smolka, "Competitive Image Colorization," In *IEEE International Conference on Image Processing (ICIP)*, 2010. 85

[Kim *et al.*, 2019] B. Kim, E. Reif, M. Wattenberg, and S. Bengio, "Do Neural Networks Show Gestalt Phenomena? An Exploration of the Law of Closure," *arXiv preprint arXiv:1903.01069*, 2019.

[Kim *et al.*, 2020] J. Kim, D. Linsley, K. Thakkar, and T. Serre, "Disentangling Neural Mechanisms for Perceptual Grouping," *International Conference on Learning Representations (ICLR)*, 2020. 30

[Kim *et al.*, 2017] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim, "Learning to Discover Cross-Domain Relations with Generative Adversarial Networks," In *International Conference on Machine Learing (ICML)*, 2017. 106, 124, 126, 128, 137

[Kingma and Ba, 2015] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," In *International Conference on Learning Representations (ICLR)*, 2015. 19, 69, 91, 114, 132

[Kingma and Dhariwal, 2018] D. P. Kingma and P. Dhariwal, "Glow: Generative Flow with Invertible 1x1 Convolutions," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 98

[Kingma *et al.*, 2016] D. P. Kingma, T. Salimans, and M. Welling, "Improving Variational Inference with Inverse Autoregressive Flow," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 85

[Kingma and Welling, 2014] D. P. Kingma and M. Welling, "Auto-encoding Variational Bayes," In *International Conference on Learning Representations (ICLR)*, 2014. 86

[Kornblith *et al.*, 2019] S. Kornblith, J. Shlens, and Q. V. Le, "Do Better ImageNet Models Transfer Better?," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 104

[Krizhevsky and Hinton, 2009] A. Krizhevsky and G. Hinton, "Learning Multiple Layers of Features from Tiny Images," Technical report, University of Toronto, 2009. 91, 109, 113, 116

[Krizhevsky *et al.*, 2012] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2012. 2, 57

[L. F. Kozachenko, 1987] N. N. L. L. F. Kozachenko, "Sample Estimate of the Entropy of a Random Vector," *Problems of Information Transmission*, 23(2):95–101, 1987. 42

[Lampert, 2010] C. H. Lampert, "An Efficient Divide-and-conquer Cascade for Nonlinear Object Detection," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010. 63

[Lampert *et al.*, 2009] C. H. Lampert, M. B. Blaschko, and T. Hofmann, "Efficient Subwindow Search: A Branch and Bound Framework for Object Localization," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 31(12):2129–2142, 2009. 63

[Lampert *et al.*, 2014] C. H. Lampert, H. Nickisch, and S. Harmeling, "Attribute-based Classification for Zero-shot Visual Object Categorization," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 36(3):453–465, 2014. 21, 30

[Larsen *et al.*, 2016] A. B. L. Larsen, S. K. Sønderby, and O. Winther, "Autoencoding Beyond Pixels using a Learned Similarity Metric," In *International Conference on Machine Learing (ICML)*, 2016. 127

[Larsson *et al.*, 2016] G. Larsson, M. Maire, and G. Shakhnarovich, "Learning Representations for Automatic Colorization," In *European Conference on Computer Vision (ECCV)*, 2016. 58, 86, 95, 96, 106

[Larsson *et al.*, 2017] G. Larsson, M. Maire, and G. Shakhnarovich, "Colorization as a Proxy Task for Visual Understanding," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 84

[Leavitt, 2011] V. M. Leavitt, “Raven Progressive Matrices,” In *Encyclopedia of Clinical Neuropsychology*, pages 2114–2115. Springer New York, 2011.

[LeCun *et al.*, 1998] Y. LeCun, C. Cortes, and C. J. C. Burges, “MNIST Handwritten Digit Database,” http://yann.lecun.com/exdb/mnist/, 1998. 104, 113, 116

[LeCun *et al.*, 1999] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio, “Object Recognition with Gradient-Based Learning,” In *Shape, Contour and Grouping in Computer Vision*, page 319, Berlin, Heidelberg, 1999. Springer-Verlag. 15

[Ledig *et al.*, 2017] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, “Photo-realistic Single Image Super-resolution Using a Generative Adversarial Network,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[Leibe *et al.*, 2004] B. Leibe, A. Leonardis, and B. Schiele, “Combined Object Categorization and Segmentation with an Implicit Shape Model,” In *ECCV Workshop on Statistical Learning in Computer Vision*, 2004.

[Leibe *et al.*, 2008] B. Leibe, A. Leonardis, and B. Schiele, “Robust Object Detection with Interleaved Categorization and Segmentation,” *International Journal of Computer Vision (IJCV)*, 77(1):259–289, 2008.

[Lempitsky and Zisserman, 2010] V. S. Lempitsky and A. Zisserman, “Learning To Count Objects in Images,” In *Conference on Neural Information Processing Systems (NeurIPS)*, 2010.

[Levin *et al.*, 2004] A. Levin, D. Lischinski, and Y. Weiss, “Colorization Using Optimization,” In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2004. 85

[Li and Wand, 2016] C. Li and M. Wand, “Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 125

[Li *et al.*, 2017] D. Li, Y. Yang, Y.-Z. Song, and T. M. Hospedales, “Deeper, Broader and Artier Domain Generalization,” In *International Conference on Computer Vision (ICCV)*, 2017. 114, 116

[Li *et al.*, 2018] H. Li, S. Jialin Pan, S. Wang, and A. C. Kot, "Domain Generalization with Adversarial Feature Learning," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[Li *et al.*, 2015] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua, "A Convolutional Neural Network Cascade for Face Detection," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 63

[Liao *et al.*, 2017] J. Liao, Y. Yao, L. Yuan, G. Hua, and S. B. Kang, "Visual Attribute Transfer Through Deep Image Analogy," In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2017. 125

[Lin *et al.*, 2017] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," In *International Conference on Computer Vision (ICCV)*, 2017. 62

[Lin *et al.*, 2014] T. Lin, M. Maire, S. J. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," In *European Conference on Computer Vision (ECCV)*, 2014. 60

[Liu *et al.*, 2017] M. Liu, T. Breuel, and J. Kautz, "Unsupervised Image-to-Image Translation Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 126, 133

[Liu and Tuzel, 2016] M.-Y. Liu and O. Tuzel, "Coupled Generative Adversarial Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 127

[Liu *et al.*, 2018] R. Liu, J. Lehman, P. Molino, F. Petroski Such, E. Frank, A. Sergeev, and J. Yosinski, "An Intriguing Failing of Convolutional Neural Networks and the CoordConv Solution," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 27, 30

[Liu *et al.*, 2016] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single Shot MultiBox Detector," In *European Conference on Computer Vision (ECCV)*, 2016. 58, 62, 64, 66, 67, 70

[Liu *et al.*, 2015] Z. Liu, P. Luo, X. Wang, and X. Tang, “Deep Learning Face Attributes in the Wild,” In *International Conference on Computer Vision (ICCV)*, 2015.

[Locatello *et al.*, 2019] F. Locatello, S. Bauer, M. Lučić, G. Rätsch, S. Gelly, B. Schölkopf, and O. F. Bachem, “Challenging Common Assumptions in the Unsupervised Learning of Disentangled Representations,” In *International Conference on Machine Learing (ICML)*, 2019. 21, 43

[Lombardi and Pant, 2015] D. Lombardi and S. Pant, “A Non-parametric k-nearest Neighbour Entropy Estimator,” *Physical Review*, 93(1), 2015. 42

[Long *et al.*, 2015] J. Long, E. Shelhamer, and T. Darrell, “Fully Convolutional Networks for Semantic Segmentation,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 24, 106, 124

[Lowe, 1999] D. G. Lowe, “Object Recognition from Local Scale-Invariant Features,” In *International Conference on Computer Vision (ICCV)*, 1999. 1, 104

[Lu *et al.*, 2017] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, “The Expressive Power of Neural Networks: A View from the Width,” In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 14

[Lübbe, 2010] E. Lübbe, *Colours in the Mind - Colour Systems in Reality: A Formula for Colour Saturation*, Books on Demand, 2010. 96

[Ma *et al.*, 2019] L. Ma, X. Jia, S. Georgoulis, T. Tuytelaars, and L. Van Gool, “Exemplar Guided Unsupervised Image-to-Image Translation,” In *International Conference on Learning Representations (ICLR)*, 2019. 126

[Ma *et al.*, 2018] S. Ma, R. Bassily, and M. Belkin, “The Power of Interpolation: Understanding the Effectiveness of SGD in Modern Over-parametrized Learning,” In *International Conference on Machine Learing (ICML)*, 2018. 19

[Mahajan *et al.*, 2018] D. Mahajan, R. Girshick, V. Ramanathan, K. He, M. Paluri, Y. Li, A. Bharambe, and L. van der Maaten, “Exploring the Limits of Weakly Supervised Pretraining,” In *European Conference on Computer Vision (ECCV)*, 2018.

[Mahendran and Vedaldi, 2015] A. Mahendran and A. Vedaldi, “Understanding Deep Image Representations by Inverting Them,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 23

[Markle and Hunt, 1988] W. Markle and B. Hunt, “Coloring a Black and White Signal using Motion Detection,” 1988, US Patent 4,755,870. 85

[Martin *et al.*, 2001] D. Martin, C. Fowlkes, D. Tal, and J. Malik, “A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics,” In *International Conference on Computer Vision (ICCV)*, 2001.

[McGreggor and Goel, 2014] K. McGreggor and A. Goel, “Confident Reasoning on Raven’s Progressive Matrices Tests,” In *Conference on Artificial Intelligence (AAAI)*, 2014. 29

[Misra *et al.*, 2017] I. Misra, A. Gupta, and M. Hebert, “From Red Wine to Red Tomato: Composition With Context,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 21, 43

[Moddemeijer, 1989] R. Moddemeijer, “On Estimation of Entropy and Mutual Information of Continuous Distributions,” *Signal Processing*, 16(3):233–248, 1989. 41

[Model Zoo] “Deep Learning Model Zoo,” `https://modelzoo.co/`. 23, 104

[Molchanov *et al.*, 2017] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, “Pruning Convolutional Neural Networks for Resource Efficient Inference,” In *International Conference on Learning Representations (ICLR)*, 2017. 57

[Morcos *et al.*, 2018] A. Morcos, M. Raghu, and S. Bengio, “Insights on Representational Similarity in Neural Networks with Canonical Correlation,” In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 2

[Morimoto *et al.*, 2009] Y. Morimoto, Y. Taguchi, and T. Naemura, “Automatic Colorization of Grayscale Images Using Multiple Images on the Web,” In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 2009. 85

[Müller and Zaum, 2005] S. Müller and D. W. Zaum, "Robust Building Detection in Aerial Images," In *International Society for Photogrammetry and Remote Sensing, Workshop CMRT*, 2005. 60

[Murari *et al.*, 2019] M. Murari, S. Manal, M. Prashant, D. Sanhita, and K. V. Santosh, "AVDNet: A Small-Sized Vehicle Detection Network for Aerial Visual Data," In *IEEE Geoscience and Remote Sensing Letters*, 2019. 62

[Nagarajan and Grauman, 2018] T. Nagarajan and K. Grauman, "Attributes as Operators," In *European Conference on Computer Vision (ECCV)*, 2018. 21, 22, 43

[Najibi *et al.*, 2019] M. Najibi, B. Singh, and L. S. Davis, "AutoFocus: Efficient Multi-Scale Inference," In *International Conference on Computer Vision (ICCV)*, 2019. 62

[Netzer *et al.*, 2011] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, "Reading Digits in Natural Images with Unsupervised Feature Learning," In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*, 2011. 116

[Novak, 1972] J. F. Novak, "Method and Apparatus for Converting Monochrome Pictures to Multi-color Pictures Electronically," 1972, US Patent 3,706,841. 85

[Olah *et al.*, 2017] C. Olah, A. Mordvintsev, and L. Schubert, "Feature Visualization," *Distill*, 2017, https://distill.pub/2017/feature-visualization. 2, 23, 102, 105

[Olshausen and Field, 1996] B. A. Olshausen and D. J. Field, "Natural Image Statistics and Efficient Coding," *Network: Computation in Neural Systems*, 7(2):333–339, 1996.

[Palatucci *et al.*, 2009] M. Palatucci, D. Pomerleau, G. E. Hinton, and T. M. Mitchell, "Zero-shot Learning with Semantic Output Codes," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2009. 30

[Pan and Yang, 2010] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Transactions on Knowledge and Data Engineering*, 22(10):1345–1359, 2010. 101

[Paninski, 2003] L. Paninski, "Estimation of Entropy and Mutual Information," *Neural Computation*, 15(6):1191–1253, 2003.

[Papernot *et al.*, 2017] N. Papernot, M. Abadi, Ú. Erlingsson, I. J. Goodfellow, and K. Talwar, "Semi-supervised Knowledge Transfer for Deep Learning from Private Training Data," In *International Conference on Learning Representations (ICLR)*, 2017. 107

[Parascandolo *et al.*, 2018] G. Parascandolo, N. Kilbertus, M. Rojas-Carulla, and B. Schölkopf, "Learning Independent Causal Mechanisms," In *International Conference on Machine Learing (ICML)*, 2018. 107

[Parkhi *et al.*, 2015] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," In *British Machine Vision Conference (BMVC)*, 2015. 134

[Paszke *et al.*, 2019] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 18

[Peng *et al.*, 2019] X. Peng, Z. Huang, X. Sun, and K. Saenko, "Domain Agnostic Learning with Disentangled Representations," In *International Conference on Machine Learing (ICML)*, 2019.

[Peng *et al.*, 2017] X. Peng, B. Usman, N. Kaushik, J. Hoffman, D. Wang, and K. Saenko, "VisDA: The Visual Domain Adaptation Challenge," *arXiv preprint arXiv:1710.06924*, 2017.

[Peng *et al.*, 2018] X. Peng, B. Usman, K. Saito, N. Kaushik, J. Hoffman, and K. Saenko, "Syn2Real: A New Benchmark for Synthetic-to-Real Visual Domain Adaptation," *arXiv preprint arXiv:1806.09755*, 2018.

[Peyre *et al.*, 2019] J. Peyre, I. Laptev, C. Schmid, and J. Sivic, "Detecting Unseen Visual Relations Using Analogies," In *International Conference on Computer Vision (ICCV)*, 2019. 21

[Polyak and Juditsky, 1992] B. T. Polyak and A. B. Juditsky, "Acceleration of Stochastic Approximation by Averaging," *SIAM Journal on Control and Optimization*, 30(4):838–855, 1992. 91

[Purushwalkam *et al.*, 2019] S. Purushwalkam, M. Nickel, H. Mulam, and M. Ranzato, "Task-Driven Modular Networks for Zero-Shot Compositional Learning," In *International Conference on Computer Vision (ICCV)*, 2019. 22

[Raghu *et al.*, 2019] M. Raghu, C. Zhang, J. Kleinberg, and S. Bengio, "Transfusion: Understanding Transfer Learning for Medical Imaging," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 24

[Rahimi and Recht, 2008] A. Rahimi and B. Recht, "Weighted Sums of Random Kitchen Sinks: Replacing Minimization with Randomization in Learning," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2008. 120

[Ramachandran *et al.*, 2017] P. Ramachandran, T. L. Paine, P. Khorrami, M. Babaeizadeh, S. Chang, Y. Zhang, M. A. Hasegawa-Johnson, R. H. Campbell, and T. S. Huang, "Fast Generation for Convolutional Autoregressive Models," *Workshop track at ICLR*, 2017.

[Rastegari *et al.*, 2016] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks," In *European Conference on Computer Vision (ECCV)*, 2016. 57

[Raven, 2003] J. Raven, "Raven Progressive Matrices," In *Handbook of nonverbal assessment*, pages 223–237. Springer, 2003. 28, 29, 34

[Razakarivony and Jurie, 2015] S. Razakarivony and F. Jurie, "Vehicle Detection in Aerial Imagery: A Small Target Detection Benchmark," *Journal of Visual Communication and Image Representation*, 34:187–203, 2015. 69, 70, 77

[Razavian *et al.*, 2014] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition," In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition Workshops*, 2014.

[Razzak *et al.*, 2018] M. I. Razzak, S. Naz, and A. Zaib, *Deep Learning for Medical Image Processing: Overview, Challenges and Future*, pages 323–350, Springer International Publishing, 2018. 1

[Redmon *et al.*, 2016] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 24, 58, 62, 64, 66, 67, 70, 106

[Redmon and Farhadi, 2017] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 62, 64

[Redmon and Farhadi, 2018] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint arXiv:1804.02767*, 2018. 58, 62

[Reed *et al.*, 2016] S. E. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee, "Learning What and Where to Draw," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[Reed *et al.*, 2017] S. E. Reed, A. van den Oord, N. Kalchbrenner, S. G. Colmenarejo, Z. Wang, Y. Chen, D. Belov, and N. de Freitas, "Parallel Multiscale Autoregressive Density Estimation," In *International Conference on Machine Learing (ICML)*, 2017. 99

[Reed *et al.*, 2015] S. E. Reed, Y. Zhang, Y. Zhang, and H. Lee, "Deep Visual Analogy-Making," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 21

[Ren *et al.*, 2015] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2015. 62

[Robicquet *et al.*, 2016] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese, "Learning Social Etiquette: Human Trajectory Prediction In Crowded Scenes," In *European Conference on Computer Vision (ECCV)*, 2016. 70, 77

[Rosenblatt, 1958] F. Rosenblatt, "The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain," *Psychological Review*, 65(6):386–408, 1958. 14

[Ross, 2014] B. C. Ross, "Mutual Information Between Discrete and Continuous Data Sets," *PLoS ONE*, 2014. 42

[Roth and Black, 2005] S. Roth and M. J. Black, "Fields of Experts: A Framework for Learning Image Priors," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005.

[Rowley *et al.*, 1998] H. A. Rowley, S. Baluja, and T. Kanade, "Neural Network-Based Face Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 20(1):23–38, 1998. 63

[Royer, 2020] A. Royer, "ODGI: Object Detection with Grouped Instances," https://github.com/ameroyer/ODGI, 2020. 70

[Royer *et al.*, 2018] A. Royer, K. Bousmalis, S. Gouws, F. Bertsch, I. Mosseri, F. Cole, and K. Murphy, "XGAN: Unsupervised Image-to-Image Translation for Many-to-Many Mappings," In *Domain Adaptation for Visual Understanding*, pages 33–49. Springer International Publishing, 2018. 26

[Royer and Kolesnikov, 2017] A. Royer and A. Kolesnikov, "Probabilistic Image Colorization," https://github.com/ameroyer/PIC, 2017. 90, 91

[Royer *et al.*, 2017] A. Royer, A. Kolesnikov, and C. H. Lampert, "Probabilistic Image Colorization," In *British Machine Vision Conference (BMVC)*, 2017. 25

[Royer and Lampert, 2020a] A. Royer and C. H. Lampert, "A Flexible Selection Scheme for Minimum-Effort Transfer Learning," In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020. 26

[Royer and Lampert, 2020b] A. Royer and C. H. Lampert, "Localizing Grouped Instances for Efficient Detection in Low-Resource Scenarios," In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020. 25

[Royer and Lampert, 2020c] A. Royer and C. H. Lampert, "The Missing Puzzle Piece for Abstract Visual Reasoning," In *submitted*, 2020. 25

[Rozantsev *et al.*, 2018] A. Rozantsev, M. Salzmann, and P. Fua, "Residual Parameter Transfer for Deep Domain Adaptation," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 24

[Rozantsev *et al.*, 2019] A. Rozantsev, M. Salzmann, and P. Fua, "Beyond Sharing Weights for Deep Domain Adaptation," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 41:801–814, 2019. 24

[Ruderman and Bialek, 1993] D. L. Ruderman and W. Bialek, "Statistics of Natural Images: Scaling in the Woods," In *Conference on Neural Information Processing Systems (NeurIPS)*, 1993.

[Rumelhart *et al.*, 1986] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning Representations by Back-propagating Errors," *Nature*, 323(6088):533–536, 1986. 18

[Russakovsky *et al.*, 2015] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. 23, 57, 93, 104, 106, 116

[Sadeghi and Farhadi, 2011] M. A. Sadeghi and A. Farhadi, "Recognition Using Visual Phrases," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2011. 21

[Saenko *et al.*, 2010] K. Saenko, B. Kulis, M. Fritz, and T. Darrell, "Adapting Visual Category Models to New Domains," In *European Conference on Computer Vision (ECCV)*, 2010. 106

[Safran and Shamir, 2018] I. Safran and O. Shamir, "Spurious Local Minima are Common in Two-Layer ReLU Neural Networks," In *International Conference on Machine Learing (ICML)*, 2018.

[Salimans *et al.*, 2016a] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved Techniques for Training GANs," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016.

[Salimans *et al.*, 2016b] T. Salimans, A. Karpathy, X. Chen, and D. P. Kingma, "Pixel-CNN," `https://github.com/openai/pixel-cnn`, 2016.

[Salimans *et al.*, 2017] T. Salimans, A. Karpathy, X. Chen, D. P. Knigma, and Y. Bulatov, "PixelCNN++: A PixelCNN Implementation with Discretized Logistic Mixture

Likelihood and Other Modifications," In *International Conference on Learning Representations (ICLR)*, 2017. 85, 86, 87, 88, 90, 91

[Salimans and Kingma, 2016] T. Salimans and D. P. Kingma, "Weight Normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 90

[Sandler *et al.*, 2018] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 70

[Sandrew, 1992] B. Sandrew, "System and Method for Digitally Coloring Images," 1992, US Patent 5,093,717.

[Sankaranarayanan *et al.*, 2018] S. Sankaranarayanan, Y. Balaji, C. D. Castillo, and R. Chellappa, "Generate To Adapt: Aligning Domains using Generative Adversarial Networks," 2018.

[Santoro *et al.*, 2018] A. Santoro, F. Hill, D. Barrett, A. Morcos, and T. Lillicrap, "Measuring Abstract Reasoning in Neural Networks," In *International Conference on Computer Vision (ICCV)*, 2018. 28, 29, 30, 31, 32, 34, 37, 47

[Santoro *et al.*, 2017] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. W. Battaglia, and T. P. Lillicrap, "A Simple Neural Network Module for Relational Reasoning," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2017. 47

[Saxton *et al.*, 2019] D. Saxton, E. Grefenstette, F. Hill, and P. Kohli, "Analysing Mathematical Reasoning Abilities of Neural Models," In *International Conference on Learning Representations (ICLR)*, 2019. 30

[Schoenauer-Sebag *et al.*, 2019] A. Schoenauer-Sebag, L. Heinrich, M. Schoenauer, M. Sebag, L. Wu, and S. Altschuler, "Multi-Domain Adversarial Learning," In *International Conference on Learning Representations (ICLR)*, 2019. 44, 48

[Schroff *et al.*, 2015] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 132

[Shalev-Shwartz and Ben-David, 2014] S. Shalev-Shwartz and S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms*, Cambridge University Press, 2014. 7, 110

[Shang *et al.*, 2016] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and Improving Convolutional Neural Networks via Concatenated Rectified Linear Units," In *International Conference on Machine Learing (ICML)*, 2016. 90

[Shegheva and Goel, 2018] S. Shegheva and A. Goel, "The Structural Affinity Method for Solving the Raven's Progressive Matrices Test for Intelligence," In *Conference on Artificial Intelligence (AAAI)*, 2018. 29

[Shrivastava *et al.*, 2017] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from Simulated and Unsupervised Images through Adversarial Training," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 126

[Simon and Rodner, 2015] M. Simon and E. Rodner, "Neural Activation Constellations: Unsupervised Part Model Discovery with Convolutional Networks," In *International Conference on Computer Vision (ICCV)*, 2015. 106

[Simoncelli and Olshausen, 2001] E. P. Simoncelli and B. A. Olshausen, "Natural Image Statistics and Neural Representation," *Annual Review of Neuroscience*, 24(1):1193–1216, 2001.

[Simonyan and Zisserman, 2015] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *International Conference on Learning Representations (ICLR)*, 2015. 57

[Singh *et al.*, 2018] B. Singh, M. Najibi, and L. S. Davis, "SNIPER: Efficient Multi-Scale Training," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2018. 62

[Sivic and Zisserman, 2003] J. Sivic and A. Zisserman, "Video Google: A text retrieval approach to object matching in videos," In *International Conference on Computer Vision (ICCV)*, 2003. 2

[Sohl-Dickstein *et al.*, 2015] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, “Deep Unsupervised Learning using Nonequilibrium Thermodynamics,” In *International Conference on Machine Learing (ICML)*, 2015.

[Spearman, 1924] C. Spearman, “The Nature of ”Intelligence” and the Principles of Cognition,” In *Journal of Philosophy*, 1924.

[Steenbrugge *et al.*, 2018] X. Steenbrugge, S. Leroux, T. Verbelen, and B. Dhoedt, “Improving Generalization for Abstract Reasoning Tasks Using Disentangled Feature Representations,” *Workshop on Relational Representation Learning at NeurIP*, 2018. 28, 29, 43

[Steinvorth, 2010] D. Steinvorth, “Finding Swimming Pools with Google Earth: Greek Government Hauls in Billions in Back Taxes,” *SPIEGEL online*, 2010, *http://www.spiegel.de/international/europe/finding-swimming-pools-with-google-earth-greek-government-hauls-in-billions-in-back-taxes-a-709703.html*. 60

[Stephens *et al.*, 2013] G. J. Stephens, T. Mora, G. Tkačik, and W. Bialek, “Statistical Thermodynamics of Natural Images,” *Physical Review Letters*, 110(1):018701, 2013.

[Sun *et al.*, 2019a] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, “Scalability in Perception for Autonomous Driving: Waymo Open Dataset,” *arXiv preprint arXiv:1912.04838*, 2019. 1

[Sun *et al.*, 2019b] Q. Sun, Y. Liu, T.-S. Chua, and B. Schiele, “Meta-Transfer Learning for Few-Shot Learning,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 114

[Sun *et al.*, 2016] S. Sun, W. Chen, L. Wang, X. Liu, and T.-Y. Liu, “On the Depth of Deep Neural Networks: A Theoretical View,” In *Conference on Artificial Intelligence (AAAI)*, 2016. 57

[Szegedy *et al.*, 2015] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going Deeper with Convolutions," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 57, 113

[Szegedy *et al.*, 2014] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing Properties of Neural Networks," In *International Conference on Learning Representations (ICLR)*, 2014. 2, 20

[Taigman *et al.*, 2014] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the Gap to Human-level Performance in Face Verification," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[Taigman *et al.*, 2017] Y. Taigman, A. Polyak, and L. Wolf, "Unsupervised Cross-Domain Image Generation," In *International Conference on Learning Representations (ICLR)*, 2017. 127, 128, 133, 135, 136, 137

[Teney *et al.*, 2020] D. Teney, P. Wang, J. Cao, L. Liu, C. Shen, and A. van den Hengel, "V-PROM: A Benchmark for Visual Reasoning Using Visual Progressive Matrices," *Conference on Artificial Intelligence (AAAI)*, 2020. 30

[TensorFlow Hub] "TensorFlow Hub," https://www.tensorflow.org/hub. 23, 104

[TensorNets] "TensorNets," https://github.com/taehoonlee/tensornets. 23, 104, 113

[Theis *et al.*, 2016] L. Theis, A. van den Oord, and M. Bethge, "A Note on the Evaluation of Generative Models," In *International Conference on Learning Representations (ICLR)*, 2016.

[Tishby and Zaslavsky, 2015] N. Tishby and N. Zaslavsky, "Deep Learning and the Information Bottleneck Principle," In *2015 IEEE Information Theory Workshop (ITW)*, 2015. 43

[Trott *et al.*, 2018] A. Trott, C. Xiong, and R. Socher, "Interpretable Counting for Visual Question Answering," In *International Conference on Learning Representations (ICLR)*, 2018.

[Tuggener *et al.*, 2018] L. Tuggener, I. Elezi, J. Schmidhuber, M. Pelillo, and T. Stadelmann, "DeepScores: A Dataset for Segmentation, Detection and Classification of Tiny Objects," In *International Conference on Pattern Recognition (ICPR)*, 2018. 63

[Tzeng *et al.*, 2017] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell, "Adversarial Discriminative Domain Adaptation," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 24

[van den Oord *et al.*, 2016a] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A Generative Model for Raw Audio," *arXiv preprint arXiv:1609.03499*, 2016.

[van den Oord *et al.*, 2016b] A. van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, and A. Graves, "Conditional Image Generation with pixelCNN Decoders," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2016. 85, 90, 96

[van den Oord *et al.*, 2016c] A. van den Oord, N. Kalchbrenner, and K. Kavukcuoglu, "Pixel Recurrent Neural Networks," In *International Conference on Machine Learing (ICML)*, 2016. 85, 86, 87, 94

[van den Oord and Schrauwen, 2014] A. van den Oord and B. Schrauwen, "Factoring Variations in Natural Images with Deep Gaussian Mixture Nodels," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014.

[Van der Horst and Bouman, 1969] G. J. Van der Horst and M. A. Bouman, "Spatiotemporal Chromaticity Discrimination," *Journal of the Optical Society of America (JOSA)*, 59(11):1482–1488, 1969. 90

[van Steenkiste *et al.*, 2019] S. van Steenkiste, F. Locatello, J. Schmidhuber, and O. Bachem, "Are Disentangled Representations Helpful for Abstract Visual Reasoning?," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 28, 29, 43

[Vedaldi *et al.*, 2009] A. Vedaldi, V. Gulshan, M. Varma, and A. Zisserman, "Multiple Kernels for Object Detection," In *International Conference on Computer Vision (ICCV)*, 2009.

[Vera *et al.*, 2018] M. Vera, P. Piantanida, and L. R. Vega, “The Role of the Information Bottleneck in Representation Learning,” In *2018 IEEE International Symposium on Information Theory (ISIT)*, 2018. 43

[Viola and Jones, 2001] P. Viola and M. Jones, “Rapid Object Detection Using a Boosted Cascade of Simple Features,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001. 21

[Viola and Jones, 2004] P. Viola and M. J. Jones, “Robust Real-Time Face Detection,” *International Journal of Computer Vision (IJCV)*, 57(2):137–154, 2004. 2, 21, 63

[Wang and Su, 2015] K. Wang and Z. Su, “Automatic Generation of Raven’s Progressive Matrices,” In *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2015.

[Wang *et al.*, 2018a] P. Wang, Q. Wu, C. Shen, A. R. Dick, and A. van den Hengel, “FVQA: Fact-Based Visual Question Answering,” *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 40(10):2413–2427, 2018. 27

[Wang and Gupta, 2016] X. Wang and A. Gupta, “Generative Image Modeling Using Style and Structure Adversarial Networks,” In *European Conference on Computer Vision (ECCV)*, 2016.

[Wang *et al.*, 2019] Y. E. Wang, G.-Y. Wei, and D. Brooks, “Benchmarking TPU, GPU, and CPU Platforms for Deep Learning,” 2019. 57

[Wang *et al.*, 2018b] Z. Wang, Z. Dai, B. Póczos, and J. G. Carbonell, “Characterizing and Avoiding Negative Transfer,” In *CVPR*, 2018. 101

[Wolf *et al.*, 2017] L. Wolf, Y. Taigman, and A. Polyak, “Unsupervised Creation of Parameterized Avatars,” In *International Conference on Computer Vision (ICCV)*, 2017. 127

[Xia *et al.*, 2018] G.-S. Xia, X. Bai, L. Zhang, S. Belongie, J. Luo, M. Datcu, and M. Pelilo, “DOTA: A Large-Scale Dataset for Object Detection in Aerial Images,” In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[Xian *et al.*, 2018] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata, "Zero-Shot Learning - A Comprehensive Evaluation of the Good, the Bad and the Ugly," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 41(9):2251–2265, 2018.

[Xie *et al.*, 2018] W. Xie, J. A. Noble, and A. Zisserman, "Microscopy Cell Counting and Detection with Fully Convolutional Regression Networks," *Computer Methods in Biomechanics and Biomedical Engineering: Imaging & Visualization*, 6(3):283–292, 2018. 60

[Yang *et al.*, 2019] F. Yang, H. Fan, P. Chu, E. Blasch, and H. Ling, "Clustered Object Detection in Aerial Images," In *International Conference on Computer Vision (ICCV)*, 2019. 62

[Yi *et al.*, 2017] Z. Yi, H. Zhang, P. Tan, and M. Gong, "DualGAN: Unsupervised Dual Learning for Image-to-Image Translation," In *International Conference on Computer Vision (ICCV)*, 2017. 124, 126, 128, 137

[Yosinski *et al.*, 2014] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How Transferable Are Features in Deep Neural Networks?," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2014. 2, 22, 106

[Yu and Koltun, 2016] F. Yu and V. Koltun, "Multi-Scale Context Aggregation by Dilated Convolutions," In *International Conference on Learning Representations (ICLR)*, 2016. 90

[Yu *et al.*, 2015] Q. Yu, Y. Yang, Y.-Z. Song, T. Xiang, and T. Hospedales, "Sketch-a-Net that Beats Humans," In *British Machine Vision Conference (BMVC)*, 2015. 24

[Zagoruyko and Komodakis, 2016] S. Zagoruyko and N. Komodakis, "Wide Residual Networks," In *British Machine Vision Conference (BMVC)*, 2016. 14

[Zalando Research, 2018] "Shop the Look with Deep Learning," 2018. 1

[Zeiler and Fergus, 2014] M. D. Zeiler and R. Fergus, "Visualizing and Understanding Convolutional Networks," In *European Conference on Computer Vision (ECCV)*, 2014. 2, 23, 105

[Zhang *et al.*, 2019a] C. Zhang, F. Gao, B. Jia, Y. Zhu, and S.-C. Zhu, "Raven: A Dataset for Relational and Analogical Visual Reasoning," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 28, 29, 30, 31, 47

[Zhang *et al.*, 2019b] C. Zhang, B. Jia, F. Gao, Y. Zhu, H. Lu, and S.-C. Zhu, "Learning Perceptual Inference by Contrasting," In *Conference on Neural Information Processing Systems (NeurIPS)*, 2019. 28, 29, 31

[Zhang *et al.*, 2016] R. Zhang, P. Isola, and A. A. Efros, "Colorful Image Colorization," In *European Conference on Computer Vision (ECCV)*, 2016. 58, 84, 86, 88, 94, 95, 96, 124

[Zhang *et al.*, 2018a] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric," In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018. 106

[Zhang *et al.*, 2018b] Y. Zhang, J. Hare, and A. Prügel-Bennett, "Learning to count objects in natural images for visual question answering," In *International Conference on Learning Representations (ICLR)*, 2018.

[Zhao and Nevatia, 2001] T. Zhao and R. Nevatia, "Car Detection in Low Resolution Aerial Images," In *International Conference on Computer Vision (ICCV)*, 2001. 60

[Zheng *et al.*, 2018] L. Zheng, Y. Yang, and Q. Tian, "SIFT Meets CNN: A Decade Survey of Instance Retrieval," *IEEE Transactions on Pattern Analysis and Machine Intelligence (T-PAMI)*, 40(5):1224–1244, 2018. 2

[Zheng *et al.*, 2015] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr, "Conditional Random Fields as Recurrent Neural Networks," In *International Conference on Computer Vision (ICCV)*, 2015. 3

[Zhu *et al.*, 2017] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks," In *International Conference on Computer Vision (ICCV)*, 2017. 106, 121, 124, 126, 128, 133, 137

[Zhu *et al.*, 2018] P. Zhu, L. Wen, X. Bian, H. Ling, and Q. Hu, "Vision Meets Drones: A Challenge, An ECCV 2018 Workshop," 2018. 63

[Zhu *et al.*, 1998] S. C. Zhu, Y. Wu, and D. Mumford, "Filters, Random Fields and Maximum Entropy (FRAME): Towards a Unified Theory for Texture Modeling," *International Journal of Computer Vision (IJCV)*, 27(2):107–126, 1998.

[Zoran and Weiss, 2011] D. Zoran and Y. Weiss, "From Learning Models of Natural Image Patches to Whole Image Restoration," In *International Conference on Computer Vision (ICCV)*, 2011.

[Zou *et al.*, 2019] D. Zou, Y. Cao, D. Zhou, and Q. Gu, "Stochastic Gradient Descent Optimizes Over-parameterized Deep ReLU Networks," *Machine Learning*, 2019. 19