DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Transfer Learning between Synthetic and Real Data

Manuel Dahnert

DEPARTMENT OF INFORMATICS

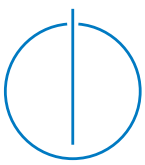
TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics: Games Engineering

Transfer Learning between Synthetic and Real Data

Lerntransfer zwischen Synthetischen und Realen Daten

Author:	Manuel Dahnert
Supervisor:	Prof. Dr. Matthias Nießner
Advisor:	Prof. Dr. Matthias Nießner
Submission Date:	15.05.2018



I confirm that this master's thesis in informatics: games engineering is my own work and I have documented all sources and material used.

Munich, 15.05.2018 Manuel Dahnert

Acknowledgments

The journey of this Master's thesis and my entire studies would not have been possible without the endless support of many people.

First of all, I would like to thank Matthias Nießner for being a great supervisor for the past 6 months. The topic of this thesis introduced me to the exciting and fascinating field of Deep Learning and considerably expanded my horizon. His broad knowledge and structured guidance helped me to avoid pitfalls and allowed me to focus on the essential parts of the topic. His exceptional commitment and positive attitude always encouraged and motivated me throughout the time of this thesis.

I would also like to thank Andreas Rössler for setting up and configuring the Deep Learning training server. An additional thank goes to the other students of the "Computer Graphics and Visualization" lab, who raised interesting discussions and shared practical tips and information.

Most importantly, I am very grateful for all the support of my friends and family, who accompanied me through every stage of my studies.

Abstract

The negative effects of domain discrepancy between the synthetic and the real-world domain, also known as "Reality Gap", prevents leveraging synthetic data as an effective source of virtually unlimited training data for supervised Deep Learning.

In this Master's Thesis we present Early-Two-Stream Domain Adaptation (ETS-DA). The key idea is to align the synthetic and real-world domain early on within a deep neural network. Our approach separates the first few layers of the network into individual streams, which capture domain-specific low-level features. These low-level features then establish a domain-invariant representation of higher-level features.

We show that our ETS-DA approach can not only significantly reduce the "Reality Gap" by individually capturing the low-level domain-specifics, but we also demonstrate that the established domain-invariant feature space can be expanded to recognize classes of images in the real-world domain for which the network has only seen synthetic training data.

Contents

1. Introduction

In recent years, Deep Learning has significantly improved the performance of many computer vision tasks. With the availability of large-scale, annotated datasets, such as ImangeNet [9], and deeper and improved network architectures, e.g. AlexNet [23], VGG-16 [42] and ResNet [17], the error rate of the classification task in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) has surpassed human accuracy [36].

However, despite the huge success in recent years, supervised Deep Learning suffers from two major problems: The necessity of large, annotated datasets and the negative effects of domain discrepancy.

Deep neural networks consist of millions of parameters and thus require huge amounts of diverse, labeled data to generalize from training data to unseen test data. However, collecting this data can be infeasible, e.g. in safety-critical scenarios such as car crashes. Additionally, to perform supervised learning, the collected data needs to be verified and labeled. This process is mostly done manually by employing human workers, which is costly and time-consuming. To overcome the process of manual data collection and annotation, computer simulations has been widely used to produce annotated synthetic data, which then can used to train a deep neural network.

The second problem of Deep Learning is the assumption, that training and test data are always drawn from the same distribution. However, in reality this assumption often does not hold. Even after extensive training with diverse data, networks still fail to generalize across domains. This results in a degradation of performance during test time [8]. The problem of domain discrepancy also appears between the synthetic and real domain and is known as “Reality Gap”. This issue prevents leveraging synthetic

data as an effective source of training data for supervised Deep Learning. Transferring knowledge from on domain to another has a long history in Machine Learning. Recent work tries to address the issue of domain discrepancy directly within deep neural networks [25, 51, 26]. These methods are later described in section 2.4.

[55] showed that earlier layers in deep convolutional neural network extract low-level features, edges and color blobs, whereas later layers represent more abstract, higher-level concepts. While it has been thought, that the lower-level features are generic, [2] demonstrated that even within the first layer of the network, the effects of domain discrepancy become evident.

To this end, we propose Early-Two-Stream Domain Adaptation (ETS-DA), a network-agnostic approach that addresses domain discrepancy between two domains. The key idea is to separate the first few layers of a given network architecture into individual streams, one per domain, while the remaining part of the network is shared between both domains. The streams individual capture the domain-specific, low-level features, which then build up a shared domain-invariant representation of higher-level features. Given the advantages of synthetic data, we focus on the adaptation between synthetic and real-world domain.

Together with ETS-DA we propose a two-staged training strategy: In the first phase we align both domains leveraging labeled data from both domains and establish a domain-invariant feature space. In the second phase, we expand this feature space by introducing additional image classes, which however, are only given by the synthetic domain.

Our results show, that our ETS-DA approach reduces the negative effects of the "Reality Gap" and that our network is able to recognize classes of images in the real-world domain for which the network has only seen synthetic training data.

2. Related Work

We briefly review the significant development of deep learning in the field of computer vision and show, how synthetic data is currently used to train deep networks. Then, we recap the concepts of Transfer Learning and Domain Adaptation and review existing approaches.

2.1. Deep neural networks

The reason of the success of deep learning in computer vision can be summarized by three factors: Annotated, large-scale datasets, deeper neural network architecture with higher capacities and the ability to train these deep networks with the computational power of GPUs.

In 2012, Krizhevsky et al. submitted a 8-layer convolutional neural network, AlexNet [23], to the ImageNet Large-Scale-Visual-Recognition Challenge (ILSVRC) [36] outperforming other state-of-the-art approaches, which used hand-crafted features, by a large margin. From this point on, deeper and more sophisticated architectures [42, 48, 17] further reduced the error rate in the 1000-class ILSVRC classification task from 16.4% to 3.57% and thus surpassed human accuracy of 5.1% [36, 18].

These deep networks with millions of parameters require large amounts of annotated data to generalize from training data to unseen test dat. One prominent dataset is ImageNet. It consists of more than 14.1 million annotated images within 22 000 categories [9]. Further, the authors in [55, 10] demonstrated that networks trained on ImageNet learn generic representations, which even transfer to other tasks.

2.2. Synthetic Data

The necessity of large, annotated datasets to train neural networks can be prohibitive in areas, where it is infeasible to collect the required amount of data and to obtain accurate ground-truth information [35, 4]. Computer simulations can produce virtual unlimited data and simultaneously provide the corresponding annotation with negligible human effort.

Consequently, synthetic data has been used in many areas to train neural networks: [43, 16, 56, 28] use synthetically generated indoor-scenes for the task of scene understanding. [35, 34, 20, 33, 30, 39] propose to exploit modern, photo-realistic computer games and existing 3D game engines as a rich source of high-quality, annotated data. Richter et al. use this data to perform semantic segmentation of outdoor scenes [35], whereas Johnson-Roberson et al. train a network to detect vehicles [20].

Chang et al. present ShapeNet, which is the result of the efforts to aggregate a large-scale, annotated repository of 3D models [7]. The vast amount of 220 000 models in 3 135 categories makes it possible to synthesize highly diverse and annotated data for various tasks, see Figure 2.1 for examples. Su et al. and Li et al. render images of ShapeNet objects to train neural networks, which predict object poses [45] and similarities between objects [24], respectively.

Figure 2.1.: Example collection of ShapeNet models from different categories.

Besides being a source for synthetic data, computer simulations can also be used as a safe and encapsulated environment. This becomes evident in the case of deep reinforcement learning, where expensive hardware interacts with its surroundings and might get damaged or damage others. [38, 22] present test platforms for motion planning of artificial agents within indoor-scenes. [4, 49] train robots in virtual environments

and employ the learned model to a real robot. [11, 40, 30, 53] are simulators for autonomous vehicles and are specially designed for incorporation with deep learning and reinforcement learning. Additionally, these simulations can be easily distributed to run in parallel and to accelerate training.

2.3. Transfer Learning

As we have seen, synthetic data can be used to train deep neural networks in areas where labeled data is limited. Also, more realistic data in terms of visual appearance and physical fidelity is beneficial compared to unrealistic models [56, 49].

However, even after extensive training on synthetic data, these models still do not generalize well to real data. This degradation of performance does not only appear between synthetic and real data, but also between other domains. One reason for this is domain discrepancy between training and test data, i.e. both are drawn from different marginal distributions, and the network fails to generalize across this gap [3, 37].

Transferring knowledge between domains and thus overcoming this gap has a long history in machine learning and has been subject to a lot of research. More recently, this problem has attracted plenty of attention in the field of deep learning [8]. To formalize this problem and to give it a theoretical basis, we recite the definitions of Transfer Learning and Domain Adaptation. For comprehensive overviews refer to [31] and [8], respectively.

Definition Following the definition of [31], Transfer Learning uses the notion of domains $\mathcal{D}$, each with a d-dimensional feature space $\mathcal{X} \in R^d$ and a marginal distribution $P(X)$. Further, it defines tasks $\mathcal{T}$ within a label space $\mathcal{Y}$ with a conditional probability distribution $P(Y|X)$, whereas X and Y denote random variables. Additionally, we define the sample set $X = \{x_1, x_2, \ldots, x_n\}$ of $\mathcal{X}$, as well as corresponding labels $Y = \{y_1, y_2, \ldots, y_n\}$ from $\mathcal{Y}$.

Using this notion, we define a source domain $\mathcal{D}^s = \{\mathcal{X}^s, P(X^s)\}$ with a task $\mathcal{T}^s = \{\mathcal{Y}^s, P(Y^s|X^s)\}$ and a target domain $\mathcal{D}^t = \{\mathcal{X}^t, P(X^t)\}$ with a task $\mathcal{T}^t = \{\mathcal{Y}^t, P(Y^t|X^t)\}$.

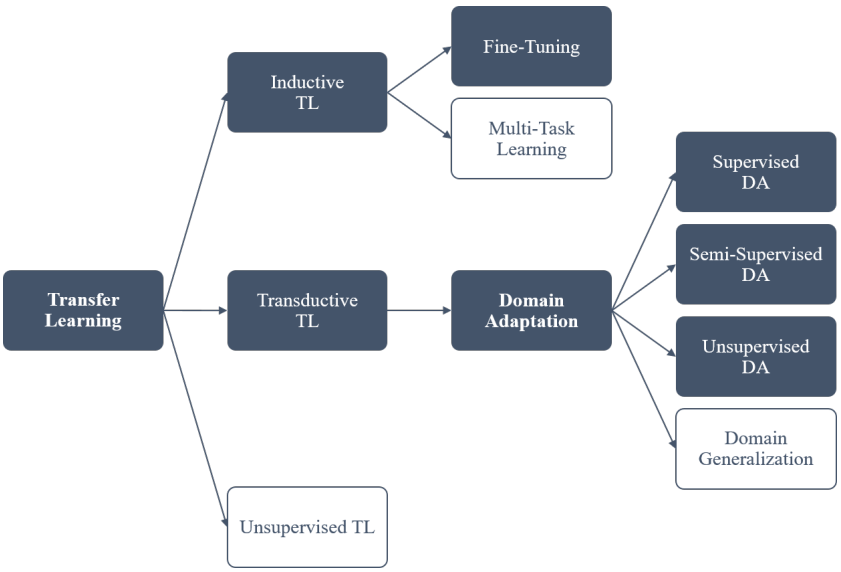


Figure 2.2.: Transfer Learning taxonomy. White boxes correspond to adjacent concepts, which are not considered in this work. Adapted from [31].

Further, we distinguish three different categories of scenarios: Inductive, transductive and unsupervised Transfer Learning.

Inductive Transfer Learning In inductive Transfer Learning the target task is different from, but related to the source tasks, whereas the source and target domains can also be the same. Further it requires some labeled samples in the target domain so that we can "induce" from the source to the target domain. We refer to this scenario in the Expansion step, cf. section 6.2, during the training of our network.

For visual applications convolutional neural networks are usually pre-trained on large-scale databases, such as ImageNet. These pre-trained models have shown to learn generic feature representations, which can be reused for other tasks and domains and thus prevent the burden of training the network from scratch. Adapting the knowledge from the pre-trained model to the new domain or task is mostly done via fine-tuning [55]. During fine-tuning the network is trained with a labeled set of samples from the target domain $\mathcal{D}^t$ and a small learning rate. [43, 56, 39] use small, real world datasets

to transfer from the trained synthetic domain to the real domain.

One problem of fine-tuning is catastrophic forgetting: While transitioning to the new target, the network "forgets" the knowledge of the source domain or tasks. Goodfellow et al. employ the standard dropout algorithm [44] to alleviate the negative effects of this forgetting [13]. Jung et al. counteract this problem by coupling the source and target networks with an additional Euclidean loss function [21].

During our Expansion step the target domain is a superset of the source domain. Therefore, we take the learned weights of the classification layer from $\mathcal{D}^s$ and $\mathcal{T}^s$ and reuse these weights for $\mathcal{D}^t$ and $\mathcal{T}^t$.

Transductive Transfer Learning Transductive Transfer Learning refers to the situation, when the source and target tasks are the same, but the source and target probability distributions are different, i.e. $\mathcal{T}^s = \mathcal{T}^t, P(X^s) \neq P(X^t)$. In the following we refer to this category as Domain Adaptation, which will be further presented in section 2.4.

Unsupervised Transfer Learning Finally, unsupervised Transfer Learning assumes neither labels in source domain nor in the target domain and the tasks are different, but related. Unsupervised Transfer Learning is usually subject to unsupervised machine learning methods, such as clustering or dimensionality reduction.

2.4. Domain Adaptation

With the advances of deep neural networks in Computer Vision, Domain Adaptation for visual applications currently receives a lot of attention [8]. Domain Adaptation can be categorized as transductive Transfer Learning: The task of both domains is identical, $\mathcal{T} = \mathcal{T}^s = \mathcal{T}^t$, whereas the probability distributions differ between the domains, i.e. $P(X^s) \neq P(X^t)$.

According to Pan and Yang transductive Transfer Learning assumes labeled data in the source domain, but no labels in the target domain [31]. However this second

assumptions is usually relaxed so that labeled target data can also be employed. This allows us to define supervised, semi-supervised and unsupervised Domain Adaptation.

Definitions For the supervised Domain Adaptation case, we assume full ground-truth information in both domains, i.e. each data sample is attached with a label. In a semi-supervised scenario, we split the target domain into two sets $\mathcal{D}^t = \mathcal{D}_1^t \cup \mathcal{D}_2^t$. $\mathcal{D}_1^t$ is provided with ground-truth information, while $\mathcal{D}_2^t$ is not. Additional it is assumed that the amount of unlabelled data is considerably larger then the labeled samples, i.e. $|\mathcal{D}_1^t| \ll |\mathcal{D}_2^t|$. Unsupervised Domain Adaptation deals with the scenario, where no labels are available in the target domain.

We mainly focus on Domain Adaptation approaches, which incorporates the adaptation problem statement directly within a deep neural network. However, we also want to give a brief overview of some approaches that were proposed before the advances of Deep Learning.

Shallow Domain Adaptation Before the success of Deep Learning, a series of "shallow" Domain Adaptation methods addresses the effects of domain discrepancy.

Re-weighting approaches try to align the probability distributions of the domains by assigning weighting factors to each source sample. One widely used measure is the Maximum Mean Discrepancy (MMD) metric [15]. It embeds the means of both probability distributions within a reproducing Hilbert space and calculates the distance between the embedded means. Transfer Component Analysis (TCA) tries to find latent features representations between the domains [32]. The data distributions of the domains then lie in proximity in the subspace spanned by these latent features. The authors of [46] propose correlation alignment (CORAL), which aligns the second-order statistics, i.e. the covariance, of both domains and thus mitigates the negative effects of domain shift.

These approaches do not incorporate label information and therefore, they can be categorized as unsupervised Domain Adaptation. For a more in-depth comparison of

existing shallow Domain Adaptation techniques, refer to [8].

Deep Domain Adaptation Given the hierarchical feature representations of deep convolutional neural networks, the goal of many deep Domain Adaptation approaches is to find a domain-invariant feature space within these representations. This domain-invariant feature space can then be deployed to other domains and other tasks.

[25] assumes domain-invariance within the first few layers of the network, since these layers extract generic, low-level features. However, [2] demonstrates that these first layers are already affected by domain shift. They apply a filter-reconstruction method to the first layers of a network to "repair" filters, which contribute to the domain discrepancy.

[47] extend the CORAL method to deep networks, which aligns the second-order statistics of the layer activations between the domains. [51, 52, 26] employ additional domain adverserial losses to the training objective. They achieve domain-invariance by confusing a domain discriminator, while simultaneously training on the original task.

[5] use three encoder networks to split the representation spaces into two domain-specific (private) and domain-invariant (shared) sub-spaces. They train this network architecture by optimizing on two difference losses and on one similarity loss.

Reality Gap One particular kind of Domain Shift is called Reality Gap and describes the domain discrepancy between synthetic and real-world data. Bridging this gap is of special interest, since synthetic data has many advantages over real-world data, see section 2.2.

[50] proposes Domain Randomization. They argue that after training a network with highly diverse synthetic data, the real world appears just as another variation. [4, 54] train Generative Adversarial Networks [14] to generate synthetic images, such that they are indistinguishable from real-world images. Rather than refer to the visual differences between synthetic and real-world data, [49] look at the Reality Gap from a physical perspective, which is especially important in the field of robotics.

3. Method Overview

Our ETS-DA approach takes 2D RGB images from two domains, synthetic and real-world, as input. The data in the synthetic domain are images of rendered 3D objects. The generation process follows the idea of the overfit-resistent method of [45] and the labels are automatically generated during the render process. The images of the real-world domain are taken from ImageNet, which contains manually annotated ground-truth information.

To achieve comparability and Domain Adaptation on a feature-based level, the images from both domains have to depict the same set of objects. Hence, the 3D models for the synthetic images and the real-world images correspond to the same set of classes, see 4 for more details.

Given this correspondence between the set of classes, the images from both domains usually exhibit a high degree of similarity in terms of their structural composition of the depicted object. For instance, a car is composed of a chassis and four wheels, regardless of whether it a rendered 3D model or real-world photography.

While being similar on a high level, synthetic images can be visually very different from real images. These differences can, for example, origin from an approximative lighting model used in the computer simulation or from artifacts introduced by the rendering process.

The main idea of ETS-DA is to exploit this high-level similarity between both domains, but also capture the domain-specific differences. It has been shown that first few layers of deep neural networks extract low-level features, such as edges or color information, the later layers represent higher-level concepts [55, 10]. Therefore, ETS-DA separates the first few layers of a deep neural network into individual streams, which extract

domain-specific low-level features. The remaining part of the network is shared between the domains and captures the high-level features. Whereas the idea of our method is general applicable to any given deep neural network architecture, the specific architecture evaluated in this thesis is described in chapter 5.

In this work, we focus on task of image classification, which is one of the fundamental tasks in computer vision. In our ETS-DA architecture, the images are fed to one of two individual input streams depending on the domain of the image. The network produces for each image a vector of scores representing the probabilities that the given image displays a specific class.

Our network trains to predict the correct ground-truth label of the given image.

The training process of our method consists of two phases: The first phase corresponds to a supervised domain adaptation scenario, in which labels in both domains are available. In this phase, the network is fed alternatingly with synthetic and real images. Having images from both domains, the network is trained to capture the domain-dependent specifics within the individual streams and the domain-invariant feature space of higher-level features.

Having this separation between individual domain-specific features and a shared domain-invariant feature space, the later one is expandible given only one of both domains. Inferring from the advantages of the use of synthetic data, we focus on expanding the domain-invariant feature space by adding additional object categories to the base classes from the synthetic domain.

In the second stage we face a combination of inductive transfer learning and unsupervised domain adaptation: We introduce novel object classes for which, however, labels are only given in the synthetic domain.

The final goal after the second training phase is, that at test time the network correctly predicts the labels of the images from the real domain, for which the network has not seen labeled images of the additional classes.

The proposed two-staged training process let us explore different options for each stage. In the supervised domain adaptation step, we compare different combinations of source

and target domain settings and the possibilities of freezing the weights of different parts of the network during optimization to enforce stronger domain adaptation.

In the second stage, we examine the effects of reusing the previously trained classifier and re-weighting the importance of the novel classes.

4. Data

To train and to evaluate our network, we prepare datasets in both domains. For the real domain, we use the widely used ImageNet [9] as data source. For the synthetic domain, we use models from ShapeNet [7] and generate images from them as described in section 4.3. Both data sources are organized according to the WordNet structure [29].

4.1. Dataset structure

WordNet is a lexical database of English words, which is organized around synonyms of words, so called synonym sets or synsets. Each synset describes a distinct concept of related words. For example “car,auto,automobile,machine,motorcar” describe one set of synonyms and refer to the concept of “a motor vehicle with four wheels”. Furthermore, synsets are hierarchically organized in so called hyponyms and form a parent-child relation. Figure 4.1 shows an example of two synsets as hyponyms with their child-synsets.

The training of our network consists of two stages. In the first stage, we train the network with 10 objects classes, whereas in the second step we introduce 7 additional classes to end up with 17 classes, see chapter 6.

Each class correspond to one synset in the WordNet structure and is used as hyponym, thus including all its child-synsets. The selected synsets with the specific number of images per class and per domain can be seen in Table 4.3.

Additionally, we split each dataset into three sets, referred to as training, validation and test set. The training set consists of 80% of the images in each dataset, whereas

- airplane, aeroplane, plane
  - airliner
  - amphibian, amphibious aircraft
  - biplane
  - bomber
  - delta wing
  - fighter, fighter aircraft, attack aircraft
  - hangar queen
  - jet, jet plane, jet-propelled plane
  - ...
- car, auto, automobile, machine, motorcar
  - ambulance
    - funny wagon
  - beach wagon, station wagon, wagon, ...
    - shooting brake
  - bus, jalopy, heap
  - cab, hack, taxi, taxicab
    - gypsy cab
    - minicab
  - ...

Figure 4.1.: Examples of two synsets as hyponyms with their respective child-synsets.

validation and test set each consist of 10% of the images. The training and validation sets is used to train the network and to evaluate performance of the network on unseen data. The test set is only used at the very end to evaluate the final performance.

	Real-world	**Synthetic**
10 classes	$Real_{10}$	$Synth_{10}$
17 classes	$Real_{17}$	$Synth_{17}$

Table 4.1.: Names of the datasets in each domain

4.2. Real-world data

Covering the real world, ImageNet is currently one of the most ambitious and most popular data source. It is a large-scale, annotated dataset, containing more than 14 million real-world images and covering more than 21,000 synsets [9]. ImageNet can be freely used for non-commercial research purposes.

To generate the real-world datasets $Real_{10}$ and $Real_{17}$, we download the specific synsets including their child synsets from ImageNet. $Real_{10}$ contains around 110 000 images, $Real_{17}$ contains around 234 000 images.

Figure 4.2.: Examples of rendered images

4.3. Synthetic data

ShapeNet is the result of the efforts to combine existing 3D model repositories into a single, large-scale repository. ShapeNet contains more than 3 million models, where 220 000 have been classified into around 3 100 WordNet synsets. ShapeNetCore, a subset of ShapeNet, consists of around 51 300 single 3D models, covering 55 synsets. Every 3D model in the ShapeNetCore dataset is manually verified, categorized and annotated with alignment information. ShapeNet can freely used for non-commercial research.

We download the entire ShapeNetCore dataset and select the specific synsets. For the $Synth_{10}$ dataset we generate around 750 000 images, for $Synth_{17}$ 1.3 million images.

Data generation We use the 3D models of ShapeNet to render 2D images. Similar to [45] we render each 3D model in front of a random background to increase image clutterness. The background images are taken from ImageNet, whereas the background images do not appear in the real-world dataset. The rendered image has a resolution of 512x512 pixels. Figure 4.2 shows some examples of rendered images.

Since the number of 3D models in synset of ShapeNet is often much smaller than the equivalent number of images of the same synset in ImageNet, we render multiple variations of each model. See Table 4.2 For each variation we randomly sample another background image and change the model rotation. The virtual camera is set with a distance to the model, such that the model entirely fits in the camera view.

The images are rendered using the standard graphics pipeline. The implemented rendering approach does not generate realistic looking images, however the technique is fast enough to render a large amount of images.

Id	Synset	3D models	Variations
0	n02691156	4 045	20
1	n02958343	3 533	20
2	n03636649	2 318	30
3	n03691459	1 597	50
4	n04401088	1 089	70
5	n03001627	6 778	10
6	n02933112	1 571	50
7	n04090263	2 373	30
8	n03211117	1 093	70
9	n03467517	797	100
10	n04530566	1 939	40
11	n04256520	3 173	20
12	n04379243	8 436	10
13	n03046257	651	120
14	n03325088	744	110
15	n03642806	460	180
16	n03991062	602	140

Table 4.2.: Each 3D model is re-used multiple times with different model parameter and a different background image.

To ensure reproducibility of the data generation process, in a first step, scene description files are generated, which contain all necessary information to render an image. This scene description file is then read by the renderer, which loads the 3D model, the 2D background image and configures the model according to the scene file, see Figure 4.3. The rendering process can be distributed across multiple machines. The renderer is written in C++ using mLib[1] and DirectX 11.

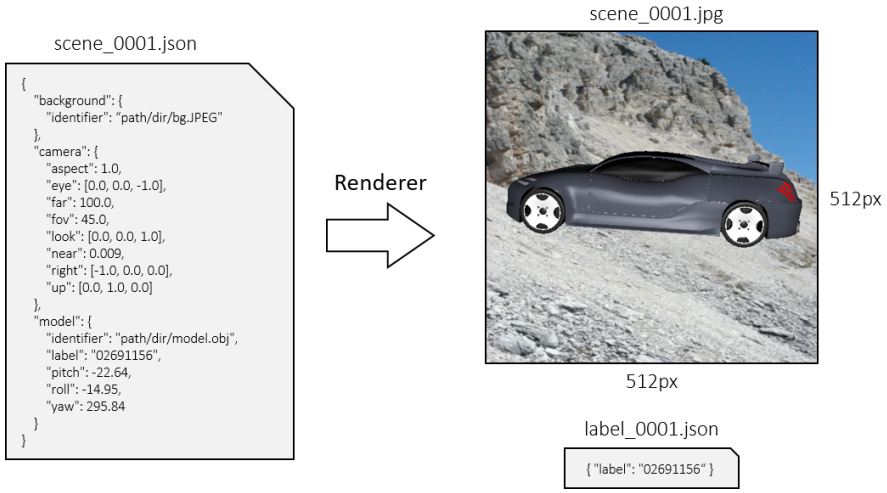


Figure 4.3.: The scene is described by a json file. For each 3D model, multiple scene descriptions are generated with varying backgrounds and object rotations. The renderer generates a 512px $\times$ 512px image based on the scene file.

[1] https://github.com/niessner/mLib

Id	Synset	Names	#real	#synth
0	n02691156	airplane, aeroplane, ...	14 324	80 900
1	n02958343	car, auto, automobile, ...	27 976	70 660
2	n03636649	lamp	4 096	69 540
3	n03691459	loudspeaker, speaker, ...	5 098	79 850
4	n04401088	telephone, phone, ...	6 657	76 230
5	n03001627	chair	25 792	67 780
6	n02933112	cabinet	5 677	78 550
7	n04090263	rifle	4 050	71 190
8	n03211117	display, video display	13 164	76 510
9	n03467517	guitar	3 779	79 700
Total DS_{10}			**110 613**	**750 910**
Average			11 061	75 091
10	n04530566	vessel, watercraft	61 182	77 560
11	n04256520	sofa, couch, lounge	10 314	63 460
12	n04379243	table	37 682	84 360
13	n03046257	clock	6 429	78 120
14	n03325088	faucet	3 832	81 840
15	n03642806	laptop	1 387	82 800
16	n03991062	flowerpot	2 441	84 280
Total DS_{17}			**233 880**	**1 303 330**
Average			13 757	76 666

Table 4.3.: Column #real shows the total number of images of a particular object class in the real dataset. Column #synth shows the total number of generated images in the synthetic dataset. Classes 0-9 correspond to the $Real_{10}$ and $Synth_{10}$ datasets. Classes 0-16 correspond to $Real_{17}$ and $Synth_{17}$.

5. Network Architecture

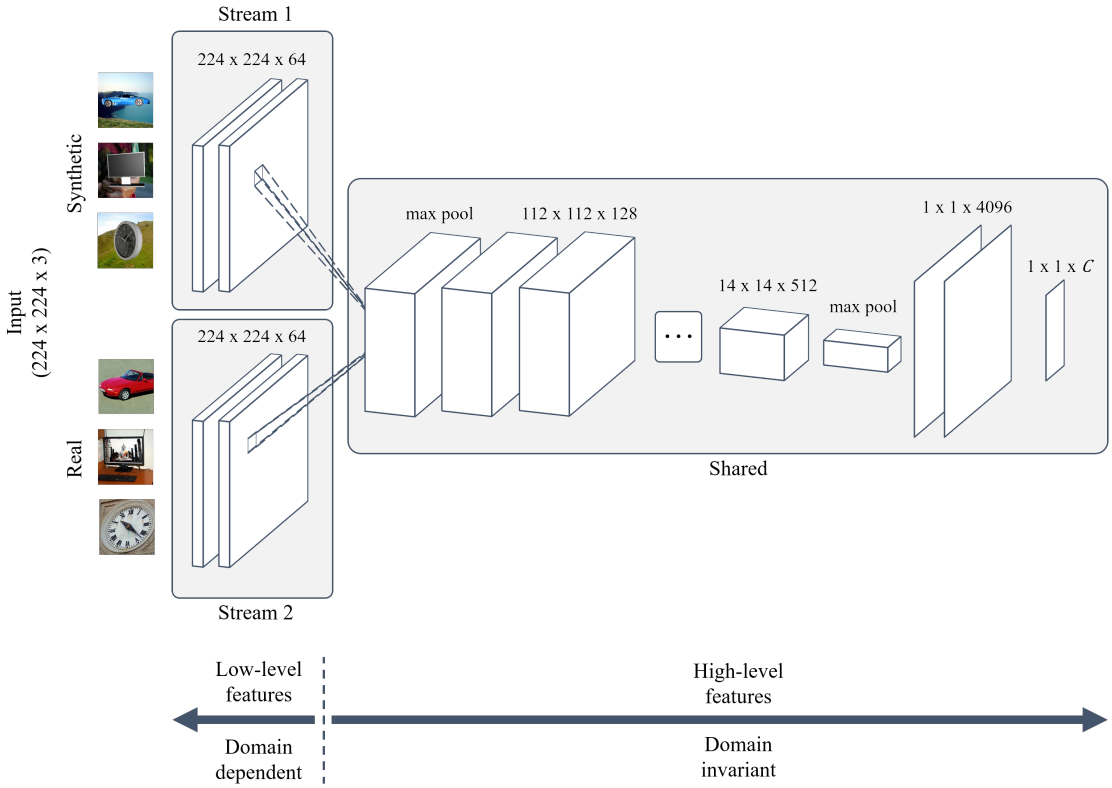


Figure 5.1.: Our Early-Two-stream Domain Adaptation approach separates the first few layers of a given network into two individual streams to extract domain-dependent low-level features. The rest of the network is shared between both domains and represents domain-invariant high-level features.

While the idea of our ETS-DA method is network-agnostic, we apply it to the VGG-16 [42] architecture. Figure 5.1 gives a general overview of the network. The concrete

architecture as evaluated in this thesis is described in Table 5.1.

The VGG network follows the idea of stacking a series of convolutional layers with small receptive fields of 3×3 and stride 1, combined with max-pool layers in between. At the end of the network, two 4096 dimensional fully-connected layers are followed by a final fully-connected layer with a dimension matching the number of classes of the classification task. In this thesis we use the VGG network with 16 weight layers as in configuration D in the original paper. It stacks 13 convolutional layers with 5 weight-less max-pool layers and the three mentioned fully-connected layers.

The main idea of ETS-DA is to separate the first few layers of the network into two individual streams, see chapter 3. The structure of both streams is identical, i.e. they are composed of the same layers with the same number of filters and channels. However, they have individual weights. The streams coalesce at a specific layer of the network, from where on both domains share the remaining part of the network. Since the network is trained alternatingly between the domains and since the images are fed to either of the streams, we utilize a switch to differentiate between the streams.

This architecture is in contrast to Siamese networks, which have tied weights between the streams [6], and e.g. two-stream architectures for video action recognition, where the streams are trained jointly on spatial and temporal data [41, 12].

The number of layers of the streams is denoted as depth d of the streams, ranging from $\{1, 2, ..., n\}$, where n refers to the maximum number of layers in the network. However, according to our idea of early Domain Adaptation, i.e. capturing the domain-specific low-level features within the first few layers, we aim for a low stream depth. In the following, we set $d = 2$.

One general issue of neural network is co-adaptation of successive layers, i.e. during training these layers form complex relationships, which can result in a degradation of performance [19]. Hence, separating the network between two co-adapted layers into the streams can complicate the training of the individual streams, since the individual streams have to reassemble the complex relationship. Using dropout can reduce this effect, but it can still appear [19]. The effects of co-adaptation can be subtle and a throughout comparison of different stream depths is outside the scope of this thesis.

However, with a stream depth of 2 the streams coalesce before the first max-pool layer. We assume that the max-pool layer ca reduce the risk of co-adaptation.

ETS-DA VGG-16 with depth 2	
real-world input stream	**synthetic input stream**
input (224 $\times$ 224 RGB image)	input (224 $\times$ 224 RGB image)
(3 $\times$ 3, 64) convolution (3 $\times$ 3, 64) convolution	(3 $\times$ 3, 64) convolution (3 $\times$ 3, 64) convolution
max-pool	
(3 $\times$ 3, 128) convolution (3 $\times$ 3, 128) convolution	
max-pool	
(3 $\times$ 3, 256) convolution (3 $\times$ 3, 256) convolution (3 $\times$ 3, 256) convolution	
max-pool	
(3 $\times$ 3, 512) convolution (3 $\times$ 3, 512) convolution (3 $\times$ 3, 512) convolution	
max-pool	
(3 $\times$ 3, 512) convolution (3 $\times$ 3, 512) convolution (3 $\times$ 3, 512) convolution	
max-pool	
4096 fully-connected 4096 fully-connected	
$\mathcal{C}$ fully-connected	

Table 5.1.: Our ETS-DA approach with depth 2 applied to the VGG-16 architecture. The dimension of the last fully-connected layer corresponds to the number of classes. In our case it can be 10 or 17.

6. Training

The training of our network is split into two stages: An Alignment and an Expansion step. An overview of the procedure is given in Figure 6.1.

Figure 6.1.: **Left:** In the first step, labeled images from both domains are fed to the network to establish a common feature space. **Middle:** In the second step, labeled images of additional classes from the synthetic domain are fed to the network to expand the established feature space. **Right:** The goal is that during testing images of these additional classes of the real domain are correctly classified.

Image Pre-processing We follow the standard image pre-processing strategies of VGG: Each RGB image is re-scaled uniformly so that the smaller side measures 256px and the original aspect ratio remains unchanged. After re-scaling, a random crop of 224px $\times$ 224px is taken from the image and the mean color of the ImageNet dataset is subtracted from the image. Additionally, there is a 50% chance the image will be flipped horizontally. During test time, we use the central 224px $\times$ 224px crop.

Initialization Our ETS-DA VGG-16 network is initialized with weights from a VGG-16 model that was pre-trained on the ILSVRC 1000-class classification tasks. In some experiments this pre-trained model is further fine-tuned on one of our domains, see subsection 7.2.3. For the streams of depth 2, i.e. the layers *conv1_1* and *conv1_2*, we copy the equivalent weights from the original model to each stream. We further replace the last 1000-dimensional fully-connected layer, which corresponded to the 1000 classes of the ILSVRC classification task, with a new fully-connected layer. The dimension of this layer has to match the number of classes of the current task: In the Alignment step the last layer has a dimension of 10, in the Expansion step a dimension of 17.

6.1. Alignment

The goal of the Alignment step is to establish a common domain-invariant feature space between the synthetic and the real domain, see Figure 6.1 (left). According to our idea, this feature space comprises higher-level concepts and thus resides within the representations of the later layers of the network. To achieve domain-invariance within this space, the individual streams have to extract the appropriate low-level features from their domain, which then build up the same higher-level features.

Scenario In this step we use the prepared datasets $Synth_{10}$ and $Real_{10}$ for the synthetic and the real domain. Both datasets consist of the same 10 classes and each image is provided with ground-truth information, see chapter 4. The dimension of the last fully-connected layer is set to 10 to match the 10 image classes, cf. $\mathcal{C}$ in Table 5.1. The

weights of this layer are randomly initialized.

Given labeled images in both domains, we can formalize this as a modified supervised Domain Adaptation scenario. Instead of one source domain and one target domain, we define both domains as source domains and the domain-invariant feature space as target domain:

$$\mathcal{D}^{s,1} = Synth_{10}$$
$$\mathcal{D}^{s,2} = Real_{10}$$
$$\mathcal{D}^{t} = \mathcal{D}^{s,1} \cap \mathcal{D}^{s,2}, \text{ where } \cap \text{ denotes the common feature space}$$

Further, the primary task $\mathcal{T} = \mathcal{T}^s = \mathcal{T}^t$ is to correctly predict the class of a given image among all 10 possible classes of the label space. Additionally, we can define a sub-task $\widetilde{\mathcal{T}}$ as follows: Divide the overall feature space into a domain-specific and a domain-invariant part. This sub-task constrains the primary task and reflects the design of the two-stream architecture.

Training sequence To train our network, we alternatingly feed images from both domains through their respective input stream. To take class imbalances in the real dataset into account, we re-weight the cross-entropy loss with balance factors for the classes. The balance factor is calculated with n_{max}/n_c, where n_{max} is the maximum number of images of a certain class in the current domain and n_c the number of images of the specific class.

Since the weights of the last fully-connected layer are newly initialized to match the 10 classes, we first train this last layer in isolation. Therefore, we freeze the weights of the entire network except for the last layer and train the network for 30 000 iterations[1] with an initial learning rate of $1e^{-3}$. The learning rate further decays by a factor of 0.1 after 10 000 and 15 000 iterations.

[1] One iteration refers to one batch of images from the synthetic domain and one batch of images from real domain

After the initial training of the last layer, the separation of the network into the two regions — the individual streams and the shared part — lets us explore multiple training options:

Model We train the entire network without freezing any weights for 30 000 iterations and initial learning rate of $1e^{-6}$. After 15 000 iterations we decay the learning rate to $1e^{-7}$.

Streams We observed that training the entire model does not help to extract the domain-specifics within the individual streams. To solve this problem, we can force the streams to adapt to the common feature space by freezing the shared part network and solely train the streams. Intuitively, this forces the streams to extract the appropriate low-level features, which are necessary to build up the higher-level features. For this scenario we train the streams for 30 000 iterations with an learning rate of $1e^{-3}$, which decays to $1e^{-4}$ after 15 000 iterations.

Shared After adapting the streams to their domain, they extract domain-specific low-level features. Subsequently training the shared part of the network with images from both domains corresponds to training the domain-invariant space. We train the shared part for 30 000 iterations with an initial learning of $1e^{-5}$ and decay it to $1e^{-6}$ after 15 000 iterations.

6.2. Expansion

The goal of the Expansion step is to expand the established domain-invariant feature space by introducing additional classes. However, labeled images are only available in the synthetic domain, see Figure 6.1 (middle). Therefore, the problem statement is two-fold and consists of an inductive Transfer Learning scenario and an unsupervised Domain Adaptation scenario.

Inductive Transfer Learning For this part of the problem statement we consider the previously established feature space as source domain and the $Synth_{17}$ dataset with labels as target domain. The source and the target task correspond to the classification

task of the Alignment step, i.e. predicting the 10 classes, and predicting the 17 classes of $Synth_{17}$, respectively.

$$\mathcal{D}^s = Synth_{10} \cap Real_{10}$$
$$\mathcal{D}^t = Synth_{17}$$
$$\mathcal{T}^s = \text{Classifying the 10 classes of } Synth_{10} \text{ and } Real_{10}$$
$$\mathcal{T}^t = \text{Classifying the 17 classes of } Synth_{17}$$

Unsupervised Domain Adaptation Further, we define the second part of the problem statement as unsupervised Domain Adaptation: Source and target task are the same and correspond to classifying the 17 classes of $Synth_{17}$ and $Real_{17}$. While labels are available in the synthetic domain, they are absent in the real-world domain. We formalize the situation as follows:

$$\mathcal{D}^s = (Synth_{10} \cap Real_{10}) \cap Synth_{17}$$
$$\mathcal{D}^t = Real_{17}$$
$$\mathcal{T} = \text{Classifying the 17 classes of } Synth_{17} \text{ and } Real_{17}$$

Training sequence We train the network with the labeled images of $Synth_{17}$, which are fed to the synthetic stream of the network.

As in the Alignment step we replace the last layer of the network so that the dimension matches the number of classes. Since the label space $\mathcal{Y}_{17}$ is a superset of the label space $\mathcal{Y}_{10}$ in the Alignment step, we examined the effect of reuse the weights of the 10-dimensional layer to initialize the new 17-dimensional layer. The weights for the additional 7 classes are randomly initialized, see section 7.3.

We sequentially train the last layer and the shared part of the network each for 30 000 iterations. While training the last layer starts with a learning rate of $1e^{-3}$ and decays

with a factor of 0.1 after 10 000 and 15 000 iterations, training the shared part starts with a learning rate of $1e^{-6}$ and decays after 15 000 iterations.

6.3. Implementation

In all trainings above, we used the Adam optimizer with the stated initial learning rates, β_1 set to 0.9, β_2 to 0.99 and ϵ to $1e^{-8}$. The network was implemented in Python 3.6 using Tensorflow [1] version 1.6. Tensorflow is configured to run the training using CUDA 9 and cuNN 7 on a NVIDIA GeForce GTX 1080 Ti, which allows a batch size of 32. The implementation of our ETS-DA VGG-16 version follows the official implementation of the version provided by Tensorflow. The pre-trained VGG-16 model was taken from the Tensorflow model-zoo[2].

One full training cycle consisting of both training steps – Alignment and Expansion – takes about 1 day.

[2] `https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models`

7. Evaluation

To evaluate the efficacy of our ETS-DA approach, we conducted a series of experiments. These experiments are split into two sets corresponding to the two training steps – Alignment and Expansion.

7.1. Comparison Methods

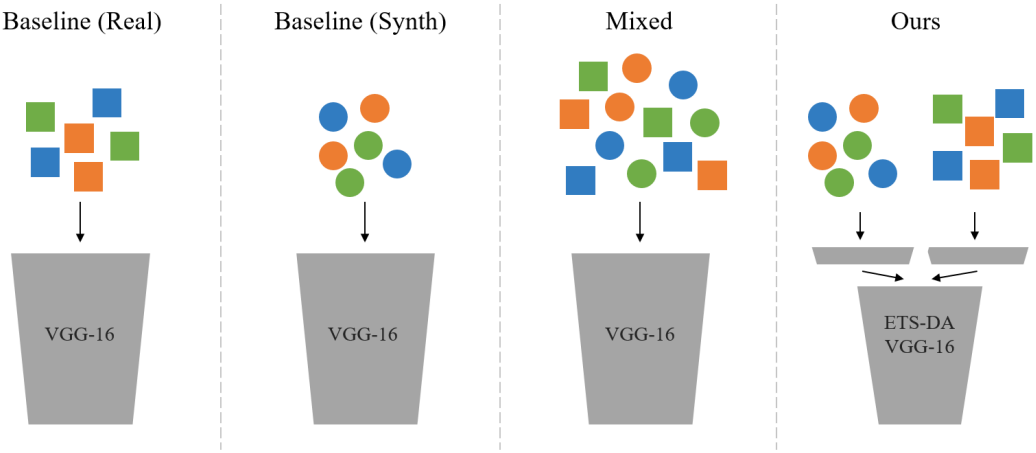


Figure 7.1.: Overview of the evaluated methods. *Baseline (Real)*, *Baseline (Synth)* and *Mixed* use the VGG-16 architecture. *Ours* uses the proposed ETS-DA architecture.

We compare our method with 3 other approaches, which do not employ our ETS-DA architecture but the standard VGG-16 architecture. For a fair comparison, we use the same training sequences and parameters as described in section 6.1 and section 6.2 for each method. Figure 7.1 gives an overview of the four different methods. Table 7.1

shows, which datasets are used during training for each method.

Baseline (Real) For this case we consider complete label information for the real domain during both training steps. To train the network we feed the real-world images to the single stream of the VGG-16 architecture. We use the $Real_{10}$ and $Real_{17}$ datasets with labels for the Alignment step and Expansion step, respectively.

Baseline (Synth) This method is similar to *Baseline (Real)* method, just that we train the network only with synthetic data during both training steps. We use the images of $Synth_{10}$ and $Synth_{17}$.

Mixed In this approach we mix the real and synthetic domain by creating a new dataset $Mixed_{10}$. Each class is composed by images from both domains with equal distribution, i.e. 50% from $Real_{10}$ and 50% from $Synth_{10}$. While we use this mixed dataset to train the network during the Alignment step, we use $Synth_{17}$ in the Expansion step.

	Baseline (Real)	**Baseline (Synth)**	**Mixed**	**Ours**
Alignment	$Real_{10}$	$Synth_{10}$	$Mixed_{10}$	$Real_{10} + Synth_{10}$
Expansion	$Real_{17}$	$Synth_{17}$	$Synth_{17}$	$Synth_{17}$

Table 7.1.: Datasets used to train the networks of the different methods.

7.2. Alignment

7.2.1. Baselines

To be able to put the results of different design decisions and different training strategies into context, we first evaluate the performance of all 3 comparison methods – *Baseline (Real)*, *Baseline (Synth)*, *Mixed* – and our ETS-DA approach. We initialize all networks with weights of the pre-trained VGG-16 model and train the networks on their designated datasets, see Table 7.1. We use the "Model" training strategy, i.e. training the entire network at once, cf. section 6.1.

	$Real_{10}$	$Synth_{10}$	*Abs. Diff*
Baseline (Real)	95.08%	46.69%	48.39%
Baseline (Synth)	73.40%	90.82%	17.42%
Mixed	93.56%	87.52%	6.04%
Ours	93.59%	90.81%	2.78%

Table 7.2.: Average per-class accuracy of all 4 methods on the $Real_{10}$ and $Synth_{10}$ test sets. The last column shows the absolute difference between both domains. Refer to Table B.1 for all class accuracies.

By comparing the values in Table 7.2 we can draw following conclusions:

- **Domain Discrepancy:** There exists a significant domain discrepancy between both domains: Solely training on one domain yields an absolute difference of 48.39% (*Baseline (Real)*) and 17.42% (*Baseline (Synth)*).
- **Reality Gap:** The difference of 17.42% between $Synth_{10}$ and $Real_{10}$ of *Baseline (Synth)* reconfirms the existence of the Reality Gap.
- **Joint Domain:** Training with data of both domains reduces the domain discrepancy to 6.04% (*Mixed*).
- **ETS-DA:** Giving the network the possibility to extract low-level features for each domain separately reduces the discrepancy to 2.78% (*Ours*).

These results indicate that our ETS-DA method can reduce the domain discrepancy between the real-world and synthetic domain in this supervised scenario. The detailed list in Table B.1 further reveals that for some classes our approach can not only achieve the same accuracy as the respective baseline, but even outperform it. For instance, the accuracy of class "Chair" of our approach (98.22%) exceeds *Baseline (Real)* (95.43%) by 2.79%. We assume that this improvement origins from the increased variety introduced by the synthetic domain.

7.2.2. Experiment: Explicit Early Adaptation

In this experiment we investigate the effect of explicitly training the individual streams, while the weights of the shared part of the network are frozen. Intuitively, this forces the streams to adapt to the existing shared feature space and thus to extract the necessary low-level features from their specific domain, which then build up the shared higher-level features.

To this end, we train two ETS-DA networks, which are both initialized with pre-trained VGG-16 weights and have already trained the last layer . We refer to this state as *Ours (Last)*.
The first network, *Ours (Model)*, is trained using the "Model" strategy and coincide with the result in subsection 7.2.1. The second network is sequentially trained with the "Streams" strategy, *Ours (Streams)*, and the "Shared" strategy, *Ours (Shared)*.

The results of the experiment in Table 7.3 shows that explicitly adapting the streams reduce the difference between the domains to 0.38%. This strategy also increases the mean accuracies by 0.44% in the real-world domain and by 2.84% in the synthetic domain.

The effects of explicit adaptation becomes even more evident if we analyze the difference between the two streams of the learned weights in the first layer of the network. Figure 7.2 visualizes these differences. It clearly shows that training the entire model at once hardly changes the first layer, which results in marginally different stream weights.

[1]The data is normalized using a power law function to give the small differences in the left image a higher contrast.

	$Real_{10}$	$Synth_{10}$	*Abs. Diff*
Ours (Last)	92.90%	76.25%	16.65%
Ours (Model)	93.59%	90.81%	2.78%
Ours (Streams)	92.32%	82.65%	9.67%
Ours (Shared)	94.03%	93.65%	0.38%

Table 7.3.: Average per-class accuracy after training with different training strategies. The values refer to the results after evaluation on the $Real_{10}$ and $Synth_{10}$ test sets. The last column shows the absolute difference between both domains. Refer to Table B.2 for all class accuracies.

Explicit adaptation forces the streams to learn domain-specific low-level features and results in noticeable differences.

Explicit Adaptation for *Mixed* Theoretically, the idea of explicit adaptation should also be applicable to the *Mixed* method. This corresponds to the adaptation of the single input stream to the joint domain of $Real_{10}$ and $Synth_{10}$. However, as we can see in Figure 7.3 an adaptation towards the synthetic domain simultaneously results in an alienation of the real domain. Due to the individual streams in ETS-DA, our method can separately adapt to each domain without performance degradation in the other domain.

The pre-trained VGG-16 model was trained on ImageNet. Since $Real_{10}$ is also drawn from ImageNet, the pre-trained model is initially biased towards the real-world domain. Therefore, adapting the streams primarily improves the synthetic domain.

7.2.3. Experiment: Additional Fine-Tuning

In the final experiment of the Alignment step, we evaluate the effects of fine-tuning the pre-trained VGG-16 model on one of the domains and then using this fine-tuned model to initialize our ETS-DA network. The fine-tuned networks coincide with the *Baseline (Real)* and *Baseline (Synth)* methods and thus will be used to initialize *Ours*

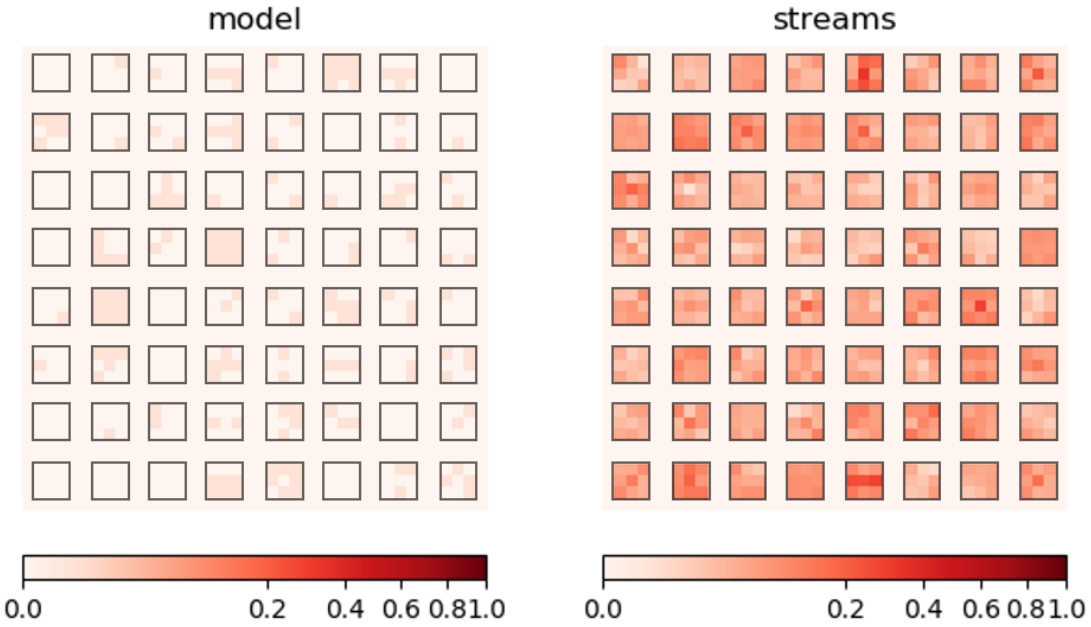


Figure 7.2.: Visualization of the differences between the weights of both first layers of the both streams[1]. The 8×8 squares correspond to the 64 filters of the first convolutional layer. **Left:** Differences after training of *Ours (Model)*. **Right:** Differences after explicitly training the streams *Ours (Streams)*. Higher values indicate bigger differences between the streams. After training the entire model, both first layers of the streams marginally differ. Explicitly training the streams helps to reflect the differences of the low-level features between the domains.

(FT-R) and *Ours (FT-S)*, respectively. In the previous experiment, we have shown the advantages of explicit adaptation. Therefore, we also employ the "Streams"–"Shared" training sequence in this experiment.

Table 7.4 shows that additional fine-tuning of the pre-trained model on one domain increases the performance in this domain, but results in a degradation in the other domain. While *Ours (N-FT)* achieves the best absolute difference between the domains (0.38%), *Ours (FT-R)* gives the best accuracy in the real-world domain. This is of particular interest, since the real-world domain is the final target domain in the Expansion step.

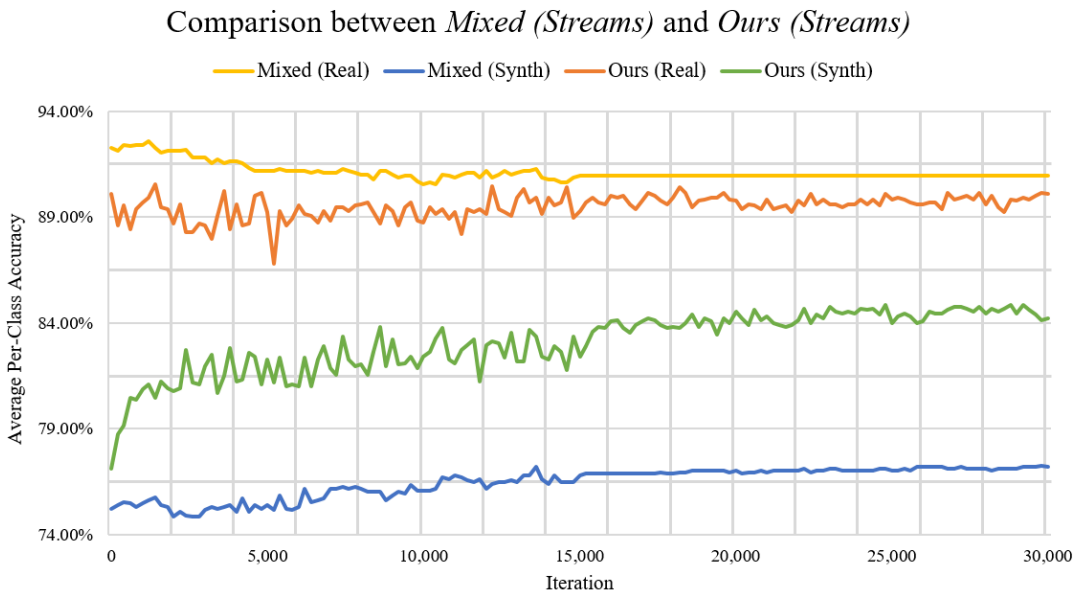


Figure 7.3.: Training progress of *Mixed (Streams)* and *Ours (Streams)*. The yellow and blue curve correspond to the *Mixed* method. The orange and green curve show our approach. Every 200 iterations a subset of the validation set (128 images per class) is evaluated with the current state of the network. ETS-DA allow both domains to adapt individually, whereas in *Mixed* improvement is mutually exclusive. After 15 000 iterations the learning rate is decreased to $1e^{-4}$ and the networks start to converge.

	$Real_{10}$	$Synth_{10}$	*Abs. Diff*
Ours (N-FT)	94.03%	93.65%	0.38%
Ours (FT-R)	94.53%	90.43%	4.10%
Ours (FT-S)	92.55%	91.81%	0.74%

Table 7.4.: Average per-class accuracy after the "Streams"–"Shared" training sequence. The values refer to the results after evaluation on the $Real_{10}$ and $Synth_{10}$ test sets. *Ours (N-FT)* refers to no additional fine-tuning of the pre-trained VGG-16 model. Refer to Table B.3 for all class accuracies.

Surprisingly, *Ours (FT-S)* performs worse than the non-fine-tuned approach in the synthetic domain, even though the initialized pre-trained model was fine-tuned on synthetic data. This results raises an interesting "Chicken-and-Egg" problem, which is discussed in section 8.

7.3. Expansion

In the last section, we saw that our ETS-DA approach can successfully adapt the streams to their specific domains. The streams extract the domain-specific low-level features that build up domain-invariant higher-level concepts in the shared part of the network. In the following experiments we examine the possibility of expanding the domain-invariant feature space by introducing additional classes only in the synthetic domain, while also recognizing the novel classes in the real domain, see section 6.2.

7.3.1. Experiment: Reusing the Last Layer

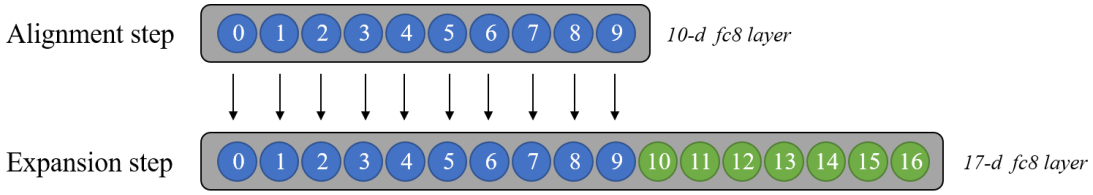


Figure 7.4.: Reusing the previously trained 10-dimensional layer to initialize the new 17-dimensional fully-connected layer. The weights for the newly introduced 7 classes are randomly initialized.

While transitioning from the Alignment step to the Expansion step, we replace the last 10-dimensional fully-connected layer with a new 17-dimensional fully-connected layer to match the 17 classes of $Synth_{17}$. Since $Synth_{10}$ is a subset of $Synth_{17}$, we are interested in the effects of reusing the previously learned weights of the last layer to initialize the weights of the same 10 classes. The remaining weights are randomly initialized.

For this experiment, we employ *Ours (FT-R)* as the base model and replace the last layer. In *Ours (R-U)* we re-use the weights of *Ours (FT-R)*, see Figure 7.4. In *Ours (R-I)*, we randomly initialize the entire last layer. For both approaches, we sequentially train the last layer and the shared part of the network using $Synth_{17}$. After training both networks are evaluated on the $Synth_{17}$ and $Real_{17}$ test sets.

Re-using the weights for the last layer increases the mean accuracy of $Real_{17}$ by 0.25%,

	$Synth_{17}$*	$Real_{17}$	$Real_{10}$	$Real_{+7}$
Ours (R-I)	85.75%	79.07%	85.70%	69.60%
Ours (R-U)	85.72%	79.32%	85.50%	70.48%

Table 7.5.: Evaluation of random initialization (*Ours (R-I)*) and re-usage the weights (*Ours (R-U)*). The values correspond to the average per-class accuracy of each dataset. $Real_{+7}$ refers to the dataset of the additional 7 classes in the real-world domain. Only the marked (*) dataset, i.e. $Synth_{17}$, was used during training.

see Table 7.5.

7.3.2. Experiment: Comparison

In the final experiment we evaluate the performance of the 3 comparison methods and our ETS-DA approach in 3 different variations. For two of our methods, we apply the "Streams"–"Shared" training strategy, fine-tune on one domain and re-use the previously learned weights. Hence, we use *Ours (FT-S)* and *Ours (FT-R)* as described in subsection 7.2.3. For our third method, we use *Ours (Model)* as described in subsection 7.2.2 and we also re-use of weights.
The 3 comparison methods – *Baseline (Real)*, *Baseline (Synth)*, *Mixed* – are trained as described in section 7.1. *Baseline (Real)* is the only method that uses labeled training data of the newly introduced classes of the real-world domain and serves as illustration of a theoretical upper bound.

By introducing the additional 7 classes to the network, some of the classes are very similar: "Cabinet" and "Chair" are comparable to "Table" and "Sofa". "Display" and "Telephone" are similar to "Laptop". These similarities are reflected in the confusion matrices, cf. Figure 7.5 and Figure 7.6. Across all methods, including ours, the most confused and thus most problematic class was "Table".

From the quantitative results in Table 7.6 and Table C.1 we can see, that our ETS-DA method outperforms the *Baseline (Synth)* and *Mixed* methods in the final real-world target domain. Our best performing configuration, *Ours (FT-R)*, increases the average

per-class accuracy on $Real_{17}$ compared to *Baseline (Synth)* and *Mixed* by 18.91% and 10.75%, respectively. Our method even surpasses *Baseline (Real)* for the classes "Display" and "Clock" and achieves a better class accuracy. Additionally, for the class "Clock" our method has never seen real-world training examples.

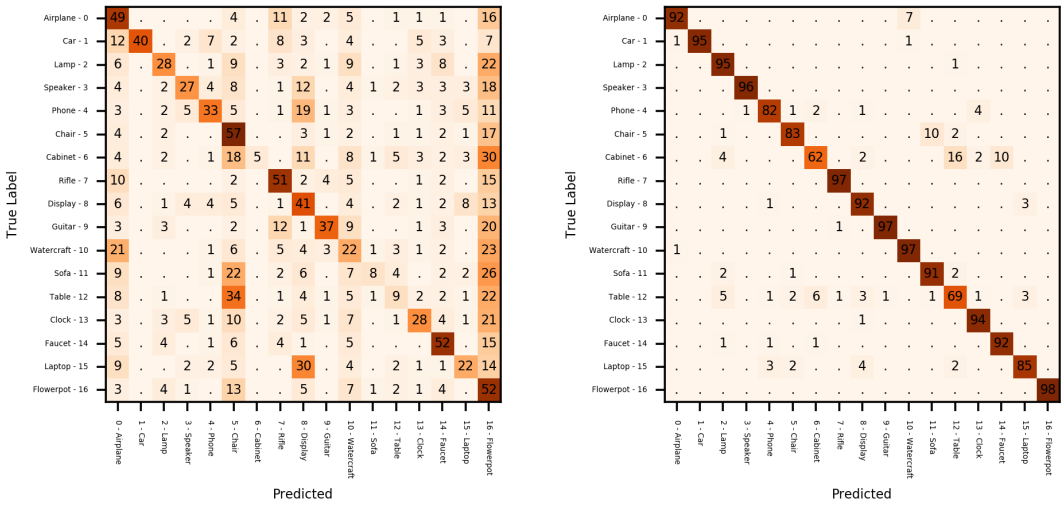
Figure 7.7 and Figure C.1 qualitatively show that ETS-DA can significantly better discriminate the additional classes within the expanded feature space than *Baseline (Synth)* and *Mixed*.

Evaluation of the $Real_{17}$ test set

Class \ Method	Baseline (Real)	Baseline (Synth)	Mixed	Ours (Model)	Ours (FT-S)	Ours (FT-R)
Airplane	*92.39%*	61.76%	72.71%	82.34%	**85.55%**	85.35%
Car	*95.86%*	74.88%	87.42%	92.85%	92.10%	**94.43%**
Lamp	*95.13%*	74.45%	85.64%	86.13%	83.21%	**91.97%**
Loudspeaker	*96.28%*	61.84%	78.47%	85.32%	88.45%	**91.59%**
Telephone	*82.46%*	56.22%	62.37%	70.31%	68.52%	**73.31%**
Chair	*83.68%*	52.21%	57.95%	67.83%	72.52%	**79.73%**
Cabinet	*62.39%*	25.83%	51.32%	60.81%	**67.14%**	60.98%
Rifle	*97.78%*	87.41%	90.86%	92.35%	92.10%	**93.58%**
Display	*92.33%*	71.37%	81.78%	84.74%	82.61%	**92.79%**
Guitar	*97.36%*	48.28%	63.85%	72.03%	84.17%	**91.29%**
Watercraft	*97.61%*	77.02%	77.41%	77.74%	79.90%	**91.47%**
Sofa	*91.47%*	59.01%	74.81%	79.94%	**85.95%**	84.01%
Table	*69.01%*	21.15%	21.12%	21.44%	24.09%	**31.84%**
Clock	*94.25%*	87.73%	89.13%	89.60%	91.30%	**94.57%**
Faucet	*92.19%*	39.58%	44.27%	52.34%	44.79%	**53.13%**
Laptop	*85.71%*	52.14%	47.86%	49.29%	**60.00%**	55.00%
Flowerpot	*98.72%*	76.07%	78.63%	74.01%	**86.32%**	83.33%
Total Accuracy	*88.77%*	60.00%	65.97%	69.70%	71.89%	**78.21%**
Avg. Accuracy	*89.68%*	60.41%	68.57%	72.89%	75.81%	**79.32%**
Avg. Acc (base classes)	*89.57%*	61.43%	73.24%	79.47%	81.64%	**85.50%**
Avg. Acc (novel classes)	*89.85%*	58.96%	61.89%	63.48%	67.48%	**70.45%**

Table 7.6.: The per-class accuracies of the evaluated methods. We compare our approach in three configurations with *Mixed* and *Baseline (Synth)*. *Baseline (Real)* serves as reference. The results show that our approach outperforms *Mixed* in each class. Among our configurations *Ours (FT-R)* performs best in most cases. The equivalent evaluation on the $Synth_{17}$ test set can be found in the appendix Table C.1.

(a) **Baseline (Real)**
Left: Synthetic domain. Right: Real domain



(b) **Baseline (Synth)**
Left: Synthetic domain. Right: Real domain

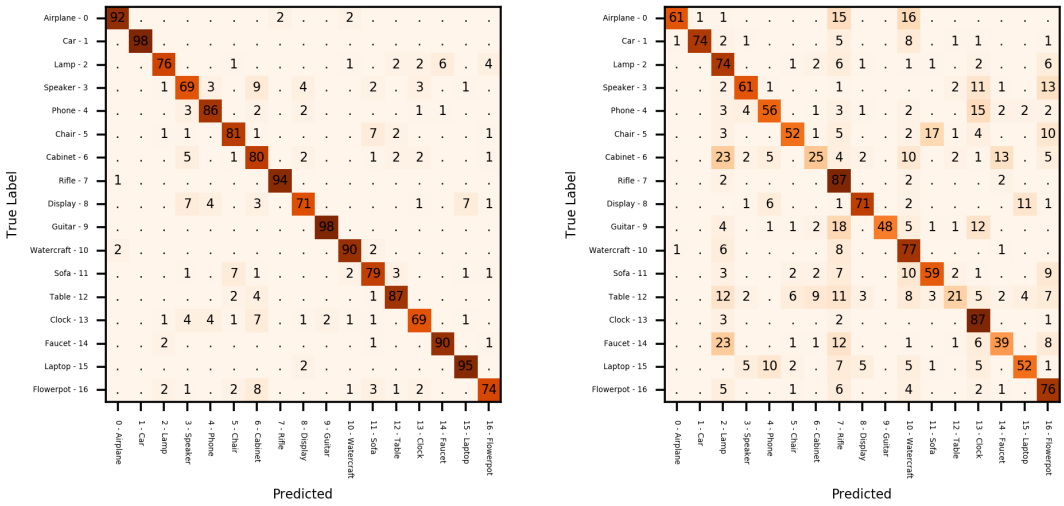
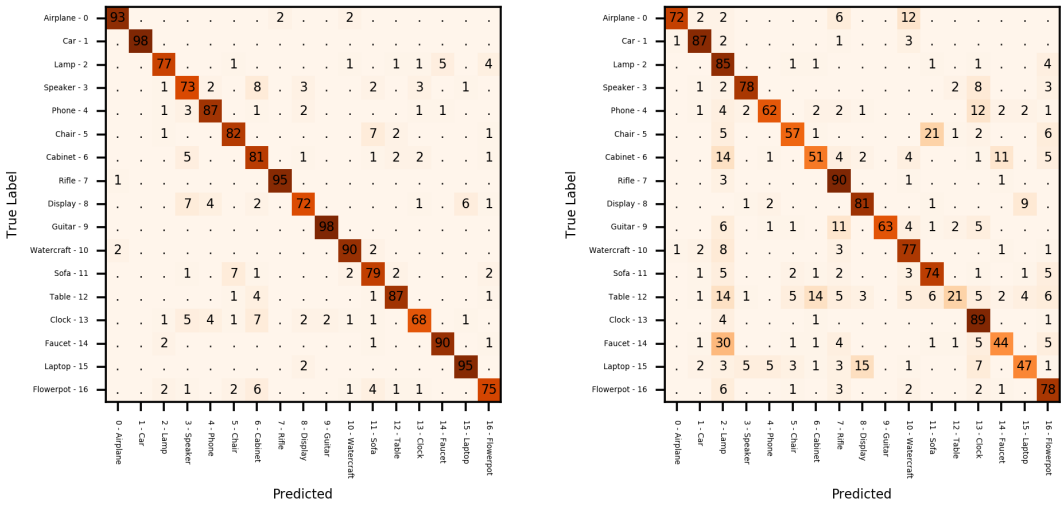


Figure 7.5.: Confusion matrices of the 17-class experiment. While *Baseline (Real)* fails in the synthetic domain, but performs well in the real domain, *Baseline (Synth)* performs well in the synthetic domain, but struggles in the real domain.

(a) **Mixed**
Left: Synthetic domain. Right: Real domain



(b) **Ours (FT-R)**
Left: Synthetic domain. Right: Real domain

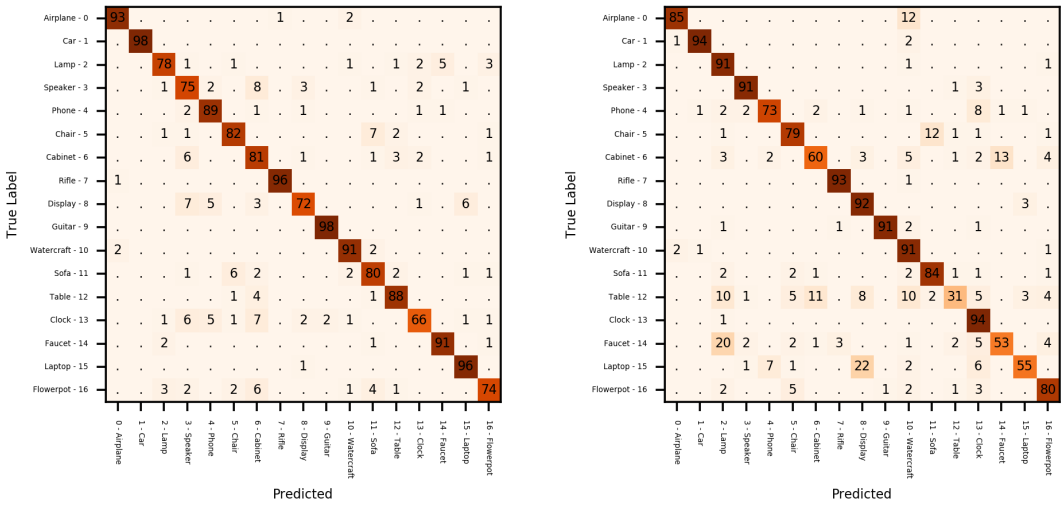


Figure 7.6.: Confusion matrices of the 17-class experiment. *Mixed* performs equally well in the synthetic domain as *Baseline (Synth)*, but struggles in the real domain, *Ours (FT-R)* instead performs well in both domains.

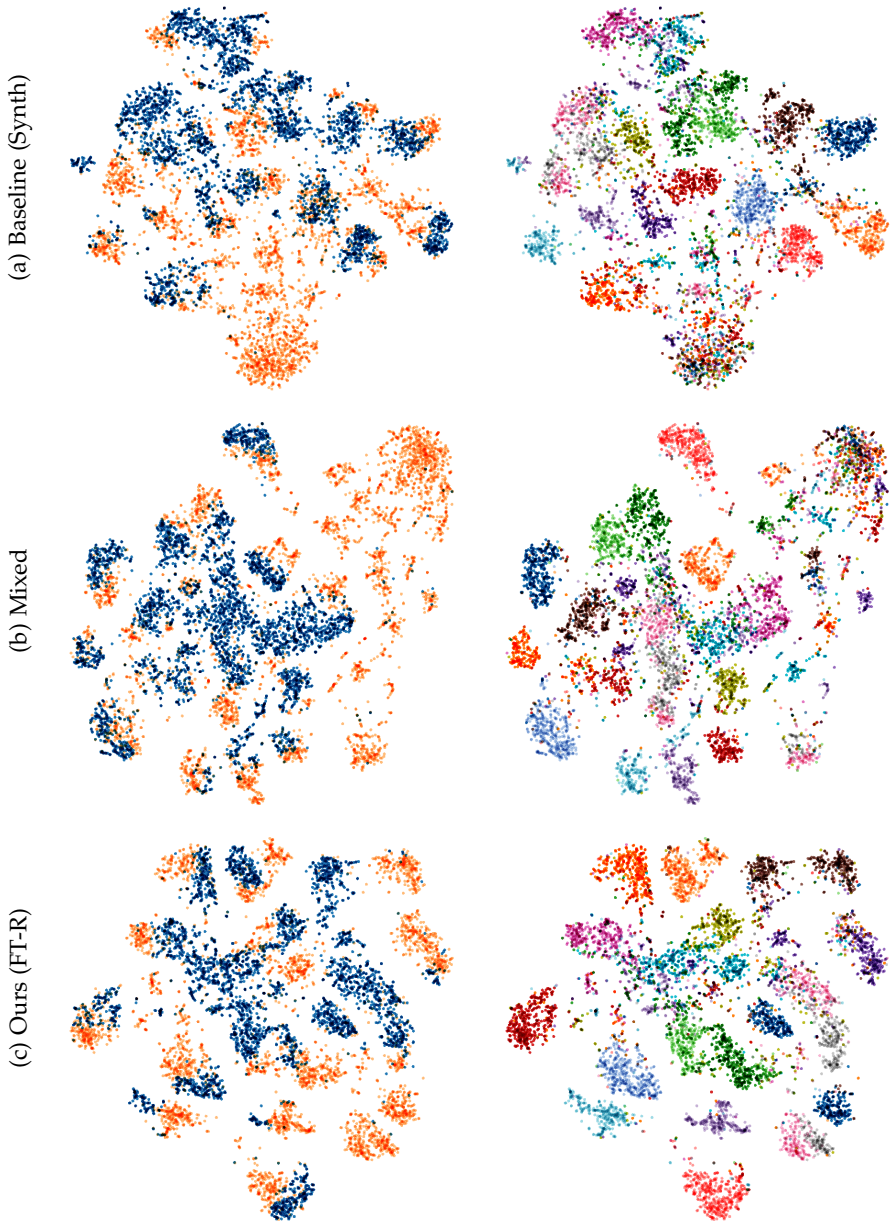


Figure 7.7.: t-SNE [27] visualizations of the 17-dimensional *fc8* feature space with perplexity 30. For visibility only 256 samples of each class are displayed. **Left:** Blue and orange dots correspond to synthetic samples (source) and real samples (target), respectively. **Right:** Each color refer to one of the 17 classes. (a) and (b) cannot discriminate the additional classes of real samples and map them to one cluster. (c) shows, that ETS-DA can discriminate between the classes, while maintaining small inter-domain distances.

8. Discussion & Limitations

The results of our Early-Two-Stream Domain Adaptation method are promising. However, it does not come without drawbacks. The values in Table 7.6 demonstrate that we outperform *Baseline (Synth)* and *Mixed*, but is still 10.36% behind *Baseline (Real)*.

Feature-based Domain Adaptation Our approach achieves Domain Adaptation on a feature-based level. However, if a particular feature is missing in the synthetic domain but normally present in the real-world domain, it cannot be learned by the network. Hence, the network fails to recognize it when applying on the real-world domain. By utilizing 3D models, which are as close as possible to the real-world, this problem can be mitigated.

Synthetic Data Generation Furthermore, the synthetic image generation currently does not incorporate any contextual information and only focuses on the depicted object. However, some classes are difficult to detect without context. This can be seen for example in Figure 8.1b, in the second row, the second image from the right. The true label of this image is "Watercraft", however our network predicts an "Clock". Supplying additional context to the synthetic image such as a more plausible surrounding, could help the network to learn to incorporate more of the context. In the concrete example of the "Watercraft", by rendering the 3D model in front of water landscape, the network could deduce the label "Watercraft" from the surrounding water.

Chicken-and-Egg Problem While the proposed "Streams"–"Shared" training strategy has proven to be an effective technique to explicitly adapt the streams to their domains,

it also revealed an interesting "Chicken-and-Egg" problem:
How should the streams learn to extract the appropriate low-level features that establish the domain-invariant feature space, if the domain-invariant features can only be established by already learned domain-specific low-level features?

One possible solution, would be to train the streams and the shared part alternatingly multiple times. This solution could potentially converge to a "true" domain-invariant feature space.

(a) Real images

(b) Synthetic images

Figure 8.1.: Example images that our network wrongly classified. The left and right tags in each image indicate the correct and the predicted labels, respectively.

9. Conclusion

In this Master's Thesis, we presented Early-Two-Stream Domain Adaptation (ETS-DA), a simple, yet effective approach to reduce the negative effects of domain discrepancy between two domains. We particularly focused on the domain discrepancy between real and synthetic data, also referred to as "Reality Gap". Bridging this gap makes it possible to leverage synthetic data as an effective source of virtually unlimited training data.

The key idea of ETS-DA is to align both domains early on within a given deep neural network. To this end, we split the first few layers of the network into individual, domain-specific streams, whereas the remaining part of the network is shared between the domains.

The training of our network consists of two phases: In the first step, the network takes labeled images from both domains to learn the domain specifics and to establish a domain-invariant feature space.

In the second step, we extend the domains by introducing additional object categories. However, labels of the additional classes are only provided for the synthetic images. Training the shared part of the network with the labeled data of the additional classes expands the domain-invariant feature space. Our results show, that ETS-DA diminishes domain discrepancy and leads to increased accuracy of the additional classes in the real domain.

The separation of the training process in two phases allows to investigate each phase in isolation. This opens interesting directions for future work.

For example, we used the standard classification loss as training objective and achieved

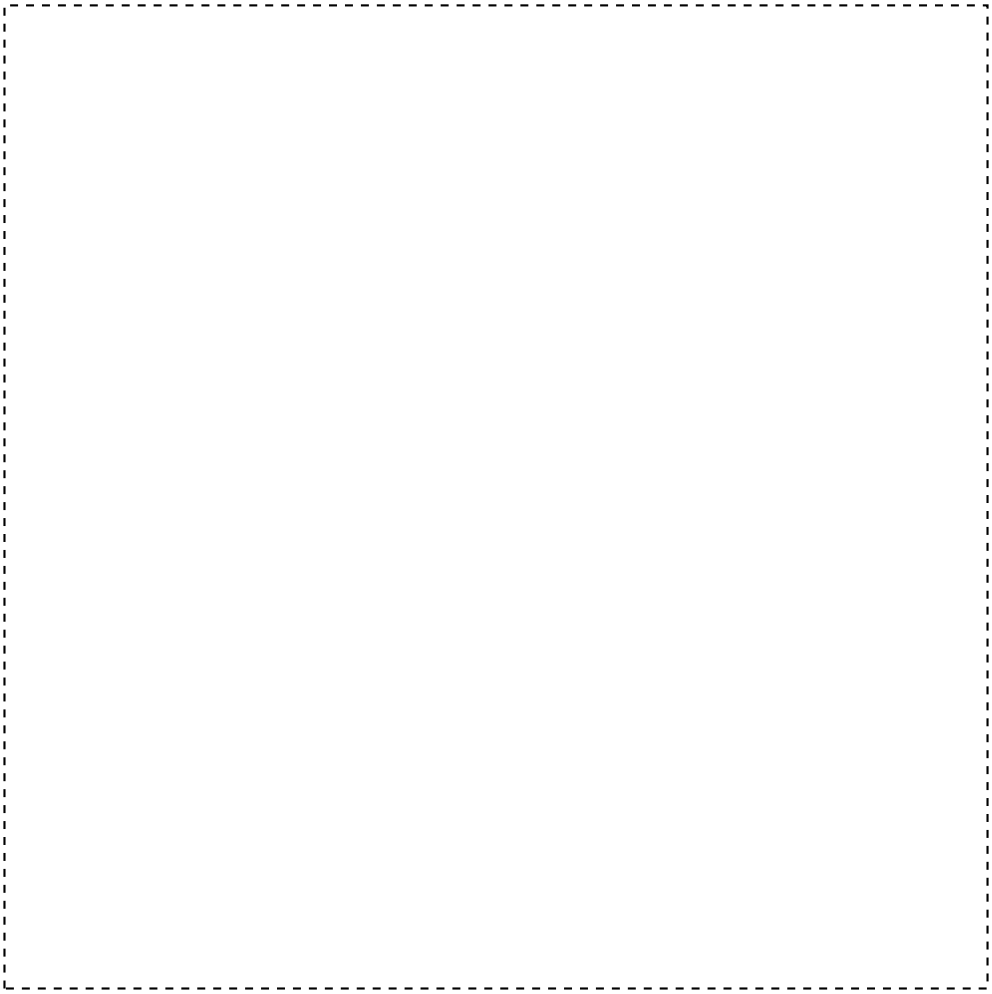
domain adaptation by freezing different parts of the network. Recent deep domain adaptation methods incorporate special loss functions into the training process. Therefore, employing such an additional objective in the ETS-DA architecture could further boost adaptation.

Another important aspect of future work goes in the direction of semi-supervised or few-shot domain adaptation. Currently, we rely on a reasonably large real-world dataset in the first training step. Hence, reducing the required amount of annotated, real-world data to a minimum to learn the domain specifics would further enlarge the possible application areas of ETS-DA.

Within the current momentum of deep domain adaptation, we want to emphasize the importance of synthetic data. Leveraging its full potential by closing the "Reality Gap" is another important step in the development of deep learning to advance to new areas where large, annotated datasets are sparse.

Appendix

A. DVD



The DVD contains the source code of the renderer and the deep learning framework. The DVD also contains a PDF version of this thesis and all cited literature. To reproduce the synthetic data, the scene description files are included, which can be read by the renderer (Requires ImageNet and ShapeNet). To reproduce the real dataset, a list of ImageNet filenames is supplemented.

B. Alignment: Additional Results

Baseline

Method / Class	Baseline (Real)		Baseline (Synth)		Mixed		Ours	
	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$
Airplane	99.30%	58.37%	75.58%	96.07%	98.74%	93.65%	99.02%	96.34%
Car	97.68%	53.27%	85.85%	98.99%	96.25%	98.57%	97.78%	99.12%
Lamp	97.57%	46.21%	84.43%	89.66%	96.59%	84.73%	94.40%	89.72%
Loudspeaker	97.06%	33.65%	75.54%	79.26%	97.26%	68.81%	96.28%	78.68%
Telephone	85.76%	35.19%	63.57%	89.16%	83.96%	86.33%	81.86%	89.49%
Chair	95.43%	80.78%	82.91%	92.81%	95.39%	88.79%	98.22%	92.37%
Cabinet	85.24%	10.57%	32.34%	86.87%	77.50%	84.70%	76.98%	87.04%
Rifle	98.27%	53.63%	91.11%	96.60%	97.04%	94.38%	96.79%	96.50%
Display	96.36%	56.31%	86.71%	80.19%	96.36%	77.58%	97.42%	80.18%
Guitar	98.10%	38.91%	56.01%	98.63%	96.52%	97.70%	97.15%	98.71%
Total Accuracy	95.84%	46.04%	78.62%	90.69%	94.77%	87.36%	95.71%	90.68%
Avg. Accuracy	95.08%	46.69%	73.40%	90.82%	93.56%	87.52%	93.59%	90.81%

Table B.1.: Baseline evaluation of all 3 comparison methods and our approach on the $Real_{10}$ and $Synth_{10}$ test set.

Explicit Early Adaptation

Method / Class	Ours (Last)		Ours (Model)		Ours (Streams)		Ours (Shared)	
	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$
Airplane	98.67%	83.05%	99.02%	96.34%	98.95%	91.97%	98.67%	98.10%
Car	97.50%	94.66%	97.78%	99.12%	97.78%	97.50%	98.57%	99.73%
Lamp	93.67%	68.63%	94.40%	89.72%	93.43%	74.83%	96.35%	93.02%
Loudspeaker	96.09%	57.72%	96.28%	78.68%	94.72%	59.93%	97.46%	86.09%
Telephone	80.51%	74.63%	81.86%	89.49%	79.61%	84.95%	83.51%	92.81%
Chair	97.98%	73.54%	98.22%	92.37%	98.26%	80.85%	98.02%	96.44%
Cabinet	76.27%	73.94%	76.98%	87.04%	74.69%	82.91%	75.75%	89.65%
Rifle	94.57%	79.87%	96.79%	96.50%	95.31%	89.53%	97.53%	98.77%
Display	97.49%	63.93%	97.42%	80.18%	97.11%	68.83%	96.66%	82.68%
Guitar	96.20%	92.51%	97.15%	98.71%	93.35%	95.20%	97.78%	99.27%
Total Accuracy	95.29%	76.17%	95.71%	90.68%	95.15%	82.55%	95.94%	93.53%
Avg. Accuracy	92.90%	76.25%	93.59%	90.81%	92.32%	82.65%	94.03%	93.66%

Table B.2.: Comparison of different training strategies. Accuracies correspond to evaluation the on the $Real_{10}$ and $Synth_{10}$ test sets.

Additional Fine-Tuning

Method / Class	Ours (N-FT)		Ours (FT-R)		Ours (FT-S)	
	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$	$Real_{10}$	$Synth_{10}$
Airplane	98.67%	98.10%	99.09%	96.58%	98.67%	96.67%
Car	98.57%	99.73%	98.57%	99.05%	97.61%	99.30%
Lamp	96.35%	93.02%	96.59%	89.44%	92.21%	91.04%
Loudspeaker	97.46%	86.09%	96.28%	80.26%	94.72%	81.96%
Telephone	83.51%	92.81%	84.11%	89.36%	80.21%	90.90%
Chair	98.02%	96.44%	97.91%	90.47%	98.10%	93.21%
Cabinet	75.75%	89.65%	80.32%	85.61%	76.45%	87.70%
Rifle	97.53%	98.77%	97.28%	97.25%	94.81%	97.21%
Display	96.66%	82.68%	97.42%	77.71%	96.58%	81.25%
Guitar	97.78%	99.27%	97.78%	98.59%	96.20%	98.86%
Total Accuracy	95.94%	93.53%	96.28%	90.31%	95.11%	91.69%
Avg. Accuracy	94.03%	93.66%	94.54%	90.43%	92.56%	91.81%

Table B.3.: Comparison of additionally fine-tuning the pre-trained model on one of the domains. These fine-tuned models are then used to initialize *Ours (FT-R)* and *Ours (FT-S)*. Accuracies correspond to evaluation the on the $Real_{10}$ and $Synth_{10}$ test sets.

C. Expansion: Additional Results

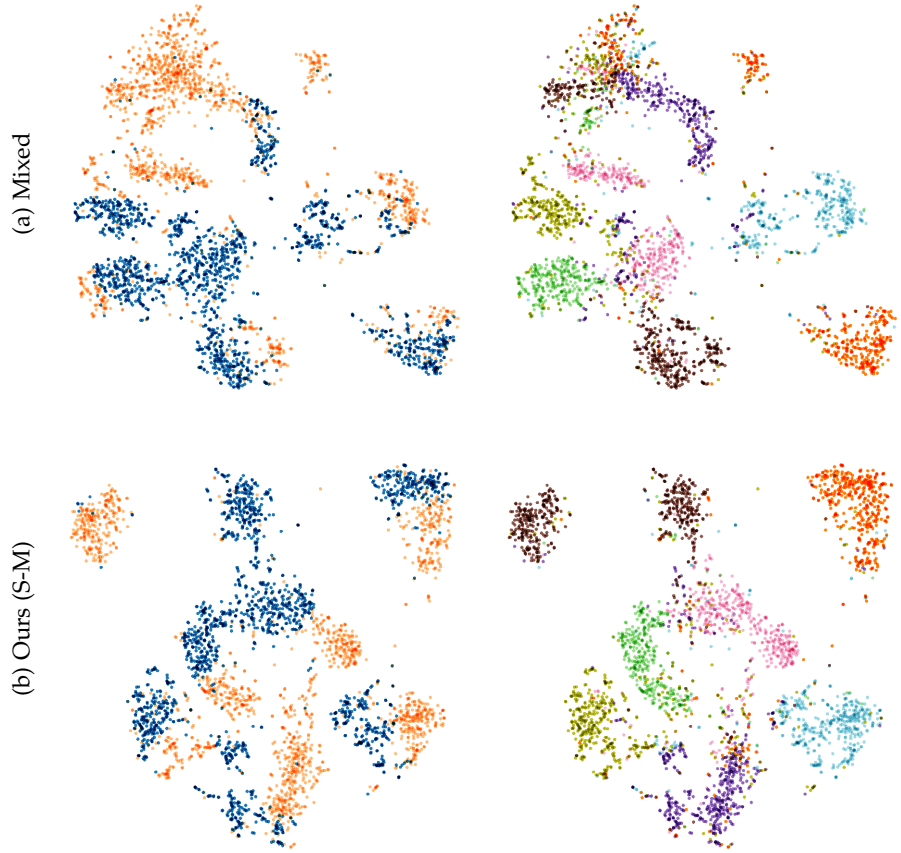


Figure C.1.: Isolated t-SNE visualizations of the additional 7 classes. The visualizations embeds the 17-dimensional *fc8* feature space with perplexity 30. For visibility only 256 samples of each class are displayed. **Left:** Blue and orange dots correspond to synthetic samples (source) and real samples (target), respectively. **Right:** Each color refer to one of the 7 classes.

Evaluation of the $Synth_{17}$ test set

Class \ Method	Baseline (Real)	Baseline (Synth)	Mixed	Ours (Model)	Ours (FT-S)	Ours (FT-R)
Airplane	51.86%	*94.67%*	95.16%	93.79%	**96.71%**	96.01%
Car	39.94%	*97.49%*	97.77%	**98.86%**	98.11%	97.80%
Lamp	30.20%	*81.47%*	82.68%	79.50%	85.11%	**83.88%**
Loudspeaker	29.82%	*70.97%*	73.58%	76.20%	**79.46%**	77.09%
Telephone	35.63%	*91.32%*	92.34%	89.16%	**93.65%**	93.51%
Chair	56.72%	*79.84%*	81.43%	83.01%	**84.61%**	82.60%
Cabinet	5.35%	*77.89%*	78.88%	**81.85%**	79.80%	80.46%
Rifle	53.05%	*94.40%*	95.39%	96.32%	**96.50%**	95.98%
Display	41.26%	*75.05%*	76.55%	74.00%	**78.68%**	77.29%
Guitar	35.41%	*98.67%*	99.11%	98.95%	**99.28%**	99.06%
Watercraft	24.90%	*90.87%*	91.33%	91.15%	**92.36%**	91.83%
Sofa	9.18%	*80.49%*	80.88%	81.11%	**82.37%**	80.63%
Table	10.12%	*84.71%*	85.11%	**88.49%**	85.42%	85.36%
Clock	29.69%	*72.36%*	71.94%	70.39%	**74.38%**	71.24%
Faucet	50.81%	*89.82%*	89.98%	**91.45%**	90.75%	90.11%
Laptop	19.98%	*93.94%*	94.19%	**96.35%**	95.53%	94.89%
Flowerpot	54.36%	*74.59%*	76.04%	76.66%	**77.58%**	74.49%
Total Accuracy	34.01%	*85.20%*	86.01%	86.37%	**87.64%**	86.59%
Avg. Accuracy	34.02%	*85.21%*	86.02%	86.31%	**87.66%**	86.60%
Avg. Acc (base classes)	37.92%	*86.18%*	87.29%	87.17%	**89.19%**	88.37%
Avg. Acc (novel classes)	28.43%	*83.82%*	84.21%	85.09%	**85.48%**	84.08%

Table C.1.: The per-class accuracies of the evaluated methods. We compare our approach in three configurations with *Mixed* and *Baseline (Real)*. *Baseline (Synth)* serves as reference.

List of Figures

List of Tables

Bibliography

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

[2] R. Aljundi and T. Tuytelaars. "Lightweight Unsupervised Domain Adaptation by Convolutional Filter Reconstruction." In: *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*. Ed. by G. Hua and H. Jégou. Vol. 9915. Lecture Notes in Computer Science. 2016, pp. 508–515. DOI: 10.1007/978-3-319-49409-8_43.

[3] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. "Analysis of Representations for Domain Adaptation." In: *Advances in Neural Information Processing Systems 19, Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems, Vancouver, British Columbia, Canada, December 4-7, 2006*. Ed. by B. Schölkopf, J. C. Platt, and T. Hofmann. MIT Press, 2006, pp. 137–144.

[4] K. Bousmalis, A. Irpan, P. Wohlhart, Y. Bai, M. Kelcey, M. Kalakrishnan, L. Downs, J. Ibarz, P. Pastor, K. Konolige, S. Levine, and V. Vanhoucke. "Using Simulation and Domain Adaptation to Improve Efficiency of Deep Robotic Grasping." In: *CoRR* abs/1709.07857 (2017). arXiv: 1709.07857.

[5] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan. "Domain Separation Networks." In: *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*. Ed. by D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett. 2016, pp. 343–351.

[6] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah. "Signature Verification Using a "Siamese" Time Delay Neural Network." In: *Proceedings of the 6th International Conference on Neural Information Processing Systems*. NIPS'93. Denver, Colorado: Morgan Kaufmann Publishers Inc., 1993, pp. 737–744.

[7] A. X. Chang, T. A. Funkhouser, L. J. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. "ShapeNet: An Information-Rich 3D Model Repository." In: *CoRR* abs/1512.03012 (2015). arXiv: 1512.03012.

[8] G. Csurka. "A Comprehensive Survey on Domain Adaptation for Visual Applications." In: *Domain Adaptation in Computer Vision Applications.* Ed. by G. Csurka. Advances in Computer Vision and Pattern Recognition. Springer, 2017, pp. 1–35. DOI: 10.1007/978-3-319-58347-1_1.

[9] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*. IEEE Computer Society, 2009, pp. 248–255. DOI: 10.1109/CVPRW.2009.5206848.

[10] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. "DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition." In: *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*. Vol. 32. JMLR Workshop and Conference Proceedings. JMLR.org, 2014, pp. 647–655.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun. "CARLA: An Open Urban Driving Simulator." In: *1st Annual Conference on Robot Learning, CoRL 2017, Mountain View, California, USA, November 13-15, 2017, Proceedings*. Vol. 78. Proceedings of Machine Learning Research. PMLR, 2017, pp. 1–16.

[12] C. Feichtenhofer, A. Pinz, and A. Zisserman. "Convolutional Two-Stream Network Fusion for Video Action Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*. IEEE Computer Society, 2016, pp. 1933–1941. DOI: 10.1109/CVPR.2016.213.

[13] I. J. Goodfellow, M. Mirza, D. Xiao, A. Courville, and Y. Bengio. "An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks." In: *CoRR* (Dec. 21, 2013). arXiv: 1312.6211v3 [stat.ML].

[14] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. "Generative Adversarial Nets." In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 2672–2680.

[15] A. Gretton, K. M. Borgwardt, M. J. Rasch, B. Schölkopf, and A. J. Smola. "A Kernel Two-Sample Test." In: *Journal of Machine Learning Research* 13 (2012), pp. 723–773.

[16] A. Handa, V. Patraucean, V. Badrinarayanan, S. Stent, and R. Cipolla. "SceneNet: Understanding Real World Indoor Scenes With Synthetic Data." In: *CoRR* abs/1511.07041 (2015). arXiv: 1511.07041.

[17] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385.

[18] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification." In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 1026–1034. DOI: 10.1109/ICCV.2015.123.

[19] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Improving neural networks by preventing co-adaptation of feature detectors." In: *CoRR* abs/1207.0580 (2012). arXiv: 1207.0580.

[20] M. Johnson-Roberson, C. Barto, R. Mehta, S. N. Sridhar, K. Rosaen, and R. Vasudevan. "Driving in the Matrix: Can virtual worlds replace human-generated annotations for real world tasks?" In: *2017 IEEE International Conference on Robotics and Automation, ICRA 2017, Singapore, Singapore, May 29 - June 3, 2017*. IEEE, 2017, pp. 746–753. DOI: 10.1109/ICRA.2017.7989092.

[21] H. Jung, J. Ju, M. Jung, and J. Kim. "Less-forgetting Learning in Deep Neural Networks." In: *CoRR* abs/1607.00122 (2016). arXiv: 1607.00122.

[22] E. Kolve, R. Mottaghi, D. Gordon, Y. Zhu, A. Gupta, and A. Farhadi. "AI2-THOR: An Interactive 3D Environment for Visual AI." In: *CoRR* abs/1712.05474 (2017). arXiv: 1712.05474.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet classification with deep convolutional neural networks." In: *Commun. ACM* 60.6 (2017), pp. 84–90. DOI: 10.1145/3065386.

[24] Y. Li, H. Su, C. R. Qi, N. Fish, D. Cohen-Or, and L. J. Guibas. "Joint embeddings of shapes and images via CNN image purification." In: *ACM Trans. Graph.* 34.6 (2015), 234:1–234:12. DOI: 10.1145/2816795.2818071.

[25] M. Long, Y. Cao, J. Wang, and M. I. Jordan. "Learning Transferable Features with Deep Adaptation Networks." In: *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*. Ed. by F. R. Bach and D. M. Blei. Vol. 37. JMLR Workshop and Conference Proceedings. JMLR.org, 2015, pp. 97–105.

[26] Z. Luo, Y. Zou, J. Hoffman, and F. Li. "Label Efficient Learning of Transferable Representations acrosss Domains and Tasks." In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*. Ed. by I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett. 2017, pp. 164–176.

[27] L. van der Maaten and G. Hinton. "Visualizing Data using t-SNE." In: *Journal of Machine Learning Research* 9 (2008), pp. 2579–2605.

[28] J. McCormac, A. Handa, S. Leutenegger, and A. J. Davison. "SceneNet RGB-D: 5M Photorealistic Images of Synthetic Indoor Trajectories with Ground Truth." In: *CoRR* abs/1612.05079 (2016). arXiv: 1612.05079.

[29] G. A. Miller. "WordNet: A Lexical Database for English." In: *Commun. ACM* 38.11 (Nov. 1995), pp. 39–41. ISSN: 0001-0782. DOI: 10.1145/219717.219748.

[30] M. Mueller, V. Casser, J. Lahoud, N. Smith, and B. Ghanem. "UE4Sim: A Photo-Realistic Simulator for Computer Vision Applications." In: *CoRR* abs/1708.05869 (2017). arXiv: 1708.05869.

[31] S. J. Pan and Q. Yang. "A Survey on Transfer Learning." In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359. ISSN: 1041-4347. DOI: 10.1109/TKDE.2009.191.

[32] S. J. Pan, I. W. Tsang, J. T. Kwok, and Q. Yang. "Domain Adaptation via Transfer Component Analysis." In: *IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009*. Ed. by C. Boutilier. 2009, pp. 1187–1192.

[33] W. Qiu and A. L. Yuille. "UnrealCV: Connecting Computer Vision to Unreal Engine." In: *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*. 2016, pp. 909–916. DOI: 10.1007/978-3-319-49409-8_75.

[34] S. R. Richter, Z. Hayder, and V. Koltun. "Playing for Benchmarks." In: *CoRR* abs/1709.07322 (2017). arXiv: 1709.07322.

[35] S. R. Richter, V. Vineet, S. Roth, and V. Koltun. "Playing for Data: Ground Truth from Computer Games." In: *CoRR* abs/1608.02192 (2016). arXiv: 1608.02192.

[36] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li. "ImageNet Large Scale Visual Recognition Challenge." In: *International Journal of Computer Vision* 115.3 (2015), pp. 211–252. DOI: 10.1007/s11263-015-0816-y.

[37] K. Saenko, B. Kulis, M. Fritz, and T. Darrell. "Adapting Visual Category Models to New Domains." In: *Computer Vision - ECCV 2010, 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV*. Ed. by K. Daniilidis, P. Maragos, and N. Paragios. Vol. 6314. Lecture Notes in Computer Science. Springer, 2010, pp. 213–226. DOI: 10.1007/978-3-642-15561-1_16.

[38] M. Savva, A. X. Chang, A. Dosovitskiy, T. A. Funkhouser, and V. Koltun. "MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments." In: *CoRR* abs/1712.03931 (2017). arXiv: 1712.03931.

[39] A. Shafaei, J. J. Little, and M. Schmidt. "Play and Learn: Using Video Games to Train Computer Vision Models." In: *CoRR* abs/1608.01745 (2016). arXiv: 1608.01745.

[40] S. Shah, D. Dey, C. Lovett, and A. Kapoor. "AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles." In: *Field and Service Robotics, Results of the 11th International Conference, FSR 2017, Zurich, Switzerland, 12-15 September 2017*. Ed. by M. Hutter and R. Siegwart. Vol. 5. Springer Proceedings in Advanced Robotics. Springer, 2017, pp. 621–635. DOI: 10.1007/978-3-319-67361-5_40.

[41] K. Simonyan and A. Zisserman. "Two-Stream Convolutional Networks for Action Recognition in Videos." In: *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13*

2014, Montreal, Quebec, Canada. Ed. by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger. 2014, pp. 568–576.

[42] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition." In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556.

[43] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. A. Funkhouser. "Semantic Scene Completion from a Single Depth Image." In: *CoRR* abs/1611.08974 (2016). arXiv: 1611.08974.

[44] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[45] H. Su, C. R. Qi, Y. Li, and L. J. Guibas. "Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views." In: *CoRR* abs/1505.05641 (2015). arXiv: 1505.05641.

[46] B. Sun, J. Feng, and K. Saenko. "Return of Frustratingly Easy Domain Adaptation." In: *CoRR* abs/1511.05547 (2015). arXiv: 1511.05547.

[47] B. Sun and K. Saenko. "Deep CORAL: Correlation Alignment for Deep Domain Adaptation." In: *Computer Vision - ECCV 2016 Workshops - Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III*. Ed. by G. Hua and H. Jégou. Vol. 9915. Lecture Notes in Computer Science. 2016, pp. 443–450. DOI: 10.1007/978-3-319-49409-8_35.

[48] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions." In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015, Boston, MA, USA, June 7-12, 2015*. IEEE Computer Society, 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[49] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke. "Sim-to-Real: Learning Agile Locomotion For Quadruped Robots." In: *CoRR* abs/1804.10332 (2018). arXiv: 1804.10332.

[50] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. "Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World." In: *CoRR* abs/1703.06907 (2017). arXiv: 1703.06907.

[51] E. Tzeng, J. Hoffman, T. Darrell, and K. Saenko. "Simultaneous Deep Transfer Across Domains and Tasks." In: *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*. IEEE Computer Society, 2015, pp. 4068–4076. DOI: 10.1109/ICCV.2015.463.

[52] E. Tzeng, J. Hoffman, K. Saenko, and T. Darrell. "Adversarial Discriminative Domain Adaptation." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*. IEEE Computer Society, 2017, pp. 2962–2971. DOI: 10.1109/CVPR.2017.316.

[53] Udacity. *Udacity's Self-Driving Car Simulator*. https://github.com/udacity/self-driving-car-sim. 2017.

[54] V. S. R. Veeravasarapu, C. A. Rothkopf, and V. Ramesh. "Adversarially Tuned Scene Generation." In: *CoRR* abs/1701.00405 (2017). arXiv: 1701.00405.

[55] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792.

[56] Y. Zhang, S. Song, E. Yumer, M. Savva, J. Lee, H. Jin, and T. A. Funkhouser. "Physically-Based Rendering for Indoor Scene Understanding Using Convolutional Neural Networks." In: *CoRR* abs/1612.07429 (2016). arXiv: 1612.07429.