CZECH TECHINCAL
UNIVERSITY
IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Tackling Domain Generalization by Generating Additional Training Data

Jakub Brož

Supervisor: doc. Mgr. Branislav Bošanský, Ph.D.
Study program: Open Informatics
Specialisation: Artificial Intelligence
May 2023

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Brož Jakub** Personal ID number: **483529**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Computer Science**

Study program: **Open Informatics**

Specialisation: **Artificial Intelligence**

II. Master’s thesis details

Master’s thesis title in English:

Tackling Domain Generalization by Generating Additional Training Data

Master’s thesis title in Czech:

Generalizace modelů na nové domény pomocí generování nových trénovacích vzorků

Guidelines:

In the Domain Generalization (DG) problem, machine learning models are trained on a set of training domains, however, they are applied on a testing domain that significantly differs from the training domains. This is an active part of machine learning with many attempts to improve the performance of the models on testing domains. One research direction is to extend the training domains by using generative models to generate new training samples. The goal of the student is to:
(1) Summarize the state of the art in Domain Generalization (or Out-of-distribution detection and generalization) focusing on works that generate additional training samples.
(2) Analyze the existing benchmarks for algorithms tackling DG containing the most recent methods and choose some benchmarking domains for experiments.
(3) Identify possible weaknesses of existing generative approaches and propose a new method for generating new training data to improve domain generalization.
(4) Compare proposed method with the existing state-of-the-art techniques.

Bibliography / sources:

[1] Gulrajani, Ishaan, and David Lopez-Paz. "In search of lost domain generalization." arXiv preprint arXiv:2007.01434 (2020).
[2] Wang, Jindong, et al. "Generalizing to unseen domains: A survey on domain generalization." IEEE Transactions on Knowledge and Data Engineering (2022).
[3] Shen, Zheyan, et al. "Towards out-of-distribution generalization: A survey." arXiv preprint arXiv:2108.13624 (2021).

Name and workplace of master's thesis supervisor:

doc. Mgr. Branislav Bošanský, Ph.D. Artificial Intelligence Center FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **31.01.2023** Deadline for master's thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

doc. Mgr. Branislav Bošanský, Ph.D. Supervisor's signature	Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

I would like to thank my supervisor, doc. Mgr. Branislav Bošanský, Ph.D., for his guidance and kind approach both during my studies and the writing of this thesis. I would also like to thank my family for their support and encouragement throughout my studies.

The access to the computational infrastructure of the OP VVV funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" is also gratefully acknowledged.

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, May 25, 2023

.....

Abstract

Current deep learning models perform very well on a wide range of tasks and are capable of outperforming humans on many of them. This is especially true for image classification tasks, where deep neural networks have proven to be very effective. However, good results can only be achieved when the training and testing data come from the same distribution. In real-world applications, domain shifts are often present, and this assumption is violated. This leads to significant performance drops when evaluated on out-of-distribution (OOD) data. Domain generalization (DG) is a task formulation that addresses this issue by training models that generalize well under domain shifts. While most existing DG methods perform poorly on OOD data, techniques based on data augmentation and style transfer have shown promise. However, current methods use style transfer only as a random data augmentation technique. We try to improve upon them by proposing a novel method that uses style transfer in a more structured way in order to simulate new synthetic domains. We also study the performance of our proposed method in synergy with the existing DG method CORAL. Furthermore, we propose an additional novel way of using style transfer for DG, targeting the validation and model selection phase instead of the training phase.

Keywords:
domain generalization, domain shift, out-of-distribution generalization, robustness, machine learning, deep learning, data generation, generative models, style transfer, image stylization, image classification

Supervisor:
doc. Mgr. Branislav Bošanský, Ph.D.
Praha 2, Karlovo náměstí 13, E-407

Abstrakt

Současné modely hlubokého učení si vedou velmi dobře v široké škále úloh a v mnoha z nich jsou schopny překonat člověka. To platí zejména pro úlohy klasifikace obrázků, kde se hluboké neuronové sítě ukázaly jako velmi efektivní. Dobrých výsledků však lze dosáhnout pouze tehdy, pokud trénovací a testovací data pocházejí ze stejné distribuce. V reálných aplikacích se často vyskytují doménové posuny a tento předpoklad je porušen. To vede k výraznému poklesu výkonu při vyhodnocování na datech mimo distribuci (OOD). Doménové zobecnění (DG) je formulace úlohy, která tento problém řeší trénováním modelů, které dobře generalizují při doménových posunech. Zatímco většina stávajících metod DG má na OOD datech špatné výsledky, techniky založené na augmentaci dat a přenesení stylu se ukázaly jako slibné. Současné metody však používají přenesení stylu pouze jako techniku náhodné augmentace dat. Pokoušíme se je vylepšit tím, že navrhujeme novou metodu, která používá přenesení stylu strukturovanějším způsobem za účelem simulace nových syntetických domén. Zkoumáme také výkonnost námi navržené metody v součinnosti se stávající metodou DG CORAL. Dále navrhujeme další nový způsob využití přenesení stylu pro DG, který se zaměřuje na fázi validace a výběru modelu namísto trénovací fáze.

Klíčová slova:
doménové zobecnění, posun domény, zobecnění mimo distribuci, robustnost, strojové učení, hluboké učení, generování dat, generativní modely, přenesení stylu, stylizace obrázků, klasifikace obrázků

Překlad názvu:
Generalizace modelů na nové domény pomocí generování nových trénovacích vzorků

Contents

Figures

Tables

Chapter 1
Introduction

Classification is one of the most studied tasks in machine learning (ML). In recent years, deep learning has dominated the field due to the increase in computational power and the availability of large datasets. Neural networks have been widely used in numerous ML tasks, achieving excellent results, in many cases even outperforming humans [2]. In particular, image classification, which is nowadays tackled using complex, well-tuned convolutional neural network (CNN) architectures, can be considered an almost solved problem where human performance has long been surpassed [1].

However, such excellent results can only be achieved when the training and testing data come from the same distribution. The Empirical Risk Minimization (ERM) principle [37], which is the basis of most ML algorithms, assumes that the training and testing samples are drawn from the same unknown joint probability distribution P_{XY}. Data that follow this assumption, where the testing samples are drawn from the same distribution as the training samples, are called in-distribution data. Learning models that perform well on in-distribution data is often relatively easy, given enough training data.

In real-world applications, this assumption on the data distribution is often violated. In practice, it may be costly or even impossible to obtain samples from the exact same domain on which the classification model is to be deployed. In some cases, the distribution may also change after the deployment of the model, and we would like for the model to perform well even under such circumstances. Optimally, we would like to train the model on the data we have available at the time of training, and for the model to generalize to the out-of-distribution (OOD) data (i.e., data from a different distribution) that the model has not seen during training. This problem is called domain generalization (DG) or sometimes OOD generalization. Models that do not account for distribution shifts often suffer large performance drops when evaluated on OOD data. Note that the ability to generalize to OOD data is natural for humans and we want the same for our ML models.

In some applications, the training data may even be collected from multiple domains that are slightly different in some way, but are still related. Using data from only one of them might result in poor performance because the number of training samples might be small. By combining them, we can get more samples, and we can also use the information about which domain each training sample comes from. This relationship between samples and domains can be exploited by the model to better generalize to unseen domains.

The problem of DG has been studied for some time now and many methods have been proposed specifically to solve it. However, the performance of these methods on OOD data is not satisfactory and they are often outperformed by simple baselines [12, 4]. Methods that seem to perform well on OOD data are mostly based on data augmentation and generating additional training data. In particular, methods based on style transfer have been shown to be very effective in this task [18, 34, 4]. Style transfer is a technique that can be used to transfer the style of one image to another.

It can be used in the DG setting to simulate previously unseen domains by changing the style of existing training samples.

In this thesis, we focus on the problem of DG and specifically on the methods that try to solve it by generating additional training data through style transfer. The existing methods use style transfer merely as a random augmentation technique to generate more diverse training data for the model [18, 34, 4]. We try to improve upon them by proposing a new method that tries to use style transfer for DG in a more structured way to simulate new synthetic domains. We also study the performance of our proposed method in synergy with the existing DG method CORAL [35]. Finally, we also propose a novel way of utilizing style transfer for DG, which targets the validation and model selection phase instead of the training phase.

1.1 Outline

The rest of this thesis is organized as follows. We start by introducing the problem of DG and related concepts in Chapter 2. In the same chapter, we also present several datasets that are often used to evaluate DG methods. Next, we introduce the concept of style transfer and some methods that achieve style transfer using deep neural networks in Chapter 3. In Chapter 4, we examine several existing methods that attempt to solve the problem of DG, with an emphasis on methods that use style transfer to generate additional training data. In the following Chapter 5, we discuss the importance of proper evaluation methodology in the DG setting and look at the performance of several state-of-the-art methods on DG benchmarks. Chapter 6 is dedicated to our novel proposed methods that try to utilize style transfer for DG in a different way than the existing methods. Finally, in Chapter 7, we evaluate the performance of our proposed methods and compare them with the existing ones.

Chapter 2
Domain generalization

In this chapter, we formally define the problem of DG and related concepts, discuss possible variants of DG, and compare it with related problems. We also present several datasets that are commonly used to evaluate the performance of DG methods.

2.1 Formal definition

To define the DG problem, we first need to understand the concept of domain and introduce some notation.

Definition 2.1 (Domain [40])**.** Let $\mathcal{X}$ denote a non-empty input space (features) and $\mathcal{Y}$ an output space (labels). A **domain** is composed of data that are sampled from a joint distribution of features and labels. We denote it as $S = \{(x_i, y_i)\}_{i=1}^{n} \sim P_{XY}$, where $x \in \mathcal{X} \subset \mathbb{R}^d, y \in \mathcal{Y} \subset \mathbb{R}$, $n \in \mathbb{N}$ is the number of samples and P_{XY} denotes the joint distribution of the features and labels. X and Y denote the corresponding random variables.

The situation where two domains S and S' are sampled from different distributions (i.e., $P_{XY} \neq P'_{XY}$) is called domain shift. Domain shift is the main challenge of DG. Given multiple domains, where domain shift occurs between each pair of domains, we can define the DG problem as follows:

Definition 2.2 (Domain generalization [40])**.** In **domain generalization**, we are given a finite set of M training (source) domains $S_{\text{train}} = \left\{ S^i \,\middle|\, i = 1, \ldots, M \right\}$, where $S^i = \{(x_j^i, y_j^i)\}_{j=1}^{n_i}$ denotes the i-th domain. The joint distributions between each pair of domains are different: $P_{XY}^i \neq P_{XY}^j$, $1 \leq i \neq j \leq M$. The goal of domain generalization is to learn a robust and generalizable predictive function $h : \mathcal{X} \to \mathcal{Y}$ from the M training domains to achieve a minimum prediction error on an unseen test (target) domain $S_{\text{test}} \sim P_{XY}^{\text{test}}$ (i.e., S_{test} cannot be accessed in training and $P_{XY}^{\text{test}} \neq P_{XY}^i$ for $i \in \{1, \ldots, M\}$):

$$\min_h \mathbb{E}_{(x,y)\in S_{\text{test}}} \left[l(h(x), y) \right], \tag{2.1}$$

where $\mathbb{E}$ is the expectation and $l(\cdot, \cdot)$ is the loss function.

In this work, we are mainly concerned with the classification task. Therefore, we assume a cross-entropy loss function in place of l. However, the concept of DG extends beyond classification and can be applied to any supervised learning task.

The objective of DG, as shown in (2.1), reveals why DG is such a challenging problem. We are interested in minimizing the expected loss on the test domain S_{test}, which is not available to us at the time of model training. The only information we have available about the testing domain S_{test} is that it is somehow related to our training domains S_{train}, but it is not from the same distribution (the definition is intentionally vague).

One way to define the relationship between domains more concretely and formally is to introduce the notion of hyperdistribution [40]. The same underlying hyperdistri-

bution $\mathcal{P}$ on (x, y) distributions is followed by both the test domain and all training domains: $P_{XY}^{\text{test}} \sim \mathcal{P}$ and $P_{XY}^{i} \sim \mathcal{P}$ for $i \in \{1, \ldots, M\}$.

Since minimizing the expectation in (2.1) is impossible without access to the target domain S_{test}, we instead want to minimize the expected risk over all possible target domains that follow the hyperdistribution $\mathcal{P}$:

$$\mathcal{E}(h) \coloneqq \mathbb{E}_{P_{XY} \sim \mathcal{P}} \mathbb{E}_{(x,y) \sim P_{XY}} [l(h(x), y)] \,. \tag{2.2}$$

Evaluating this expected risk $\mathcal{E}(h)$ is still unfeasible, but it can be estimated using a finite set of source domains S_{train}:

$$\hat{\mathcal{E}}(h) \coloneqq \frac{1}{M} \sum_{i=1}^{M} \frac{1}{n^i} \sum_{j=1}^{n^i} l(h(x_j^i), y_j^i) \,. \tag{2.3}$$

However, because the source and target domains are not sampled from the same distribution, the expected risk $\hat{\mathcal{E}}(h)$ can be very far from the true risk $\mathcal{E}(h)$. Consequently, a more sophisticated approach than simply minimizing $\hat{\mathcal{E}}(h)$ is needed to solve DG problems.

2.2 Variations

Single-source vs. multi-source DG. We can further distinguish between single-source and multi-source DG. In the single-source DG, only one training domain is available (i.e., $M = 1$). In the multi-source DG, there is more than one training domain available (i.e., $M \geq 2$). As we will see in Chapter 4, some methods dealing with DG require more than one source domain to be applicable (i.e., they require domain labels). Others do not distinguish between source domains and are therefore applicable to both single-source and multi-source DG. In the case of multi-source DG, they simply ignore the domain labels and treat all source domains as a single source.

Single-target vs. multi-target DG. Another distinction can be made based on the number of target domains. If we have only one target domain, as defined in Definition 2.2, we speak of a single-target DG. However, sometimes we may have more than one target domain and be interested in measuring performance on each of them individually. This is known as multi-target DG. However, since the target domains are not available during training, we cannot optimize the model specifically for each target domain. Instead, a single model is trained to generalize well on all target domains, and then its performance is measured on each of them separately.

2.3 Related problems

Supervised learning. Similarly to DG, the objective of supervised learning is to learn a predictive function $h : \mathcal{X} \to \mathcal{Y}$ from training data S_{train} to achieve a minimum prediction error on testing data S_{test} (same as (2.1)). The difference is that in supervised learning, we assume that the training and testing data are sampled from the same distribution P_{XY} (i.e., no domain shift occurs). Because of this, the estimation of the expected risk (2.3) is much more reliable. This makes the problem of supervised learning solvable by standard ERM [37]. On the other hand, DG is more useful in real-world applications, where the assumption of sampling from the same distribution usually does not hold.

Domain adaptation. Domain adaptation (DA) is a problem that is closely related to DG. Both deal with the problem of domain shift, and their objective is the same (i.e., to minimize the prediction error on the target domain S_{test} (2.1)). The difference is that in DA, we have access to unlabeled (or sparsely labeled) data from the target domain S_{test} during training. In other words, the marginal distribution P_X^{test} of the target domain is known. This information can be used during training to adapt the model to the target domain S_{test}. Note that all DG methods can be used for DA problems by simply not using the unlabeled target data during training. However, this puts them at a disadvantage compared to DA methods, which can leverage the unlabeled target data. The opposite is not true. DA-specific methods usually cannot be used for DG problems without modification.

2.4 Datasets

DG has been studied for several years, and several datasets have been compiled for evaluation purposes and to ease the comparison of different approaches to DG. In this chapter, we present some of the most widely used datasets that can be used to study the DG problem. All of these datasets consist of multiple domains and are therefore suitable for multi-source DG.

These datasets do not distinguish between source and target domains. Typically, methods are evaluated in a leave-one-out fashion, where one domain is used as the target domain and the rest of the domains are used as source domains. In this way, a model is trained for each domain individually in a multi-source, single-target setting. We can then measure the performance of the given method on each domain separately or average the performance across all domains.

2.4.1 Digits

The MNIST dataset [22] of handwritten digits has become the de facto standard dataset for debugging and demonstration purposes in the field of image classification. Naturally, several variants of MNIST have been created for the same purposes in DG. Most notably, these include Colored MNIST [3] (C-MNIST), Rotated MNIST [9] (R-MNIST), and MNIST-M [5], all three of which consist of modified digits from the standard MNIST dataset. C-MNIST colors each digit either red or green with strong spurious correlation with the class label (correlation is reversed in the testing domain), R-MNIST rotates digits by different angles in each domain, and MNIST-M blends randomly cropped photographs into the background and their inverse into the foreground of the images.

Figure 2.1: Examples from MNIST, MNIST-M, SVHN and SYN datasets. Image source: [43]

Figure 2.2: Examples of dog class from all four domains of the PACS dataset. Image source: [43]

Another widely used dataset with digits is the Street View House Numbers (SVHN) dataset [27]. It is mainly used in object detection tasks, but its cropped variant can also be used for image classification. Ganin et al. [5] have introduced a synthetic variant of SVHN called SYN, which was "*generated from Windows fonts by varying the text, positioning, orientation, background and stroke colors, and amount of blur.*"

Figure 2.1 shows examples from MNIST, MNIST-M, SVHN, and SYN datasets that can be bundled together to form a multi-domain dataset for DG problems.

2.4.2 PACS

PACS [24] is a dataset that has been put together from other smaller datasets specifically for the purposes of DG. It consists of 4 domains, 7 classes and 9991 images. The available domains are Art, Cartoon, Photo and Sketch. PACS classes include dog, elephant, giraffe, guitar, horse, house and person. Despite its smaller size, it is one of the most widely used datasets for DG. Figure 2.2 shows examples of the dog class from all four domains of the PACS dataset.

2.4.3 OfficeHome

OfficeHome [38] is similar to the PACS dataset and was also created for DG purposes. It contains images of objects from office and home environments. It consists of 4 domains, 65 classes, and 15,500 images. The domains used are Art, Clipart, Product, and Real World. However, the OfficeHome dataset is not as well-curated as the PACS dataset. Images from the Art, Product, and Real World domains are often very similar, so the assumption of different distributions does not hold so well. Despite these shortcomings, OfficeHome is still widely used for DG evaluation. Figure 2.3 shows examples of different classes from all four domains of the OfficeHome dataset.

Figure 2.3: Examples of several classes from all four domains of the OfficeHome dataset. Image source [38]

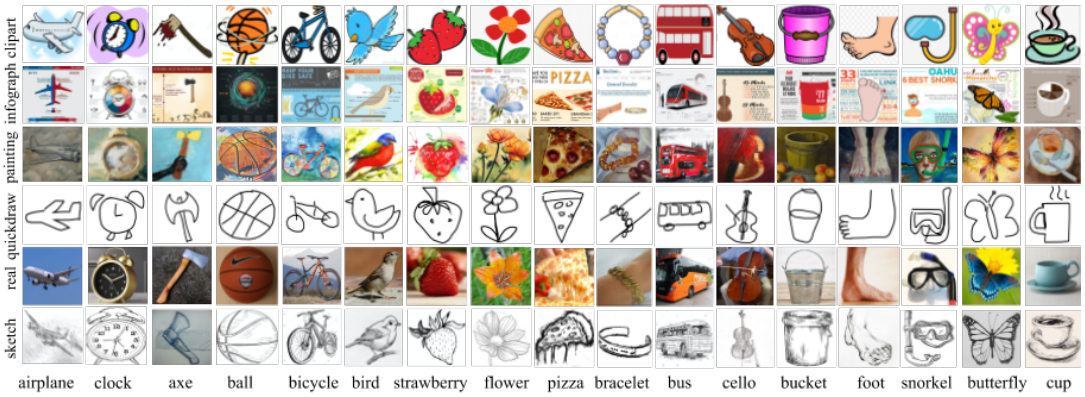

Figure 2.4: Examples of several classes from all six domains of the DomainNet dataset. Image source: [30]

2.4.4 DomainNet

DomainNet [30] is an ImageNet-like dataset [32]. It was originally created for the purposes of DA. However, since both DA and DG deal with generalization under domain shift, it is also suitable for DG. It consists of 6 domains, 345 classes, and about 0.6 million images. In contrast to the previous datasets, DomainNet is a large-scale dataset with substantially more classes and is therefore more challenging. The domains used are clipart, infograph, painting, quickdraw, real, sketch. Figure 2.4 shows examples for a subset of classes and all six domains of the DomainNet dataset.

2.4.5 WILDS

WILDS [21] is "*a curated collection of benchmark datasets that represent distribution shifts faced in the wild*". It has been collected for the purpose of studying DG and subpopulation shifts. The WILDS collection emphasizes that the datasets are from real-world applications. It is very diverse as it contains 10 datasets of different sizes, input and output types, application areas, domains, classes, etc. In addition to image datasets for image classification, it also contains datasets with natural language, code or molecular graphs. Details of the WILDS datasets can be seen in Figure 2.5.

	Domain generalization					Subpopulation shift	Domain generalization + subpopulation shift			
Dataset	iWildCam	Camelyon17	RxRx1	OGB-MolPCBA	GlobalWheat	CivilComments	FMoW	PovertyMap	Amazon	Py150
Input (x)	camera trap photo	tissue slide	cell image	molecular graph	wheat image	online comment	satellite image	satellite image	product review	code
Prediction (y)	animal species	tumor	perturbed gene	bioassays	wheat head bbox	toxicity	land use	asset wealth	sentiment	autocomplete
Domain (d)	camera	hospital	batch	scaffold	location, time	demographic	time, region	country, rural-urban	user	git repository
# domains	323	5	51	120,084	47	16	16 x 5	23 x 2	2,586	8,421
# examples	203,029	455,954	125,510	437,929	6,515	448,000	523,846	19,669	539,502	150,000
Train example						What do Black and LGBT people have to do with bicycle licensing?			Overall a solid package that has a good quality of construction for the price.	`import numpy as np ... norm=np.___`
Test example						As a Christian, I will not be patronizing any of those businesses.			I *loved* my French press, it's so perfect and came with all this fun stuff!	`import subprocess as sp p=sp.Popen() stdout=p.___`
Adapted from	Beery et al. 2020	Bandi et al. 2018	Taylor et al. 2019	Hu et al. 2020	David et al. 2021	Borkan et al. 2019	Christie et al. 2018	Yeh et al. 2020	Ni et al. 2019	Raychev et al. 2016

Figure 2.5: Details of the WILDS dataset with example images. Image source: [21]

Chapter 3
Style transfer

In this chapter, we introduce style transfer as a general technique for image manipulation, along with some approaches that achieve style transfer using deep neural networks.

Style transfer is a technique that allows transferring the style of one image to another. It usually takes two images as input, a content image and a style image, and produces a stylized image as output. Its goal is to transfer the style from the style image to the content image while preserving the content. In this setting, style is understood as the texture, color, and local visual patterns of the image, while content is understood as the actual objects and their arrangement in the image.

The emergence of style transfer was motivated mainly by the desire to automatically create artistic images and to allow users to easily generate their own stylized content on demand. However, style transfer has many other applications. For us, the most important application is the ability to generate additional synthetic training data for ML models. This way, style transfer can be thought of as a data augmentation or generation technique that can be used to improve the generalization of ML models.

3.1 Neural style transfer

Style transfer has been studied for some time before the advent of deep learning. However, it was not until the introduction of deep neural networks that it became possible to achieve high-quality style transfer in real time. Style transfer that utilizes deep neural networks is often referred to as neural style transfer. Gatys et al. [6] were the first to introduce neural style transfer. Since then, many different approaches have been developed to improve the quality and speed of neural style transfer. Jing et al. [19] provide a comprehensive overview of the existing methods.

The general idea of neural style transfer is to use a neural network, usually pre-trained on an image classification task, to extract a representation of content and style from input images. Then, a stylized output image is found by an optimization process that simultaneously minimizes the distances between the extracted content and style representations and the corresponding representations of the output image.

3.1.1 Extracting content and style representations

By training on an image classification task, CNNs develop a hierarchy of feature maps (i.e., the output of intermediate layers) that capture information about the input image at different levels of abstraction [6]. The lower layers capture the low-level information, such as edges and textures, that are more related to the style of the image. On the other hand, the higher layers capture high-level information, such as objects and their arrangement, which is more related to the content of the image. The key contribution of Gatys et al. [6] is the finding that the representations of both content and style can be extracted from the feature maps and that they are separable to some extent.

Obtaining the content representation from the content image I_c is straightforward. Let $\mathcal{F}^l(I_c) \in \mathbb{R}^{C \times H \times W}$ be the feature maps of the content image I_c at some intermediate layer l of the pre-trained network, where C is the number of channels and H and W are the height and width of the feature maps at layer l, respectively. Then the content representation at layer l is simply $\mathcal{F}^l(I_c)$.

Getting the style representation from the style image I_s is more complicated. Gatys et al. [6] propose using the so-called Gram-based representation. They use a feature space built on top of the feature maps of the neural network designed to capture texture information. This feature space consists of correlations between individual filters (channels) of the feature maps. The Gram-based representation is obtained by computing the Gram matrix $\mathcal{G}(\mathcal{F}^l(I_s)') \in \mathbb{R}^{C \times C}$ over the reshaped feature maps $\mathcal{F}^l(I_s)' \in \mathbb{R}^{C \times (HW)}$:

$$\mathcal{G}(\mathcal{F}^l(I_s)') = \left[\mathcal{F}^l(I_s)'\right] \left[\mathcal{F}^l(I_s)'\right]^\top \tag{3.1}$$

or alternatively

$$\mathcal{G}(\mathcal{F}^l(I_s)')_{ij} = \sum_{k=1}^{HW} \mathcal{F}^l(I_s)'_{ik} \mathcal{F}^l(I_s)'_{jk} \,. \tag{3.2}$$

The Gram matrix $\mathcal{G}(\mathcal{F}^l(I_s)')$ is then used to represent the style at layer l. When computing the Gram matrix, the sums are taken over the spatial dimensions of the feature maps, and thus the spatial information is lost. This is desirable because the style representation should capture only the texture information and not the spatial information. Note that the choice of the Gram-based representation is not the only possible choice for the style feature space. We merely use it to extract global feature distributions (in this case, second-order statistics) from feature maps with spatial information removed.

3.1.2 Image reconstruction

With the representations of content and style ready, the stylized output image I_o is found by an optimization process that tries to minimize the following loss function:

$$\mathcal{L}(I_c, I_s, I_o) = \alpha \mathcal{L}_{content}(I_c, I_o) + \beta \mathcal{L}_{style}(I_s, I_o) \,, \tag{3.3}$$

where α and β are weights that control the importance of content and style loss, respectively [6].

The content loss $\mathcal{L}_{content}$ is defined as the squared error between the content representations (feature maps) extracted from the content image I_c and the output image I_o at layer l:

$$\mathcal{L}_{content}(I_c, I_o) = \frac{1}{2} \sum_{i=1}^{C} \sum_{j=1}^{HW} \left[\mathcal{F}^l(I_c)'_{ij} - \mathcal{F}^l(I_o)'_{ij}\right]^2 \,. \tag{3.4}$$

To improve the quality of the stylized output image, Gatys et al. [6] suggest using multiple layers to compute the style loss. The style loss $\mathcal{L}^l_{style}$ at a specific layer l is defined as the squared error between the style representations (Gram matrices) extracted from the style image I_s and the output image I_o at layer l:

$$\mathcal{L}^l_{style}(I_s, I_o) = \frac{1}{4C^2(HW)^2} \sum_{i=1}^{C} \sum_{j=1}^{C} \left[\mathcal{G}(\mathcal{F}^l(I_s)')_{ij} - \mathcal{G}(\mathcal{F}^l(I_o)')_{ij} \right]^2 . \tag{3.5}$$

The loss $\mathcal{L}^l_{style}$ is further normalized by the factor $\frac{1}{4C^2(HW)^2}$ to make it independent of the size of the feature maps of layer l. The total style loss $\mathcal{L}_{style}$ is then defined as the weighted sum of the style losses on individual layers:

$$\mathcal{L}_{style}(I_s, I_o) = \sum_{l \in L} w_l \mathcal{L}^l_{style}(I_s, I_o) \,, \tag{3.6}$$

where L is a subset of the network layers used to compute the style loss and w_l are the corresponding weights that control the importance of the style loss at each layer.

The gradients of the loss function (3.3) with respect to the feature maps $\mathcal{F}^l(I_o)$ of the output image I_o can be found analytically. From these gradients, gradients with respect to the individual pixel values of the output image I_o can be computed using backpropagation. The optimization process starts with a white noise image and iteratively updates its pixel values using a standard gradient descent algorithm until convergence.

3.2 Improving neural style transfer

The image reconstruction process described in Section 3.1.2 is very time-consuming because it uses gradient descent to iteratively update the individual pixel values of the output image. To speed up the process, one can use a decoder network to generate the output image directly from the content and style representations.

Such an approach has been proposed by Huang et al. [14]. It follows the general encoder-decoder architecture used in several other deep learning applications. In addition, Huang et al. [14] propose to use a novel normalization layer called adaptive instance normalization (AdaIN). The AdaIN layer replaces the Gram-based representation used by Gatys et al. [6] and combines the content and style representations into a single input for the decoder network. Examples of stylized images produced by this approach are shown in Figure 3.2.

3.2.1 AdaIN layer

The AdaIN layer is a simple extension of the instance normalization (IN) layer [36]. Given an input batch $x \in \mathbb{R}^{N \times C \times H \times W}$, the IN layer is defined as

$$\mathrm{IN}(x) = \gamma \left(\frac{x - \mu(x)}{\sigma(x)} \right) + \beta \,, \tag{3.7}$$

where $\gamma, \beta \in \mathbb{R}^{N \times C}$ are affine parameters learned from the data and N is the batch size. IN differs from the widely used batch normalization (BN) layer [17] in that $\mu(x)$ and $\sigma(x)$ are computed over spatial dimensions independently for each channel and sample:

$$\mu_{nc}(x) = \frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W} x_{nchw}\,, \tag{3.8}$$

$$\sigma_{nc}(x) = \sqrt{\frac{1}{HW}\sum_{h=1}^{H}\sum_{w=1}^{W}\left(x_{nchw} - \mu_{nc}(x)\right)^2 + \epsilon}\,, \tag{3.9}$$

where ϵ is a small constant added for numerical stability.

Assuming x is a content input batch and y is a style input batch, the AdaIN layer is defined as

$$\mathrm{AdaIN}(x, y) = \sigma(y)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y)\,. \tag{3.10}$$

AdaIN replaces the learnable affine parameters γ and β in (3.7) with the statistics $\sigma(y)$ and $\mu(y)$ of the style input y. In other words, the AdaIN layer aligns the mean and variance of the content input x to match the mean and variance of the style input y. Note that, similar to IN, the statistics $\mu(y)$ and $\sigma(y)$ are computed over spatial dimensions independently for each channel and sample (see (3.8) and (3.9)). This results in preserving the spatial information of the content input x, while changing its feature distribution to match the distribution of the style input y.

3.2.2 Architecture

The architecture proposed by Huang et al. [14] is shown in Figure 3.1. It consists of an encoder network f, a decoder network g, and a single AdaIN layer in between. The decoder mostly mirrors the encoder with some minor changes. All pooling layers are replaced by nearest-neighbor upsampling layers, and all normalization layers are removed. The encoder uses weights pre-trained on an image classification task and is frozen during both training and inference, while the decoder is trained from scratch using the loss function defined in (3.13).

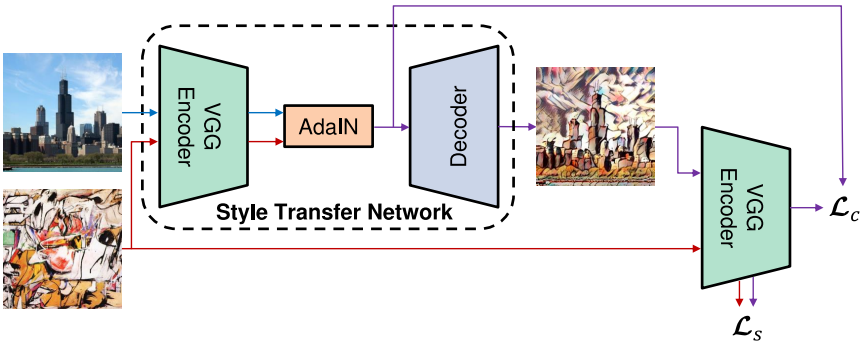

Figure 3.1: Encoder-decoder architecture of the style transfer network with AdaIN layer. Image source: [14].

Both the content image I_c and the style image I_s are fed into the encoder network f to extract their feature maps $f(I_c)$ and $f(I_s)$. These feature maps are then combined using the AdaIN layer as described in (3.10) to produce the target feature maps t:

$$t = \mathrm{AdaIN}\big(f(I_c), f(I_s)\big)\,. \tag{3.11}$$

The stylized output image I_o is then generated by the decoder network g from the target feature maps t:

$$I_o = g(t)\,. \tag{3.12}$$

3.2.3 Loss function

Similarly to (3.3), the loss function used by the AdaIN style transfer method consists of content loss $\mathcal{L}_{content}$ and style loss $\mathcal{L}_{style}$:

$$\mathcal{L}(I_c, I_s) = \mathcal{L}_{content}(I_c, I_s) + \lambda \mathcal{L}_{style}(I_c, I_s)\,, \tag{3.13}$$

where λ is the style loss weight, which controls the strength of stylization.

As ilustrated in Figure 3.1, the content loss $\mathcal{L}_{content}$ is computed as the Euclidean distance between the target feature maps t and the feature maps of the output image:

$$\mathcal{L}_{content}(I_c, I_s) = \|f(g(t)) - t\|_2\,. \tag{3.14}$$

This ensures that the content of the content image I_c encoded in the target feature maps t is preserved in the output image I_o.

The style loss $\mathcal{L}_{style}$ is defined as the Euclidean distance between the IN statistics of the input style image I_s and the output image I_o at selected layers of the encoder network f:

$$\mathcal{L}_{style}(I_c, I_s) = \sum_{l \in L} \left[\left\| \mu(\mathcal{F}^l(g(t))) - \mu(\mathcal{F}^l(I_s)) \right\|_2 + \left\| \sigma(\mathcal{F}^l(g(t))) - \sigma(\mathcal{F}^l(I_s)) \right\|_2 \right], \tag{3.15}$$

where L is a subset of the network layers used to compute the style loss. This ensures that the IN statistics that encode the style of the style image I_s are matched by the output image I_o.

In addition to the ability to control the content-style tradeoff by adjusting the style loss weight λ during training, Huang et al. [14] also propose a method to control the strength of the style transfer at inference time. This is achieved by introducing a style interpolation parameter α, which controls the amount of style transfer performed by the AdaIN layer:

$$I_o = g\Big((1-\alpha) f(I_c) + \alpha \mathrm{AdaIN}\big(f(I_c), f(I_s)\big)\Big)\,. \tag{3.16}$$

If $\alpha = 0$, the AdaIN layer is not applied at all and $t = f(I_c)$. On the other hand, if $\alpha = 1$, the AdaIN layer is fully applied as described in (3.11). Setting α to a value between 0 and 1 allows for a smooth transition in stylization strength.

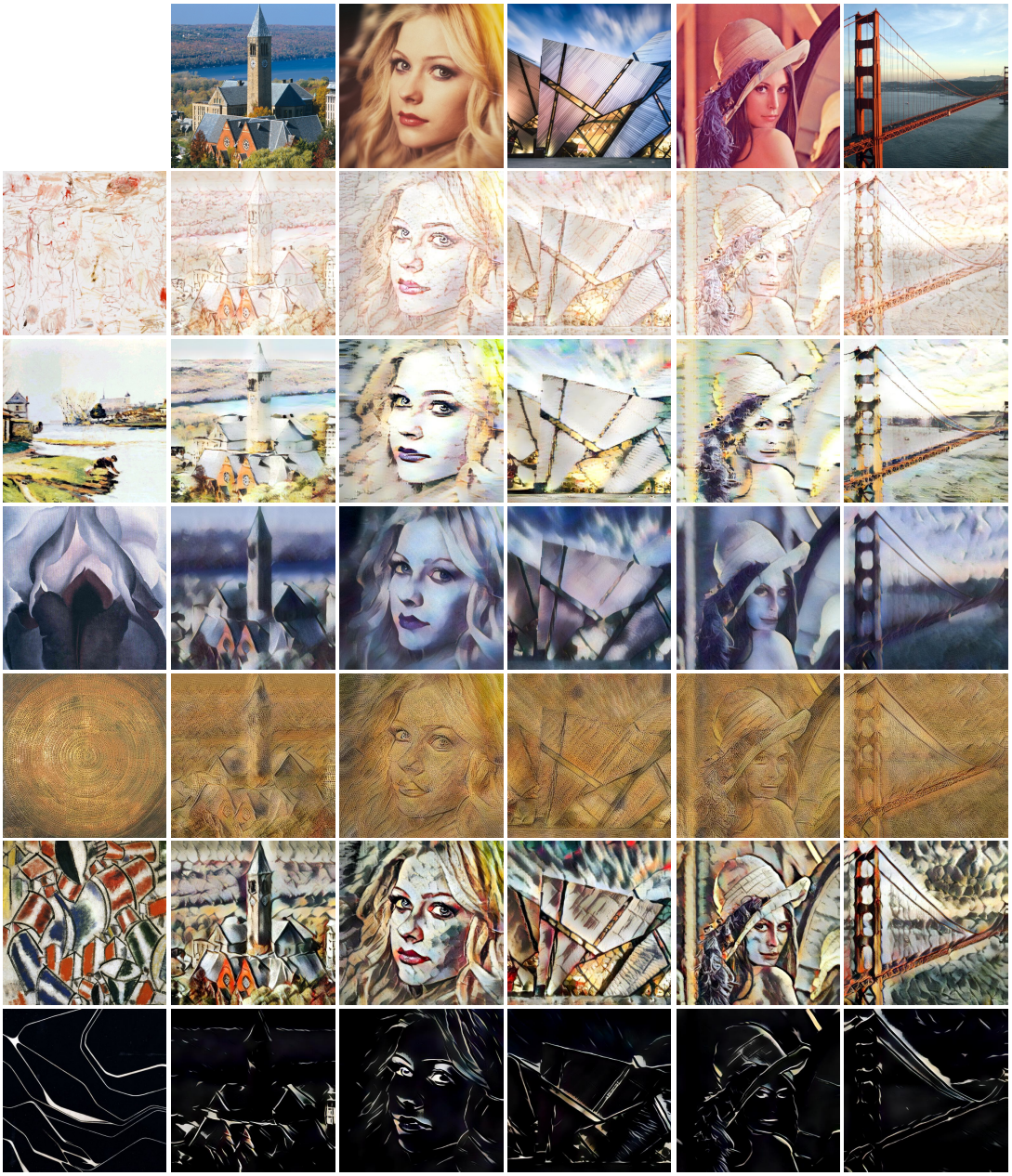

Figure 3.2: Examples produced by the encoder-decoder style transfer network with AdaIN layer. Columns correspond to content images, and each row has been stylized with a different style image. Image source: [14].

Chapter 4
Domain generalization methods

As mentioned before, DG is an area that has been studied for more than a decade now. A plethora of different methods and approaches have been developed to improve the performance of ML models under domain shift. Wang et al. [40] and Zhou et al. [43] have tried to provide a comprehensive overview of the existing approaches. In this chapter, we look at some of them in more detail, with an emphasis on methods that utilize style transfer to improve generalization under domain shift through data generation.

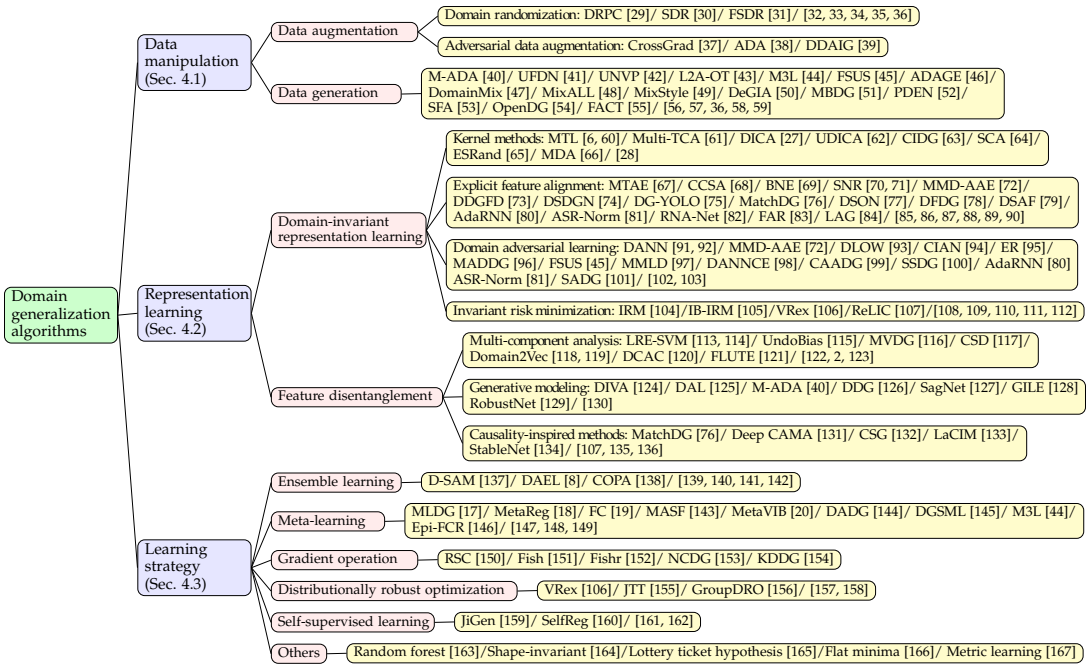

Figure 4.1: Taxonomy of domain generalization methods. Image source: [40]
(Note that the references correspond to the references in the original source)

Figure 4.1 illustrates the overwhelming amount of methods that have been developed to tackle DG, as well as the taxonomy of DG methods [40]. The methods can be categorized into three general groups based on how they approach the domain shift problem: data manipulation, representation learning, and learning strategy. We will look at each of these groups in more detail in the following sections.

4.1 Data manipulation

Methods in this category focus on manipulating the input training data itself, rather than changing the training procedure or model architecture. They do this by either augmenting the existing data or generating additional data from it. The performance

of ML models is highly dependent on the quantity and quality of available training data. Data manipulation techniques are useful even in the absence of domain shift, as they can help improve the robustness and generalization of ML models and prevent overfitting. Having more and more diverse samples is almost always beneficial for model performance.

We can reformulate the general objective of DG, as shown in (2.1), for data manipulation methods as:

$$\min_h \Big(\mathbb{E}_{(x,y)} \left[l(h(x), y) \right] + \mathbb{E}_{(x',y)} \left[l(h(x'), y) \right] \Big) , \tag{4.1}$$

where $x' = \mathcal{M}(x)$ denotes the manipulated data using a manipulation function $\mathcal{M}(\cdot)$ [40]. Thus, we train the model not only on the original input x, but also on its manipulated version x'. Based on how the function $\mathcal{M}$ is defined, we can further divide these methods into two subcategories: data augmentation and data generation.

4.1.1 Data augmentation

Simple data augmentation techniques are widely used, especially in computer vision, to improve the robustness of ML models and reduce their overfitting. Random computer vision augmentation techniques, such as cropping, flipping, rotating, scaling, adding noise, or changing color, can be easily incorporated into DG training in place of manipulating function $\mathcal{M}$ and can have a positive impact on generalization. Because of their ease of implementation, they are a popular choice as generalization techniques. Despite being random and not exploiting domain information, their impact on generalization under domain shift has been shown to be positive [12].

Several methods have been proposed that attempt to systematically augment the input data using adversarial training that exploits domain information. Shankar et al. [33] has proposed CrossGrad, an adversarial augmentation technique that perturbs the input data in the direction of the largest domain change. To this end, in addition to the label classifier h, it also trains a domain classifier G, which outputs the domain label. Both classifiers, h and G, are trained simultaneously in an adversarial manner. Using the gradient of G, the input for h is perturbed along the direction of the largest domain change, while enforcing that its label does not change. This way, CrossGrad tries to simulate samples from unseen domains.

Another DG method that also uses adversarial training has been proposed by Volpi et al. [39]. Unlike the previous method, it does not use the domain labels and can therefore be considered a single-source DG method. It is an iterative procedure in which adversarial examples are appended to the training data in each iteration, alternating with the training of the classifier h. The adversarial examples are created by perturbing the actual input data so that they come from a fictive target domain that is "hard" under the current model (i.e., they come from the worst-case fictive target distribution). This is achieved by the adversarial perturbation being based on the gradient of the loss function l with respect to the input data.

4.1.2 Data generation

Data generation is another popular data manipulation method that attempts to generate new diverse data in order to generalize to unseen domains. Here, the data manipulation function $\mathcal{M}(\cdot)$ is often implemented using a deep generative model such as a variational autoencoder (VAE) [20] or a generative adversarial network (GAN)

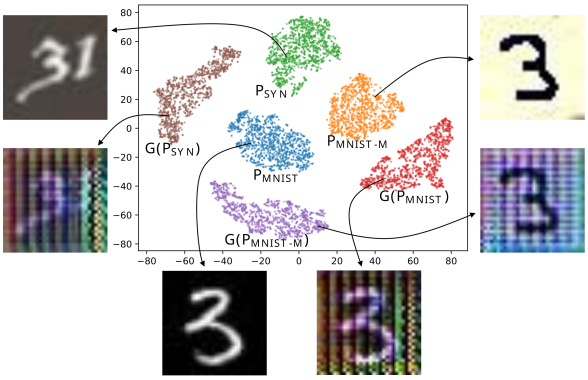

Figure 4.2: Motivational ilustration of L2A-OT effect on digit datasets. Image source: [44]

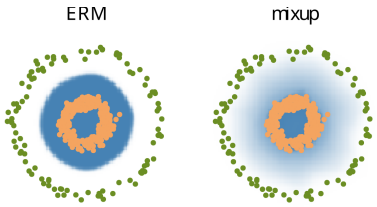

Figure 4.3: Effect of mixup ($\alpha = 1$) on a toy problem. Green: Class 0. Orange: Class 1. Blue shading indicates $p(y = 1|x)$. Image source: [42]

[10]. However, new examples can also be generated from existing samples in an easier way. Methods based on style transfer also fall into this category, as they generate new samples from existing ones by changing their style. However, since this is the main focus of this thesis, we will discuss them separately in Section 4.1.3.

Zhou et al. [44] have proposed a data generation method called L2A-OT. In this method, new samples are generated in each iteration from pseudo-novel domains using a generator network. The generator takes as input the original image and a novel domain label. Novel domains are synthesized from existing domains by maximizing the distribution divergence between the original and the novel domain. L2A-OT uses Wasserstein distance as the divergence measure, with the cost function defined as the cosine distance between images mapped into the latent space. The latent space is learned using an additional fixed network called critic, which is pre-trained using domain classification loss. To ensure semantic consistency of novel domains, cycle consistency and classification losses are imposed on the generator. This ensures that the newly generated diverse samples retain their semantic meaning as well as their label. An illustration of the effect of L2A-OT on digit domains can be seen in Figure 4.2.

An example of a data generation method that does not use deep generative models is mixup [42]. It uses a fairly simple technique that generates new data by linearly interpolating between existing samples and their corresponding labels. The linear interpolation is done on the raw inputs, not in latent space. The weights for the linear combination are sampled from a Beta distribution. As a result, a model trained with this procedure behaves linearly between training examples, improving generalization capabilities. The effect of mixup method on the toy problem can be seen in Figure 4.3. We can see the linear transition of the decision boundary between classes, contrary to the standard ERM.

4.1.3 Style transfer

Biased towards texture. Geirhos et al. [7] were the first to propose the use of style transfer as a form of data generation technique in ML. While they did not directly target DG, their work served as a source of inspiration for later research in this area. Specifically, Geirhos et al. [7] used the AdaIN style transfer method [14] and the Painter by Numbers dataset [28] as a style source to create a new dataset, Stylized-ImageNet (SIN). SIN was created by applying a random style to each image from the original ImageNet (IN) dataset [32].

They then compared the behavior of CNNs trained on SIN with the behavior of

(a) Texture image
81.4% **Indian elephant**
10.3% indri
8.2% black swan

(b) Content image
71.1% **tabby cat**
17.3% grey fox
3.3% Siamese cat

(c) Texture-shape cue conflict
63.9% **Indian elephant**
26.4% indri
9.6% black swan

Figure 4.4: Example from a cue conflict experiment and the corresponding predictions of a CNN trained on the standard ImageNet dataset. Image source: [7]

CNNs trained on the original IN dataset. Using a texture-shape cue conflict experiment, which measures the ratio of texture to shape decisions made, they showed that CNNs trained on the IN dataset are heavily biased towards texture because they rely almost exclusively on texture information to classify images. Humans, on the other hand, rely primarily on shape information to classify images. However, they also found that this texture bias was not inherent to the CNN architecture itself, but rather a feature of the IN dataset, which is how most CNNs are pre-trained nowadays. By training CNNs exclusively on SIN, Geirhos et al. [7] were able to significantly reduce the texture bias in these models.

An example from the cue conflict experiment and the corresponding predictions of a CNN trained on IN can be seen in Figure 4.4. We can see that the model assigned the cue conflict image based on texture, indicating that it is biased towards texture. On the other hand, both humans and CNNs trained on SIN would be more likely to assign the cue conflict image based on its shape.

Inspired by these findings, they also experimented with using SIN as a source of data augmentation. By training jointly on both IN and SIN datasets, they were able to improve classification performance on the original IN dataset, as well as on transfer learning tasks such as object detection. Furthermore, this approach improved the robustness of CNNs to a variety of distortions and corruptions (such as noise, blur, weather effects, compression, pixelation, etc.), suggesting that style transfer could be used to improve generalization in a wide range of ML tasks as a technique for generating new data. Potentially, it could also be used to improve generalization under domain shift.

Data Augmentation via Style Randomization. Building on the work of Geirhos et al. [7], Jackson et al. [18] propose to use style transfer as a form of data generation technique for DG. However, they use a different style transfer method proposed by Ghiasi et al. [8]. This style transfer method is also pre-trained on the Painter by Numbers dataset. Unlike the AdaIN style transfer method, Ghiasi et al. [8] use a style embedding space $z \in \mathbb{R}^{100}$ as input to the generator (decoder) network. This allows sampling arbitrary styles that are not available in a finite style source dataset and also speeds up the style transfer process. The authors sample z from a multivariate Gaussian distribution $\mathcal{N}(\mu, \Sigma)$, where μ and Σ are the empirical mean and covariance of style embeddings from the Painter by Numbers dataset. Additionally, control over the strength of style transfer is achieved by interpolating between the sampled style embedding and the style embedding of the content image:

$$z = \alpha\mathcal{N}(\mu, \Sigma) + (1 - \alpha)P(I_c)\,, \tag{4.2}$$

where P is the style predictor network and $\alpha \in [0, 1]$ is a parameter that controls the strength.

Incorporating this style transfer method into the training process is straightforward. Given an augmentation ratio $a : u$ of stylized (augmented) to original (unaugmented) images and a strength of style transfer α, each training image is stylized using a sampled style embedding z (as shown in (4.2)) with probability corresponding to the augmentation ratio. Otherwise, the image remains unchanged.

They find that this method improves standard classification performance on some datasets without domain shift, but not all. However, it consistently improves DG performance under domain shift. When combined with traditional data augmentation methods (such as random cropping, flipping, color jittering, etc.), it improves DG performance even further. The authors also perform experiments to find optimal values for the ratio $a : u$ and strength α hyperparameters. However, the differences in performance are not significant (except for $\alpha = 0$, which corresponds to no augmentation).

They further show that the method improves transfer learning performance in a monocular depth estimation task. Collecting real-world data to train depth estimation models is expensive and time-consuming. Using synthetic data from virtual environments is a common approach to reduce the cost of data collection. However, this introduces a domain shift between the synthetic training data and the real-world test data. Style transfer as a data augmentation can be used to reduce this domain shift. Specifically, the authors show that their method improves performance on real-world data, producing sharper output depth maps and fewer artifacts. Overall, Jackson et al. [18] demonstrate that style transfer can be used as a form of data generation for DG and can improve model robustness to domain shift.

Domain Generalization via Image Stylization. Another method that uses style transfer to generate new training data for DG problems has been proposed by Somavarapu et al. [34]. They use the AdaIN style transfer method, which requires an external style source dataset for pre-training and also an input style image for each style transfer during training. The training process is very similar to the one proposed by Jackson et al. [18]. During training, each image is stylized with a predefined probability p, otherwise it is left unchanged. However, they propose a modification that allows style transfer to be used for DG tasks without an external style source.

In addition to stylization by a random style image sampled from the external style source dataset, they also propose two other methods that do not require an additional external style source during training: inter-source and intra-source stylization. Inter-source stylization is performed by sampling the style image from other training (source) domains. For example, in the case of the PACS dataset, a sketch image could be stylized as a photo image. Intra-source stylization is more extreme, where the style image is sampled from the same domain as the content image. For example, a photo image could be stylized as another photo image.

All three methods (external, inter-source, and intra-source stylization) improve DG performance compared to the baseline without style transfer. As expected, the performance of intra-source stylization is worse compared to the method, which uses external style source dataset. However, inter-source stylization performance is surprisingly comparable to the method with external styles (while external stylization performance is still slightly better). This suggests that the style variability between

source domains in some datasets is sufficient for style transfer to be effective and that the external style source may not be completely necessary during training.

They also analyze a variant where the AdaIN style transfer method is pre-trained only on the source domains instead of the external style source dataset. However, this variant performs worse than the one with access to the external style source during pre-training. This suggests that the external style source dataset is still needed for pre-training the style transfer model to reach its full potential. Using external style images also during training seems to give the best results.

Rethinking Domain Generalization Baselines. Borlino et al. [4] propose to use style transfer data augmentation as a new baseline for DG problems and show that simple style transfer data augmentation method similar to those discussed above can outperform several state-of-the-art DG methods (including mixup discussed in Section 4.1.2) on several DG datasets (including PACS and OfficeHome). They also use the AdaIN style transfer method and propose using training domains as the only style source, similar to the inter-source stylization method [34]. They also find that not allowing the style transfer method to access external style source data during pre-training slightly reduces performance.

Similar to Jackson et al. [18], Borlino et al. [4] also perform an analysis of the hyperparameters p and α. They find that the optimal value of the style transfer strength is $\alpha = 1$ when using the AdaIN style transfer method as a data augmentation technique, regardless of the dataset or the stylization probability p. On the other hand, the optimal value for p is not so clear. With $\alpha = 1$, even small values of p can give good results, and the differences between different values of p are not significant.

4.2 Representation learning

Representation learning is a category of ML models in which we explicitly think of the model as composed of two parts. The first part of the model, the featurizer, is responsible for learning a good representation (features) that hopefully captures the underlying distribution of the input data. The second part of the model, the predictor, then uses this intermediate (latent) representation to make a prediction.

In our case, the prediction function h is decomposed as $h = f \circ g$, where $g : \mathcal{X} \to \mathcal{H}$ is a representation learning function (featurizer), $f : \mathcal{H} \to \mathcal{Y}$ is the classifier function (predictor), and $\mathcal{H}$ denotes the latent feature space. This decomposition is used by many DG methods. They can be divided into two subcategories based on how they approach learning the feature extraction function g: domain-invariant representation learning and feature disentanglement.

4.2.1 Domain-invariant representation learning

Methods in this category try to minimize the difference between learned feature representations of individual source domains. The argument is that if the representation learned by the featurizer g is invariant to domain labels, it is general and will perform well for unseen target domains. There have been many attempts to achieve domain invariance in latent feature space.

Probably the most notable and also most discussed method in this category is the Invariant Risk Minimization (IRM) proposed by Arjovsky et al. [3]. Instead of making the representations of different source domains match directly, IRM enforces

the optimal classifier f to be the same for all source domains. This can be formulated as:

$$\begin{aligned} \min_{\substack{g:\mathcal{X}\to\mathcal{H} \\ f:\mathcal{H}\to\mathcal{Y}}} \quad & \sum_{i=1}^{M} \hat{\mathcal{E}}^i(f \circ g) \\ \text{subject to} \quad & f \in \operatorname*{argmin}_{\bar{f}:\mathcal{H}\to\mathcal{Y}} \hat{\mathcal{E}}^i(\bar{f} \circ g) \qquad \forall\, i \in \{1, \dots, M\}\,, \end{aligned} \tag{4.3}$$

where $\hat{\mathcal{E}}^i(\cdot)$ is the estimated risk on the source domain S^i. However, Rosenfeld et al. [31] later proved that IRM lacks severely in formal guarantees and demonstrated that IRM fails to generalize under domain shift and does not improve on standard ERM on DG problems.

One method that attempts to explicitly align the latent feature spaces of different domains is Deep CORAL (or simply CORAL) [35]. CORAL stands for CORrelation ALignment and was originally developed for unsupervised DA. In its original formulation, CORAL tries to minimize the difference between the second-order statistics of the source and target domains. Specifically, it minimizes the squared Frobenius norm of the difference between the source and target covariance matrices using the following loss:

$$\mathcal{L}_{\text{CORAL}} = \|C_S - C_T\|_F^2\,, \tag{4.4}$$

where C_S and C_T are the covariance matrices of source and target computed in the latent feature space produced by the featurizer g. The CORAL loss $\mathcal{L}_{\text{CORAL}}$ is then combined with the classification loss $\mathcal{L}_{\text{CE}}$ to form the final objective function:

$$\mathcal{L} = \mathcal{L}_{\text{CE}} + \lambda \mathcal{L}_{\text{CORAL}}\,, \tag{4.5}$$

where λ is a weight that trades off between classification accuracy and domain alignment.

CORAL can be adapted to the DG setting by aligning only the correlations between the source domains, instead of aligning the source and target domains. This is done by computing the covariance matrix for each source domain separately and then summing over the differences between the covariance matrices of all pairs of source domains:

$$\mathcal{L}_{\text{CORAL}} = \frac{2}{M(M-1)} \sum_{i \neq j} \|C_S^i - C_S^j\|_F^2\,, \tag{4.6}$$

where C_S^i is the covariance matrix of the ith source domain. In this case, the CORAL loss $\mathcal{L}_{\text{CORAL}}$ is normalized by the number of source domain pairs to ensure that the loss does not grow with the number of domains.

4.2.2 Feature disentanglement

Disentangled representation learning methods aim to decompose each sample into independent parts that together contain all the information about the sample. They do this by learning a function that maps the features produced by the featurizer g to another latent space composed of several independent subspaces. Each subspace consists of a subset of dimensions, and together the subspaces span the entire latent space. Some subspaces are domain-shared (i.e., they contain domain-invariant features), while others are domain-specific. Samples can then be classified using only the domain-shared subspaces, which should be invariant to domain shift.

Ilse et al. [15] propose a semi-supervised method called Domain Invariant Variational Autoencoder (DIVA), which attempts to learn three independent latent subspaces, domain, class, and residual, using a VAE-type framework. This is achieved by using an encoder-decoder architecture with a single decoder and three encoders, one for each subspace. During training, an objective similar to the variational lower bound is used to ensure that the latent subspaces are independent and that the decoder is able to reconstruct the input from them. To further improve the disentanglement, two other auxiliary classifiers are used to predict the domain and class labels from the respective latent subspaces. This also ensures that the domain and class subspaces contain the desired information (i.e., domain-specific and domain-invariant features, respectively). Only the class encoder and the auxiliary class classifier are used for prediction.

4.3 Learning strategy

DG has also been addressed by many general ML paradigms, such as ensemble learning, meta-learning, gradient operation-based learning, distributionally robust optimization, or self-supervised learning. We will look at only two of them in more detail: ensemble learning and meta-learning.

4.3.1 Ensemble learning

Ensemble learning is a powerful ML technique that combines the predictions of multiple models to boost their performance. Apart from traditional ensemble learning, which trains the same model multiple times, several non-traditional approaches have been proposed specifically for DG. By using special ensemble architectures and training strategies, domain information can be exploited to improve generalization under domain shift.

For example, Mancini et al. [25] have trained a special neural network architecture consisting of multiple domain-specific classifiers (experts), one for each source domain, and a single domain classifier. The domain-specific classifiers are trained to predict correct class labels only on their respective source domains. They share the weights of all layers except the last one, which is domain-specific. The domain classifier is trained to assign samples to the correct source domain. At inference time, the predictions of the domain-specific classifiers are combined according to the weights predicted by the domain classifier.

4.3.2 Meta-learning

Meta-learning, often called learning-to-learn, is an alternative ML paradigm in which models are trained in multiple training episodes on related tasks, using experience from previous episodes to improve their learning performance. Meta-learning has also been used to improve generalization in DG.

Li et al. [23] have proposed MLDG, a meta-learning strategy for DG. MLDG splits the source domains into meta-train and meta-test domains in each iteration to simulate the domain shift during training. Gradient steps are required to improve not only on meta-train but also on meta-test. This is achieved by combining the standard cross-entropy loss on meta-train with the loss on meta-test, which is computed using parameters after a gradient step on metatrain. This requires a second derivative of the loss function with respect to the model parameters. Such an approach is model agnostic and can be therefore used with any model.

Chapter 5
Performance of state-of-the-art methods

In this chapter, we discuss the importance of proper evaluation methodology in DG research, introduce several testbeds that have been created to allow fair comparison of proposed DG methods and look at the results they provide.

5.1 Evaluation methodology

Gulrajani et al. [12] point out that many works in DG research use methodologically unsound model selection methods when evaluating the performance of proposed methods. Some works use testing (target) data to guide the model selection process (i.e., tuning of hyperparameters, choice of model checkpoint, etc.), which is exactly what we want to avoid in the DG setting. This data leakage leads to biased performance measures and makes it extremely difficult to compare different methods. Furthermore, researchers often use different datasets and backbone architectures, making fair comparisons almost impossible. Gulrajani et al. [12] further point out that many works do not report standard deviations of their results, making it difficult to assess the significance of the results.

5.1.1 Selection methods

To this end, Gulrajani et al. [12] recommend that test data should not be used for model selection (i.e., the oracle selection method). They further suggest that all DG methods should be responsible for specifying the model selection method they use and suggest two model selection methods suitable for DG settings: training-domain validation set and leave-one-domain-out cross-validation.

Training-domain validation set. This method splits each source domain into training and validation subsets. The training subsets are used to train the model as individual source domains. The validation subsets of all training domains are then combined into a single validation set. Finally, the model is selected to maximize performance on this validation set.

Leave-one-domain-out cross-validation. Given M source domains, we train M models with the same hyperparameters, but each model is trained on $M - 1$ domains with one domain held out. Each model is then evaluated on the held-out domain and the performance is averaged over all M models. Finally, we select the model with the highest average performance and retrain it again on all M domains.

5.2 Testbeds

For the reasons pointed out above, several testbeds have been created to provide a fair comparison of proposed DG methods and to aid future research. They aim to provide a

Algorithm	C-MNIST	R-MNIST	VLCS	PACS	OfficeHome	TerraInc	DomainNet	Avg
ERM	51.5 ± 0.1	98.0 ± 0.0	77.5 ± 0.4	85.5 ± 0.2	66.5 ± 0.3	**46.1 ± 1.8**	40.9 ± 0.1	66.6
IRM	52.0 ± 0.1	97.7 ± 0.1	**78.5 ± 0.5**	83.5 ± 0.8	64.3 ± 2.2	**47.6 ± 0.8**	33.9 ± 2.8	65.4
GroupDRO	52.1 ± 0.0	98.0 ± 0.0	76.7 ± 0.6	84.4 ± 0.8	66.0 ± 0.7	43.2 ± 1.1	33.3 ± 0.2	64.8
Mixup	52.1 ± 0.2	**98.0 ± 0.1**	77.4 ± 0.6	84.6 ± 0.6	**68.1 ± 0.3**	**47.9 ± 0.8**	39.2 ± 0.1	66.7
MLDG	51.5 ± 0.1	97.9 ± 0.0	77.2 ± 0.4	84.9 ± 1.0	66.8 ± 0.6	**47.7 ± 0.9**	41.2 ± 0.1	66.7
CORAL	51.5 ± 0.1	**98.0 ± 0.1**	**78.8 ± 0.6**	**86.2 ± 0.3**	**68.7 ± 0.3**	**47.6 ± 1.0**	**41.5 ± 0.1**	**67.5**
MMD	51.5 ± 0.2	97.9 ± 0.0	**77.5 ± 0.9**	84.6 ± 0.5	66.3 ± 0.1	42.2 ± 1.6	23.4 ± 9.5	63.3
DANN	51.5 ± 0.3	97.8 ± 0.1	**78.6 ± 0.4**	83.6 ± 0.4	65.9 ± 0.6	46.7 ± 0.5	38.3 ± 0.1	66.1
CDANN	51.7 ± 0.1	97.9 ± 0.1	77.5 ± 0.1	82.6 ± 0.9	65.8 ± 1.3	45.8 ± 1.6	38.3 ± 0.3	65.6
MTL	51.4 ± 0.1	97.9 ± 0.0	77.2 ± 0.4	84.6 ± 0.5	66.4 ± 0.5	45.6 ± 1.2	40.6 ± 0.1	66.2
SagNet	51.7 ± 0.0	98.0 ± 0.0	**77.8 ± 0.5**	**86.3 ± 0.2**	68.1 ± 0.1	**48.6 ± 1.0**	40.3 ± 0.1	67.2
ARM	**56.2 ± 0.2**	**98.2 ± 0.1**	77.6 ± 0.3	85.1 ± 0.4	64.8 ± 0.3	45.5 ± 0.3	35.5 ± 0.2	66.1
VREx	51.8 ± 0.1	97.9 ± 0.1	**78.3 ± 0.2**	84.9 ± 0.6	66.4 ± 0.6	46.4 ± 0.6	33.6 ± 2.9	65.6
RSC	51.7 ± 0.2	97.6 ± 0.1	77.1 ± 0.5	**85.2 ± 0.9**	65.5 ± 0.9	**46.6 ± 1.0**	38.9 ± 0.5	66.1

Table 5.1: Average target domain accuracies (in %, ± SE) from the DomainBed benchmark using the training-domain validation set as the model selection method. Data source: [11]

unified framework for evaluating DG methods and typically include implementations of several DG methods and their benchmarks on several DG datasets. They use the same datasets, backbone architectures, and model selection methods for all methods, which makes comparison of results much easier. They also compare the proposed methods to a baseline in the form of ERM [37], which does not account for domain shift at all.

5.2.1 DomainBed

DomainBed [11] introduced by Gulrajani et al. [12] is a "*a PyTorch [29] testbed for domain generalization*". It contains implementations of many DG methods (including IRM, mixup, MLDG, and CORAL, which we discussed in Chapter 4) and benchmarks them on several DG datasets (including PACS, OfficeHome, and DomainNet, which we discussed in Section 2.4).

It uses the ResNet-18 or ResNet-50 [13] CNN architecture as the backbone of the classification model and allows the use of grid search for hyperparameter tuning. It also provides three model selection criteria: training-domain validation set, leave-one-domain-out cross-validation, and test-domain validation set (i.e., oracle). DomainBed also features standard data augmentation techniques (e.g., random cropping, horizontal flipping, or color jittering), as they have been shown to reduce some variation between domains and improve performance under domain shift [12]. DomainBed can be easily extended with new algorithms and datasets. We show benchmark results from DomainBed in Table 5.1.

5.2.2 DeepDG

DeepDG [41] introduced by Wang et al. [40] is "*an easy-to-learn, easy-to-extend, and for-fair-comparison toolkit based on PyTorch for domain generalization*". DeepDG builds on DomainBed by simplifying the execution of experiments and adding some new features.

It also uses ResNet-18 and ResNet-50 architectures as the backbone for the classification models and uses hyperparameter grid search similar to DomainBed. However, at the time of writing, DeepDG offers a smaller number of algorithms and datasets compared to DomainBed. Table 5.2 shows the benchmark results of several algorithms on PACS and OfficeHome datasets in the DeepDG testbed.

	PACS					OfficeHome				
Algorithm	**Art**	**Cartoon**	**Photo**	**Sketch**	**Avg**	**Art**	**Clipart**	**Product**	**Real**	**Avg**
ERM	83.20	81.70	96.65	83.69	86.31	67.66	55.92	77.70	80.47	70.44
DANN	87.26	83.45	95.33	84.35	87.60	67.49	56.66	76.73	79.21	70.02
Mixup	**89.36**	82.08	96.65	84.63	**88.18**	67.41	**58.24**	78.46	**80.84**	**71.24**
RSC	87.84	80.33	**97.72**	81.50	86.85	66.30	55.21	76.95	79.00	69.36
MMD	85.74	83.58	95.51	83.46	87.07	66.71	56.54	78.37	79.00	70.36
CORAL	86.77	**84.04**	94.85	**85.95**	87.90	66.58	56.17	**78.55**	79.76	70.27
GroupDRO	84.18	83.15	96.11	83.94	86.84	66.87	57.04	77.97	79.69	70.39
ANDMask	84.91	76.45	**97.72**	81.19	85.07	62.30	54.78	74.99	78.31	67.59
Vrex	87.11	82.85	96.95	84.07	87.75	**68.19**	56.29	78.35	80.42	70.81

Table 5.2: Average target domain accuracies (in %) from the DeepDG benchmark on PACS and OfficeHome datasets using the training-domain validation set as the model selection method. Data source: [41]

5.3 Performance

The benchmark results shown do not contradict each other and provide similar results. Minor discrepancies may be caused by the fact that none of the methods is thoroughly tuned and both testbeds use slightly different experimental settings. As can be seen from both benchmarks, none of the existing methods can significantly improve on standard ERM (especially in the DomainBed benchmark). On the contrary, some methods specifically designed to deal with DG are inferior to ERM, which does not deal with domain shift in any special way. Methods that show some improvement over ERM do so only marginally.

This is likely due to the fact that, unlike other work in DG research, Gulrajani et al. [12] carefully tune the hyperparameters of the ERM algorithm and use strong data augmentation techniques during training. They also employ a larger network architecture (ResNet-50) than most previous works. This suggests that standard techniques used to improve in-distribution generalization may also be effective for OOD generalization and generalization under domain shift. Therefore, Gulrajani et al. [12] suggest the use of large network architectures, strong data augmentation techniques, and careful hyperparameter tuning with ERM as a baseline for future DG research.

One DG method that seems to provide a consistent performance improvement over ERM is CORAL, which we discussed in Section 4.2.1. Although this improvement is consistent across most datasets of both benchmarks, it is still marginal. The performance of other methods seems to be highly dependent on the dataset. For example, mixup, discussed in Section 4.1.2, works very well on some datasets, but is outperformed by ERM baseline on others.

Gulrajani et al. [12] also point out that the model selection method is important and has a significant impact on the results. Using the training-domain validation set selection method outperforms leave-one-domain-out cross-validation on most datasets and algorithms. They further show that using oracle selection (i.e., using testing data for model selection) leads to better performance of all methods. They report about +2% accuracy improvement for most methods when using the oracle selection method compared to training-domain validation set. Although these results are corrupted by the data leakage and should not be reported as true performance of given models, they suggest that there is possible headroom for future improvement. Models that perform better on target domains do exist and can be trained. However, current methodologically sound model selection methods without data leakage are unable to find them.

Overall, the results of both benchmarks are rather pessimistic and there is still a lot of room for improvement. Most current methods fail to improve on standard ERM, and those which do, improve only marginally. This casts doubt on the effectiveness of current DG methods and their ability to address the problem of domain shift, and raises other important questions for discussion. Is ERM as good as it gets? Is it even possible to achieve better performance in DG tasks? Are current datasets and benchmarks suitable for DG research?

Gulrajani et al. [12] point out the power of data augmentation techniques and careful hyperparameter tuning and suggest that future DG research should focus on these aspects. Data augmentation, if properly applied, can remove the spurious correlations that vary between domains and help learn more domain-invariant representations that would generalize better to unseen domains. Borlino et al. [4] join this call for new methods and further suggest using style transfer as a data augmentation technique, which has been shown to be effective in the DG setting.

Chapter 6
Style transfer in domain generalization

In this chapter, we explore the use of style transfer in DG in more depth than was previously done by the methods discussed in Section 4.1.3. Motivated by the success of data augmentation methods in DG [12] and style transfer as a data augmentation technique [7], we focus on the use of style transfer as a data generation method for improving DG performance. Style transfer has been used in DG before, but only as a random augmentation method without any structure [18, 34, 4]. We propose a novel method that uses style transfer to generate new data in a more structured way and study its effect alone and in combination with the CORAL DG method [35]. Furthermore, we propose a novel data generation method based on style transfer that focuses on model selection and validation procedure instead of changing the training procedure.

6.1 Random stylization

As already mentioned, style transfer has been used in DG before, but only as a random augmentation method without any structure [18, 34, 4]. The methods proposed by these works are almost identical and are based on the same idea. They differ only in the choice of style transfer method and the choice of style source. Jackson et al. [18] use the style transfer method proposed by Ghiasi et al. [8], which was pre-trained on the Painter by Numbers dataset [28] and sample style embeddings from latent space. Somavarapu et al. [34] and Borlino et al. [4] use the AdaIN style transfer method [14], also pre-trained on the Painter by Numbers dataset and sample style images from other source domains. Below, we describe the random stylization method as we implemented it in our own experiments to be able to compare it with our proposed method.

The random stylization method is very straightforward. We use the AdaIN style transfer method pre-trained on the Painter by Numbers dataset. We also use the Painter by Numbers dataset as a source of style images during training, as it yields the best results. We adopt the way of constructing a batch of training data from DomainBed [11]. The batch is divided into M equally sized parts, each part containing images from one source domain. The random stylization method has two hyperparameters: the probability of stylization p and the strength of stylization α. When constructing a batch of training data, each image is stylized with probability p using a randomly sampled style image from the Painter by Numbers dataset with stylization strength α as described in (3.16). This results in a batch of training data where p of all samples are stylized on average.

An example of a training batch generated by this random stylization method is shown in Figure 6.1. We can see that the stylization is random and unstructured and serves only as a random data augmentation method. In this example, we use the PACS dataset with $M = 3$ source domains (Cartoon, Photo, and Sketch) and 36 images per batch. This results in 12 images per source domain. In this example, we set the stylization probability $p = 0.25$ and the stylization strength $\alpha = 1$.

Figure 6.1: Random stylization batch example from the PACS dataset.
batch size $= 36$, $p = 0.25$, $\alpha = 1$

6.2 Aligned stylization

The random stylization method, as described in Section 6.1, is very simple and straightforward. Despite its simplicity, it has been shown to be effective in generalizing under domain shift. However, it has no structure and its goal is only to provide more diverse training data. We try to improve this method by introducing a more structured way of stylizing training data with a specific goal in mind. We call this method aligned stylization.

The idea behind aligned stylization is to simulate more source domains by duplicating a part of each source domain within a batch and stylizing each duplicate with a different style image. This results in more synthetic source domains within each batch because we consider each stylization group to be a separate source domain. Furthermore, each training image occurs multiple times within a batch but is stylized with a different style each time. This way, the model should be forced to learn more domain/style invariant features after averaging the gradients within each batch. Our hypothesis is that style-dependent features will be averaged out and only style-invariant features will remain, resulting in better generalization under domain shift. Below, we describe the aligned stylization method in more detail.

We use the same Painter by Numbers dataset and AdaIN style transfer method as in the random stylization method. The aligned stylization method has two hyperparameters: the number of style groups G and the strength of the stylization α. The batch construction is again based on DomainBed. As before, the batch is divided into M equally sized parts, each part containing images from one source domain. However, these parts are further divided into G groups. The groups are duplicated within each source domain, but each duplicate is stylized with a newly sampled style image, except for the first group, which is left in its original form. The stylization is applied with the stylization strength α as described in (3.16). This results in a batch of training data where exactly $\frac{G-1}{G}$ of all samples are stylized. Furthermore, the batch contains G times more synthetic source domains compared to the random stylization method.

Figure 6.2 shows an example of a training batch generated by our proposed aligned stylization method. Again, we use the PACS dataset with $M = 3$ source domains (Cartoon, Photo, and Sketch) and 36 images per batch. As before, this results in 12 images per source domain. However, in this case, these domain batches are further divided into 4 groups, each containing 3 images, since we are using $G = 4$. These groups contain the same duplicated images within each source domain, but each group is stylized with a different style image, except for the first original group. This results in each image appearing 4 times within a batch, but each time with a different style.

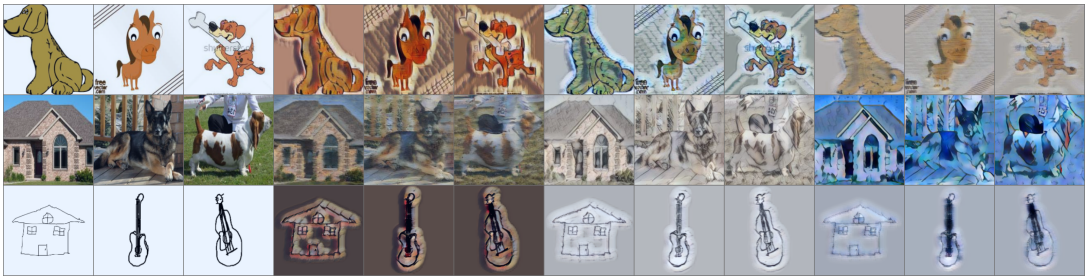

Figure 6.2: Aligned stylization batch example from the PACS dataset.
batch size = 36, $G = 4$, $\alpha = 1$

6.2.1 CORAL synergy

The aligned stylization method may improve generalization under domain shift on its own, as described above. However, we hypothesize that it may work even better in combination with the CORAL DG method [35] described in Section 4.2.1. When applied to DG problems, CORAL attempts to align second-order statistics computed in feature space between individual source domains. Specifically, it tries to align the covariance matrices between each pair of source domains in a given batch by adding a CORAL loss term to the original loss function (see (4.6)).

By using aligned stylization, we can simulate more source domains within each batch. Each stylization group is treated as a separate source domain by assigning a different domain label to each group. Furthermore, each training image occurs multiple times within a batch, but each time within a different stylization group (i.e., belonging to a different synthetic source domain). By combining aligned stylization with CORAL, in addition to aligning the original source domains, CORAL also attempts to align the covariance matrices between groups of the same images with different styles. We conjecture that by combining these two methods, we can force the model to learn even more domain/style invariant features than with either method alone, and thus further improve generalization under domain shift. Note that this combination is only possible with aligned stylization, as random stylization does not simulate more source domains within a batch, but only augments individual images.

6.3 Stylized validation

Both random stylization and aligned stylization are data generation methods that are used only during training. They work by generating more diverse training data, which improves generalization under domain shift by helping the model learn more domain-invariant features. Gulrajani et al. [12] find that there is a significant gap between model performance when using the held-out training-domain validation set for model selection versus the oracle selection method (i.e., test-domain validation set). Inspired by this finding, we propose a novel data generation method that focuses on validation and model selection procedure instead of augmenting training data.

The gap mentioned above could be explained by the fact that in the case of training-domain validation, the validation data come from the same distribution as the training data. However, the goal of DG is to generalize to previously unseen domains. Therefore, it makes sense to validate and select the model based on its performance on unseen domains as well. The performance achieved on held-out training data may not be a good indicator of performance on unseen domains, since the model was trained on

data from the same distributions.

One way to address this issue is to use the leave-one-domain-out cross-validation procedure proposed by Gulrajani et al. [12], as described in Section 5.1.1. However, this method has been shown to perform poorly in practice. This may be due to the fact that DG datasets contain a small number of source domains, and by holding out one source domain for validation, we significantly reduce the variety of training data and thus the ability of the model to generalize.

In an attempt to address this issue and close the gap between oracle and validation performance, we propose a novel data generation method that utilizes style transfer and specifically targets the validation procedure. We use the same AdaIN style transfer method and Painter by Numbers dataset as in the previous methods. However, instead of using it to generate more training data, we use it to generate new validation data in order to simulate previously unseen domains.

Stylized validation does not change the training procedure in any way and can therefore be used in combination with any DG method. It has two hyperparameters: the number of validation styles $\mathcal{S}_{val}$ and the stylization strength α. We first sample $\mathcal{S}_{val}$ style images from the Painter by Numbers dataset and create a new validation domain for each pair of source domain and style image. Each validation domain contains images from the corresponding source domain stylized with the corresponding style image using the stylization strength α as described in (3.16). Note that only the part of the source domain held out for validation is stylized and used for validation. This results in $M \cdot \mathcal{S}_{val}$ validation domains. The original unstylized source domains are not used for validation in this case.

During validation, performance is measured on each validation domain separately and then averaged across all validation domains with equal weights. This means that we treat all validation domains equally, regardless of the number of images they contain. This is motivated by the fact that we want the model to generalize equally well to all unseen domains. Finally, we select the model with the best average performance across all validation domains.

Chapter 7
Experimental evaluation

In this chapter, we describe the experimental setup and implementation details of the experiments. We then look at the search for the best values of some of the hyperparameters. Finally, we present and discuss the results of the experiments in which we compare our proposed methods with ERM [37] baseline and random stylization.

7.1 Experimental setup

In this section, we describe in detail the experimental setup of the experiments presented in Section 7.3. Our goal is to compare our proposed method, aligned stylization, with the baseline ERM method and random stylization. Furthermore, we analyze the effect of CORAL [35] synergy with aligned stylization. Finally, we also study the effect of stylized validation on the performance of the models, both alone and in combination with other methods.

7.1.1 Prestylized datasets

As mentioned in Sections 6.1 and 6.2, we use the AdaIN style transfer method [14] as our style transfer method of choice. We use the implementation and models pre-trained on the Painter by Numbers dataset [28] provided by Inoue [16]. Specifically, we use the style transfer method as a preprocessing step to create prestylized datasets [26]. Prestyling makes the training process easier and faster, since we do not have to style images on the fly during training. However, it also results in a limited number of styles.

We randomly sample 1000 style images from the Painter by Numbers dataset and create a stylized version of the PACS and OfficeHome datasets for each of the 1000 styles. We use only PACS [24] and OfficeHome [38] datasets for our experiments as their size allows us to create prestylized versions for all 1000 styles in a reasonable amount of time and disk space. These prestylized datasets are then used as a source of stylized images for both training stylization (random and aligned) and stylized validation. We use the same 1000 styles for all experiments to ensure a fair comparison. We also study the effect of limiting the number of styles available during training of random and aligned stylization.

7.1.2 Framework

We use Python for all experiments, in particular the PyTorch deep learning library [29]. We implement our own framework for the evaluation of DG methods, inspired by DomainBed [11]. This framework eases working with prestylized datasets, as it requires the implementation of specific dataset and sampler classes. It includes implementations of our proposed methods: aligned stylization and stylized validation. Furthermore, we also implemented our own random stylization, since no code was available for this

method at the time of writing. In addition to the methods mentioned above, we use DomainBed's implementation of CORAL and adapt it to our framework. The source code of our framework is available in the attachment and its structure is described in Appendix C.

7.1.3 Hardware

All experiments were run on the Research Centre for Informatics (RCI) cluster, a high-performance computing infrastructure intended for research purposes. Specifically, the experiments were run in parallel on multiple nodes with AMD EPYC 7543 32-core CPUs and NVIDIA TESLA A100 40GB GPUs. Despite the high performance of the cluster, it took several days to complete the experiments due to their large number. The prestyling step is also very time-consuming. Therefore, we do not compare the methods in terms of training time, but focus only on the performance of the models.

7.1.4 Training

We use ResNet-50 [13] as the backbone for the classification models in all experiments with weights pre-trained on ImageNet dataset [32]. The classification head is replaced by a simple linear layer with the number of outputs equal to the number of classes in the given dataset. ResNet-50 takes images of size 224x224 as input, so all images must be resized to this size.

We use the Adam optimizer and cross-entropy loss for all experiments (except CORAL, which adds a penalty term to the loss). Since we encountered exploding gradients during training, we also use gradient clipping. Similar to DomainBed, we use basic image augmentation techniques during training. Specifically, we use random horizontal flipping, random cropping, random grayscale, and color jitter, which randomly changes the brightness, contrast, saturation, and hue of the images. We do not use these basic augmentation techniques during validation and testing, only during training.

We deal with the class imbalance of the PACS and OfficeHome datasets by sampling with replacement during training according to the frequency of each class. The models are trained in a multi-source, single-target setting, where one of the domains is held out and used as the target domain, and the remaining domains are used for training. This is repeated for all domains and the results are stored separately for each domain and also averaged over all domains. The target domain is used in its entirety for evaluation, while the source domains are split into training and validation sets in an 80:20 ratio. We train all models for 2500 steps (i.e., on 2500 batches) and evaluate them on both validation and test sets every 50 steps. This results in 50 checkpoints, which are then used by the model selection method to select the best model. Since we are using pre-trained models, the training curves seem to saturate after this number of steps, and training for longer does not seem to improve performance.

We repeat each experiment multiple times with different seeds to reduce the effect of randomness and obtain more reliable results. This allows us to report the standard error (SE) of the mean test accuracy for each experiment. In addition to all random operations, the seed also determines the train-validation splits of the source domains and the limited subset of styles used for stylization and style validation. Therefore, to ensure a fair comparison, we use the same set of seeds for all methods and hyperparameter values.

7.1.5 Hyperparameters

For most of the hyperparameters, we use the same values as in DomainBed or values that are commonly used in deep learning. For some of the hyperparameters, namely the number of validation styles, the number of style groups in aligned stylization, and the CORAL penalty weight λ, we perform a search for the best value in the next Section 7.2. The limit on the number of training styles in random and aligned stylization is studied in the main results in Section 7.3.

We adjust the commonly used batch size of 256 used in GPU-accelerated deep learning to 240 to be divisible by the number of source domains in the PACS and OfficeHome datasets. This results in 80 images per source domain per batch, since both datasets have 4 domains, but one of them is always used as the target domain. The domain batch size of 80 is nice to work with as it allows us to experiment with different values for the number of groups in the aligned stylization (domain batch size must be divisible by the number of groups). Furthermore, we use learning rate of 0.0001, weight decay of 0.0001, and gradient clipping with maximum norm of 1.0.

We set the styling strength $\alpha = 1$ for the AdaIN style transfer method, as it was found to give the best results. The styling probability p is set to 0.1 for random stylization because Somavarapu et al. [34] use it as a default value and other works mention that performance is not very sensitive to this hyperparameter.

7.2 Hyperparameter search

In this section, we look at the results of searching for the best values of some of the hyperparameters mentioned in Section 7.1.5. We perform the search on the PACS dataset only, using 3 different seeds for each hyperparameter value and target domain. We search for values of hyperparameters that are new to our proposed methods and for which we have no intuition. We also search for the best value of the CORAL penalty weight λ, since it is an important hyperparameter of the CORAL method. The search is performed in isolation, i.e. we keep all other hyperparameters fixed. Also note that basic image augmentation is not used in these searches, and therefore the results differ slightly from the results in Tables 5.1 and 7.4.

7.2.1 CORAL λ

We first look at the effect of the CORAL penalty weight λ on the performance of the model and compare it to the baseline ERM method. In CORAL, λ is used to trade off between the classification loss and the CORAL penalty term. The higher the value of λ, the more emphasis is placed on the alignment of the source domains. We perform the search on the PACS dataset with 3 different seeds for each value of λ and target domain. The values of λ we test are 0.125, 0.25, 0.5, 1.0, 2.0, 4.0, and 8.0. The results are shown in Table 7.1.

The first thing to note is that CORAL outperforms the ERM baseline in average performance for all values of λ. This is consistent with the results from the testbed benchmarks in Section 5.3. CORAL seems to help the most on the Art and Sketch domains. On the other hand, the performance on the Cartoon domain is surprisingly worse than that of ERM baseline for all values of λ. However, this changes when CORAL is used in combination with basic image augmentation (see Tables 5.2 and 7.4).

Alg.	λ	Art	Cartoon	Photo	Sketch	Average
ERM		82.99 ± 0.40	**82.57 ± 1.10**	**96.47 ± 0.60**	64.60 ± 4.75	81.66 ± 1.14
CORAL	0.125	85.66 ± 0.17	**80.70 ± 2.03**	**96.97 ± 0.39**	69.46 ± 0.57	**83.20 ± 0.55**
	0.25	**87.13 ± 0.77**	79.49 ± 1.87	96.03 ± 0.26	**71.25 ± 0.89**	**83.47 ± 0.51**
	0.5	**85.19 ± 2.24**	**81.64 ± 0.74**	96.33 ± 0.07	69.20 ± 1.38	**83.09 ± 0.50**
	1.0	85.21 ± 1.04	**82.18 ± 0.59**	96.31 ± 0.44	70.03 ± 1.26	**83.43 ± 0.06**
	2.0	84.49 ± 0.34	**81.63 ± 0.51**	96.19 ± 0.55	70.05 ± 1.04	**83.09 ± 0.50**
	4.0	82.70 ± 0.72	**81.30 ± 1.08**	**97.17 ± 0.35**	**69.85 ± 3.37**	**82.75 ± 1.03**
	8.0	**86.25 ± 1.70**	80.38 ± 0.20	95.87 ± 0.54	**72.23 ± 0.88**	**83.68 ± 0.32**

Table 7.1: CORAL penalty weight λ hyperparameter search results on the PACS dataset. Average target domain accuracies (in %, ± SE) over 3 runs.

Another thing to observe is that the performance of CORAL is actually not that sensitive to the value of λ, at least in this case. The performance on individual domains varies a bit, but the average performance is very similar for all values of λ. The best value of λ seems to be 8.0 as it achieves the best average performance. However, the value 8.0 seems to be a bit extreme and the differences between the other values of λ are so negligible that we decide to use the value 1.0 for the main experiments, as this is also the default value used for CORAL in DomainBed.

7.2.2 Number of validation styles

The next hyperparameter we look at is the number of styles used for the stylized validation $\mathcal{S}_{val}$. As described in Section 6.3, $\mathcal{S}_{val}$ styles are randomly sampled from the external set of styles and used to create a new validation set for each source domain to simulate unseen domains. Note that stylized validation becomes computationally expensive as the number of styles increases, since the model must be evaluated on each combination of style and source domain validation set. Also bear in mind that stylized validation does not use the original source domain validation sets. Here we also perform the search on the PACS dataset with 3 different seeds for each value of $\mathcal{S}_{val}$ and target domain. We test values of 1, 2, 4, 8, 16, 32, and 64 for the number of validation styles $\mathcal{S}_{val}$. The results of this search are shown in Table 7.2.

As expected, the average performance of the model increases with increasing number of styles used for stylized validation. When using a small number of styles, the baseline model validated on the original validation sets performs better than the models using stylized validation. Using a larger number of styles can surpass the baseline model, but

StyVal	$\mathcal{S}_{val}$	Art	Cartoon	Photo	Sketch	Average
False		82.99 ± 0.40	**82.57 ± 1.10**	96.47 ± 0.60	**64.60 ± 4.75**	**81.66 ± 1.14**
True	1	82.49 ± 1.12	78.97 ± 1.01	96.41 ± 0.54	61.84 ± 1.73	79.93 ± 0.61
	2	**82.94 ± 1.37**	78.47 ± 0.76	**96.77 ± 0.48**	56.95 ± 1.08	78.78 ± 0.15
	4	**83.11 ± 1.52**	78.47 ± 0.76	**97.33 ± 0.19**	59.47 ± 1.59	79.59 ± 0.84
	8	**84.10 ± 1.49**	**80.79 ± 1.15**	96.49 ± 0.50	**62.48 ± 2.66**	**80.96 ± 0.73**
	16	**85.17 ± 0.97**	78.65 ± 0.83	96.53 ± 0.48	**68.14 ± 3.02**	**82.12 ± 0.68**
	32	**84.47 ± 1.38**	79.42 ± 2.01	**96.81 ± 0.44**	**67.99 ± 2.87**	**82.17 ± 0.98**
	64	**85.22 ± 0.94**	79.39 ± 1.98	96.53 ± 0.48	**68.05 ± 2.93**	**82.30 ± 1.20**

Table 7.2: Stylized validation $\mathcal{S}_{val}$ hyperparameter search results on the PACS dataset. Average target domain accuracies (in %, ± SE) over 3 runs.

Styling	G	Art	Cartoon	Photo	Sketch	Average
none		82.99 ± 0.40	$\mathbf{82.57 \pm 1.10}$	$\mathbf{96.47 \pm 0.60}$	64.60 ± 4.75	81.66 ± 1.14
align	4	$\mathbf{88.64 \pm 0.56}$	$\mathbf{83.45 \pm 0.09}$	$\mathbf{95.73 \pm 0.14}$	$\mathbf{86.25 \pm 0.70}$	$\mathbf{88.52 \pm 0.32}$
	8	$\mathbf{88.22 \pm 0.26}$	81.53 ± 0.44	95.69 ± 0.12	$\mathbf{84.64 \pm 1.18}$	87.52 ± 0.30
	16	$\mathbf{88.28 \pm 0.10}$	81.00 ± 0.59	94.85 ± 0.28	$\mathbf{86.20 \pm 0.89}$	87.58 ± 0.30

Table 7.3: Aligned stylization G hyperparameter search results on the PACS dataset. Average target domain accuracies (in %, ± SE) over 3 runs.

the improvement is marginal and not significant, taking into account the standard errors. As using a large number of styles is computationally expensive and the performance seems to saturate around 16 styles, we decide to use $\mathcal{S}_{val} = 16$ for the main experiments.

Similar to Section 7.2.1, the biggest improvement can be observed in the Art and Sketch domains. Again, the performance of Cartoon surprisingly degrades regardless of the number of styles used.

7.2.3 Number of aligned stylization groups

Finally, we take a look at the effect of the number of style groups G on the performance of the aligned stylization method. As described in Section 6.2, during aligned stylization, the parts of the batch that belong to individual source domains are further split into G groups. The groups within each domain are duplicated and stylized with different styles. Thus, G controls both the number of synthetic source domains in the given batch and the number of times each image is duplicated within that batch. The domain batch size (number of images per source domain per batch) must be divisible by G. The search is again performed on the PACS dataset with 3 different seeds for each value of G and target domain. We test values of 4, 8, and 16 for the number of style groups G, since these values divide the domain batch size of 80 without remainder. In this case, the number of training styles from which we randomly sample the style for each group is limited to 64. The results of this search can be seen in Table 7.3.

The results show that the aligned stylization significantly improves upon the baseline method without any stylization. Again, the improvement is most noticeable on the Art and Sketch domains, with the Sketch domain achieving an improvement of more than 20%. Similar performance gains can be achieved by basic augmentation techniques that include, among others, random color jitter (see ERM baseline in Table 7.4). Images from the PACS Sketch domain are simple black line drawings with a white background. We assume that by augmenting training images with color (either by random color jitter or by stylization), the model is able to learn more robust features that are not dependent on the color and thus generalize better to the black and white images from the Sketch domain. On the other hand, performance on the Photo domain degrades slightly compared to the baseline.

Smaller values of G seem to perform better than larger values. It may be that a smaller number of styles (such as $G = 4$) is sufficient to increase the diversity of the batch and that a larger number of styles does not provide any additional benefit. On the other hand, larger values of G result in too many duplicate images in the batch, reducing the overall information content of the batch. The best value of G is 4, as it achieves the best performance on all domains and improves on the baseline even on the Cartoon domain. We therefore use $G = 4$ for the main experiments.

7.3 Comparison of methods

In this section, we present and discuss the main results of our experiments. We study the impact of our proposed methods on the performance of the model and compare them to the ERM baseline. We also compare our novel aligned stylization method with the random stylization method [18, 34, 4]. Furthermore, the methods were designed to work in combination with each other. Therefore, in addition to comparing individual methods, we also study the effect of combining them. We are particularly interested in the synergy of aligned stylization with CORAL, as described in Section 6.2.1. The only combination of methods that we do not study is the combination of CORAL with random stylization, since they were not designed to work together. We also analyze how performance changes with increasing limit on the number of training styles in random and aligned stylization. Values of 8, 64, and 256 are used for the limit on the number of training styles $\mathcal{S}_{train}$.

The comparison is performed on the PACS and OfficeHome datasets. Basic image augmentation techniques have been found to significantly improve DG performance [12]. Therefore, we present results with basic image augmentation enabled in this section, as we are mainly interested in improving current state-of-the-art methods. Results without basic image augmentation are shown separately in Appendix A. The hyperparameters are set as described in Section 7.1.5 or to the values found by the hyperparameter search in Section 7.2. The results are shown in Tables 7.4 and 7.5 for PACS and OfficeHome datasets, respectively. Each combination of methods and hyperparameters is run 5 times with different seeds, and the average accuracies in the target domain are reported with standard errors. This results in 22 rows $\times$ 4 domains $\times$ 5 seeds $=$ 440 runs in total for each dataset. The best results in each column are highlighted in bold, taking into account the standard errors.

7.3.1 PACS

The accuracies achieved on the PACS dataset are generally very high. Specifically, on the Photo domain, the accuracies are very close to 97%. This is due to the fact that the Photo domain is very similar to the ImageNet dataset, which is used to pre-train the ResNet-50 backbone of the models. Therefore, generalizing to the Photo domain is not very challenging. Accuracies for the remaining domains are lower, but still very high.

CORAL improves the ERM baseline method on all domains except Photo, where performance is already very high. This is consistent with the results from the testbed benchmarks in Section 5.3 and shows that CORAL improves generalization to unseen domains. The improvement is most noticeable on the Cartoon and Sketch domains, probably because these domains are the most different from the ImageNet on which the model is pre-trained. This trend can be observed for all other DG methods as well.

Random stylization outperforms the baseline ERM method on all domains except Photo, similar to CORAL. Furthermore, it also outperforms CORAL on the same domains. This confirms that style transfer data augmentation is a powerful method for DG setting. The response to different values of $\mathcal{S}_{train}$ is interestingly quite unstable and varies from domain to domain. Random stylization also achieves the best performance of all methods on the Cartoon domain. However, the lead over the aligned stylization is only marginal and not significant considering the standard errors.

The aligned stylization method suffers the greatest performance loss on the Photo

Alg.	StyVal	Styling	$\mathcal{S}_{train}$	Art	Cartoon	Photo	Sketch	Average
ERM	False	none		85.81 ± 0.67	81.57 ± 0.73	**96.72 ± 0.27**	81.69 ± 1.72	86.45 ± 0.55
		align	8	88.77 ± 0.53	**84.96 ± 0.61**	95.75 ± 0.20	88.31 ± 0.27	89.45 ± 0.23
			64	88.26 ± 0.46	**84.73 ± 0.37**	95.99 ± 0.20	89.20 ± 0.10	89.54 ± 0.18
			256	88.62 ± 0.48	**85.24 ± 0.70**	95.99 ± 0.19	**89.38 ± 0.58**	**89.81 ± 0.20**
		rand	8	88.89 ± 0.20	**85.06 ± 0.58**	96.37 ± 0.14	86.06 ± 0.82	89.10 ± 0.21
			64	88.70 ± 0.43	**84.78 ± 0.69**	96.36 ± 0.15	87.56 ± 0.96	89.35 ± 0.46
			256	87.84 ± 0.77	**85.27 ± 0.48**	96.47 ± 0.20	86.16 ± 1.07	88.94 ± 0.37
	True	none		85.55 ± 0.59	81.60 ± 0.90	**96.61 ± 0.19**	83.63 ± 0.89	86.85 ± 0.42
		align	8	88.84 ± 0.27	84.39 ± 0.49	95.88 ± 0.09	87.48 ± 0.42	89.15 ± 0.19
			64	**89.12 ± 0.41**	84.13 ± 0.47	95.90 ± 0.14	**89.52 ± 0.46**	89.67 ± 0.18
			256	**89.72 ± 0.19**	84.39 ± 0.57	96.22 ± 0.10	**89.80 ± 0.70**	**90.03 ± 0.26**
		rand	8	**89.21 ± 0.95**	**85.34 ± 0.26**	96.44 ± 0.24	85.89 ± 1.32	89.22 ± 0.47
			64	88.92 ± 0.36	**84.53 ± 0.95**	**96.56 ± 0.16**	88.07 ± 0.40	**89.52 ± 0.39**
			256	**89.20 ± 0.38**	**85.35 ± 0.47**	**96.63 ± 0.08**	87.63 ± 0.54	**89.70 ± 0.28**
CORAL	False	none		86.28 ± 0.41	84.06 ± 0.52	**96.63 ± 0.39**	83.92 ± 0.54	87.72 ± 0.24
		align	8	88.74 ± 0.21	83.89 ± 1.16	96.01 ± 0.35	86.28 ± 0.67	88.73 ± 0.43
			64	88.98 ± 0.18	84.61 ± 0.43	96.18 ± 0.20	87.01 ± 1.71	89.20 ± 0.38
			256	88.66 ± 0.65	84.66 ± 0.39	96.05 ± 0.19	**89.54 ± 0.49**	**89.73 ± 0.25**
	True	none		86.59 ± 0.33	83.26 ± 0.40	**96.87 ± 0.17**	83.66 ± 0.26	87.60 ± 0.20
		align	8	**89.29 ± 0.40**	**84.83 ± 0.33**	96.36 ± 0.18	86.61 ± 0.92	89.27 ± 0.25
			64	**89.36 ± 0.39**	84.44 ± 0.41	96.06 ± 0.20	88.49 ± 0.59	89.59 ± 0.14
			256	88.99 ± 0.39	**84.91 ± 0.35**	95.90 ± 0.12	**90.11 ± 0.29**	**89.98 ± 0.08**

Table 7.4: Comparison of methods on the PACS dataset with basic image augmentation. Average target domain accuracies (in %, ± SE) over 5 runs.

domain. However, it improves significantly on all other domains compared to ERM and even outperforms CORAL, similar to random stylization. In contrast to random stylization, the performance of aligned stylization seems to increase with increasing limit on the number of training styles $\mathcal{S}_{train}$, especially on the Sketch domain. When comparing the performance of aligned and random stylization, aligned stylization comes out on top when the results are averaged over all domains. The largest difference is observed on the Sketch domain, where aligned stylization excels. On average, aligned stylization achieves about 1% higher accuracy than random stylization. In combination with stylized validation, the average performance of aligned stylization is the best of all methods, closely followed by random stylization. The synergy between aligned stylization and CORAL is not as strong as we expected in this case. In fact, the performance of this combination is comparable to the performance of aligned stylization alone.

Stylized validation does not seem to improve performance when used with ERM baseline or CORAL. The only exception is the Sketch domain, where the performance of ERM improves by about 2% when stylized validation is used. On the other hand, stylized validation works well with both stylization methods and slightly improves their performance in all domains. We suspect that this is related to the fact that the stylization methods, unlike the ERM and CORAL methods, saw other stylized images during training. However, we are not sure of the exact reason for this behavior.

7.3.2 OfficeHome

The accuracies achieved on the OfficeHome dataset are lower than those on PACS in general. The OfficeHome dataset is more challenging compared to PACS because it contains more classes (65 vs. 7), making it more difficult to correctly classify the images. Furthermore, as mentioned in Section 2.4.3, the Art, Product, and Real domains in the OfficeHome dataset are very similar to each other. Therefore, the assumption of a

Alg.	StyVal	Styling	$\mathcal{S}_{train}$	Art	Clipart	Product	Real	Average
ERM	False	none		65.16 ± 0.23	48.50 ± 0.28	76.09 ± 0.30	78.50 ± 0.22	67.06 ± 0.10
		align	8	63.58 ± 0.30	51.47 ± 0.42	73.62 ± 0.34	75.87 ± 0.31	66.14 ± 0.07
			64	64.10 ± 0.35	52.89 ± 0.21	73.12 ± 0.18	75.74 ± 0.22	66.46 ± 0.14
			256	63.50 ± 0.40	53.23 ± 0.30	73.16 ± 0.23	75.59 ± 0.28	66.37 ± 0.20
		rand	8	65.72 ± 0.37	49.41 ± 0.70	76.21 ± 0.23	78.68 ± 0.12	67.50 ± 0.15
			64	66.13 ± 0.55	50.51 ± 0.72	76.03 ± 0.34	78.54 ± 0.30	67.80 ± 0.12
			256	65.87 ± 0.40	51.76 ± 0.48	75.51 ± 0.34	79.04 ± 0.12	68.04 ± 0.13
	True	none		63.86 ± 0.59	46.89 ± 0.77	74.85 ± 0.67	78.66 ± 0.17	66.06 ± 0.13
		align	8	63.75 ± 0.50	52.05 ± 0.27	73.59 ± 0.40	75.90 ± 0.31	66.32 ± 0.23
			64	63.63 ± 0.29	53.01 ± 0.29	73.53 ± 0.29	75.84 ± 0.08	66.50 ± 0.16
			256	64.04 ± 0.28	53.18 ± 0.22	73.37 ± 0.45	75.74 ± 0.30	66.58 ± 0.13
		rand	8	66.13 ± 0.49	50.83 ± 0.66	76.01 ± 0.27	78.76 ± 0.25	67.93 ± 0.22
			64	65.67 ± 0.42	51.61 ± 0.64	75.54 ± 0.19	78.68 ± 0.18	67.87 ± 0.12
			256	66.37 ± 0.40	51.91 ± 0.15	75.81 ± 0.31	78.64 ± 0.13	68.18 ± 0.12
CORAL	False	none		**68.65 ± 0.56**	50.60 ± 0.27	**77.71 ± 0.21**	**80.36 ± 0.11**	**69.33 ± 0.18**
		align	8	66.18 ± 0.23	53.61 ± 0.08	75.20 ± 0.16	78.02 ± 0.26	68.25 ± 0.05
			64	66.81 ± 0.31	54.66 ± 0.27	75.01 ± 0.16	77.73 ± 0.27	68.55 ± 0.18
			256	65.50 ± 0.43	**55.06 ± 0.30**	74.65 ± 0.33	77.53 ± 0.28	68.19 ± 0.16
	True	none		**69.14 ± 0.69**	49.79 ± 0.25	**77.65 ± 0.12**	**79.94 ± 0.48**	**69.13 ± 0.16**
		align	8	66.63 ± 0.36	53.92 ± 0.19	75.30 ± 0.25	77.80 ± 0.26	68.41 ± 0.16
			64	66.72 ± 0.41	**55.31 ± 0.21**	74.34 ± 0.34	77.66 ± 0.28	68.51 ± 0.18
			256	66.17 ± 0.18	**55.24 ± 0.31**	74.79 ± 0.45	77.47 ± 0.21	68.42 ± 0.14

Table 7.5: Comparison of methods on the OfficeHome dataset with basic image augmentation. Average target domain accuracies (in %, ± SE) over 5 runs.

different distribution does not hold so well and the results must be interpreted with caution, as the domain shifts between these domains are not that large. This is also reflected in the results, as the general behavior of the methods is different from that seen on the PACS dataset.

The first thing we notice is that CORAL performs better on the OfficeHome dataset compared to PACS. It is able to improve on the ERM baseline in all domains and achieves the best average performance. This is again consistent with the results from the testbed benchmarks in Section 5.3 and confirms that CORAL does indeed improve generalization in the DG setting. CORAL achieves the best performance on the three similar domains (Art, Product and Real). However, it is outperformed by both stylization methods on Clipart, which is the only domain that is significantly different from the others.

Random stylization is able to slightly improve performance on the Clipart domain compared to the ERM baseline. However, performance on other domains is comparable to ERM and the differences are not significant. This confirms that random stylization is able to improve generalization under true domain shift. On average, random stylization achieves slightly better performance than ERM, but the improvement is marginal. Using a larger limit on the number of training styles $\mathcal{S}_{train}$ seems to help a bit on the Clipart domain. However, performance on other domains is not affected by increasing the number of styles.

Unlike the random stylization, the performance of the aligned stylization on the Art, Product, and Real domains drops significantly compared to the ERM baseline. It is therefore worse than both random stylization and ERM on these domains individually and on average. We believe this is caused by the fact that the domains are too similar, and aligned stylization can decrease performance on in-distribution data compared to ERM. However, performance on the Clipart domain is improved the most by aligned stylization. In fact, aligned stylization outperforms ERM, CORAL, and random stylization on Clipart, and in combination with CORAL and stylized

validation achieves the best overall performance on this domain. This result is most important to us because Clipart is the only domain that differs significantly from the rest, and therefore the only situation where true domain shift occurs in the OfficeHome dataset. Using larger values of $\mathcal{S}_{train}$ slightly improves performance on Clipart, but seems to decrease performance on the other domains. The effect of combining aligned stylization with CORAL seems to be additive in this case, with no additional benefit.

Stylized validation does not seem to help much in the case of the OfficeHome dataset. Surprisingly, it reduces performance when used only with the ERM and CORAL methods. However, it seems to help a little when used in combination with both stylization methods. This is similar to the behavior seen on the PACS dataset.

Chapter 8
Conclusion

In this work, we have explored the problem of domain generalization (DG) in the context of image classification. Despite being a well-studied problem, most of the existing DG methods are not yet able to achieve satisfactory results under domain shifts that are common in real-world applications. We have studied the existing DG methods and their performance, focusing on those methods that tackle the DG problem by generating additional training data. It has been shown that data augmentation methods, and especially methods using style transfer, are successful in improving performance on out-of-distribution (OOD) data. Therefore, we focused our attention on style transfer methods and their application in the DG setting.

Inspired by the success of style transfer methods in both DG [18, 34, 4] and other applications [7], we have proposed two novel methods that use style transfer to improve DG performance. The first method, aligned stylization, is an attempt to improve existing random stylization methods by simulating synthetic domains with style transfer in individual batches. The second method, stylized validation, utilizes style transfer for DG in a novel way by applying it in the validation and model selection phase instead of on the training data.

We have thoroughly tested the performance of the proposed methods on PACS [24] and OfficeHome [38] datasets and compared them with existing DG methods. We have also analyzed the effect of several hyperparameters on the performance of the proposed methods and performed a search for the optimal values of these hyperparameters. Furthermore, we studied the effect of combining the proposed methods and hypothesized that the combination of aligned stylization and the CORAL DG method [35] could further improve the performance.

We have found that the performance of stylization methods varies greatly across datasets and even domains. Therefore, choosing the right datasets for evaluation in the DG setting is crucial and results should be interpreted with caution. We have shown that the aligned stylization method can improve generalization under strong domain shifts, but that its performance suffers slightly on in-distribution data. Aligned stylization was also shown to perform slightly better than random stylization in the true DG setting, but the difference is only marginal. Furthermore, this improvement could be due to the fact that aligned stylization is more aggressive than random stylization, since the ratio of stylized to original images is higher.

The hypothesis that aligned stylization might perform better in combination with CORAL, an existing DG method, proved to be false. We found that the improvement from this combination is additive at best.

The style validation method seems to have no or even a small negative effect on OOD generalization when used only with the ERM [37] and CORAL methods. However, when combined with both random and aligned stylization methods, it slightly improves performance under domain shift. We suspect that this is related to the fact that the stylization methods, unlike the ERM and CORAL methods, saw other stylized images during training. However, we are not sure of the exact reason for this behavior.

Although the proposed methods improve generalization performance in the true DG setting, the improvement over existing methods is only marginal. There is also a large variance in the results across seeds, indicating that DG evaluation is highly unstable and that the results should be interpreted with caution. Despite these shortcomings, we believe that the proposed methods are a step in the right direction and that further research in this area is needed.

8.1 Future Work

Due to time constraints, we were not able to test the performance of the proposed methods on larger and more diverse datasets, such as DomainNet [30] or WILDS [21]. We believe that measuring the performance on larger and more complex datasets is crucial for obtaining more reliable results and would provide more insight into the generalization capabilities of the proposed methods.

In our work, we only performed a simple isolated search for the optimal values of some of the hyperparameters. We believe that a more thorough hyperparameter grid search would be beneficial for the performance of the proposed methods and could provide more insight into the effects of individual hyperparameters. However, due to the huge time requirements of such a search, we were not able to perform it.

Another interesting direction for future research is to investigate the effect of different style transfer methods used for stylization. In our work, we used only one style transfer method, but there are many other methods that could be used instead and may be more appropriate for the DG setting. The choice of style transfer method goes hand in hand with the choice of external style source dataset. We used the Painter by Numbers dataset [28] for our research, which contains only artistic images. This affects the possible styles that can be achieved by the style transfer, and thus the performance of the stylization methods. We believe that using a more diverse style source dataset could further improve the performance of the stylization methods.

While the stylization methods have been shown to improve performance on OOD data with strong domain shift, they unfortunately also suffer from performance drops on in-distribution data. This is a major problem for real-world applications where the models need to perform well on both in-distribution and OOD data. Therefore, we believe that more research is needed in this area.

Bibliography

[1] Md Zahangir Alom, Tarek M. Taha, Christopher Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Brian C Van Esesn, Abdul A S. Awwal, and Vijayan K. Asari. *The History Began from AlexNet: A Comprehensive Survey on Deep Learning Approaches*. 2018. arXiv: 1803.01164.

[2] Laith Alzubaidi, Jinglan Zhang, Amjad J Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, Mohammed A Fadhel, Muthana Al-Amidie, and Laith Farhan. "*Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions*". In: *Journal of big Data* (2021).

[3] Martin Arjovsky, Léon Bottou, Ishaan Gulrajani, and David Lopez-Paz. *Invariant Risk Minimization*. 2020. arXiv: 1907.02893.

[4] Francesco Cappio Borlino, Antonio D'Innocente, and Tatiana Tommasi. "*Rethinking Domain Generalization Baselines*". In: *2020 25th International Conference on Pattern Recognition (ICPR)*. 2021.

[5] Yaroslav Ganin and Victor Lempitsky. "*Unsupervised Domain Adaptation by Backpropagation*". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2015.

[6] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. "*Image Style Transfer Using Convolutional Neural Networks*". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[7] Robert Geirhos, Patricia Rubisch, Claudio Michaelis, Matthias Bethge, Felix A. Wichmann, and Wieland Brendel. *ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness*. 2022. arXiv: 1811.12231.

[8] Golnaz Ghiasi, Honglak Lee, Manjunath Kudlur, Vincent Dumoulin, and Jonathon Shlens. *Exploring the structure of a real-time, arbitrary neural artistic stylization network*. 2017. arXiv: 1705.06830.

[9] Muhammad Ghifary, W. Bastiaan Kleijn, Mengjie Zhang, and David Balduzzi. "*Domain Generalization for Object Recognition With Multi-Task Autoencoders*". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2015.

[10] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "*Generative Adversarial Networks*". In: *Communications of the ACM* (2020).

[11] Ishaan Gulrajani and David Lopez-Paz. *DomainBed*. https://github.com/facebookresearch/DomainBed. 2021.

[12] Ishaan Gulrajani and David Lopez-Paz. *In Search of Lost Domain Generalization*. 2020. arXiv: 2007.01434.

[13] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "*Deep Residual Learning for Image Recognition*". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[14] Xun Huang and Serge Belongie. "*Arbitrary Style Transfer in Real-Time With Adaptive Instance Normalization*". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[15] Maximilian Ilse, Jakub M. Tomczak, Christos Louizos, and Max Welling. "*DIVA: Domain Invariant Variational Autoencoders*". In: *Proceedings of the Third Conference on Medical Imaging with Deep Learning*. 2020.

[16] Naoto Inoue. *pytorch-AdaIN*. `https://github.com/naoto0804/pytorch-AdaIN`. 2018.

[17] Sergey Ioffe and Christian Szegedy. "*Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*". In: *Proceedings of the 32nd International Conference on Machine Learning (ICML)*. 2015.

[18] Philip T. Jackson, Amir Atapour-Abarghouei, Stephen Bonner, Toby P. Breckon, and Boguslaw Obara. "*Style Augmentation: Data Augmentation via Style Randomization*". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2019.

[19] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. "*Neural Style Transfer: A Review*". In: *IEEE Transactions on Visualization and Computer Graphics* (2020).

[20] Diederik P Kingma and Max Welling. *Auto-Encoding Variational Bayes*. 2022. arXiv: `1312.6114`.

[21] Pang Wei Koh et al. "*WILDS: A Benchmark of in-the-Wild Distribution Shifts*". In: *Proceedings of the 38th International Conference on Machine Learning (ICML)*. 2021.

[22] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "*Gradient-based learning applied to document recognition*". In: *Proceedings of the IEEE* (1998).

[23] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy Hospedales. "*Learning to Generalize: Meta-Learning for Domain Generalization*". In: *Proceedings of the AAAI Conference on Artificial Intelligence* (2018).

[24] Da Li, Yongxin Yang, Yi-Zhe Song, and Timothy M. Hospedales. "*Deeper, Broader and Artier Domain Generalization*". In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[25] Massimiliano Mancini, Samuel Rota Bulò, Barbara Caputo, and Elisa Ricci. "*Best Sources Forward: Domain Generalization through Source-Specific Nets*". In: *IEEE International Conference on Image Processing*. 2018.

[26] Claudio Michaelis, Benjamin Mitzkus, Robert Geirhos, Evgenia Rusak, Oliver Bringmann, Alexander S. Ecker, Matthias Bethge, and Wieland Brendel. *stylize-datasets*. `https://github.com/bethgelab/stylize-datasets`. 2019.

[27] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. "*Reading Digits in Natural Images with Unsupervised Feature Learning*". In: *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*. 2011.

[28] Kiri Nichol. *Painter by Numbers, WikiArt.org*. `https://www.kaggle.com/c/painter-by-numbers/`. 2016.

[29] Adam Paszke et al. "*PyTorch: An Imperative Style, High-Performance Deep Learning Library*". In: *Advances in Neural Information Processing Systems*. 2019.

[30] Xingchao Peng, Qinxun Bai, Xide Xia, Zijun Huang, Kate Saenko, and Bo Wang. "*Moment Matching for Multi-Source Domain Adaptation*". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019.

[31] Elan Rosenfeld, Pradeep Ravikumar, and Andrej Risteski. *The Risks of Invariant Risk Minimization*. 2021. arXiv: 2010.05761.

[32] Olga Russakovsky et al. "*ImageNet large scale visual recognition challenge*". In: *International journal of computer vision* (2015).

[33] Shiv Shankar, Vihari Piratla, Soumen Chakrabarti, Siddhartha Chaudhuri, Preethi Jyothi, and Sunita Sarawagi. *Generalizing Across Domains via Cross-Gradient Training*. 2018. arXiv: 1804.10745.

[34] Nathan Somavarapu, Chih-Yao Ma, and Zsolt Kira. *Frustratingly Simple Domain Generalization via Image Stylization*. 2020. arXiv: 2006.11207.

[35] Baochen Sun and Kate Saenko. "*Deep CORAL: Correlation Alignment for Deep Domain Adaptation*". In: *Computer Vision – ECCV 2016 Workshops*. 2016.

[36] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. *Instance Normalization: The Missing Ingredient for Fast Stylization*. 2017. arXiv: 1607.08022.

[37] V. Vapnik. "*Principles of Risk Minimization for Learning Theory*". In: *Advances in Neural Information Processing Systems*. 1991.

[38] Hemanth Venkateswara, Jose Eusebio, Shayok Chakraborty, and Sethuraman Panchanathan. "*Deep Hashing Network for Unsupervised Domain Adaptation*". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[39] Riccardo Volpi, Hongseok Namkoong, Ozan Sener, John C Duchi, Vittorio Murino, and Silvio Savarese. "*Generalizing to Unseen Domains via Adversarial Data Augmentation*". In: *Advances in Neural Information Processing Systems*. 2018.

[40] Jindong Wang, Cuiling Lan, Chang Liu, Yidong Ouyang, Tao Qin, Wang Lu, Yiqiang Chen, Wenjun Zeng, and Philip Yu. "*Generalizing to Unseen Domains: A Survey on Domain Generalization*". In: *IEEE Transactions on Knowledge and Data Engineering* (2022).

[41] Jindong Wang and Wang Lu. *DeepDG: Deep Domain Generalization Toolkit*. https://github.com/jindongwang/transferlearning/tree/master/code/DeepDG. 2022.

[42] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. *mixup: Beyond Empirical Risk Minimization*. 2018. arXiv: 1710.09412.

[43] Kaiyang Zhou, Ziwei Liu, Yu Qiao, Tao Xiang, and Chen Change Loy. "*Domain Generalization: A Survey*". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2023).

[44] Kaiyang Zhou, Yongxin Yang, Timothy Hospedales, and Tao Xiang. "*Learning to Generate Novel Domains for Domain Generalization*". In: *Computer Vision – ECCV 2020*. 2020.

Appendix A
Results without data augmentation

Tables A.1 and A.2 contain a comparison of methods on the PACS [24] and Office-Home [38] datasets, respectively, without basic image augmentation. Otherwise, the experiments follow the same setup as described in Chapter 7. The results are averaged over 5 runs with different seeds and the standard error is reported together with the mean. The best results in each column are highlighted in bold, taking into account the standard errors.

PACS. As expected, the results without basic image augmentation are worse compared to ones with image augmentation. This is most noticeable on the Cartoon and Sketch domains. The overall behavior of the methods is similar to the results seen in Table 7.4.

When basic image augmentation is disabled, the aligned stylization method seems to outperform random stylization more significantly. On average, the aligned stylization achieves the best results. Both stylization methods are able to outperform the ERM [37] baseline with image augmentation enabled (see Table 7.4), given that the limit on the number of training styles $\mathcal{S}_{train}$ is high enough. This means that style transfer augmentation is more suitable for the DG setting than the standard image augmentation. However, as seen in Table 7.4, using both together gives the best results.

Stylized validation is able to improve performance only slightly on Art, Photo and Sketch, but surprisingly suffers performance drop on the Cartoon domain. This is different from the results with image augmentation enabled, where stylized validation

Alg.	StyVal	Styling	$\mathcal{S}_{train}$	Art	Cartoon	Photo	Sketch	Average
ERM	False	none		83.60 ± 1.44	81.66 ± 0.82	96.12 ± 0.40	66.62 ± 2.88	82.00 ± 0.76
		align	8	87.23 ± 0.73	**82.60 ± 0.52**	95.32 ± 0.28	83.96 ± 0.85	87.28 ± 0.36
			64	88.37 ± 0.40	**83.15 ± 0.55**	95.71 ± 0.11	85.91 ± 0.60	88.29 ± 0.23
			256	88.60 ± 0.45	**83.40 ± 0.61**	95.80 ± 0.19	86.17 ± 1.06	**88.49 ± 0.39**
		rand	8	86.88 ± 0.55	82.20 ± 0.61	95.96 ± 0.18	75.37 ± 3.95	85.10 ± 0.99
			64	86.89 ± 1.13	**82.65 ± 0.67**	95.99 ± 0.24	84.03 ± 1.82	87.39 ± 0.44
			256	88.20 ± 0.59	81.59 ± 0.59	96.41 ± 0.13	84.68 ± 1.88	87.72 ± 0.58
	True	none		84.35 ± 0.84	79.88 ± 0.92	**96.35 ± 0.29**	67.84 ± 1.81	82.10 ± 0.42
		align	8	88.41 ± 0.48	**83.20 ± 0.15**	95.77 ± 0.23	82.35 ± 1.65	87.43 ± 0.47
			64	88.32 ± 0.39	82.60 ± 0.43	95.54 ± 0.11	86.33 ± 0.43	88.20 ± 0.16
			256	**89.00 ± 0.46**	**82.42 ± 0.66**	95.99 ± 0.16	**88.04 ± 0.24**	**88.86 ± 0.13**
		rand	8	87.28 ± 0.49	82.51 ± 0.28	**96.46 ± 0.15**	80.74 ± 1.50	86.75 ± 0.36
			64	87.53 ± 0.42	**83.04 ± 0.69**	96.13 ± 0.19	83.73 ± 0.95	87.61 ± 0.24
			256	**88.49 ± 0.65**	**82.54 ± 0.74**	95.81 ± 0.16	84.30 ± 0.40	87.78 ± 0.13
CORAL	False	none		86.23 ± 0.99	**82.91 ± 0.87**	**96.92 ± 0.34**	70.83 ± 1.92	84.22 ± 0.53
		align	8	88.05 ± 0.48	81.89 ± 0.88	95.44 ± 0.15	82.76 ± 0.52	87.03 ± 0.14
			64	87.90 ± 0.52	**83.42 ± 0.45**	95.58 ± 0.08	84.54 ± 0.58	87.86 ± 0.15
			256	87.68 ± 0.45	81.96 ± 0.59	95.80 ± 0.10	86.56 ± 1.15	88.00 ± 0.29
	True	none		87.07 ± 0.35	80.73 ± 0.87	**96.46 ± 0.39**	71.36 ± 2.01	83.91 ± 0.80
		align	8	88.00 ± 0.36	82.12 ± 0.39	95.59 ± 0.19	83.44 ± 0.93	87.29 ± 0.25
			64	88.43 ± 0.32	**82.86 ± 0.47**	95.70 ± 0.10	86.79 ± 0.19	88.44 ± 0.19
			256	**89.56 ± 0.50**	82.35 ± 0.35	95.77 ± 0.16	**87.57 ± 0.61**	**88.81 ± 0.21**

Table A.1: Comparison of methods on the PACS dataset without basic image augmentation. Average target domain accuracies (in %, ± SE) over 5 runs.

had very little impact on the final accuracy of ERM and CORAL [35] methods.

OfficeHome. The difference between the results with and without basic image enhancement on the OfficeHome [38] dataset is marginal. This may be due to the fact that classification on the OfficeHome dataset is more difficult, and therefore simple image augmentation does not have a large impact.

The overall behavior of the methods is almost identical to that seen in Table 7.5. CORAL seems to achieve the best results and outperforms both stylization methods on Art, Product, and Real domains. However, aligned stylization is able to significantly outperform CORAL on the Clipart domain, similar to the results with image augmentation.

Alg.	StyVal	Styling	$\mathcal{S}_{train}$	Art	Clipart	Product	Real	Average
ERM	False	none		64.29 ± 0.49	48.04 ± 0.54	75.77 ± 0.14	79.15 ± 0.11	66.81 ± 0.20
		align	8	63.12 ± 0.37	51.64 ± 0.22	72.89 ± 0.31	76.44 ± 0.17	66.02 ± 0.07
			64	62.97 ± 0.65	52.96 ± 0.33	72.61 ± 0.29	76.23 ± 0.26	66.19 ± 0.19
			256	62.60 ± 0.52	53.43 ± 0.50	72.55 ± 0.23	75.59 ± 0.14	66.04 ± 0.21
		rand	8	65.43 ± 0.32	50.10 ± 0.38	75.85 ± 0.40	78.56 ± 0.23	67.49 ± 0.23
			64	66.04 ± 0.60	50.99 ± 0.46	75.71 ± 0.30	78.15 ± 0.11	67.72 ± 0.20
			256	65.32 ± 0.20	51.62 ± 0.53	75.78 ± 0.23	78.44 ± 0.34	67.79 ± 0.16
	True	none		63.92 ± 0.71	46.52 ± 1.06	75.63 ± 0.35	78.01 ± 0.65	66.02 ± 0.43
		align	8	63.34 ± 0.63	51.87 ± 0.35	73.21 ± 0.46	76.38 ± 0.23	66.20 ± 0.24
			64	63.18 ± 0.59	53.23 ± 0.24	72.66 ± 0.21	75.28 ± 0.27	66.09 ± 0.20
			256	62.95 ± 0.50	53.47 ± 0.23	72.80 ± 0.52	75.48 ± 0.46	66.18 ± 0.21
		rand	8	65.28 ± 0.61	49.68 ± 0.53	75.95 ± 0.34	78.69 ± 0.12	67.40 ± 0.19
			64	66.17 ± 0.22	51.88 ± 0.24	75.95 ± 0.20	78.61 ± 0.26	68.15 ± 0.09
			256	65.35 ± 0.55	52.43 ± 0.13	75.38 ± 0.33	78.16 ± 0.13	67.83 ± 0.13
CORAL	False	none		**67.75 ± 0.37**	49.44 ± 0.35	**77.37 ± 0.47**	**81.04 ± 0.14**	**68.90 ± 0.10**
		align	8	65.56 ± 0.39	54.24 ± 0.26	74.84 ± 0.25	77.74 ± 0.12	68.09 ± 0.08
			64	65.22 ± 0.48	55.02 ± 0.28	74.24 ± 0.28	77.49 ± 0.24	67.99 ± 0.11
			256	65.46 ± 0.22	**55.22 ± 0.35**	74.34 ± 0.16	77.21 ± 0.30	68.06 ± 0.17
	True	none		**66.35 ± 1.07**	48.50 ± 0.57	**77.74 ± 0.15**	**80.99 ± 0.23**	68.40 ± 0.25
		align	8	65.47 ± 0.18	54.69 ± 0.40	75.22 ± 0.44	77.51 ± 0.25	68.22 ± 0.15
			64	65.45 ± 0.27	**55.15 ± 0.45**	74.33 ± 0.24	77.18 ± 0.22	68.03 ± 0.18
			256	65.19 ± 0.22	**55.77 ± 0.30**	74.30 ± 0.41	76.92 ± 0.24	68.04 ± 0.15

Table A.2: Comparison of methods on the OfficeHome dataset without basic image augmentation. Average target domain accuracies (in %, ± SE) over 5 runs.

Appendix B
Additional experiments

Here, we present additional experiments that we performed during the development of our method. We include them here for completeness. These experiments were performed in the DomainBed framework [11] with default hyperparameters.

Digits. Before we settled on using style transfer as our method for generating synthetic data, we explored other methods. Our hypothesis was that the performance of models on ODD data could be improved by generating synthetic domains from an existing ones. To test this hypothesis, we used the digits datasets. MNIST was used as the source domain and SYN was used as the target domain. MNIST-M is a dataset that was created by blending a randomly cropped photographs into the images from MNIST [5]. Therefore, we used it in place of the synthetic domain.

Table B.1 shows the results of this experiment. The results are averaged over 5 runs with different seeds and we also report the standard error. The models were trained both with and without image augmentation.

We can see that adding synthetic domain as a new source domain significantly improves the performance of the model. Adding MNIST-M as a source domain even outperforms basic image augmentation. However, when we use image augmentation together with synthetic domain, the performance is even better.

Augment	Source domains	SYN
False	MNIST	18.6 ± 1.5
	MNIST, MNIST-M	48.8 ± 1.8
True	MNIST	39.2 ± 1.2
	MNIST, MNIST-M	**58.9 ± 1.4**

Table B.1: Results of experiment on digits datasets (in %, ± SE) over 5 runs.

StylizedImageNet-10. We also performed an experiment on a dataset similar to the StylizedImageNet proposed by Geirhos et al. [7]. We used a subset of ImageNet [32] that contains only 10 classes because the original dataset is too large. We created 10 prestyled domains by applying different styles to the ImageNet images using the AdaIN style transfer method [14].

2 of the prestyled domains were used for validation and another 2 were used as target domains. The remaining 6 domains were used for training as synthetic source domains along with the original domain. However, we gradually increased the number of source domains used for training. Again, we trained the models with and without image augmentation. Since all domains contained the same images, we used split ratios of 0.7, 0.15, and 0.15 for training, validation, and testing, respectively, to avoid data leakage. Table B.2 shows the results of this experiment. For completeness, we also report the performance on the original test split.

We can see that the performance of the model increases significantly with the number

of source domains used for training. The performance on the original test split also improves, but not as much as on the synthetic domains. We can also see that the performance on the two synthetic target domains is slightly different, even though they were both created from the same source domain. This means that the final accuracy depends on the style image used for style transfer. Surprisingly, basic image augmentation seems to have a negative effect on the performance of the model when combined with larger number of synthetic domains.

Augment	$\mathcal{S}_{train}$	**Original**	**Style A**	**Style B**	**Avg**
False	0	97.8 ± 0.3	58.1 ± 1.0	56.9 ± 3.1	70.9
	1	98.6 ± 0.1	79.4 ± 0.8	83.7 ± 1.5	87.2
	3	**98.8 ± 0.1**	87.1 ± 0.8	88.1 ± 1.1	91.3
	6	**99.0 ± 0.1**	**91.7 ± 0.2**	**91.9 ± 0.4**	**94.2**
True	0	98.1 ± 0.2	62.9 ± 2.0	57.2 ± 1.9	72.7
	1	98.4 ± 0.1	83.2 ± 1.4	86.5 ± 0.5	89.4
	3	98.6 ± 0.1	88.3 ± 0.4	88.1 ± 0.7	91.6
	6	98.7 ± 0.1	**91.2 ± 0.6**	**91.3 ± 0.3**	93.7

Table B.2: Results of experiment on StylizedImageNet-10 dataset (in %, ± SE) over 5 runs.

Appendix C
Source code attachment

Figure C.1 shows the directory structure of the source code attachment. A more detailed description of the `StylizedDomainGeneralization` framework is provided in the `README.md` file.

```
./
└── StylizedDomainGeneralization/ - root directory of the framework
    ├── README.md - README file, contains a detailed description of the framework,
    │   installation instructions and instructions on how to replicate the experiments
    ├── configs/ - directory containing YAML configuration files
    │   └── default.yaml - default configuration file
    ├── data/ - directory containing symbolic links to datasets
    ├── experiments/ - directory containing definitions of experiments
    ├── models/ - directory containing pretrained AdaIN style transfer models
    ├── requirements.txt - requirements file for pip package manager
    ├── run.batch - batch file specification for SLURM jobs submission
    ├── stylizeddomaingeneralization/ - source code directory
    │   ├── datasets.py - file containing dataset classes
    │   ├── model.py - file containing model classes
    │   ├── scripts/ - directory containing scripts
    │   │   ├── summary.py - script for summarizing results
    │   │   ├── train.py - script for training a model
    │   │   └── ⋮
    │   └── ⋮
    └── submit.sh - script that transforms experiment to individual SLURM jobs
```

Figure C.1: Source code attachment directory structure.